Materiały do przedmiotu:

Podstawy Systemów Operacyjnych

Laboratorium nr 1:

Wirtualizacja środowisk oraz tworzenie maszyn wirtualnych

Autor:

Albert Rachwał

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Pierwsze laboratorium stanowi wprowadzenie do praktyki pracy w środowisku wirtualnym, które jest fundamentem całego kursu. Wirtualizacja pozwala uruchamiać wiele systemów operacyjnych na jednym komputerze, co czyni ją niezwykle użytecznym narzędziem do nauki, testowania i administracji. Dzięki niej możliwe jest izolowanie eksperymentów, szybkie przywracanie poprzednich stanów oraz bezpieczne wykonywanie zadań, które na fizycznej maszynie byłyby ryzykowne. Studenci poznają tu ideę działania maszyn wirtualnych oraz powiązane z nimi pojęcia, dowiadują się, czym jest hypervisor i jaką rolę pełni w zarządzaniu zasobami, a także uczą się, czym różni się wirtualizacja od konteneryzacji. Ćwiczenia polegają na samodzielnym utworzeniu i skonfigurowaniu maszyny wirtualnej z wybranym systemem Linux, przydzieleniu jej odpowiednich zasobów oraz przygotowaniu środowiska pracy, które stanie się bazą dla kolejnych zajęć.

Cel ćwiczenia/laboratorium

- Zrozumienie roli systemu operacyjnego oraz wymienienie elementów które go tworzą.
- Zrozumienie pojęcia wirtualizacji oraz roli hiperwizora w zarządzaniu zasobami sprzętowymi.
- Rozróżnienie hiperwizorów typu 1 i typu 2 oraz wskazanie ich typowych zastosowań.
- Umiejętność utworzenia i skonfigurowania podstawowej maszyny wirtualnej w środowisku VirtualBox.
- Konfiguracja parametrów Maszyny Wirtualnej (przydział CPU, RAM, przestrzeni dyskowej, interfejsów sieciowych).
- Wykorzystanie mechanizmu snapshotów do tworzenia punktów przywracania systemu.
- Praktyczne skonfigurowanie sieci wirtualnej w trybie NAT oraz Bridge i przetestowanie komunikacji.
- Instalacja i podstawowa administracja systemem Linux uruchomionym w maszynie wirtualnej.
- Zrozumienie ograniczeń wirtualizacji oraz porównanie jej do konteneryzacji.
- Utworzenie prostego kontenera Linux (np. w Dockerze) i uruchomienie w nim aplikacji testowej.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

- Porównanie czasu uruchamiania maszyny wirtualnej i kontenera.
- Zrozumienie roli wirtualizacji i konteneryzacji w nowoczesnych środowiskach chmurowych i DevOps.
- Nabycie praktycznych umiejętności pozwalających na samodzielne budowanie, uruchamianie i zarządzanie maszynami wirtualnymi i kontenerami w środowisku laboratoryjnym.
- Umiejętność pisania prostych plików dockerfile oraz docker-compose.yml do konfigurowania aplikacji i usług w kontenerach.
- Znajomość podstawowych narzędzi systemowych.
- Umiejętność wykorzystywania masek (wildcards) do masowego zarządzania plikami i katalogami.
- Wykształcenie nawyku korzystania z manuala systemowego (man, --help).
- Umiejętność nawigowania się po manualu systemowym

Instrukcja laboratoryjna/ćwiczeniowa

Warunki zaliczenia przedmiotu

Sugerowaną formą zaliczenia niniejszego przedmiotu są dwa kolokwia. Zakres pierwszego kolokwium może obejmować:

1. Operacje na plikach i katalogach
2. Podstawowe narzędzia systemowe
3. Maski i masowe przetwarzanie plików i katalogów
4. Prawa dostępu
5. Przetwarzanie potokowe
6. Strumienie i przekierowania
7. Filtracja danych
8. Edycja tekstu narzędziem sed
9. Analiza danych narzędziem awk

Zakres drugiego kolokwium może obejmować:

1. Procesy i sygnały.
2. Monitorowanie procesów.
3. Komunikacja międzyprocesowa.
4. Wiedzę dotyczącą podstawowych usług systemowych.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

5. Kompilacja programów napisanych w językach C++ lub Rust.
6. Tworzenie prostych programów klient-serwer.
7. Zarządzanie kontami użytkowników
8. Analiza logów systemowych

Czym jest system operacyjny i z jakich elementów się składa

System operacyjny to najważniejsze oprogramowanie komputera, które odpowiada za zarządzanie sprzętem oraz udostępnianie zasobów aplikacjom i użytkownikowi. Można go traktować jako pośrednika pomiędzy człowiekiem a maszyną – dzięki niemu programy nie muszą samodzielnie zajmować się obsługą procesora, pamięci, dysku czy urządzeń peryferyjnych, ponieważ wszystkie te funkcje są realizowane przez warstwę systemową.

Na system operacyjny składa się wiele współpracujących ze sobą elementów. Jednym z pierwszych, z którym mamy do czynienia przy starcie komputera, jest bootloader. To niewielki program, którego zadaniem jest uruchomienie systemu i przekazanie sterowania dalej. Najpopularniejszym bootloaderem w środowisku Linux jest GRUB, czyli GNU GRUB – GRand Unified Bootloader. To właśnie on pozwala na wybór systemu operacyjnego przy starcie, obsługuje wiele systemów plików, potrafi uruchamiać różne jądra Linuksa oraz przekazywać do nich parametry startowe. W praktyce, jeśli użytkownik widzi na ekranie menu wyboru systemu operacyjnego, to pracuje właśnie GRUB.

Kiedy bootloader załaduje jądro do pamięci, sterowanie przejmuje kernel – serce systemu operacyjnego. To on odpowiada za zarządzanie procesorem, pamięcią, urządzeniami wejścia i wyjścia, systemem plików oraz procesami. W Linuksie jądro ma charakter modułowy, co oznacza, że sterowniki i dodatkowe funkcje mogą być doładowywane w czasie działania systemu. Z jądrem ściśle współpracują biblioteki systemowe, które udostępniają programom zestaw funkcji i interfejsów API, pozwalając aplikacjom komunikować się ze sprzętem w sposób ujednolicony.

Na kolejnym poziomie znajduje się powłoka systemowa, która zapewnia użytkownikowi interfejs do komunikacji z systemem. W przypadku powłok tekstowych, takich jak bash, zsh czy fish, jest to wiersz poleceń, w którym można wpisywać instrukcje wykonywane następnie przez system. Alternatywą są

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

środowiska graficzne, jak GNOME czy KDE, które pełnią tę samą rolę, ale w bardziej przyjaznej dla użytkownika formie.

System operacyjny zarządza również systemem plików, czyli logiczną strukturą przechowywania danych na nośnikach. W Linuksie często spotykane są systemy ext4, XFS czy Btrfs, a w Windowsie – NTFS. Innym kluczowym elementem są menedżery procesów i pamięci, które czuwają nad uruchamianiem i planowaniem zadań oraz pilnują, by procesy nie wchodziły sobie w drogę w dostępie do zasobów. Ważną rolę pełnią także sterowniki urządzeń, które umożliwiają komunikację pomiędzy jądrem a sprzętem takim jak karty sieciowe, procesory graficzne czy dyski.

Istotną część systemu stanowią procesy działające w tle – tak zwane demony. To one odpowiadają za realizację podstawowych funkcji, takich jak zdalny dostęp do systemu (sshd), wykonywanie cyklicznych zadań (cron) czy inicjalizację usług (systemd). Całość dopełniają aplikacje systemowe i narzędzia użytkowe – programy, które pozwalają na konfigurację, administrację i codzienne korzystanie z systemu.

Cały cykl uruchamiania systemu operacyjnego przebiega etapami. Najpierw BIOS lub UEFI inicjalizuje sprzęt. Następnie startuje bootloader – często właśnie GRUB – który ładuje jądro systemu do pamięci. Jądro rozpoczyna inicjalizację procesora, pamięci, sterowników oraz montuje system plików. Potem rolę przejmuje proces inicjalizacyjny, np. systemd, uruchamiający kolejne usługi i demony. Na końcu użytkownik otrzymuje powłokę tekstową lub graficzne środowisko pracy, dzięki któremu może rozpocząć interakcję z komputerem.

Wirtualizacja

Wirtualizacja jest technologią, która umożliwia uruchamianie wielu niezależnych środowisk obliczeniowych na jednym fizycznym komputerze lub serwerze. Dzięki oprogramowaniu – tzw. hiperwizorowi – zasoby sprzętowe takie jak procesor, pamięć RAM, dysk czy sieć mogą zostać podzielone pomiędzy wiele maszyn wirtualnych (VM – Virtual Machines). Każda z nich działa jak osobny komputer, posiada własny system operacyjny i aplikacje, mimo że korzysta ze wspólnych fizycznych zasobów.

Historia wirtualizacji sięga lat 60., ale prawdziwy przełom nastąpił w latach 90., gdy VMware opracowało rozwiązania pozwalające uruchamiać wiele systemów

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

operacyjnych równolegle na komputerach klasy PC. Obecnie wirtualizacja jest filarem nowoczesnych centrów danych i chmury obliczeniowej.

Zalety wirtualizacji obejmują m.in.: efektywne wykorzystanie zasobów sprzętowych, szybsze wdrażanie nowych środowisk, minimalizację przestojów dzięki snapshotom i replikacji, ułatwione zarządzanie oraz redukcję kosztów. Wirtualizacja zwiększa także bezpieczeństwo – w razie awarii lub infekcji maszyna wirtualna może zostać szybko przywrócona do poprzedniego stanu.

Warto również wspomnieć o konteneryzacji, która stanowi alternatywę dla wirtualizacji pełnych systemów. Kontenery nie wymagają uruchamiania całego systemu operacyjnego, lecz współdzielą jądro hosta. Dzięki temu są lżejsze, szybsze w uruchamianiu i szczególnie popularne w środowiskach DevOps oraz mikroserwisach.

Wirtualizacja i konteneryzacja to technologie szeroko wykorzystywane w praktyce inżynierskiej – od środowisk developerskich i testowych po produkcyjne klastry chmurowe. Umiejętność tworzenia, konfiguracji i zarządzania maszynami wirtualnymi jest zatem kluczową kompetencją współczesnego inżyniera IT.

Hiperwizor

Hiperwizor (ang. hypervisor, nazywany także monitorem maszyn wirtualnych – Virtual Machine Monitor, VMM) to specjalne oprogramowanie, które tworzy i zarządza maszynami wirtualnymi. Działa jako warstwa pośrednia między fizycznym sprzętem (hostem) a systemami operacyjnymi działającymi wewnątrz maszyn wirtualnych (gośćmi). Jego zadaniem jest podział zasobów sprzętowych – procesora, pamięci RAM, przestrzeni dyskowej i interfejsów sieciowych – tak, aby każda maszyna wirtualna mogła działać niezależnie i izolowana od innych, mimo że wszystkie współdzielą ten sam sprzęt fizyczny.

Wyróżniamy dwa główne typy hiperwizorów:

1. Typ 1 – „bare-metal” (nagie żelazo)
    - Instalowane bezpośrednio na sprzęcie fizycznym – zastępują klasyczny system operacyjny.
    - Mają bezpośredni dostęp do procesora, pamięci i innych zasobów sprzętowych.

- Cechują się wysoką wydajnością, niskimi opóźnieniami i większą stabilnością.
- Stosowane w centrach danych, chmurach i środowiskach serwerowych.
- Przykłady: VMware ESXi, Microsoft Hyper-V (w wersji serwerowej), Xen, KVM (Kernel-based Virtual Machine w Linuxie).

2. Typ 2 – „hosted” (na systemie gospodarza)
   - Uruchamiane jako aplikacja w istniejącym systemie operacyjnym.
   - Nie komunikują się bezpośrednio ze sprzętem, tylko za pośrednictwem systemu hosta.
   - Są łatwiejsze w instalacji i obsłudze, ale mniej wydajne niż typ 1.
   - Popularne w zastosowaniach domowych, edukacyjnych i developerskich.
   - Przykłady: Oracle VirtualBox, VMware Workstation, Parallels Desktop (macOS).

Hiperwizor odpowiada za szereg funkcji kluczowych dla wirtualizacji:

- Tworzenie i uruchamianie maszyn wirtualnych – inicjalizacja wirtualnych CPU, pamięci, dysków i kart sieciowych.
- Zarządzanie zasobami – dynamiczne przydzielanie i zwalnianie pamięci RAM, procesora czy przestrzeni dyskowej między VM.
- Izolacja środowisk – zapewnia, że awaria lub infekcja jednej maszyny nie wpływa na pozostałe ani na hosta.
- Snapshoty i klonowanie – umożliwia zatrzymanie stanu maszyny w danym momencie i późniejsze przywrócenie.
- Migracja maszyn – w środowiskach serwerowych pozwala przenosić VM między hostami (np. live migration w KVM/VMware).
- Obsługa wirtualnych urządzeń – np. wirtualne karty sieciowe, kontrolery dysków, GPU w trybie passthrough.

Hiperwizor ma bezpośredni wpływ na wydajność środowiska wirtualnego. W przypadku hiperwizorów typu 1, które działają bezpośrednio na sprzęcie fizycznym, osiągana wydajność jest bardzo zbliżona do tej znanej z pracy na serwerze fizycznym. Dlatego rozwiązania te stosowane są przede wszystkim w profesjonalnych serwerowniach i w infrastrukturze chmur publicznych, takich jak AWS, Microsoft Azure czy Google Cloud Platform. Z kolei hiperwizory typu 2,

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

uruchamiane jako aplikacje w systemie operacyjnym gospodarza, obarczone są dodatkowym narzutem związanym z podwójną warstwą komunikacji pomiędzy maszyną wirtualną, systemem hosta a sprzętem. Powoduje to spadek wydajności, jednak ten typ hiperwizora znakomicie sprawdza się w testach, nauce i środowiskach edukacyjnych.

Wirtualizacja oprócz korzyści wydajnościowych wnosi również istotne aspekty związane z bezpieczeństwem. Przede wszystkim zapewnia izolację – każda maszyna wirtualna działa niezależnie, dzięki czemu infekcja w jednej z nich nie musi oznaczać zagrożenia dla pozostałych. Ogromnym ułatwieniem są również snapshoty, pozwalające w prosty sposób cofnąć system do wcześniejszego, czystego stanu. Wirtualizacja daje również możliwość segmentacji sieciowej, dzięki której można wydzielić poszczególne maszyny wirtualne do osobnych sieci, zwiększając poziom bezpieczeństwa. Trzeba jednak pamiętać, że istnieją także zagrożenia – przykładem są ataki typu VM escape, które polegają na wydostaniu się z maszyny wirtualnej do systemu gospodarza. Choć są one trudne do przeprowadzenia, stanowią realne ryzyko, dlatego tak ważne są regularne aktualizacje hiperwizora.

Hiperwizory są obecnie fundamentem wielu dziedzin współczesnej informatyki. Stanowią podstawę działania chmur obliczeniowych, takich jak AWS, Azure czy GCP, gdzie pozwalają na elastyczne przydzielanie zasobów. Wykorzystywane są również w obszarze DevOps oraz testowania oprogramowania, ponieważ umożliwiają szybkie odtwarzanie gotowych środowisk. Mają ogromne znaczenie w zapewnieniu ciągłości działania systemów i planach odtwarzania po awarii (disaster recovery). Spotyka się je także w środowiskach edukacyjnych, gdzie służą do nauki i testów.

Konteneryzacja

Konteneryzacja (ang. containerization) to metoda uruchamiania aplikacji w lekkich, izolowanych środowiskach zwanych kontenerami, które współdzielą jądro systemu operacyjnego hosta.

W przeciwieństwie do maszyn wirtualnych, kontener nie wymaga osobnego systemu operacyjnego – zawiera tylko aplikację oraz jej zależności (biblioteki, konfiguracje, zmienne środowiskowe). Dzięki temu kontenery są lżejsze, szybsze w uruchamianiu i

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

bardziej przenośne niż maszyny wirtualne. Technologie kontenerowe

Na rynku istnieje wiele rozwiązań wspierających konteneryzację. Najpopularniejszym i najczęściej używanym narzędziem jest Docker, który stał się niemal synonimem samego pojęcia kontenerów. Obok niego rozwijany jest Podman, będący alternatywą działającą bez centralnego demona, co poprawia bezpieczeństwo i prostotę wdrożeń. Warto wspomnieć także o LXC/LXD, które oferują lekkie kontenery systemowe i w pewnym stopniu przypominają maszyny wirtualne, oraz o rozwiązaniach takich jak CRI-O czy containerd, będących środowiskami uruchomieniowymi szczególnie często wykorzystywanymi w ekosystemie Kubernetes.

Orkiestracja kontenerów

Wraz ze wzrostem popularności konteneryzacji i uruchamianiem coraz większej liczby kontenerów pojawiła się konieczność ich automatycznego zarządzania. Ręczne kontrolowanie kilkudziesięciu czy setek instancji byłoby niepraktyczne, dlatego opracowano systemy orkiestracji. Najważniejszym i najpowszechniej stosowanym jest Kubernetes, który stał się standardem de facto w tej dziedzinie. Umożliwia on automatyczne skalowanie aplikacji, równoważenie obciążenia, a także bezpieczne wdrażanie nowych wersji oprogramowania z możliwością szybkiego wycofania zmian w razie problemów. Prostszą, lecz mniej rozbudowaną alternatywą jest Docker Swarm, który oferuje podstawowe mechanizmy zarządzania klastrem i sprawdza się w mniejszych projektach.

Kontener składa się z kilku elementów:

- Aplikacja – właściwy program, który chcemy uruchomić.
- Zależności aplikacji – biblioteki, pliki konfiguracyjne, język uruchomieniowy (np. Python, Java).
- Obraz kontenera (image) – statyczny pakiet zawierający aplikację i wszystkie potrzebne komponenty.
- Silnik kontenerów (container runtime) – oprogramowanie uruchamiające kontenery (np. Docker Engine, containerd, CRI-O).
- System operacyjny hosta – dostarcza wspólne jądro, które jest współdzielone przez wszystkie kontenery.

Cechy charakterystyczne kontenerów:

- Lekkość – kontenery nie powielają całego systemu operacyjnego (w przeciwieństwie do VM), dlatego zajmują mniej miejsca i zużywają mniej zasobów.
- Szybkość – startują w sekundach (VM zwykle w dziesiątki sekund lub minuty).
- Izolacja – każdy kontener działa w odizolowanym środowisku (namespaces, cgroups w Linuxie).
- Przenośność – obraz kontenera można uruchomić na każdym systemie z odpowiednim silnikiem (np. Docker), niezależnie od platformy sprzętowej.
- Skalowalność – łatwo uruchamiać dziesiątki czy setki kopii tego samego kontenera.

Zalety konteneryzacji:

- Szybkie uruchamianie aplikacji.
- Oszczędność zasobów w porównaniu do VM.
- Łatwe testowanie i rozwój – jeden obraz = identyczne środowisko u każdego programisty.
- Elastyczność – łatwo przenosić aplikacje między laptopem, serwerem a chmurą.
- Skalowanie – uruchamianie wielu kopii aplikacji w klastrze.
- DevOps – naturalna integracja z pipeline'ami CI/CD.

Wady i ograniczenia:

- Mniejsza izolacja – wszystkie kontenery dzielą jądro hosta, co stwarza ryzyko tzw. container escape.
- Złożoność przy dużej skali – zarządzanie setkami kontenerów wymaga narzędzi orkiestracyjnych.
- Bezpieczeństwo – wymagają dodatkowych mechanizmów (AppArmor, SELinux, seccomp).
- Persistent storage – trudniejsze niż w VM, wymaga specjalnych wolumenów lub zewnętrznych baz danych.

Instalacja i przykłady użycia środowiska docker

Aby zaisntalować docker na systemie opartym na debianie należy wykonać następujące polecenia:

sudo apt update

sudo apt install docker.io -y

sudo systemctl enable docker

sudo systemctl start docker

Aby sprawdzić czy docker działa można wpisać:

docker --version

docker run hello-world

Polecenie hello-world uruchamia prosty kontener testowy. Pobranie obrazu i uruchomienie interaktywnej powłoki systemu ubuntu w konenerze możemy uzyskać za pomocą polecenia:

docker run -it ubuntu bash

Gdzie:

- -it = interaktywny terminal
- ubuntu = nazwa obrazu
- bash = polecenie uruchomione w kontenerze

Zostało to przedstawione na rysunku 1, aby wyjść z kontenera należy wpisać: exit.

Docker udostępnia kilka podstawowych poleceń do zarządzania obrazami i kontenerami. Aby sprawdzić, jakie obrazy mamy już pobrane lokalnie, używamy komendy docker images. Do sprawdzania, które kontenery są aktualnie uruchomione, służy docker ps. Jeżeli chcemy zobaczyć pełną listę wszystkich kontenerów, w tym także zatrzymanych, należy użyć docker ps -a.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

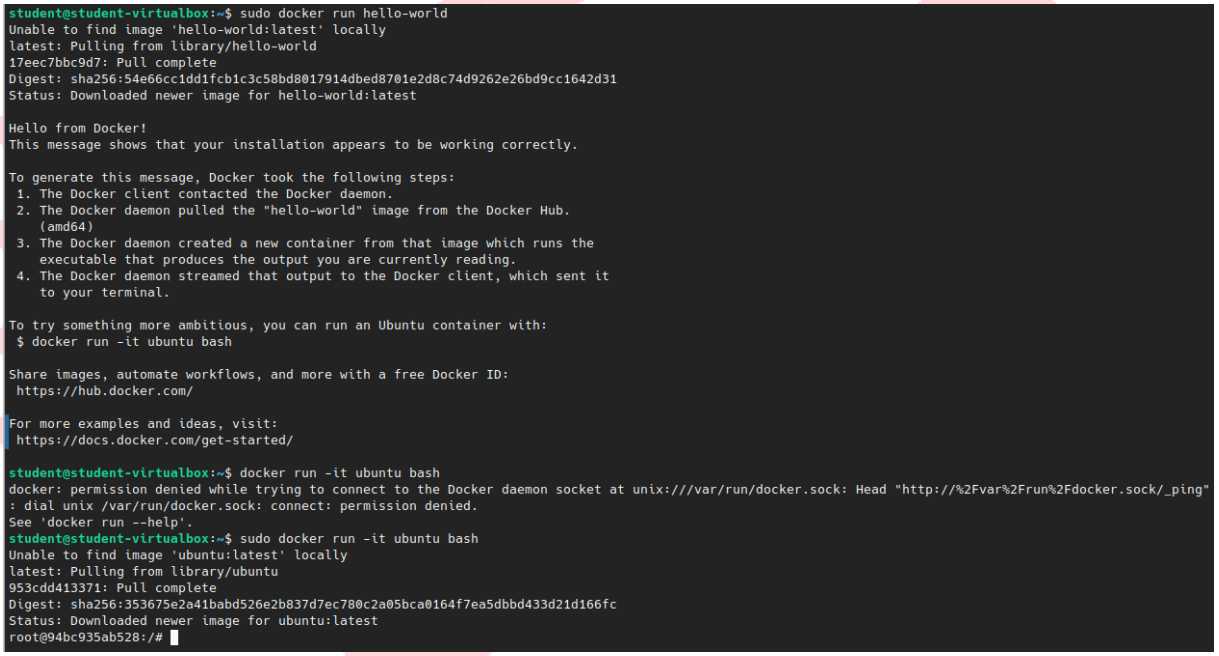
Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Jeżeli interesuje nas zalogowanie się do działającego kontenera i praca w jego wnętrzu, możemy użyć polecenia docker exec. Na przykład:

docker exec -it <id> bash

otwiera powłokę bash wewnątrz kontenera o wskazanym identyfikatorze lub nazwie. Dzięki flagom -it mamy interaktywny terminal, w którym możemy wykonywać komendy tak, jakbyśmy pracowali bezpośrednio na tym systemie. Jeśli w kontenerze nie ma basha, można spróbować użyć sh.

Zatrzymanie działającego kontenera wykonujemy za pomocą polecenia docker stop <id>, gdzie <id> to identyfikator lub nazwa kontenera. Jeśli kontener nie jest już potrzebny, można go całkowicie usunąć poleceniem docker rm <id>.



Rys. 1. Uruchomienie kontenera z systemem Ubuntu.

Dockerfile – budowa obrazu kontenera

Plik Dockerfile jest skryptem opisującym sposób zbudowania obrazu kontenera. Zawiera on zestaw poleceń, które Docker wykonuje kolejno w celu utworzenia środowiska aplikacji – instalacji pakietów, kopiowania plików, konfiguracji zależności i

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

uruchamiania programu. Dzięki temu możemy w pełni zautomatyzować proces tworzenia obrazu, zamiast ręcznie wykonywać kolejne kroki w terminalu. Dockerfile jest podstawą idei Infrastructure as Code – środowisko aplikacji jest zdefiniowane w postaci kodu, co zapewnia powtarzalność i łatwość wdrożeń.

Poniżej znajduje się przykładowy plik Dockerfile budujący prosty serwer WWW w Pythonie:

```
# 1. Obraz bazowy – na nim opiera się nasz kontener
FROM python:3.11-slim
# 2. Ustawienie katalogu roboczego wewnątrz kontenera
WORKDIR /app
# 3. Skopiowanie plików aplikacji do obrazu
COPY . /app
# 4. Instalacja zależności (jeśli istnieje plik requirements.txt)
RUN pip install --no-cache-dir -r requirements.txt
# 5. Określenie polecenia, które ma zostać wykonane po uruchomieniu kontenera
CMD ["python", "app.py"]
```

Omówienie budowy pliku dockerfile:

1. FROM – definiuje obraz bazowy, np. ubuntu, python, node, nginx.
2. WORKDIR – ustawia katalog roboczy, w którym będą wykonywane dalsze instrukcje.
3. COPY – kopiuje pliki z hosta do obrazu.
4. RUN – uruchamia polecenia w trakcie budowy obrazu (np. instalacja pakietów).
5. CMD lub ENTRYPOINT – definiuje, co ma się wykonać przy uruchamianiu kontenera.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Aby zbudować obraz z pliku Dockerfile, należy wykonać:

```
docker build -t myapp .
```

gdzie -t myapp nadaje nazwę obrazowi, a . oznacza, że Dockerfile znajduje się w bieżącym katalogu.

Uruchomienie kontenera z tego obrazu odbywa się przez:

```
docker run -p 8080:8080 myapp
```

Parametr -p mapuje port hosta (lewa strona) na port kontenera (prawa strona), co pozwala uzyskać dostęp do aplikacji np. w przeglądarce.

Przykład 2 – Dockerfile dla prostego serwera Nginx. Ten obraz uruchamia serwer Nginx, który serwuje pliki HTML znajdujące się w katalogu html hosta.

```
FROM nginx:alpine
COPY ./html /usr/share/nginx/html
EXPOSE 80
```

Docker Compose – uruchamianie wielu kontenerów

W bardziej złożonych projektach często istnieje potrzeba uruchomienia kilku kontenerów współpracujących ze sobą, np. aplikacja backendowa + baza danych + front-end. Zarządzanie nimi pojedynczo byłoby nieefektywne, dlatego Docker udostępnia narzędzie Docker Compose, które umożliwia definiowanie i uruchamianie wielu usług jednocześnie przy pomocy jednego pliku YAML (docker-compose.yml).

Poniższy przykład definiuje prostą aplikację składającą się z serwera aplikacji (Python Flask) i bazy danych PostgreSQL:

```
version: '3.9'
services:
  web:
      build: .
      container_name: flask_app
      ports:
      - "5000:5000"
```

```
    environment:
    - DATABASE_URL=postgresql://user:password@db:5432/mydb
    depends_on:
    - db
db:
    image: postgres:15
    container_name: postgres_db
    restart: always
    environment:
    POSTGRES_USER: user
    POSTGRES_PASSWORD: password
    POSTGRES_DB: mydb
    volumes:
    - pgdata:/var/lib/postgresql/data
volumes:
 pgdata:
```

Omówienie budowy przykładowego docker-compose:

1. services – lista kontenerów (usług) wchodzących w skład aplikacji, np. web, db.
2. build – wskazuje, że obraz ma być zbudowany z lokalnego Dockerfile.
3. image – można wskazać gotowy obraz z Docker Hub.
4. ports – mapowanie portów hosta i kontenera.
5. environment – definiowanie zmiennych środowiskowych.
6. volumes – trwałe przechowywanie danych (np. baza danych).
7. depends_on – określa zależności między kontenerami (np. web zależy od db).

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

W celu zbudowanie i uruchomienia środowiska zdefiniowanego w pliku docker-compose możemy użyć polecenia :

docker-compose up --build

Zatrzymanie usług:

docker-compose down

Sprawdzenie uruchomionych kontenerów:

docker-compose ps

W praktyce plik Dockerfile pełni rolę przepisu określającego sposób budowy pojedynczego obrazu aplikacji, na przykład serwera API, aplikacji webowej lub mikroserwisu. To w nim definiuje się wszystkie kroki potrzebne do przygotowania środowiska uruchomieniowego, takie jak instalacja bibliotek, kopiowanie plików źródłowych czy konfiguracja poleceń startowych. Dzięki temu możliwe jest odtworzenie identycznego obrazu aplikacji na dowolnej maszynie, co eliminuje problemy wynikające z różnic w konfiguracjach systemowych. Z kolei plik docker-compose.yml służy do definiowania i uruchamiania zestawów współpracujących kontenerów tzw. stosów aplikacyjnych (ang. stacks). Pozwala on opisać w jednym miejscu wszystkie usługi niezbędne do działania systemu, takie jak serwer aplikacyjny, baza danych, broker komunikatów czy narzędzie monitorujące. Dzięki temu można w prosty sposób lokalnie odwzorować złożone środowisko produkcyjne złożone z wielu komponentów, które zostaną uruchomione jednym poleceniem. W typowym scenariuszu Dockerfile definiuje sposób budowy obrazu aplikacji, natomiast docker-compose.yml określa, w jaki sposób ten obraz ma być uruchomiony wraz z pozostałymi elementami środowiska, tworząc kompletny i spójny ekosystem uruchomieniowy.

Języki Programowania związane z systemami operacyjnymi

Historia rozwoju systemów operacyjnych jest nierozerwalnie związana z językami programowania, w których tworzono ich kolejne generacje. Najniższe warstwy

systemów – takie jak bootloadery, inicjalizacja sprzętu czy obsługa przerwań – od zawsze były implementowane w Assemblerze, który pozwalał na bezpośrednią kontrolę nad architekturą procesora. Jednak prawdziwym fundamentem, na którym opiera się większość współczesnych systemów, jest język C. To właśnie w nim napisane są jądra takich systemów jak Linux, BSD, Unix, Windows NT czy macOS, co czyni go podstawowym narzędziem w pracy twórców systemów operacyjnych.

Z biegiem czasu zaczęto korzystać również z C++, choć w znacznie mniejszym stopniu niż z C. Ten język, dzięki mechanizmom programowania obiektowego, znajduje zastosowanie przede wszystkim w warstwach wyższego poziomu – na przykład w menedżerach okien, elementach interfejsu graficznego czy niektórych modułach systemów czasu rzeczywistego. W ostatnich latach coraz większą rolę w tej dziedzinie odgrywa Rust, który zyskuje popularność ze względu na bezpieczeństwo zarządzania pamięcią i nowoczesne podejście do programowania systemowego. Rust jest już wykorzystywany do tworzenia sterowników w jądrze Linuksa, a także w projektach takich jak Redox OS czy środowisko graficzne COSMIC Desktop.

W historii pojawiały się też inne próby budowania systemów operacyjnych w różnych językach, jednak większość z nich miała charakter eksperymentalny bądź edukacyjny. W latach 80. i 90. powstawały systemy w Pascalu i Modula-2, jak choćby Lilith czy Oberon, tworzone przez Niklausa Wirtha jako projekty akademickie. Firma Sun Microsystems w 1996 roku opracowała JavaOS, który miał działać na urządzeniach embedded, lecz nie zdobył popularności. Eksperymentowano również z językiem Go, czego przykładem był projekt GopherOS, jednak język ten nie nadaje się dobrze do implementacji najniższych warstw systemowych ze względu na runtime i mechanizm garbage collectora. Apple podejmowało próby wykorzystania Swift w rozwoju macOS i iOS, ale język ten pełni rolę raczej na poziomie aplikacyjnym, podczas gdy sam system nadal opiera się na C, C++ i Objective-C. Popularne języki wysokiego poziomu, takie jak Python czy Ruby, znajdują zastosowanie wyłącznie w warstwie użytkowej, np. w automatyzacji administracyjnej, lecz nie w jądrze. Wyjątek stanowi Ada, która wciąż jest stosowana w systemach czasu rzeczywistego (RTOS), zwłaszcza w lotnictwie i wojsku, choć nie w pełnych systemach operacyjnych, a jedynie w krytycznych modułach.

Rodziny dystrybucji Linux

Świat Linuksa jest niezwykle różnorodny i opiera się na kilku dużych, bazowych rodzinach dystrybucji, z których wywodzą się setki pochodnych systemów. Jedną z najstarszych i najbardziej stabilnych jest Debian, ceniony za konserwatywne podejście do aktualizacji i ogromne repozytoria oprogramowania. To właśnie na nim opierają się tak popularne dystrybucje jak Ubuntu i jego liczne warianty (Kubuntu, Xubuntu, Linux Mint). Drugą dużą gałęzią jest Red Hat/Fedora, czyli dystrybucje skupione wokół środowisk serwerowych i korporacyjnych – z Red Hat Enterprise Linux (RHEL) jako wersją komercyjną i Fedorą jako bardziej nowoczesnym, „testowym" polem doświadczalnym. Alternatywą dla Debiana i Red Hata jest Arch Linux, który stawia na zasadę prostoty i pełnej kontroli użytkownika – system ten dostarcza minimalną bazę, a resztę każdy konfiguruje sam. Arch stał się inspiracją dla innych projektów, m.in. Manjaro, które oferuje prostszy proces instalacji przy zachowaniu zalet rolling release. Warto wspomnieć także o openSUSE, które ma silne wsparcie społeczności i profesjonalne narzędzia administracyjne, oraz o dystrybucjach specjalistycznych, takich jak Kali Linux do testów bezpieczeństwa czy Tails, ukierunkowany na prywatność i anonimowość.

Pod względem popularności i praktyczności najczęściej wybierane przez użytkowników są Ubuntu i jego pochodne, ponieważ łączą łatwość obsługi z bogatym wsparciem społeczności. Na serwerach królują Debian, Ubuntu Server oraz CentOS/AlmaLinux (jako klony Red Hata). Z kolei użytkownicy zaawansowani i entuzjaści często wybierają Archa, aby mieć pełną kontrolę i najnowsze pakiety. Ostatecznie „najlepsza" dystrybucja zależy od potrzeb – dla początkujących zwykle rekomenduje się Linux Mint lub Kubuntu. Mint bazuje na Ubuntu, ale dostarcza wyjątkowo przyjazny interfejs i zestaw preinstalowanych aplikacji, dzięki czemu użytkownik po instalacji otrzymuje kompletny, gotowy do pracy system. Z kolei Kubuntu to alternatywa dla klasycznego Ubuntu, korzystająca ze środowiska graficznego KDE Plasma. Jest ono znacznie lżejsze i bardziej konfigurowalne niż standardowe GNOME, a jednocześnie nowoczesne wizualnie, co sprawia, że Kubuntu często uchodzi za wybór idealny dla osób, które zaczynają swoją przygodę z Linuksem i chcą systemu łatwego w obsłudze, a przy tym elastycznego i szybkiego.

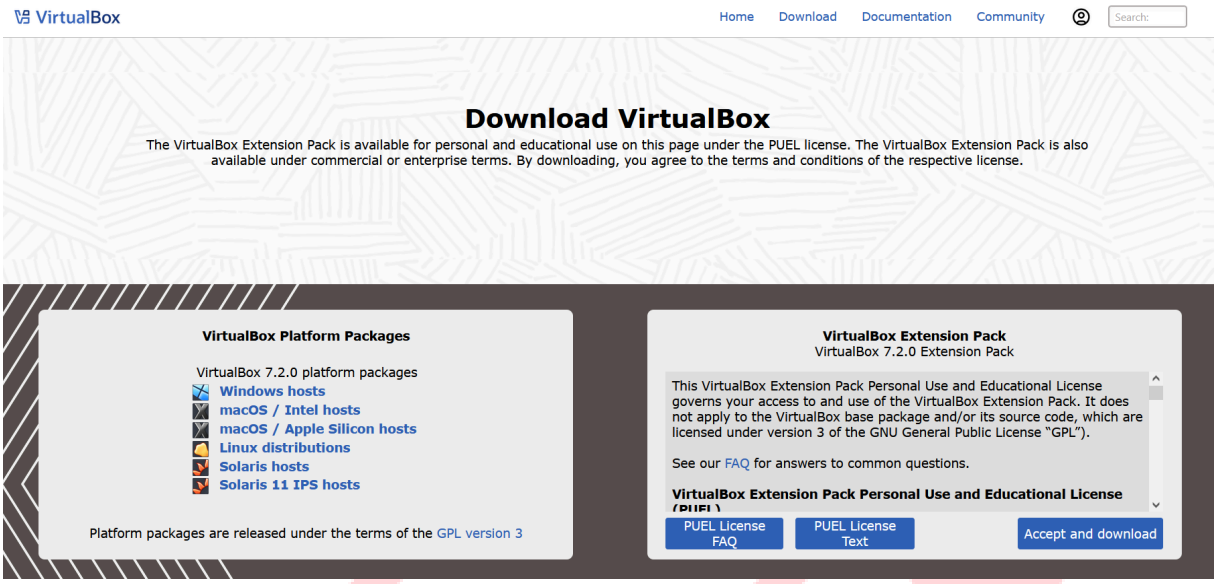
Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Warto wspomnieć również o dystrybucjach, które zyskały szczególną popularność wśród specjalistów IT i twórców oprogramowania. Fedora Workstation uchodzi za system chętnie wybierany przez programistów i architektów ze względu na świeże wersje kompilatorów i bibliotek, a także bliskie powiązanie ze światem Red Hata. Manjaro i jego bardziej zaawansowane pochodne, jak EndeavourOS czy Garuda Linux, oferują wygodę użytkowania przy jednoczesnym dostępie do zasobów Arch Linuksa, co doceniają twórcy gier oraz osoby potrzebujące najnowszych sterowników graficznych i bibliotek multimedialnych. W obszarze uczenia maszynowego i sztucznej inteligencji od lat dominującą rolę pełni Ubuntu, które dzięki oficjalnemu wsparciu NVIDII jest naturalnym wyborem przy pracy z CUDA i frameworkami takimi jak TensorFlow czy PyTorch. W niszy obliczeń naukowych i wysokowydajnych środowisk obliczeniowych można wyróżnić Clear Linux od Intela, zoptymalizowany pod nowoczesne procesory, oraz openSUSE Tumbleweed, oferujący solidne narzędzia administracyjne i stabilność cenioną w zastosowaniach enterprise.

Instalacja Linuxa w środowisku wirtualnym

W ramach niniejszej instrukcji korzystamy z oprogramowania VirtualBox, które można za darmo pobrać ze strony: https://www.virtualbox.org/wiki/Downloads. Rysunek 2 przedstawia strone internetową skąd można pobrać środowisko WirtualBox, które jest sugerowane do wykorzystania w całym niniejszym kursie podstaw systemów operacyjnych.

