1. WSTĘP

1.1. WIADOMOŚCI PODSTAWOWE

Linux jest systemem operacyjnym dla komputerów PC wyposażonych w procesory 386, 486 lub Pentium. Został opracowany na początku lat dziewięćdziesiątych przez Linusa Torvaldsa, a dalej jest rozwijany przez programistów całego świata.

System operacyjny to program zarządzający sprzętem komputerowym i oprogramowaniem użytkownika. Współdziałamy z nim poprzez interfejs użytkownika. Ten interfejs pozwala systemowi operacyjnemu na przyjmowanie i interpretowanie instrukcji wydanych przez użytkownika. Zarządzanie plikami, zarządzanie programami i współdziałanie z użytkownikiem to zadania wszystkich systemów operacyjnych. Linux, podobnie jak wszystkie wersje Unixa, ma dwie dodatkowe cechy. Jest systemem wielodostępnym i wielozadaniowym. Jako system wielozadaniowy pozwala na wykonywanie kilku zadań w tym samym czasie – gdy jedno zadanie jest wykonywane, można pracować nad następnym. Na przykład można edytować nowy plik już wtedy, gdy drukuje się edytowany wcześniej. Jako system wielodostępny pozwala na logowanie się do systemu kilku użytkownikom w tym samym czasie, a każdy użytkownik współdziała z nim poprzez swój własny terminal. Podobnie jak Unix Linux może być podzielony na cztery główne części:

- jądro,
- powłokę,
- system plików,
- programy użytkowe.

Jądro jest programem, który uruchamia inne programy i zarządza urządzeniami, takimi jak dyski czy drukarki.

Powłoka jest interfejsem użytkownika. Odbiera polecenia wydawane przez użytkownika i wysyła je do jądra w celu wykonania.

System plików określa porządek, w jakim są one zachowane na urządzeniu takim jak dysk. Pliki są umieszczane w katalogach. Każdy katalog może zawierać dowolną liczbę podkatalogów, w których przechowywane są pliki.

Programy użytkowe są specjalizowanymi programami, takimi jak edytory, kompilatory i programy komunikacyjne, które wykonują standardowe operacje. Można również tworzyć własne programy użytkowe.

Architektura systemu Linux przedstawiona jest na rysunku 1. Jądro, powłoka i system plików stanowią razem podstawową strukturę systemu operacyjnego.

Dzięki tym trzem elementom można uruchamiać programy, zarządzać plikami i współdziałać z systemem. Ponadto Linux ma programy użytkowe, które zostały dołączone po to, by zapewnić standardowe cechy systemu.

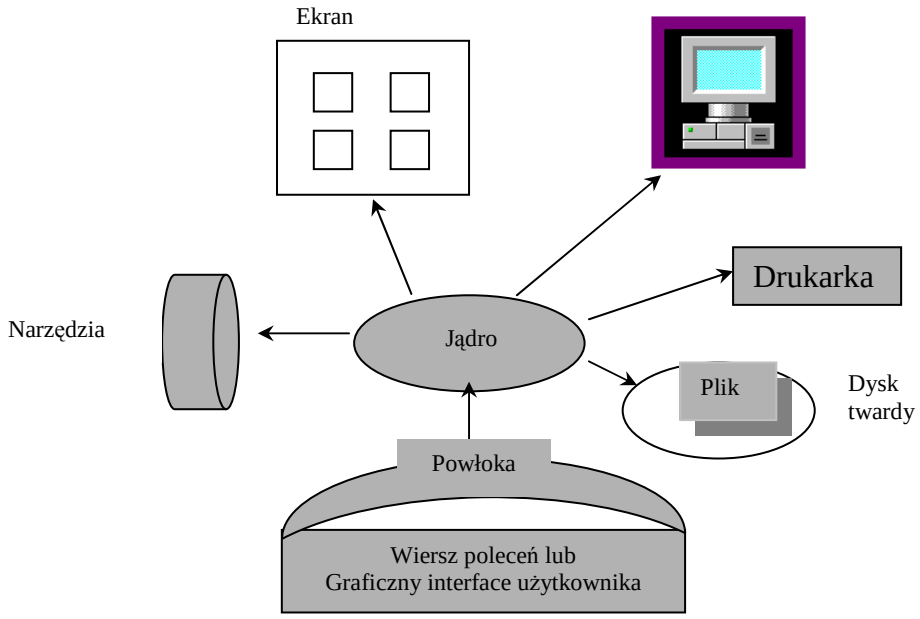
Powłoka jest jakby interpreterem. Interpretuje polecenia wprowadzane przez użytkownika i przekazuje je do jądra systemu. Interfejs powłoki jest bardzo prosty. Składa się z monitu, po którym należy wprowadzić polecenie i zaakceptować klawiszem [ENTER]. Alternatywą dla interfejsu wiersza poleceń jest interfejs graficzny, zwany X-Windows. Posiada on kilka okienkowych programów zarządzania, z których można korzystać. Powłoka to nie tylko interpretowanie poleceń. Udostępnia także środowisko do konfigurowania systemu i programowania. Ma ona również swój własny język programowania, który pozwala na pisanie programów wykonujących polecenia Linuxa. Każdy użytkownik w systemie Linux ma swój własny interfejs. Aktualnie istnieją trzy główne powłoki: Bourna, Korna i C. Domyślną powłoką użytkownika Linuxa jest Bourne Again, popularnie zwana BASH. Powłoka BASH zapewnia specjalne możliwości edycji wiersza poleceń. Można łatwo modyfikować wprowadzone polecenia, poruszając się po całym wierszu za pomocą ← →. [BACKSPACE] usuwa znak znajdujący się przed kursorem. Powłoka BASH przechowuje listę zwaną listą historii, która zawiera poprzednio wprowadzone polecenia. W jednym wierszu można umieścić więcej niż jedno polecenie, należy je wówczas oddzielić ;. Można także wprowadzać polecenie w kilku wierszach, wpisując (\) zanim naciśniemy [ENTER]. Znak ten maskuje [ENTER] i pozwala na kontynuowanie wprowadzania polecenia w następnym wierszu.

Powłoka BASH ma również funkcje uzupełniania zdarzeń historii wywoływane przez [ESC+TAB]. Podobnie jak przy standardowym uzupełnianiu wiersza poleceń wprowadzamy tę część zdarzenia historii, którą chcemy. Następnie naciskamy klawisz [ESC], a za nim [TAB]. Zdarzenie pasujące do wprowadzonego tekstu zostanie znalezione i wykorzystane do uzupełnienia wiersza poleceń. Jeśli więcej niż jedno zdarzenie historii pasuje do tego co wprowadzamy, usłyszymy dźwięk i możemy wprowadzić więcej znaków, by pomóc w jednoznacznej identyfikacji zdarzenia.

1.2. KONTA UŻYTKOWNIKÓW

Nigdy użytkownik nie dostaje się do systemu bezpośrednio. Linux oferuje interfejs, przez który użytkownik może z nim współpracować. Linux potrafi uruchomić kilka takich interfejsów jednocześnie, przyjmując kilku użytkowników w tym samym czasie. Konkretnemu użytkownikowi wydaje się, że jest jedyną osobą pracującą w systemie. Takie interfejsy użytkownika zwykle są nazywane kontami. Aby uzyskać dostęp do systemu, trzeba mieć skonfigurowany swój własny interfejs. Administrator systemu tworzy w systemie konto, nadając mu

nazwę użytkownika i hasło. Swoje konto użytkownik wykorzystuje do logowania się i korzystania z systemu.



Rys. 1. Jądro, powłoka i interfejs użytkownika

Użytkownik uprzywilejowany (root) posiada specjalne konto przeznaczone dla zadań administrowania systemem. Każde konto jest identyfikowane przez nazwę użytkownika i zabezpieczone hasłem. Wpisywane hasło nie pojawia się na ekranie. Jeśli zostanie wprowadzona błędnie nazwa lub hasło, na ekranie pojawi się komunikat błędu i zaproszenie do ponownego zalogowania. Kiedy nazwa i hasło zostaną poprawnie wprowadzone, użytkownik jest zalogowany w systemie. Pojawi się wówczas monit wiersza poleceń, którym może być znak $. Dla użytkownika root monitem jest znak #. Monit może być poprzedzony nazwą hosta i nazwą katalogu, w którym jesteśmy.

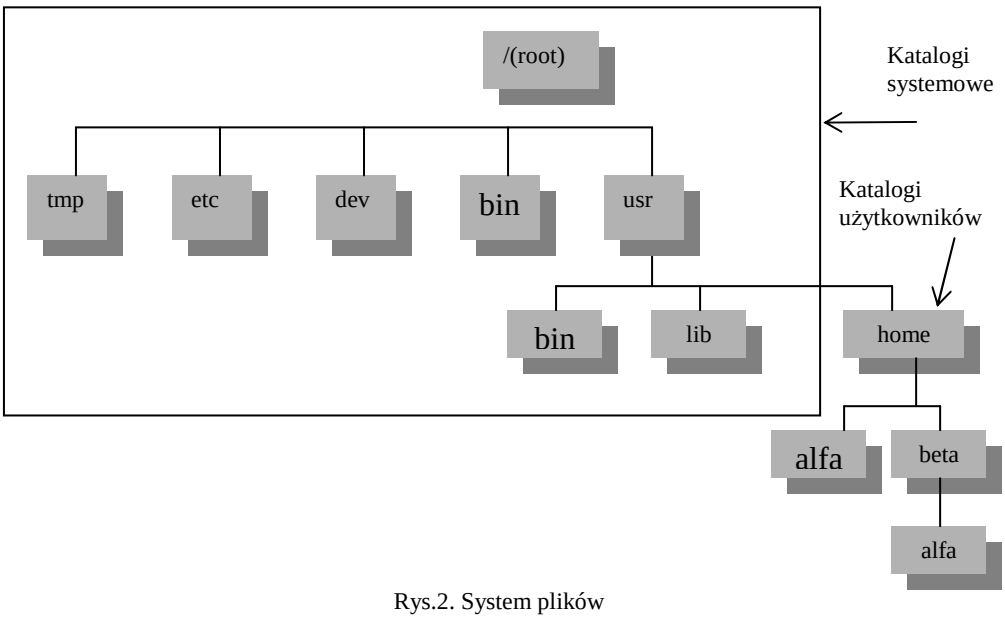
Swoje hasło użytkownik może zmienić poleceniem **`passwd`**. Pojawi się pytanie o aktualne hasło. Po poprawnym wpisaniu go użytkownik zostanie spytany o nowe hasło. Po wprowadzeniu tegoo hasła system zażąda jego powtórzenia. Polecenie **`passwd`** rejestruje w systemie nowe hasło. Teraz do logowania należy używać nowego hasła:

```
passwd
old password:
new password:
retype new password:
```

Po zakończeniu pracy użytkownik musi się wylogować. Aby zakończyć sesję, należy wydać polecenie **`logout`** lub **`exit`** albo [CTRL]+d. Powoduje to powrót do monitu logowania, a system operacyjny czeka, aż zaloguje się następna osoba.

1.3. CHARAKTERYSTYKA SYSTEMU PLIKÓW

W Linuxie pliki są umieszczone w katalogach, podobnie jak ma to miejsce w DOS-ie. Katalogi z kolei są ze sobą hierarchicznie powiązane w jedną strukturę plików. Nazwa pliku może składać się z liter, cyfr oraz niektórych znaków specjalnych, np. znaku podkreślenia. Jej długość może wynosić do 256 znaków. Linux rozróżnia duże i małe litery. Kropka nie posiada specjalnego znaczenia, ponieważ linux traktuje kropkę jak każdy inny znak. Pliki, których nazwy rozpoczynają się kropką są „ukryte". Traktowane są jak wszystkie inne pliki, z wyjątkiem tego, iż nie wyświetli ich polecenie **`ls`**, chyba że użytkownik zastosuje opcję –a. Plik .**profile** jest przykładem pliku ukrytego. Wszystkie pliki mają jeden format fizyczny – ciąg bajtów zakończony znakiem EOF (End-Of-File; Ctrl-D). Ten hierarchiczny system plików jednolicie traktuje pliki zwykłe, katalogi oraz pliki opisujące urządzenia zewnętrzne. Katalogi każdego użytkownika są w rzeczywistości połączone z katalogami pozostałych użytkowników. Są one ułożone w strukturę hierarchiczną drzewa (rys.2), rozpoczynając od katalogu głównego (root), będącego korzeniem.



Rys.2. System plików

Wszystkie pozostałe katalogi wychodzą od tego pierwszego. Katalog główny / jest to korzeń drzewa. Niektóre katalogi są standardowymi katalogami zarezerwowanymi do użytku przez system. Katalogi systemowe zawierają pliki i programy używane do uruchomienia i utrzymywania systemu.

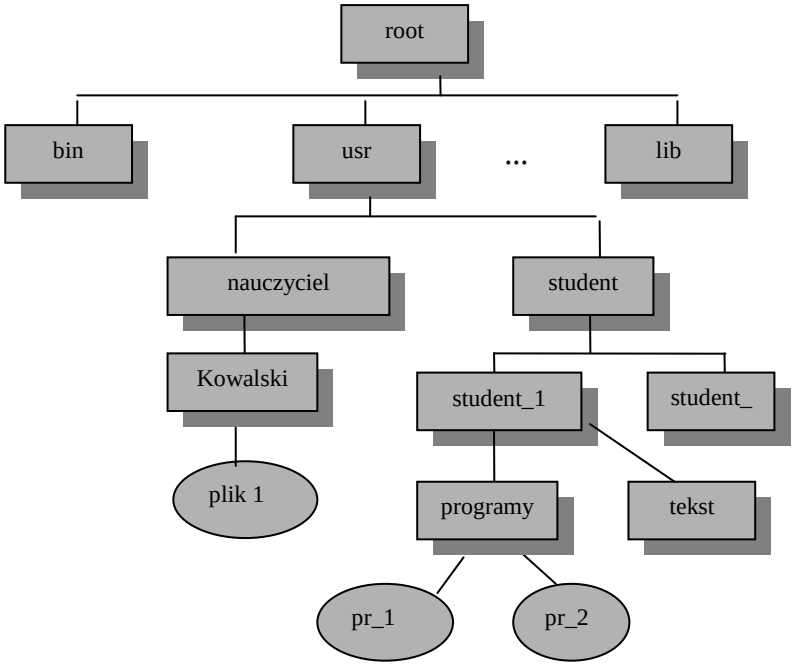
Znaczenie poszczególnych katalogów systemowych jest następujące:

- /bin, /usr/bin – zawierają większość komend systemowych i programy usługowe w wersji binarnej,
- /dev – mieszczą się w nim pliki specjalne, reprezentujące urządzenia rzeczywiste (dyski twarde, elastyczne, drukarki) i pseudourządzenia (konsola systemowa, obszar swap, wirtualny terminal),
- /etc – przechowywana jest w nim większość plików i programów umożliwiających konfigurację systemu,
- /tmp – używany przez komendy, jak i przez użytkowników do przechowywania plików tymczasowych,
- /home – katalog przeznaczony na katalogi domowe użytkowników systemu,
- /proc – wirtualny system plików, zawierający informacje o systemie i uruchomionych procesach.

Przy rejestrowaniu nazwy użytkownika w systemie zostaje z nią związany katalog osobisty danego użytkownika, oznaczany symbolem **~**. Katalog ten staje się katalogiem bieżącym w chwili rozpoczynania przez użytkownika sesji przy terminalu. Każda nazwa pliku, którą podaje wówczas użytkownik, np. jako parametr polecenia, odnosi się do plików z katalogu bieżącego. Każdy plik można znaleźć rozpoczynając poszukiwanie od katalogu pierwotnego, tj. od korzenia drzewa katalogów.

Jeśli katalogiem bieżącym jest w danym momencie katalog **student_1**, a chcemy wydać polecenie dostępu do pliku **plik1** w katalogu **Kowalski**, to powinno ono zawierać argument **../../nauczyciel/Kowalski/plik1,** przy czym **..** wskazują katalog nadrzędny. Powtórzone dwa razy wskazują na katalog **usr** i z niego rozpocznie się poszukiwanie pliku **plik1** przez katalogi **nauczyciel** i **Kowalski.**

W każdym katalogu (oprócz root) występuje element domyślny, stanowiący nazwę katalogu nadrzędnego, oznaczony dwiema kropkami **(..)**, oraz inny element, stanowiący nazwę katalogu, w którym on sam się znajduje, oznaczony **(.)**. Innymi słowy, każdy katalog wskazuje na samego siebie oraz na swój katalog nadrzędny. Rysunek 3 przedstawia przykładowy system plików.



Rys.3. Przykładowy system plików

1.4. ZNAKI SPECJALNE NAZW PLIKÓW: *, ?, [ ]

Nazwy plików są najczęściej argumentami stosowanymi w poleceniach. Często można znać jedynie część nazwy pliku lub chcieć odwołać się do kilku nazw plików. Powłoka oferuje zestaw znaków specjalnych, które poszukują, dopasowują i generują listę plików. Tymi znakami specjalnymi są gwiazdka, znak zapytania i nawiasy kwadratowe. Znaki ***** i **?** określają niepełną nazwę pliku, nawiasy kwadratowe pozwalają na określenie zestawu dopuszczalnych znaków, jakie mają być poszukiwane. Można łączyć nawiasy kwadratowe z innymi znakami specjalnymi.