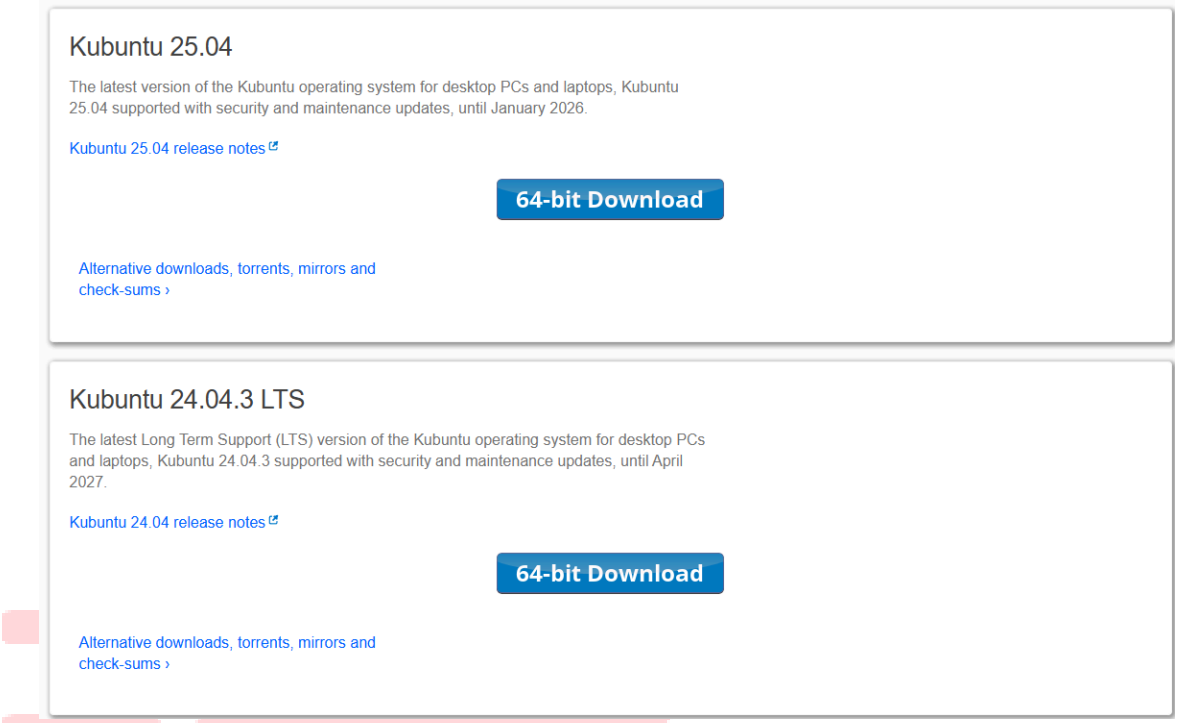


Rys. 2. Strona internetowa VirtualBox, dostępna pod linkiem: https://www.virtualbox.org/wiki/Downloads

W kwestii rozpoczęcia pracy z nowym systemem operacyjnym, można to zrobić pobierając z oficjalnego źródła plik .iso co umożliwi instalacje i konfigurację systemu od zera, lub użycie gotowego obrazu systemu operacyjnego z pliku z rozszerzeeniem ova. Przykładowo dla systemu Kubuntu kwestia wyboru instalatora jest dość prosta. Można je znaleźć na stronie: https://kubuntu.org/getkubuntu/, co zostało przedstawione na rysunku 3.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

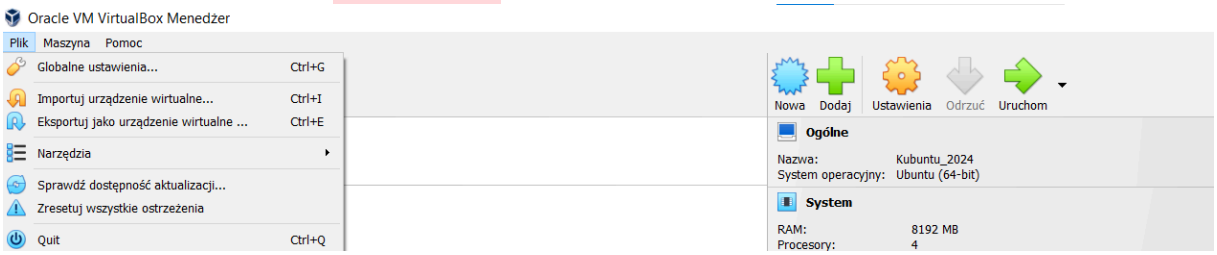


Rys. 3. Strona intenretowa Kubuntu, dostępna pod linkiem https://kubuntu.org/getkubuntu/

Do celów kursu można korzystać z dowolnej dystrybucji systemu Linux, sugeruje się instalacji systemu operacyjnego bazujacego na debianie, dla pełnej zgodności z listingami i instrukcjami zamieszczonymi w niniejszym skrypcie.

Może do tego celu zostać wykorzystamy obraz systemu operacyjnego Kubuntu przedstawiony powyżej, lub np. Kali Linux dostępny za darmo do pobrania ze strony:

https://www.kali.org/get-kali/#kali-platforms



Rys. 4. Widok startowy programu VirtualBox

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Na rysunku 4 przedstawiono widok startowy programu VirtualBox, który umożliwia zarządzanie maszynami wirtualnymi. W tym miejscu można rozpocząć pracę z systemami wirtualnymi na kilka sposobów. Niebieski przycisk „Nowa” pozwala utworzyć maszynę od podstaw (rysunek 5) – student definiuje wówczas jej parametry sprzętowe (takie jak pamięć RAM, liczba procesorów czy rozmiar dysku – rysunek 6), a następnie instaluje system operacyjny z obrazu ISO, podobnie jak na fizycznym komputerze.

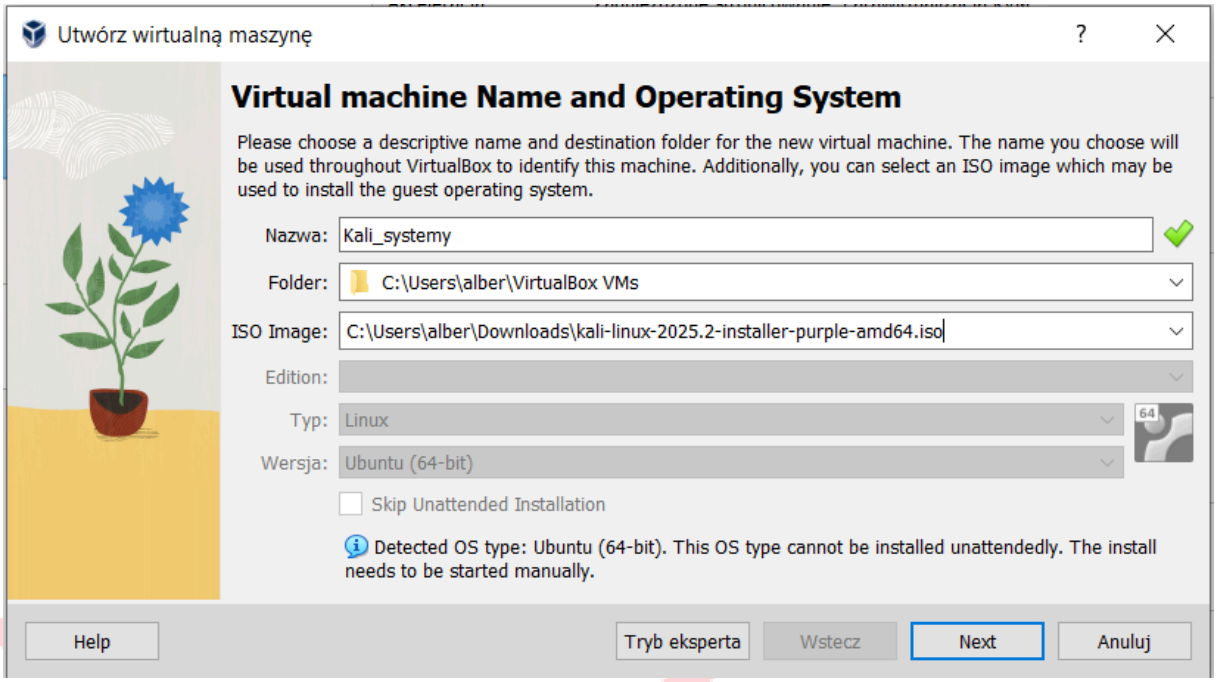
Drugą opcją jest przycisk „Dodaj”, który służy do podłączenia już istniejącej maszyny wirtualnej. Jeżeli student posiada plik konfiguracyjny .vbox lub wirtualny dysk .vdi, może w prosty sposób włączyć go do listy maszyn i kontynuować pracę bez konieczności ponownej instalacji systemu.

Kolejna możliwość to funkcja „Importuj urządzenie wirtualne”, wykorzystywana do wgrywania gotowych maszyn zapisanych w formacie .ova lub .ovf. Takie pliki są najczęściej przygotowanymi szablonami z preinstalowanym systemem, co pozwala natychmiast rozpocząć pracę bez czasochłonnej konfiguracji.

Dostępna jest również opcja „Eksportuj jako urządzenie wirtualne”, która działa odwrotnie – zapisuje skonfigurowaną maszynę w postaci pliku .ova. Dzięki temu można łatwo przenieść całe środowisko na inny komputer lub udostępnić je innym osobom.

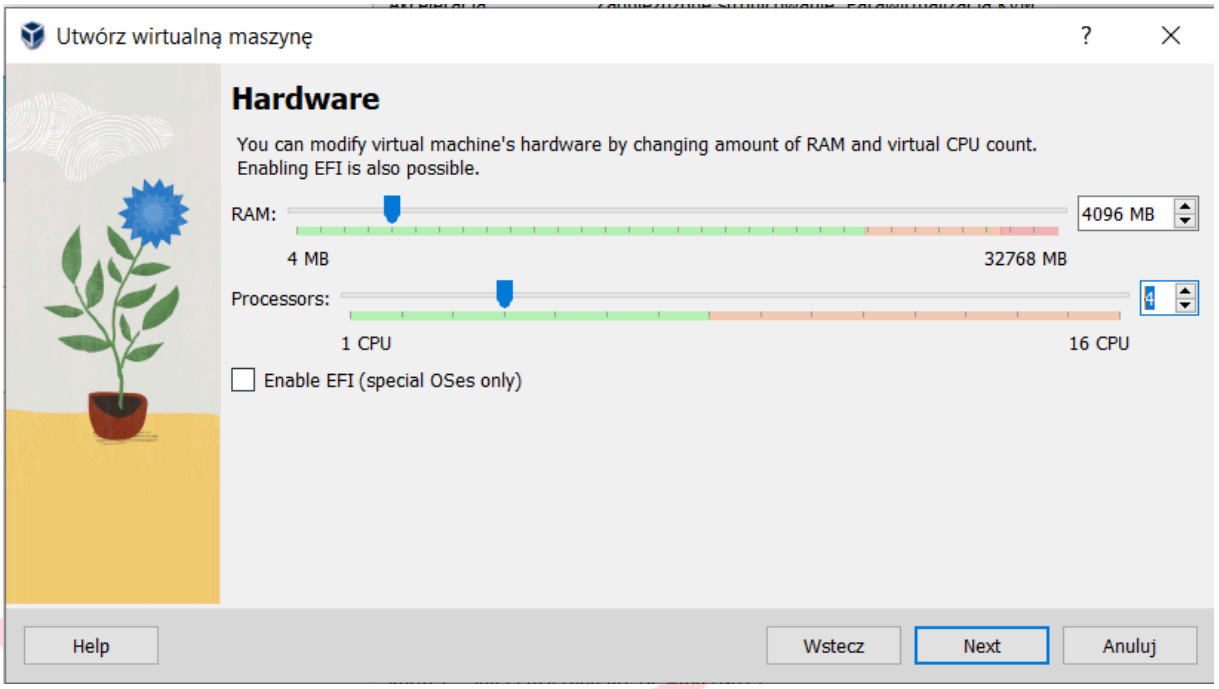
Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00



Rys. 5. Widok programu VirtualBox pozwalający na utworznie nowego obrazu z pliku ISO.

W przypadku konfiguracji maszyny wirtualnej warto zachować kompromis między płynnością działania a obciążeniem komputera gospodarza. Systemy takie jak Kubuntu czy Kali Linux uruchomią się już przy 2 GB pamięci RAM i jednym rdzeniu procesora, jednak w praktyce praca w takim środowisku będzie dość powolna. Optymalnym rozwiązaniem na potrzeby zajęć jest przydzielenie od 2 do 4 rdzeni procesora oraz około 4 GB pamięci RAM, co zapewnia płynność działania interfejsu graficznego, terminala i narzędzi administracyjnych, a jednocześnie nie przeciąża większości komputerów, nawet tych starszych. Dysk wirtualny warto ustawić na poziomie 30–40 GB z dynamicznym przydzielaniem miejsca, co wystarczy zarówno na system, jak i dodatkowe pakiety instalowane podczas laboratoriów.



Rys. 6. Widok programu VirtualBox pozwalający na konfiguracje parametrów maszyny wirtualnej.

Tryby sieciowe w VirtualBox

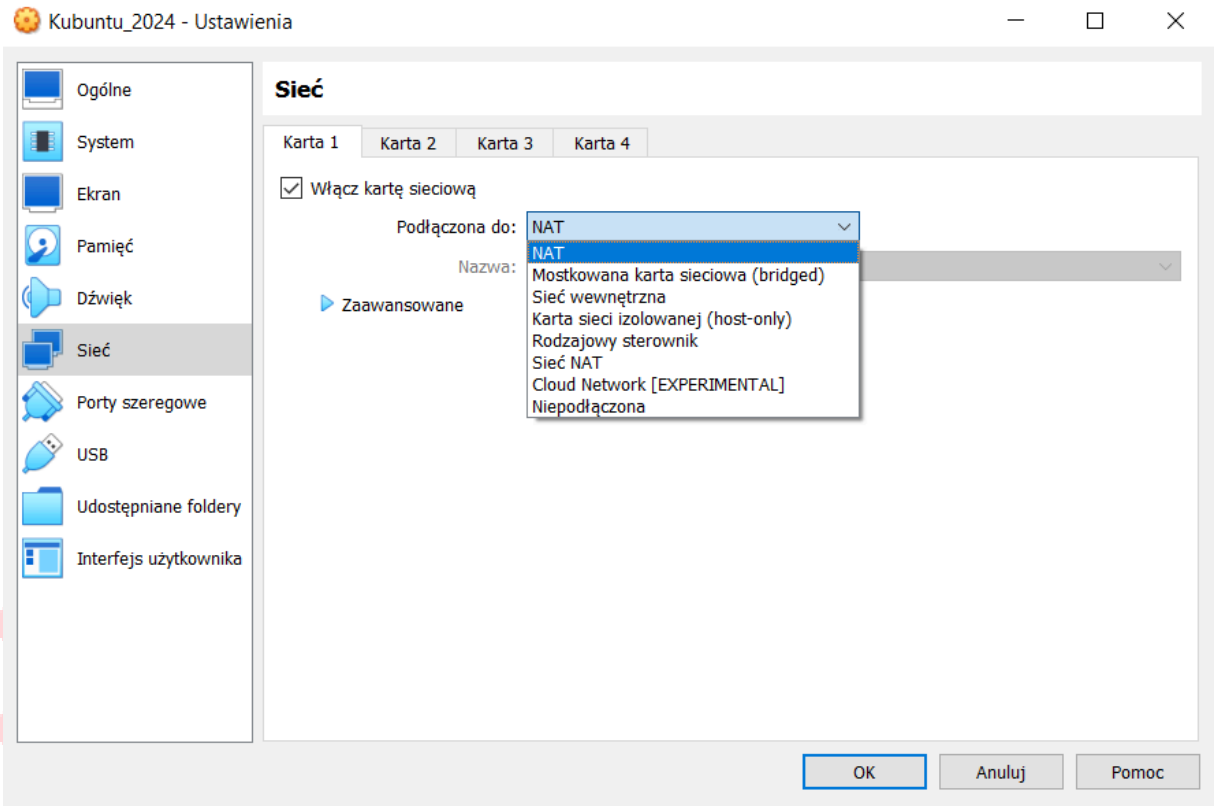
Na rysunku 7 przedstawiono tryby pracy karty sieciowej w programie VirtualBox. To ustawienia, które określają w jaki sposób maszyna wirtualna będzie komunikować się z siecią i hostem. Wybór trybu wpływa na to, czy system uruchomiony wewnątrz maszyny będzie widoczny w sieci lokalnej, będzie miał dostęp do internetu, czy też pozostanie całkowicie odizolowany.

Opis trybów karty sieciowej w VirtualBox:

- NAT (Network Address Translation) – domyślny tryb, w którym maszyna wirtualna korzysta z internetu za pośrednictwem hosta. VM ma dostęp do internetu, ale nie jest bezpośrednio widoczna w sieci lokalnej. To rozwiązanie wygodne i bezpieczne, dobre do podstawowych zastosowań.
- Mostkowana karta sieciowa (Bridged) – VM działa jak osobny komputer w sieci lokalnej. Otrzymuje własny adres IP z tego samego serwera DHCP, co host. Dzięki temu inne urządzenia w sieci widzą maszynę jak zwykły

komputer. Ten tryb jest przydatny do testów serwerów czy usług, które mają być dostępne dla innych użytkowników.

- Sieć wewnętrzna (Internal Network) – maszyny wirtualne mogą komunikować się tylko ze sobą w ramach danej wewnętrznej sieci, nie mają dostępu do hosta ani internetu. Sprawdza się przy tworzeniu izolowanych środowisk testowych.
- Karta sieci izolowanej (Host-Only) – VM ma połączenie tylko z komputerem-gospodarzem (hostem), bez dostępu do internetu. Przydatne, jeśli chcemy testować komunikację między hostem a VM, zachowując izolację od sieci zewnętrznych.
- Rodzajowy sterownik (Generic Driver) – tryb zaawansowany, pozwalający używać dodatkowych, rzadziej spotykanych technologii sieciowych, np. UDP Tunnel. Zwykle stosowany w specyficznych projektach.
- Sieć NAT (NAT Network) – bardziej rozbudowana wersja NAT. Pozwala podłączyć wiele maszyn do jednej wspólnej wirtualnej sieci NAT, umożliwiając im wzajemną komunikację, przy zachowaniu dostępu do internetu.
- Cloud Network [EXPERIMENTAL] – tryb eksperymentalny, pozwalający łączyć maszyny wirtualne poprzez wirtualną sieć działającą w chmurze VirtualBox. Stosowany raczej do testów nowych funkcji.
- Niepodłączona (Not Attached) – karta sieciowa istnieje, ale nie jest podłączona do żadnej sieci. Maszyna nie ma wówczas żadnego połączenia sieciowego.

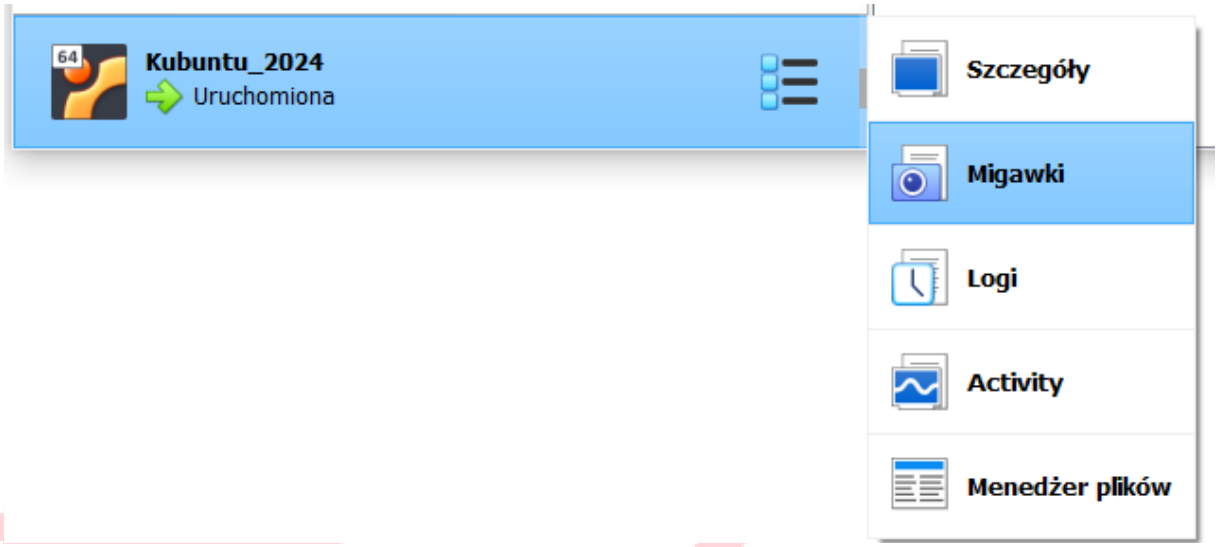


Rys. 7. Widok ustawień programu VirtualBox pozwalający na konfiguracje sieci.

Migawki

Migawki (ang. snapshots) to mechanizm umożliwiający zapisanie aktualnego stanu maszyny wirtualnej, systemu plików lub wolumenu dyskowego w określonym momencie czasu. Działają one podobnie do punktu przywracania, pozwalają „zamrozić" bieżącą konfigurację, pamięć RAM, zawartość dysków oraz stan uruchomionych procesów, aby w razie potrzeby móc powrócić do tego dokładnie samego miejsca. W kontekście maszyn wirtualnych migawka pozwala eksperymentować z systemem bez ryzyka utraty danych. Można np. zainstalować nowe oprogramowanie, przetestować konfigurację lub przeprowadzić aktualizację, a następnie jednym kliknięciem przywrócić poprzedni stan. W VirtualBoxie tworzenie migawki odbywa się z poziomu interfejsu graficznego (Rys. 8.) lub za pomocą polecenia w terminalu. Migawki można również łączyć w łańcuchy, tworząc historię zmian i umożliwiając powrót do dowolnego etapu pracy. W systemach plików, takich

jak ZFS czy Btrfs, mechanizm migawek działa na poziomie bloków danych i pozwala błyskawicznie odtworzyć cały wolumen lub przywrócić pojedyncze pliki.



Rys. 8. Widok tworzenia migawek w VirtualBox

Podstawy pracy z systemem Linux

Podstawowe polecenia powłoki bash stanowią fundament pracy w systemach Linux i pozwalają użytkownikowi w pełni zarządzać zarówno systemem plików, jak i procesami, a także uzyskiwać szczegółowe informacje o środowisku pracy. Już na samym początku istotne jest opanowanie takich komend, jak pwd, ls, cp, mv czy rm, które umożliwiają poruszanie się po katalogach, kopiowanie plików, zmianę ich nazw oraz usuwanie. Ważną rolę odgrywa także echo, które służy nie tylko do wyświetlania tekstu, lecz także do sprawdzania wartości zmiennych środowiskowych, na przykład $PATH. Do szybkiego podglądu zawartości plików używa się komendy cat, natomiast do przeszukiwania całych katalogów – polecenia find, które dzięki dodatkowym warunkom i maskom pozwala na precyzyjne wyszukiwanie danych.

Maski plików, czyli tzw. wildcards, mają tutaj szczególne znaczenie. Znak * zastępuje dowolny ciąg znaków, natomiast ? reprezentuje dokładnie jeden znak. Pozwala to wykonywać operacje na wielu plikach jednocześnie, co czyni pracę w terminalu znacznie efektywniejszą. Przykładowo, polecenie rm *.log usuwa wszystkie pliki

zakończone rozszerzeniem .log, a ls data??.txt pokaże wszystkie pliki, których nazwa zaczyna się od słowa „data” i kończy na dwóch dodatkowych znakach. Maski mogą być wykorzystywane w praktycznie każdym poleceniu operującym na plikach, nie tylko w ls czy rm, ale także w cp, mv, cat, chmod, tar czy find. Dzięki nim można selektywnie wybierać zestawy plików według wzorców nazw, rozszerzeń lub fragmentów tekstu. Oprócz podstawowych symboli * i ?, powłoka Linuksa obsługuje również wyrażenia w nawiasach kwadratowych [], które umożliwiają wskazanie konkretnych zakresów lub grup znaków. Na przykład ls raport[1-5].txt wyświetli tylko pliki raport1.txt do raport5.txt, a cp test[ab]* kopia/ skopiuje wszystkie pliki, których nazwa zaczyna się od „testa” lub „testb”. Maski można łączyć, tworząc bardziej złożone wzorce, np. rm *[0-9].log usunie pliki z cyfrą przed rozszerzeniem .log, a mv *.{jpg,png,gif} obrazki/ przeniesie wszystkie pliki graficzne do wskazanego katalogu. Warto pamiętać, że maski są rozwijane przez powłokę przed wykonaniem polecenia, dlatego ich działanie zależy od aktualnej lokalizacji w systemie plików i faktycznej zawartości katalogu.

Poza tymi podstawowymi narzędziami istnieje szereg innych komend, które są niezbędne w codziennej pracy administratora czy użytkownika systemu Linux. Do zmiany bieżącego katalogu służy cd, przy czym można posługiwać się zarówno ścieżkami absolutnymi (cd /etc), jak i względnymi (cd ..). Tworzenie katalogów umożliwia polecenie mkdir, a ich usuwanie – rmdir. Do tworzenia pustych plików lub aktualizacji daty modyfikacji wykorzystuje się touch. Podgląd zawartości plików w sposób bardziej interaktywny niż w przypadku cat zapewniają less, tail i head. Wyszukiwanie wzorców w plikach, kluczowe w analizie logów, realizuje polecenie grep.

Zarządzanie dostępem i bezpieczeństwem plików wymaga znajomości chmod (nadawanie uprawnień) oraz chown (zmiana właściciela). Praca z procesami systemowymi odbywa się przy użyciu poleceń ps, top lub bardziej rozbudowanego htop, a ich kończenie możliwe jest dzięki kill oraz killall. Nie można także pominąć narzędzi dokumentacyjnych – man pozwala otworzyć podręcznik dowolnej komendy (np. man ls), a history umożliwia przeglądanie i ponowne uruchamianie ostatnio wpisanych poleceń. Szczególnie istotne jest także sudo, które pozwala wykonywać operacje z uprawnieniami administratora i tym samym zarządzać całym systemem.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Praca w powłoce bash może być znacząco usprawniona dzięki prostym trikom, które pozwalają oszczędzić czas i uniknąć zbędnego wpisywania komend. Warto pamiętać, że klawisz strzałki w górę umożliwia przeglądanie historii poleceń – zamiast wpisywać to samo wielokrotnie, można szybko odszukać i ponownie uruchomić wcześniejszą komendę. Z kolei naciśnięcie TAB po wpisaniu części nazwy polecenia lub katalogu automatycznie uzupełnia resztę, a gdy możliwości jest kilka, bash wyświetli dostępne opcje do wyboru. Przydatną funkcją jest również możliwość kopiowania i wklejania tekstu w konsoli: zaznaczony fragment można wkleić w inne miejsce za pomocą środkowego przycisku myszy, co bywa niezawodne tam, gdzie klasyczne skróty CTRL+C i CTRL+V nie działają.

Klawisz CTRL+R pozwala przeszukać historię poleceń – wystarczy wpisać fragment szukanej komendy, aby szybko ją odnaleźć i ponownie wykonać. Skrót CTRL+A przenosi kursor na początek linii, a CTRL+E na jej koniec, co przyspiesza edycję dłuższych poleceń. Z kolei CTRL+U usuwa wszystko przed kursorem, a CTRL+K – wszystko po nim. Bardzo przydatne bywa także !!, które powtarza ostatnie polecenie, a w połączeniu z sudo pozwala błyskawicznie wykonać je ponownie z uprawnieniami administratora, np. sudo !!.

Manual systemowy

Manual systemowy (ang. manual pages, w skrócie man pages) to wbudowana dokumentacja systemów Unix/Linux, dostępna bezpośrednio z poziomu terminala. Dzięki niemu użytkownik może w każdej chwili sprawdzić opis polecenia, funkcji, biblioteki czy pliku konfiguracyjnego, bez potrzeby korzystania z internetu. Polecenie man (od manual) otwiera odpowiednią stronę podręcznika i pozwala zapoznać się ze szczegółami działania narzędzi systemowych.

Każda strona manuala posiada określoną strukturę. Najczęściej można znaleźć tam:

- NAME – nazwę i krótki opis polecenia lub funkcji,
- SYNOPSIS – sposób użycia, czyli ogólną składnię,
- DESCRIPTION – szczegółowe wyjaśnienie,
- OPTIONS – dostępne opcje i argumenty,
- EXAMPLES – przykłady użycia (nie zawsze występują),

- SEE ALSO – odwołania do pokrewnych poleceń lub dokumentacji.

Manual podzielony jest na numerowane sekcje, co pozwala rozróżniać elementy o tej samej nazwie, ale innym przeznaczeniu. Najczęściej spotykany podział to:

1 – polecenia użytkownika (np. ls, cp, grep),

2 – wywołania systemowe (np. open, read, write),

3 – funkcje biblioteczne języka C (np. printf, malloc),

4 – pliki specjalne i urządzenia,

5 – formaty plików i konwencje (np. /etc/passwd),

6 – gry i programy rozrywkowe,

7 – różne zagadnienia, konwencje i standardy,

8 – polecenia administracyjne i narzędzia systemowe (np. mount, shutdown).

Dzięki temu można łatwo rozróżnić, czy interesuje nas opis polecenia powłoki, czy funkcji programistycznej. Na przykład man 1 printf otworzy dokumentację polecenia powłoki, a man 3 printf – funkcji w języku C.

Jak wyszukiwać w manualach? Istnieją trzy podstawowe sposoby:

- Klasyczne użycie man – podajemy nazwę polecenia, np. man mkdir.
- Opcja -k – wyszukuje słowa kluczowe w całym manualu, np. man -k permissions.
- Polecenie apropos – podobne do -k, ale przeszukuje także opisy, np. apropos networking.

Przykłady użycia opcji man

- Określenie sekcji: man 2 intro – wyświetla stronę intro z sekcji system calls.
- Sprawdzenie sekcji polecenia: man -f ls – pokazuje, w których sekcjach występuje dokumentacja ls.

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

- Przegląd wszystkich stron: man -a intro – wyświetla kolejno wszystkie strony intro.
- Wyszukiwanie wyrażeń: man -k cd – szuka wpisów zawierających frazę „cd”.
- Sprawdzenie lokalizacji pliku manuala: man -w ls – pokazuje ścieżkę pliku manuala dla ls.
- Wymuszenie rozróżniania wielkości liter: man -I printf – wyszukuje dokładnie printf, a nie Printf.

Manual wykorzystuje program less jako przeglądarkę tekstu, co oznacza, że można przewijać treść strzałkami, wyszukiwać słowa kluczowe (/fraza w dół, ?fraza w górę), powtarzać wyszukiwanie (n, N), wywołać pomoc (h) czy zakończyć pracę (q).

Znaczenie manuala jest trudne do przecenienia. To narzędzie zawsze dostępne offline, niezależnie od internetu, a przy tym gwarantuje zgodność informacji z wersją systemu i programów zainstalowanych lokalnie. Uczy także samodzielności i pracy z dokumentacją – jednej z podstawowych umiejętności w systemach Unix/Linux.

Polecenie Rename

Polecenie rename służy do masowej zmiany nazw plików, używając wzorców i zamiany tekstu, a nie tylko przenoszenia pojedynczych plików (jak mv). Bazowa składnia, dla wersji Perl-owej rename, wygląda tak:

rename [opcje] 's/wzorzec_zamiany/zamiennik/' pliki...

gdzie:

- s/…/…/ to operator substitute, który określa, co zostaje znalezione, a na co ma być zamienione.
- wzorzec_zamiany - fragment nazwy, który chcemy modyfikować (mogą tu być użyte metaznaki regularne).
- zamiennik - nowy fragment, który zastępuje wzorzec.
- maska_plików - lista plików do przekształcenia (np. *.txt, file?.log).

Dzięki temu rename potrafi realizować różnorodne operacje:

- Zmiana rozszerzeń: np. .txt → .bak poprzez rename 's/\.txt$/.bak/' *.txt.

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

- Usuwanie lub zastępowanie fragmentów nazwy: np. usunięcie prefiksu temp_ przez rename 's/^temp_//' temp_*.*.
- Zmiana prefiksu lub fragmentu wewnętrznego: np. slang_ → sl_ za pomocą rename 's/slang_/sl_/' *.txt.
- Translacja znaków: np. zamiana małych liter na wielkie - rename 'y/a-z/A-Z/' *.txt.

Warto też pamiętać, że istnieją różne implementacje rename w różnych systemach Linux, które mogą obsługiwać różne poziomy funkcjonalności. Zaleca się najpierw przetestować działanie rename przy pomocy opcji -n, która nie wykonuje zmian, a jedynie wyświetla symulowane rezultaty. Na przykład:

```
rename -n 's/\.txt$/.bak/' *.txt
```

Ćwiczenia samodzielne

1. Utwórz nową maszynę wirtualną w VirtualBoxie z wybraną dystrybucją Linuxa (np. Kali lub Kubuntu).
2. Spróbuj użyć gotowego pliku VDI (np. z Kali Linuxem) i uruchomić go jako istniejący dysk.
3. Zrób klona oraz migawkę maszyny wirtualnej, czym się różnią?
4. Przetestuj działanie dwóch maszyn wirtualnych równocześnie – uruchom dwie VM i sprawdź, jak obciążają CPU hosta.
5. Skonfiguruj zasoby maszyny – przydziel 2 rdzenie CPU i 4 GB RAM, a następnie sprawdź, jak zmiana tych parametrów wpływa na działanie systemu.
6. Przetestuj tryby sieciowe w VirtualBoxie: NAT → sprawdź, czy masz dostęp do internetu, Bridged → sprawdź adres IP i porównaj z hostem.
7. Zbadaj różnicę między VM a kontenerem – zainstaluj Dockera w swojej maszynie i uruchom prosty kontener (np. Ubuntu).
8. Utwórz plik dockerfile oraz docker compose z prostą aplikacją flask, bazując na części teoretycznej niniejszej instrukcji.
9. W katalogu domowym utwórz główny katalog projektu o nazwie Projekt i wejdź do niego. Następnie w jednym poleceniu stwórz podkatalogi: zrodla, testy, dokumentacja, logi.
10. W katalogu zrodla utwórz puste pliki reprezentujące kod: skrypt_glowny.py, biblioteka_a.py, biblioteka_b.py, helper_01.sh.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

11. W katalogu testy utwórz pliki logów: raport_01.log, raport_02.log, error_2025.txt.
12. Kopiowanie selektywne: Skopiuj wyłącznie pliki z rozszerzeniem .log z katalogu testy do katalogu logi, używając maski *.
13. Zainstaluj pakiet rename aby za pomocą jednego polecenia zmienić wszystkie rozszerzenia plików z .log na .backup.
14. Usuwanie z maską: Usuń wszystkie pliki w katalogu zrodla, których nazwa zaczyna się na biblioteka, stosując odpowiednią maskę. Przed usunięciem użyj polecenia ls z tą samą maską, aby sprawdzić, które pliki zostaną usunięte.
15. Używając maski napisz polecenie które w sposób masowy skopiuje wszystkie pliki z jednego katalogu, dzielące wspólny wzorzec jakim jest nazwa kończąca się cyfrą, do innego wskazanego katalogu.
16. Wyszukiwanie Złożone: Użyj polecenia find, aby w katalogu Projekt odnaleźć wszystkie pliki, których nazwa:
    - zaczyna się na literę r
    - oraz mają rozmiar większy niż 0 bajtów (Na potrzeby tego zadania można utworzyć plik wypełniony tekstem).
    - Wynik tego wyszukiwania (ścieżki do plików) przekieruj do nowego pliku o nazwie raport_wyszukiwania.txt w katalogu dokumentacja.
17. Wyszukiwanie Wzorca (grep): Dodaj kilka linii tekstu do pliku raport_wyszukiwania.txt (użyj echo lub edytora), z których przynajmniej dwie zawierają słowo Plik, a jedna słowo Błąd. Następnie użyj polecenia grep do:
    - Wyświetlenia wszystkich linii zawierających słowo Plik, ignorując wielkość liter.
    - Wyświetlenia linii zawierającej słowo Błąd oraz dwóch linii kontekstu znajdujących się przed nią.
18. Użyj polecenia man, aby wyświetlić dokumentację polecenia grep. Odszukaj sekcję SYNOPSIS i zacytuj składnię. Odszukaj w dokumentacji sekcję SEE ALSO i podaj przynajmniej dwa polecenia, które są tam wymienione jako pokrewne.
19. Wywołaj manual dla polecenia grep i wykonaj następujące kroki nawigacyjne, notując klawisze użyte do każdej operacji:
    - W otwartym manualu dla grep odszukaj frazę: context.
    - Ponów wyszukiwanie frazy context do następnego jej wystąpienia.
    - Ponów wyszukiwanie frazy context do poprzedniego jej wystąpienia.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

- Odszukaj sekcję EXAMPLES. Jakiego pojedynczego klawisza (lub kombinacji) należy użyć, aby przejść bezpośrednio do końca strony manuala?
- Jakiego klawisza należy użyć, aby wyświetlić pełną pomoc/listę komend w programie less (przeglądarce manuala)? Opuść pomoc i zamknij manual.

Pytania pomocnicze

1. Czym różni się hypervisor typu 1 od hypervisora typu 2?
2. Jakie są zalety wirtualizacji w środowiskach developerskich?
3. Wyjaśnij różnicę między snapshotem a klonem maszyny wirtualnej.
4. Na czym polega wirtualizacja sieci w VirtualBox?
5. Jakie są ograniczenia wirtualizacji w porównaniu do konteneryzacji?
6. Na czym polega różnica między NAT a Bridged Networking w VM?
7. Jak wyświetlić dokumentację polecenia w systemie Linux?
8. Czym jest maska i do czego służy podczas używania poleceń z linii komend?
9. Do czego służy polecenie rename?
10. Czym jest dockerfile?

Podsumowanie

Podczas pierwszego laboratorium studenci zetknęli się z fundamentami pracy w środowisku systemów operacyjnych, poznając zarówno zagadnienia związane z wirtualizacją, jak i podstawy obsługi systemu w powłoce tekstowej. Ćwiczenia rozpoczęły się od konfiguracji VirtualBoxa, w tym doboru odpowiednich zasobów sprzętowych dla maszyn wirtualnych oraz omówienia różnych trybów działania kart sieciowych. Instalacja wybranego systemu umożliwiła praktyczne prześledzenie pełnego cyklu uruchamiania systemu. Dzięki temu studenci zrozumieli rolę systemu operacyjnego jako warstwy pośredniczącej pomiędzy sprzętem a użytkownikiem.

Istotnym elementem zajęć była nauka korzystania z podstawowych poleceń powłoki Bash, takich jak pwd, ls, cd, cp, mv, rm, touch czy mkdir, które pozwalają na swobodne poruszanie się po systemie plików i wykonywanie elementarnych operacji. Studenci poznali także mechanizm masek plikowych, który umożliwia wykonywanie operacji na wielu plikach jednocześnie. Ćwiczenia praktyczne uświadomiły

uczestnikom, jak szybko analizować zawartość plików i przeszukiwać dane w terminalu, co stanowi podstawę dalszej pracy administracyjnej.

Szczególną uwagę poświęcono pracy z manualem systemowym (man), który stanowi integralną część każdego systemu uniksowego. Studenci zapoznali się ze strukturą stron podręcznika oraz nauczyli się nawigacji i wyszukiwania w dokumentacji. Dzięki temu mogli zrozumieć, że manual systemowy jest nie tylko źródłem wiedzy offline, ale również narzędziem kształcącym samodzielność i umiejętność pracy z dokumentacją techniczną.

Całość zajęć unaoczniła studentom, że wirtualizacja i znajomość powłoki są fundamentem pracy w środowiskach informatycznych – od edukacji, przez administrację, po profesjonalne wykorzystanie w serwerowniach i chmurach obliczeniowych. Opanowanie podstawowych poleceń oraz umiejętność korzystania z manuala systemowego przygotowuje uczestników do dalszych, bardziej złożonych ćwiczeń, stanowiąc solidny punkt wyjścia do pełniejszego zrozumienia działania systemów operacyjnych.

Literatura

1. Stallings W.: Systemy operacyjne : struktura i zasady budowy, Wyd. Naukowe PWN, 2006.
2. Tanenbaum A. S.: Systemy operacyjne. Wydanie IV, Helion, 2015.
3. Silbershatz A., Galvin P. B., Gagne G.: Podstawy systemów operacyjnych, Wydanie 7, WNT 2006.
4. Ebrahim M., Mallett A.: Skrypty powłoki systemu Linux. Zagadnienia zaawansowane, Wydanie II, Helion 2019.
5. Nemeth E., i in.: Unix i Linux. Przewodnik administratora systemów. Wydanie V, Helion 2018.
6. Love R.: Linux programowanie systemowe, Helion 2008.
7. Bar M.: Linux : systemy plików, Wyd. RM, 2002.
8. The Linux Command Handbook – Learn Linux Commands for Beginners https://www.freecodecamp.org/news/the-linux-commands-handbook/ (data ostatniego dostępu 25.09.25)

Fundusze Europejskie dla Rozwoju Społecznego | Rzeczpospolita Polska | Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Materiały do przedmiotu:

Podstawy Systemów Operacyjnych

Laboratorium nr 2:

Powłoki systemów operacyjnych, podstawowa obsługa systemów plików oraz operacje na plikach i katalogach

Autor:

Albert Rachwał

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Drugie laboratorium koncentruje się na powłoce systemowej, czyli podstawowym interfejsie użytkownika w systemach uniksowych. To właśnie powłoka interpretuje polecenia i umożliwia bezpośrednią komunikację z systemem operacyjnym, a jednocześnie daje ogromne możliwości automatyzacji pracy. Studenci poznają hierarchię katalogów w systemach Linux oraz znaczenie kluczowych lokalizacji, takich jak katalog główny, katalog użytkownika czy katalogi zawierające pliki konfiguracyjne. Omawiane są podstawowe operacje na plikach i katalogach: tworzenie, kopiowanie, przenoszenie, usuwanie oraz nadawanie uprawnień. Szczególną uwagę zwraca się na mechanizm praw dostępu, właścicieli plików oraz różnice pomiędzy linkami twardymi i symbolicznymi. W ramach zajęć uczestnicy uczą się także korzystać ze zmiennych środowiskowych i aliasów, które pozwalają spersonalizować powłokę i usprawnić codzienną pracę. Dzięki temu studenci zdobywają fundament niezbędny do sprawnego poruszania się po systemie oraz wykonywania bardziej zaawansowanych operacji w kolejnych laboratoriach. Ważnym elementem jest również wprowadzenie do systemu kontroli wersji Git, który jest jednym z podstawowych narzędzi programistów i administratorów. Studenci znają podstawowe polecenia związane z zakładaniem i zarządzaniem własnym repozytorium, a także poznają sposób przechowywania i wersjonowania plików. Dzięki temu mogą od razu zrozumieć, jak praca z plikami w systemie operacyjnym łączy się z praktyką inżynierską w projektach zespołowych.

Cel ćwiczenia/laboratorium

- Zrozumienie roli powłoki systemu operacyjnego jako interfejsu między użytkownikiem a jądrem systemu.
- Poznanie podstawowych rodzajów powłok (sh, bash, zsh) i ich cech charakterystycznych.
- Umiejętność uruchamiania poleceń w terminalu i interpretacji ich wyników.
- Nabycie umiejętności pracy z systemem plików w Linuxie.
- Zrozumienie hierarchii katalogów zgodnie ze standardem FHS (Filesystem Hierarchy Standard).
- Rozróżnianie typów plików w systemie Linux.
- Opanowanie podstawowych poleceń do poruszania się po systemie plików (pwd, ls, cd).
- Wykonywanie operacji na plikach (touch, cp, mv, rm, cat, file).

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

- Wykonywanie operacji na katalogach (mkdir, rmdir, tree).
- Umiejętność korzystania z poleceń do wyszukiwania plików (find, locate, which, whereis).
- Poznanie podstawowych narzędzi dostępnych z poziomu powłoki.
- Zrozumienie różnicy między dowiązaniem symbolicznym a twardym.
- Tworzenie i usuwanie linków (ln, ln -s).
- Umiejętność wyświetlania i edycji zawartości plików.

Instrukcja laboratoryjna/ćwiczeniowa

Powłoka systemowa

Powłoka systemowa (ang. shell) to jeden z najważniejszych elementów systemu operacyjnego, stanowiący interfejs pomiędzy użytkownikiem a jądrem systemu. Dzięki niej człowiek może wydawać polecenia komputerowi i otrzymywać odpowiedzi w postaci wyników działania programów. Można powiedzieć, że powłoka pełni rolę tłumacza – interpretuje wpisane przez użytkownika komendy i przekazuje je dalej do systemu, a następnie prezentuje rezultaty w zrozumiałej formie.

Powłoka składa się zasadniczo z dwóch elementów. Pierwszym jest interpreter poleceń, który odpowiada za odczytywanie wpisywanych komend i ich wykonywanie. Drugim jest język skryptowy, który umożliwia tworzenie plików zawierających sekwencje poleceń. Dzięki temu powłoka nie jest tylko prostym „terminalem”, ale również wyspecjalizowanym narzędziem automatyzacji pracy, pozwalającym budować skrypty do wykonywania zadań administracyjnych, przetwarzania danych czy uruchamiania aplikacji w określonych warunkach.

Celem powłoki jest zapewnienie użytkownikowi wygodnego i elastycznego środowiska pracy. W odróżnieniu od graficznych interfejsów użytkownika (GUI), powłoki tekstowe dają pełną kontrolę nad systemem, umożliwiają wykonywanie operacji, które w interfejsie graficznym byłyby zbyt czasochłonne lub wręcz niemożliwe. To właśnie dlatego administratorzy i zaawansowani użytkownicy systemów Linux i Unix preferują pracę w powłoce – pozwala ona szybko wykonywać skomplikowane operacje, automatyzować powtarzalne zadania i sprawnie zarządzać systemem.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Istnieje wiele rodzajów powłok, które rozwijały się wraz z systemami operacyjnymi. Klasycznym przykładem jest sh (Bourne Shell), jedna z pierwszych powłok systemowych w systemach Unix. Jej rozwinięciem i dziś najczęściej używaną wersją jest Bash (Bourne Again Shell), dostępna niemal w każdej dystrybucji Linuksa. Inne popularne powłoki to Zsh (Z Shell), ceniona za rozbudowane funkcje, takie jak autouzupełnianie poleceń czy możliwość łatwej personalizacji, oraz Fish (Friendly Interactive Shell), nastawiona na prostotę obsługi i intuicyjność. Każda z nich ma swoje cechy szczególne, ale cel pozostaje ten sam – zapewnić użytkownikowi efektywny sposób komunikacji z systemem. W ramach zajęć skupiamy się przede wszystkim na powłoce Bash, ponieważ jest ona standardem w większości współczesnych dystrybucji systemu Linux i stanowi praktyczną podstawę do dalszej nauki.

Powłoki systemowe można więc traktować jako kręgosłup pracy w środowisku Unix/Linux. Dzięki nim użytkownik otrzymuje pełnię możliwości, jakie oferuje system, a zarazem narzędzie do automatyzacji i usprawniania codziennych zadań. Dla administratora czy programisty opanowanie powłoki to fundament pracy, dla zwykłego użytkownika – możliwość zrozumienia, jak naprawdę działa komputer.

System plików

System plików to fundament organizacji danych w systemie operacyjnym. Można go porównać do biblioteki, to on decyduje, gdzie i w jaki sposób zostaną zapisane informacje, jak będą one uporządkowane, a także jak użytkownik i programy będą mogły z nich korzystać. Dzięki systemowi plików przechowywanie danych na dyskach, pamięciach USB czy innych nośnikach jest nie tylko możliwe, ale i efektywne.

Podstawowym zadaniem systemu plików jest nadawanie struktury zapisanym informacjom. Dane przechowywane w komputerze fizycznie istnieją jako ciąg bajtów na dysku, ale bez odpowiedniego sposobu organizacji byłyby one bezużyteczne. System plików pozwala pogrupować te dane w postaci plików i katalogów, nadać im nazwy, a także przechowywać dodatkowe informacje – metadane, takie jak data utworzenia, ostatniej modyfikacji, właściciel czy uprawnienia dostępu. Dzięki temu

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

użytkownik widzi dane w uporządkowanej formie, a nie jako chaotyczny strumień zer i jedynek.

W systemach uniksowych i linuksowych obowiązuje hierarchiczna struktura katalogów, zwana standardem FHS (Filesystem Hierarchy Standard). Wszystko zaczyna się od katalogu głównego oznaczonego ukośnikiem /. To właśnie z niego rozgałęziają się wszystkie inne katalogi. Wśród nich najważniejsze to: /etc zawierający pliki konfiguracyjne systemu, /home przechowujący katalogi domowe użytkowników, /var z logami i plikami tymczasowymi czy /usr z programami i bibliotekami systemowymi. Taka struktura ułatwia orientację w systemie, a jednocześnie pozwala utrzymać porządek i rozdzielić różne typy danych. Warto wspomnieć także o katalogach takich jak /bin, który zawiera podstawowe polecenia systemowe dostępne dla wszystkich użytkowników, oraz /tmp, w którym przechowywane są pliki tymczasowe usuwane po restarcie systemu.

System plików nie ogranicza się jednak tylko do samego zapisu i odczytu danych. Odpowiada także za bezpieczeństwo i kontrolę dostępu. W systemach Linux każdy plik i katalog ma określonego właściciela, grupę oraz zestaw uprawnień – możliwość odczytu, zapisu i wykonania. Dzięki temu administrator może decydować, kto i w jakim zakresie ma dostęp do poszczególnych zasobów. To mechanizm, który chroni system przed nieautoryzowanymi zmianami i utratą integralności danych.

W praktyce istnieje wiele różnych implementacji systemów plików. Starsze systemy, takie jak FAT, nadal można spotkać w prostych nośnikach przenośnych. W systemach Linux najczęściej stosuje się nowocześniejsze rozwiązania – ext4, XFS, Btrfs – które oferują większą wydajność, obsługę ogromnych przestrzeni dyskowych, a także dodatkowe funkcje, jak migawki (snapshots) czy kontrola integralności danych. Każdy system plików ma swoje zalety i ograniczenia, dlatego wybór konkretnego zależy od przeznaczenia środowiska – inne mechanizmy sprawdzą się w domowym komputerze, a inne w serwerze baz danych czy w chmurze. System plików jest więc kluczowym elementem systemu operacyjnego – to dzięki niemu komputer potrafi przechowywać, organizować i chronić dane.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Operacje na plikach i katalogach w systemie Linux

Praca z systemem plików w systemach Linux opiera się na wykonywaniu różnorodnych operacji na plikach i katalogach. Do najważniejszych należą tworzenie, kopiowanie, przenoszenie, usuwanie, a także nadawanie im odpowiednich uprawnień. Każda z tych czynności odbywa się w powłoce za pomocą zestawu poleceń, które umożliwiają sprawne zarządzanie strukturą danych.

Tworzenie i usuwanie plików oraz katalogów Podstawowym poleceniem do utworzenia pustego pliku jest touch, które tworzy nowy plik lub aktualizuje datę modyfikacji istniejącego. Nowy katalog można utworzyć przy pomocy mkdir, a usunąć za pomocą rmdir (dla pustych katalogów) lub rm -r (dla katalogów wraz z ich zawartością). Dzięki temu użytkownik może w prosty sposób budować i modyfikować strukturę katalogów w systemie.

Kopiowanie i przenoszenie danych Do kopiowania plików i katalogów służy polecenie cp, natomiast przenoszenie lub zmiana nazwy odbywa się za pomocą mv. W obu przypadkach można wskazać źródło i miejsce docelowe, a także opcje takie jak zachowanie atrybutów plików (-p) czy kopiowanie rekursywne katalogów (-r). Operacje te pozwalają na reorganizację systemu plików i dostosowanie jego struktury do potrzeb użytkownika. Przy pracy z uprawnieniami niezwykle pomocne jest polecenie ls -l, które w szczegółowej formie pokazuje właściciela pliku, grupę oraz przypisane prawa dostępu.

Identyfikacja plików i katalogów

W pracy administratora systemu Linux niezwykle ważną rolę odgrywają narzędzia pozwalające szybko odnajdywać pliki i określać ich typ. Polecenie find umożliwia wyszukiwanie plików i katalogów w określonej lokalizacji z uwzględnieniem wielu kryteriów, takich jak nazwa, rozmiar, właściciel czy data modyfikacji, dzięki czemu sprawdza się w diagnostyce i porządkowaniu systemu plików. Przykładowo, polecenie find /etc -name "*.conf" wyszuka wszystkie pliki konfiguracyjne w katalogu /etc, a find /home -type f -size +10M pokaże pliki większe niż 10 MB w katalogu domowym użytkowników. Dla kontrastu, locate korzysta z gotowej bazy indeksów aktualizowanej przez system, co sprawia, że działa błyskawicznie, choć wyniki mogą

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

być nieaktualne, jeśli baza nie została odświeżona. Typowe wywołanie locate passwd natychmiast pokaże wszystkie pliki zawierające w ścieżce słowo „passwd”, co jest znacznie szybsze niż ręczne przeszukiwanie dysku.

Narzędzie file pozwala sprawdzić rzeczywisty typ pliku na podstawie jego nagłówka, a nie tylko rozszerzenia – dzięki temu można odróżnić np. dokument tekstowy od pliku wykonywalnego. Polecenie file /bin/ls zwróci informację, że jest to plik wykonywalny ELF, natomiast file dokument.txt potwierdzi, że dany plik zawiera zwykły tekst ASCII lub UTF-8. Z kolei polecenie which wskazuje dokładną lokalizację programu w systemie, przeszukując katalogi wymienione w zmiennej środowiskowej PATH. Na przykład which python3 pokaże, w którym katalogu znajduje się interpreter Pythona, a which ls ujawni lokalizację polecenia ls w katalogu /bin/ls.

Razem te narzędzia tworzą zestaw umożliwiający sprawne wyszukiwanie, klasyfikowanie i identyfikację plików. Dzięki ich znajomości administrator może szybciej diagnozować problemy, weryfikować konfiguracje i utrzymywać porządek w systemie.

Uprawnienia do plików i katalogów

Uprawnienia dostępu Jedną z kluczowych cech systemu plików Linux jest mechanizm uprawnień. Każdy plik i katalog ma przypisanego właściciela (user), grupę (group) oraz zestaw uprawnień dla właściciela, grupy i pozostałych użytkowników (others). Uprawnienia te definiują, czy dany podmiot może plik odczytać (r – read), zmodyfikować (w – write) lub uruchomić (x – execute).

Zarządzanie uprawnieniami Uprawnieniami zarządza się poleceniem chmod. Istnieją dwie podstawowe metody ich zapisu: symboliczna i oktalna.

W metodzie symbolicznej używa się liter określających podmiot (u – użytkownik, g – grupa, o – inni, a – wszyscy) oraz znaków +, -, = do dodawania, odbierania lub ustawiania uprawnień, np.:

- chmod u+x skrypt.sh → dodanie prawa do wykonania dla właściciela.
- chmod go-w dokument.txt → odebranie prawa zapisu grupie i innym.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

W metodzie oktalnej uprawnienia reprezentowane są przez liczby w systemie ósemkowym, gdzie:

- 4 = read (r),
- 2 = write (w),
- 1 = execute (x). Sumując wartości, uzyskujemy pełne prawa:
- 7 = rwx,
- 6 = rw-,
- 5 = r-x,
- 4 = r--.

Dzięki temu zapis chmod 755 katalog oznacza: właściciel ma pełne prawa (7 = rwx), grupa i inni użytkownicy mają prawo odczytu i wykonania (5 = r-x).

Zmiana właściciela i grupy Uprawnienia ściśle wiążą się z właścicielem pliku i grupą. Do ich modyfikacji służy polecenie chown (zmiana właściciela) i chgrp (zmiana grupy). Na przykład:

- chown student dokument.txt → przypisanie pliku użytkownikowi „student”.
- chown student:admin dokument.txt → zmiana właściciela i grupy jednocześnie.

Dostęp do katalogów Warto zauważyć, że prawa do katalogów działają nieco inaczej niż do plików. Aby móc „wejść” do katalogu, potrzebne jest prawo x (execute), odczyt (r) pozwala na wyświetlenie listy plików, a zapis (w) umożliwia dodawanie i usuwanie plików w tym katalogu. Przykładowo katalog z uprawnieniami rwxr-x--- będzie dostępny tylko dla właściciela i grupy, ale inni użytkownicy nie będą mieli do niego żadnego dostępu.

Mechanizm linkowania, aliasy i zmienne środowiskowe

W systemach Linux linki to mechanizm pozwalający na tworzenie alternatywnych odwołań do plików i katalogów. Linki twarde (hard links) działają na poziomie systemu plików – są dodatkowymi nazwami wskazującymi na ten sam fizyczny plik (inode). Oznacza to, że plik może mieć wiele równorzędnych nazw, a dopóki istnieje choć jeden link, dane nie zostaną utracone. Linki symboliczne (symlinks) są

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

natomiast specjalnymi plikami przechowującymi ścieżkę do innego pliku lub katalogu. Działają podobnie jak skróty w systemie Windows – jeśli plik docelowy zostanie usunięty, symlink pozostanie, ale nie będzie prowadził do żadnych danych.

Powłoka systemowa w systemach Linux nie ogranicza się do przyjmowania poleceń użytkownika. Jest ona także odpowiedzialna za tworzenie i utrzymywanie środowiska, w którym uruchamiane są programy. Kluczową rolę pełnią tu zmienne środowiskowe – specjalne pary klucz–wartość, które przechowują informacje konfiguracyjne dotyczące systemu i powłoki. To właśnie dzięki nim programy wiedzą, gdzie szukać bibliotek, jakie jest aktualne ustawienie języka czy w jakich katalogach należy szukać wykonywalnych poleceń.

Jedną z najważniejszych zmiennych w systemie Linux jest PATH. Zawiera ona listę katalogów, w których powłoka szuka plików wykonywalnych. Gdy użytkownik wpisuje polecenie, takie jak ls, powłoka nie ma wbudowanej wiedzy o tym, gdzie znajduje się ten program – sprawdza kolejno katalogi wymienione w zmiennej PATH, aż znajdzie odpowiedni plik binarny. Zawartość tej zmiennej można podejrzeć poleceniem:

```
echo $PATH
```

Podobnie, polecenie printenv pozwala wyświetlić wszystkie aktualnie ustawione zmienne środowiskowe, takie jak informacje o bieżącym użytkowniku (USER), katalogu domowym (HOME), czy aktualnie używanej powłoce (SHELL).

Zmienne można także definiować samodzielnie. Na przykład:

```
export MYVAR="Hello World"
```

pozwala stworzyć zmienną o nazwie MYVAR, której wartość będzie dostępna w bieżącej sesji powłoki i wszystkich programach uruchomionych z tego środowiska. Mechanizm ten jest niezwykle przydatny do konfiguracji systemów, skryptów oraz do przekazywania parametrów programom bez konieczności ich wpisywania w linii poleceń.

Drugim elementem, który pozwala znacząco zwiększyć wygodę pracy w powłoce, są aliasy. Alias to krótka nazwa, pod którą kryje się dłuższe polecenie. Na przykład polecenie:

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

```
alias ll='ls -la'
```

powoduje, że wpisanie ll w terminalu wywoła szczegółowe listowanie zawartości katalogu (ls -la). Dzięki aliasom można tworzyć własne skróty do często używanych komend, a także unikać pomyłek – wielu administratorów ustawia alias rm='rm -i', aby polecenie usuwania plików zawsze pytało o potwierdzenie.

Zarówno aliasy, jak i zmienne środowiskowe można ustawiać na stałe, tak aby były dostępne w każdej nowej sesji terminala. W tym celu służy plik konfiguracyjny powłoki – .bashrc (dla powłoki Bash). Znajduje się on w katalogu domowym użytkownika (~/.bashrc) i jest wykonywany przy każdym uruchomieniu nowej sesji powłoki interaktywnej. Edytując ten plik, użytkownik może na przykład:

dodać nowe katalogi do zmiennej PATH,

zdefiniować własne aliasy,

zmienić wygląd promptu (PS1),

ustawić zmienne środowiskowe, które będą dostępne zawsze po zalogowaniu.

Przykład fragmentu pliku .bashrc:

```
# Dodanie katalogu ~/scripts do PATH
export PATH=$PATH:~/scripts


# Alias dla wygodniejszego listowania plików
alias ll='ls -la --color=auto'
```

Warto pamiętać, że plik .bashrc jest wykonywany przy każdym uruchomieniu nowej sesji interaktywnej powłoki, a wprowadzone w nim zmiany można natychmiast zastosować poleceniem:

```
source ~/.bashrc.
```

Dzięki tym mechanizmom użytkownik może dopasować środowisko pracy do własnych potrzeb – skrócić długie polecenia, automatycznie konfigurować programy

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

czy zyskać łatwiejszy dostęp do własnych narzędzi. To sprawia, że praca w powłoce staje się nie tylko efektywniejsza, ale też przyjemniejsza, a system lepiej odpowiada indywidualnym preferencjom użytkownika.

Systemy kontroli wersji

System kontroli wersji stanowi jedno z kluczowych narzędzi we współczesnej informatyce, zarówno w pracy indywidualnej, jak i zespołowej. Jego podstawowym zadaniem jest śledzenie zmian w plikach – w szczególności w kodzie źródłowym – oraz umożliwienie powrotu do dowolnego etapu rozwoju projektu. Dzięki temu możliwe jest nie tylko zachowanie historii rozwoju oprogramowania, ale także równoległa praca wielu osób nad tym samym projektem bez ryzyka utraty spójności czy nadpisania wyników pracy. W praktyce system kontroli wersji działa jak inteligentne archiwum: każda zmiana jest rejestrowana wraz z informacjami o autorze, czasie jej wprowadzenia oraz komentarzem opisującym jej cel. Pozwala to nie tylko na transparentność procesu, ale także na analizę i zrozumienie ewolucji projektu.

Najbardziej rozpowszechnionym systemem tego typu jest Git, stworzony w 2005 roku przez Linusa Torvaldsa na potrzeby rozwoju jądra Linux. Git należy do kategorii rozproszonych systemów kontroli wersji (DVCS – Distributed Version Control Systems), co oznacza, że każdy uczestnik projektu posiada na swoim komputerze pełną kopię repozytorium, obejmującą całą historię zmian. W odróżnieniu od systemów scentralizowanych (takich jak Subversion czy CVS), Git nie wymaga stałego połączenia z serwerem – użytkownik może swobodnie pracować offline, a następnie synchronizować swoje zmiany z innymi. Model rozproszony zwiększa nie tylko elastyczność, ale także bezpieczeństwo, ponieważ każda kopia repozytorium jest pełnoprawnym backupem.

Z perspektywy inżynierskiej Git wprowadza kluczowe pojęcia, które należy opanować: repozytorium, commit, branch, merge i remote. Repozytorium to centralne miejsce, w którym przechowywana jest historia projektu. Commit reprezentuje pojedynczą zmianę wprowadzoną do repozytorium, a gałęzie (branches) pozwalają

rozwijać różne funkcjonalności niezależnie od siebie, bez ingerencji w główną wersję projektu. Operacje takie jak merge umożliwiają późniejsze łączenie tych prac, a remotes określają zdalne repozytoria (np. na GitHubie czy GitLabie), które służą do współdzielenia kodu w zespole.

Systemy kontroli wersji – a w szczególności Git – stanowią dziś nieodzowny element profesjonalnego warsztatu informatyka. Są stosowane w niemal każdym projekcie programistycznym, od małych bibliotek open-source po ogromne systemy korporacyjne. Umiejętność ich wykorzystania jest nie tylko wymogiem technicznym, ale również elementem kultury pracy w zespołach projektowych, które opierają swoją działalność na przejrzystości, dokumentowaniu i możliwości odtwarzania zmian. Poznanie Gita otwiera drogę do praktyk stosowanych w realnym przemyśle IT.

Praktyczne przykłady pracy z systemem Git

Pierwszym krokiem w pracy z systemem kontroli wersji jest utworzenie repozytorium lokalnego. W nowym katalogu należy wydać polecenie git init, które zakłada repozytorium i tworzy ukryty katalog .git. Od tego momentu każda zmiana plików może być rejestrowana. Na przykład:

```
mkdir ~/GitLab2 && cd ~/GitLab2

git init

echo "Pierwsza notatka" > notes.txt

git add notes.txt

git commit -m "Initial commit – utworzono plik notes.txt"
```

Stan repozytorium można sprawdzić poleceniem git status, a historię wprowadzonych zmian – git log.

Kolejny etap to modyfikacja plików i obserwacja, jak Git śledzi kolejne wersje. Po edycji pliku notes.txt w edytorze i dopisaniu nowej treści, należy użyć poleceń git add i git commit. Różnice pomiędzy wersjami można podejrzeć za pomocą:

```
git diff          # zmiany w katalogu roboczym

git diff --staged # zmiany przygotowane do zatwierdzenia
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Istotną praktyką jest także ignorowanie plików, które nie powinny być wersjonowane. W tym celu tworzy się plik .gitignore, w którym umieszcza się odpowiednie wzorce. Przykładowo:

```
mkdir build
echo "plik tymczasowy" > build/temp.o
echo "build/" > .gitignore
git add .gitignore
git commit -m "Dodano .gitignore – pomijanie katalogu build"
```

Git pozwala również na integrację z systemem plików. Polecenie git mv umożliwia jednoczesną zmianę nazwy pliku w systemie i w repozytorium:

```
git mv notes.txt dokumentacja.txt
git commit -m "Zmieniono nazwę pliku notes.txt na dokumentacja.txt"
```

Analogicznie git rm służy do usuwania plików wraz z odnotowaniem tego w historii projektu.

Praca zdalna wymaga dodania repozytorium umieszczonego na serwerze, np. GitHubie lub GitLabie. Po dodaniu zdalnego repozytorium można wysłać do niego projekt:

```
git remote add origin https://github.com/uzytkownik/lab2repo.git
git push -u origin master
```

Od tej chwili projekt może być synchronizowany za pomocą git pull i git push.

Przydatne są także polecenia umożliwiające korektę i cofanie zmian. Niepoprawnie opisany commit można poprawić dzięki git commit --amend. Plik dodany do obszaru poczekalni można wycofać za pomocą git reset HEAD plik. Przywrócenie pliku do wersji z ostatniego commita realizuje się poleceniem git checkout -- plik.

Ważnym elementem pracy w Git są gałęzie. Nową gałąź można utworzyć i przełączyć się na nią poleceniami:

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

```
git branch test
```

```
git checkout test
```

Po wprowadzeniu zmian w gałęzi test można je scalić z główną wersją projektu poleceniem git merge. Całą historię projektu w formie graficznej prezentuje:

```
git log --oneline --graph
```

Ćwiczenia samodzielne

1. Sprawdź, jaka powłoka jest aktualnie używana w Twoim systemie. Wyświetl listę wszystkich powłok dostępnych w systemie.
2. Wyświetl strukturę katalogu głównego / i zidentyfikuj przeznaczenie katalogów: /bin, /etc, /home, /usr, /var.
3. Utwórz w katalogu domowym nowy katalog lab2, a w nim podkatalogi: src, logi, dane. Sprawdź hierarchię utworzonych katalogów poleceniem tree.
4. Utwórz w katalogu src trzy pliki: plik1.txt, plik2.txt, plik3.txt. Do każdego z nich dopisz kilka linijek tekstu, używając echo z przekierowaniem > i >>.
5. Skopiuj plik plik1.txt do katalogu logi i zmień jego nazwę na backup_plik1.txt. Następnie przenieś plik plik2.txt do katalogu dane. Usuń plik plik3.txt.
6. Utwórz katalog lab_uprawnienia i w nim plik dane.txt. Sprawdź jego uprawnienia (ls -l). Zidentyfikuj właściciela i grupę pliku.
7. Zmień uprawnienia pliku dane.txt, tak aby:

- właściciel mógł czytać i zapisywać,
- grupa mogła tylko czytać,
- inni nie mieli żadnych praw.

8. Zapisz polecenie w formie symbolicznej i numerycznej.
9. W katalogu src utwórz plik skrypt.sh, który wyświetla tekst „Witaj w Linuxie”. Nadaj mu uprawnienia do wykonania tylko dla właściciela. Sprawdź jego typ poleceniem file.
10. Uprawnienia SCRIPT: W katalogu src utwórz pusty plik deploy.sh. Zmień jego uprawnienia tak, aby właściciel miał pełne prawa (rwx), grupa tylko prawo do odczytu (r), a inni nie mieli żadnych praw. Podaj zarówno symboliczną, jak i numeryczną formę użytego polecenia chmod.
11. Utwórz dowiązanie twarde i symboliczne do pliku skrypt.sh. Usuń oryginał i sprawdź, które z dowiązań wciąż działa.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

12. Wyszukaj wszystkie pliki .txt w katalogu domowym, korzystając z find. Następnie odnajdź ścieżkę do programu bash poleceniami which i whereis.
13. Wyświetl listę wszystkich procesów w systemie i przekieruj wynik do pliku procesy.txt. Następnie przefiltruj wiersze zawierające nazwę bash i policz ich liczbę.
14. Utwórz plik notatki.txt, dodaj do niego kilka wierszy tekstu, a następnie wyświetl tylko jego pierwsze trzy i ostatnie dwie linie. Skorzystaj z poleceń head i tail.
15. Utwórz katalog projektGit, przejdź do niego i zainicjalizuj repozytorium. Dodaj plik readme.txt z opisem projektu i wykonaj pierwszy commit z komunikatem Initial commit.
16. Zmień zawartość pliku readme.txt, dopisz kilka linijek tekstu i sprawdź status repozytorium. Następnie dodaj plik do poczekalni i wykonaj commit z komunikatem Added new content.
17. Wprowadź kolejną zmianę w pliku readme.txt i sprawdź różnice między katalogiem roboczym a repozytorium. Następnie dodaj plik do poczekalni i porównaj wynik z git diff --staged.
18. Utwórz plik .gitignore, który pomija wszystkie pliki z rozszerzeniem .log oraz plik .env zawierający dane konfiguracyjne. Sprawdź, czy Git poprawnie ignoruje te pliki.
19. Wprowadź kilka kolejnych commitów do pliku readme.txt (np. dodaj sekcję Autor, Instrukcja, Licencja). Wyświetl historię projektu w formie skróconej i graficznej.
20. Cofnij plik readme.txt z obszaru poczekalni i przywróć jego zawartość do stanu z ostatniego commita. Sprawdź wynik działania w historii projektu.
21. Utwórz nową gałąź o nazwie feature, wprowadź w niej zmiany w pliku readme.txt, a następnie scal je z gałęzią główną. Zaobserwuj wynik łączenia w historii projektu.
22. Utwórz plik helloGit.sh zawierający prosty skrypt powłoki, nadaj mu prawa do wykonania i dodaj do repozytorium. Wprowadź modyfikację i wykonaj kolejny commit. Porównaj wersje skryptu w historii.
23. Wykonaj commit z błędnym opisem, a następnie popraw jego treść przy użyciu polecenia git commit --amend. Sprawdź w historii, czy opis został zmieniony.
24. Sklonuj publiczne repozytorium z GitHuba. Przejdź do katalogu projektu, wyświetl historię zmian i sprawdź autora najstarszego commita.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Pytania pomocnicze

1. Czym jest powłoka systemu operacyjnego i jaka jest jej podstawowa rola?
2. Wymień co najmniej trzy popularne powłoki w systemach Linux i podaj ich różnice.
3. Jakie polecenie służy do sprawdzenia aktualnego katalogu roboczego?
4. Jak wyświetlić zawartość katalogu wraz z ukrytymi plikami?
5. Jakie są podstawowe katalogi w Linuxie zgodnie z FHS i jakie pełnią funkcje (np. /etc, /home, /var)?
6. Jak utworzyć nowy plik z poziomu terminala?
7. Jakie polecenia służą do kopiowania, przenoszenia i usuwania plików?
8. Jak utworzyć katalog wraz z całą ścieżką podkatalogów jednym poleceniem?
9. W jaki sposób można podejrzeć zawartość dużego pliku tekstowego bez otwierania go w edytorze?
10. Czym różni się dowiązanie twarde od dowiązania symbolicznego?
11. Jakie polecenie umożliwia wyszukanie pliku na podstawie wzorca w całym systemie?
12. W jaki sposób sprawdzić, w jakim katalogu znajduje się plik wykonywalny programu (np. bash)?
13. Jak uruchomić prosty skrypt bash zapisany w pliku?
14. Jak system plików kontroluje dostęp do pojedynczych plików i katalogów?
15. Dlaczego znajomość podstawowych poleceń systemu plików jest kluczowa dla administratora systemu?

Podsumowanie

W ramach drugiego laboratorium studenci poznali praktyczne aspekty pracy z systemem operacyjnym Linux z poziomu powłoki Bash. Zajęcia miały na celu rozwinięcie umiejętności sprawnego poruszania się po strukturze systemu plików, zarządzania plikami i katalogami, a także zrozumienia mechanizmów uprawnień użytkowników oraz podstaw pracy z systemem kontroli wersji Git. Pierwsza część laboratorium była poświęcona nawigacji po systemie plików i wykorzystaniu podstawowych poleceń powłoki. Studenci nauczyli się poruszać między katalogami (cd, pwd), przeglądać ich zawartość (ls, tree), tworzyć i usuwać pliki oraz katalogi

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

(touch, mkdir, rm, cp, mv). Omówiono również hierarchię katalogów zgodną z Filesystem Hierarchy Standard (FHS), wskazując znaczenie najważniejszych lokalizacji takich jak /etc, /bin, /home, /usr czy /var. Wprowadzono pojęcia ścieżki bezwzględnej i względnej, przekierowań (>, >>, <, |) oraz masek i wzorców (*, ?) używanych przy wyszukiwaniu i przetwarzaniu wielu plików jednocześnie. Druga część ćwiczeń dotyczyła systemu uprawnień w Linuxie. Studenci poznali model trójdzielny, w którym każdy plik posiada właściciela (owner), grupę (group) oraz zestaw uprawnień dla pozostałych użytkowników (others). Omówiono trzy typy uprawnień – odczyt (r), zapis (w) i wykonanie (x) – oraz ich różne znaczenia dla plików i katalogów. W praktyce uczestnicy modyfikowali prawa dostępu poleceniem chmod, zarówno w formie symbolicznej (u+rwx, g+rx, o-r) jak i numerycznej (chmod 750 plik.sh). Zademonstrowano również użycie poleceń chown i chgrp do zmiany właściciela i grupy pliku, a także konsekwencje błędnie ustawionych uprawnień dla bezpieczeństwa systemu. Trzecia część laboratorium wprowadziła system kontroli wersji Git jako narzędzie do zarządzania zmianami w projektach programistycznych. Studenci utworzyli lokalne repozytorium (git init), dodawali i zatwierdzali zmiany (git add, git commit), śledzili historię (git log), porównywali wersje plików (git diff) oraz poznali znaczenie obszaru poczekalni (staging area). Omówiono także rolę pliku .gitignore w wykluczaniu plików konfiguracyjnych i tymczasowych oraz podstawowe operacje na gałęziach (git branch, git checkout, git merge). Ćwiczenia te pozwoliły uczestnikom zrozumieć ideę wersjonowania kodu i pracy zespołowej w środowisku projektowym.

Literatura

1. Chmod linux permision, link: https://linuxize.com/post/chmod-command-in-linux/ (data ostatniego dostępu 25.09.25).
2. File Systems in Operating System, link: https://www.geeksforgeeks.org/operating-systems/file-systems-in-operating-system/ (data ostatniego dostępu 25.09.25).
3. Git toutorial: https://www.w3schools.com/git/default.asp (data ostatniego dostępu 25.09.25).

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

4. Using Git on CommandLine, https://www.geeksforgeeks.org/git/using-git-on-commandline/ (data ostatniego dostępu 25.09.25).

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Materiały do przedmiotu:

Podstawy Systemów Operacyjnych

Laboratorium nr 3:

Przetwarzanie potokowe i zastosowanie filtrów w systemach operacyjnych, mechanizmy wejścia i wyjścia oraz manipulacja strumieniami danych

Autor:

Albert Rachwał

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Na trzecich zajęciach uwaga zostaje skierowana na mechanizmy, które sprawiają, że powłoka jest tak elastycznym narzędziem do pracy z danymi. System operacyjny traktuje dane w postaci strumieni, a powłoka daje możliwość ich kierowania, łączenia i przetwarzania. Studenci uczą się, czym są standardowe strumienie wejścia, wyjścia i błędów, a także jak stosować przekierowania oraz potoki do budowania złożonych ciągów poleceń. Omawiane są klasyczne filtry, takie jak narzędzia do wyszukiwania, wycinania fragmentów tekstu, sortowania czy liczenia słów, a także bardziej zaawansowane metody pracy na plikach konfiguracyjnych i logach systemowych. Zajęcia wprowadzają również temat sygnałów procesów i ich wpływu na działanie aplikacji. W efekcie studenci uczą się tworzyć własne pipeline'y, które w praktyce pozwalają szybko analizować dane, filtrować logi czy generować raporty w czasie rzeczywistym.

Cel ćwiczenia/laboratorium

1. Zrozumienie pojęcia strumieni danych w systemie Linux (stdin, stdout, stderr).
2. Opanowanie podstaw przekierowań wejścia i wyjścia w powłoce.
3. Poznanie działania potoków i ich zastosowania w łączeniu wielu poleceń.
4. Nabycie umiejętności filtrowania danych za pomocą polecenia grep.
5. Umiejętność wyszukiwania wzorców z użyciem egrep i wyrażeń regularnych.
6. Poznanie podstawowych możliwości programu awk do obróbki tekstu w kolumnach.
7. Opanowanie poleceń edycyjnych sed i ich zastosowania w prostych transformacjach tekstu.
8. Wykorzystywanie sort i uniq do porządkowania i analizowania danych.
9. Umiejętność zliczania słów, linii i znaków za pomocą wc.
10. Wycinanie fragmentów tekstu przy użyciu cut.
11. Analiza dużych plików tekstowych za pomocą head i tail.
12. Składanie kilku filtrów w jednym potoku do realizacji złożonego zadania (np. analiza logów systemowych).
13. Wprowadzenie do pracy z danymi w formacie JSON za pomocą narzędzia jq.
14. Wykorzystanie narzędzia csvkit do analizy plików CSV w terminalu.
15. Opanowanie interaktywnego wyszukiwania danych w strumieniu przy użyciu fzf.

Instrukcja laboratoryjna/ćwiczeniowa

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Strumienie

W systemach Unix i Linux (a także w większości środowisk POSIX) strumienie standardowe (standard streams) stanowią zasadniczą abstrakcję wejścia i wyjścia, dzięki której programy mogą komunikować się ze światem zewnętrznym w sposób ujednolicony. Gdy program się uruchamia, domyślnie są dla niego otwarte trzy standardowe kanały: standard input (stdin), standard output (stdout) oraz standard error (stderr). Strumień stdin służy do odbioru danych wejściowych (np. od użytkownika lub z innego programu), stdout do wypisywania wynikowych danych, a stderr do przekazywania komunikatów diagnostycznych i błędów. W językach C i C++ są one dostępne jako stdin, stdout, stderr (lub odpowiedniki w iostream: std::cin, std::cout, std::cerr).

Strumienie standardowe mają kluczowe znaczenie, ponieważ umożliwiają redirekcję i łączenie poleceń bez potrzeby tworzenia tymczasowych plików, powłoka może kierować strumień wyjścia jednego programu do wejścia kolejnego (przez potok |), albo zapisywać wyjście do pliku (> lub >>), albo oddzielnie przekierowywać błędy (2>) albo łączyć strumienie (2>&1).

Dzięki temu narzędzia typu grep, sed, awk i inne mogą działać jako filtry, które przyjmują dane z stdin, przetwarzają je i wypisują wynik na stdout, tworząc elastyczne łańcuchy przetwarzania.

W praktyce oznacza to, że jeśli uruchomisz komendę:

```
ls -l | grep “txt”
```

to ls -l wysyła listę plików na swój stdout, który jest automatycznie przekazany jako stdin dla grep. grep filtruje linie zawierające „txt” i wypisuje je na swój stdout, a wynik trafia na ekran (lub dalej w potok). W tym procesie stderr obu programów (błędy) zostaje domyślnie skierowany do terminala, chyba że jawnie przekierujemy go inaczej.

Warto też zwrócić uwagę na zachowanie buforowania, stdout i stderr mają domyślnie różne strategie buforowania (stdout zwykle buforowany liniowo, a stderr niebuforowany), co w niektórych sytuacjach ma wpływ na kolejność wyświetlania komunikatów, zwłaszcza w potokach lub przy przekierowaniach.

Potoki

W systemach Unix i Linux potok (ang. pipeline, potocznie nazywany też „rurą”) to konstrukcja umożliwiająca przekierowanie strumienia danych pomiędzy dwoma (lub

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

więcej) procesami, w taki sposób, że wyjście standardowe (stdout) jednego programu staje się wejściem standardowym (stdin) następnego programu. Dzięki temu można łączyć efekty wielu prostych narzędzi w jeden kontinuum przetwarzania, bez konieczności tworzenia plików pośrednich. To podejście pozwala budować złożone operacje tekstowe jako ciąg modularnych kroków, gdzie każdy program wykonuje jedną specjalistyczną funkcję: filtrowanie, sortowanie, agregację, transformację i przekazuje wynik dalej.

W momencie uruchomienia linii z potokiem, powłoka (shell) tworzy w systemie anonimowe kanały (pipe) i odpowiednio łączy deskryptory plików procesów: jeden proces pisze do końca zapisu potoku (write-end), drugi odczytuje z końca odczytu potoku (read-end). Mechanizm ten jest realizowany poprzez wywołanie systemowe pipe(), które zwraca dwie połączone końcówki potoku (jeden do zapisu, drugi do odczytu). Następnie powłoka zwykle używa fork() by uruchomić procesy potomne i przekierowuje ich standardowe wejścia / wyjścia na te końcówki potoku. W efekcie procesy te działają równolegle , gdy program po lewej produkuje dane, trafiają one do bufora potoku, a proces po prawej je konsumuje, gdy jest gotowy. System operacyjny dba o buforowanie danych między procesami, aby synchronizacja przebiegała płynnie,jeśli producent (proces piszący) wyprodukuje dane szybciej niż konsument je odczyta, zapisują się one w buforze, a gdy bufor się zapełni, piszący zostaje zablokowany, dopóki konsument nie zrobi miejsca. Dzięki temu użytkownik może traktować potok jak jednolity kanał danych między programami, niezależnie od liczby procesów pośrednich.

Typowa składnia potoku w shellu to:

```
polecenie1 | polecenie2
```

gdzie znak | oznacza „kierowanie wyjścia pierwszego procesu do wejścia drugiego”. Oznacza to, że zamiast wypisywać wynik na ekran lub do pliku, pierwszy program wysyła swoje dane wprost do kolejnego programu. Możliwe jest także łączenie wielu poleceń w łańcuch:

```
polecenie1 | polecenie2 | polecenie3 | ...
```

W takim ciągu każdy kolejny program odbiera dane od poprzedniego, przetwarza je i przekazuje dalej. Przykład:

```
cat /var/log/syslog | grep error | sort | uniq -c
```

Tutaj kolejno:

1. cat wypisuje całą zawartość pliku logów,

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

2. grep error filtruje linie zawierające „error”,
3. sort porządkuje te linie,
4. uniq -c liczy unikalne wystąpienia każdej linii.

W efekcie w jednej komendzie otrzymujemy zagregowaną statystykę błędów.

Potoki można także łączyć z innymi mechanizmami: można przekierowywać wyjście potoku do pliku, jak również używać narzędzia tee, które kopiuje strumień danych jednocześnie do pliku i dalej do kolejnego procesu:

```
polecenie1 | tee kopia.txt | polecenie2
```

W tym wypadku wynik programu polecenie1 zostaje zapisany do kopia.txt i przekazany dalej do polecenie2. Można też przekierować błąd (stderr) do /dev/null tzw. „czarna dziura” by go ukryć:

```
polecenie 2> /dev/null
```

Zamiast jednego wielkiego programu, lepiej jest składać wyjściową architekturę przetwarzania danych z prostych filtrów, połączonych potokami. Dzięki temu kod staje się przejrzysty, elastyczny i łatwy do komponowania, można łatwo podmieniać kolejne etapy lub dodawać nowe filtry bez modyfikacji całego systemu.

Polecenie grep

Polecenie grep (ang. global regular expression print) jest klasycznym narzędziem systemów operacyjnych z rodziny Unix/Linux, służącym do wyszukiwania w strumieniach danych lub plikach linii tekstu zawierających określony wzorzec. Nazwa programu wywodzi się ze składni edytora ed, w którym polecenie g/re/p oznaczało: „dla wszystkich linii pasujących do wzorca re – wydrukuj je”. Mechanizm działania grep opiera się na sekwencyjnym przetwarzaniu danych: narzędzie odczytuje kolejne linie wejściowe, sprawdza ich zgodność z podanym wzorcem i jeśli występuje dopasowanie wypisuje linię lub jej fragment na standardowe wyjście. W przypadku przetwarzania wielu plików grep domyślnie dołącza do wyniku nazwę pliku oraz numer linii, co ułatwia identyfikację kontekstu wystąpienia.

Zasadniczą cechą grep jest wykorzystanie wyrażeń regularnych, które pozwalają definiować wzorce o różnym stopniu złożoności, od prostych słów po struktury obejmujące grupowania, alternatywy i kwantyfikatory. W zależności od implementacji dostępne są różne warianty składni: BRE (Basic Regular Expressions), ERE (Extended Regular Expressions, aktywowane przełącznikiem -E) oraz w przypadku

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

GNU grep również PCRE (Perl Compatible Regular Expressions, opcja -P). Dzięki temu narzędzie umożliwia dopasowania obejmujące zarówno proste wyszukiwanie ciągów znaków, jak i zaawansowane reguły tekstowe.

grep oferuje liczne opcje rozszerzające jego funkcjonalność. Najczęściej stosowane to:

- -i – ignorowanie wielkości liter (tryb case-insensitive),
- -v – odwrócenie dopasowania, czyli wyświetlenie linii, które nie zawierają wzorca,
- -n – dodanie numeru linii przy każdym dopasowaniu,
- -c – zliczenie liczby dopasowań zamiast ich wyświetlania,
- -l – wypisanie jedynie nazw plików, w których występuje dopasowanie,
- -r – rekurencyjne przeszukiwanie katalogów.

Wariant grep -F (dawniej fgrep) traktuje wzorzec dosłownie, bez interpretacji metaznaków, co przyspiesza wyszukiwanie prostych fraz.

Dzięki przetwarzaniu danych linia po linii grep charakteryzuje się dużą wydajnością i niskim zapotrzebowaniem na pamięć operacyjną. Nie wymaga wczytywania całego pliku do pamięci, co sprawia, że idealnie nadaje się do pracy z dużymi zbiorami danych, takimi jak logi systemowe, raporty czy pliki konfiguracyjne. W nowoczesnych implementacjach (np. GNU grep) zastosowano dodatkowe optymalizacje, w tym algorytmy wyszukiwania oparte na metodzie Boyera–Moore'a, buforowanie wejścia i wyjścia oraz opcję --color, pozwalającą na podświetlanie dopasowań w wynikach.

W kontekście praktycznym grep jest jednym z podstawowych narzędzi analizy i diagnostyki systemów Linux. Stosuje się go m.in. do wyszukiwania błędów w logach (grep ERROR /var/log/syslog), odnajdywania fragmentów kodu źródłowego (grep "main" *.c) lub filtrowania danych konfiguracyjnych. W połączeniu z innymi narzędziami powłoki, takimi jak sed, awk, cut, sort, uniq czy wc tworzy zaawansowane potoki przetwarzania tekstu, umożliwiające tworzenie złożonych operacji analitycznych bez konieczności użycia języków programowania wysokiego poziomu. Przykładowo, polecenie grep "root" /etc/passwd | wc -l pozwala zliczyć liczbę wystąpień danego użytkownika, a grep -E "[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+" access.log umożliwia identyfikację adresów IP w pliku dziennika.

Zasada działania i struktura programu AWK

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

AWK (od nazwisk autorów: Alfred Aho, Peter Weinberger i Brian Kernighan) to język skryptowy zaprojektowany do przetwarzania tekstu, analizowania danych i raportowania w środowiskach Unix/Linux. Już od lat 70. stanowi jedno z najważniejszych narzędzi w arsenale administratorów i deweloperów do pracy z plikami tekstowymi, plikami CSV, logami, tabelami danych, pozwalając na łączenie prostych filtrów, agregacji i operacji kolumnowych w jeden spójny proces. AWK można traktować jako „inteligentny filtr", który czyta dane linia po linii, dzieli je na pola, dopasowuje wzorce i wykonuje akcje w oparciu o te wzorce.

AWK działa w trybie przetwarzania rekord po rekordzie (domyślnie rekord to jedna linia tekstu). Każdy rekord zostaje podzielony na pola (domyślnie według białych znaków, spacji i tabulacji), które dostępne są jako zmienne $1, $2, …, $NF (gdzie NF oznacza liczbę pól w danym rekordzie). Cała linia jest dostępna jako $0. Można modyfikować sposób dzielenia wejścia, zmieniając zmienną FS (separator pól) lub RS (separator rekordów).

Program AWK składa się z sekwencji par wzorzec → akcja (pattern → action). Gdy linia wejścia pasuje do wzorca (lub gdy warunek logiczny jest spełniony), wykonywana jest określona akcja. Jeśli wzorzec zostanie pominięty, a akcja nie zostanie określona, AWK domyślnie wypisuje bieżącą linię (równoważne print $0). Program może zawierać też specjalne bloki BEGIN { … }, które wykonują się przed przetwarzaniem rekordów, oraz END { … }, które wykonują się po przetworzeniu wszystkich linii. AWK oferuje zmienne, operatory arytmetyczne i tekstowe, instrukcje warunkowe (if ... else), pętle (for, while), funkcje użytkownika, tablice asocjacyjne (klucz → wartość) i wyrażenia regularne. Dzięki temu można w jednym skrypcie AWK zrealizować skomplikowane transformacje danych, sumy, średnie, grupowania, filtrowanie warunkowe, agregacje i wiele innych operacji.

AWK łączy elastyczność narzędzi tekstowych (grep, sed) z możliwościami języka programistycznego. Jego główną zaletą jest to, że pozwala w bardzo skondensowany sposób zapisać operacje na kolumnach i rekordach: np. wyciągnięcie drugiej kolumny tylko dla rekordów spełniających warunek, obliczenie sumy lub średniej wartości kolumny, zgrupowanie według klucza, liczenie częstości występowania i generowanie zestawień. AWK często stosuje się w potokach (pipe), jako element przetwarzania danych pomiędzy komendami shellowymi.

Ponadto AWK potrafi czytać wejście sekwencyjnie, nie ładuje całego pliku do pamięci, co czyni go użytecznym do pracy na dużych plikach tekstowych. Istniejąc iersje awk (jak gawk, mawk, nawk) oferujące dodatkowe funkcje, na przykład w gawk można korzystać z rozszerzeń, bibliotek, funkcji sieciowych lub obsługi CSV.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Poniżej kilka scenariuszy użycia AWK:

- W prostej wersji: awk '{ print $2, $4 }' plik.csv - wyświetlenie 2. i 4. pola każdej linii CSV (domyślnym separatorem pola jest spacja / tabulacja).
- Z warunkiem: awk '$4 > 100 { print $2, $4 }' plik.csv - dla linii, gdzie czwarte pole jest większe niż 100, wypisz 2. i 4. pole.
- Agregacja i suma: awk '{ sum += $4 * $5 } END { print sum }' plik.csv - sumuje iloczyn pola 4 i pola 5 dla wszystkich linii, a wynik wypisuje na końcu.
- Grupowanie według kategorii: awk -F',' '{ cat = $3; value[cat] += $4 } END { for (c in value) print c, value[c] }' plik.csv - rozdziela pola sep. przecinkiem (CSV), a następnie sumuje wartości w polu 4 według klucza kategorii w polu 3.
- Bloki BEGIN i END: awk 'BEGIN { print "Start"; } { print $0 } END { print "Koniec"; }' plik.txt - najpierw wypisze „Start”, potem każdą linię, a na końcu „Koniec”.

W awk, jeśli chcemy wyświetlić liczbę z określoną liczbą miejsc dziesiętnych, możemy skorzystać z funkcji printf lub sprintf z odpowiednim formatem. Na przykład printf "%.2f\n", x spowoduje, że zmienna x zostanie wypisana z dokładnością do dwóch miejsc po przecinku. Jeśli potrzebne jest klasyczne zaokrąglanie (np. .5 w górę), można wewnętrznie dodać 0.5 przed konwersją lub napisać własną funkcję „round” (np. function round(x) { return int(x + 0.5) }). Dzięki opisanym powyżej możliwościom awk często stanowi środkowy lub końcowy etap potoków, łącząc filtrowanie, transformację i agregację w jednym skrypcie, co czyni go bardzo wygodnym narzędziem na etapie przygotowywania raportów lub obróbki wejściowych danych.

Polecenie sed

sed, czyli stream editor, to narzędzie tekstowe działające w trybie non-interaktywnym, przeznaczone do przetwarzania i transformowania danych wejściowych linia po linii. Zamiast otwierać plik w interaktywnym edytorze, sed bierze tekst jako strumień (stdin lub plik), wykonuje określone operacje (np. substytucje, usuwanie, wstawianie) zgodnie ze skryptem poleceń, i wypisuje wynik na wyjście standardowe Na poziomie implementacyjnym sed korzysta z przestrzeni nazw pattern space, gdzie trzyma bieżącą linię, oraz opcjonalnie hold space, w której można przechować fragment tekstu do późniejszego użycia. Dla każdej linii wejściowej, sed wykonuje cykl: czyta linię, aplikuje sekwencję poleceń skryptu, wypisuje zmodyfikowaną linię (o ile nie zostanie usunięta), i przechodzi do następnej linii. Najczęściej używanym poleceniem sed jest s (substitute), które pozwala zastąpić fragment tekstu zgodnie z wyrażeniem

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

regularnym: składnia s/wzorzec/nowy/ (lub z flagą g dla wszystkich wystąpień w linii). Można ograniczyć substytucję do określonego zakresu linii lub warunków, np. 3,5 s/foo/bar/ (tylko pomiędzy liniami 3 i 5) lub /regex/ s/old/new/ (jeśli linia pasuje do regex).

sed oferuje wiele innych poleceń poza substytucją: d (delete) usuwa linię, i\ lub a\ pozwalają wstawiać tekst przed lub po linii, q kończy działanie narzędzia wcześnie, N scala kolejne linie w pattern space, a polecenia warunkowe i etykietowania (b, :label) umożliwiają złożone konstrukcje logiczne.

sed ma opcję -i, która pozwala edytować plik "in-place", czyli nadpisać go bezpośrednio (z użyciem pliku tymczasowego pod spodem). Dzięki temu można stosować sed jako automatyczny edytor pliku bez potrzeby tworzenia kopii ręcznie.

Największym atutem sed jest jego szybkość i efektywność pamięciowa, ponieważ przetwarza dane w sposób liniowy (bez konieczności ładowania całego pliku do pamięci), bardzo dobrze nadaje się do pracy na dużych plikach. Dla wielu prostych transformacji tekstu (zastępowanie, usuwanie linii, modyfikacje wzorca) sed jest często pierwszym narzędziem do zastosowania.

Ćwiczenia samodzielne

Listing do zadań 1-6, w których sugerowane jest użycie polecenia awk.
Ponieważ listing zawiera nagłówek, a awk przetwarza tekst linia po linii, za pomocą polecenia NR==1{next} możesz wskazać aby awk pominął pierwszy wiersz:

```
ID,Produkt,Kategoria,Cena,Sztuk
1,Laptop,Elektronika,3500,2
2,Mysz,Elektronika,120,5
3,Monitor,Elektronika,890,3
4,Kawa,Spożywcze,45,10
5,Cukier,Spożywcze,8,25
6,Książka,Edukacja,55,4
7,Klawiatura,Elektronika,250,3
8,Herbata,Spożywcze,30,6
9,Tablet,Elektronika,1500,1
10,Długopis,Edukacja,4,50
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

1. Podaj nazwy produktów oraz ich ceny, możesz do tego celu zastosować polecenie awk.
2. Wyświetl tylko te produkty, których cena jest wyższa niż 100zł.
3. Policz ile produktów należy do poszczególnych kategorii.
4. Policz całkowitą wartość wszystkich produktów wyrażoną wzorem cena x liczba sztuk.
5. Wskaż najdroższy prodkt i wpisz jego nazwe.
6. Wyświetl nazwę kategorii i średnią cenę produktów w tej kategorii, zaokrąglając wynik do dwóch miejsc po przecinku.

Listing do zadań 7 - 12, w których sugerowane jest użycie polecenia sed:

```
2025-10-06 08:13:45 INFO  User 'anna' logged in
2025-10-06 08:14:02 ERROR Failed to open file /etc/passwd
2025-10-06 08:15:17 INFO  User 'antoni' logged out
2025-10-06 08:16:01 WARN  Disk usage above 90%
2025-10-06 08:17:55 ERROR Network timeout while connecting to
10.0.0.5
2025-10-06 08:20:11 INFO  User 'jan' logged in
```

7. Usuń wszystkie linie zawierające słowo INFO i zapisz wynik do pliku bledy.txt, można do tego celu użyć polecenie sed.
8. Używając polecenia sed zaanonimizuj adresy IP zastępując je wybraną frazą lub ciągiem znaków.
9. Zmień w każdej lini tekstu myślniki w dacie na ukośniki.
10. Dodaj do pliku z logami puste linie oraz nadmiarowe separatory, a następnie napisz polecenie które je usunie.
11. Wyodrębnij wpisy zawierające fraze „ERROR” i zapisz do pliku error.txt.
12. W pliku logów usuń komentarze zaczynające się od #, ale tylko jeśli nie są na początku pliku (czyli usuń komentarze w środku, zostaw nagłówek).
13. Wykorzystaj przekierowania błędów. Uruchom polecenie ls /root przekierowując standardowe wyjście do wynik.txt, a błędy do blad.txt. Sprawdź zawartość obu plików.
14. Użyj polecenia tee, aby jednocześnie wyświetlić wynik ls /bin i zapisać go do pliku bin.txt.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

15. Wykorzystaj grep, aby wyszukać wszystkie wystąpienia słowa „root” w pliku /etc/passwd. Następnie policz, ile linii zawiera to słowo (wc -l).
16. Wyświetl tylko nazwy użytkowników systemu z /etc/passwd (pierwsze pole, separator :) używając cut. Następnie posortuj je i usuń duplikaty (sort | uniq).
17. Użyj grep i egrep, aby porównać działanie obu poleceń przy wyszukiwaniu linii zawierających adres IP w pliku /var/log/syslog. Spróbuj użyć wyrażenia regularnego [0-9]+\.[0-9]+\.[0-9]+\.[0-9]+.
18. Połącz następujące polecenia w potok. Wyświetl procesy (ps aux), wybierz tylko ich nazwy (awk '{print $11}'), policz ile razy występuje każda (sort | uniq -c), a wynik zapisz do pliku procesy.txt.
19. Używając awk -F:, wyświetl drugie pole (UID) z pliku /etc/passwd, oblicz jego sumę i średnią wartość.
20. Z pliku /etc/passwd wyciągnij nazwy użytkowników do users.txt, a ich powłoki (/bin/bash, /usr/sbin/nologin) do shells.txt. Następnie połącz je w jeden plik konto.txt.
21. Użyj tr, aby w pliku dane.txt zamienić wszystkie małe litery na wielkie, a następnie usunąć cyfry.
22. Znajdź wszystkie pliki w katalogu /etc, które zawierają w nazwie słowo „conf”, używając find oraz locate. Przekieruj wyniki do pliku konfiguracje.txt.
23. Użyj find i xargs, aby policzyć, ile linii kodu zawierają wszystkie pliki .sh we wskazanym katalogu (find ~ -name "*.sh" | xargs wc -l).

    Listing do zadań 24 i 25:

```
imię,nazwisko,ocena
Anna,Nowak,4.5
Piotr,Kowalski,3.0
Jan,Kamiński,5.0
Maria,Zielińska,4.0
Tomasz,Wiśniewski,3.5
```

24. Wyświetl tylko imiona i nazwiska (csvcut -c imię,nazwisko), a następnie sprawdź statystyki (csvstat).
25. Wyświetl zawartość pliku CSV studenci.csv w formie czytelnej tabeli za pomocą column -t -s,.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

26. Użyj fzf, aby interaktywnie przeszukać plik /etc/passwd. Po wybraniu linii skopiuj ją do schowka (xclip lub pbcopy – jeśli dostępne).
27. Użyj pv do wizualizacji postępu kopiowania pliku (rozmiar pliku powinien być na tyle duży by wizualizacja była możliwa).
28. Wykorzystaj parallel do jednoczesnego przetworzenia kilku plików .txt przy pomocy grep.
29. Pobierz przykładowy plik JSON z https://jsonplaceholder.typicode.com/posts, zapisz jako posts.json, i wyświetl tylko tytuły wpisów (jq .[].title).
30. Za pomocą jq -r wyświetl wartości z klucza .userId z pliku JSON i policz, ile razy każdy identyfikator występuje (sort | uniq -c).
31. Wyświetl listę wszystkich użytkowników z pliku /etc/passwd, a następnie w potoku policz, ilu ich jest. Zapisz wynik do pliku liczba_uzytkownikow.txt.
32. Wypisz tylko procesy użytkownika root, następnie posortuj wynik alfabetycznie i wyświetl pierwsze 10 rekordów. Połącz wszystkie polecenia w jeden potok.
33. Policz, ile różnych powłok (np. /bin/bash, /usr/sbin/nologin) występuje w pliku /etc/passwd. Wynik przedstaw w formie liczby wystąpień każdej powłoki.
34. Wyświetl rozmiary wszystkich plików w katalogu /var/log, posortuj malejąco i pokaż 10 największych. Zapisz wynik do pliku top_logs.txt.
35. Policz, ile razy w logach systemowych występują różne typy komunikatów (INFO, WARN, ERROR). Do analizy wykorzystaj plik logów systemowych — w większości dystrybucji Linuxa znajduje się on w katalogu:
    - /var/log/syslog – (Debian, Ubuntu, Mint)
    - /var/log/messages – (Fedora, CentOS, openSUSE)

    Możesz skopiować fragment tego pliku do katalogu domowego i pracować na kopii, np. poleceniem:

    ```
    cp /var/log/syslog ~/logs.txt
    ```
36. Znajdź wszystkie pliki .conf w katalogu /etc, pogrupuj je według podkatalogów i policz, ile występuje w każdym z nich.
37. Wyświetl 10 katalogów w katalogu domowym użytkownika, które zajmują najwięcej miejsca na dysku. Wynik posortuj malejąco według rozmiaru.

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

38. Stwórz potok, który:
    - znajdzie wszystkie pliki .log w katalogu /var/log,
    - wybierze te większe niż 1 MB,
    - wyświetli ich rozmiary,
    - posortuje malejąco i zapisze wynik do pliku duze_logi.txt.
39. Znajdź w pliku /etc/services wszystkie unikalne numery portów i policz, ile ich jest. Wyświetl tylko porty o numerach większych niż 1000.
40. Wyświetl 5 procesów o najwyższym zużyciu pamięci w systemie. Użyj potoku do posortowania procesów według użycia pamięci i wyświetlenia tylko nazwy procesu oraz procentowego zużycia pamięci.

Pytania pomocnicze

1. Czym są strumienie danych stdin, stdout i stderr?
2. Jakie polecenie w powłoce służy do przekierowania standardowego wyjścia do pliku?
3. Jaka jest różnica między > a >> w przekierowaniach?
4. Do czego służy 2> w kontekście przekierowań?
5. Na czym polega idea potoków w systemie Linux?
6. Jak działa operator „|” i dlaczego jest tak użyteczny?
7. W jaki sposób grep różni się od egrep?
8. Jakie przykłady zastosowania wyrażeń regularnych można wykorzystać w egrep?
9. Do czego służy narzędzie awk i w jakich sytuacjach jest najczęściej używane?
10. Jakie możliwości edycji tekstu oferuje sed? Podaj przykład prostego zastosowania.
11. Jak przy pomocy sort i uniq policzyć unikalne adresy IP w logach serwera?
12. Jak polecenie wc -l różni się od wc -w?
13. Jakie polecenia wykorzystasz, aby zobaczyć tylko pierwsze 20 linii pliku?

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

14. Jak narzędzie jq pomaga w pracy z danymi JSON?

15. Czym różni się podejście klasycznych filtrów (grep, awk, sed) od nowoczesnych narzędzi jak csvkit czy fzf?

Podsumowanie

Podczas trzeciego laboratorium studenci poznali kluczowe mechanizmy przetwarzania danych tekstowych w systemach Unix/Linux, oparte na koncepcji strumieni, potoków i filtrów. Zajęcia rozpoczęły się od omówienia idei trzech standardowych strumieni: stdin, stdout i stderr, które stanowią podstawowy model komunikacji procesów z otoczeniem. Studenci zrozumieli znaczenie redirekcji (>, >>, <, 2>, 2>&1) oraz sposób, w jaki powłoka umożliwia przekierowywanie i łączenie danych pomiędzy procesami bez użycia plików pośrednich. Dzięki temu mogli zaobserwować, że każde narzędzie systemowe, które czyta dane ze strumienia wejściowego i zapisuje wynik na wyjście standardowe, może stać się elementem większego procesu przetwarzania informacji.

Druga część zajęć poświęcona była potokom (pipeline), które umożliwiają sekwencyjne przetwarzanie danych przez wiele programów. Studenci poznali zasadę ich działania na poziomie systemowym, powłoka tworzy kanały komunikacyjne, a procesy potomne wymieniają dane w sposób synchroniczny za pomocą buforów. Praktyczne przykłady pozwoliły zrozumieć, że złożone operacje tekstowe można budować jako modularne łańcuchy prostych narzędzi, z których każde pełni ściśle określoną funkcję. Wprowadzenie narzędzia tee oraz przekierowań błędów (2>, /dev/null) umożliwiło pełniejsze zrozumienie przepływu danych w powłoce oraz roli buforowania strumieni.

W dalszej części laboratorium uczestnicy zgłębili praktyczne zastosowania narzędzia grep, służącego do filtrowania i wyszukiwania danych w strumieniach tekstowych przy użyciu wyrażeń regularnych. Poznali zarówno jego podstawową składnię, jak i najczęściej używane opcje (-i, -v, -n, -c, -r, -E, -F), a także zastosowania w analizie logów, plików konfiguracyjnych i kodu źródłowego. Ćwiczenia praktyczne ukazały, że grep stanowi podstawowy filtr w potokach, umożliwiający selekcję danych według zadanych kryteriów logicznych.

Kolejnym etapem było zapoznanie z językiem AWK, który rozszerza możliwości grep i sed o logikę warunkową, operacje kolumnowe oraz funkcje obliczeniowe. Studenci przeanalizowali strukturę programu AWK w postaci par „wzorzec → akcja” oraz bloki

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

BEGIN i END, poznając zastosowania w analizie tabelarycznych danych tekstowych, obliczeniach i raportowaniu. Z kolei sed został przedstawiony jako edytor strumieniowy służący do automatycznych modyfikacji danych tekstowych w trybie liniowym, w tym zastępowania, usuwania i wstawiania fragmentów tekstu bez potrzeby interakcji z użytkownikiem.

Zajęcia unaoczniły studentom, że potoki i filtry tekstowe stanowią nieodłączny element przetwarzania danych w systemach Linux, gdzie każda prosta komenda, połączona z innymi, tworzy elastyczny i skalowalny system przetwarzania danych. Poznane narzędzia (grep, awk, sed) ukazały, jak w środowisku powłoki można realizować zaawansowane analizy tekstu i transformacje danych bez użycia zewnętrznych języków programowania. Dzięki temu laboratorium uczestnicy zdobyli praktyczne umiejętności niezbędne do pracy administracyjnej, analitycznej i automatyzacji procesów w systemach operacyjnych klasy Unix/Linux.

Literatura

1. Working with data streams on the Linux command line, link:

   https://opensource.com/article/18/10/linux-data-streams, (data ostatniego dostępu 25.09.25)

2. Piping in Unix or Linux, link:

   https://www.geeksforgeeks.org/linux-unix/piping-in-unix-or-linux/, (data ostatniego dostępu 25.09.25)

3. AWK command in Unix/Linux with examples

   https://www.geeksforgeeks.org/linux-unix/awk-command-unixlinux-examples/, (data ostatniego dostępu 25.09.25)

4. Sed Command in Linux/Unix With Exampleshttps://www.geeksforgeeks.org/linux-unix/sed-command-in-linux-unix-with-examples/, (data ostatniego dostępu 25.09.25)

Fundusze Europejskie dla Rozwoju Społecznego | Rzeczpospolita Polska | Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Materiały do przedmiotu:

Podstawy Systemów Operacyjnych

Laboratorium nr 4:

Edycja plików tekstowych i binarnych

Autor:

Albert Rachwał

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Kolejne laboratorium poświęcone jest edycji plików, zarówno w postaci czytelnego tekstu, jak i w formacie binarnym. Studenci poznają edytory tekstowe dostępne w środowisku terminalowym, które stanowią podstawowe narzędzie pracy administratora – od prostych, intuicyjnych rozwiązań po edytory dające pełną kontrolę nad konfiguracją systemu. Omawiane są również różnice pomiędzy plikami tekstowymi a binarnymi oraz sposoby ich reprezentacji. Uczestnicy zajęć mają okazję przyjrzeć się strukturze plików wykonywalnych i dowiedzieć się, jak analizować ich zawartość przy pomocy narzędzi systemowych. Zajęcia te uczą, że edycja i diagnostyka plików jest umiejętnością niezbędną do pracy w każdym środowisku serwerowym, a znajomość narzędzi tekstowych i binarnych pozwala rozwiązywać problemy także w sytuacjach awaryjnych, kiedy graficzne środowisko użytkownika nie jest dostępne. W ramach zajęć studenci nie tylko uczą się obsługi edytorów konsolowych, ale także wykorzystują je do pisania własnych prostych skryptów powłoki. Tworzenie skryptów w Bashu pozwala na automatyzację powtarzalnych zadań, takich jak operacje na plikach, przetwarzanie tekstu czy monitorowanie systemu. Dzięki temu edycja plików tekstowych nie ogranicza się do konfiguracji, lecz staje się wprowadzeniem do programowania w środowisku systemowym.

Cel ćwiczenia/laboratorium

- Poznanie roli edytorów tekstowych w systemach Linux i ich zastosowań w pracy administratora.
- Opanowanie podstaw pracy w edytorze nano (otwieranie plików, zapisywanie zmian, wyszukiwanie tekstu).
- Zrozumienie idei trybów pracy w vim (tryb normalny, edycji, poleceń).
- Opanowanie podstawowych skrótów w vim (edycja tekstu, zapisywanie, wychodzenie).
- Poznanie różnic między klasycznym vim a nowoczesnym neovim.
- Umiejętność edycji plików konfiguracyjnych w katalogu /etc (np. hosts, passwd, ssh/sshd_config).
- Nabycie umiejętności edycji plików binarnych przy pomocy hexdump, xxd, bvi.
- Poznanie narzędzia hexyl i jego zalet w analizie plików binarnych.
- Rozróżnianie plików tekstowych i binarnych na podstawie ich zawartości i narzędzi (file, strings).
- Analiza plików wykonywalnych ELF przy pomocy strings.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

- Wykorzystanie narzędzia readelf do podglądu nagłówków i sekcji pliku ELF.
- Analiza kodu maszynowego z użyciem objdump.
- Nabycie umiejętności tworzenia i edycji prostych skryptów powłoki (bash).
- Rozróżnienie między prostą edycją plików a analizą niskopoziomową binariów.
- Uświadomienie znaczenia poprawnej edycji plików konfiguracyjnych dla działania całego systemu.

Instrukcja laboratoryjna/ćwiczeniowa

Rodzaje plików

Pliki w systemach operacyjnych można podzielić na różne kategorie w zależności od ich zawartości i przeznaczenia. Najprostszą i najbardziej intuicyjną formą jest plik tekstowy, który przechowuje dane w postaci czytelnej dla człowieka, przy użyciu znaków kodowanych w standardach takich jak ASCII czy UTF-8. Dzięki temu można go bezpośrednio otworzyć w edytorze tekstowym, przeczytać i zmodyfikować. Do tej grupy zaliczają się zarówno zwykłe notatki czy dokumenty, jak i wiele plików konfiguracyjnych systemu, które opisują ustawienia w formie prostych instrukcji zapisanych jako tekst.

Drugim rodzajem są pliki binarne, które zawierają dane zapisane w postaci zakodowanej, trudnej do bezpośredniego odczytania przez człowieka. Mogą to być zarówno programy wykonywalne, jak i obrazy, dźwięki czy archiwa, czyli wszelkie informacje przechowywane w surowej postaci binarnej. Analiza takich plików wymaga specjalnych narzędzi, które potrafią wyświetlić ich zawartość w formie heksadecymalnej lub zinterpretować poszczególne sekcje. Pliki binarne są więc mniej „przyjazne” do ręcznej edycji, ale pozwalają przechowywać złożone struktury danych, które bezpośrednio interpretuje system lub oprogramowanie.

Szczególną rolę pełnią pliki konfiguracyjne, czyli pliki tekstowe zawierające ustawienia systemu operacyjnego lub aplikacji. Dzięki nim można sterować działaniem usług, zmieniać parametry logowania czy definiować reguły bezpieczeństwa. W systemach uniksowych i linuksowych standardowo znajdują się one w katalogu /etc. Przykładowo plik /etc/hosts mapuje nazwy domenowe na adresy IP, a /etc/ssh/sshd_config zawiera konfigurację serwera SSH. Poprawna edycja tych

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

plików jest krytyczna dla prawidłowego funkcjonowania systemu, dlatego administratorzy zawsze muszą wykonywać ją z najwyższą ostrożnością.

W kontekście uruchamiania programów istotne miejsce zajmują pliki wykonywalne, w systemach uniksowych i linuksowych najczęściej w formacie ELF (Executable and Linkable Format). Zawierają one zakodowany kod maszynowy, który może być bezpośrednio interpretowany przez procesor. Pliki ELF mają rozbudowaną strukturę, obejmującą nagłówki, sekcje kodu, dane i informacje potrzebne do linkowania. Analiza ich zawartości pozwala lepiej zrozumieć sposób działania programów, a narzędzia takie jak readelf czy objdump umożliwiają wgląd w niskopoziomowe szczegóły działania systemu.

Warto również wspomnieć o skryptach powłoki, które są specyficzną odmianą plików tekstowych zawierających sekwencję poleceń przeznaczonych do wykonania w powłoce systemowej. Najpopularniejszym przykładem są skrypty Bash, które pozwalają automatyzować codzienne zadania administracyjne i programistyczne. Skrypt może wykonywać operacje na plikach, uruchamiać aplikacje, przetwarzać dane czy monitorować stan systemu. Dzięki swojej prostocie i elastyczności stanowią one podstawę pracy w środowiskach linuksowych, a jednocześnie wprowadzają studentów w świat programowania systemowego.

Kodowanie plików

Podstawą pracy z plikami tekstowymi jest zrozumienie sposobu, w jaki dane są reprezentowane w postaci bajtów. Każdy plik tekstowy zapisuje znaki w określonym systemie kodowania, takim jak ASCII, UTF-8 czy starsze standardy ISO-8859. W systemach Linux domyślnie stosowane jest kodowanie UTF-8, które umożliwia zapis całego zestawu znaków Unicode, obejmującego symbole narodowe i znaki specjalne. Różnice w kodowaniu mogą prowadzić do błędnej interpretacji znaków – na przykład pojawiania się tzw. „krzaczków” przy otwieraniu pliku w innym środowisku lub systemie operacyjnym. W kontekście przenoszenia plików między systemami należy również zwracać uwagę na sposób oznaczania końca linii: w systemach Unix/Linux jest to pojedynczy znak LF (Line Feed), natomiast w systemach Windows sekwencja CRLF (Carriage Return + Line Feed). Świadomość tych różnic jest kluczowa przy edycji i wymianie danych tekstowych, zwłaszcza w projektach wieloplatformowych. Narzędzia takie jak file, iconv czy edytory w trybie

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

hexadecymalnym pozwalają zidentyfikować i konwertować kodowanie znaków, zapobiegając błędom interpretacji danych.

Terminalowe edytory tekstowe

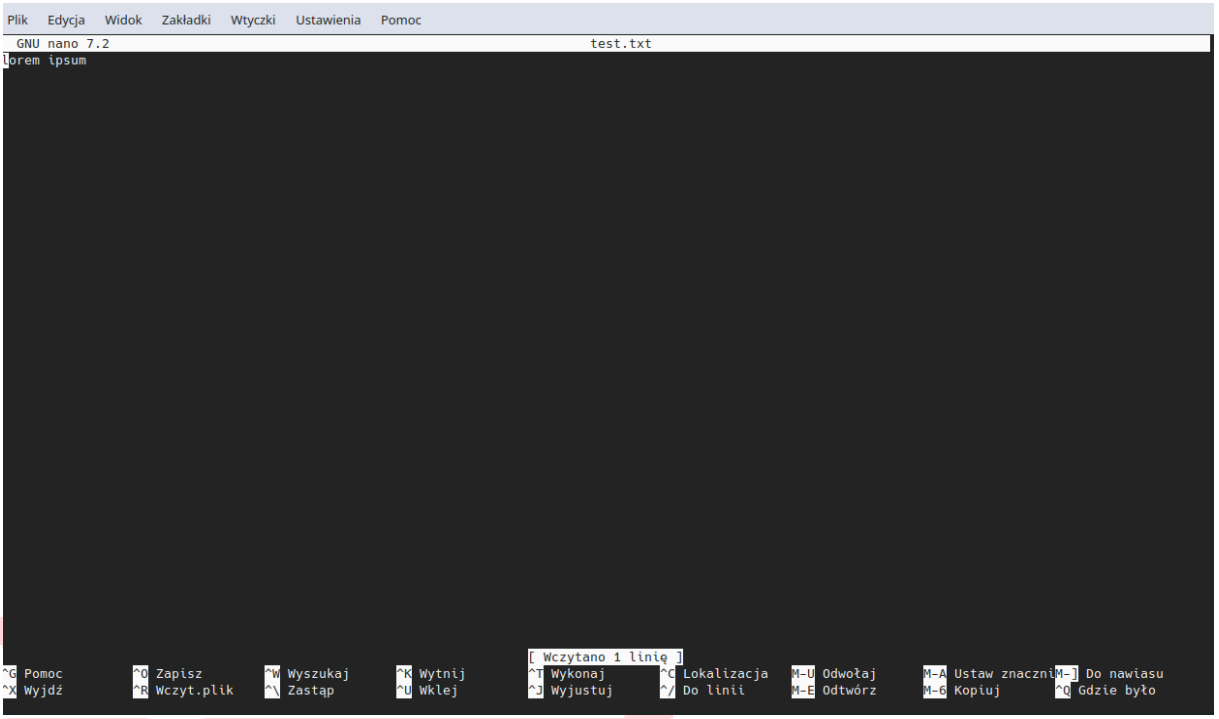
Po omówieniu sposobu reprezentacji tekstu w plikach oraz różnic wynikających z kodowania znaków, kolejnym krokiem jest poznanie narzędzi umożliwiających ich bezpośrednią edycję. W środowisku Linux podstawą pracy administracyjnej są edytory terminalowe, które działają w trybie tekstowym i pozwalają modyfikować pliki bez potrzeby używania interfejsu graficznego. Ich znajomość jest niezbędna w sytuacjach, gdy administrator pracuje zdalnie, w środowisku serwerowym lub naprawia system w trybie awaryjnym.

Edytor nano

Najprostszym i jednocześnie bardzo popularnym edytorem jest nano, przedstawiony na rysuknu 1. Został on zaprojektowany z myślą o łatwości obsługi – wszystkie najważniejsze skróty klawiszowe są widoczne na dole ekranu, dzięki czemu nawet początkujący użytkownik może szybko opanować podstawy. Nano świetnie sprawdza się do szybkiej edycji plików konfiguracyjnych czy krótkich notatek. Choć nie oferuje rozbudowanych możliwości automatyzacji czy pracy z kodem, jego prostota czyni go narzędziem niezwykle przydatnym w codziennych zadaniach administracyjnych.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00



Rys. 1. Edytor nano.

Podstawowe operacje wykonywane najczęściej w edytorze nano:

- Ctrl+O Zapisz plik (Write Out)
- Ctrl+X Wyjdź z edytora (Exit)
- Ctrl+R Wczytaj inny plik do bieżącego dokumentu
- Ctrl+G Wyświetl pomoc (Help)

Poniższe operacje służą do nawigacji wewnątrz edytora oraz wyszukiwania wzorców:

- Ctrl+Y / Ctrl+V Przewiń ekran o stronę w górę / w dół
- Ctrl+A / Ctrl+E Przejdź na początek / koniec bieżącej linii
- Ctrl+W Wyszukaj tekst
- Alt+W Wyszukaj ponownie (następne wystąpienie)
- Ctrl+C Wyświetl numer linii i kolumny (pozycję kursora)
- Ctrl+_ Przejdź do konkretnej linii (wpisz numer)

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Poniższe skróty są przydatne w celu edycji tekstu

- Ctrl+K Wytnij bieżącą linię (Cut)
- Ctrl+U Wklej (Uncut)
- Ctrl+J Wyjustuj (wyrównaj) akapit
- Ctrl+T Uruchom sprawdzanie składni (Spell Check – jeśli dostępne)
- Ctrl+^ (Ctrl + Shift + 6) Zaznacz tekst (ustawia początek zaznaczenia)
- Alt+6 Skopiuj zaznaczenie
- Ctrl+\ Zastąp wyszukany tekst innym (Replace)

Poniżej przedstawiono inne zaawansowane funkcje edytora i przydatne skróty

- Alt+A Włącz/wyłącz tryb zaznaczania tekstu
- Alt+U / Alt+E Cofnij / ponów operację
- Alt+Shift+3 Włącz podświetlanie składni (jeśli skonfigurowane)
- Alt+N Włącz numerację linii
- Ctrl+T Wykonaj zewnętrzne polecenie lub skrypt (np. kompilacja)
- Alt+Shift+U Przejdź do początku pliku
- Alt+Shift+E Przejdź na koniec pliku

Przydatne opcje uruchamiania nano

- nano plik.txt Otwiera plik do edycji
- nano -l plik.txt Włącza numerowanie linii
- nano -m plik.txt Aktywuje obsługę myszy
- nano -i plik.txt Automatyczne wcięcia
- nano -c plik.txt Pokazuje numer bieżącej linii/kolumny
- nano +5 plik.txt Otwiera plik od razu na linii 5
- nano -B plik.txt Tworzy kopię zapasową pliku przed zapisem

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Rys. 2. Przykład edytora vim

Edytor vim

Znacznie bardziej zaawansowanym i jednocześnie kultowym narzędziem jest Vim (skrót od Vi Improved), przedstawiony na rysunku 2. Edytor ten powstał jako rozwinięcie klasycznego vi i stanowi jego współczesną, wieloplatformową wersję, oferującą znacznie szersze możliwości. Vim wyróżnia się nietypowym podejściem do edycji, opartym na trybach pracy, które pozwalają oddzielić czynności pisania od wykonywania poleceń. Podstawowy tryb to tryb normalny (ang. Normal mode), w którym wpisywane klawisze traktowane są jako komendy sterujące, a nie bezpośredni tekst do wstawienia. W tym trybie użytkownik może poruszać się po pliku, kopiować fragmenty (yy), usuwać linie (dd), cofać operacje (u) czy przechodzić do konkretnych fragmentów dokumentu (gg, G). Tryb wstawiania (ang. Insert mode) aktywuje się po naciśnięciu klawisza i (ang. insert), a (ang. append, wstaw po kursorem) lub o (ang. open line, otwarcie nowej linii poniżej). W tym trybie Vim zachowuje się jak klasyczny edytor tekstu, umożliwiając swobodne pisanie i wprowadzanie treści. Powrót do trybu normalnego następuje po wciśnięciu klawisza Esc. Z kolei tryb poleceń (ang. Command-line mode) uruchamia się po naciśnięciu

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

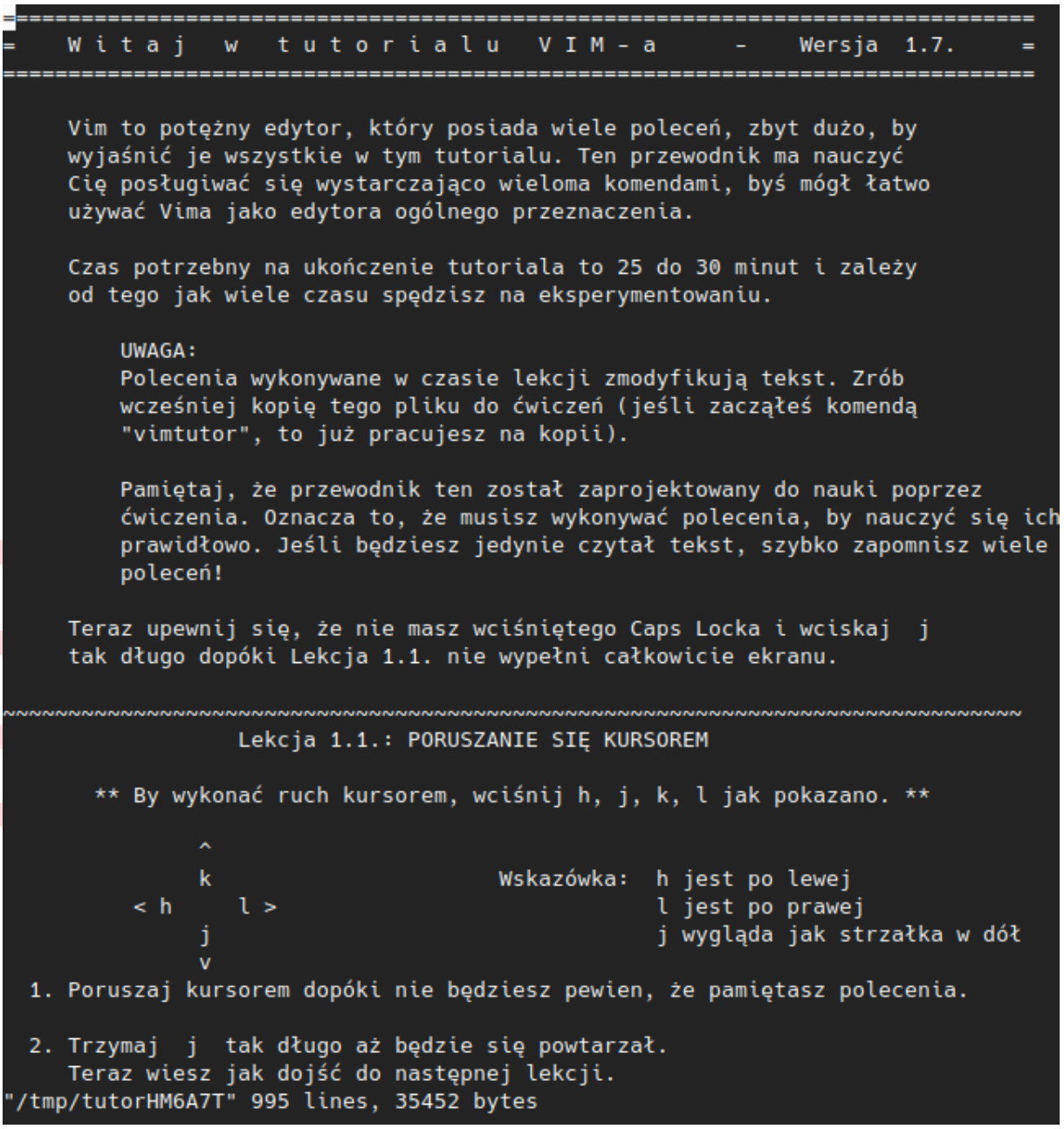
Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

dwukropka (:) w trybie normalnym. Pozwala on na wykonywanie złożonych operacji takich jak zapisywanie pliku (:w), wychodzenie z edytora (:q, :q!), zapisywanie i wyjście jednocześnie (:wq lub :x), wyszukiwanie i zamiana tekstu (:%s/stare/nowe/g), a także uruchamianie zewnętrznych poleceń systemowych (:!ls, :!cat plik.txt).