- **Gwiazdka *** oznacza dowolny ciąg znaków, można jej używać do oznaczania nazw plików zaczynających się lub kończących dowolnym zestawem znaków.
- **Znak zapytania ?** oznacza dokładnie jeden dowolny znak. Można używać więcej niż jednego znaku zapytania w każdym miejscu wzorca.
- Dowolne znaki umieszczone w nawiasach kwadratowych **[ ]** oznaczają dokładnie jeden spośród wymienionych znaków.

Znaki specjalne mogą być maskowane za pomocą znaku „\”. Aby nazwa pl? odnosiła się do dokładnie jednego pliku pl?, a nie do całej grupy plików o 3-literowych nazwach rozpoczynających się znakami pl, należy za pomocą „\” zamaskować znaczenie symbolu „?” – a więc zastosować nazwę **pl\?.**

Przykłady nazw zawierających metaznaki

*.c – pliki, które mają rozszerzenie c,

dok? – pliki, których nazwy rozpoczynają się od słowa „dok”, za którym następuje jeden dowolny znak,

dok[1x] – pliki, których nazwy rozpoczynają się od „dok”, a kończą się znakiem 1 lub x,

dok[A-D] – pliki, których nazwy rozpoczynają się od „dok”, a kończą się literą A, B, C lub D,

***** – wszystkie pliki z wyjątkiem ukrytych,

.* – wszystkie pliki ukryte.

Jeśli w katalogu bieżącym są pliki: pl1?.c, pl2?.o, pl13?.o, pl4.o, to polecenie **`ls pl[1-3]\?.[co]`** spowoduje wyświetlenie nazw: pl1?.c, pl2?.o.

2. PODSTAWOWE KOMENDY LINUXA

Formatem polecenia jest nazwa polecenia, za którą następują opcje, a dopiero za nimi argumenty:

nazwa polecenia **opcje** **argumenty**

System Linux oferuje zestaw poleceń wykonujących podstawowe operacje zarządzania plikami, takie jak listowanie, wyświetlanie i drukowanie, kopiowanie, zmiana nazwy itp. Użytkownik ma prawo do tworzenia i usuwania katalogów oraz zmiany katalogu roboczego. Często używa również najprostszych poleceń:

`pwd` – wyświetla bezwzględną ścieżkę do katalogu bieżącego użytkownika,

`clear` – czyści ekran, ustawia monit w lewym górnym rogu ekranu,

`cal` – umożliwia wyświetlenie kalendarza na bieżący miesiąc albo podany miesiąc i rok,

`who` – wyświetla listę zalogowanych aktualnie użytkowników, ich terminali, godziny zalogowania oraz nazwę komputera, z którego się logowali,

`who am i` – umożliwia wyświetlenie na ekranie danych o sobie,

`touch plik` **–** tworzy pusty **plik,**

`lpr plik` – drukuje **plik** na drukarce.

Pliki są umieszczane w kolejce i drukowane po jednym w tle. Można drukować kilka plików, np. polecenie **`lpr plik1 plik2`** umożliwia wydrukowanie zawartości plików o nazwach **plik1** oraz **plik2**.

W trakcie drukowania pozycję drukowania można zobaczyć w dowolnej chwili dzięki poleceniu **`lpq`**. Podaje ono ID zadania, rozmiar w bajtach i plik tymczasowy, w którym jest ono aktualnie przechowywane. Jeśli użytkownik chce usunąć zadanie, musi skorzystać z polecenia:

`lprm nazwa zadania`

2.1. WYŚWIETLANIE ZAWARTOŚCI KATALOGU ROBOCZEGO

Polecenie **`ls`** służy do listowania plików i katalogów w katalogu roboczym. Nie wyświetla plików ukrytych. Polecenia **`ls`** można używać z różnymi opcjami:

`-a` – wypisuje wszystkie pliki, razem z plikami ukrytymi,

`-l` – podaje wszystkie dane (pełną informację) o plikach i katalogach,

`-F` – dopisuje po każdym katalogu ukośnik, gwiazdkę po nazwach plików wykonywalnych i znak @ po nazwach plików powiązanych,

`-R` – wypisuje wszystkie zagnieżdżone podkatalogi, umieszczone poniżej katalogu roboczego (rekurencyjne wyświetlanie podkatalogów wraz z zawartością),

`-i` – wypisuje numer i-węzła dla każdego pliku,

`-s` – podaje dodatkowo rozmiary pliku,

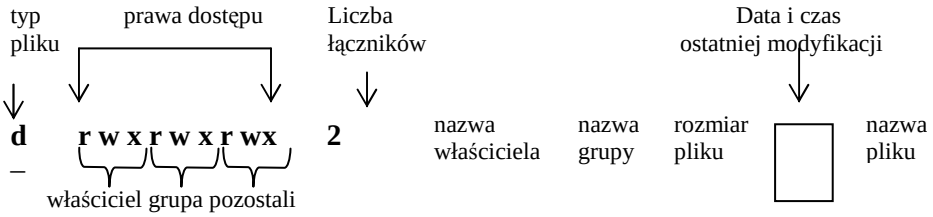
`-S` – sortuje wg wielkości plików,

`-t` – sortuje pliki wg daty ostatniej modyfikacji; nowsze pliki znajdą się na początku,

`-u` – sortuje wg daty ostatniego dostępu,

`-X` – wyświetla pliki posortowane według rozszerzeń.

Efektem wykonania komendy **`ls -l`** jest wyświetlenie zawartości danego katalogu w pełnej postaci. Każda linijka opisuje jeden plik i wygląda następująco:



Znak pierwszy określa typ pliku. Jeśli jest to znak „–”, plik jest plikiem zwykłym, jeśli jest to litera „d”, będzie to plik specjalnego rodzaju, jakim jest katalog. Oprócz plików zwykłych i katalogów mogą występować inne rodzaje plików, które zostały omówione w rozdziale 5.

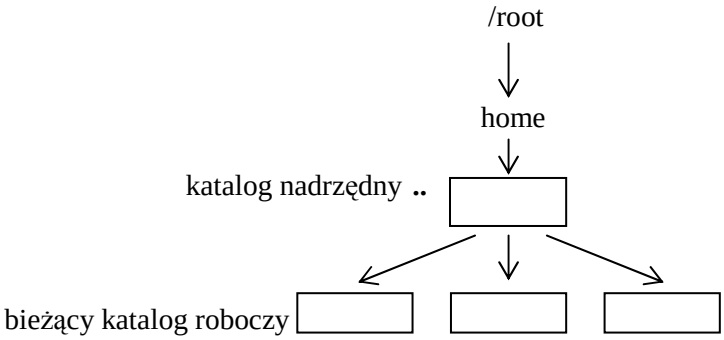
2.2. OPERACJE NA KATALOGACH

Katalogi można zmieniać za pomocą polecenia **`cd`**. Zmiana ta powoduje, że dany katalog staje się katalogiem roboczym. Po zalogowaniu się do systemu katalogiem roboczym jest katalog domowy użytkownika (~). Kiedy tworzone jest konto użytkownika, system tworzy również katalog domowy (home directory) dla danego użytkownika. Polecenie **`cd`** przyjmuje jako argument nazwę katalogu, do którego chcemy przejść:

`cd nazwa katalogu`

Można również zmienić katalog na inny, używając jego pełnej ścieżki. Symbolu dwóch kropek **..** można używać do przedstawienia katalogu nadrzędnego. Dosłownie oznaczają one jego ścieżkę. Polecenie **`cd ..`** pozwala na przejście użytkownika do katalogu nadrzędnego. Jeśli chcemy powrócić do katalogu domowego, wystarczy wprowadzić samo polecenie **`cd`** bez argumentu.

Każdy katalog ma katalog nadrzędny (z wyjątkiem katalogu korzenia). Kiedy katalog jest tworzony, wykonywane są dwa wpisy: jeden przedstawiony jako kropka **.** , drugi jako dwie kropki **..** . Jedna kropka oznacza nazwę ścieżki katalogu, a dwie – nazwę ścieżki katalogu nadrzędnego:



Do plików i katalogów można odwoływać się używając bezwzględnych i względnych nazw ścieżek.

Ścieżka względna rozpoczyna się od katalogu roboczego, jest prosta w użyciu, korzystamy z niej gdy tylko jest to możliwe.

Ścieżka bezwzględna jest długa, skomplikowana, używamy jej tylko wtedy, gdy musimy. Aby ścieżkę tę uprościć (od katalogu korzenia do katalogu domowego użytkownika), można używać znaku specjalnego **tyldy ~** , który oznacza bezwzględną ścieżkę katalogu macierzystego.

Przykład

`cat ~/nazwa pliku` – odwołanie do pliku z katalogu home

Katalogi można tworzyć poleceniem:

`mkdir opcje katalogi`

Aby jednym poleceniem utworzyć zagnieżdżoną strukturę katalogów, należy zastosować opcję **`-p`**. Polecenie **`mkdir -p kat1/kat2`** tworzy katalog **kat1** z podkatalogiem **kat2**. Opcja **`-m prawa dostępu`** pozwala na stworzenie katalogu z określonymi prawami dostępu.

2.3. PRZEGLĄDANIE ZAWARTOŚCI PLIKÓW

Polecenie:

`cat opcje nazwa pliku`

wyświetla całą treść pliku na ekranie. Dostępne opcje:

`-b` – numerowanie niepustych wierszy,

`-n` – numerowanie wszystkich wierszy,

-s – zastąpienie kilku kolejnych pustych wierszy jednym,
-v – dodatkowe drukowanie znaków kontrolnych.

Problem pojawia się wówczas, gdy plik jest duży, ponieważ polecenie nie umożliwia stronicowania. Można obejrzeć tylko ostatni ekran tekstu. Ograniczenia te mogą zostać pokonane dzięki poleceniom **more** i **less**, które wyświetlają treść pliku po jednej stronie:

more [**opcje**] [**nazwa pliku**]

Dostępne opcje:

+# – rozpoczęcie wyświetlania od wiersza o numerze #,
-l – ignorowanie znaków wysuwu strony,
-s – zakaz wyświetlania kilku sąsiednich pustych wierszy.

Klawisz **Enter** przewija zawartość o jeden wiersz w dół, a klawisz **spacji** umożliwia przejście do następnej strony. Klawisz **h** wyświetla pomoc do wszystkich poleceń, **b** pozwala na cofnięcie się do tyłu o pełny ekran, **q** natomiast przerywa wyświetlanie, pozwala na wyjście z polecenia **more**.
Polecenie:

less [**nazwa pliku**]

jest bardzo podobne do **more**, wyświetla treść pliku i umożliwia przeglądanie go za pomocą strzałek ↑, ↓ .
Polecenie:

head [**opcje**] [**pliki**]

powoduje wyświetlenie pierwszych 10 wierszy podanych plików. Gdy jest wiele plików, nazwa pliku poprzedza jego zawartość na wyjściu. Dostępne opcje:

-cn[b|k|m] – wyświetlenie pierwszych **n** bajtów, bloków (**b**), kilobajtów (**k**) albo megabajtów (**m**),
-# – zmiana liczby wierszy do wyświetlenia,
-q – zakaz wypisywania nazwy pliku.

Polecenie:

tail [**opcje**] [**pliki**]

powoduje wyświetlenie ostatnich 10 wierszy podanego pliku. Dostępne opcje:

-cn[b|k|m] – wyświetlenie ostatnich **n** bajtów, bloków (**b**), kilobajtów (**k**) albo megabajtów (**m**),
-n – wyświetlenie ostatnich n wierszy,
-v – wyświetlenie nazwy pliku w wierszu tytułowym.

Polecenie:

wc opcje pliki

powoduje zliczanie liczby znaków, słów i wierszy w pliku tekstowym. Dostępne opcje:

-c – tylko liczba znaków,
-l – tylko liczba wierszy,
-w – tylko liczba słów.

Polecenie:

cut opcje pliki

umożliwia wycięcie serii pól lub kolumn z wiersza pliku wejściowego. Musi być podana jedna z opcji **-b**, **-c** albo **-f**. W każdej z tych opcji trzeba podać listę, która może zawierać liczby oddzielone przecinkami albo pola zdefiniowane przez znaki "**-**".

Dostępne opcje:

-b lista – wybranie znaków z pozycji podanych na liście,
-c lista – wybranie kolumn podanych na liście,
-f lista – wybranie pól (oddzielonych tabulatorami albo znakami separatorów) z listy,
-dc – używane razem z **-f** do podania znaku separatora pól, którym jest **c**.

Przykład

Polecenie **cut -d: -f1,3 /etc/passwd** – wypisuje nazwy i identyfikatory wszystkich użytkowników.

Polecenie:

sort opcje pliki

umożliwia sortowanie wierszy podanych plików. Dostępne opcje:

+n-m – ustawienie klucza sortowania między polami n i m,
-b – ignorowanie spacji na początku wiersza,
-c – sprawdzenie, czy podane pliki nie są już posortowane. Jeżeli tak, to program kończy się komunikatem błędu,
-d – ignorowanie znaków interpunkcyjnych w czasie sortowania,
-f – ignorowanie małych/wielkich liter,

-i – ignorowanie niedrukowalnych znaków ASCII,

-m – połączenie dwóch plików wejściowych,

-n – sortowanie numeryczne,

-o plik – przeadresowanie standardowego urządzenia wyjściowego do pliku,

-r – odwrócenie kolejności sortowania,

-t znak – podanie znaku separatora kolumn (domyślnie spacja albo tabulacja),

-u – usunięcie powtarzających się wierszy.

Przykład

sort +2n -t: /etc/passwd – sortuje plik **passwd** numerycznie względem trzeciej kolumny.

2.4. ZMIANA NAZWY, PRZENOSZENIE PLIKÓW

Nazwę pliku można zmienić poleceniem:

mv opcje stara nazwa nowa nazwa

Opcja –**i** (zabezpieczenie przed nadpisaniem) sprawdza, czy plik o nowej nazwie istnieje. Jeśli tak, pojawia się pytanie o nadpisanie pliku:

mv -i stara nazwa nowa nazwa

Jeśli operacja **mv** wykonywana jest pomiędzy katalogami, plik zostaje przeniesiony do podanego katalogu:

mv opcje nazwa pliku nazwa katalogu/nowa nazwa

Dostępne opcje:

-b – utworzenie kopii zapasowej pliku przed zniszczeniem jego zawartości,

-f – brak pytania o potwierdzenie przed zniszczeniem zawartości plików,

-u – przeniesienie plików tylko wtedy, kiedy są nowsze niż pliki docelowe o tej samej nazwie.

Można przenosić nie tylko pliki zwykłe, ale również całe katalogi. Polecenie:

```
mv katalog1 katalog2
```

powoduje, że **katalog1** przeniesiony zostanie jako podkatalog katalogu **katalog2**.

2.5. KOPIOWANIE PLIKÓW

Polecenie **cp** tworzy kopię pliku źródłowego najpierw przez utworzenie pliku, a następnie skopiowanie do niego danych. Polecenie ma następującą postać:

```
cp opcje plik źródłowy plik docelowy
```

Jeśli więc istnieje plik o tej nazwie, zostaje nadpisany:

```
cp -i plik źródłowy plik docelowy
```

Opcja **-i** powoduje, że polecenie **cp** najpierw sprawdzi, czy plik docelowy już istnieje. Jeśli tak, użytkownik zostanie zapytany, czy nadpisać istniejący plik. Dostępne inne opcje:

-b – utworzenie kopii plików przed zniszczeniem ich zawartości,

-f – wymuszenie kopiowania i zniszczenia zawartości istniejących plików,

-l – utworzenie twardego dowiązania zamiast kopiowania,

-s – utworzenie dowiązania symbolicznego zamiast kopiowania,

-v – wyświetlenie nazwy każdego pliku w czasie kopiowania,

-r – kopiowanie całych katalogów wraz z podkatalogami,

-p – kopiowanie plików do katalogu docelowego z zachowaniem hierarchii podkatalogów.

Aby skopiować plik do innego katalogu niż roboczy, należy jedynie użyć nazwy katalogu jako drugiego argumentu. Aby skopiować pliki **plik1** oraz **plik2** do katalogu **katalog**, należy wykonać komendę:

```
cp opcje plik1 plik2 katalog
```

Jeśli chcemy przekopiować plik do innego katalogu ze zmianą jego nazwy, używamy polecenia:

```
cp opcje nazwa pliku nazwa katalogu/nowa nazwa pliku
```

Można również kopiować i przenosić całe katalogi. Polecenia **cp** oraz **mv** przyjmują nazwę katalogu jako pierwszy argument. Aby skopiować katalog, polecenie **cp** wymaga opcji **–r**. Powoduje to rekursję, czyli kopiowanie katalogów wraz z podkatalogami i ich zawartością:

```
cp -r katalog1 katalog2
```

Katalog1 jest kopiowany jako podkatalog katalogu **katalog2**

2.6. USUWANIE PLIKÓW

Polecenie **rm** umożliwia usuwanie plików. Po słowie kluczowym **rm** wymienia się nazwy plików, które należy wymazać. Można usuwać kilka plików jednocześnie:

```
rm opcje plik1 plik2
```

Dostępne opcje:

–f – usuwa pliki bez żadnego uprzedzenia,
–r – pozwala na usuwanie katalogu ze wszystkimi podkatalogami i plikami,
–i – żąda potwierdzenia dla każdego usuwanego pliku,
–v – wyświetla nazwy każdego kasowanego pliku.