Dzięki takiemu podejściu Vim zapewnia ogromne możliwości kontroli i edycji tekstu, umożliwiając wykonywanie nawet bardzo złożonych operacji bez potrzeby odrywania rąk od klawiatury. Jednocześnie wymaga od użytkownika opanowania pewnej „filozofii pracy" – rozumienia, w którym trybie się znajduje, oraz świadomego przełączania między nimi. Początkującym użytkownikom może to wydawać się nieintuicyjne, lecz w praktyce znacznie przyspiesza pracę zaawansowanym użytkownikom.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00



Vim zawiera wbudowany interaktywny samouczek, vimtutor przedstawiony na rysunku 3, który można uruchomić poleceniem:

vimtutor

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Jest to przewodnik w formie praktycznych ćwiczeń, przeprowadzający użytkownika krok po kroku przez najważniejsze funkcje edytora — poruszanie się po pliku, wstawianie i usuwanie tekstu, zapisywanie zmian, wyszukiwanie czy operacje na wielu liniach. Samouczek jest częścią standardowej instalacji Vima i zajmuje około 20–30 minut. Uznaje się go za najskuteczniejszy sposób rozpoczęcia nauki obsługi tego edytora, gdyż wymaga aktywnego wykonywania poleceń w rzeczywistym środowisku terminalowym.

Vim działa w oparciu o różne tryby pracy, które pozwalają oddzielić proces pisania tekstu od wykonywania poleceń i sterowania edytorem. Kluczowe jest rozumienie, w którym trybie aktualnie się znajdujemy.

- Tryb normalny (Esc) – podstawowy tryb Vima, w którym klawisze traktowane są jako polecenia. Umożliwia poruszanie się po pliku, kopiowanie, usuwanie i wykonywanie operacji edycyjnych.
- Tryb wstawiania (i, a, o) – pozwala na wprowadzanie tekstu.
- i – wstaw tekst w bieżącym miejscu.
- a – dopisz tekst za kursorem.
- o – otwórz nową linię poniżej i przejdź do edycji.
- Z trybu wstawiania wychodzi się klawiszem Esc.
- Tryb poleceń (:) – służy do wykonywania komend takich jak zapis, wyjście, wyszukiwanie lub uruchamianie zewnętrznych poleceń systemowych.

Vim oferuje precyzyjne sterowanie kursorem bez użycia myszki. Poruszanie się odbywa wyłącznie przy pomocy klawiatury, co znacznie przyspiesza pracę w terminalu.

- h, j, k, l – ruch w lewo, w dół, w górę, w prawo.
- 0 / $ – przejście na początek lub koniec bieżącej linii.
- gg / G – skok na początek lub koniec całego pliku.
- w / b – przeskok do następnego lub poprzedniego słowa.
- Ctrl+f / Ctrl+b – przewinięcie strony w dół lub w górę.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

W trybie normalnym użytkownik może szybko wykonywać operacje na liniach i fragmentach tekstu. Większość komend edycyjnych jest intuicyjna i opiera się na prostych skrótach.

- x – usuń znak pod kursorem.
- dd – usuń bieżącą linię.
- yy – skopiuj (yank) linię.
- p – wklej linię poniżej kursora.
- u – cofnij ostatnią operację (undo).
- Ctrl+r – przywróć cofniętą zmianę (redo).
- . – powtórz ostatnie wykonane polecenie.

Vim umożliwia błyskawiczne wyszukiwanie i podmianę tekstu w całym pliku, zarówno w sposób interaktywny, jak i za pomocą wyrażeń regularnych.

- /tekst – wyszukaj frazę w dół pliku.
- ?tekst – wyszukaj frazę w górę pliku.
- n / N – przejdź do następnego lub poprzedniego wystąpienia szukanego tekstu.
- :%s/stare/nowe/g – zamień wszystkie wystąpienia słowa stare na nowe w całym pliku.

Zapis i wyjście z pliku wykonuje się z poziomu trybu poleceń. W Vimie nie ma przycisku „Zapisz” – wszystko odbywa się przez komendy wprowadzone po dwukropku.

- :w – zapisuje plik (write).
- :q – wychodzi z edytora (quit) tylko wtedy, gdy nie ma niezapisanych zmian.
- :wq lub :x – zapisuje i wychodzi jednocześnie.
- :q! – wychodzi bez zapisywania zmian (force quit).
- :w nowy.txt – zapisuje bieżący plik pod nową nazwą.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Vim oferuje szereg przydatnych poleceń konfiguracyjnych, które można wywoływać zarówno w danym pliku, jak i zapisać na stałe w pliku konfiguracyjnym ~/.vimrc.

- :set number – włącza numerację linii.
- :syntax on – aktywuje kolorowanie składni.
- :set autoindent – włącza automatyczne wcięcia, przydatne w kodzie źródłowym.
- :set mouse=a – umożliwia korzystanie z myszki w terminalu.
- :split plik2.txt – dzieli ekran i otwiera drugi plik w osobnym oknie.
- :e plik2.txt – otwiera inny plik w tym samym oknie.

Nowoczesnym rozwinięciem vima jest neovim, który zachowuje kompatybilność z oryginałem, ale jednocześnie wprowadza wiele usprawnień i nowych funkcji. Neovim jest lepiej przystosowany do integracji z nowoczesnymi narzędziami programistycznymi, wspiera rozszerzenia i wtyczki, a także posiada aktywną społeczność rozwijającą jego możliwości. Dzięki temu staje się wygodnym środowiskiem pracy dla osób, które chcą połączyć minimalizm i szybkość vima z funkcjonalnością współczesnych edytorów kodu.

Obok edytorów interaktywnych warto wspomnieć o narzędziu takim jak less, które nie służy do edycji, lecz do podglądu zawartości plików. Less umożliwia szybkie przewijanie dokumentów, wyszukiwanie fraz i sprawne poruszanie się po dużych plikach tekstowych, takich jak logi systemowe czy długie pliki konfiguracyjne. W praktyce jest niezastąpiony wtedy, gdy trzeba coś przejrzeć bez ryzyka przypadkowej zmiany zawartości.

Pliki binarne

Podstawową cechą odróżniającą pliki tekstowe od binarnych jest sposób, w jaki dane są w nich reprezentowane i interpretowane przez system operacyjny oraz programy użytkowe. Plik tekstowy zawiera dane zapisane w postaci znaków kodowanych zgodnie z określonym standardem (np. ASCII lub UTF-8), dzięki czemu jego zawartość może być bezpośrednio odczytana i modyfikowana w edytorze tekstowym. Każdy znak odpowiada konkretnemu bajtowi lub sekwencji bajtów, a koniec linii oznaczany jest znakami sterującymi LF lub CRLF. Tego typu pliki wykorzystywane są

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

powszechnie w konfiguracjach systemowych, skryptach, logach i kodzie źródłowym programów. Plik binarny natomiast przechowuje dane w postaci bajtów, które nie mają jednoznacznego odwzorowania na znaki czy symbole tekstowe. Może zawierać liczby, obrazy, dane wykonywalne lub skompresowane struktury binarne, które wymagają interpretacji przez odpowiedni program. Otworzenie takiego pliku w zwykłym edytorze tekstowym skutkuje wyświetleniem nieczytelnych znaków, ponieważ edytor próbuje zinterpretować dane binarne jako znaki tekstu. Struktura plików binarnych często zawiera nagłówki, metadane oraz fragmenty kodu maszynowego, a jej przypadkowa modyfikacja może prowadzić do trwałego uszkodzenia pliku. Administrator powinien umieć rozpoznać typ pliku i dobrać odpowiednie narzędzie do jego analizy. W systemach Linux wstępne rozpoznanie rodzaju pliku umożliwia polecenie file, które analizuje sygnaturę bajtową i zwraca informację o formacie danych. Do przeglądania zawartości plików binarnych stosuje się narzędzia takie jak hexdump, xxd czy strings, pozwalające odczytać dane w reprezentacji heksadecymalnej lub wyszukać w nich czytelne fragmenty tekstu.

Zanim przystąpi się do edycji lub modyfikacji pliku, warto sprawdzić jego rzeczywisty typ, strukturę i integralność. W systemach Linux istnieje zestaw narzędzi diagnostycznych, które pozwalają szybko zidentyfikować zawartość pliku, odczytać jego metadane oraz porównać różne wersje. Dzięki nim można uniknąć prób edycji danych w niewłaściwym formacie, co mogłoby doprowadzić do ich uszkodzenia. Do podstawowych narzędzi tego typu należą:

- file – rozpoznaje typ pliku na podstawie jego sygnatury bajtowej, niezależnie od rozszerzenia. Pozwala odróżnić plik tekstowy od binarnego oraz określić jego format (np. ELF, JPEG, PDF). Przykład użycia:

  ```
  file /bin/ls
  ```

- stat – wyświetla szczegółowe informacje o pliku: rozmiar, prawa dostępu, właściciela, daty modyfikacji i numer inode. Jest przydatne przy analizie zmian i uprawnień. Przykład użycia:

  ```
  stat dokument.txt
  ```


Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

- strings – wyszukuje w pliku binarnym fragmenty tekstu możliwe do odczytania. Pomaga w identyfikacji fragmentów kodu, komunikatów błędów lub metadanych w plikach wykonywalnych. Przykład użycia:

```
strings /bin/bash | grep version
```

- diff – porównuje dwa pliki tekstowe, wskazując różnice linia po linii. Użyteczne przy analizie zmian w skryptach, logach czy konfiguracjach. Przykład użycia:

```
diff config_old.txt config_new.txt
```

- cmp – porównuje pliki bajt po bajcie i zgłasza pierwszą wykrytą różnicę, co czyni go idealnym narzędziem do pracy z plikami binarnymi. Przykład użycia:

```
cmp obraz1.bin obraz2.bin
```

- md5sum, sha256sum – generują sumy kontrolne plików w postaci skrótu kryptograficznego, pozwalając zweryfikować integralność plików oraz upewnić się, że nie zostały one zmodyfikowane lub uszkodzone w trakcie przesyłania. Przykład użycia:

```
sha256sum archiwum.iso
```

Porównywanie plików w systemach Linux pozwala nie tylko wykryć różnice między ich zawartością, ale także zrozumieć, jak dane ewoluują w czasie. Polecenia diff i cmp stanowią podstawę mechanizmów śledzenia zmian wykorzystywanych w systemach kontroli wersji, takich jak Git. Dzięki temu te same zasady analizy różnic między plikami tekstowymi są stosowane później na poziomie całych projektów, gdzie umożliwiają porównywanie, scalanie i archiwizację kolejnych wersji kodu.

Narzędzia do analizy i edycji plików binarnych

Praca z plikami binarnymi wymaga specjalnych narzędzi, które pozwalają spojrzeć na dane w ich surowej, niskopoziomowej postaci. Edytory i podglądarki binarne, takie jak hexdump, xxd, bvi czy hexyl, umożliwiają analizę oraz modyfikację zawartości plików bajt po bajcie, co jest kluczowe zarówno w diagnostyce systemów, jak i w inżynierii wstecznej.

hexdump to klasyczne narzędzie służące do podglądu plików binarnych w formie heksadecymalnej i ASCII. Umożliwia analizę struktury danych zapisanych na dysku i

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

pozwala zobaczyć surową zawartość bajt po bajcie. Jest używane m.in. do weryfikacji nagłówków plików, sprawdzania różnic pomiędzy dwiema wersjami binarek albo do diagnostyki uszkodzonych danych. Szczególnie przydatna jest opcja kanonicznego widoku, gdzie obok adresów bajtów widać zarówno wartości hex, jak i odpowiadające im znaki ASCII.

Najważniejsze opcje:

-C – kanoniczny widok: adres, wartości hex i ASCII w jednej tabeli.

-n N – ograniczenie wyjścia do pierwszych N bajtów pliku.

-s N – rozpoczęcie analizy od określonego offsetu (pominięcie pierwszych bajtów).

-v – wyświetlenie pełnej zawartości pliku, bez zastępowania powtarzających się linii symbolem *.

xxd pełni podobną funkcję jak hexdump, lecz jest znacznie bardziej elastyczne. Pozwala nie tylko wyświetlać plik w postaci heksadecymalnej, ale również przekształcać go w tekstowy zapis hex, a następnie odtwarzać binarkę z tego zapisu. Dzięki temu jest wykorzystywane zarówno do analizy, jak i do modyfikacji – student może wyeksportować plik, dokonać zmian w edytorze tekstowym, a później odbudować go z powrotem w formie binarnej. To sprawia, że xxd bywa przydatne w inżynierii wstecznej, testach bezpieczeństwa i przy naprawie uszkodzonych plików.

Najważniejsze opcje:

xxd plik – podstawowy podgląd zawartości pliku w hexie.

-c N – określenie liczby bajtów wyświetlanych w jednej linii (domyślnie 16).

-p – zapis „plain”, tylko ciąg znaków hex, bez adresów i ASCII.

-r – odtworzenie pliku binarnego z heksadecymalnej reprezentacji.

-g1 – grupowanie wyjścia po jednym bajcie, co ułatwia analizę.

bvi (Binary Visual Editor) to edytor plików binarnych wzorowany na popularnym vi. W przeciwieństwie do prostych viewerów umożliwia interaktywną edycję bajtów w pliku. Dzięki temu można ręcznie zmieniać zawartość plików binarnych – od testowych

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

eksperymentów, jak zmiana znaku w kodzie ASCII, aż po poważniejsze modyfikacje nagłówków plików czy fragmentów programów wykonywalnych. Obsługa opiera się na filozofii vima, więc osoby zaznajomione z tym edytorem łatwo odnajdą się w pracy z bvi.

Najważniejsze opcje:

bvi plik – otwarcie pliku do edycji w hexie.

bmore plik – tryb tylko do odczytu, wygodny do bezpiecznego podglądu.

bvedit plik – edycja z możliwością konwersji między hexem a tekstem.

hexyl to nowoczesny i bardziej przyjazny odpowiednik narzędzi takich jak hexdump. Jego największą zaletą jest kolorowe wyróżnianie bajtów, dzięki czemu nawet osoby początkujące mogą szybciej zrozumieć strukturę danych. Kolory pozwalają natychmiast odróżnić nagłówki, dane powtarzalne czy nietypowe sekwencje. Narzędzie to świetnie sprawdza się w edukacji, przy wizualizacji działania plików binarnych, ale też w praktyce administratorów i programistów, gdy potrzebny jest szybki i czytelny podgląd zawartości pliku.

Najważniejsze opcje:

hexyl plik – standardowy podgląd zawartości pliku w kolorze.

-n N – wyświetlenie jedynie pierwszych N bajtów.

-s N – rozpoczęcie analizy od wskazanego offsetu.

Pliki wykonywalne

Pliki wykonywalne w systemach uniksowych i linuksowych mają zwykle format ELF (Executable and Linkable Format). Jest to standardowy sposób przechowywania programów, bibliotek i obiektów, dzięki któremu system operacyjny wie, jak załadować kod do pamięci i uruchomić go na procesorze. Analiza plików ELF pozwala zajrzeć „pod maskę” programów, sprawdzić ich strukturę i dowiedzieć się, jak działają. Do tego celu używa się kilku wyspecjalizowanych narzędzi, które pozwalają rozpoznawać, diagnozować i analizować kod maszynowy.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Polecenie file jest najprostszym narzędziem do sprawdzenia, z jakim typem pliku mamy do czynienia. W przypadku ELF potrafi wskazać, że dany plik jest programem wykonywalnym, biblioteką dynamiczną czy obiektem do linkowania. Dodatkowo podaje architekturę (np. x86-64), typ pliku (statyczny, dynamiczny), a także podstawowe informacje o formacie. Dzięki temu już na samym początku można upewnić się, że dany plik faktycznie jest wykonywalny i dla jakiego środowiska został zbudowany.

Najważniejsze opcje:

file plik – automatyczne rozpoznanie typu pliku.

-i plik – pokazuje typ MIME pliku, przydatny np. przy analizie przesyłanych danych.

Polecenie strings pozwala wydobyć z pliku binarnego wszystkie fragmenty, które wyglądają jak czytelny tekst w ASCII lub UTF-8. Nawet w skompilowanych programach znajdują się łańcuchy znaków – komunikaty błędów, fragmenty interfejsu, ścieżki plików czy adresy URL. Dzięki strings można je łatwo wyciągnąć i przeanalizować. Narzędzie to bywa wykorzystywane w analizie bezpieczeństwa, gdyż pozwala wykryć zaszyte dane albo podejrzane komunikaty w binarkach.

Najważniejsze opcje:

strings plik – wydobycie wszystkich czytelnych napisów.

-n N – pokazuje tylko napisy o długości co najmniej N znaków (domyślnie 4).

-t x – wyświetla również offsety w pliku, gdzie znajdują się napisy (np. w hex).

readelf to specjalistyczne narzędzie stworzone do analizy plików ELF. Umożliwia podgląd nagłówków, tabel sekcji i segmentów, symboli, a także informacji o zależnościach bibliotecznych. Dzięki niemu można sprawdzić m.in. architekturę, typ pliku, punkt wejścia programu czy listę funkcji eksportowanych przez bibliotekę. Jest to narzędzie bardziej szczegółowe i precyzyjne niż file, używane przez programistów i administratorów, którzy chcą dokładnie zrozumieć strukturę pliku wykonywalnego.

Najważniejsze opcje:

readelf -h plik – nagłówki pliku ELF (architektura, typ, punkt wejścia).

readelf -S plik – lista sekcji pliku (np. .text, .data, .rodata).

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

readelf -s plik – tabela symboli, pokazuje funkcje i zmienne.

readelf -d plik – zależności dynamiczne (jakie biblioteki są wymagane).

objdump to narzędzie bardziej zaawansowane, które potrafi zdeasemblować plik ELF, czyli pokazać jego kod maszynowy w postaci asemblera. Dzięki temu można zobaczyć, jakie instrukcje procesora będą faktycznie wykonywane. To narzędzie bywa używane w debugowaniu, optymalizacji i inżynierii wstecznej. W połączeniu z strings i readelf daje pełny obraz działania programu: od nagłówków, przez symbole, aż po kod.

Najważniejsze opcje:

objdump -d plik – deasemblacja kodu maszynowego.

objdump -h plik – nagłówki sekcji.

objdump -t plik – tabela symboli.

objdump -x plik – pełna analiza nagłówków i metadanych.

Pliki konfiguracyjne systemu w katalogu /etc

Katalog /etc to jedno z najważniejszych miejsc w systemach uniksowych i linuksowych. Zawiera on pliki konfiguracyjne, które decydują o sposobie działania całego systemu – od podstawowych ustawień sieciowych, po zarządzanie użytkownikami i usługami. Można powiedzieć, że /etc pełni rolę „centrum sterowania" systemem, a jego zawartość jest kluczowa dla administratora. Edytowanie tych plików wymaga ostrożności, ponieważ najmniejszy błąd w składni może spowodować, że usługa nie uruchomi się poprawnie albo cały system stanie się niestabilny.

Wśród plików znajdujących się w katalogu /etc na szczególną uwagę zasługują trzy, które użytkownicy spotykają najczęściej. /etc/hosts odpowiada za lokalne mapowanie nazw domenowych na adresy IP. Dzięki niemu można przypisać konkretnej nazwie (np. test.local) określony adres, co bywa przydatne podczas testowania aplikacji czy w środowiskach bez dostępu do serwerów DNS. Kolejnym niezwykle istotnym plikiem jest /etc/passwd, w którym przechowywane są informacje o kontach użytkowników, takie jak nazwa loginu, identyfikator użytkownika (UID), identyfikator grupy (GID) czy katalog domowy. Dawniej zawierał on również hasła, lecz obecnie są one trzymane w

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

bardziej zabezpieczonym pliku /etc/shadow. Trzecim przykładem jest /etc/ssh/sshd_config, czyli główny plik konfiguracyjny dla usługi SSH. To właśnie w nim administrator określa zasady zdalnego logowania, takie jak dopuszczalne metody uwierzytelniania, port, na którym działa serwer, czy ograniczenia dostępu dla wybranych użytkowników.

Ogromne znaczenie ma także kwestia uprawnień do plików konfiguracyjnych. Z reguły mogą je edytować wyłącznie administratorzy (root), podczas gdy zwykli użytkownicy mają co najwyżej możliwość ich odczytu. Dzięki temu krytyczne ustawienia systemowe są chronione przed przypadkowymi zmianami. Uprawnienia odgrywają tu rolę nie tylko w aspekcie bezpieczeństwa, lecz także integralności działania systemu – niewłaściwie nadane prawa mogłyby umożliwić nieautoryzowanemu użytkownikowi manipulację konfiguracją usług, co narażałoby system na awarie lub ataki. Dlatego poprawna edycja plików w katalogu /etc zawsze powinna odbywać się w trybie uprzywilejowanym, a zmiany należy wprowadzać ostrożnie i po wcześniejszym wykonaniu kopii zapasowej.

Zastosowanie Midnight Commander

Midnight Commander (MC) jest klasycznym menedżerem plików działającym w trybie tekstowym (TUI - Text User Interface), który stanowi funkcjonalny odpowiednik graficznych eksploratorów plików w środowiskach CLI. Narzędzie to pozwala w sposób szybki i przejrzysty wykonywać większość operacji na plikach i katalogach bez konieczności wpisywania długich poleceń w terminalu. Dzięki temu MC stanowi efektywne rozwiązanie dla administratorów systemów oraz użytkowników pracujących zdalnie przez SSH, gdzie dostęp do interfejsu graficznego jest ograniczony lub niemożliwy.

Po uruchomieniu polecenia mc użytkownik otrzymuje dwupanelowy interfejs, w którym lewy i prawy panel mogą niezależnie przeglądać różne katalogi. Między panelami można przełączać się klawiszem TAB, natomiast operacje na plikach wykonywane są za pomocą klawiszy funkcyjnych. Najczęściej używane skróty to: F3 – podgląd pliku, F4 – edycja, F5 – kopiowanie, F6 – przenoszenie, F7 – tworzenie katalogu oraz F8 – usuwanie. Dzięki temu MC znacząco przyspiesza pracę w

środowisku konsolowym i redukuje ryzyko błędów wynikających z ręcznego wpisywania poleceń.

Midnight Commander umożliwia zarówno przeglądanie, jak i edycję plików tekstowych oraz binarnych. Wbudowany podgląd (viewer) pozwala analizować zawartość pliku w trybie tekstowym, heksadecymalnym lub binarnym, co czyni go przydatnym narzędziem podczas zajęć z analizy struktur plików. Funkcja edycji (internal editor) obsługuje kolorowanie składni i prostą nawigację po tekście, dzięki czemu może być używana jako lekki zamiennik edytorów takich jak nano czy vim w prostych zastosowaniach.

Jedną z mocnych stron MC jest integracja z powłoką systemową. Każdy panel może być powiązany z lokalnym lub zdalnym systemem plików – obsługiwane są m.in. protokoły SSH, FTP oraz SFTP. Dzięki temu możliwe jest bezpośrednie kopiowanie plików pomiędzy serwerami bez konieczności użycia zewnętrznych narzędzi transferowych. MC potrafi również otwierać i przeglądać zawartość archiwów takich jak .tar, .zip czy .rpm w taki sam sposób, jak zwykłe katalogi, co pozwala na szybkie analizowanie ich struktury bez rozpakowywania.

Dodatkową zaletą Midnight Commandera jest wbudowany pasek poleceń, który umożliwia wpisywanie i uruchamianie komend powłoki bez wychodzenia z programu. Pozwala to łączyć klasyczne operacje terminalowe z wygodą interfejsu panelowego – przykładowo, można skopiować plik w jednym panelu i natychmiast uruchomić jego analizę poleceniem file czy hexdump.

Ćwiczenia samodzielne

Zadania należy wykonywać w zwykłej powłoce bash w katalogu domowym. Nie wymagają uprawnień administratora.

1. Otwórz w edytorze nano plik dane.txt, wpisz kilka linijek tekstu, zapisz zmiany i zamknij edytor.
2. Utwórz w vim nowy plik notatki.txt, dodaj trzy linie tekstu, zapisz i wyjdź poleceniem :wq.
3. Przećwicz w vim trzy podstawowe tryby pracy: tryb wstawiania (i), tryb normalny (ESC) oraz tryb poleceń (:).

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

4. W vim wpisz polecenie :set number, aby włączyć numerowanie linii. Zapisz to ustawienie w pliku konfiguracyjnym ~/.vimrc.
5. Zainstaluj i uruchom neovim, a następnie otwórz nim plik .bashrc. Porównaj interfejs i obsługę z klasycznym vim.
6. Otwórz plik /etc/hosts w nano i dodaj nową linię z wpisem 127.0.0.1 test.local.

   Uwaga: Pliki takie, jak /etc/hosts lub /etc/passwd są systemowe i wymagają uprawnień sudo do edycji.

7. Otwórz plik /etc/passwd poleceniem less, a następnie w vim wyszukaj słowo root poleceniem /root.
8. Użyj poleceń cat i file dla plików /etc/hosts i /etc/passwd, aby porównać ich typy oraz sposób wyświetlania zawartości.
9. Utwórz plik binarny poleceniem:

   dd if=/dev/urandom of=binarka.bin bs=512 count=1

10. Podejrzyj zawartość pliku binarka.bin przy pomocy hexdump -C binarka.bin.
11. Użyj polecenia xxd do przekształcenia pliku binarnego w zapis heksadecymalny:

    xxd binarka.bin > binarka.hex

12. Odtwórz plik binarny z heksadecymalnego poleceniem:

    xxd -r binarka.hex binarka_odtworzona.bin

13. Porównaj oba pliki (binarka.bin i binarka_odtworzona.bin) przy pomocy polecenia cmp i sprawdź, czy są identyczne.
14. Użyj poleceń file binarka.bin oraz file dane.txt, aby porównać typ pliku binarnego i tekstowego.
15. Wykonaj polecenie strings /bin/ls | head -n 20 i znajdź wśród wyników fragmenty tekstu zrozumiałe dla człowieka.
16. Wyświetl nagłówki pliku ELF przy pomocy readelf -h /bin/bash i zanotuj typ pliku, architekturę oraz rozmiar nagłówka.
17. Zbadaj strukturę kodu pliku /bin/echo poleceniem objdump -d /bin/echo | head -n 40 i znajdź fragment z instrukcjami call, mov lub printf.
18. Wyświetl sekcje pliku ELF przy pomocy readelf -S /bin/echo i sprawdź, gdzie znajdują się sekcje .text i .data.

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

19. Zainstaluj edytor binarny bvi, otwórz nim plik binarka.bin, zmodyfikuj pierwszy bajt i porównaj różnice poleceniem xxd lub diff.
20. Napisz prosty skrypt analiza_bin.sh, który przyjmuje nazwę pliku jako argument i wypisuje kolejno: typ pliku (file), liczbę ciągów tekstowych (strings | wc -l) oraz rozmiar w bajtach (stat -c %s).
21. Zainstaluj narzędzie Midnight Commander (mc) i uruchom je poleceniem mc. Zapoznaj się z interfejsem, nawigacją po katalogach (klawisze TAB, ENTER, F10) oraz podstawowymi skrótami klawiszowymi.
22. Otwórz w mc dwa panele – w jednym przejdź do katalogu domowego, w drugim do /bin. Znajdź plik wykonywalny ls i podejrzyj jego zawartość przy pomocy F3 w trybie podglądu binarnego (hex view). Następnie otwórz plik dane.txt w trybie tekstowym i porównaj sposób wyświetlania zawartości.
23. Wykorzystując mc, skopiuj plik binarka.bin do katalogu test/ i zmień jego nazwę na kopiabin.bin. Zwróć uwagę, że zmiana rozszerzenia nie wpływa na faktyczny typ pliku – sprawdź to poleceniem file kopiabin.bin.

Pytania pomocnicze

1. Czym różni się plik tekstowy od binarnego?
2. Jakie są trzy podstawowe edytory tekstowe w systemach Linux?
3. Jakie są podstawowe funkcje edytora nano?
4. W jaki sposób vim różni się od prostych edytorów takich jak nano?
5. Jakie są trzy tryby pracy w edytorze vim?
6. Jakie polecenie w vim służy do zapisania i wyjścia z pliku?
7. Czym różni się neovim od klasycznego vim?
8. W jakim katalogu znajdują się najważniejsze pliki konfiguracyjne systemu Linux? Podaj przykłady.
9. Do czego służy polecenie hexdump?
10. Jakie narzędzie pozwala na edycję plików binarnych w sposób interaktywny?
11. Jakie korzyści daje użycie hexyl zamiast klasycznego hexdump?
12. Do czego służy polecenie strings w analizie plików wykonywalnych?
13. Jakie informacje można uzyskać z pliku ELF za pomocą readelf?
14. Do czego wykorzystuje się objdump?
15. Dlaczego poprawna edycja plików konfiguracyjnych jest krytyczna dla działania systemu?

Podsumowanie

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Podczas czwartego laboratorium studenci poznali różnice pomiędzy plikami tekstowymi i binarnymi oraz narzędzia służące do ich edycji i analizy. W pierwszej części zajęć omówiono sposób reprezentacji danych w plikach tekstowych, znaczenie kodowania znaków (UTF-8, ASCII) oraz rolę znaków końca linii w systemach Unix i Windows. Następnie uczestnicy zapoznali się z pracą w edytorach terminalowych – nano, vim i neovim – ucząc się podstawowych trybów pracy, zapisywania zmian, wyszukiwania oraz edycji plików konfiguracyjnych. Szczególną uwagę zwrócono na tworzenie i personalizację plików .vimrc oraz na dobre praktyki pracy z plikami systemowymi.

Druga część laboratorium dotyczyła analizy i edycji plików binarnych. Studenci zapoznali się z narzędziami hexdump, xxd i strings, umożliwiającymi interpretację zawartości plików na poziomie bajtowym. Poznali także zasadę działania edytorów heksadecymalnych (hexedit) i zrozumieli ryzyko związane z bezpośrednią modyfikacją danych binarnych. Kolejnym etapem było wykorzystanie poleceń diff i cmp do porównywania plików oraz analiza różnic między wersjami tekstowymi i binarnymi. Ćwiczenia praktyczne pokazały również, jak wyniki poprzednich poleceń (np. grep lub awk) mogą być zapisywane do plików tekstowych i poddawane dalszej edycji.

Zajęcia miały na celu nie tylko opanowanie obsługi narzędzi edycyjnych, lecz także zrozumienie istoty reprezentacji danych w systemie plików. Dzięki nim studenci zdobyli umiejętność bezpiecznej pracy z plikami konfiguracyjnymi i diagnostycznymi oraz poznali podstawy analizy danych binarnych, stanowiące punkt wyjścia do kolejnych laboratoriów dotyczących przetwarzania, automatyzacji i skryptowania w systemie Linux.

Literatura

1. Vim Editor in Linux, Link: https://www.geeksforgeeks.org/linux-unix/getting-started-with-vim-editor-in-linux/, (data ostatniego dostępu 25.09.25)

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

2. Nano text editor in Linux, Link: https://www.geeksforgeeks.org/linux-unix/nano-text-editor-in-linux/, (data ostatniego dostępu 25.09.25)
3. Binary File Viewer in Linux: A Comprehensive Guide, Link: https://linuxvox.com/blog/binary-file-viewer-linux/, (data ostatniego dostępu 25.09.25)

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Materiały do przedmiotu:

Podstawy Systemów Operacyjnych

Laboratorium nr 5:

Podstawy zarządzania i konfiguracji usług w systemie

Autor:

Albert Rachwał

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Piąte laboratorium rozszerza perspektywę z pojedynczych procesów i plików na całość usług działających w systemie. Uczestnicy dowiadują się, czym są usługi systemowe i demony, jak są uruchamiane przy starcie i w jaki sposób można nimi zarządzać. Szczególną uwagę poświęca się współczesnym mechanizmom inicjalizacji, w tym systemowi systemd, który pozwala definiować jednostki usług, kontrolować ich cykl życia oraz analizować zależności. Zajęcia rozpoczynają się od analizy mechanizmu tworzenia i cyklu życia procesów, zrozumienia ich hierarchii (rodzic–potomek), identyfikatorów (PID, PPID) oraz stanów (proces aktywny, uśpiony, zombie czy osierocony). Studenci uczą się rozpoznawać, monitorować i sterować procesami za pomocą poleceń ps, top, htop, kill, pkill oraz pstree. Studenci uczą się analizować strukturę i zależności pomiędzy procesami, rozpoznawać usługi uruchamiane przy starcie, a także diagnozować ich stan i reagować na błędy. W dalszej części laboratorium omówione zostaną narzędzia umożliwiające zarządzanie cyklem życia usług (systemctl), ich logami (journalctl), a także konfigurację automatycznych zadań z wykorzystaniem cron i anacron. Ćwiczenia obejmują również prace z rzeczywistymi usługami, takimi jak serwer SSH, Nginx, system zapory UFW czy mechanizmy bezpieczeństwa AppArmor i Fail2Ban.

Cel ćwiczenia/laboratorium

- Poznanie modelu procesów w systemie Linux oraz ich reprezentacji w tablicy procesów.
- Zrozumienie zależności między procesami rodziców i potomków oraz mechanizmu dziedziczenia PID i PPID.
- Identyfikacja stanów procesów (ACTIVE, SLEEP, STOPPED, ZOMBIE, ORPHAN).
- Opanowanie podstawowych narzędzi do monitorowania procesów (ps, top, htop, pstree).
- Poznanie mechanizmów komunikacji między procesami poprzez sygnały (kill, pkill, trap).
- Zrozumienie, czym różnią się procesy krótkotrwałe od demonów (działających w tle).

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

- Poznanie zasad inicjalizacji systemu i roli procesu PID 1 (systemd).
- Zrozumienie koncepcji usług systemowych i ich roli w architekturze systemu operacyjnego.
- Poznanie zasad działania menedżera usług systemd i jego relacji z procesami systemowymi.
- Opanowanie podstawowych narzędzi do monitorowania, uruchamiania i konfiguracji usług (systemctl, service, journalctl).
- Rozróżnienie między tradycyjnym mechanizmem inicjalizacji init a współczesnym podejściem opartym na jednostkach .service.
- Zrozumienie struktury plików jednostek usługowych i ich podstawowych sekcji (Unit, Service, Install).
- Nabycie umiejętności diagnozowania problemów z uruchamianiem usług, analizy logów i identyfikacji błędów.
- Konfiguracja i testowanie kluczowych usług systemowych, takich jak sshd, nginx, cron czy fail2ban.
- Zrozumienie zasad automatyzacji zarządzania usługami z wykorzystaniem narzędzi takich jak Ansible.
- Poznanie podstawowych metod zabezpieczania usług przy użyciu UFW (firewall) i AppArmor (kontrola dostępu).
- Uświadomienie znaczenia automatycznego uruchamiania usług przy starcie systemu i ich wpływu na stabilność i bezpieczeństwo środowiska.

Instrukcja laboratoryjna/ćwiczeniowa

Procesy w systemie Linux

Proces w systemie Linux jest podstawową jednostką wykonawczą reprezentującą uruchomiony program. Każdy proces posiada własną przestrzeń adresową, licznik rozkazów, zestaw rejestrów oraz kontekst wykonania, który umożliwia jego planowanie przez jądro systemu. Informacje o wszystkich aktywnych procesach są przechowywane w tzw. tablicy procesów – strukturze danych jądra, zawierającej m.in. identyfikator procesu (PID), identyfikator procesu nadrzędnego (PPID), nazwę użytkownika, stan wykonania, zużycie zasobów oraz ścieżkę do programu uruchomionego przez proces. Procesy tworzą hierarchiczną strukturę, w której każdy nowy proces jest tworzony przez inny proces poprzez wywołanie systemowe fork(), a następnie może zastąpić swój kod nowym programem dzięki funkcji exec(). Proces nadrzędny pozostaje odpowiedzialny za odbieranie statusu zakończenia potomka, a

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

jego brak prowadzi do powstania procesu osieroconego, który zostaje automatycznie przejęty przez proces systemd (PID 1). Jeśli rodzic zakończy się przed odebraniem statusu dziecka, powstaje proces zombie, widoczny w tablicy procesów ze statusem Z. Każdy proces znajduje się w jednym z kilku stanów: aktywnym (R), uśpionym (S), zatrzymanym (T), zombie (Z) lub oczekującym (D). System planuje ich wykonanie na podstawie priorytetów i kwantów czasu. Identyfikacja procesów odbywa się za pomocą narzędzi takich jak ps, top, htop czy pstree, które prezentują dane o bieżących procesach, ich hierarchii i zużyciu zasobów. Kontrola nad procesami realizowana jest przez sygnały systemowe (kill, pkill, trap), pozwalające je zatrzymać, wznowić lub zakończyć w kontrolowany sposób. Uruchomienie procesu w tle (&) powoduje odłączenie go od terminala, co umożliwia równoległe wykonywanie wielu zadań.

Hierarchia, dziedziczenie i zarządzanie procesami w systemie Linux

System operacyjny Linux opiera się na hierarchicznej strukturze procesów, w której każdy proces (z wyjątkiem procesu inicjalizacyjnego) ma swojego rodzica. Nowy proces jest tworzony przez inny działający proces za pomocą wywołania systemowego fork(), które kopiuje kontekst rodzica, tworząc niemal identyczny proces potomny. Oba procesy, zarówno rodzic, jak i dziecko wykonują się równolegle, przy czym identyfikator PID (Process ID) jest unikalny dla każdego procesu, a PPID (Parent Process ID) wskazuje na jego proces nadrzędny. Dzięki temu w systemie powstaje logiczne drzewo procesów, którego korzeniem jest proces systemd (PID 1) – najstarszy proces w systemie, pełniący rolę inicjalizatora wszystkich pozostałych. Dziedziczenie obejmuje wiele aspektów środowiska, takich jak deskryptory plików, zmienne środowiskowe, katalog roboczy czy maska uprawnień. W praktyce oznacza to, że dziecko rozpoczyna działanie w niemal identycznych warunkach jak rodzic, ale może natychmiast po utworzeniu wykonać nowe polecenie za pomocą funkcji exec(), zastępując swoją przestrzeń adresową innym programem.

Podczas wykonywania procesów mogą wystąpić dwa szczególne przypadki: proces osierocony oraz proces zombie. Proces osierocony to taki, którego rodzic zakończył działanie, zanim proces potomny zdążył się zakończyć. Aby zapobiec utracie kontroli nad tego typu procesami, system automatycznie przypisuje im nowego rodzica,

proces systemd, który przejmuje ich nadzór i odbiera kody zakończenia po ich zakończeniu. Dzięki temu system utrzymuje spójność tablicy procesów i nie dopuszcza do akumulacji nieusuniętych wpisów. Z kolei proces zombie (ang. defunct process) to proces, który zakończył działanie, ale jego rodzic nie odebrał jeszcze statusu zakończenia za pomocą funkcji wait() lub waitpid(). W efekcie jądro utrzymuje minimalny wpis w tablicy procesów, oznaczony statusem Z, zawierający tylko identyfikator i kod wyjścia. Choć zombie nie zużywają zasobów procesora ani pamięci, ich nadmiar może świadczyć o błędach w logice programów lub skryptów tworzących wiele procesów potomnych. Wykrycie takich procesów możliwe jest przy użyciu poleceń ps -l, top lub htop, gdzie w kolumnie STAT pojawia się litera Z. Osierocone procesy można rozpoznać po tym, że ich PPID = 1, natomiast zombie po tym, że nie mają już aktywnego procesu potomnego. Systemd regularnie czyści osierocone i zombie procesy, ale administrator może ręcznie zdiagnozować ich występowanie, np. poprzez ps -ef | grep defunct.

Zarządzanie procesami w systemie Linux obejmuje ich tworzenie, monitorowanie, wstrzymywanie, wznawianie i kończenie. Narzędzia takie jak ps, top, htop czy pstree umożliwiają obserwację struktury procesów, ich priorytetów i zużycia zasobów. Polecenie pstree prezentuje pełną hierarchię procesów w formie drzewa, co pozwala łatwo zidentyfikować zależności rodzic–potomek. Kontrola nad procesami odbywa się przy użyciu sygnałów systemowych – asynchronicznych komunikatów wysyłanych przez jądro lub użytkownika. Najczęściej stosowane sygnały to SIGTERM (zakończenie procesu w sposób kontrolowany), SIGKILL (natychmiastowe przerwanie bez możliwości obsługi) oraz SIGSTOP i SIGCONT (wstrzymanie i wznowienie procesu). Wysyłanie sygnałów realizuje się poleceniami kill, pkill, killall lub poprzez skróty klawiaturowe, np. Ctrl+C (SIGINT). Dzięki mechanizmowi trap w skryptach powłoki możliwe jest przechwytywanie sygnałów i wykonywanie zdefiniowanych akcji, np. zapisu logu lub czyszczenia zasobów – przed zakończeniem programu. Uruchomienie procesu w tle przy pomocy symbolu & odłącza go od terminala, pozwalając na równoczesne wykonywanie wielu zadań. Takie procesy można później monitorować poleceniami jobs, fg i bg. W przypadku długotrwałych operacji stosuje się również narzędzie nohup, które umożliwia kontynuowanie pracy procesu po zamknięciu terminala.

Demony i usługi systemowe

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Demony (ang. daemons) stanowią szczególną klasę procesów w systemie Linux, działających w tle i niezależnych od interakcji użytkownika. Ich zadaniem jest realizacja ciągłych lub okresowych funkcji systemowych: od obsługi połączeń sieciowych, przez planowanie zadań, po monitorowanie stanu sprzętu i bezpieczeństwa. Typowy demon uruchamia się automatycznie podczas startu systemu i pozostaje aktywny do jego wyłączenia. Proces ten jest odłączony od terminala, nie ma przypisanego interaktywnego wejścia/wyjścia, a jego standardowe komunikaty są kierowane do logów systemowych. Demony rozpoznaje się zwykle po nazwie zakończonej literą „d", np. sshd (Secure Shell Daemon), crond (scheduler zadań) czy systemd-journald (rejestrator zdarzeń). Ich działanie oparte jest na architekturze wieloprocesowej, w której nadrzędny proces zarządza instancjami potomnymi obsługującymi poszczególne żądania. Dzięki temu możliwe jest skalowanie, równoległość i izolacja zadań. Współczesne systemy Linux wykorzystują mechanizmy kontrolne, które monitorują stan demonów i automatycznie restartują je w przypadku awarii, zapewniając wysoką dostępność i odporność systemu.

Współczesnym standardem zarządzania usługami w systemach Linux jest systemd, który zastąpił klasyczny mechanizm inicjalizacji init. Systemd jest pierwszym procesem w systemie (PID 1), pełniącym rolę nadrzędnego menedżera wszystkich procesów i usług. Jego podstawowym zadaniem jest uruchamianie, nadzorowanie oraz utrzymywanie usług w określonych stanach, zgodnie z ich zależnościami i konfiguracją. Każda usługa jest opisana w postaci jednostki konfiguracyjnej (unit) zapisanej w pliku .service. Pliki te znajdują się w katalogach /lib/systemd/system/ lub /etc/systemd/system/ i składają się z trzech głównych sekcji: [Unit] (zawierającej opis, zależności i kolejność uruchamiania), [Service] (definiującej sposób startu procesu, użytkownika, katalog roboczy i polecenia restartu) oraz [Install] (określającej tryb włączenia usługi przy starcie systemu). Dzięki temu administrator ma pełną kontrolę nad cyklem życia usług, może je uruchamiać (systemctl start), zatrzymywać (systemctl stop), restartować (systemctl restart), włączać automatycznie przy starcie (systemctl enable) lub tymczasowo wyłączać (systemctl disable). Status bieżący usługi można sprawdzić poleceniem systemctl status, które prezentuje jej PID, kod zakończenia oraz ostatnie wpisy z dziennika systemowego. Analiza logów w systemd odbywa się za pomocą narzędzia journalctl, które pozwala filtrować komunikaty po jednostkach (journalctl -u nginx), czasie, użytkowniku lub priorytecie.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

systemd pełni rolę nadrzędnego menedżera procesów i usług, integrującego w jednym mechanizmie inicjalizację, nadzór i logowanie. Każda usługa zdefiniowana w postaci jednostki .service może mieć określone zależności względem innych komponentów systemu, np. After=, Before=, Requires= czy Wants=, co pozwala kontrolować kolejność i warunki uruchamiania. Mechanizm ten umożliwia także równoległe startowanie usług, co znacząco skraca czas inicjalizacji systemu w porównaniu z klasycznym init, który działał sekwencyjnie. Systemd nadzoruje procesy potomne i potrafi automatycznie restartować usługę po awarii (Restart=on-failure), a także monitorować jej stan poprzez interfejs D-Bus lub gniazda komunikacyjne (socket activation). Rejestracja zdarzeń odbywa się centralnie poprzez usługę systemd-journald, co umożliwia analizę logów w ujęciu per-usługa lub globalnym (journalctl -u <nazwa_usługi>). W praktyce administracyjnej zarządzanie usługami sprowadza się do operacji wykonywanych przy pomocy poleceń systemctl start|stop|restart|enable|disable|status, które pozwalają sterować ich cyklem życia i konfiguracją automatycznego uruchamiania

Automatyzacja i planowanie zadań (cron, anacron, ansible)

Automatyzacja zadań w systemach operacyjnych pozwala na cykliczne wykonywanie czynności konserwacyjnych, kopii zapasowych, aktualizacji czy monitoringu, bez konieczności interwencji użytkownika. W systemie Linux podstawowymi narzędziami odpowiedzialnymi za planowanie i harmonogramowanie zadań są cron oraz anacron, które współdziałają z systemd i mogą działać równolegle. Ich konfiguracja opiera się na zasadzie deklaratywnego określania czasu uruchamiania zadań — w minutach, godzinach, dniach, miesiącach i dniach tygodnia — co czyni je prostymi, a zarazem niezwykle skutecznymi mechanizmami automatyzacji.

Narzędzie cron (ang. chronos, czas) uruchamia proces crond, który cyklicznie analizuje tablice zadań użytkowników i systemu. Konfiguracja zadań odbywa się poprzez edycję pliku crontab — dla bieżącego użytkownika przy użyciu polecenia crontab -e, a dla systemu w plikach /etc/crontab lub katalogach /etc/cron.*.

Każdy wpis w tym pliku składa się z pięciu pól czasowych oraz polecenia do wykonania:

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

min godz dzień_mies mies dzień_tyg polecenie

Na przykład:

0 */2 * * * /home/student/backup_incremental.sh

oznacza wykonanie skryptu co 2 godziny, o pełnej godzinie, w każdy dzień tygodnia i miesiąca.

Cron umożliwia bardzo elastyczne określanie harmonogramów — można wskazywać pojedyncze wartości (np. 15 10 * * *), zakresy (1-5), listy (1,3,5) lub kroki (*/10), co pozwala precyzyjnie planować zadania.

Wpisy crontaba są odczytywane dynamicznie — po ich zapisaniu demon automatycznie je uwzględnia bez konieczności restartu systemu. Wyniki działania poleceń można przekierować do plików logów lub przesłać pocztą lokalną użytkownika, co umożliwia kontrolę poprawności wykonania zadań. W systemach wieloużytkownikowych możliwe jest również definiowanie zadań globalnych w /etc/crontab lub w katalogach /etc/cron.*, co pozwala administratorowi na centralne zarządzanie cyklicznymi operacjami systemowymi.

Cron działa w sposób ścisły — wykonuje zadanie tylko wtedy, gdy system jest aktywny w zadanym momencie. W praktyce oznacza to, że jeśli komputer był wyłączony w czasie planowanego uruchomienia, zadanie nie zostanie wykonane. Aby rozwiązać ten problem wprowadzono narzędzie anacron, które nadzoruje wykonanie zadań w interwałach dobowych, tygodniowych lub miesięcznych, niezależnie od czasu ostatniego uruchomienia. Dzięki temu po włączeniu systemu zadania pominięte w poprzednich cyklach zostaną wykonane automatycznie. Konfiguracja anacrona odbywa się poprzez plik /etc/anacrontab, który zawiera definicje w formacie:

okres_dni opóźnienie_minuty identyfikator polecenie

Przykładowo:

1 10 backup_incremental /home/student/backup_incremental.sh

uruchomi zadanie raz dziennie, z 10-minutowym opóźnieniem po starcie systemu. W środowiskach serwerowych cron i anacron często działają równolegle — pierwszy

realizuje zadania wymagające precyzyjnych interwałów, drugi zapewnia ich niezawodność w kontekście restartów lub przestojów.

W bardziej złożonych środowiskach, w których zarządzanie wieloma hostami wymaga spójności i powtarzalności, stosuje się narzędzia klasy Configuration Management, takie jak Ansible. Ansible opiera się na architekturze bezagentowej, gdzie komunikacja odbywa się przez SSH, bez instalowania dodatkowego oprogramowania na hostach zdalnych. Konfiguracja jest zapisywana w plikach YAML w formie playbooków, które opisują stan docelowy systemu. Przykładowy fragment:

```
- hosts: localhost
  tasks:
      - name: Uruchom usługę nginx
      service:
      name: nginx
      state: started
      - name: Włącz cron przy starcie
      service:
      name: cron
      enabled: yes
```

Po uruchomieniu komendy ansible-playbook service_playbook.yml narzędzie doprowadza system do opisanego stanu, jeśli usługa działa, pozostaje niezmieniona; jeśli nie, zostaje włączona. Taki sposób pracy określa się mianem idempotentności, co oznacza, że wielokrotne uruchomienie playbooka nie zmienia efektu końcowego. Ansible umożliwia zatem automatyzację zarządzania usługami, aktualizacjami, konfiguracją sieci oraz bezpieczeństwem w sposób deklaratywny, z pełnym zapisem logów i możliwością audytu. Dzięki integracji z systemd, cron i ufw narzędzie to

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

stanowi praktyczne rozszerzenie koncepcji automatyzacji lokalnej w kierunku zarządzania zdalnymi środowiskami serwerowymi.

Podstawowe usługi odpowiedzialne za bezpieczeństwo

Bezpieczeństwo systemu operacyjnego opiera się na zestawie współpracujących mechanizmów, od kontroli dostępu, przez filtrowanie ruchu sieciowego, po analizę logów i izolację aplikacji. Usługi takie jak SSH, UFW, Fail2Ban i AppArmor tworzą wielowarstwową strukturę ochrony, obejmującą zarówno komunikację sieciową, jak i wewnętrzne procesy systemu.

SSH – bezpieczna zdalna administracja

SSH (Secure Shell) jest podstawowym protokołem komunikacji zdalnej w systemach uniksowych, zastępującym niezabezpieczone narzędzia takie jak telnet czy rlogin. Umożliwia szyfrowane połączenia terminalowe, tunelowanie ruchu sieciowego oraz zdalne uruchamianie komend.

Demon sshd, uruchamiany jako usługa systemowa, nasłuchuje domyślnie na porcie 22/TCP i odpowiada za uwierzytelnianie użytkowników przy pomocy haseł lub kluczy kryptograficznych. Konfiguracja serwera znajduje się w pliku /etc/ssh/sshd_config, gdzie można definiować m.in. metody autoryzacji, porty nasłuchu czy listy dozwolonych użytkowników.

Bezpieczne praktyki administracyjne obejmują wyłączenie logowania root'a (PermitRootLogin no), wymuszanie logowania kluczem publicznym oraz ograniczanie dostępu do wybranych adresów IP przy użyciu zapory sieciowej. Logi połączeń i prób uwierzytelnienia można analizować za pomocą:

journalctl -u ssh

sudo less /var/log/auth.log

Dzięki temu SSH stanowi podstawowe narzędzie zarządzania zdalnego, ale także punkt wejścia, który musi być ściśle monitorowany i chroniony.

UFW – prosty firewall systemowy

UFW (Uncomplicated Firewall) to nakładka upraszczająca konfigurację zapory iptables lub nftables. Umożliwia definiowanie reguł dostępu do usług poprzez wskazywanie portów lub nazw usług.

Podstawowy model działania polega na blokowaniu całego ruchu przychodzącego i selektywnym dopuszczaniu tylko pożądanych połączeń. W typowej konfiguracji administrator wykonuje:

sudo ufw enable

sudo ufw allow 22

sudo ufw allow 80

sudo ufw status verbose

Dzięki temu system przyjmuje tylko ruch SSH i HTTP, a wszystkie inne połączenia są odrzucane. Reguły zapory są trwałe i obowiązują po restarcie systemu, co pozwala zapewnić spójny poziom ochrony usług sieciowych.

UFW integruje się z systemd jako usługa (ufw.service) i jest często pierwszą linią obrony serwera przed nieautoryzowanym dostępem. Logi dotyczące zablokowanych połączeń są rejestrowane w /var/log/ufw.log i mogą być analizowane w celu wykrywania prób ataków sieciowych.

Fail2Ban – ochrona przed atakami brute-force

Fail2Ban to usługa monitorująca logi systemowe i reagująca na wielokrotne próby nieudanych logowań. Analizując pliki takie jak /var/log/auth.log, wykrywa charakterystyczne wzorce błędnych logowań i tymczasowo blokuje adresy IP, z których pochodzą ataki typu brute-force.

Działanie usługi opiera się na zestawie tzw. jaili (reguł ochrony), które określają, jakie logi mają być analizowane, jakie wzorce traktować jako atak oraz jak długo utrzymać blokadę. Przykładowa konfiguracja sekcji w pliku /etc/fail2ban/jail.local:

[sshd]

enabled = true

port = ssh

maxretry = 3

bantime = 600

Oznacza, że po trzech nieudanych próbach logowania przez SSH, adres IP zostanie zablokowany na 10 minut.

Fail2Ban wykorzystuje systemowe mechanizmy firewall (np. UFW, iptables), aby egzekwować blokady, i współpracuje z systemd-journald, co umożliwia analizę logów za pomocą:

sudo fail2ban-client status sshd

sudo journalctl -u fail2ban

Dzięki temu stanowi automatyczną warstwę ochrony przed prostymi próbami włamań i skanowania usług.

AppArmor – kontrola dostępu na poziomie jądra

AppArmor jest mechanizmem MAC (Mandatory Access Control), zapewniającym dodatkowy poziom izolacji dla procesów systemowych. Działa na poziomie jądra Linuxa, ograniczając, do jakich plików, katalogów i zasobów może uzyskać dostęp dany program. Mechanizm ten chroni system przed eskalacją uprawnień w przypadku przejęcia procesu, nawet jeśli napastnik uzyska dostęp do usługi, nie może wyjść poza dozwolony zakres działania.

Każdy proces objęty ochroną posiada przypisany profil bezpieczeństwa, zdefiniowany w plikach /etc/apparmor.d/. Przykładowo, profil usr.sbin.nginx określa, do jakich katalogów serwer Nginx może mieć dostęp (np. /var/www/), a jakie operacje są zabronione (np. odczyt /etc/passwd).

AppArmor może działać w trybie enforce (wymuszanie reguł) lub complain (rejestrowanie naruszeń bez blokowania). Status aktywnych profili można sprawdzić poleceniem:

sudo aa-status

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Logi naruszeń można sprawdzić poleceniem:

sudo journalctl -t apparmor

Ćwiczenia samodzielne

1. Uruchom terminal i wyświetl listę procesów poleceniem:

   ps -ef | head

   Zwróć uwagę na kolumny PID, PPID, CMD.

   Polecenie: Znajdź w tabeli proces bash lub zsh (Twoja powłoka) oraz sprawdź jego rodzica (PPID).

   Wskazówka: Użyj ps -f -p <PID> aby wyświetlić szczegóły konkretnego procesu.

2. Napisz prosty skrypt child.sh:

   #!/bin/bash

   echo "Proces potomny działa..."

   sleep 60

   Uruchom go w tle poleceniem:

   bash child.sh &

   Wyświetl jego PID (echo $!), sprawdź jego obecność w tablicy procesów (ps -ef | grep child.sh) i zidentyfikuj jego proces nadrzędny (PPID).

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

3. Uruchom powyższy skrypt, a następnie w innym terminalu zakończ powłokę-rodzica (exit). Sprawdź, czy proces potomny nadal działa i kto stał się jego nowym rodzicem.

   Wskazówka: użyj ps -o ppid= -p <PID> – jeśli PPID=1, proces został przejęty przez systemd (proces osierocony).

4. Utwórz skrypt zombie.sh:

```
#!/bin/bash
if [ "$1" == "child" ]; then
  exit 0
else
  bash $0 child &
  sleep 5
  ps -l | grep Z
fi
```

   Uruchom bash zombie.sh i obserwuj tablicę procesów. Zwróć uwagę na proces w stanie Z. Wskazówka: STAT = Z oznacza zombie, czyli proces zakończony, ale nieodebrany przez rodzica.

5. Zmodyfikuj poprzedni skrypt tak, aby proces potomny żył dłużej niż rodzic:

```
#!/bin/bash
if [ "$1" == "child" ]; then
  sleep 60
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

```
else
  bash $0 child &
  exit 0
fi
```

Uruchom skrypt i po kilku sekundach sprawdź, czy proces potomny istnieje (ps -ef | grep bash) i jaki ma PPID.

Wniosek: Powinien zostać przejęty przez systemd.

6. Uruchom w tle prosty proces:

```
sleep 300 &
```

Zapisz jego PID (echo $!) i wykonaj kolejno:

- kill -STOP <PID> # wstrzymanie procesu
- kill -CONT <PID> # wznowienie
- kill -TERM <PID> # zakończenie

Obserwuj zmiany w kolumnie STAT w top lub ps -l.

Wskazówka: T = zatrzymany, S = uśpiony, brak = aktywny.

7. Utwórz skrypt trap_demo.sh:

```
#!/bin/bash
trap "echo 'Otrzymano SIGINT (Ctrl+C)!'; exit 0" SIGINT
trap "echo 'Otrzymano SIGTERM!'; exit 0" SIGTERM
```


Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

```
echo "PID: $$"
while true; do
  sleep 1
done
```

Uruchom go i spróbuj zakończyć proces przez Ctrl+C oraz kill -TERM <PID>.

Zaobserwuj komunikaty pułapek. Jeśli komunikat nie pojawia się po naciśnięciu Ctrl+C, upewnij się, że skrypt jest uruchomiony w bieżącej sesji, a nie w tle.

Wskazówka: trap pozwala przechwytywać sygnały i reagować na nie własnym kodem.

8. Przeanalizuj drzewo procesów za pomocą polecenia:

```
pstree -p | less
```

Znajdź proces systemd (PID 1) i sprawdź, jakie procesy są jego potomkami. Wskazówka: pstree -up pokaże także użytkowników i PID-y.

Jeśli polecenie pstree nie jest dostępne, można je zainstalować poleceniem:

```
sudo apt install psmisc -y
```

9. Uruchom trzy kopie sleep 100 w tle i sprawdź:

```
sleep 100 &
sleep 100 &
sleep 100 &
```

Wyświetl wszystkie procesy sleep i zakończ je wszystkie jednocześnie poleceniem pkill sleep. Wskazówka: pkill działa na nazwę procesu, nie wymaga PID-ów.

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

10. Utwórz skrypt monitor.sh:

```
#!/bin/bash
PROC="sleep"
while true; do
  COUNT=$(pgrep -c "$PROC")
  echo "$(date): Procesów $PROC = $COUNT"
  sleep 3
done
```

Uruchom kilka procesów sleep i zobacz, jak skrypt reaguje na ich pojawianie się i znikanie. Polecenie: Rozszerz skrypt tak, by automatycznie kończył wszystkie procesy sleep, jeśli liczba przekroczy 3.

11. W tym ćwiczeniu należy skonfigurować dwa kontenery Docker – jeden jako serwer SSH, drugi jako klient. Należy prześledzić, jak działa demon sshd, jak przebiega proces autoryzacji i jak usługa reaguje w logach systemowych.

Instrukcja wykonania:

- W Laboratorium pierwszym zostały przedstawione polecenia niezbędne do zainstalowania dockera.
- Uruchom kontener z serwerem SSH:

  docker run -d --name ssh-server --hostname server -p 2222:22 linuxserver/openssh-server

  Jeśli obraz nie jest dostępny, można zamiast tego uruchomić czysty system i doinstalować serwer SSH:

  docker run -d --name ssh-server debian bash -c "apt update && apt install -y openssh-server && service ssh start && tail -f /dev/null"

- Uruchom drugi kontener, który będzie klientem:

  docker run -it --rm --name ssh-client ubuntu bash

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Zainstaluj w nim klienta SSH:

```
apt update && apt install openssh-client -y
```

- Sprawdź adres IP kontenera serwera:

  docker inspect -f '{{range.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' ssh-server

- Połącz się z serwerem:

  ssh root@<adres_ip_serwera> -p 22

  Przy pierwszym połączeniu zaakceptuj klucz hosta.

- W kontenerze serwera sprawdź logi, by zobaczyć, że połączenie zostało odnotowane:

  ```
  journalctl -u ssh
  ```

12. Jednym z podstawowych zadań administratora jest automatyzacja kopii zapasowych. W tym ćwiczeniu należy wykorzystać skrypt wykonujący przyrostowe kopie katalogu, a następnie zaplanować jego cykliczne uruchamianie przy użyciu usług cron oraz anacron. Pozwoli to zrozumieć różnicę między zadaniami uruchamianymi cyklicznie a tymi wykonywanymi po restarcie.

Instrukcja wykonania:

- Utwórz katalog testowy:

  ```
  mkdir -p ~/projekty/test_backup
  ```

- Utwórz skrypt ~/backup_incremental.sh:

  ```
  #!/bin/bash
  SRC="$HOME/projekty/test_backup"
  DEST="$HOME/backups"
  DATE=$(date +%Y-%m-%d_%H-%M)
  ```


Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

```
mkdir -p "$DEST"
tar --create --gzip --listed-incremental="$DEST/snapshot.snar" \
--file="$DEST/backup_$DATE.tar.gz" "$SRC"
echo "$(date): Wykonano kopię przyrostową." >> "$DEST/backup.log"
```

- Uwaga: Plik snapshot.snar jest automatycznie tworzony przez tar i zawiera informacje o poprzednim stanie kopii. Nie należy go usuwać między kolejnymi uruchomieniami.
- Nadaj skryptowi uprawnienia:

```
chmod +x ~/backup_incremental.sh
```

- Dodaj zadanie do crona poleceniem:

```
crontab -e
```

i dopisz:

```
0 */2 * * * /home/student/backup_incremental.sh
```

(wykona się co 2 godziny).

- Skonfiguruj Anacron, aby uruchamiał skrypt po restarcie:

```
sudo nano /etc/anacrontab
```

Dodaj wpis:

```
1   10  backup_incremental  /home/student/backup_incremental.sh
```

- Sprawdź po kilku cyklach, czy w katalogu ~/backups pojawiły się archiwa i log.

13. Aby zaobserwować działanie cron'a zmień wpis z poprzedniego zadania aby wykonywał kopie zapasową co 3 minuty.
14. Serwery WWW, takie jak Nginx czy Apache, to jedne z najważniejszych usług. Ich działanie oparte jest na demonach obsługujących żądania HTTP. W tym zadaniu należy zainstalować i skonfigurować Nginx, oraz zmienić zawartość strony głównej i przeanalizować logi, aby zrozumieć cykl życia usługi sieciowej.

Instrukcja wykonania:

- Zainstaluj Nginx poleceniem:

  sudo apt install nginx -y

- Sprawdź status usługi. Jeśli usługa jest już uruchomiona, możesz pominąć komendę start.:

  systemctl status nginx

- Uruchom stronę testową w przeglądarce lub terminalu:

  curl http://localhost

- Zmień zawartość strony głównej:

  echo "<h1>Moja własna strona Nginx</h1>" | sudo tee /var/www/html/index.html

- Przeładuj konfigurację i sprawdź efekty:

  sudo systemctl reload nginx

  curl http://localhost

- Wyświetl logi:

  journalctl -u nginx | tail

  cat /var/log/nginx/access.log | tail

15. Wiele usług systemowych w Linuxie działa w tle, zapewniając kluczowe funkcje, jak komunikacja między procesami (DBus), obsługa sieci (NetworkManager) czy logowanie zdarzeń (journald). W tym zadaniu sprawdzisz, jak są uruchamiane i jak można diagnozować ich stan.

Instrukcja wykonania:

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

- Wyświetl listę aktywnych usług:

  systemctl list-units --type=service --state=running

- Znajdź i zapisz krótki opis działania następujących usług:
  A) dbus.service
  B) NetworkManager.service
  C) systemd-journald.service
- Sprawdź logi dla jednej z nich:

  journalctl -u dbus | tail

- Zatrzymaj usługę NetworkManager i zaobserwuj skutki:

  sudo systemctl stop NetworkManager

  ping 8.8.8.8

  Następnie uruchom ją ponownie:

  sudo systemctl start NetworkManager

16. Ansible jest narzędziem do automatyzacji administracji i konfiguracji. Wykorzystuje moduły, które w prosty sposób zarządzają usługami, pakietami i plikami konfiguracyjnymi. W tym ćwiczeniu należy użyć Ansible do włączania i uruchamiania wybranych usług systemowych.

    Instrukcja wykonania:

- Zainstaluj Ansible:

  sudo apt install ansible -y

- Utwórz plik service_playbook.yml:

  - hosts: localhost

    tasks:

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

```
- name: Uruchom usługę nginx
      service:
      name: nginx
      state: started