Polecenie to jest nieodwracalne, nie można odtworzyć pliku po jego usunięciu. Aby zabezpieczyć się przed tego rodzaju sytuacją, należy używać opcji **–i**, pojawia się wówczas pytanie o potwierdzenie usunięcia każdego z plików:

```
rm -i plik1 plik2
```

Pusty katalog, czyli katalog nie zawierający żadnych plików ani podkatalogów, można usunąć poleceniem:

```
rmdir katalog
```

natomiast polecenie:

```
rm -r katalog
```

usuwa katalog ze wszystkimi podkatalogami. Najpierw usuwa całą zawartość katalogu, a następnie sam **katalog**.

2.7. ĆWICZENIA

1. Utwórz katalog **nauka**, a w nim podkatalogi **alfa**, **beta**, **gama**. W katalogu beta utwórz podkatalog **grupa1**, a w katalogu **gama** podkatalogi **grupa2** oraz **grupa3**.
2. Odszukaj pliki o nazwie **gcc.** Wybierz jeden z nich i przekopiuj do katalogu **gama** pod nazwę **mojgcc.**
3. W katalogu **grupa2** utwórz plik **tekst**, wpisując do niego nazwę wydziału oraz kierunek studiów.
4. Obejrzyj powstałą strukturę katalogów.
5. Przenieś katalog **gama** do katalogu **alfa** i przekopiuj do niego plik **tekst** pod nazwę **mojgcc**. Co stanie się z istniejącym plikiem?
6. Usuń katalog **beta**.
7. Usuń plik **tekst** z katalogu **grupa2**.
8. Usuń katalog **gamma** z potwierdzeniem usuwanych plików.
9. Usuń jednym poleceniem całą pozostałą strukturę katalogów utworzoną uprzednio.
10. Zaproponuj polecenie, które wylistuje nazwy plików w katalogu nadrzędnym w stosunku do katalogu domowego.
11. Wyjaśnij, kiedy polecenie `cp a b c` jest poprawne i jaki będzie skutek jego wykonania.

3. STANDARDOWE WEJŚCIE/WYJŚCIE, POTOKI, PRZEKIEROWANIA

3.1. STANDARDOWE WYJŚCIE – PRZEKIEROWANIA

Wszystkie pliki Linuxa są logicznie zorganizowane jako ciągły strumień bajtów. Fizycznie natomiast pliki są ułożone jako rozrzucone bloki w pamięci dyskowej. Poza wywołaniami systemowymi użytkownik nigdy nie odwołuje się do fizycznej struktury pliku. Dla użytkownika wszystkie pliki mają identyczną organizację – strumienia bajtów. Każdemu procesowi w chwili uruchomienia przypisane są automatycznie standardowe strumienie: wejściowy i wyjściowy. Dane wejściowe z klawiatury są umieszczane w strumieniu danych jako ciąg bajtów. Dane wyprowadzane z polecenia czy programu są również umieszczane w strumieniu danych w postaci ciągu bajtów. Wejściowy ciąg danych jest w Linuxie nazwany standardowym wejściem (KLAWIATURA), a wyjściowy (EKRAN) – standardowym wyjściem.

Standardowe wejście lub wyjście użytkownik może zmienić dzięki przekierowaniom, np. można skierować standardowe wyjście do pliku, a nie na ekran. Aby to było możliwe, należy umieścić operator przekierowania > i nazwę pliku w wierszu poleceń.

Polecenie **`ls`** wyświetla na ekranie pliki znajdujące się w bieżącym katalogu. Standardowe wyjście można przekierować do pliku np**. `ls >alfa`,** wówczas wynik polecenia **`ls`** zamiast na ekranie znajdzie się w pliku o nazwie **alfa**. Jeśli plik o nazwie **alfa** (docelowy) istnieje, to zostaje nadpisany, jeśli nie istnieje, to zostanie utworzony.

Do przekierowania można także używać operatora >>**.** Polecenie **`ls >>alfa`** powoduje dopisanie do pliku **alfa** (gdy plik o takiej nazwie istnieje) wyniku polecenia **`ls`**. Jeśli plik **alfa** nie istnieje, to zostanie utworzony. Operator przekierowania jest wykonywany wcześniej niż zasadnicza komenda. Polecenie:

`cat plik1 plik2 > plik3`

można wykorzystać do łączenia plików; **plik1** zostaje połączony z plikiem **plik2** i zapamiętany w pliku **plik3.**

3.2. ĆWICZENIA

1. Wykonaj komendę **`ls -l`**.
2. Sprawdź czy plik o nazwie **dane** istnieje, jeśli tak, usuń istniejący plik.
3. Wykonaj komendę **`ls -l > dane`**.
4. Sprawdź jaka jest zawartość pliku **dane**: czy jest tam zawarty również opis pliku **dane**?
5. Jaki był rozmiar pliku **dane** przed wykonaniem komendy **`ls -l`** (odczytaj z pliku **dane**)?
6. Jaka jest teraz zawartość i wielkość pliku **dane**?
7. Wykonaj komendę **`ls -i >> dane`**.
8. Sprawdź jaka jest zawartość pliku **dane**.
9. Usuń plik **dane**.
10. Wykonaj komendę **`ls -i >> dane`**.
11. Sprawdź jaka jest zawartość pliku **dane**: czy jest tam zawarty również opis pliku **dane**?
12. Wykonaj komendę **`cat dane > dane`**.
13. Co się stało z plikiem **dane**? Sprawdź jaki jest rozmiar i zawartość pliku **dane**.
14. Wyjaśnij dlaczego tak się stało.

3.3. STANDARDOWE WEJŚCIE – PRZEKIEROWANIA

Tak jak standardowe wyjście można również przekierować standardowe wejście. Może być ono odbierane z pliku, a nie z klawiatury. Operatorem przekierowania standardowego wejścia jest znak mniejszości **<.**

Polecenie **`cat<alfa`** powoduje, iż standardowe wejście jest przekierowane tak, żeby pobierać dane z pliku **alfa**, a nie z klawiatury. Zawartość pliku **alfa** jest odczytywana ze standardowego wejścia przez operację przekierowania. Następnie polecenie **`cat`** odczytuje standardowe wejście i wyświetla zawartość pliku **alfa**.

Można łączyć operacje przekierowań zarówno standardowego wejścia, jak i standardowego wyjścia. Komenda **`cat<alfa>beta`** powoduje, że polecenie **`cat`** odbiera dane ze standardowego wejścia i wysyła je na standardowe wyjście. Jednak standardowe wejście jest przekierowane tak, by dane były odbierane z pliku **alfa**, a standardowe wyjście tak, by dane znalazły się w pliku **beta**.

Kiedy wykonujemy jakieś polecenie, istnieje możliwość wystąpienia błędu. System zgłasza wówczas komunikat o błędzie, który jest wyświetlany na ekranie, czyli standardowym wyjściu. Podobnie jak standardowe wyjście czy wejście można również przekierować standardowe wyjście błędów. Dzięki temu komunikaty o błędach mogą nie pojawiać się na ekranie, ale zostaną zapisane do pliku. Do standardowych operacji strumieni można odwołać się przez ich numery:

0 – standardowe wejście (klawiatura),
1 – standardowe wyjście (monitor),
2 – standardowe wyjście błędów (monitor).

Domyślne przekierowanie wyjścia błędu (**2**) działa na standardowym wyjściu (monitorze), czyli **1**. Jeśli chcemy wyświetlić zawartość pliku **test**, który nie istnieje, np. **`cat test`**, komunikat o błędzie pojawi się na monitorze.

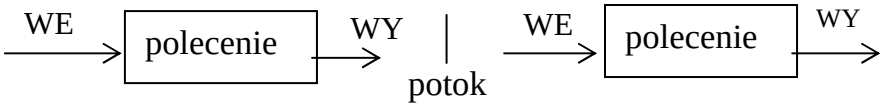
Komenda **`cat test 2>błąd`** spowoduje przekierowanie komunikatu o błędach na standardowy strumień określony cyfrą **2**, czyli standardowe wyjście błędów. Komunikat o błędzie pojawi się w pliku **błąd**.

W powłoce BASH można odwołać się do strumienia standardowego przez poprzedzenie znakiem **&** jego numeru, np. **&1** odnosi się do standardowego wyjścia. Operacja **2>&1** przekierowuje standardowe wyjście błędów na standardowe wyjście. Polecenie **`cat test 1>dane 2>&1`** ma jako argument nazwę nieistniejącego pliku. Wynikający z tego komunikat o błędzie jest przekierowywany do pliku **dane**. Zarówno wartość standardowego wyjścia, jak i standardowego wyjścia błędów zostanie zachowana w tym samym pliku.

Ten sam efekt można uzyskać wydając polecenie **`cat test >& dane.`** Aby rozdzielić strumień wyjściowy i strumień błędów, zapisując je do oddzielnych plików, można użyć polecenia: **polecenie 1>wyniki 2>błąd.**

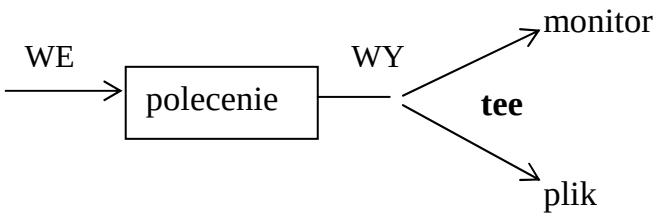
3.4. POTOKI

Mogą mieć miejsce takie sytuacje, w których chcemy przesłać standardowe wyjście do innego polecenia, a nie do pliku wynikowego. Na przykład chcemy wydrukować na drukarce listę nazw plików. Aby wykonać to zadanie, potrzebujemy wykorzystać dwa polecenia: **`ls`** do stworzenia listy nazw plików oraz polecenia **`lpr`** do wysłania tej listy na drukarkę. Wynik polecenia **`ls`** musimy wykorzystać jako dane wejściowe dla polecenia **`lpr`**. Aby takie połączenie wykonać, trzeba wykorzystać tzw. **potok**. Komenda, która to umożliwia będzie miała postać: **`ls|lpr`.** Operator potoku | (**pionowa kreska**), umieszczony między poleceniami, tworzy połączenie między nimi:



Operację potoku można łączyć z innymi cechami powłoki, takimi jak znaki specjalne, np. polecenie **`ls *.c|lpr`** powoduje wydrukowanie na drukarce nazw wszystkich plików, które mają rozszerzenie c, czyli plików źródłowych języka C.

Gdy chcemy przekierować standardowe wyjście do pliku, a równocześnie wyświetlić wyprowadzoną zawartość na ekranie, musimy skorzystać z polecenia **`tee`,** które rozgałęzia strumień wyjściowy. Polecenie **`tee`** kopiuje standardowe wyjście do pliku. Jako argument przyjmuje nazwę nowego pliku, do którego kopiowane jest standardowe wyjście:



3.5. ĆWICZENIA

1. Utwórz plik o nazwie **proba,** składający się przynajmniej z 5 linii tekstu.
2. Wyświetl posortowaną zawartość pliku **proba** na ekranie i zapisz do pliku **prsort**.

3. Wyświetl zawartość katalogu nadrzędnego względem katalogu home. Strumień wyjściowy tego polecenia skieruj na wejście polecenia **more**. Co otrzymasz na monitorze?
4. Policz ilość znaków, słów oraz linii, jakie zostaną wyświetlone na monitorze po wykonaniu polecenia **who**.
5. Zawartość pliku **proba** wyświetl na ekranie oraz skopiuj do pliku **gamma.**
6. Do pliku **pl1** przekieruj polecenie **ls -i**, natomiast do pliku **pl2** polecenie **ls -s**.
7. Zapisz w pliku **wynik** posortowane połączone pliki **pl1** oraz **pl2** z numeracją wierszy.
8. Zapisz w pliku **wynik** oraz wyświetl na ekranie posortowane pliki **pl1** oraz **pl2** z dołączoną numeracją wierszy.
9. Używając przekierowania dołącz plik **pl1** do pliku **pl2**.
10. Zaproponuj polecenie tworzące plik **pl3**, będący połączeniem plików **pl1** i **pl2**.

4. POLECENIA SŁUŻĄCE DO PORÓWNYWANIA ORAZ SZUKANIA PLIKÓW

4.1. PORÓWNYWANIE PLIKÓW

Polecenie:

cmp opcje plik1 plik2

służy do porównywania dwóch plików bajt po bajcie. Wskazuje on dokładnie, w którym miejscu (jeśli w ogóle występuje) znaleziono pierwszą różnicę. Polecenie **cmp** może także wypisywać różnice znak po znaku. W przypadku identycznych plików nie pojawia się żaden komunikat. Jeśli nie chcemy, by były wyświetlane szczegółowe informacje, a jedynie kod zakończenia programu, należy użyć opcji **–s**.

Kod zakończenia:

0 – pliki są takie same,
1 – pliki różnią się,
2 – dostęp do plików zakończył się niepowodzeniem.

Polecenie **cmp** nadaje się przede wszystkim do porównywania plików binarnych.

Dostępne opcje:

-c – wyświetlenie różniących się znaków,
-l – wyświetlenie przesunięć i wartości ósemkowych różniących się bajtów.

Polecenie:

diff opcje plik1 plik2

Polecenie **diff** porównuje dwa pliki, będące argumentami i wyświetla różnice w formie informacji edycyjnych, umożliwiających modyfikacje jednego pliku na drugi.

Dostępne opcje:

-u – powoduje, że różnice wyświetlane są w formacie GNU unified,
-c – wyświetla 3-wierszowy kontekst wokół każdej różnicy.

Aby utworzone poleceniem **diff** informacje edycyjne można było wykorzystać do odtwarzania poprzedniej wersji pliku oraz przechodzenia do nowszej, standardowe wyjście polecenia **diff** trzeba przekierować do pliku, np.

```
diff –u plik1 plik2>łata
```

Do odtworzenia pliku na podstawie tak uzyskanego pliku różnicowego (łaty) służy polecenie **patch** z przeadresowanym strumieniem wejściowym następującej postaci:

```
patch < łata
```

Aby przywrócić poprzednią zawartość pliku (zdjąć łatę), należy zastosować polecenie **patch** z opcją –**R: patch -R< lata**.

Należy zwrócić uwagę na fakt, że polecenie **patch** nie wymaga żadnych parametrów, nazwa pliku do modyfikacji zawarta jest w pliku różnicowym.

4.2. WYSZUKIWANIE W PLIKU CIĄGU ZNAKÓW

Polecenie **grep** jest uniwersalnym programem przeznaczonym do wyszukiwania w pliku wierszy zawierających określony wzorzec. Dane ze strumienia wejściowego czytane są wiersz po wierszu i znajdywane są wiersze zawierające podany ciąg znaków. Uniwersalność programu **grep** bierze się z możliwości tworzenia złożonych wzorców za pomocą znaków uogólniających. Wzorzec do polecenia **grep** konstruuje się między innymi w następujący sposób:

Kropka „.” – oznacza dowolny pojedynczy znak,
[a-z] – literę z podanego przedziału,
***** – zero lub więcej znaków.

Można również określić położenie wzorca w wierszu:

Znak ^ – na początku wzorca symbolizuje początek linii tekstu,

Znak ^ – po otwierającym nawiasie kwadratowym pełni rolę zaprzeczenia,
Znak $ – na końcu wzorca symbolizuje koniec linii tekstu.

Przykłady wzorców:

abc – wypisz linijki zawierające łańcuch „abc”,
^abc – wypisz linijki zaczynające się od „abc”,
abc$ – wypisz linijki kończące się na „abc”,
a..c – wypisz linijki zawierające „a” i „c”, przedzielone dwoma dowolnymi znakami,
a*c – wypisz linijki zawierające „a” i „c”, przedzielone dowolnymi znakami,
^[Cc] – wskaż wiersze mające w pierwszej kolumnie znaki C lub c,
^[^Cc] – wskaż wszystkie wiersze nie mające w pierwszej kolumnie znaku C lub c.

Jeśli wzorzec zawiera odstępy lub znaki specjalne, należy ująć go w apostrofy. Jeśli znak specjalny ma być użyty we wzorcu jako znak zwykły, wtedy należy poprzedzić go znakiem „ \”:

```
grep opcje wzorzec plik
```

Dostępne opcje:

-i – ignoruje wielkość liter przy porównywaniu tekstu ze wzorcem,
-v – wypisuje te linie, które nie zawierają podanego wzorca,
-c – podaje tylko ilość linijek odpowiadających wzorcowi,
-n – przed każdą linijką odpowiadającą wzorcowi podaje jej numer.

Jeśli szukany wzorzec jest alternatywą, należy użyć operatora „|”, który maskowany jest znakiem \.

4.3. ĆWICZENIA

1. Utwórz plik **tekst** następującej treści:
   System operacyjny LINUX
   to system wielodostępny.
   System ten jest również
   wielozadaniowy.

2. Wyświetl wszystkie linijki zawierające ‘system’ przy rozróżnieniu dużych i małych liter.
3. Wyświetl wszystkie linijki zawierające ‘**system**’ ignorując wielkość liter.
4. Ile linijek zawiera wyraz ‘**system**’ (ignorując wielkość liter)?

5. Wyświetl wszystkie linijki nie zawierające '**system**' ignorując wielkość liter.
6. Wyświetl wszystkie linijki zaczynające się od '**Sy**'.
7. Wyświetl wszystkie linijki kończące się na 'y'.
8. Wyświetl wszystkie linijki, w których '**oper**' poprzedza '**Sys**'.
9. Odszukaj w katalogu **/usr/include** w plikach nagłówkowych (pliki z rozszerzeniem h) ciąg znaków '**pow**'.
10. Wyświetl te linie, które zawierają słowo '**to**' lub '**ten**'.
11. Rozważ następujące polecenia:

```
grep ^[^d-] plik
grep '...x' plik
grep –v '^[cC]' plik.f > plikbk.f
ls –la ..|grep user1
grep '\.$' plik
grep '.$' plik
grep 'int\|}' *.c
```

4.4. WYSZUKIWANIE PLIKÓW

Kiedy w różnych katalogach mamy wiele plików, zachodzi potrzeba umiejętnego ich wyszukiwania. Służą do tego polecenie **find, whereis, which**. Polecenie:

whereis nazwa pliku

podaje ścieżki dostępu do plików o nazwie **plik**. Natomiast komenda:

which nazwa pliku

podaje ścieżkę dostępu do pliku, który jest wykonywany po wydaniu polecenia nazwa pliku. Powyższe polecenia umożliwiają wyszukiwanie plików po ich nazwach. Bardziej uniwersalne jest polecenie **find**, które pozwala na wyszukiwanie plików wg różnych kryteriów:

find katalog startowy szukania **opcje** **kryterium**

Dostępne opcje:

-name – szukanie wg nazwy,

-type – szukanie wg typu. Wymagany jest jednoznakowy argument, którym jest jeden ze znaków:

d – katalog,

f – plik zwykły,
b – plik specjalny blokowy,
c – plik specjalny znakowy,
s – semafor,
l – link symboliczny.

`-size` rozmiar – szukanie wg rozmiaru pliku w blokach – **b**, znakach – **c**, słowach –**w** lub kilobajtach –**k**, np. **`-size +100c`** – szukane są pliki o rozmiarze większym niż 100 znaków, natomiast **`-size –100w`** – szukane są pliki o rozmiarze mniejszym niż 100 słów.

`-mtime` czas – szukanie wg liczby dni, jakie minęły od ostatniej modyfikacji, np. **`-mtime +3`** – szuka plików modyfikowanych więcej niż 3 dni temu, natomiast **`-mtime -3`** – pozwala na szukanie plików modyfikowanych mniej niż 3 dni temu.

`-atime` czas – szukanie wg liczby dni, jaka minęła od ostatniego dostępu.

`-user` użytkownik – szukanie plików, których właścicielem jest użytkownik.

`-perm` prawa_dostępu – szukanie wg praw dostępu pliku podanych w formie 3 lub 4 cyfr z przedziału <0-7>. Polecenie **`-perm 100`** – pozwala na odnalezienie plików, które mają ustawione przynajmniej prawo x dla właściciela.

`-exec polecenie` – powoduje wykonanie podanego po **`exec`** polecenia na odnalezionych plikach. Polecenie to jednak musi być zakończone następującym ciągiem znaków **{}** **;** np.

`find /  -name '*.c' -exec cat {} \;`

`-ok` polecenie – powoduje wykonanie polecenia podanego po **`ok`** na odnalezionych plikach. Wymaga jednak dodatkowo potwierdzenia każdego z nich, np. **`find / -name '*.c' -ok cat {} \;`**

`-newer` plik1 – szukanie plików modyfikowanych później niż plik **plik1**.

Przy poszukiwaniu można korzystać również z operatorów logicznych OR, NOT, AND. Operator NOT polecenia **`find`** zapisuje się w postaci znaku „**!**” wykrzyknika. Wykrzyknik umieszczony przed kryterium poszukiwania neguje to kryterium, np. polecenie **`find . ! -name 'at'`** pozwala odszukać wszystkie pliki mające nazwy różne od **at**. Operator logiczny OR zapisujemy –**`o`** , natomiast operator AND jako **`–a,`** np. polecenie **`find . -name 'at' –o –type d`** pozwala na wyszukanie plików o nazwie **at** lub typie **d**. Jeśli plik spełnia jedno lub

drugie kryterium, jest uznawany za pasujący do wzorca. Kiedy kilka opcji zostanie podanych w wierszu poleceń, tworzą one operację AND.

Polecenie **`find / -name 'at' >p1`** rozpoczyna poszukiwania od głównego katalogu i szuka plików o nazwie **at**, a następnie zapisuje rezultat poszukiwań w pliku **p1** (pełną nazwę każdego znalezionego pliku).

4.5. POMOC

Wszystkie polecenia opisane są w podręczniku elektronicznym. Można z niego skorzystać za pomocą komendy **`man`**, z nazwą polecenia, o którym chcemy uzyskać informacje jako parametr. Dokumenty elektroniczne dotyczące poleceń mogą być podzielone na różne poziomy, zaczynając od pierwszego, np. **`man 8 crontab`** (wyświetlenie dokumentacji 8 poziomu na temat **`crontab`**). Program **`man`** ma własny zestaw poleceń, które uruchamia się wciskając pojedyncze klawisze. Naciśnięcie [**spacji**] powoduje przejście do następnej strony, natomiast klawisza [**b**] powrót do strony poprzedniej. Po zakończeniu czytania należy nacisnąć klawisz [**q**], aby zakończyć pracę z programem **`man`** i powrócić do wiersza poleceń.

Program **`man`** ma możliwość przeszukiwania, którą uruchamia się ukośnikiem **[/]** lub **[?]** w zależności od tego, czy szukamy w przód, czy do tyłu. Na dole ekranu pojawi się wiersz, w którym można wpisać szukane słowa. Możliwe jest powtórzenie poszukiwania przez naciśniecie klawisza [**n**].

Polecenia **`whatis, aproppos`** oraz **`info`** służą do przeszukiwania bazy danych tytułów dokumentacji elektronicznej i wyświetlają każdy znaleziony tytuł wraz z krótkim opisem. **`Whatis`** poszukuje tytułów przez znalezienie całych słów. Polecenie **`apropos`** działa tak samo jak **`whatis`**, z tą różnicą, że poszukiwane są wzorce, a nie całe słowa.

5. SYSTEM PLIKOWY

System plików służy do tworzenia środków przeznaczonych do organizowania danych i korzystania z nich w sposób wygodny dla użytkownika. Użytkownik systemu ogólnego przeznaczenia może przechowywać dane w plikach o dowolnych rozmiarach. Każdy plik jest zbiorem danych, które użytkownik traktuje jako pewną całość – może to być katalog, program, zbiór procedur albo wyników eksperymentu. Plik jest jednostką logiczną, na której system plików wykonuje pewne operacje. Nośnik, na którym przechowuje się pliki, bywa zazwyczaj

podzielony na bloki o ustalonej długości, a system plików musi przydzielać odpowiednie liczby bloków dla każdego pliku.

W systemie LINUX jeden logiczny system plików widziany przez użytkownika może składać się z kilku fizycznych systemów plików znajdujących się na różnych urządzeniach. Z kolei każde urządzenie fizyczne może być podzielone na urządzenia logiczne, a każde z nich może definiować własny system plików.

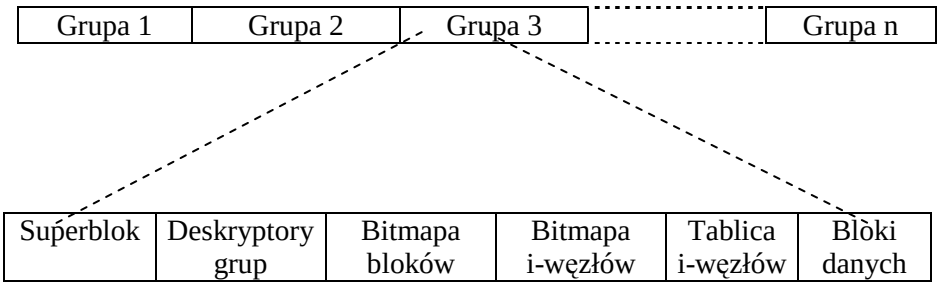
System plików jest „prawą ręką" systemu operacyjnego. Umożliwia logiczny podział urządzeń służących do przechowywania danych (dyski twarde, dyskietki, CD-ROM-y) w celu zbudowania struktury mogącej w sposób szybki i bezpieczny odwoływać się do plików określonych przez użytkownika.

Odwołania do pliku realizowane są przez funkcje systemowe, które zasłaniają przed programistą wewnętrzną budowę określonego systemu plików. Linux obsługuje rekordową ilość systemu plików od rdzennego ext2, poprzez vfat, iso9960 i wiele innych. Aby połączyć tę różnorodność w jedną całość - zastosowano tzw. VFS, wirtualny system plików, który umożliwia dostęp do każdego pliku poprzez zastosowanie tych samych funkcji systemowych. Moduł wirtualnego systemu plików decyduje, gdzie jest umieszczony określony plik i zarządza dostępem w sposób całkowicie niewidoczny dla użytkownika.

Głównym korzeniem jest /, od niego niczym gałęzie rozprzestrzeniają się odnogi. Zastosowanie takiego hierarchicznego systemu plików umożliwia zasłonięcie jego fizycznych struktur. Dzięki zamontowaniu każdego systemu plików w katalogu użytkownik nawet nie zdaje sobie sprawy, że porusza się w systemie, którego wewnętrzna struktura jest całkowicie niewidoczna.

5.1. MACIERZYSTY SYSTEM PLIKÓW EXT2

Macierzystym systemem plików Linuxa jest system ext2. Logiczna budowa tego systemu została pokazana na rysunku 4.



Rys.4. Macierzysty system plików Linuxa ext2

System etx2 jest podzielony na wiele grup. Każda grupa przechowuje dane, które są szczególnie ważne dla całego systemu plików. Są to: kopie superbloku

oraz deskryptory wszystkich grup. Poprzez ten zabieg otrzymujemy system plików, który jest w dużym stopniu odporny na awarie. Oprócz przechowywania wspólnych danych w każdej grupie znajduje się mapa bitowa zajętości bloków, mapa bitowa zajętości i-węzłów, tablica i-węzłów oraz bloki z danymi.

Podczas montowania systemu plików odczytywany jest tylko superblok pierwszej grupy. Jeżeli uległ on awarii, możemy spróbować odczytać kopię superbloku umieszczoną w grupie 2. Jednak jeśli i ta część bloku została uszkodzona, pozostaje nam szansa na odczytanie superbloku z innych grup. Superblok jest w ten sposób chroniony, ponieważ stanowi bardzo newralgiczną część systemu plików.

Z punktu widzenia użytkownika plik jest identyfikowany przez nazwę ścieżkową. Na jej podstawie system operacyjny musi zlokalizować bloki dyskowe, czyli miejsce, w którym plik jest przechowywany na dysku. Pliki zwykłe zawierają dane, natomiast katalogi to pliki binarne zawierające listę plików (w tym także innych katalogów), które się w nim znajdują. Każda pozycja w katalogu opisuje jeden plik i stanowi parę informacji złożoną z nazwy pliku oraz jego numeru i-węzła. Jest to mechanizm spajający plik opisywany przez dany i-węzeł z jego położeniem w drzewie katalogowym. Sam plik nie zawiera więc żadnych informacji o tym, w jakim miejscu drzewa katalogowego jest zlokalizowany, a w katalogu nie są zawarte żadne informacje na temat atrybutów pliku (przeciwnie niż w systemach plikowych opartych na tablicy FAT). Wszystkie atrybuty pliku oraz adresy bloków danych znajdują się w i-węźle pliku, który jest podstawą lokalizacji bloków dyskowych. Wszystkie i-węzły mają unikalne numery (w ramach jednego fizycznego urządzenia w jednym systemie plikowym). W i-węźle przechowywane są następujące informacje o pliku:

- typ pliku. W Linuxie występują następujące typy plików:
  - **-** – plik zwykły,
  - **d** – katalog,
  - **p** – łącze nazwane FIFO,
  - **b** – plik specjalny blokowy,
  - **c** – plik specjalny znakowy,
  - **l** – link symboliczny,
  - **s** – gniazdo (semafor),
- identyfikator właściciela oraz grupy pliku,
- prawa dostępu,
- rozmiar pliku w bajtach,
- ostatni czas dostępu, modyfikacji,
- czas utworzenia i skasowania,
- liczba dowiązań,
- liczba bloków dyskowych zajmowanych przez plik,
- adresy dyskowe.

Pole opisujące rozmiar pliku ma długość 32 bity. Największa liczba binarna 32-bitowa to 2^{32}-1, zatem maksymalny teoretyczny rozmiar pliku to 4GB. W kolejnym rozdziale poznamy kolejne ograniczenie.

5.2. SPOSÓB ADRESOWANIA BLOKÓW DYSKOWYCH

Adresy dyskowe umieszczone w i-węźle wskazują na bloki danych. W i-węźle przechowywanych jest 12 adresów bezpośrednich, jeden adres bloku pojedynczo pośredniego oraz po jednym adresie dla bloków podwójnie i potrójnie pośrednich. Sposób adresowania bloków dyskowych podaje rysunek 5.

Chcąc odczytać dane z pliku jądro najpierw musimy odczytać i-węzeł pliku, a dopiero później bloki dyskowe z danymi. Do przechowywania adresów bloków dyskowych służy tablica składająca się z 15 elementów, która jest umieszczona w i-węźle. Pierwszych 12 adresów to adresy (numery) bloków bezpośrednich. Zakładając, że rozmiar bloku wynosi 1024 bajty, przez takie adresowanie możemy mieć dostęp bezpośrednio do 12 · 1024 (rozmiar bloku) danych. Co należy zrobić, gdy plik ma np. 100KB. Jak zaadresować kolejne części pliku? Służą do tego pozostałe trzy wpisy w tablicy adresów. Rekord 13 jest adresem bloku pojedynczo pośredniego, następne dwa rekordy to adresy bloków podwójnie i potrójnie pośrednich.

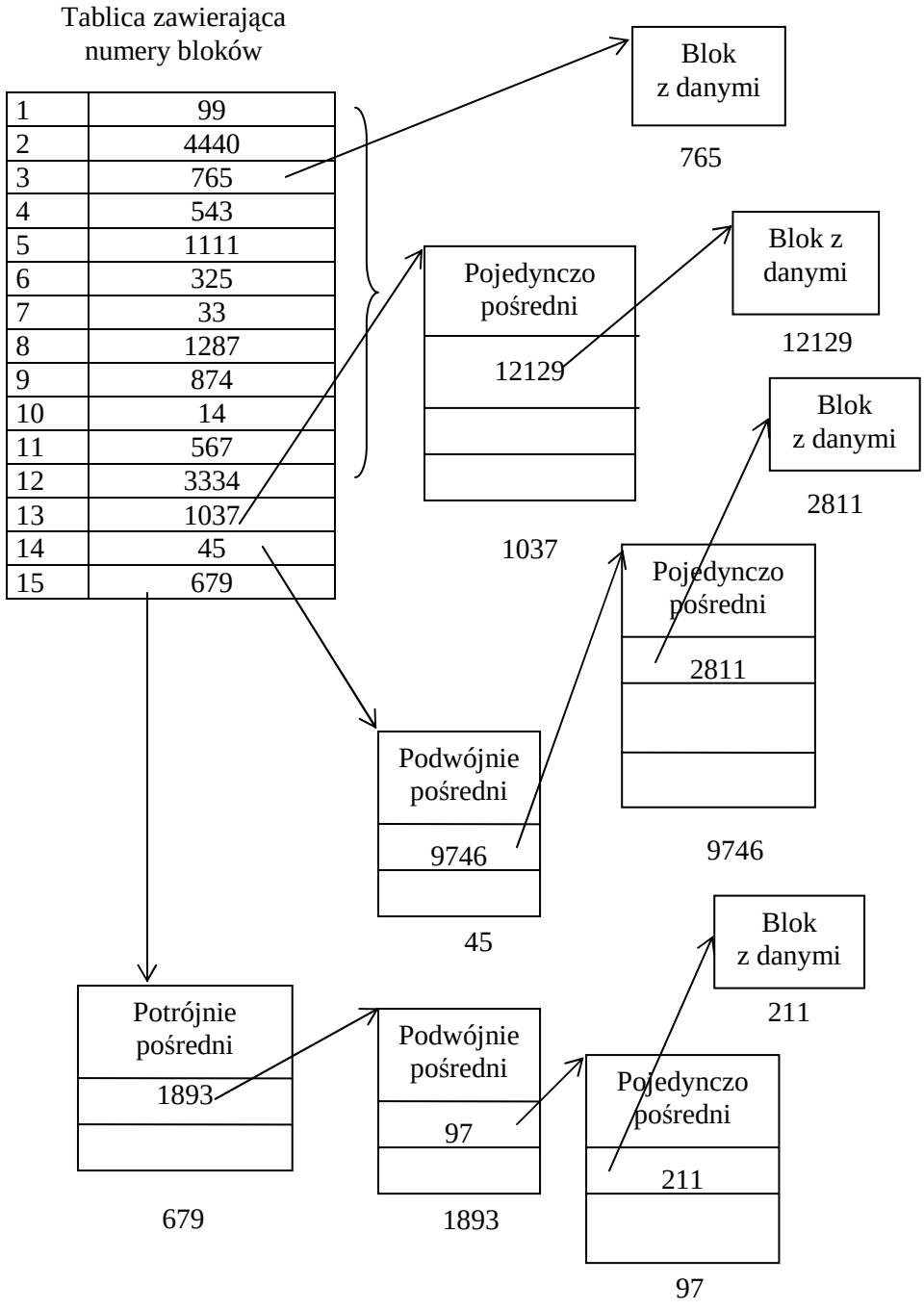
Blok pojedynczo pośredni to taki blok, który nie zawiera danych pliku lecz, przechowuje 1024/4=256 następnych adresów bloków dyskowych. Podobne informacje przechowywane są w blokach podwójnie i potrójnie pośrednich. W bloku podwójnie pośrednim przechowywane są adresy bloków pojedynczo pośrednich, a w bloku potrójnie pośrednim adresy bloków podwójnie pośrednich.