- name: Upewnij się, że cron włącza się przy starcie
      service:
      name: cron
      enabled: yes
```

- Uruchom playbook:

  ansible-playbook service_playbook.yml

- Obserwuj wynik:

  Jeśli Ansible wykonał zmianę, pojawi się changed.

  Jeśli stan już był poprawny, pojawi się ok.

- Porównaj wynik działania z ręcznym sprawdzeniem usług:

  systemctl status nginx

  systemctl is-enabled cron

- Wyświetl logi Ansible z ostatniego uruchomienia:

  cat /home/$USER/.ansible.log

- Jeśli nie widać logu, Ansible wyświetla wynik na ekranie — każdy blok ok lub changed oznacza wykonanie modułu.
- W pliku YAML dopisz kolejną usługę (np. fail2ban) i ponownie uruchom playbook.

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

- Zwróć uwagę, że Ansible nie zmienia nic, jeśli stan jest już zgodny – to tzw. idempotentność. Ansible działa w sposób deklaratywny – nie wydaje pojedynczych poleceń jak użytkownik, ale „doprowadza system do oczekiwanego stanu”.

17. Fail2Ban chroni system przed próbami włamań, analizując logi usług (np. SSH) i blokując adresy IP, które wykonują zbyt wiele nieudanych logowań. W tym ćwiczeniu należy skonfigurować jego podstawową ochronę, a następnie wywołać sytuację, która spowoduje zadziałanie blokady. Fail2Ban aktywnie reaguje na zdarzenia z logów systemowych, a jego działanie można monitorować i modyfikować dynamicznie.

Instrukcja wykonania:

- Aby wykonać to zadanie uruchom usługę ssh:

  sudo systemctl enable --now ssh

- Zainstaluj usługę fail2ban:

  sudo apt install fail2ban -y

- Skopiuj i aktywuj plik konfiguracyjny:

sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local

sudo nano /etc/fail2ban/jail.local

- Włącz sekcję:

[sshd]

enabled = true

maxretry = 3

bantime = 600

- Uruchom i sprawdź status usługi:

sudo systemctl restart fail2ban

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

sudo systemctl status fail2ban

- Sprawdź, czy ochrona dla SSH działa:

  sudo fail2ban-client status sshd

- Z drugiego terminala (lub innego hosta) spróbuj 3 razy błędnie zalogować się przez SSH. Po kilku próbach, sprawdź:

  sudo fail2ban-client status sshd

- Zweryfikuj, że Twój adres IP został zbanowany (pojawi się w sekcji Banned IP list).
  Dodatkowo możesz podejrzeć logi:

  sudo tail -n 10 /var/log/fail2ban.log

- Odbanuj się:

  sudo fail2ban-client set sshd unbanip 127.0.0.1

18. W tym zadaniu należy włączyć i skonfigurujesz zaporę UFW, otworzyć potrzebne porty (SSH, HTTP), a następnie sprawdzić które porty są blokowane. Firewall nie zatrzymuje działania usług, ale decyduje, kto ma do nich dostęp. To fundamentalne dla zrozumienia różnicy między działającą usługą a usługą dostępną z zewnątrz. Należy mieć świadomość, że bezpieczeństwo sieciowe wymaga zarządzania portami, nie tylko procesami.

    Instrukcja wykonania:

- Zainstaluj i włącz zaporę:

  sudo apt install ufw -y

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

sudo ufw enable

- Sprawdź aktualny stan i domyślną politykę:

  sudo ufw status verbose

- Zezwól na podstawowe porty:

  sudo ufw allow 22

  sudo ufw allow 80

- Uruchom Nginx i upewnij się, że strona działa lokalnie:

  sudo systemctl start nginx

  curl http://localhost

- Następnie z innego hosta (lub innego kontenera) spróbuj połączyć się z Twoim systemem po porcie 80 i 8080:

  curl http://<IP_Hosta>:80

  curl http://<IP_Hosta>:8080

  Powinieneś zauważyć, że połączenie na porcie 8080 jest blokowane.

- Sprawdź logi firewalla:

  sudo less /var/log/ufw.log

- W razie potrzeby zresetuj ustawienia:

  sudo ufw reset

19. W tym ćwiczeniu należy sprawdzić, jak profil AppArmor zabezpiecza usługę i co się dzieje po jego wyłączeniu. AppArmor to nie firewall, lecz system kontroli dostępu na poziomie systemu plików i procesów. Profil bezpieczeństwa określa, co aplikacja może zrobić, dzięki temu nawet jeśli usługa zostanie przejęta, jej skutki będą ograniczone. To kluczowy przykład separacji obowiązków i zasady najmniejszych uprawnień (Least Privilege).

Instrukcja wykonania:

- Sprawdź, czy AppArmor działa:

  sudo systemctl status apparmor

- Sprawdź aktywne profile i znajdź nginx:

  sudo aa-status | grep nginx

- Wyświetl zawartość profilu, aby zobaczyć, jakie katalogi są dozwolone:

  sudo cat /etc/apparmor.d/usr.sbin.nginx | grep -v '^#'

- Tymczasowo wyłącz profil:

  sudo aa-disable /etc/apparmor.d/usr.sbin.nginx

- Uruchom ponownie usługę Nginx i spróbuj uzyskać dostęp do pliku spoza jej katalogu. Zwróć uwagę, że dostęp może być teraz możliwy – co pokazuje, jaką rolę pełnił profil AppArmor.

  sudo systemctl restart nginx

  sudo cat /etc/passwd > /var/www/html/test.html

  curl http://localhost/test.html

- Włącz ponownie profil:

  sudo aa-enforce /etc/apparmor.d/usr.sbin.nginx

- Przeanalizuj logi AppArmor i odnotuj, które operacje były blokowane.

  sudo journalctl -t apparmor

Pytania pomocnicze

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

1. Czym jest proces w systemie Linux i jakie informacje zawiera jego wpis w tablicy procesów?
2. Jakie są różnice między procesem rodzicem, potomnym, osieroconym i zombie?
3. Jakie znaczenie mają identyfikatory PID i PPID?
4. Jak sprawdzić, które procesy aktualnie działają w systemie?
5. Co oznacza kolumna STAT w wynikach poleceń ps lub top?
6. W jaki sposób można wstrzymać, wznowić i zakończyć działanie procesu przy użyciu sygnałów?
7. Jak działa mechanizm pułapek (trap) i w jakich sytuacjach jest używany?
8. Czym różni się proces uruchomiony w tle (&) od procesu działającego na pierwszym planie?
9. Jak można zobaczyć hierarchię procesów w systemie i ustalić, które z nich są potomkami systemd?
10. Jakie polecenia służą do monitorowania procesów w czasie rzeczywistym?
11. Czym jest demon (usługa systemowa) i czym różni się od zwykłego procesu?
12. Co oznacza, że systemd jest procesem PID 1 i dlaczego jest to ważne dla działania systemu?
13. Jak sprawdzić, czy dana usługa jest aktywna, włączona przy starcie lub zatrzymana?
14. Jakie polecenia systemctl służą do:
    a. uruchamiania,
    b. zatrzymywania,
    c. restartowania,
    d. włączania i wyłączania usług przy starcie systemu?
15. Gdzie znajdują się pliki jednostek usługowych (.service) i jakie mają podstawowe sekcje?
16. Jak można stworzyć własną usługę w systemd dla prostego skryptu powłoki?
17. W jaki sposób można przeanalizować logi konkretnej usługi przy użyciu journalctl?
18. Czym różnią się klasyczne polecenia service od systemctl?
19. Jak sprawdzić zależności między usługami (np. które są uruchamiane przed innymi)?
20. Co oznacza status „failed” i jak diagnozować jego przyczynę?

Podsumowanie

W trakcie piątego laboratorium studenci poznali mechanizmy zarządzania procesami i usługami systemowymi w systemie Linux. Zajęcia rozpoczęły się od praktycznej analizy cyklu życia procesów – ich tworzenia, dziedziczenia, stanu oraz sposobów sterowania sygnałami. Omówiono różnice między procesami zwykłymi, osieroconymi

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

i zombie, a także pokazano, jak system przekazuje zarządzanie osieroconymi procesami do systemd (PID 1). Druga część laboratorium skupiła się na nowoczesnym zarządzaniu usługami z wykorzystaniem systemd, w tym na analizie struktur jednostek .service, obsłudze dziennika systemowego oraz diagnostyce błędów uruchamiania usług. Uczestnicy samodzielnie konfigurowali i uruchamiali demony takie jak SSH czy Nginx, uczyli się planować zadania cykliczne (cron, anacron) oraz zabezpieczać system (fail2ban, ufw, apparmor). Podkreślono rolę procesów jako fundamentalnego pojęcia, które łączy poziom programów użytkownika z mechanizmami systemowymi. Wiedza ta stanowi niezbędny wstęp do dalszych tematów związanych z monitorowaniem, optymalizacją i konteneryzacją środowisk serwerowych.

Literatura

1. Mastering Linux Daemons: A Comprehensive Guide, Link: https://linuxvox.com/blog/daemons-linux/, (data ostatniego dostępu 25.09.25)
2. How to Manage Process in Linux, Link: https://www.geeksforgeeks.org/linux-unix/process-management-in-linux/, (data ostatniego dostępu 25.09.25)
3. AppArmor Cheat Sheet for Linux System Administrators, Link: https://computingforgeeks.com/apparmor-cheat-sheet-for-linux-system-administrators/, (data ostatniego dostępu 25.09.25)
4. anacron command in linux with examples, Link: https://www.geeksforgeeks.org/linux-unix/anacron-command-in-linux-with-examples/, (data ostatniego dostępu 25.09.25)

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Materiały do przedmiotu:

Podstawy Systemów Operacyjnych

Laboratorium nr 6:

Zarządzanie oprogramowaniem i menedżerami pakietów

Autor:

Albert Rachwał

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Na szóstych zajęciach studenci poznają jeden z kluczowych mechanizmów systemów Linux – zarządzanie oprogramowaniem. W systemach tych każdy program jest dostarczany w postaci pakietu, a menedżer pakietów odpowiada za jego instalację, konfigurację, aktualizację i usuwanie. Omawiane są różnice pomiędzy popularnymi systemami pakietów, a także struktura repozytoriów, które gromadzą oprogramowanie i aktualizacje bezpieczeństwa. Studenci uczą się, jak wyszukiwać programy, instalować je z oficjalnych repozytoriów oraz jak zarządzać dodatkowymi źródłami oprogramowania. Zajęcia pokazują również alternatywne metody dystrybucji aplikacji, takie jak pakiety niezależne od dystrybucji czy systemy oparte na podejściu deklaratywnym. W części praktycznej uczestnicy samodzielnie instalują wybrane narzędzia, konfigurują środowisko oraz uczą się stosować aktualizacje, co pozwala utrzymać system w bezpiecznym i stabilnym stanie.

Cel ćwiczenia/laboratorium

1. Zrozumienie pojęcia pakietu oprogramowania, jego struktury i roli w systemie Linux.
2. Poznanie architektury i działania systemu zarządzania pakietami (APT, DPKG).
3. Opanowanie podstawowych operacji na pakietach: instalacja, aktualizacja, usuwanie.
4. Umiejętność wyszukiwania i analizy dostępnych pakietów w repozytoriach.
5. Poznanie struktury pliku /etc/apt/sources.list oraz katalogu /etc/apt/sources.list.d/.
6. Zrozumienie mechanizmu aktualizacji systemu oraz poprawek bezpieczeństwa.
7. Umiejętność instalacji pakietów z plików lokalnych (.deb) i rozwiązywania zależności.
8. Rozróżnienie instalacji z repozytorium, ze źródeł i z plików binarnych.
9. Poznanie różnic między menedżerami pakietów w różnych dystrybucjach (APT, YUM/DNF, Pacman).
10. Zrozumienie problemu zależności, konfliktów i mechanizmów ich rozwiązywania.
11. Umiejętność zarządzania dodatkowymi repozytoriami i kluczami GPG.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

12. Zapoznanie się z nowoczesnymi systemami dystrybucji aplikacji (Snap, Flatpak, AppImage).
13. Wprowadzenie do koncepcji deklaratywnego zarządzania pakietami (Nix/NixOS).
14. Poznanie cross-platformowych narzędzi instalacji oprogramowania (Homebrew/Linuxbrew).
15. Uświadomienie znaczenia utrzymania aktualnego i bezpiecznego systemu pakietów.

Instrukcja laboratoryjna/ćwiczeniowa

Wprowadzenie do zarządzania oprogramowaniem w systemach Linux

Zarządzanie oprogramowaniem jest jednym z kluczowych elementów administracji systemami operacyjnymi z rodziny Linux. Każda dystrybucja, dostarcza użytkownikowi zestaw narzędzi pozwalających na instalowanie, aktualizowanie, konfigurowanie oraz usuwanie aplikacji w sposób bezpieczny i zautomatyzowany. W odróżnieniu od systemów, w których oprogramowanie instaluje się ręcznie poprzez uruchamianie instalatorów, Linux korzysta z centralnego modelu zarządzania pakietami. Oprogramowanie jest w nim dystrybuowane w postaci pakietów, które są niewielkimi archiwami zawierającymi skompilowany kod programu, jego zależności, metadane i ewentualne skrypty instalacyjne. Pakiety te są gromadzone w repozytoriach – specjalnych katalogach dostępnych lokalnie lub przez Internet, które stanowią zaufane źródło oprogramowania dla danej dystrybucji.

Rola menedżerów pakietów polega na pośredniczeniu między użytkownikiem a repozytoriami. Narzędzia takie jak APT, DPKG, YUM czy Pacman automatyzują proces instalacji poprzez rozwiązywanie zależności między pakietami, pobieranie odpowiednich wersji z serwerów oraz aktualizację list dostępnych programów. Dzięki temu użytkownik nie musi samodzielnie pobierać plików wykonywalnych ani dbać o zgodność wersji bibliotek, wszystkie te zadania przejmuje na siebie system zarządzania pakietami. To właśnie ta mechanika sprawia, że Linux może być utrzymywany w stanie spójnym i bezpiecznym nawet na tysiącach maszyn jednocześnie.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Podstawowym celem zarządzania oprogramowaniem jest zapewnienie stabilności i bezpieczeństwa systemu. Regularne aktualizacje dostarczane przez repozytoria obejmują zarówno poprawki błędów, jak i łatki bezpieczeństwa. Administratorzy mogą definiować, które pakiety mają być aktualizowane automatycznie, a które pozostają w wersjach stabilnych. Dzięki temu system pozostaje funkcjonalny, a jednocześnie odporny na znane zagrożenia. Równie ważne jest kontrolowanie pochodzenia oprogramowania, tylko pakiety pochodzące z podpisanych repozytoriów są uznawane za zaufane. Weryfikacja podpisów GPG chroni przed instalacją złośliwego kodu i gwarantuje integralność danych.

Współczesne dystrybucje Linuksa kładą też duży nacisk na prostotę obsługi. Polecenia takie jak apt install, apt update czy apt upgrade pozwalają w jednym wierszu terminala zrealizować procesy, które w tle angażują złożone mechanizmy pobierania, rozpakowywania i konfigurowania pakietów. Dla użytkownika końcowego jest to proces przejrzysty i bezpieczny, a dla administratora w pełni kontrolowalny.

Pojęcie pakietu i repozytorium

Każde oprogramowanie instalowane w systemie Linux jest dystrybuowane w postaci pakietu. Pakiet to skompresowane archiwum zawierające wszystkie elementy niezbędne do zainstalowania i uruchomienia programu, zarówno pliki binarne, biblioteki, dokumentację, metadane jak również skrypty pomocnicze, które automatyzują czynności instalacyjne i konfiguracyjne. W przypadku dystrybucji opartych na Debianie, takich jak Ubuntu, formatem tym jest plik .deb. W środowiskach Red Hat, Fedora czy openSUSE stosuje się z kolei format .rpm. Niezależnie od rodzaju systemu, istota pozostaje ta sama: pakiet stanowi jednostkę dystrybucji i kontroli oprogramowania.

Struktura pakietu .deb jest precyzyjnie zdefiniowana. W jego wnętrzu znajdują się co najmniej trzy kluczowe składniki: debian-binary, czyli plik określający wersję formatu pakietu, control.tar.gz zawierający metadane oraz data.tar.gz, w którym umieszczane są właściwe pliki programu. W archiwum control zapisane są informacje takie jak nazwa i wersja pakietu, jego zależności (Depends, Recommends, Conflicts), opis, architektura docelowa oraz skrypty uruchamiane przed i po instalacji (preinst, postinst, prerm, postrm). Dzięki tej strukturze system potrafi w sposób

deterministyczny przeprowadzić proces instalacji, weryfikując zgodność wersji bibliotek i unikając konfliktów między komponentami.

Zależności odgrywają w tym mechanizmie kluczową rolę. Większość aplikacji korzysta z wielu bibliotek współdzielonych, które muszą być obecne w odpowiedniej wersji. System zarządzania pakietami analizuje więc deklaracje w plikach kontrolnych i automatycznie pobiera brakujące komponenty z repozytoriów. Właśnie dzięki temu użytkownik może jednym poleceniem zainstalować złożony pakiet, a menedżer sam zadba o kompletność środowiska. W przypadku, gdy instalacja lokalna (dpkg -i) nie spełnia zależności, mechanizm apt -f install naprawia konfigurację, pobierając brakujące elementy.

Repozytorium to z kolei zorganizowany zbiór pakietów oraz metadanych, stanowiący źródło oprogramowania dla systemu. Każde repozytorium posiada strukturę katalogów odpowiadającą architekturze (amd64, arm64) i gałęzi dystrybucji (stable, testing, security). W głównym katalogu repozytorium znajdują się pliki indeksujące, w tym Packages.gz lub Packages.xz, które zawierają listę dostępnych pakietów wraz z ich kontrolnymi sumami SHA256, rozmiarami i ścieżkami do plików .deb. Dodatkowo obecne są pliki Release oraz InRelease, które określają metadane repozytorium, jego wersję, podpis GPG i informacje o integralności. Dzięki temu system może zweryfikować autentyczność źródła oraz uniknąć instalacji pakietów pochodzących z niezaufanych lokalizacji.

Repozytoria dzielą się na różne typy w zależności od ich pochodzenia i przeznaczenia. Repozytoria oficjalne są utrzymywane przez zespół danej dystrybucji i stanowią podstawowe źródło oprogramowania. Repozytoria dodatkowe, takie jak PPA (Personal Package Archives) w systemach Ubuntu, pozwalają na dystrybucję oprogramowania przez niezależnych twórców. Z kolei repozytoria lokalne wykorzystywane są w środowiskach zamkniętych, administrator może stworzyć własne źródło pakietów, generując plik indeksowy za pomocą narzędzia dpkg-scanpackages i udostępniając go przez protokół file:// lub http://.

W praktyce repozytoria są sercem systemu dystrybucji oprogramowania. To one decydują o dostępnych wersjach pakietów, cyklu aktualizacji i poziomie bezpieczeństwa. Dzięki centralizacji tego mechanizmu, administrator może w prosty

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

sposób utrzymywać spójność środowisk wielu komputerów, wystarczy, że wszystkie korzystają z tego samego zestawu repozytoriów i kluczy GPG. Każdy pakiet zainstalowany z repozytorium może być następnie weryfikowany, aktualizowany lub usuwany bez ingerencji w pozostałe elementy systemu, co zapewnia stabilność i powtarzalność konfiguracji.

Klasyczne menedżery pakietów

Systemy z rodziny Debian i Ubuntu, a także liczne ich pochodne, wykorzystują dwa ściśle współpracujące ze sobą narzędzia do zarządzania oprogramowaniem, DPKG oraz APT. Oba pełnią odmienne, lecz komplementarne role w hierarchii zarządzania pakietami: DPKG działa na najniższym poziomie i odpowiada bezpośrednio za instalację, konfigurację i usuwanie poszczególnych pakietów .deb, natomiast APT (Advanced Package Tool) stanowi warstwę wyższego poziomu, która pośredniczy między użytkownikiem a repozytoriami, automatyzując proces pobierania, aktualizacji i rozwiązywania zależności.

Podstawową jednostką pracy narzędzia dpkg jest pojedynczy plik pakietu. Administrator może przy jego użyciu zainstalować program lokalnie, bez udziału zewnętrznych źródeł, poleceniem dpkg -i nazwa_pakietu.deb. Polecenie to rozpakowuje archiwum .deb, umieszcza pliki w odpowiednich lokalizacjach systemowych oraz wykonuje skrypty instalacyjne z katalogu DEBIAN/. W przeciwieństwie do APT, dpkg nie potrafi samodzielnie rozwiązywać zależności – jeśli w systemie brakuje wymaganych bibliotek, instalacja zakończy się błędem. Wówczas niezbędne jest uruchomienie polecenia apt -f install, które automatycznie pobierze i uzupełni brakujące pakiety. Narzędzie dpkg posiada też funkcje diagnostyczne, umożliwiające np. wyświetlenie listy wszystkich zainstalowanych pakietów (dpkg -l), identyfikację pakietu, do którego należy dany plik (dpkg -S /ścieżka/pliku), czy całkowite usunięcie pakietu wraz z jego konfiguracją (dpkg -P).

Z kolei APT rozszerza możliwości DPKG o inteligentne zarządzanie repozytoriami i zależnościami. Działa w oparciu o listę źródeł zdefiniowaną w plikach /etc/apt/sources.list oraz w katalogu /etc/apt/sources.list.d/. Każdy wpis w tych plikach określa lokalizację repozytorium, gałąź systemu (np. stable, bookworm, focal) oraz komponenty (main, universe, security). Gdy użytkownik uruchamia polecenie apt update, APT pobiera z każdego repozytorium aktualne pliki indeksowe

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

(Packages.gz, Release, InRelease), a następnie tworzy lokalną bazę danych dostępnych pakietów w katalogu /var/lib/apt/lists/. Na tej podstawie możliwe jest późniejsze wyszukiwanie, porównywanie wersji i instalacja oprogramowania.

Jedną z największych zalet APT jest jego mechanizm rozwiązywania zależności. W momencie instalacji pakietu (apt install), narzędzie analizuje jego metadane i automatycznie dobiera wszystkie wymagane biblioteki i komponenty. Dzięki temu użytkownik może w prosty sposób zainstalować złożony program, który w rzeczywistości składa się z wielu współzależnych elementów. Mechanizm ten pozwala także unikać konfliktów wersji i tzw. „broken packages", które w systemach bez takiej kontroli prowadziłyby do niestabilności.

APT udostępnia szereg innych użytecznych poleceń, z których najczęściej stosowane to:

- apt upgrade – aktualizacja wszystkich pakietów do nowszych wersji bez zmiany zależności,
- apt full-upgrade – pełna aktualizacja systemu z możliwością usunięcia pakietów kolidujących,
- apt remove i apt purge – odpowiednio usunięcie programu z zachowaniem lub usunięciem plików konfiguracyjnych,
- apt autoremove – automatyczne czyszczenie nieużywanych zależności,
- apt clean i apt autoclean – czyszczenie pamięci podręcznej pobranych archiwów .deb w katalogu /var/cache/apt/archives/.

Warstwa APT obejmuje też zarządzanie kluczami GPG, które służą do weryfikacji autentyczności repozytoriów. Każde repozytorium posiada własny klucz publiczny, a podpisane nim pliki Release są sprawdzane przy każdej aktualizacji. Mechanizm ten gwarantuje integralność oprogramowania i zapobiega instalacji pakietów pochodzących z nieautoryzowanych źródeł. Dla repozytoriów zewnętrznych (np. PPA) klucze można dodawać narzędziem add-apt-repository, które automatycznie umieszcza nowy wpis w katalogu /etc/apt/sources.list.d/ oraz pobiera odpowiedni klucz GPG.

Ważnym elementem architektury APT są także katalogi konfiguracyjne i bazy danych systemu pakietów. W katalogu /etc/apt/ znajdują się pliki konfiguracyjne i źródła repozytoriów, /var/lib/apt/ zawiera metadane o stanie pakietów i listach dostępnych

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

wersji, natomiast /var/cache/apt/ gromadzi pobrane archiwa .deb, które mogą być ponownie wykorzystane w przypadku reinstalacji. Cały system tworzy zatem spójną strukturę, umożliwiającą zarówno automatyzację, jak i pełną kontrolę nad procesem instalacji i aktualizacji.

Menedżery alternatywne oraz instalacja pakietów z plików lokalnych

Choć systemy oparte na Debianie i Ubuntu dominują w wielu środowiskach, świat Linuksa jest zróżnicowany i obejmuje liczne dystrybucje, które wypracowały własne systemy zarządzania pakietami. Pomimo różnic w składni i formacie pakietów, ich cel pozostaje ten sam: zapewnienie prostego, spójnego i bezpiecznego sposobu instalacji oprogramowania oraz utrzymania systemu w stanie aktualnym. Wśród najważniejszych alternatywnych menedżerów pakietów warto wymienić YUM, DNF oraz Pacman, czyli narzędzia stosowane odpowiednio w rodzinach Red Hat, Fedora i Arch Linux. Każde z nich reprezentuje inne podejście do tego samego problemu, wynikające z filozofii danej dystrybucji.

W ekosystemie Red Hat i jego pochodnych (np. CentOS, Fedora) podstawowym formatem pakietu jest RPM (Red Hat Package Manager). Zawiera on dane binarne, pliki konfiguracyjne, metadane oraz informacje o zależnościach w postaci specyfikacji .spec. Zarządzanie tymi pakietami odbywa się za pomocą menedżera YUM (Yellowdog Updater, Modified), który przez wiele lat stanowił standardowe narzędzie w tych systemach. YUM automatycznie rozwiązywał zależności, pobierał aktualizacje i oferował przyjazną składnię podobną do APT. W nowszych wersjach Fedorę i Red Hata zastąpił jego następca DNF (Dandified YUM), przepisany w języku Python z ulepszonym silnikiem rozwiązywania zależności i bardziej przejrzystym systemem wtyczek. DNF korzysta z bibliotek libsolv i libdnf, co zwiększa szybkość działania i precyzję analizy konfliktów między pakietami. Typowe polecenia, takie jak dnf install, dnf update czy dnf remove, pełnią analogiczne funkcje do swoich odpowiedników w APT, co czyni zarządzanie oprogramowaniem w różnych dystrybucjach stosunkowo intuicyjnym dla doświadczonego użytkownika.

Inne podejście reprezentuje Pacman, charakterystyczny dla Arch Linuksa i jego pochodnych (np. Manjaro). Ten menedżer pakietów łączy w sobie prostotę i wysoką wydajność. Pacman operuje na pakietach w formacie .pkg.tar.zst, które są lekkimi archiwami z metadanymi w formacie prostych plików tekstowych. W odróżnieniu od

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

APT czy DNF, Pacman nie rozdziela warstwy niskiego i wysokiego poziomu – jest jedynym narzędziem odpowiedzialnym za całość procesu instalacji, aktualizacji i usuwania oprogramowania. Cały system repozytoriów Arch Linuksa opiera się na zasadzie rolling release, co oznacza, że użytkownik nie przeprowadza aktualizacji systemu do nowych wersji dystrybucji, lecz na bieżąco pobiera najnowsze pakiety. Pacman wyróżnia się minimalistyczną składnią (pacman -S, pacman -R, pacman -Sy), a jego projekt kładzie nacisk na transparentność i pełną kontrolę użytkownika. W połączeniu z menedżerem AUR (Arch User Repository) stanowi niezwykle elastyczne środowisko pracy dla zaawansowanych użytkowników.

Mimo różnic między APT, DNF czy Pacmanem, wszystkie te narzędzia działają w oparciu o ten sam schemat: zarządzają pakietami binarnymi, korzystają z repozytoriów zawierających indeksy metadanych i zapewniają spójność systemu poprzez kontrolę zależności. W praktyce oznacza to, że niezależnie od używanej dystrybucji, proces instalacji pakietu sprowadza się do wywołania jednego polecenia, które automatycznie pobiera wszystkie potrzebne komponenty i integruje je z systemem.

Szczególnym przypadkiem instalacji jest sytuacja, w której pakiet nie pochodzi z repozytorium, lecz został pobrany ręcznie w postaci pliku .deb lub .rpm. W dystrybucjach debianowych do tego celu wykorzystuje się polecenie dpkg -i nazwa_pakietu.deb, które instaluje pakiet lokalnie, bez odwoływania się do repozytoriów. Jeśli podczas instalacji wystąpią braki w zależnościach, można je automatycznie naprawić przy użyciu apt -f install. Analogicznie, w systemach opartych na Red Hat instalacja lokalna odbywa się poleceniem rpm -i, a ewentualne problemy rozwiązuje się za pomocą dnf install --allowerasing. Instalacja z plików lokalnych jest szczególnie przydatna w środowiskach zamkniętych lub odizolowanych od Internetu, gdzie administrator posiada dostęp jedynie do wybranych pakietów pobranych wcześniej.

W kontekście pracy laboratoryjnej warto również wspomnieć o możliwości tworzenia lokalnych repozytoriów, które pełnią funkcję prywatnych źródeł pakietów. W przypadku Debiana można to zrealizować, kopiując kilka plików .deb do własnego katalogu (np. ~/repo), a następnie wygenerować indeks przy użyciu narzędzia dpkg-scanpackages > Packages. Tak przygotowany katalog można dodać do

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

systemu jako repozytorium lokalne poprzez wpis w pliku /etc/apt/sources.list z adresem file:///home/użytkownik/repo/. Dzięki temu można testować lub dystrybuować własne pakiety w sposób zgodny z mechanizmami APT, bez potrzeby ręcznej instalacji.

Aktualizacje i bezpieczeństwo

Proces aktualizacji oprogramowania stanowi jeden z kluczowych elementów utrzymania bezpieczeństwa i stabilności systemów operacyjnych z rodziny Linux. System zarządzania pakietami nie tylko umożliwia instalację nowych programów, lecz także odpowiada za bieżące utrzymywanie ich w wersjach zgodnych z najnowszymi poprawkami błędów i luk bezpieczeństwa. W przeciwieństwie do systemów, w których użytkownik samodzielnie wyszukuje aktualizacje, Linux opiera się na centralnym modelu repozytoriów, dzięki czemu aktualizacje są dystrybuowane w sposób kontrolowany, podpisany kryptograficznie i automatycznie powiązany z wersją systemu.

Narzędzie APT oferuje różne poziomy modernizacji pakietów. Polecenie apt update synchronizuje lokalne listy dostępnych pakietów z repozytoriami, pobierając aktualne indeksy (Packages.gz, Release, InRelease). Następnie komenda apt upgrade instaluje nowsze wersje pakietów już obecnych w systemie, zachowując jednak dotychczasowe zależności, oznacza to, że żaden pakiet nie zostanie usunięty ani zastąpiony, jeśli wymagałoby to zmiany struktury zależności. To rozwiązanie zapewnia wysoką stabilność, dlatego jest domyślnie stosowane w systemach produkcyjnych.

W przypadkach, gdy konieczne jest przeprowadzenie głębszej modernizacji, używa się polecenia apt full-upgrade (dawniej dist-upgrade). Ta forma aktualizacji pozwala na usunięcie lub zastąpienie pakietów, których wersje kolidują z nowymi zależnościami. Jest stosowana między innymi przy przejściu między kolejnymi wydaniami dystrybucji, gdzie aktualizacje obejmują kluczowe komponenty systemu, takie jak jądro, biblioteki systemowe czy środowisko graficzne. W celu zminimalizowania ryzyka, APT analizuje całe drzewo zależności, proponując użytkownikowi dokładny zestaw zmian przed ich zatwierdzeniem.

Aktualizacje w systemach Linux można podzielić na trzy podstawowe kategorie: funkcjonalne, poprawkowe oraz bezpieczeństwa. Aktualizacje funkcjonalne

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

wprowadzają nowe wersje programów lub rozszerzenia istniejących funkcji, natomiast poprawkowe eliminują błędy i zwiększają stabilność. Najistotniejsze z punktu widzenia bezpieczeństwa są aktualizacje typu security updates, dostarczane przez specjalne repozytoria. W systemach Debian i Ubuntu znajdują się one zwykle pod adresem security.debian.org lub security.ubuntu.com. Zawierają one jedynie minimalne zmiany kodu, łatkiy eliminujące konkretne luki bezpieczeństwa, bez wprowadzania nowych funkcjonalności, co gwarantuje ich wysoką kompatybilność z istniejącym systemem. Dzięki temu administrator może regularnie aktualizować system bez obawy o destabilizację środowiska produkcyjnego.

W celu zapewnienia ciągłości ochrony systemu administratorzy często korzystają z mechanizmu automatycznych aktualizacji. W Debianie i Ubuntu realizowane jest to za pomocą pakietu unattended-upgrades. Po jego aktywacji system samodzielnie sprawdza dostępność poprawek bezpieczeństwa, pobiera je w tle i instaluje zgodnie z wcześniej zdefiniowaną polityką. Taki model sprawdza się szczególnie w środowiskach serwerowych i systemach, które muszą działać bez przerw, minimalizując ryzyko eksploatacji niezałatanych podatności. Administrator ma jednocześnie możliwość definiowania, które repozytoria mają być objęte automatycznymi aktualizacjami (np. tylko „security” lub również „updates”).

Bezpieczeństwo procesu aktualizacji jest dodatkowo wzmacniane poprzez mechanizmy kryptograficzne. Każdy pakiet znajdujący się w repozytorium jest podpisany cyfrowo kluczem GPG, a jego integralność jest weryfikowana przez APT przy każdej instalacji lub aktualizacji. Jeśli podpis się nie zgadza lub klucz nie jest zaufany, system odrzuca pakiet. Dzięki temu złośliwe oprogramowanie nie może zostać wprowadzone do systemu bez odpowiedniego podpisu. W plikach Release i InRelease repozytoriów przechowywane są sumy kontrolne SHA256 wszystkich pakietów oraz podpis cyfrowy całego indeksu, co stanowi podstawę zaufanego łańcucha dystrybucji.

W praktyce utrzymanie bezpieczeństwa systemu wymaga również odpowiedniego zarządzania aktualizacjami. Zbyt częste instalowanie eksperymentalnych wersji pakietów może prowadzić do niestabilności, podczas gdy zaniedbanie aktualizacji otwiera drogę do wykorzystania znanych podatności. Z tego względu administratorzy stosują zasadę równowagi, aktualizacje krytyczne są wdrażane niezwłocznie,

natomiast aktualizacje funkcjonalne często trafiają najpierw do środowisk testowych. W dużych organizacjach proces ten realizowany jest za pomocą wewnętrznych repozytoriów i narzędzi do zarządzania konfiguracją (np. Ansible, Puppet), które umożliwiają kontrolowane wdrażanie poprawek na wielu maszynach jednocześnie.

Nowoczesne systemy dystrybucji aplikacji

Wraz z rozwojem technologii i rosnącą różnorodnością środowisk systemowych pojawiła się potrzeba stworzenia bardziej uniwersalnych metod dystrybucji oprogramowania, które uniezależniają aplikacje od specyfiki danej dystrybucji Linuksa. Klasyczne menedżery pakietów, takie jak APT, DNF czy Pacman, wymagają utrzymywania dedykowanych repozytoriów i pakietów skompilowanych dla określonej wersji systemu, co komplikuje proces dystrybucji w środowisku wieloplatformowym. W odpowiedzi na te ograniczenia powstały nowoczesne systemy instalacji aplikacji – Snap, Flatpak i AppImage – które opierają się na idei konteneryzacji, izolacji środowisk uruchomieniowych oraz przenośności. Uzupełniają je rozwiązania deklaratywne, takie jak Nix czy Homebrew, które redefiniują sposób budowania i zarządzania pakietami na różnych systemach operacyjnych.

Jednym z najbardziej rozpowszechnionych współczesnych systemów dystrybucji aplikacji jest Snap, opracowany przez firmę Canonical — twórcę systemu Ubuntu. Snap wykorzystuje konteneryzację, uruchamiając każdą aplikację w izolowanym środowisku zwanym sandboxem, które posiada własny zestaw bibliotek i zależności. Dzięki temu użytkownik może zainstalować nowszą wersję programu bez ryzyka kolizji z bibliotekami systemowymi. Pakiety Snap mają rozszerzenie .snap i są dystrybuowane za pośrednictwem centralnego repozytorium Snap Store. Ich aktualizacja odbywa się automatycznie, co gwarantuje, że użytkownik zawsze korzysta z najnowszej, bezpiecznej wersji oprogramowania. Mechanizm ten jest szczególnie przydatny w środowiskach desktopowych i serwerowych, gdzie stabilność i bezpieczeństwo mają znaczenie nadrzędne. Wadą tego rozwiązania jest natomiast zwiększone zużycie przestrzeni dyskowej, ponieważ każda aplikacja zawiera własny zestaw bibliotek, oraz dłuższy czas uruchamiania, wynikający z warstwy izolacji.

Drugim z nowoczesnych rozwiązań jest Flatpak, rozwijany przez społeczność GNOME i Fundację Freedesktop. Podobnie jak Snap, Flatpak izoluje aplikacje w

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

środowiskach kontenerowych, jednak wprowadza bardziej otwarty i zdecentralizowany model dystrybucji. Flatpaki instalowane są z repozytoriów zwanych remote, z których najpopularniejszym jest Flathub – ogólnodostępne źródło oprogramowania, niezależne od konkretnej dystrybucji. Flatpak wykorzystuje warstwy współdzielonych bibliotek systemowych zwanych runtimes, co ogranicza duplikację danych i zmniejsza rozmiar instalacji w porównaniu do Snapów. Aplikacje uruchamiane w tym modelu są odseparowane od systemu macierzystego, ale mają możliwość komunikacji z nim poprzez kontrolowane interfejsy, takie jak portal Flatpak. Dzięki temu użytkownik może precyzyjnie określać, do jakich zasobów – np. plików, urządzeń czy sieci – aplikacja ma mieć dostęp. Flatpak zyskał dużą popularność w środowiskach graficznych GNOME i KDE, a jego otwarty charakter sprawia, że coraz więcej dystrybucji włącza go do swoich domyślnych repozytoriów.

Trzecim rozwiązaniem z tej grupy jest AppImage, które wyróżnia się brakiem potrzeby instalacji. Aplikacja w formacie .AppImage jest samowystarczalnym plikiem wykonywalnym, zawierającym wszystkie niezbędne biblioteki. Użytkownik może ją pobrać i uruchomić bez żadnej ingerencji w system – nie wymaga ona uprawnień administratora, a jej usunięcie sprowadza się do skasowania pliku. Taki model idealnie wpisuje się w filozofię przenośności i prostoty, szczególnie w zastosowaniach testowych, edukacyjnych czy przenośnych środowiskach roboczych. Wadą AppImage jest brak centralnego zarządzania aktualizacjami oraz trudniejsza integracja z systemem (np. brak automatycznych skrótów w menu), co ogranicza jego zastosowanie w środowiskach korporacyjnych.

Odrębną kategorię nowoczesnych narzędzi stanowią Nix i Homebrew, które reprezentują podejście deklaratywne i wieloplatformowe. Nix wprowadza koncepcję niezmiennych środowisk (immutable environments), w których każda wersja pakietu instalowana jest w osobnym katalogu na podstawie dokładnej specyfikacji zależności. Dzięki temu możliwe jest współistnienie wielu wersji tego samego programu oraz natychmiastowy rollback do poprzedniego stanu systemu. Nix wykorzystuje własny język deklaratywny, który pozwala precyzyjnie definiować środowisko systemowe i użytkowe. Jest to rozwiązanie niezwykle cenione w środowiskach programistycznych, gdzie reprodukowalność konfiguracji ma kluczowe znaczenie. Z kolei Homebrew (lub jego wariant dla Linuksa — Linuxbrew) wywodzi się z systemu macOS i stanowi prosty, skryptowy menedżer pakietów działający w

przestrzeni użytkownika. Pozwala on instalować oprogramowanie w katalogu domowym bez uprawnień administratora, co czyni go popularnym narzędziem wśród deweloperów pracujących na systemach, do których nie mają pełnego dostępu administracyjnego.

Ćwiczenia samodzielne

1. Wyświetl wersję menedżera APT i DPKG zainstalowanego w Twoim systemie. Sprawdź, w jakim katalogu znajduje się ich plik wykonywalny.
2. Zaktualizuj listę dostępnych pakietów i sprawdź, czy system posiada pakiety oczekujące na aktualizację.
3. Wyświetl listę wszystkich zainstalowanych pakietów w systemie i zapisz wynik do pliku tekstowego w katalogu domowym.
4. Wyszukaj pakiet zawierający edytor tekstowy (np. nano, vim lub micro) i zainstaluj go przy użyciu polecenia apt install.
5. Wyświetl szczegółowe informacje o zainstalowanym pakiecie (apt show), a następnie znajdź jego plik konfiguracyjny w katalogu /etc/.
6. Odinstaluj wskazany pakiet, zachowując jego pliki konfiguracyjne (apt remove). Następnie usuń go całkowicie wraz z konfiguracją (apt purge).
7. Wyszukaj wszystkie pakiety, których nazwa zawiera ciąg znaków python3, i zapisz ich listę do pliku python3_list.txt.
8. Zainstaluj pakiet z pliku .deb, który pobierzesz ręcznie z oficjalnego repozytorium Debiana lub Ubuntu. Po instalacji sprawdź poprawność zależności przy użyciu apt -f install.
9. Wyświetl listę wszystkich repozytoriów zdefiniowanych w systemie (z pliku /etc/apt/sources.list oraz katalogu /etc/apt/sources.list.d/).
10. Dodaj nowe repozytorium PPA (np. ppa:deadsnakes/ppa) i zainstaluj z niego wybrany pakiet. Po zakończeniu usuń repozytorium i odśwież listę pakietów.
11. Zidentyfikuj, który pakiet jest właścicielem wybranego pliku binarnego (np. /usr/bin/ls).
12. Wyczyść pamięć podręczną APT (apt clean) i sprawdź, ile miejsca zostało zwolnione w systemie.
13. Zainstaluj dowolną aplikację z wykorzystaniem Flatpak, np. flatpak install flathub org.gnome.Calculator. Następnie uruchom ją i zwróć uwagę, gdzie jest zainstalowana.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

14. (Dla chętnych) Porównaj sposób instalacji tego samego programu w APT i Flatpak – zwróć uwagę na różnice w rozmiarze, izolacji, lokalizacji plików i wersji.
15. (Dla chętnych) Utwórz lokalne repozytorium APT:
    1. utwórz katalog np. ~/repo,
    2. skopiuj do niego kilka plików .deb,
    3. wygeneruj indeks za pomocą dpkg-scanpackages,
    4. dodaj repozytorium do systemu przez file://,
    5. przetestuj instalację pakietu z lokalnego źródła.

Pytania pomocnicze

1. Czym różni się pakiet od repozytorium w systemie Linux?
2. Jakie są trzy główne menedżery pakietów w popularnych dystrybucjach Linuxa?
3. Jakie polecenie w apt służy do instalacji nowego pakietu?
4. W jaki sposób można usunąć pakiet z systemu przy użyciu dnf lub yum?
5. Jak zaktualizować wszystkie pakiety w systemie za pomocą apt?
6. Czym różni się apt update od apt upgrade?
7. Jak wyszukać dostępne pakiety w repozytorium? Podaj przykład polecenia.
8. W jakim pliku konfiguracyjnym w Debianie/Ubuntu przechowywane są adresy repozytoriów?
9. Do czego służy polecenie apt policy?
10. Jak zainstalować pakiet z pliku .deb w Ubuntu/Debianie?
11. Jakie są różnice między klasycznymi pakietami a aplikacjami instalowanymi przez snap lub flatpak?
12. Jak rozwiązuje się problem brakujących zależności podczas instalacji pakietu?
13. Czym wyróżnia się NixOS w podejściu do zarządzania pakietami?
14. Jak działa Homebrew (Linuxbrew) i w jakich przypadkach jest przydatny?
15. Dlaczego regularne aktualizacje pakietów bezpieczeństwa są tak istotne?
16. Dlaczego centralne zarządzanie pakietami zwiększa bezpieczeństwo systemu?
17. Jakie ryzyka niesie instalacja pakietów spoza oficjalnych repozytoriów i jak można je minimalizować?

Podsumowanie

Podczas laboratorium studenci poznali zasady zarządzania oprogramowaniem w systemach Linux oraz praktyczne zastosowanie menedżerów pakietów w różnych

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

dystrybucjach. Ćwiczenia obejmowały instalację, aktualizację i usuwanie pakietów, analizę repozytoriów, weryfikację podpisów GPG oraz rozwiązywanie zależności między komponentami.

Zajęcia pokazały, że menedżer pakietów jest nie tylko narzędziem instalacyjnym, ale także kluczowym elementem utrzymania bezpieczeństwa i stabilności systemu. Mechanizmy takie jak podpisy kryptograficzne, automatyczne aktualizacje i kontrola repozytoriów pozwalają zachować integralność środowiska.

Studenci zapoznali się również z nowoczesnymi systemami dystrybucji aplikacji – Snap, Flatpak i Nix – które opierają się na izolacji i przenośności. Pozwala to lepiej zrozumieć kierunek rozwoju współczesnych ekosystemów Linuksa.

Po zakończeniu laboratorium uczestnicy powinni potrafić samodzielnie zarządzać cyklem życia pakietów, analizować konfigurację źródeł oprogramowania oraz stosować dobre praktyki utrzymania i aktualizacji systemu.

Literatura

1. Mastering Linux Repositories: A Comprehensive Guide, Link: https://linuxvox.com/blog/linux-repositories/, (data ostatniego dostępu 25.09.25)
2. Understanding Package Managers and systemctl, LInk: https://www.geeksforgeeks.org/linux-unix/understanding-package-managers-and-systemctl/, (data ostatniego dostępu 25.09.25)
3. AppImage vs Snap vs Flatpak Linux Package Formats Compared, Link: https://itsfoss.gitlab.io/post/appimage-vs-snap-vs-flatpak-linux-package-formats-compared/, (data ostatniego dostępu 25.09.25)

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Materiały do przedmiotu:

Podstawy Systemów Operacyjnych

Laboratorium nr 7:

Kompilacja oprogramowania dostarczanego w postaci źródłowej

Autor:

Albert Rachwał

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Siódme laboratorium przenosi studentów na poziom bliżej sprzętu i systemu – do procesu kompilacji oprogramowania. Celem jest zrozumienie, jak z kodu źródłowego powstaje działający program. Studenci poznają kolejne etapy kompilacji, od działania preprocesora, przez tłumaczenie na kod maszynowy, aż po linkowanie. Wprowadzane są narzędzia wspierające ten proces, zarówno tradycyjne, jak i nowoczesne. Uczestnicy uczą się pobierać źródła programów, analizować zależności, korzystać z plików konfiguracyjnych kompilacji i instalować własnoręcznie zbudowane aplikacje. Zajęcia pokazują także różnicę pomiędzy oprogramowaniem instalowanym z gotowych pakietów a tym kompilowanym lokalnie, zwracając uwagę na kontrolę, jaką daje to administratorowi oraz programiście. Praktyczne ćwiczenia obejmują tworzenie i kompilowanie prostych programów oraz wykorzystanie narzędzi do zarządzania procesem budowy większych projektów.

Cel ćwiczenia/laboratorium

- Zrozumienie procesu kompilacji w językach systemowych (C/C++ i Rust).
- Poznanie etapów kompilacji: preprocesor, kompilacja, linkowanie.
- Umiejętność kompilacji prostego programu w C/C++ przy użyciu gcc/g++.
- Poznanie narzędzia make i zasad działania plików Makefile.
- Umiejętność budowania projektów C++ z użyciem cmake.
- Pobieranie kodu źródłowego z internetu (wget, git clone).
- Instalacja programów skompilowanych ze źródeł w systemie (/usr/local/bin).
- Rozróżnianie instalacji binarnej (z pakietu) od kompilacji ze źródeł.
- Poznanie podstaw narzędzia Cargo w języku Rust.
- Kompilacja prostego programu Rust z użyciem cargo build.
- Umiejętność uruchamiania testów jednostkowych w Rust (cargo test).
- Porównanie procesu budowy w C++ (Make, CMake) i Rust (Cargo).
- Poznanie repozytorium crates.io i instalacji bibliotek Rust.
- Zrozumienie różnic między zarządzaniem zależnościami w C++ (Conan) a w Rust (Cargo).
- Uświadomienie roli języków systemowych (C, C++, Rust) w rozwoju systemów operacyjnych.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Definicja i cele kompilacji

Proces kompilacji to mechanizm tłumaczenia kodu źródłowego napisanego w języku wysokiego poziomu (np. C, C++, Rust) na kod maszynowy (bajty) zrozumiały dla procesora. W praktyce kompilacja nie jest jednym krokiem. Składa się ona z kilku etapów: preprocessing, kompilacji, asemblera oraz linkowania. W pierwszym etapie preprocesor rozwija dyrektywy takie jak #include, #define i warunkowe #ifdef, łącząc kod nagłówkowy z głównym plikiem i rozszerzając makra. Następnie etap kompilacji analizuje już przetworzony kod, sprawdza poprawność składni, optymalizacje i przekształca go w kod asemblera. Asembler (lub moduł backendowy) zamienia kod asemblerowy w plik obiektowy (.o) zawierający kod maszynowy, ale jeszcze niezależny, może zawierać niezdefiniowane symbole (np. funkcje zewnętrzne). Ostatecznie linker łączy wszystkie pliki obiektowe oraz biblioteki (statyczne lub dynamiczne), rozwiązuje odniesienia do symboli, i generuje finalną postać programu jako plik wykonywalny lub bibliotekę dzieloną (ELF, PE itp.).

Kompilacja ma ogromne zastosowanie: pozwala zoptymalizować kod, wykrywać błędy już na etapie budowania (przed wykonaniem), kontrolować zależności bibliotek i wersji, a także umożliwia dystrybucję oprogramowania w formie niezależnej od środowiska deweloperskiego (jako binaria). Ponadto proces kompilacji jest niezbędny w systemach, gdzie bezpieczeństwo i wydajność są krytyczne, daje to kontrolę nad flagami optymalizacji, generowaniem symboli debug, kontrolą wersji bibliotek i integracją z systemami budowania (make, cmake, cargo itp.). Dla studentów i administratorów zrozumienie tego procesu oznacza lepszą świadomość, co się dzieje „pod maską” przy budowaniu programów, łatwiejsze debugowanie, i umiejętność dostosowania opcji kompilatora do potrzeb projektu.

Kompilacja w C++ – praktyka i narzędzia

make to narzędzie do automatyzacji procesu budowania oprogramowania, szczególnie przy projektach składających się z wielu plików źródłowych i zależności. System make pozwala zapisać reguły (instrukcje), jakie pliki zależą od jakich oraz

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

jakie polecenia należy wykonać, by odtworzyć dany plik (cel). Taki przepis zapisuje się w pliku o nazwie Makefile. Przy wywołaniu make, program odczytuje Makefile i decyduje, które cele trzeba odbudować na podstawie zależności i dat modyfikacji plików, jeżeli plik źródłowy jest nowszy niż obiektowy, zostaje przebudowany. Dzięki temu make potrafi budować tylko to, co się zmieniło, zamiast kompilować wszystko "od nowa". Make domyślnie szuka plików o nazwach GNUmakefile, makefile lub Makefile w bieżącym katalogu (w tej kolejności). Jeśli Twój plik nazywa się inaczej, trzeba wywołać make z opcją -f, np. make -f nazwa_makefile.

Makefile składa się z reguł w postaci:

cel: zależności

<tab> polecenie

gdzie cel to plik, który chcemy uzyskać, zależności to pliki, od których on zależy, a polecenie to komenda (np. kompilator), którą make uruchomi, żeby zaktualizować cel, jeśli zależności zostały zmienione. Warto dodać, że pierwsza reguła w Makefile staje się regułą domyślną (to, co make wykona, gdy nie podasz celu).

Make wspiera pojęcia "reguł domyślnych", reguł wzorca, automatycznych zmiennych (jak $@, $<, $^) oraz cele "phony" (pozorne cele, które nie odpowiadają rzeczywistym plikom, np. clean).

Przykład: kompilacja „Hello, World!" ręcznie, oraz poprzez make

Utwórz plik hello.cpp z prostą zawartością:

```
#include <iostream>
int main() {
        std::cout << "Hello, World!" << "\n";
        return 0;
}
```

Aby skompilować program ręcznie, bez użycia Makefile, można użyć polecenia:

```
g++ hello.cpp -o hello
```

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Powinien powstać plik wykonywalny hello, aby go uruchomić należy wpisać:

./hello

Jeśli wszystko działa, uruchomi się program wyświetlający „Hello, World!”.

Zamiast ręcznego wpisywania poleceń, utwórz plik Makefile w tym samym katalogu:

```
# nazwijmy nasz plik Makefile
CXX = g++
CXXFLAGS = -std=c++17 -Wall
hello: hello.cpp
	$(CXX) $(CXXFLAGS) hello.cpp -o hello
clean:
	rm -f hello
```

(Uwaga: w linii z $(CXX)… pierwszym znakiem musi być tabulator, nie spacje.)

Teraz w terminalu wpisz:

make

Make przeczyta regułę hello, sprawdzi, czy hello.cpp jest nowszy niż hello (lub czy hello nie istnieje), i wykona komendę kompilującą hello.cpp do binarki. Następnie uruchom:

./hello

Powinieneś zobaczyć „Hello, World!”. Jeśli potem ponownie wpiszesz make, a plik hello.cpp się nie zmienił, make nie będzie ponownie kompilować — stwierdzi, że cel hello jest aktualny i zakończy działanie bez odtwarzania komend.

Aby usunąć wygenerowany plik, użyj polecenia:

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

make clean.

Make wywoła regułę clean, która usuwa plik hello.

Kompilacja w Rust – narzędzie Cargo i ekosystem

Rust to nowoczesny język systemowy zaprojektowany z myślą o bezpieczeństwie, wydajności i współbieżności. Podobnie jak C i C++, kompiluje się bezpośrednio do kodu maszynowego, jednak proces budowania w Rust jest znacznie bardziej zautomatyzowany i zintegrowany. Zamiast wielu rozproszonych narzędzi (kompilator, linker, make, menedżer pakietów), Rust dostarcza jednolity ekosystem obejmujący zarówno kompilator (rustc), jak i menedżera projektu, zależności oraz system budowania jakim jest Cargo.

Proces kompilacji w Rust można rozumieć na kilku poziomach:

- Analiza i budowanie projektu – Cargo odczytuje plik konfiguracyjny Cargo.toml, który definiuje nazwę projektu, typ (binarny lub biblioteka), wersję, zależności (dependencies), a także opcje kompilacji i testów.
- Preprocessing i parsing – Rust nie używa klasycznego preprocesora (#define, #include), lecz system modułów (mod, use, crate) i makr (macro_rules!, procedural macros). Wszystko odbywa się w ramach jednej spójnej składni i nie wymaga osobnych kroków przetwarzania tekstowego.
- Kompilacja jednostek (crate) – każda paczka Rust (crate) kompilowana jest niezależnie przez rustc. Wynikiem jest artefakt w postaci biblioteki (.rlib, .so, .dll) lub programu binarnego (ELF, PE).
- Linkowanie i optymalizacja - Rust, wykorzystując LLVM, wykonuje zaawansowaną optymalizację kodu (inlining, dead-code elimination, constant folding) oraz linkuje niezbędne moduły, tworząc finalny plik binarny.
- Dzięki ścisłej integracji z LLVM Rust generuje kod o porównywalnej wydajności do C++, ale z gwarancjami bezpieczeństwa pamięci bez konieczności używania garbage collectora.

Podstawą kompilacji jest plik konfiguracyjny Cargo.toml, który pełni rolę centralnego opisu projektu. Zawiera on nazwę pakietu, wersję, typ (program lub biblioteka), a także zestaw zależności wraz z ich wersjami. Cargo analizuje ten plik i zarządza procesem budowania, uruchamiając kompilator rustc dla poszczególnych modułów. Rust nie korzysta z klasycznego preprocesora, lecz z własnego systemu modułów i makr, które są częścią języka, co eliminuje typowe błędy związane z

makrodefinicjami i dołączaniem nagłówków. Każdy moduł (crate) jest kompilowany niezależnie, a następnie łączony w końcowy plik binarny lub bibliotekę. Rust wykorzystuje backend LLVM, który odpowiada za generowanie kodu maszynowego oraz optymalizacje na poziomie asemblera, co zapewnia wysoką efektywność wynikowych plików wykonywalnych.

Środowisko Cargo automatyzuje wszystkie etapy pracy z projektem. Utworzenie nowego programu, jego kompilacja, uruchomienie i testowanie odbywa się z użyciem prostych poleceń: cargo new, cargo build, cargo run, cargo test. Wynikiem kompilacji są pliki umieszczane w katalogu target/, który zawiera zarówno pliki obiektowe, jak i finalny binarny artefakt. Cargo rozpoznaje zmiany w kodzie źródłowym i rekompiluje jedynie zmodyfikowane moduły, co znacząco skraca czas pracy. Dodatkowo rozróżnia dwa tryby budowania: debugowy i zoptymalizowany (--release), pozwalając przełączać się między nimi zależnie od potrzeb testowania lub wdrażania.

W odróżnieniu od tradycyjnych narzędzi, takich jak Make czy CMake, Cargo integruje w jednym miejscu zarządzanie zależnościami, testami i dokumentacją. Biblioteki zewnętrzne pobierane są automatycznie z centralnego repozytorium crates.io, a ich wersje i konfiguracja są zapisywane w pliku Cargo.lock, który zapewnia pełną powtarzalność budowy projektu na różnych maszynach. System wersjonowania oparty o semantyczne reguły pozwala precyzyjnie określać, z jakich wersji bibliotek należy korzystać, a Cargo samodzielnie rozwiązuje konflikty między nimi. Dla projektów o złożonej strukturze, obejmujących wiele modułów, Cargo potrafi budować hierarchię zależności i zarządzać kolejnością ich kompilacji.

Rust i Cargo wprowadzają nową jakość w zakresie bezpieczeństwa i utrzymania kodu. Mechanizmy typów i pożyczania danych (borrow checker) sprawiają, że wiele błędów pamięciowych jest eliminowanych jeszcze na etapie kompilacji. Dzięki integracji z analizatorami statycznymi, takimi jak Clippy, oraz automatycznym formatowaniem kodu (rustfmt), środowisko wymusza spójność i poprawność stylistyczną już podczas tworzenia oprogramowania. Cargo umożliwia także generowanie dokumentacji w sposób zautomatyzowany, na podstawie komentarzy w kodzie (cargo doc), oraz prowadzenie testów jednostkowych i integracyjnych (cargo test) w tym samym cyklu budowy, bez konieczności konfiguracji zewnętrznych frameworków.

Istotnym elementem ekosystemu jest także możliwość tworzenia bibliotek w różnych formatach: natywnych (rlib), współdzielonych (cdylib) lub statycznych (staticlib). Dzięki temu Rust może być z powodzeniem wykorzystywany do budowy komponentów współpracujących z kodem w C i C++, co otwiera drogę do stopniowej migracji istniejących systemów na nowocześniejsze rozwiązania. Integracja na poziomie binarnym odbywa się poprzez interfejs FFI (Foreign Function Interface), a proces linkowania jest w pełni wspierany przez Cargo i rustc.

Pod względem praktycznym, praca z Cargo eliminuje wiele typowych problemów spotykanych w starszych językach. Nie ma potrzeby ręcznego tworzenia skryptów Makefile, śledzenia zależności ani aktualizowania bibliotek w różnych katalogach systemowych. Wszystko odbywa się w ramach jednego narzędzia, które zachowuje spójność projektu, zapewnia automatyczne pobieranie i kompilację potrzebnych modułów oraz umożliwia ich łatwe ponowne wykorzystanie. Taki model pracy znacząco skraca czas konfiguracji i redukuje ryzyko błędów środowiskowych.

Ćwiczenia samodzielne

1. Utwórz plik hello.c z prostym programem „Hello, world!” i skompiluj go przy użyciu gcc hello.c -o hello. Uruchom i sprawdź wynik.
2. Wyświetl etapy kompilacji: użyj opcji -E, -S, -c aby zobaczyć efekty preprocesora, asemblera i pliku obiektowego.
3. Dodaj plik nagłówkowy greetings.h z deklaracją funkcji i plik greetings.c z definicją. Skompiluj obie części i połącz w jeden program (gcc main.c greetings.c -o greet).
4. Użyj flag kompilatora -Wall -Wextra -O2 i zaobserwuj różnice między wersjami bez optymalizacji i zoptymalizowaną.
5. Skompiluj program z debugowaniem: dodaj flagę -g, uruchom gdb ./hello, wykonaj polecenia run, list, break main, step.
6. Utwórz katalog projekt1/ z plikami main.c, math_utils.c, math_utils.h. Zdefiniuj funkcję int square(int x) i użyj jej w main.
7. Utwórz w katalogu projekt1/ prosty plik Makefile, który:

- kompiluje plik main.c do programu app,
- posiada reguły all i clean,
- używa zmiennych:
- CC = gcc

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

- CFLAGS = -Wall -Wextra -O2
- czyści pliki wynikowe poleceniem make clean.

Uruchom poniższe polecenie i sprawdź efekty działania reguł:

make

./app

make clean

8. Rozszerz istniejący Makefile, aby:
- automatycznie kompilował wszystkie pliki .c w katalogu src/,
- zapisywał pliki obiektowe w build/,
- używał reguł wzorcowych i zmiennych automatycznych:

%.o: %.c

$(CC) $(CFLAGS) -c $< -o $@

- oraz zmiennej $@ w regule linkowania:

app: $(OBJS)

$(CC) $(CFLAGS) $^ -o $@

Sprawdź działanie i zrób testowy plik src/test.c, aby zobaczyć, że Make kompiluje tylko to, co się zmieniło.

9. Dodaj możliwość wyboru kompilatora przy uruchomieniu:

make CC=clang

oraz zdefiniuj dwa zestawy flag:

CFLAGS_DEBUG = -g -Wall

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

CFLAGS_RELEASE = -O2 -Wall

Następnie utwórz dwa cele:

debug:

$(MAKE) CFLAGS="$(CFLAGS_DEBUG)"

release:

$(MAKE) CFLAGS="$(CFLAGS_RELEASE)"

Porównaj rozmiary plików wynikowych z obu konfiguracji.

10. Dodaj do projektu plik math_utils.c z funkcją int square(int x) { return x*x; }.

    Skompiluj go i utwórz bibliotekę statyczną:

    ar rcs libmath.a math_utils.o

    Połącz ją z programem main.c:

    gcc main.c -L. -lmath -o app

    Uruchom program i sprawdź, czy działa poprawnie.

11. Utwórz bibliotekę współdzieloną:

    gcc -shared -fPIC -o libmath.so math_utils.c

    Następnie skompiluj program:

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

gcc main.c -L. -lmath -o app_shared

Ustaw zmienną środowiskową:

export LD_LIBRARY_PATH=.

Uruchom ./app_shared i sprawdź poprawność działania.

12. Dodaj do Makefile cel install, który kopiuje pliki:

install:

install -d $(HOME)/.local/bin

install app $(HOME)/.local/bin/

Uruchom:

make install

Sprawdź, czy program został skopiowany do ~/.local/bin.

13. Uzupełnij Makefile o:

- reguły all, clean, debug, release, install,
- zmienne automatyczne i wzorcowe,
- wybór kompilatora (gcc/clang),
- kompilację biblioteki (statycznej i współdzielonej),

- czyszczenie katalogów (build/, bin/, lib/).

Przetestuj, że komenda make all buduje pełen projekt, make debug działa z flagą -g, a make clean usuwa wszystkie artefakty.

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

14. Utwórz plik CMakeLists.txt dla prostego projektu HelloCMake, definiując minimum cmake_minimum_required(VERSION 3.10) i add_executable(hello main.cpp).
15. Skonfiguruj i zbuduj projekt w osobnym katalogu build/ przy użyciu cmake .. && make.
16. Dodaj do CMake plik biblioteki (add_library(math STATIC math_utils.cpp)) i połącz z target_link_libraries.
17. Zastosuj różne konfiguracje kompilacji: cmake -DCMAKE_BUILD_TYPE=Debug oraz Release i porównaj rozmiar binariów.
18. Zainstaluj projekt lokalnie do /usr/local/ przez sudo make install, a następnie uruchom zainstalowaną wersję programu.
19. twórz nowy projekt binarny w Rust:

    cargo new hello_rust --bin

    cd hello_rust

    Przejrzyj wygenerowaną strukturę plików (Cargo.toml, src/main.rs).

    Skompiluj i uruchom program przy pomocy:

    cargo build

    cargo run

20. Uruchom testy wbudowane w Cargo:

    cargo test

21. Sprawdź, jakie katalogi utworzył Cargo w folderze target/ (debug, deps itp.)
22. Wyjaśnij, jaka jest różnica między cargo build a cargo check.
23. Zbuduj projekt w trybie zoptymalizowanym:

    cargo build --release

    Porównaj rozmiar plików w target/debug/ i target/release/ oraz czas uruchomienia programu w obu trybach:

    time ./target/debug/hello_rust

    time ./target/release/hello_rust

    Sprawdź różnice i zapisz wnioski dotyczące wpływu flag optymalizacyjnych (--release) na wydajność i rozmiar binariów.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

24. Utwórz bibliotekę: cargo new math_lib --lib. Dodaj funkcję fn square(x: i32) -> i32 { x * x }.
25. W projekcie binarnym utwórz zależność lokalną do math_lib w Cargo.toml (math_lib = { path = "../math_lib" }).
26. Zbuduj projekt z biblioteką i przetestuj import (use math_lib::square;).
27. Dodaj test jednostkowy w pliku tests/integration_test.rs i uruchom cargo test.
28. Utwórz dokumentację projektu: cargo doc --open.
29. Dodaj zewnętrzną zależność do projektu (rand = "0.8") i napisz program losujący liczby.
30. Wyświetl drzewo zależności: cargo tree.
31. Utwórz plik .cargo/config.toml definiujący alias cargo r = "run --release".
32. Sprawdź projekt analizatorem statycznym: cargo clippy (analiza stylu i błędów).
33. Utwórz archiwum dystrybucyjne: cargo package i obejrzyj wygenerowany plik .crate w katalogu target/package/.
34. Utwórz Makefile, który w katalogu rust_app/ uruchamia cargo build --release.
35. Wynikowy plik target/release/rust_app skopiuj do katalogu bin/.
36. Dodaj cel rust_test, który uruchamia testy jednostkowe (cargo test) i raportuje status.
37. Utwórz cel hybrid, który buduje i linkuje wspólną bibliotekę Rust z C++ (FFI).
38. Rust: crate-type = ["cdylib"]
39. C++: g++ main.cpp -Lrust_lib/target/release -lrustlib -o hybrid_app
40. Dodaj cel package, który archiwizuje cały projekt (tar -czf build.tar.gz bin/ include/ src/ Makefile).

Pytania pomocnicze

1. Jakie są główne etapy procesu kompilacji w językach C/C++?
2. Do czego służy preprocesor?
3. Jakie polecenie w Linuxie skompiluje prosty program hello.cpp?
4. Do czego służy plik Makefile?
5. Jak działa narzędzie cmake i czym różni się od make?
6. Dlaczego instalacja do /usr/local/bin jest preferowana przy własnoręcznej kompilacji?
7. Jakie są zalety instalacji ze źródeł w porównaniu z pakietami binarnymi?
8. Czym jest Cargo w ekosystemie Rust?
9. Jak utworzyć nowy projekt w Rust przy pomocy Cargo?
10. Jak uruchomić testy jednostkowe w projekcie Rust?
11. Czym różni się cargo build od cargo run?
12. Jak zainstalować bibliotekę z crates.io w projekcie Rust?
13. Do czego służy Conan w ekosystemie C/C++?

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

14. Dlaczego Rust zyskuje popularność w kontekście systemów operacyjnych?

Podsumowanie

W ramach laboratorium studenci poznali pełny cykl życia programu kompilowanego ze źródeł – od analizy kodu, przez tłumaczenie i linkowanie, po instalację gotowego pliku binarnego. Zrozumieli różnicę pomiędzy uruchamianiem aplikacji z gotowych pakietów a budowaniem ich samodzielnie z kodu źródłowego, co daje większą kontrolę nad procesem, środowiskiem i optymalizacją.

W części praktycznej zapoznano się z tradycyjnymi narzędziami kompilacji wykorzystywanymi w językach C i C++, takimi jak make i cmake, umożliwiającymi automatyzację procesu budowy większych projektów. Następnie przedstawiono nowoczesne podejście reprezentowane przez język Rust i jego ekosystem Cargo, który integruje w jednym narzędziu funkcje kompilatora, menedżera zależności i systemu testowania.

Ćwiczenia pozwoliły porównać klasyczny model kompilacji, oparty na regułach Makefile, z podejściem zautomatyzowanym i deklaratywnym w Cargo. Uczestnicy prześledzili proces tworzenia bibliotek, uruchamiania testów jednostkowych, konfiguracji środowiska oraz instalacji aplikacji lokalnie w systemie.

Uzyskana wiedza stanowi podstawę do samodzielnego kompilowania i analizy oprogramowania open source oraz do dalszej pracy z językami systemowymi, gdzie zrozumienie działania kompilatora i narzędzi budujących ma kluczowe znaczenie dla bezpieczeństwa, wydajności i stabilności systemu.

Literatura

1. Compiling Programs in Linux: A Beginner's Step-by-Step Guide, Link: https://www.spsanderson.com/steveondata/posts/2025-02-07/,(data ostatniego dostępu 25.09.25)
2. C++ Compilation process, Link:

Fundusze Europejskie dla Rozwoju Społecznego | Rzeczpospolita Polska | Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

https://www.geeksforgeeks.org/cpp/how-to-compile-a-cpp-program-using-gcc/, (data ostatniego dostępu 25.09.25)

3. How to Use the Rust Compiler 'rustc' (with examples), Link: https://commandmasters.com/commands/rustc-common/, (data ostatniego dostępu 25.09.25)

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Materiały do przedmiotu:

Podstawy Systemów Operacyjnych

Laboratorium nr 8:

Zarządzanie kontami użytkowników i grup, nadawanie uprawnień oraz konfiguracja polityk bezpieczeństwa. Monitorowanie procesów, rejestrów zdarzeń i logów systemowych

Autor:

Albert Rachwał

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Spis treści

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Wprowadzenie, tematyka zajęć

Ostatnie laboratorium stanowi zwieńczenie całego cyklu i koncentruje się na bezpieczeństwie systemu. Studenci poznają mechanizmy zarządzania użytkownikami i grupami, a także sposoby nadawania i odbierania uprawnień. Omawiane są rozszerzone mechanizmy kontroli dostępu, konfiguracja zasad bezpieczeństwa oraz rola modułów odpowiedzialnych za uwierzytelnianie. Uczestnicy dowiadują się także, jak monitorować procesy i jak interpretować logi systemowe, aby wykrywać nieprawidłowości i reagować na zagrożenia. W części praktycznej zakładane są konta użytkowników, modyfikowane ich prawa, a następnie analizowane zdarzenia zapisane w logach. Zajęcia pokazują, że system operacyjny to nie tylko narzędzie do uruchamiania aplikacji, ale także złożony mechanizm kontroli i ochrony danych. Dzięki temu studenci wychodzą z kursu wyposażeni w wiedzę pozwalającą nie tylko korzystać z Linuxa, lecz także bezpiecznie nim zarządzać.

Cel ćwiczenia/laboratorium

- Zapoznanie z modelem użytkowników i grup w systemie Linux oraz ich reprezentacją w plikach konfiguracyjnych /etc/passwd, /etc/shadow, /etc/group.
- Przećwiczenie tworzenia, modyfikowania i usuwania kont użytkowników oraz grup z wykorzystaniem poleceń useradd, usermod, passwd, groupadd.
- Poznanie zasad przypisywania użytkowników do grup oraz weryfikacji przynależności (id, groups, gpasswd).
- Zrozumienie różnic między kontami interaktywnymi a systemowymi.
- Zapoznanie z klasycznym modelem kontroli dostępu (DAC) oraz rozszerzeniami w postaci list ACL.
- Przećwiczenie nadawania i modyfikowania uprawnień do plików i katalogów przy użyciu chmod, chown, chgrp, setfacl, getfacl.
- Analiza działania specjalnych bitów uprawnień – SUID, SGID i Sticky Bit.
- Zapoznanie z zasadami bezpiecznego korzystania z uprawnień administracyjnych (sudo, visudo).
- Omówienie podstaw działania modułów PAM w kontekście haseł, blokad logowania i polityki bezpieczeństwa.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

- Poznanie różnic między modelami kontroli dostępu DAC i MAC oraz przykładowych implementacji (SELinux, AppArmor).

- Zapoznanie z narzędziami monitorowania procesów (ps, top, htop, pidstat) oraz analizy obciążenia systemu.
- Przećwiczenie identyfikacji procesów nadmiernie wykorzystujących zasoby i zapisu wyników obserwacji do pliku.
- Analiza logów systemowych przy użyciu journalctl oraz plików w katalogu /var/log/.
- Wykrywanie i interpretacja nieudanych prób logowania oraz innych zdarzeń bezpieczeństwa.
- Zastosowanie systemu auditd do monitorowania zmian w plikach krytycznych.
- Zrozumienie znaczenia synchronizacji czasu i retencji logów dla poprawnej analizy zdarzeń

Instrukcja laboratoryjna/ćwiczeniowa

Zarządzanie użytkownikami i grupami w systemach Linux

Zarządzanie użytkownikami i grupami jest jednym z podstawowych, a zarazem najważniejszych elementów administracji systemem operacyjnym Linux. Stanowi ono fundament bezpieczeństwa i kontroli dostępu, ponieważ to właśnie użytkownicy i grupy użytkowników są jednostkami, którym system przypisuje prawa i uprawnienia do wykonywania operacji na zasobach. W odróżnieniu od systemów jedno-użytkownikowych, Linux, jako system wielodostępny wymaga precyzyjnego określenia, kto może wykonywać określone działania, na jakich danych i w jakim zakresie.

Model użytkownika w systemie Linux

Każdy użytkownik w systemie jest reprezentowany przez unikalny identyfikator liczbowy - UID (User Identifier). Użytkownikom przypisywane są również atrybuty opisowe, takie jak nazwa konta, katalog domowy, powłoka logowania oraz przynależność do jednej lub więcej grup. Informacje te są przechowywane w plikach konfiguracyjnych, m.in.:

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

- /etc/passwd – podstawowy plik opisujący konta użytkowników, zawierający m.in. nazwę użytkownika, UID, GID (identyfikator grupy głównej), katalog domowy i powłokę logowania.
- /etc/shadow – plik przechowujący zaszyfrowane hasła użytkowników oraz informacje o ich ważności (np. data wygaśnięcia, okres blokady). Dostęp do tego pliku ma wyłącznie administrator (root).
- /etc/group – plik definiujący grupy systemowe oraz użytkowników należących do każdej z nich.

Struktura użytkowników systemu obejmuje dwa główne typy kont:

- Użytkownicy systemowi (system users) – konta tworzone automatycznie przez system lub aplikacje, wykorzystywane do uruchamiania usług i demonów w odizolowany sposób. Przykładem może być użytkownik www-data dla serwera Apache lub mysql dla bazy danych MySQL.
- Użytkownicy interaktywni (human users) – konta przeznaczone dla rzeczywistych użytkowników, logujących się do systemu i wykonujących operacje interaktywne.

Grupy użytkowników i ich rola

W systemie Linux każdy użytkownik należy przynajmniej do jednej grupy – głównej (primary group), a opcjonalnie do wielu grup dodatkowych (supplementary groups). Grupy stanowią mechanizm umożliwiający nadawanie wspólnych uprawnień wielu użytkownikom jednocześnie, co upraszcza zarządzanie bezpieczeństwem i dostępem do plików.

Każda grupa posiada unikalny GID (Group Identifier) i jest opisana w pliku /etc/group. Uprawnienia plików i katalogów definiowane są na poziomie trzech kontekstów: właściciela (user), grupy (group) i pozostałych (others). Dzięki temu możliwe jest np. przyznanie dostępu do katalogu projektowego tylko członkom określonej grupy.

Polecenia administracyjne i operacje na użytkownikach

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

Zarządzanie kontami użytkowników odbywa się przy pomocy zestawu narzędzi administracyjnych, które pozwalają tworzyć, modyfikować, blokować i usuwać konta. Do najważniejszych należą:

- useradd – tworzy nowe konto użytkownika, wraz z katalogiem domowym, wpisem w plikach /etc/passwd i /etc/shadow, oraz domyślną powłoką.
- usermod – modyfikuje istniejące konto, np. zmienia hasło, katalog domowy, grupy, powłokę logowania.
- passwd – umożliwia ustawienie lub zmianę hasła użytkownika.
- userdel – usuwa konto użytkownika, opcjonalnie wraz z jego katalogiem domowym.
- groupadd, groupmod, groupdel – analogiczne polecenia do zarządzania grupami systemowymi.
- gpasswd – pozwala przypisywać użytkowników do grup oraz ustawiać hasła grupowe.

Administrator może również weryfikować identyfikatory użytkowników poleceniami id, whoami, groups lub finger, które wyświetlają informacje o aktualnym kontekście użytkownika i jego grupach.

Uprawnienia, role i zasady bezpieczeństwa

Zarządzanie użytkownikami nie ogranicza się do tworzenia kont, a jego istotą jest również właściwe przypisanie uprawnień. W systemie Linux uprawnienia mogą być nadawane w kilku warstwach:

- Uprawnienia plików i katalogów (DAC) – tradycyjny model kontroli dostępu, w którym określa się, kto może czytać, zapisywać lub wykonywać plik. Odbywa się to poleceniami chmod, chown i chgrp.
- Listy ACL (Access Control Lists) – rozszerzenie modelu DAC, umożliwiające przypisywanie indywidualnych uprawnień konkretnym użytkownikom lub grupom dla pojedynczego pliku lub katalogu (setfacl, getfacl).
- Uprawnienia specjalne – mechanizmy takie jak SUID, SGID i Sticky Bit, które zmieniają sposób wykonywania programów i dostęp do katalogów (np. umożliwiają wykonywanie programu z uprawnieniami właściciela).
- Uprawnienia administracyjne (root, sudo) – użytkownik root posiada pełnię uprawnień w systemie. Aby zapewnić bezpieczeństwo, w praktyce używa się

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

polecenia sudo, które tymczasowo nadaje użytkownikowi określone uprawnienia administracyjne, zgodnie z definicją w pliku /etc/sudoers (edytowanym przez visudo).

Bezpieczeństwo kont użytkowników

Z perspektywy bezpieczeństwa, zarządzanie kontami użytkowników wymaga przestrzegania szeregu zasad. Przede wszystkim należy stosować zasadę minimalnych uprawnień, unikać pracy na koncie root oraz wymuszać stosowanie silnych haseł. Mechanizmy PAM (Pluggable Authentication Modules) umożliwiają konfigurację polityki haseł, takich jak minimalna długość, złożoność czy czas ważności. Dodatkowo można wdrożyć blokady kont po zbyt wielu nieudanych próbach logowania (np. pam_faillock), co zapobiega atakom typu brute force.

Istotnym aspektem jest także kontrola nieaktywnych kont i ich automatyczne blokowanie (usermod -L, chage -E). W systemach wieloużytkownikowych dobrą praktyką jest regularny audyt listy kont (awk -F: '{print $1}' /etc/passwd) i usuwanie nieużywanych lub tymczasowych użytkowników.

Konfiguracja polityk bezpieczeństwa w systemach Linux

Polityka bezpieczeństwa systemu operacyjnego to zestaw reguł i mechanizmów, które określają, jakie działania mogą wykonywać użytkownicy, procesy i aplikacje w danym środowisku. Konfiguracja tych polityk obejmuje definiowanie, wdrażanie i utrzymywanie zasad, które ograniczają zakres dostępnych operacji w systemie — nawet dla użytkowników posiadających uprawnienia administracyjne (root). Celem takiego podejścia jest zbudowanie warstwowego modelu ochrony, w którym potencjalne naruszenie jednego z poziomów nie powoduje utraty bezpieczeństwa całego systemu.

Właściwie zaprojektowana i wdrożona polityka bezpieczeństwa stanowi fundament odporności systemu operacyjnego. Chroni przed skutkami błędów oprogramowania, ataków typu exploit oraz nieautoryzowanego dostępu. W systemach Linux wyróżnia się dwa główne modele kontroli dostępu:

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

- DAC (Discretionary Access Control) – tradycyjny model, w którym o dostępie do plików i katalogów decydują uprawnienia właściciela i grupy. Każdy obiekt systemu posiada trzy zestawy praw: dla właściciela (user), grupy (group) oraz pozostałych użytkowników (others), które obejmują możliwość odczytu (r), zapisu (w) i wykonania (x). W tym modelu użytkownik ma kontrolę nad własnymi plikami i może samodzielnie zmieniać ich uprawnienia.
- MAC (Mandatory Access Control) – model oparty na centralnie narzuconych regułach bezpieczeństwa, które obowiązują wszystkie procesy i użytkowników, niezależnie od ich indywidualnych uprawnień wynikających z DAC. Nawet użytkownik root podlega tym ograniczeniom. MAC umożliwia tworzenie złożonych i precyzyjnych polityk, które definiują, jakie operacje mogą wykonywać konkretne procesy w odniesieniu do konkretnych zasobów.

Celem konfiguracji polityk bezpieczeństwa jest wdrożenie zasad, które zapewnią integralność, poufność i dostępność zasobów systemowych. Osiąga się to poprzez realizację kilku kluczowych zasad:

- Zasada najmniejszych uprawnień (Least Privilege) – każdy proces i użytkownik powinien mieć dostęp jedynie do tych zasobów, które są absolutnie niezbędne do wykonania jego zadań.
- Ograniczanie skutków błędów i ataków – w przypadku naruszenia jednego komponentu systemu, jego zdolność do wpływania na pozostałe elementy powinna być minimalna.
- Konsolidacja i przewidywalność – polityki bezpieczeństwa powinny być definiowane centralnie i stosowane jednolicie, co ułatwia kontrolę i audyt.
- Audytowalność i odpowiedzialność – każde naruszenie polityki musi być rejestrowane, aby możliwe było późniejsze prześledzenie przyczyny i podjęcie działań naprawczych.
- Elastyczność – polityki muszą być dostosowywane do zmian w środowisku informatycznym, nie tracąc przy tym spójności i restrykcyjności.

Narzędzia i mechanizmy konfiguracji polityk bezpieczeństwa

System Linux oferuje szereg rozbudowanych mechanizmów i narzędzi służących do egzekwowania polityk bezpieczeństwa. Wspólnie tworzą one wielowarstwowy system kontroli dostępu, audytu i izolacji procesów.

1. Linux Security Modules (LSM)

LSM stanowi warstwę w jądrze systemu Linux, umożliwiającą dołączanie różnych implementacji polityk bezpieczeństwa. Mechanizm ten udostępnia specjalne „punkty zaczepienia” (hooks) w kluczowych operacjach systemowych, takich jak otwieranie plików, uruchamianie procesów czy komunikacja sieciowa. Moduły bezpieczeństwa (np. SELinux, AppArmor, Smack) mogą w tych miejscach wprowadzać własne decyzje o zezwoleniu lub odmowie wykonania danej operacji. Dzięki temu Linux staje się systemem, w którym kontrola dostępu może być rozszerzana i personalizowana bez modyfikowania samego jądra.

2. SELinux (Security-Enhanced Linux)

SELinux to jedna z najpełniejszych implementacji modelu MAC. Każdemu obiektowi systemu (plikowi, katalogowi, procesowi, gniazdu sieciowemu) przypisywana jest etykieta bezpieczeństwa (label), która określa jego typ i kontekst bezpieczeństwa. Reguły SELinux (policies) definiują, które typy obiektów mogą wykonywać określone operacje na innych typach. Polityka ta jest niezależna od standardowych uprawnień DAC i obowiązuje globalnie. SELinux obsługuje także modele oparte na rolach (RBAC) oraz wielopoziomowe bezpieczeństwo (MLS).

System ten może działać w trzech trybach: enforcing (aktywnie egzekwuje reguły), permissive (tylko rejestruje naruszenia) oraz disabled (wyłączony). Administracja SELinux odbywa się przy pomocy narzędzi takich jak semanage, setsebool, restorecon, sestatus czy audit2why.

3. AppArmor

AppArmor to alternatywne, uproszczone rozwiązanie typu MAC, szeroko stosowane w dystrybucjach takich jak Ubuntu czy openSUSE. W przeciwieństwie do SELinux, który bazuje na etykietach, AppArmor przypisuje profil bezpieczeństwa do konkretnego programu lub procesu, identyfikując go po ścieżce do pliku wykonywalnego. Profil określa, do jakich zasobów dany program może mieć dostęp i jakie operacje może wykonywać. AppArmor jest uznawany za łatwiejszy w konfiguracji i utrzymaniu, szczególnie w środowiskach akademickich i laboratoryjnych. Profile mogą działać w trybie enforce (blokowanie nieautoryzowanych działań) lub complain (jedynie raportowanie).

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

4. PAM (Pluggable Authentication Modules)

System PAM zapewnia modularną architekturę uwierzytelniania użytkowników. Pozwala on konfigurować zasady dotyczące haseł, logowania, blokad kont oraz limitów dostępu. Dzięki modułom takim jak pam_tally2, pam_faillock czy pam_cracklib administrator może wymusić określoną złożoność haseł, ograniczyć liczbę prób logowania, zablokować konto po przekroczeniu limitu błędów lub wymusić okresową zmianę hasła. Pliki konfiguracyjne PAM znajdują się w katalogu /etc/pam.d/, a każda usługa systemowa (np. sshd, login, sudo) może mieć własny zestaw reguł uwierzytelniania.

5. Auditd (system audytu)

Auditd to demon odpowiedzialny za rejestrowanie zdarzeń bezpieczeństwa w systemie. Umożliwia on definiowanie reguł audytu (audit rules), które wskazują, jakie działania powinny być monitorowane, np. dostęp do plików, zmiana uprawnień, modyfikacje konfiguracji czy próby uruchomienia określonych programów. Zgromadzone dane można analizować narzędziami ausearch, aureport czy auditctl. Auditd jest często zintegrowany z SELinux i AppArmor, co pozwala na korelację zdarzeń naruszeń z konkretnymi procesami i użytkownikami.

6. Dodatkowe mechanizmy bezpieczeństwa systemowego

W ramach konfiguracji polityk bezpieczeństwa można uwzględnić także inne komponenty, takie jak firewalle (iptables, nftables), mechanizmy izolacji procesów (namespaces, cgroups), systemy sandboxowania (np. Firejail), a także narzędzia ograniczające zestaw wywołań systemowych (seccomp) lub uprawnienia jądra (Linux capabilities). Wszystkie te elementy współtworzą kompleksową architekturę zabezpieczeń, która ogranicza ryzyko eskalacji uprawnień i nadużyć w środowisku wieloużytkownikowym.

Monitorowaniu procesów, rejestrów zdarzeń i logów systemowych

W systemach uniksowych monitorowanie procesów i logów stanowi kluczowy mechanizm obserwowalności (observability), czyli zdolności systemu do odsłaniania swojego stanu wewnętrznego poprzez metryki, zdarzenia i ślady. Na poziomie procesów Linux udostępnia bogaty model introspekcji oparty o przestrzenie nazw, cgroups oraz wirtualne systemy plików /proc i /sys. Każdy proces ma własny

podkatalog /proc/<PID> z metadanymi (m.in. deskryptory plików, użycie pamięci, mapy adresowe, parametry uruchomienia), co pozwala analizować zachowanie programów bez ingerencji w ich kod. Narzędzia w rodzaju ps, top, htop, pidstat, vmstat, iostat czy mpstat prezentują te informacje w czasie rzeczywistym, umożliwiając identyfikację procesów konsumujących CPU, pamięć, I/O i gniazda sieciowe, a także diagnozę zjawisk takich jak zakleszczenia, wycieki pamięci czy nadmierna liczba wątków. W razie potrzeby możliwa jest inspekcja na poziomie wywołań systemowych (strace) i profilowanie (perf, eBPF), które ujawniają szczegółową przyczynę degradacji wydajności lub błędów wykonania. W środowiskach produkcyjnych monitorowanie bywa powiązane z kontrolą (cgroups, limity zasobów), tak by niepożądany proces nie mógł wyeksploatować całej maszyny.

Równolegle do obserwacji procesów, system rejestruje zdarzenia w dziennikach (logach), które pełnią rolę kanonicznego zapisu historii działania systemu i aplikacji. W nowoczesnych dystrybucjach centralnym elementem jest systemd-journald, który zbiera komunikaty z jądra, usług i aplikacji, przechowując je w indeksowanej, binarnej formie dostępnej przez journalctl. Klasyczny system syslog (np. rsyslog, syslog-ng) wykorzystuje pojęcia facility i severity, umożliwiając kierowanie komunikatów do odpowiednich plików w /var/log/ oraz dalsze ich przekazywanie do zewnętrznych kolektorów. Dzięki temu możliwe jest filtrowanie logów po jednostkach (unitach) systemd, priorytecie, czasie, hoście źródłowym czy identyfikatorze procesu, a także budowa złożonych reguł routingu, retencji i kompresji. Korekcyjny proces log rotation (np. logrotate) zapobiega przepełnieniu dysku, utrzymując równowagę między dostępnością danych historycznych a kosztem ich przechowywania. Szczególną kategorię stanowi audyt bezpieczeństwa (auditd) oraz zdarzenia jądra widoczne w buforze dmesg, które wspierają dochodzenia powłamaniowe i analizę incydentów. Dbałość o spójny czas (NTP/chrony) i jednolite strefy czasowe jest tu krytyczna, bez synchronizacji zegara korelacja zdarzeń między hostami staje się zawodna.

Dane z monitoringu i logów nabierają wartości dopiero wtedy, gdy są interpretowane w kontekście. Dobre praktyki obejmują: wymuszanie logowania w formacie strukturalnym (pola klucz–wartość, JSON), dodawanie metadanych (host, jednostka, identyfikator żądania, identyfikator użytkownika), jednoznaczne poziomy ważności (DEBUG/INFO/WARN/ERROR/CRITICAL) oraz konserwatywne logowanie danych wrażliwych (minimalizacja i anonimizacja zgodna z politykami prywatności). W skali

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

większych środowisk wykorzystuje się systemy agregacji i wyszukiwania (np. ELK/Opensearch, Graylog, Loki), które umożliwiają indeksowanie, wizualizację, alertowanie i korelację zdarzeń z wielu serwerów. Z punktu widzenia reakcji operacyjnej alerting powinien opierać się na regułach wykrywających symptomy problemów (np. gwałtowny wzrost błędów 5xx, spadek przepustowości, powtarzające się restarty jednostek, anomalie w zużyciu CPU/RSS/IOPS), przy jednoczesnym ograniczaniu fałszywych alarmów przez progowanie, histerezę i tłumienie burzy zdarzeń. Wreszcie kwestie ładu i zgodności: polityka retencji logów, kontrola integralności (sumy, WORM, podpisy), uprawnienia do odczytu logów, rozdzielenie ról operatorów i deweloperów oraz wersjonowanie konfiguracji (infrastructure/pipelines as code) decydują o tym, czy system monitorowania staje się wiarygodnym narzędziem dowodowym i diagnostycznym, czy tylko zbiorem nieprzetworzonych komunikatów. W dobrze zaprojektowanym środowisku monitoring procesów i rejestrów zdarzeń tworzy spójny ekosystem: metryki wskazują kiedy dzieje się coś nieprawidłowego, a logi i ślady pozwalają zrozumieć dlaczego, dzięki czemu zespół może szybko wykrywać, diagnozować i usuwać usterki, a także proaktywnie zapobiegać ich nawrotom.

Ćwiczenia samodzielne

1. Utwórz użytkownika student1 z katalogiem domowym i ustaw dla niego hasło.
2. Utwórz grupę labgroup i przypisz do niej student1.
3. Zmień powłokę użytkownika student1 na /usr/sbin/nologin i sprawdź, że nie może się zalogować normalnie.
4. Odblokuj konto student1 i zmień mu powłokę z powrotem na /bin/bash.
5. Zmień katalog domowy użytkownika student1 na /home/stu1_new przy zachowaniu zawartości dotychczasowego katalogu.
6. Utwórz użytkownika student2, ale od razu przypisz go do grup labgroup i students przy tworzeniu.
7. Uwzględnij użytkownika student2 w grupie labgroup bez usuwania go z innych grup (użyj opcji append).
8. Zablokuj konto student2 (tak, by nie mógł się logować), ale nie usuwaj go całkowicie.
9. Usuń użytkownika student2 wraz z jego katalogiem domowym.

10. Sprawdź pliki /etc/passwd, /etc/shadow i /etc/group, aby upewnić się, że zmiany są poprawne (np. czy użytkownicy/grupy tam występują lub zostali usunięci).
11. Uruchom polecenie ps aux oraz top, aby wyświetlić listę aktualnie uruchomionych procesów w systemie. Zidentyfikuj, które z nich należą do użytkownika root, a które do innych użytkowników. Zapisz wynik do pliku procesy.txt i wskaż procesy o najwyższym zużyciu pamięci.
12. Zainstaluj i uruchom program htop. Zidentyfikuj procesy systemowe oraz procesy uruchomione przez użytkownika. Zwróć uwagę na kolumny CPU%, MEM% i TIME+. Zapisz wynik do pliku (np. htop > monitor.txt).
13. Wyświetl dziennik systemowy z ostatniego rozruchu przy użyciu:

    journalctl -b

    Następnie przefiltruj tylko komunikaty błędów (journalctl -p err -b) i zapisz wynik do pliku bledy_systemowe.log. Odpowiedz, jakie usługi najczęściej generują błędy.

14. Przejdź do katalogu /var/log/ i sprawdź zawartość plików auth.log, syslog oraz kern.log. Zidentyfikuj w pliku auth.log wszystkie nieudane próby logowania. Zapisz wyniki do pliku nieudane_logowania.txt.

    Użyj komendy:

    grep "Failed" /var/log/auth.log

15. Zainstaluj i uruchom usługę auditd. Dodaj regułę audytu, która będzie śledzić zmiany w pliku /etc/passwd:

    auditctl -w /etc/passwd -p wa -k monitor_passwd

    Zmień dane użytkownika lub dodaj nowego, a następnie wyświetl log audytu:

    ausearch -k monitor_passwd

    Zapisz raport do pliku audit_passwd.log.

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

16. Użyj polecenia ss -tuln lub netstat -tuln, aby wyświetlić listę otwartych portów i usług nasłuchujących w systemie. Zidentyfikuj, które usługi są dostępne z sieci zewnętrznej.
17. Uruchom narzędzie Wireshark lub jego wersję terminalową tshark. Wykonaj polecenie ping 8.8.8.8 w innym terminalu, a następnie zatrzymaj przechwytywanie. Zastosuj filtr icmp i zidentyfikuj pakiety „Echo request" oraz „Echo reply".
18. Utwórz nowe konto testowe testuser. Następnie spróbuj kilkukrotnie zalogować się błędnym hasłem przez su testuser. Użyj polecenia:

    ```
    journalctl -u ssh.service -p warning
    ```

    lub równoważnie:

    ```
    grep "Failed password" /var/log/auth.log
    ```

    Zidentyfikuj liczbę prób nieudanych logowań i nazwę konta, którego dotyczyły. Odpowiedz, jak system rejestruje takie zdarzenia i które moduły PAM za to odpowiadają.

Pytania pomocnicze

1. Jakie informacje przechowywane są w plikach /etc/passwd, /etc/shadow i /etc/group?
2. Jakie polecenie tworzy nowego użytkownika, a jakie nową grupę?
3. W jaki sposób można zmienić powłokę logowania użytkownika i jego katalog domowy?
4. Czym różni się konto systemowe od konta interaktywnego?
5. Jak można tymczasowo zablokować konto użytkownika, nie usuwając go z systemu?
6. Jak sprawdzić, do jakich grup należy użytkownik?
7. Jakie są trzy podstawowe prawa dostępu do plików i katalogów w systemie Linux?
8. Co oznacza zapis chmod 755 plik.txt?
9. Do czego służy polecenie setfacl i w jakich sytuacjach jest przydatne?
10. Na czym polegają uprawnienia SUID, SGID i Sticky Bit?
11. W jaki sposób można nadać użytkownikowi prawo do wykonywania operacji administracyjnych bez logowania jako root?
12. Czym różni się model DAC od MAC?
13. Jakie mechanizmy bezpieczeństwa wykorzystują model MAC w systemach Linux?
14. W jaki sposób moduły PAM wpływają na proces logowania i bezpieczeństwo haseł?
15. Jakie polecenia służą do monitorowania bieżących procesów w systemie Linux?
16. Jak sprawdzić, które procesy najbardziej obciążają CPU lub pamięć RAM?

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze Środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

17. Jakie informacje można uzyskać z katalogu /proc/<PID>?
18. Jak wyświetlić logi systemowe z bieżącego rozruchu przy pomocy journalctl?
19. Jak przefiltrować logi konkretnej usługi (np. SSH) lub konkretnego poziomu błędów?
20. W jakich plikach można znaleźć informacje o nieudanych próbach logowania?
21. Do czego służy auditd i jak można za jego pomocą monitorować zmiany w plikach systemowych?
22. Dlaczego synchronizacja czasu jest istotna przy analizie logów i incydentów bezpieczeństwa?

Podsumowanie

W ramach laboratorium przeanalizowano pełny cykl zarządzania użytkownikami, grupami i uprawnieniami w systemie Linux – od tworzenia kont i przypisywania ich do grup, po konfigurację zasad bezpieczeństwa i kontroli dostępu.

W części praktycznej zrealizowano zadania obejmujące modyfikację kont, zmianę powłoki logowania, blokowanie i odblokowywanie użytkowników, a także analizę plików konfiguracyjnych /etc/passwd, /etc/shadow i /etc/group. Następnie omówiono działanie mechanizmów PAM oraz zastosowanie narzędzi auditd, ps, top, htop i journalctl do monitorowania aktywności systemu i wykrywania nieprawidłowości.

Ćwiczenia pozwoliły zrozumieć, w jaki sposób Linux egzekwuje polityki bezpieczeństwa, zarządza tożsamością użytkowników i rejestruje zdarzenia w logach. Zdobyte umiejętności stanowią podstawę dalszej pracy administracyjnej, obejmującej kontrolę dostępu oraz reagowanie na incydenty bezpieczeństwa w środowiskach wieloużytkownikowych.

Literatura

1. User Management in Linux, Link: https://www.geeksforgeeks.org/linux-unix/user-management-in-linux/, (data ostatniego dostępu 25.09.25)
2. Linux Security Command Cheat Sheet, Link: https://www.geeksforgeeks.org/linux-unix/linux-security-command-cheat-sheet/ , (data ostatniego dostępu 25.09.25)

Fundusze Europejskie dla Rozwoju Społecznego | Rzeczpospolita Polska | Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00

3. Mastering Wireshark on Linux: A Comprehensive Guide , Link: https://linuxvox.com/blog/wireshark-on-linux/, (data ostatniego dostępu 25.09.25)

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Projekt współfinansowany ze środków Unii Europejskiej w ramach: FUNDUSZE EUROPEJSKIE DLA ROZWOJU SPOŁECZNEGO 2021-2027 (FERS) "POLLUB z nami nowoczesne technologie" FERS.01.05-IP.08-0319/23-00