Należy zwrócić uwagę, że dostęp do pliku małego (12 · rozmiar bloku) jest bardzo szybki, natomiast do dużych plików znacznie wolniejszy.

Przykład

Oblicz ilość bloków dyskowych oraz maksymalny rozmiar pliku, jeśli blok ma rozmiar 128 bajtów.

W pierwszych 12 adresach przechowywane są numery bloków bezpośrednich. Poprzez adresowanie bezpośrednie możemy zaadresować do 12 bloków. W adresie 13 przechowywany jest adres bloku pojedynczo pośredniego. Dzięki niemu możemy zaadresować (rozmiar bloku/4) bloki dyskowe. W rozważanym przykładzie będzie to: 128/4=32 bloki dyskowe.



Rys.5. Sposób adresowania bloków dyskowych

W adresie 14 przechowywany jest adres bloku podwójnie pośredniego. Dzięki niemu możemy zaadresować (rozmiar bloku/4) bloki pojedynczo pośrednie. Te z kolei pozwalają zaadresować (rozmiar bloku/4) bloki dyskowe. Przez blok podwójnie pośredni można zaadresować: $32 \cdot 32 = 1024$ bloki dyskowe.

W adresie 15 przechowywany jest adres bloku potrójnie pośredniego. Dzięki niemu możemy zaadresować (rozmiar bloku/4) bloki podwójnie pośrednie. Te z kolei pozwalają zaadresować (rozmiar bloku/4) bloki pojedynczo pośrednie, które umożliwiają zaadresowanie (rozmiar bloku/4) bloki dyskowe. Przez blok potrójnie pośredni można zaadresować: $32 \cdot 32 \cdot 32 = 32768$ bloków dyskowych.

Maksymalna liczba bloków dyskowych, którą można zaadresować wynosi:

$$12 + 32 + 1024 + 32768 = 33836$$

Maksymalny rozmiar pliku można obliczyć mnożąc rozmiar bloku dyskowego przez maksymalną ilość bloków dyskowych:

$$128 \cdot 33836 = 4331008 \text{ bajtów}$$

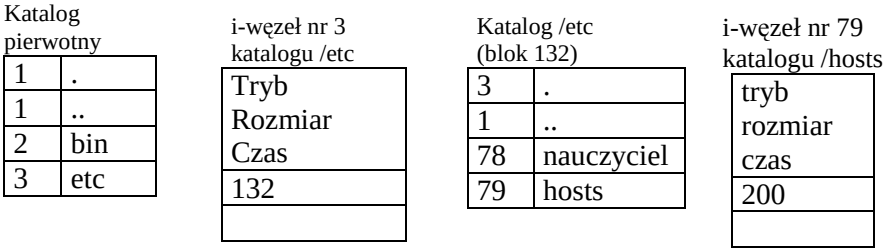
5.3. ZAMIANA NAZWY ŚCIEŻKOWEJ NA I - WĘZEŁ

Jądro w swojej wewnętrznej strukturze nie posługuje się nazwami plików, lecz rozróżnia je za pomocą i-węzła. Oznacza to, że nazwa ścieżkowa musi zostać zamieniona na odpowiedni i-węzeł. Zamiana rozpoczyna się od bieżącego katalogu lub od katalogu głównego. Jeśli jądro chce odczytać np. plik **/etc/hosts**, musi najpierw odczytać i-węzeł korzenia plików, ponieważ nazwa zaczyna się od /, co ilustruje rysunek 6.

Po odczytaniu i-węzła sprawdzane są prawa dostępu oraz typ pliku. Jeśli jest to katalog, to wczytywany jest blok lub bloki z danymi (zależy to od wielkości katalogu). Blok lub bloki są przeszukiwane w celu znalezienia nazwy **etc**. Numery bloków z danymi znajdują się w strukturze i-węzła.

Po znalezieniu łańcucha znaków **etc** jądro odczytuje ze struktury katalogu (ext2_dir_entry) skojarzony z nazwą **etc** i-węzeł. W tym momencie możemy już odczytać i-węzeł katalogu **etc**.

Kolejnym krokiem jest odczytanie bloków dyskowych, które zawierają dane o zawartości katalogu **etc.** Następnie w odczytanych blokach poszukiwany jest ciąg znaków **hosts**. Jeśli taka nazwa zostanie znaleziona, to pobierany jest i-węzeł. W tym momencie jądro zakończyło przekształcanie nazwy ścieżkowej w i-węzeł.



Rys.6. Sposób zamiany nazwy ścieżkowej na numer i-węzła

5.4. DOWIĄZANIA DO PLIKÓW – LINKI

Ważną i pożyteczną cechą systemu plikowego, a w szczególności ext2 są linki. Umożliwiają one nadanie wielu nazw jednemu plikowi. Rozróżniamy dwa rodzaje dowiązań:

- linki twarde,
- linki symboliczne.

W różnych częściach systemu możemy utworzyć linki, które będą wskazywały na jeden plik. Nie musimy w ten sposób tworzyć wielu kopii tego samego pliku i możemy zaoszczędzić miejsce na dysku. Linki twarde i symboliczne pełnią podobne funkcje, lecz różny jest mechanizm ich działania i nie zawsze mogą być one stosowane wymiennie.

5.4.1. LINKI TWARDE

Linki twarde umożliwiają tworzenie dwóch lub więcej nazw dla jednego i-węzła. Dotyczą one tego samego fizycznego pliku dyskowego, ale każdy z nich stanowi oddzielne wejście w odpowiednich katalogach. Można je utworzyć komendą **`ln`**:

`ln oryginalna nazwa dodatkowa nazwa`

Dodatkowe nazwy nazywane są łącznikami. Aby sprawdzić ile nazw ma plik, należy użyć polecenia **`ls – l`** i odczytać liczbę łączników. Poleceniem **`ls –i`** należy się upewnić jakie są numery i-węzłów. Numer i-węzła to unikalny numer używany przez system do identyfikacji konkretnego pliku. Jeśli dwie nazwy mają ten sam numer węzła, wówczas odnoszą się do tego samego pliku. Aby połączyć plik z katalogu bieżącego z innym katalogiem, należy użyć polecenia:

`ln nazwa pliku nazwa katalogu/nowa nazwa`

Polecenie **`ln plik linktw`** powoduje utworzenie w bieżącym katalogu nowego wejścia dla pliku o nazwie **linktw** z tym samym numerem węzła co plik. Link twardy – plik **linktw** jest również plikiem zwykłym i jest równorzędny z plikiem **plik**. Ponieważ numery i-węzłów w ramach danego urządzenia są niepowtarzalne, linki twarde mogą być tworzone jedynie w obrębie jednej partycji (w tym samym systemie plikowym). Link twardy można utworzyć tylko do pliku zwykłego.

5.4.1.1. ĆWICZENIA

1.Wykonaj komendy: **`ln plik lintw`** oraz **`ln lintw lintw3`**:
- porównaj numery i-węzłów plików: **plik**, **lintw**, **lintw2**,
- porównaj wszystkie atrybuty plików,
- sprawdź jak zmieniła się ilość dowiązań w plikach,
- porównaj zawartość wszystkich plików,
- zmień zawartość jednego z nich i sprawdź jak wygląda zawartość pozostałych,
- usuń plik o nazwie plik i zaobserwuj jak zmienia się ilość dowiązań w pozostałych plikach.

2.Sprawdź numery i-węzłów plików w komendach **`mv`** i **`cp`**:
- odczytaj numer węzła wybranego pliku, np. **plik**,
- wykonaj komendę **`mv plik zmiana`,**
- odczytaj numer i-węzła pliku **zmiana**,
- wykonaj komendę **`cp plik kopia`,**
- odczytaj numery i-węzłów obu plików.

Uwaga!

Komenda **`mv`** powoduje jedynie usunięcie z katalogu wejściowego pozycji opisującej plik, będący pierwszym parametrem, a następnie dodanie odpowiedniej pozycji w katalogu docelowym – nowa nazwa pliku i stary numer węzła. Tak się dzieje, gdy oba pliki należą do tego samego systemu plikowego. W przeciwnym razie numery i-węzłów są różne i odbywa się pełne kopiowanie bloków z danymi pomiędzy dyskami lub partycjami.

5.4.2. LINKI SYMBOLICZNE

Łączniki utworzone bez parametru **s** tworzą tzw. łączniki sztywne. Nie zadziałają one, kiedy próbujemy łączyć plik z plikiem w katalogu innych użytkowników, który jest zlokalizowany w innym systemie plików (system plików mogą tworzyć dowolne urządzenia pamięci fizycznych). Aby pokonać to ograniczenie, należy używać linków symbolicznych. Łącznik symboliczny przechowuje ścieżkę do pliku, z którym jest połączony, dzięki czemu może przekraczać fizyczne granice urządzeń. Nie jest to bezpośredni łącznik sztywny, ale raczej informacja o tym, jak zlokalizować konkretny plik. Łącznik symboliczny tworzymy poleceniem:

`ln - s ścieżka/nazwa nazwa łącznika`

Polecenie **`ln -s /home/ala/lista lunch`** - tworzy łącznik o nazwie **lunch** do pliku **home/ala/lista.** Łącznik symboliczny ma własny typ pliku, reprezentowany literą **l**. Litera **l** wskazuje, że jest to łącznik symboliczny, a nie zwykły plik. Rozmiar pliku jest mały, bo przechowuje ścieżkę innego pliku. Łączników symbolicznych można używać do tworzenia łączników z katalogami. Tworzymy więc inną nazwę, przez którą można odwoływać się do katalogu.

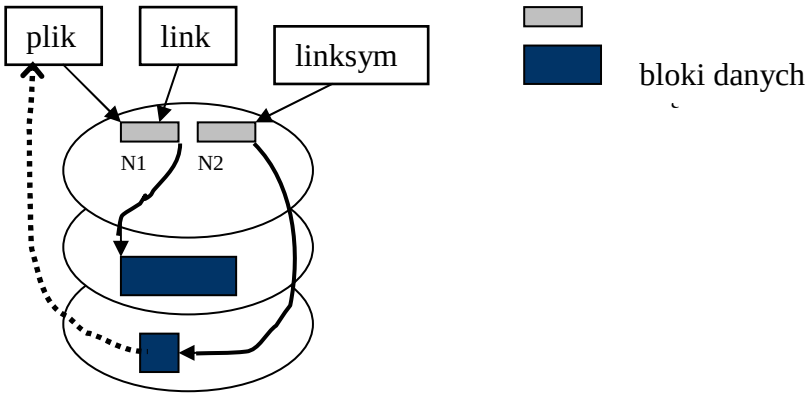
5.4.2.1. ĆWICZENIA

1. Wykonaj komendę **`ln –s plik linksymb`**:
   - porównaj atrybuty obu plików (typ, data, rozmiar, nazwa),
   - porównaj numery i-węzłów obu plików,
   - porównaj zawartości obu plików,
   - zmień plik **linksymb** (dodaj coś do niego),
   - sprawdź, czy zmienił się również **plik**,
   - sprawdź czy zmieniły się rozmiary obu plików,
   - usuń plik **plik**,
   - wylistuj zawartość pliku **linksymb**,
   - utwórz inny plik w tym samym katalogu o nazwie **plik**,
   - sprawdź, czy ma inny numer i-węzła,
   - wylistuj plik **linksymb**.
2. Wykonaj komendę **`ln –s /home drzwi`**:
   - sprawdź atrybuty pliku **drzwi**,
   - zastanów się, jaki powinien być efekt wykonania komendy **`ls drzwi`**, a następnie sprawdź, czy miałeś rację,

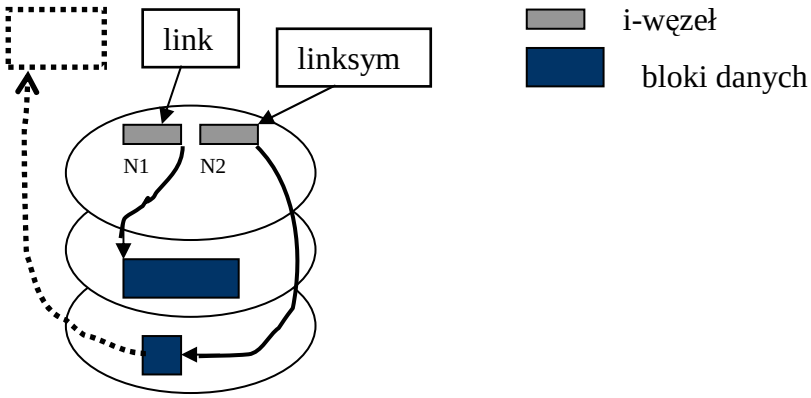
- zastanów się, jaki powinien być efekt wykonania komendy **`ls -l drzwi`**, a następnie sprawdź, czy miałeś rację,
- wykonaj komendę **`cd drzwi`**, a następnie **`ls`**,
- sprawdź komendą **`pwd`** bieżącą ścieżkę.

Organizacja na dysku dowiązań twardych i symbolicznych przedstawiona została na rysunku 7. Na dysku znajduje się plik **plik** identyfikowany przez numer i-węzła N1. W i-węźle pamiętane są adresy bloków dyskowych, które zajmuje. Gdy tworzymy link twardy **linkt** do pliku **plik,** liczba dowiązań do pliku zostaje zwiększona o 1. Link twardy ma ten sam numer i-węzła co plik **plik,** czyli N1, który wskazuje na bloki dyskowe pliku **plik**. Dowiązanie symboliczne ma własny numer i-węzła N2, na podstawie którego odczytujemy bloki dyskowe. Zawierają one ścieżkę dostępu do pliku oryginalnego i poprzez jego numer i-węzła (N1) możemy odczytać bloki dyskowe zajmowane przez plik. Po usunięciu pliku **plik** liczba dowiązań do pliku zostaje zmniejszona o 1. Zawartość pliku można jednak odczytać dzięki dowiązaniu twardemu **linkt**. Poprzez link symboliczny nie można odwołać się do pliku **plik**, ponieważ wskazuje on na ścieżkę do pliku, którego nie ma we wskazanym katalogu.

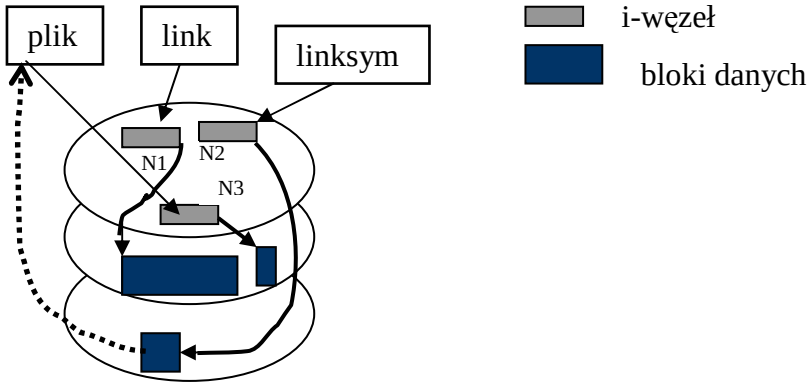
Jeśli utworzymy plik o takiej samej nazwie co usunięty uprzednio, to będzie miał nowy numer i-węzła (N3), a jego dane zapisane będą w innych blokach dyskowych. Poprzez link twardy sięgamy do pliku pierwotnego **plik**, natomiast poprzez link symboliczny do pliku **plik** utworzonego po usunięciu pliku oryginalnego.



Po usunięciu pliku **plik**



Po utworzeniu nowego pliku o nazwie **plik**



Rys. 7. Organizacja na dysku dowiązań twardych i symbolicznych

6. PRAWA DOSTĘPU DO PLIKÓW I KATALOGÓW

6.1. PRAWA DOSTĘPU DO PLIKÓW

Każdy plik i katalog posiada zestaw praw dostępu określających, kto ma dostęp do pliku i jakie ma prawa. Można je wyświetlić komendą **`ls -l`** (rozdział 2.1). Istnieją trzy kategorie użytkowników:

- właściciel pliku,
- grupa do której należy właściciel,
- pozostali użytkownicy systemu.

Prawa dostępu można nadawać plikom i katalogom. Każdy plik ma ściśle określone prawa dostępu stwierdzające, czy określony użytkownik jest uprawniony do odczytania lub zapisania pliku bądź do jego wykonania. Każdy użytkownik może mieć dowolną kombinację tych praw. Są one całkowicie niezależne i posiadanie jakiegokolwiek z nich nie jest warunkiem posiadania innego. W przypadku pliku prawa są interpretowane w następujący sposób:

- **`r`** – prawo pisania oznacza zezwolenie na modyfikację zawartości pliku,
- **`w`** – prawo czytania umożliwia oglądanie zawartości pliku, oznacza jednocześnie prawo do kopiowania,
- **`x`** – prawo do uruchomienia pliku wykonywalnego.

6.2. PRAWA DOSTĘPU DO KATALOGÓW

Te same kategorie praw **–** czytania, pisania i wykonywania odnoszą się do katalogów:

- **`w`** – prawo czytania umożliwia przeszukiwanie zawartości katalogu, jest interpretowane jako prawo wypisywania zawartości (komenda **`ls`**),
- **`r`** – prawo pisania daje możliwość modyfikowania zawartości katalogu, umożliwia dodawanie nowych oraz usuwanie dotychczasowych plików z katalogu,
- **`x`** – prawo wykonywania w stosunku do katalogu pozwala na dostęp do plików zapisanych w nim oraz na wejście do danego katalogu **–** uczynienie go katalogiem bieżącym (komenda **`cd katalog`**).

Prawa dostępu konieczne do wykonania poleceń zawarte są w poniższej tabelce.

WYKONYWANE CZYNNOŚCI	PLIK	KATALOG
	r w x	**r w x**
wejście do katalogu komenda **`cd`**	**- - -**	**- - x**
przeglądanie, jakie pliki znajdują się w katalogu komenda **`ls`**	**- - -**	**r- -**
przeglądanie, jakie pliki znajdują się w katalogu i jakie mają atrybuty komenda **`ls –l, ls –s, ls -F`**	**- - -**	**r- x**
utworzenie nowego pliku	**- - -**	**- w -**
zmiana nazwy pliku	**- - -**	**- w-**
usunięcie pliku	**- - -**	**- w-**
czytanie pliku	**r - -**	**- - x**
zapis do pliku istniejącego	**- w -**	**- - x**
wykonywanie pliku binarnego	**- -x**	**- - x**
wykonywanie skryptu powłoki	**r - x**	**- - x**

6.3. POLECENIA UMOŻLIWIAJĄCE ZMIANĘ PRAW DOSTĘPU

Do zmiany uprawnień użytkowników w stosunku do pojedynczego pliku służy polecenie **`chmod`**. Wymaga ono określenia, czyje uprawnienia należy zmienić, na jakie i w stosunku do którego pliku. Prawa dostępu mogą być podane numerycznie (w formacie ósemkowym) albo symbolicznie. Podając prawa dostępu numerycznie korzystamy z polecenia:

`chmod` **`numeryczny_kod_uprawnien`** **`nazwa pliku`**

Prawo		**binarnie**	**dziesiętnie**
czytania	r - -	1 0 0	4
pisania	- w -	0 1 0	2
wykonywania	- - x	0 0 1	1

Punkty przysługujące poszczególnym kategoriom użytkowników należy złożyć razem. Prawa dostępu w postaci **rw-r--r--** interpretujemy następująco:

prawo dostępu właściciela	rw-	4+2+0=6
prawa dostępu grupy	r--	4+0+0=4
prawa dostępu grupy	r--	4+0+0=4

Cały zestaw praw należy przedstawić jako liczbę trzycyfrową 644 – jest to numeryczny kod uprawnień i podać jako pierwszy argument komendy **chmod**, czyli **chmod 644 nazwa_pliku.**

Prawa dostępu można również ustawiać poleceniem:

chmod kto_ uprawnienia nazwa pliku

Blok **kto_uprawnienia** składa się z trzech elementów:

- Klasy praw dostępu, która określa komu nadawane są prawa i może być jednym, bądź kilkoma, spośród symboli:
  a – wszyscy użytkownicy,
  u – właściciel pliku,
  g – grupa pliku,
  o – inni użytkownicy.

Pominięcie symbolu kategorii nadaje wszystkim (właścicielowi, grupie, pozostałym) takie same prawa.

- Operatora, który jest jednym z następujących znaków:
  – oznacza odebranie prawa,
  + oznacza dodanie prawa,
  = ustala nowe uprawnienia niezależnie od stanu poprzedniego.
- Typu praw dostępu, który jest jednym, bądź kilkoma, spośród symboli: **r, w, x**.

Przykład

chmod u-w plik1 – zabiera **właścicielowi** prawo **pisania** do pliku **plik1**,

chmod a=rw plik1 – nadaje **wszystkim** prawo **rw** do pliku **plik1**,

chmod u+w-x plik1 – dodaje właścicielowi prawo **pisania** do pliku **plik1,** grupie odbiera prawo wykonania tego pliku,

chmod u+w,og+r-x plik1 – nadaje **właścicielowi** prawo **pisania** do pliku **plik1**, a członkom grupy, która jest grupą pliku oraz pozostałym użytkownikom systemu nadaje prawo czytania i odbiera prawo wykonania,

chmod 644 plik1 – nadaje prawo **rw** dla właściciela i prawo **r** dla wszystkich innych i jest równoznaczna komendzie **chmod u+rw,og+r plik1**.

Można zestawiać ze sobą kilka ciągów znaków reprezentujących nowe prawa, oddzielając je przecinkami.

Mimo że inni użytkownicy mogą mieć dostęp do pliku, jedynie jego właściciel może zmienić jego prawa dostępu. Jeśli chcemy dać innym użytkownikom kontrolę nad prawami dostępu do jednego z plików, poleceniem **chown** można zmienić właściciela pliku na innego użytkownika.

Polecenie **chown** przekazuje kontrolę nad plikiem innemu użytkownikowi. Jako pierwszy argument przyjmuje nazwę innego użytkownika. Za nazwą tą można umieścić listę plików, nad którymi przekazuje się kontrolę. Polecenie ma następującą postać:

chown opcje właściciel pliku nazwa pliku

Poleceniem **chown ania plik1 –** użytkownik daje kontrolę nad plikiem **plik1** Ani.

Można również zmienić grupę pliku za pomocą polecenia **chgrp**. Polecenie to przyjmuje za pierwszy argument nazwę nowej grupy dla pliku lub plików:

chgrp opcje grupa nazwy plików

Poleceniem, postaci **chgrp grupa_1 plik1 plik2**, użytkownik zmienia grupę plików **plik1** i **plik2** na **grupa_1**.

6.4. ROZSZERZONE PRAWA DOSTĘPU DO PLIKU

Prawo **x** na poziomie użytkownika lub grupy może być zastąpione przez **s** (dla użytkownika nosi ono nazwę **SUID**, natomiast dla grupy **SGID**). Znaczenie tych praw zostanie omówione w rozdziale 7.3 dotyczącym atrybutów procesów.

Prawo **x** może być zastąpione także literą **t** (ang. save **t**ext mode). Prawo to (sticky bit **–** lepki bit) w przypadku katalogów oznacza, że pliki z tego katalogu może wymazać użytkownik posiadający jedynie prawo pisania do pliku lub właściciel pliku (stosowane np. dla systemowego katalogu tmp). W przypadku plików powoduje to, utrzymanie w pamięci binariów programu nawet po zakończeniu procesu, który z nich korzystał.

6.5. ĆWICZENIA

1. W katalogu /home/fa001 znajdują się pliki **plik1** oraz **plik2**. Ustawione są do nich prawa dostępu **rw-r-----.** Chcesz skopiować plik **plik1** do katalogu **/home/fa002** oraz przenieść **plik2** do katalogu **/home/fa002**. Opisy plików **fa001** i **fa002** z katalogu **/home** mają postać:

```
drwxr----- 10 fa001 fitec-00 1024 May 7 9:15 fa001
drwxr----- 10 fa002 fitec-00 1024 May 7 9:15 fa002
```

Jakie są niezbędne prawa dostępu umożliwiające wykonanie tych poleceń, jeśli jesteś właścicielem pliku **fa001**? Podaj wszystkie polecenia konieczne do wykonania tych operacji z zaznaczeniem, przez którego użytkownika mają one być wykonane.

2. Jakie prawa zostaną nadane plikowi **p1** poleceniami:
   `chmod 574 p1` **jeśli posiadał prawa** `rwxr-----`
   `chmod u=rw,g=r,o-r p174 p1` **jeśli posiadał prawa** `rw--w-rw-`
3. Utwórz w swoim domowym katalogu dwa pliki o nazwach: pisz oraz czytaj. Plik pisz udostępnij kolegom z twojej grupy tak, aby mieli możliwość jego edycji, natomiast czytaj jedynie w trybie oglądania zawartości bądź możliwości skopiowania. Pamiętaj o prawach do katalogu.
4. Sprawdź, czy możesz usunąć plik, do którego nie masz żadnych praw dostępu.
5. Jakie komendy zmieniają liczbę dowiązań do pliku? Sprawdź to.
6. Oblicz jaki może być maksymalny rozmiar pliku i ile zajmuje bloków dyskowych, jeśli blok jest wielkości 128 bajtów.
7. Wypełnij poniższą tabelkę zaznaczając symbolami „+” prawa dostępu konieczne do wykonania poleceń 1 – 13.

Założenia ogólne:

- istnieją katalogi **kat1**, **kat2**,

Założenia do punktów 1 – 11:

- istnieje plik **pl1** w katalogu **kat1**,
- istnieje plik **pl2** w katalogu **kat2**.

Założenia do punktów 12 – 13:

- istnieje plik **pl1** w katalogu **kat1**,
- nie istnieje plik **pl2** w katalogu **kat2**

nr	**1**			**2**			**3**			**4**			**5**			**6**			**7**		
pr.	r	w	x	r	w	x	r	w	x	r	w	x	r	w	x	r	w	x	r	w	x
kat1																					
pl1																					
kat2																					
pl2																					

nr	8			9			10			11			12			13		
pr.	r	w	x	r	w	x	r	w	x	r	w	x	r	w	x	r	w	x
kat1																		
pl1																		
kat2																		
pl2																		

1. **cd kat1**
2. **ls kat1**
3. **ls –s kat1**
4. **ls –F kat1**
5. **ls –i kat1**
6. **pico kat1/pl1 (pełna edycja pliku edytorem – możliwość odczytu i zapisu modyfikacji)**
7. **cat kat1/pl1**
8. **cat > kat1/pl1**
9. **rm kat1/pl1**
10. **cp kat1/pl1 kat2/pl2**
11. **mv kat1/pl1 kat2/pl2**
12. **ln kat1/pl1 kat2/pl2**
13. **ln –s kat1/pl1 kat2/pl2**

7. PROCESY

7.1. PODSTAWOWE WIADOMOŚCI O PROCESACH

Proces jest to pojedynczy program wykonujący się we własnej, wirtualnej przestrzeni adresowej. Zadania czy polecenia mogą się składać z wielu procesów realizujących określoną czynność. Proces jest więc elementarną, dynamiczną jednostką. Proste komendy wykonują się jako jeden proces, natomiast złożone polecenia, korzystające z potoków, powodują uruchomienie jednego procesu dla każdego segmentu potoku. Na przykład, komenda **`cat xyz`** wygeneruje jeden proces, który będzie aktywny aż do momentu kiedy zakończone zostanie wykonywanie komendy **`cat`**. Podobnie powłoka użytkownika jest procesem wykonywanym do momentu zakończenia przez użytkownika sesji. System Linux może w jednym momencie wykonywać, a właściwie sprawiać wrażenie wykonywania kilku procesów. W rzeczywistości większość komputerów ma tylko jeden procesor i może w każdym momencie wykonywać tylko jeden proces.

Odnosi się jednak wrażenie jednoczesnego wykonywania kilku procesów dzięki szybkiemu przełączaniu między nimi i wykonywaniu każdego z nich przez ułamki sekundy. Przełączania lub podział czasu, są tak szybkie, że każdy użytkownik ma wrażenie, jakby pracował w systemie sam. Zarządzanie przydziałem czasu procesora (CPU) odbywa się przez kontrolę procesów. Najważniejsze typy procesów to:

- procesy interakcyjne (pierwszoplanowe lub tła),
- procesy kolejkowane (*batch processes*) – nie związane z żadnym terminalem,
- demony – inicjowane zwykle w czasie startu systemu procesy o charakterze usługowym, które wykonując się w tle oczekują na zlecenia wydawane przez inne procesy systemu.

Każdemu procesowi system przypisuje niepowtarzalny numer, zwany PID. Jest on wykorzystywany do identyfikowania procesów, a szczególnie przydatny do ich przerywania.

Nowy proces tworzy się za pomocą funkcji systemowej zwanej rozwidleniem – **`fork`**, wykonanej przez proces macierzysty. Zawiera on kopię przestrzeni adresowej procesu pierwotnego (ten sam program i te same zmienne, z tymi samymi wartościami). Proces „dziecko” jest więc dokładną kopią procesu macierzystego, lecz ma własny identyfikator PID. Oba procesy (macierzysty i potomny) kontynuują działanie od instrukcji występującej po **`fork`**, z jedną różnicą – funkcja **`fork`** przekazuje zero do nowego procesu (do potomka), natomiast do procesu macierzystego przekazuje (niezerowy) identyfikator procesu potomnego. Po rozwidleniu, w wyniku wykonania funkcji systemowej **`exec`**, treść programu zawarta w przestrzeni adresowej procesu dziecka ulega wymianie na treść programu, który ma być wykonany w ramach danego procesu. Treść nowego programu zastępuje tę odziedziczoną od rodzica, lecz bez zmian pozostaje środowisko procesu (ustawienia zmiennych, strumień wejściowy, wejściowy i błędów) oraz priorytet procesu. W opisany sposób tworzone są wszystkie procesy systemowe.

7.2. ZARZĄDZANIE PROCESAMI

W systemie Linux rozróżniamy dwa typy procesów: jądra i użytkowe. Procesy jądra funkcjonują całkowicie wewnątrz jądra. Sam system operacyjny składa się z kilku procesów:

- **sched –** Swapper jest pierwszym procesem uruchamianym po włączeniu komputera. Proces ten przerzuca całe procesy z i do pamięci, kiedy zasoby pamięci się wyczerpują. Proces ten ma PID równe 0;

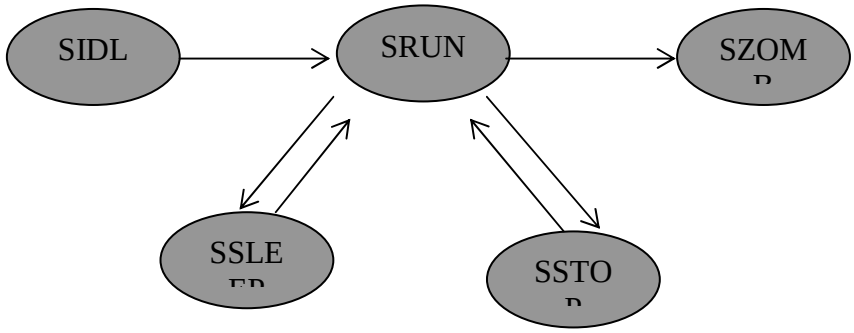
- **init** – jest to drugi proces uruchamiany po włączeniu komputera. Uruchamia on większość innych procesów w systemie, np. proces **getty** przesyłany do każdego terminala. Proces init ma PID = 1;
- **vhand** – proces ten zarządza sektorami pamięci komputera. Służy on do zapewnienia efektywnej pracy poszczególnych sektorów pamięci. Proces vhand ma PID = 2;
- **bdflush** – proces ten zapewnia, że informacje dotyczące systemu plików, znajdujące się w pamięci są co jakiś czas (mniej więcej 10 sekund) uaktualniane lub przerzucane na dysk. Minimalizuje to zasięg zniszczeń, które mogą się pojawić w przypadku awarii systemu lub zaniku napięcia zasilającego.

Każdy proces ma określony priorytet, przy czym obowiązuje zasada: im większa liczba, tym niższy priorytet. Procesy systemowe mają priorytety o wartościach ujemnych i nie mogą być usunięte, procesy użytkowników mają priorytety dodatnie, czyli nie mogą być wykonywane przed systemowymi. Do szeregowania procesów stosowany jest wielokolejkowy algorytm priorytetowy.

Do zarządzania procesami jądro wykorzystuje różnego rodzaju bloki kontrolne. Są to następujące struktury danych: struktura procesu, struktura tekstu programu (tablica procesu), tablice odwzorowania stron i struktura użytkownika. Stan procesu, opisany w strukturze procesu, może przyjmować następujące wartości:

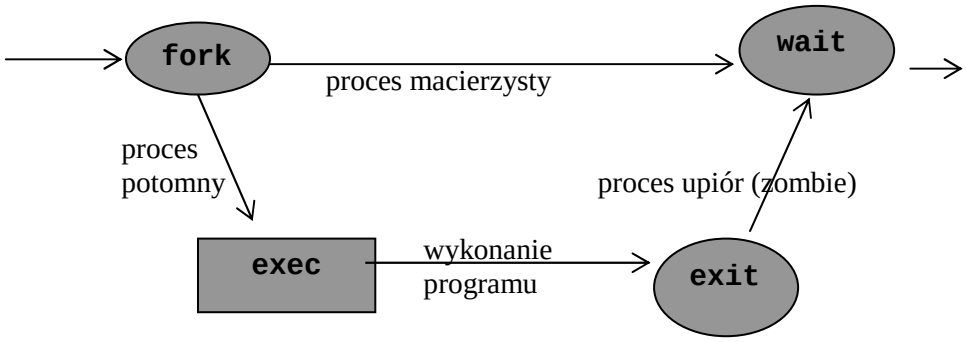
SIDL – stan pośredni podczas tworzenia procesu,
SRUN – wykonywalny,
SSLEEP – oczekujący (na zdarzenie),
SSTOP – zawieszony (proces jest zatrzymany przez sygnał lub proces macierzysty),
SZOMB – stan pośredni podczas usuwania procesu; proces w tym stanie nazywamy procesem uśpionym.

Proces po utworzeniu znajduje się w stanie SIDL. Stan ten ulega zmianie na SRUN, gdy zostaną przydzielone wystarczające zasoby do rozpoczęcia wykonywania procesu. Od tej chwili stan procesu oscyluje między SRUN, SSLEEP i SSTOP, dopóki proces się nie zakończy. Proces zakończony znajduje się w stanie SZOMB do czasu powiadomienia procesu macierzystego o tym fakcie. Stany procesu przedstawia rysunek 8.



Rys. 8. Stany procesu

Funkcja systemowa **`exec(plik,arg1,...,argn)`** umożliwia procesowi wykonanie kodu zawartego we wskazanym pliku, przy czym plik ten zawiera polecenia linuxa lub program w postaci binarnej, natomiast arg1,...,argn są wartościami parametrów tego polecenia. Jego realizacja polega na wprowadzeniu w obszar pamięci operacyjnej procesu nowego kodu i danych. Następną instrukcją procesu jest wówczas pierwsza instrukcja wczytanego kodu. Rozwidlanie procesów przedstawia rysunek 9.

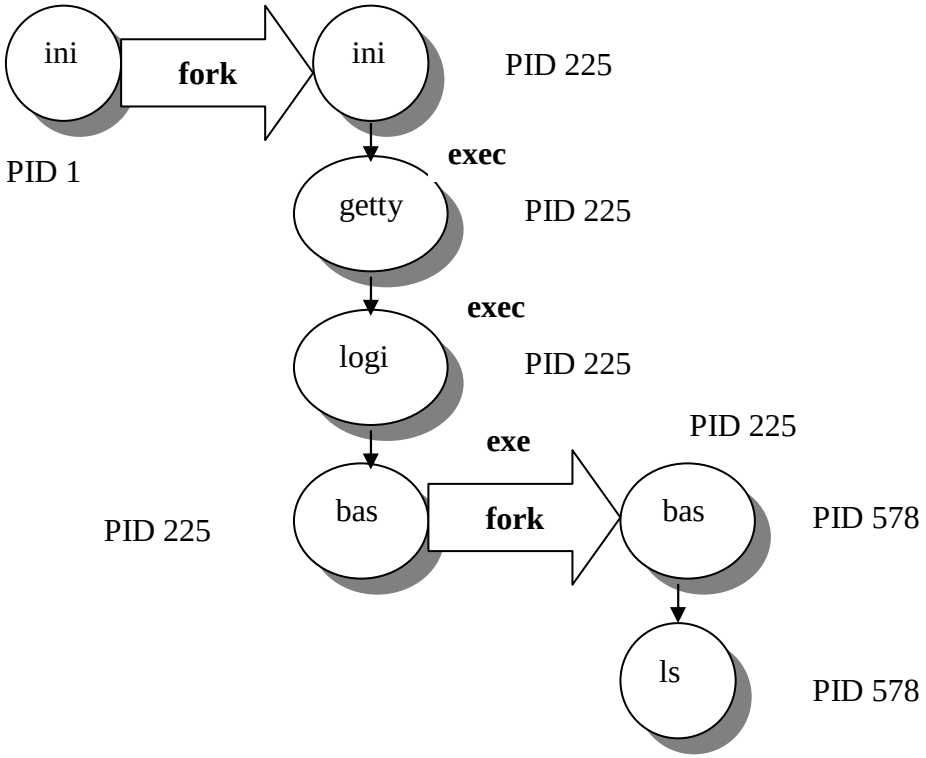


Rys. 9. Rozwidlanie procesów

Funkcja **`wait`** jest zwykle używana do synchronizacji wykonania procesu macierzystego i procesu potomnego, szczególnie wtedy gdy oba procesy mają dostęp do tych samych plików. Proces może zakończyć działanie za pomocą funkcji systemowej **`exit`**, a jego proces macierzysty może oczekiwać na to zdarzenie używszy wywołania funkcji **`wait`**.

Wszystkie procesy użytkowników są potomkami jednego pierwotnego procesu, zwanego początkowym (**`init`**), którego identyfikator jest równy 1 (PID=1). Każdy port terminala gotowego do prowadzenia konwersacji ma aktywny proces

getty, zainicjowany dla niego przez proces **init**. Proces **getty** określa początkowe parametry linii terminala i oczekuje na nazwę otwierającą sesję z użytkownikiem (**login**), którą przekazuje za pomocą funkcji **exec** jako argument do procesu **login**. Proces **login** pobiera hasło użytkownika, szyfruje je i porównuje wynik z zaszyfrowanym napisem z pliku /etc/passwd. Jeśli porównanie wypada pomyślnie, to zezwala się użytkownikowi na wejście do systemu. Proces **login** rozpoczyna wykonywanie procesu **shell**, czyli interpretatora poleceń (np. BASH), ustawiając liczbowy identyfikator użytkownika procesu według danych rejestracyjnych użytkownika.



Rys.10. Fork-and-exec – tworzenie procesu

Przez resztę sesji użytkownik komunikuje się z tym właśnie procesem **shell**. Sam proces **shell** rozwidla się na podprocesy w celu wykonania poleceń przekazywanych mu przez użytkownika. Rysunek 10 ilustruje tworzenie procesu w systemie. Kiedy proces kończy swoje działanie, proces rodzic zostaje o tym poinformowany za pomocą odpowiedniego sygnału. Proces **init** podejmuje

wówczas działanie umożliwiające rozpoczęcie sesji na zwolnionym terminalu innemu użytkownikowi, poprzez rozwidlenie i wykonanie programu **`getty`**.

7.3. ATRYBUTY PROCESU

Procesy zwykłe także otrzymują jednoznacznie je identyfikujący numer, tzw. PID (process ID). To właśnie na jego podstawie jądro wie z jakim procesem ma do czynienia. Oprócz numeru identyfikacyjnego proces ma przypisane identyfikatory użytkownika, przez którego został powołany do życia oraz identyfikatory związane z przynależnością do pewnych grup. Pewne atrybuty decydują o prawach dostępu procesu do zasobów systemu, takich jak pliki i urządzenia. Z każdym procesem związanych jest wiele atrybutów. Najważniejszymi z nich są:

PID – identyfikator procesu (process ID),

PPID – identyfikator procesu przodka,

UID – identyfikator użytkownika, który proces uruchomił (jest on właścicielem procesu),

EUID – efektywny identyfikator użytkownika (effective user ID), który określa jakie prawa przysługują danemu procesowi,

RUID – rzeczywisty identyfikator użytkownika (real user ID), który rozpoczął proces; RUID różni się od EUID, jeżeli uruchomiony program miał ustawione rozszerzone prawa dostępu – SUID (s zamiast x na pozycji użytkownika),

RGID – rzeczywisty identyfikator grupy użytkownika (real group ID),

EGID – efektywny identyfikator grupy użytkownika (effective group); Effective Group ID – różni się od RGID, jeśli uruchomiono program z ustawionym prawem SGID (s zamiast x na pozycji grupy),

TIME – czas trwania,

TTY – terminal,

COM, **CMD** – faktyczne polecenie, które uruchomiło proces,

NI – Liczba **nice** mająca wpływ na priorytet procesu, określa poziom uprzejmości procesu,

SIZE – wielkość pamięci wirtualnej procesu,

RSS – wielkość użytej pamięci rzeczywistej,

STAT – aktualny stan procesu; może on przybrać wartość:

- **R** – run (działający),
- **S** – sleep (uśpiony),
- **D** – oczekujący na operację dyskową,
- **T** – stopped lub traced (zatrzymany lub śledzony),
- **Z** – zombie (proces, który zakończył swoje działanie, zwolnił wszystkie używane zasoby, ale nie otrzymał potwierdzenia przyjęcia sygnału zakończenia od procesu rodzica).

Status dodatkowo może być oznaczony symbolami:

- **W**– nie ma strony zaalokowanej w pamięci operacyjnej,
- **L** – ma stronę w pamięci,
- **<** – wysoki priorytet,
- **N** – obniżony priorytet,

PRI – aktualny priorytet procesu (obliczany dynamicznie),

STIME – czas rozpoczęcia procesu.

Atrybuty procesu EUID i EGID mają ścisły związek z rozszerzonymi prawami **SUID** oraz **SGID** ustawionymi dla programu (pliku wykonywalnego). Pozwalają one na wykonanie pliku użytkownikowi i ustawiają identyfikator odpowiednio użytkownika lub grupy dla procesu w czasie wykonania. Uruchomione w ten sposób procesy mają takie same prawa jak właściciel (w przypadku SUID) bądź grupa wykonywanego programu (dla SGID), a nie jak użytkownik, który uruchamia dany proces lub jego grupa. Jest to sytuacja, w której EUID jest różne od RUID (dla SUID), a EGID różne od RGID (dla SGID) i ma ona wpływ na pliki dostępne dla danego procesu.

7.4. KONTROLA PROCESÓW I OBCIĄŻENIA PROCESORA

Poleceniem służącym do wyświetlania aktualnie wykonywanych procesów jest komenda **ps**. Wyświetla ona następujące informacje dotyczące wszystkich wykonywanych aktualnie przez użytkownika procesów:

numer PID,
nazwę procesu.

Wybrane opcje polecenia **ps**:

- **a** – wyświetlenie procesów wszystkich użytkowników,
- **l** – szczegółowy format wyjściowy,
- **m** – przegląd alokacji pamięci,

r – wyświetlenie tylko procesów działających,

u – wyświetlenie nazwy właściciela procesu i czasu uruchomienia,

x – wyświetlenie procesów nie związanych z terminalem,

-e – wyświetlenie wszystkich procesów,

-o – umożliwienie zdefiniowania listy wyświetlanych parametrów,

-u – wyświetlenie procesów danego użytkownika.

Aby wyświetlić informacje dotyczące procesów pewnego konkretnego użytkownika, należy użyć komendy:

```
ps –U nazwa_użytkownika
```

Jeśli stosuje się opcję **-o** dla zdefiniowania własnego formatu wyświetlanych informacji, należy po niej umieścić oddzielone przecinkami atrybuty. Pełna ich lista dostępna jest w manualu i zawiera m.in. **cmd, pid, user, ruid, ruser, euid, euser, rgid, rgroup, egid, egroup**. Opcja **--sort** umożliwia posortowanie procesów wg danego klucza. Przykładowe polecenie:

```
ps –eo pid,user,ruid,rgid,cmd --sort pid
```

powoduje wyświetlenie następujących informacji o procesach: identyfikator, nazwa właściciela, rzeczywisty identyfikator użytkownika, rzeczywisty identyfikator grupy oraz polecenie uruchamiające proces. Dane te będą posortowane według identyfikatora.

W systemie Linux przydzielanie zasobów procesora pomiędzy procesy, które wykonywane są jednocześnie realizowane jest za pomocą algorytmu bazującego na priorytetach procesów. Priorytety przydzielane są dynamicznie i obliczane przez system operacyjny na podstawie wartości nice (NI), ilości użytego czasu procesora i innych czynników. Liczba nice może być obniżona tylko przez użytkownika uprzywilejowanego (root), zwykły użytkownik może podwyższyć „poziom uprzejmości” swojego procesu, obniżając tym samym jego priorytet.

Polecenie **nice** daje użytkownikowi możliwość wpływania na szeregowanie procesów – pozwala na zainicjowanie procesu z niższym priorytetem. Wartość, o którą może być zmniejszony priorytet procesu, określa parametr **nice** (maksymalnie 19, domyślnie 10).I tak np. polecenie:

```
nice cp plik_1 plik_2&
```

powoduje zmniejszenie o 10 priorytetu odpowiadającego poleceniu **cp** wykonywanemu w tle.

Priorytet procesu oraz wartość parametru **nice** można poznać wykonując polecenie **`ps -l`** i odczytując wartości w kolumnach PRI i NI.

Polecenie **`top`** wyświetla informacje o aktywnych procesach z uwzględnieniem najbardziej obciążających. Możliwa jest praca interaktywna. Poprzez linię komend możliwa jest ingerencja w stan procesu:

r – zmiana wartości nice dla wskazanego zadania (użytkownik nieuprzywilejowany może ją jedynie podnieść, a tym samym zmniejszyć priorytet procesu),

h lub **?** – wyświetlenie pomocy,

k – wysłanie sygnału do procesu (trzeba podać PID procesu),

q – zakończenie.

Polecenie **`pstree`** umożliwia wyświetlenie informacji o procesach w postaci drzewa. Procesy potomne zostają wyświetlone z prawej strony procesów macierzystych.

Polecenie **`kill`** służy do wysyłania sygnałów do procesów. Ma następującą składnię:

```
kill –sygnał pid
```

`kill -l` wyświetla listę sygnałów, które można wysłać. Najważniejsze z nich to:

2 SIGINIT Ctrl+c – przerwanie wykonywania procesu,
3 SIGQUIT Ctrl+\ – zakończenie procesu oraz zrzut pamięci,
9 SIGKILL – zabicie procesu,
15 SIGTERM – miękkie zakończenie procesu (domyślny),
19 SIGSTOP Ctrl+z – zatrzymanie procesu.

Domyślnym sygnałem jest 15, nakazujący procesowi zakończenie działania. Użycie opcji –9 gwarantuje, że proces zostanie usunięty.

7.5. URUCHAMIANIE PROCESÓW W TRYBIE PIERWSZOPLANOWYM I W TLE

Załóżmy, że musimy uruchomić czasochłonny proces, lecz chcemy w trakcie jego działania wykonywać inne czynności. Linux pozwala na uruchomienie procesu w tle i dalsze niezakłócone korzystanie z linii poleceń. Przez dopisanie znaku **&** na końcu dowolnego polecenia spowodujemy rozpoczęcie go w tle i natychmiastowy powrót do linii poleceń. Proces wykonywany w tle to proces

normalnie wykonujący się, tyle że użytkownik może w tym samym czasie wydawać inne komendy z poziomu powłoki.

Uruchamianie poleceń w tle odbywa się przez umieszczenie znaku & na końcu wiersza poleceń. Po uruchomieniu polecenia w tle wyświetlany jest numer zadania użytkownika oraz numer procesu, np. [1] 543. W tło można przenieść więcej niż jedno polecenie. Każde jest klasyfikowane jako zadanie. Nadawana jest mu nazwa oraz numer. Listę zadań działających w tle wypisuje polecenie **jobs**. Każda pozycja na liście składa się z numeru zadania w nawiasach kwadratowych, informacji, czy zadanie jest zatrzymane, czy działa oraz jego nazwy. Znak „+" oznacza, że zadanie jest właśnie wykonywane, a znak „-", że czeka w kolejce do wykonania. System nie informuje o zakończeniu zadania. Zadanie działające w tle można przenieść na plan pierwszy poleceniem **fg**. Jeśli w tle działa więcej niż jedno zadanie, należy podać jego numer po symbolu % jako argument polecenia, np. **fg %2**.

Zadanie działające na pierwszym planie można umieścić w tle. Aby było to możliwe, aktualnie wykonywane zadanie trzeba przerwać (zatrzymać). Kombinacja klawiszy [**CTRL+Z**] powoduje wstrzymanie procesu, dopóki nie zostanie on wznowiony. Przerwane zadanie poleceniem **bg** można umieścić w tle, np. **bg %2**.

% – oznacza bieżące zadanie,
%n – oznacza zadanie o numerze n.

Zadanie działające w tle można zatrzymać wymuszając jego zakończenie, dzięki poleceniu **kill**. Polecenie **kill** przyjmuje jako argument numer zadania użytkownika lub pid procesu. Numer zadania użytkownika musi być poprzedzony znakiem **%** (np. **kill %2**). Można go odczytać używając polecenia **jobs**.

Polecenie

```
sleep 100&
```

wykona proces sleep, który będzie „spał" w tle przez 100 sekund. Proces czasochłonny można zawsze przerzucić w tło. Zamiast wciskać CTRL + c (co spowodowałoby zamknięcie procesu pierwszoplanowego) i powtórnie uruchamiać go w tle po prostu trzeba nacisnąć **CTRL+z**, a następnie użyć polecenie **bg**.

Nie jest celowe przerzucanie programu w tło i z powrotem. Zdaje to egzamin, gdy korzystamy z terminali tekstowych. Jeśli piszemy tekst pod edytorem i chcemy na chwilę przejść do linii poleceń, aby wydać jakąś komendę, a następnie powrócić do pisania, naciskamy CTRL +Z, wpisujemy **bg**, aby przenieść edytor w tło. W linii poleceń wpisujemy inne polecenie, a następnie komendą **fg** przenosimy edytor z tła. Edytor pojawia się na ekranie w takim samym stanie, w jakim go opuściliśmy. Zdarza się czasami, że trzeba przerwać proces, czy to z powodu dziwnego działania, czy dlatego, że się zawiesił i nie reaguje na żadne sygnały wprowadzane z klawiatury. Jeśli chodzi o jakikolwiek proces wykonywany w tle, po prostu należy przycisnąć klawisz **<DEL>.**

Przykład 1

Uruchom dwa procesy, których strumienie wyjściowe oraz błędów przekierowane są do plików, wstrzymaj ich działanie, a następnie wznów je ponownie, przeanalizuj strumień wyjściowy polecenia jobs i zakończ je:

```
find / -user nazwa_użytkownika >&plik
Ctrl+z
find / -user nazwa_użytkownika >&plik2
Ctrl+z
%1
Ctrl+z
jobs
kill %1
kill %2
```

Przykład 2

Uruchom dwa procesy, których strumienie wyjściowe oraz błędów przekierowane są do plików, wstrzymaj ich działanie, a następnie przenieś je w tło, przeanalizuj strumień wyjściowy polecenia jobs, przenieś procesy w tryb bezpośredni:

```
find / -user nazwa_użytkownika >&plik
Ctrl+Z
jobs
bg %1
jobs
fg %1
jobs
```

7.6. SYSTEM PLIKOWY PROC

W katalogu głównym systemu plikowego znajduje się katalog proc. Jest to punkt montowania wirtualnego system plików, który nie zarządza zbiorami znajdującymi się na dysku ani na żadnym urządzeniu, lecz znajduje się w pamięci operacyjnej. Daje on dostęp do danych na temat uruchomionych procesów i jądra systemu. Znajdują się tam pliki: kcore, będący obrazem pamięci systemu, cpuinfo, zawierający dane na temat procesora, meminfo, przechowujący informacje o pamięci operacyjnej oraz katalogi odpowiadające wszystkim uruchomionym

procesom, których nazwy są identyfikatorami (PID) procesów. Tekstowe pliki znajdujące się w tych katalogach zapewniają dostęp do danych o procesach i można z nich odczytać atrybuty procesów.

7.7. ĆWICZENIA

W katalogu **/home/wilki/janek** znajduje się plik **prog1**, którego opis (efekt polecenia **`ls –l`**) wygląda następująco:

```
-rws—x---  1  janek  wilki 1200 Sept 30 20:08  prog1
```

1. Użytkownik **pawel**, będący członkiem grupy **wilki**, wydał polecenie: **`/home/wilki/janek/prog1`**.
   Podaj wartości atrybutów ruser i euser procesu uruchomionego przez niego.
2. Podaj wartości atrybutów ruser i euser procesu uruchomionego przez użytkownika **janek** za pomocą polecenia **`/home/wilki/janek/prog1`**.
3. Użytkownik **marek**, będący członkiem grupy **lwy**, wydał polecenie: **`/home/wilki/janek/prog1`**.
   Co możesz powiedzieć na temat uruchomionego w ten sposób procesu, podaj wartości atrybutów ruser i euser.
4. Wydaj polecenia **`top`** i **`find`** z dowolnymi poprawnymi parametrami, a następnie odszukaj w systemie plikowym proc informacje na temat uruchomionych procesów.

8. WSPÓŁBIEŻNOŚĆ I SYNCHRONIZACJA

8.1. PODSTAWOWE POJĘCIA

Proces jest to ciąg czynności wykonywanych za pośrednictwem ciągu rozkazów, których wynikiem jest realizacja pewnych zadań systemowych. Pojęcie to można rozszerzyć, aby obejmowało zarówno zadania określone przez użytkownika, jak i zadania systemowe, wówczas wykonywanie programu użytkownika nazwiemy również procesem. Mówimy, że proces jest wykonywany przez procesor. Procesor jest czymś co wykonuje rozkazy; w zależności od rodzaju rozkazów procesor może być zrealizowany wyłącznie sprzętowo lub jako połączenie sprzętu i oprogramowania. Pojęciami procesu i procesora można z łatwością posłużyć się przy interpretowaniu zarówno współbieżności, jak i niedeterminizmu.

Współbieżność można rozumieć jako uaktywnienie kilku procesów w tym samym czasie. Przy założeniu, iż istnieje tyle samo procesorów co procesów, nie napotykamy żadnych logicznych przeszkód. Jeżeli jednak – jak to zdarza się częściej – procesorów jest mniej niż procesów, wówczas można uzyskać pozorną współbieżność przełączając procesory od jednych procesów do drugich. Jeśli przełączenia są dokonywane w dostatecznie małych okresach czasu, to system obserwowany przez dłuższy czas pozwoli stworzyć iluzję współbieżności.

Jeżeli będziemy rozważać procesy jako ciągi czynności, które można przerywać między kolejnymi krokami, to niedeterminizm przejawi się w nieprzewidywalnym porządku, w którym mogą występować przerwania. Rozważania możemy podsumować w następujący sposób: Proces jest ciągiem czynności, a więc jest dynamiczny, podczas gdy program jest ciągiem instrukcji, czyli jest statyczny.

Procesy wewnątrz systemu komputerowego oczywiście nie działają w odosobnieniu. Z jednej strony muszą one współdziałać, aby uzyskać zamierzony cel – wykonać pracę zleconą przez użytkownika. Z drugiej strony współzawodniczą o korzystanie z ograniczonych zasobów, takich jak procesory, pamięć operacyjna lub pliki. Współdziałanie i współzawodnictwo wymagają pewnego sposobu komunikacji między procesami. Rodzaje komunikacji są następujące:

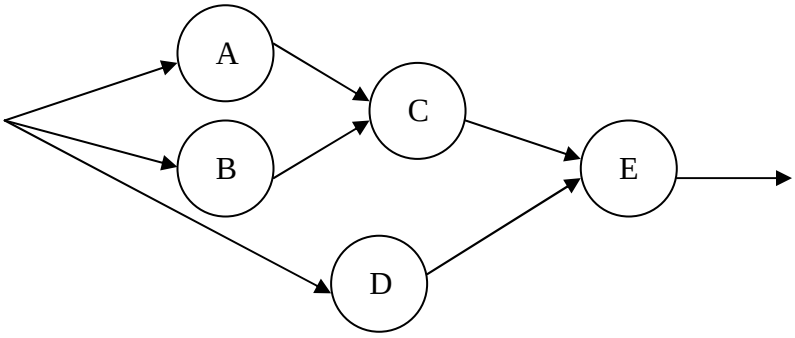
1. **Wzajemne wykluczanie** – zasoby systemowe można określić jako podzielne, czyli takie, z których kilka procesów może korzystać współbieżnie oraz niepodzielne, czyli takie, z których za każdym razem może korzystać tylko jeden proces. Niepodzielność zasobów wynika z następujących faktów:
   - natura fizyczna danego zasobu jest taka, że nie praktykuje się jego dzielenia, czego typowym przykładem jest drukarka;
   - zasoby są tego rodzaju, że gdyby korzystało z nich kilka procesów współbieżnie, to czynności jednego procesu mogłyby zakłócać czynności drugiego. Szczególnie popularnym przykładem jest miejsce w pamięci operacyjnej zawierające wartości zmiennej, z której korzysta więcej niż jeden proces.

   Do zasobów niepodzielnych należy zatem większość urządzeń zewnętrznych, plików zapisywalnych i takich obszarów danych, które ulegają zmianom. Do zasobów podzielnych należą jednostki centralne, pliki przeznaczone wyłącznie do czytania i te obszary pamięci operacyjnej, w których znajdują się same procedury lub dane chronione przed wprowadzaniem do nich zmian. Wzajemne wykluczanie polega na zapewnieniu takich warunków działania systemu, by tylko jeden proces na raz mógł korzystać z zasobów niepodzielnych.

2. **Synchronizacja** – na ogół nie można przewidzieć jaka będzie szybkość jednego procesu w stosunku do drugiego, ponieważ zależy ona od tego, jak przedstawia się częstotliwość przerwań każdego procesu oraz jak często i jak długo każdy proces może korzystać z procesora. Mówimy, że procesy

przebiegają względem siebie asynchronicznie. Aby jednak uzyskać zadowalającą współpracę, wyznacza się pewne punkty, w których muszą one synchronizować swoje działanie. Poza te punkty jeden proces nie może przejść dopóty, dopóki drugi nie zakończy pewnego etapu swego działania. Zadaniem systemu operacyjnego jest stworzenie mechanizmów umożliwiających synchronizację tych procesów.

Praca współbieżna polega na tym, że składające się na nią czynności odbywają się równocześnie. Wykonywanie elementarnych czynności to **akcje**. W danym zadaniu, będącym zbiorem procesów, można wyróżnić czasowe sekwencje pewnych akcji – są to **procesy sekwencyjne**. Dwie akcje **są potencjalnie współbieżne**, gdy nie jest dla nich określona kolejność czasowa. Muszą one należeć do różnych procesów. Zależność czasową pomiędzy poszczególnymi akcjami można przedstawić za pomocą grafu pierwszeństwa (ang. precedence graph). Węzły odpowiadają akcjom, natomiast strzałki ilustrują powiązania między nimi – następstwo czasowe. Graf z rysunku 11 przedstawia zadanie, w którym akcje A i B mogą zachodzić równocześnie, akcja C może się rozpocząć po ich zakończeniu, a do rozpoczęcia akcji E wymagane jest zakończenie akcji C oraz D.



Rys. 11. Graf pierwszeństwa

Dla poprawnej realizacji zadania konieczna jest synchronizacja procesów, za sprawą której proces przechodziłby do stanu zawieszenia w oczekiwaniu na zajście jakiegoś zdarzenia i byłby reaktywowany przez akcję innego procesu.

3. **Blokada** – jeżeli kilka procesów współzawodniczy o zasoby, to może wytworzyć się taka sytuacja, w której każdy proces – aby móc nadal działać – będzie musiał skorzystać z tych zasobów, których używa inny proces; wówczas żaden proces nie będzie mógł kontynuować swej pracy. Taką sytuację nazywamy blokadą lub zakleszczeniem (ang. deadlock). Można to rozumieć jako stan systemu, w którym procesy pozostają zawieszone w

oczekiwaniu na zdarzenia, które nigdy nie zajdą. Jednym z zadań systemu operacyjnego jest niedopuszczanie do powstania blokady albo przynajmniej ograniczanie jej skutków.

8.2. OPERACJE SEMAFOROWE

Podstawowym problemem odnoszącym się do współpracy procesów jest wykluczanie wzajemne, rozumiane jako zapewnienie wyłączności działania w pewnym obszarze, nazywanym **sekcją krytyczną**, jednemu procesowi.

Punktem zwrotnym w rozwoju teorii systemów operacyjnych i przetwarzania równoległego był rok 1965, kiedy to Dijkstra'e przedstawił rozwiązanie problemu wykluczania, wprowadzając pojęcia semaforów oraz wykonywanych na nich operacji czekaj (**wait**) i sygnalizuj (**signal**). Podstawową ideą mechanizmu semaforowego jest założenie, że procesy nie mogące wejść do sekcji krytycznej są zawieszane (wait), a po zwolnieniu jej przez inny proces – reaktywowane (signal). Zasady wchodzenia do sekcji krytycznej są następujące:

1. Jeśli sekcja jest wolna, proces wchodzi do niej bezpośrednio.
2. Jeśli sekcja jest zajęta, to proces jest zawieszany i oczekuje na jej zwolnienie.
3. Po zwolnieniu sekcji krytycznej proces sprawdza, czy żaden inny proces nie jest zawieszony w oczekiwaniu na wejście do niej, jeśli tak, to jednemu z nich umożliwia wejście do sekcji.

Aby zapewnić taki dostęp do sekcji krytycznej, należy związać z nią pewną zmienną typu semafor. Semafor jest to nieujemna liczba całkowita, na której z wyjątkiem nadawania wartości początkowych, można wykonywać jedynie operacje **czekaj** (określaną również mianem **wait** lub **P**) i **sygnalizuj** (określaną również mianem **signal** lub **V**), które są niepodzielne – nic nie może przerwać ich wykonania.

Operacja sygnalizuj (signal, V) odpowiada podniesieniu semafora. W jej wyniku następuje zwiększenie wartości semafora **s** o 1, przy czym traktuje się tę operację jako niepodzielną. Z jej niepodzielności wynika, że wykonanie operacji sygnalizuj nie jest równoważne wykonaniu instrukcji przypisania s:=s+1. Operacja ta jest wykonywana po wyjściu procesu z sekcji krytycznej:

V(s): s:=s+1

Operacja czekaj (wait, P) odpowiada oczekiwaniu na podniesienie semafora, a następnie jego opuszczenie. W jej wyniku następuje zmniejszenie wartości semafora s o 1, o ile wartość ta jest dodatnia. Jeśli nie jest to możliwe (s=0), proces wywołujący tę operację jest zawieszany i będzie mógł być nadal wykonywany tylko wówczas, gdy inny proces zwiększy wartość semafora o 1 w wyniku operacji sygnalizuj. Również i ta operacja jest niepodzielna. Jeżeli kilka procesów było

wstrzymanych, to po uzyskaniu przez semafor wartości dodatniej tylko jeden proces może zakończyć wykonywanie tej operacji:

P(s): wstrzymanie procesu aż s>0
s:= s - 1

Istnieje ścisłe powiązanie obu wprowadzonych operacji. Jeśli proces inicjuje operację P na semaforze opuszczonym (s=0), to jest zawieszany do czasu wykonania operacji V na tym semaforze przez inny proces. W przypadku gdy kilka procesów jest zawieszonych, po wykonaniu operacji V musi być wybrany jeden z nich, który pomyślnie ukończy operację P.

Zasoby niepodzielne można chronić przed tym, aby równocześnie korzystało z nich kilka procesów. W tym celu zapobiega się współbieżnemu wykonywaniu przez procesy tych fragmentów programu, poprzez które uzyskują one dostęp do zasobów. Te fragmenty programu nazywa się **sekcjami krytycznymi**. Wzajemne wykluczanie, w odniesieniu do użytkownika, zasobów można traktować jako wzajemne wykluczanie w odniesieniu do wykonywania sekcji krytycznej.

Wykluczanie można osiągnąć w prosty sposób, umieszczając każdą sekcję krytyczną między operacjami czekaj i sygnalizuj wykonywanymi na jednym semaforze, którego wartością początkową jest 1. Sekcję krytyczną programuje się w następujący sposób:

czekaj (nazwa semafora)
sekcja krytyczna
sygnalizuj (nazwa semafora)

Warunki, jakie musi spełniać wzajemne wykluczanie są następujące:

1. Symetria i równouprawnienie procesów – procesy są traktowane jako równoważne, nie mogą mieć przypisanych priorytetów, wszystkie decyzje dotyczące ich wyboru muszą być względem nich sprawiedliwe.
2. Niezależność szybkości procesów – nie wolno czynić żadnych założeń dotyczących względnej szybkości.
3. Skończony czas podejmowania decyzji – jeżeli więcej niż jeden proces chce wejść do sekcji krytycznej, to decyzja o tym, który z nich zostanie wybrany musi być podjęta w skończonym czasie.
4. Niezależność zaproponowanego mechanizmu od działania procesu poza sekcją krytyczną.

Jeśli zmienna semaforowa została zainicjowana na wartość n większą od 1, to w sekcji krytycznej może znaleźć się równocześnie n procesów. Bardzo ważną

odmianą semaforów są semafory binarne. Mogą one przyjmować dwie wartości: zero (false – semafor opuszczony) lub jeden (true – semafor podniesiony).

Semafory są uniwersalnym mechanizmem, pozwalającym oprócz znanego już problemu wykluczenia wzajemnego rozwiązać wiele innych problemów synchronizacji i komunikacji.

Przykład

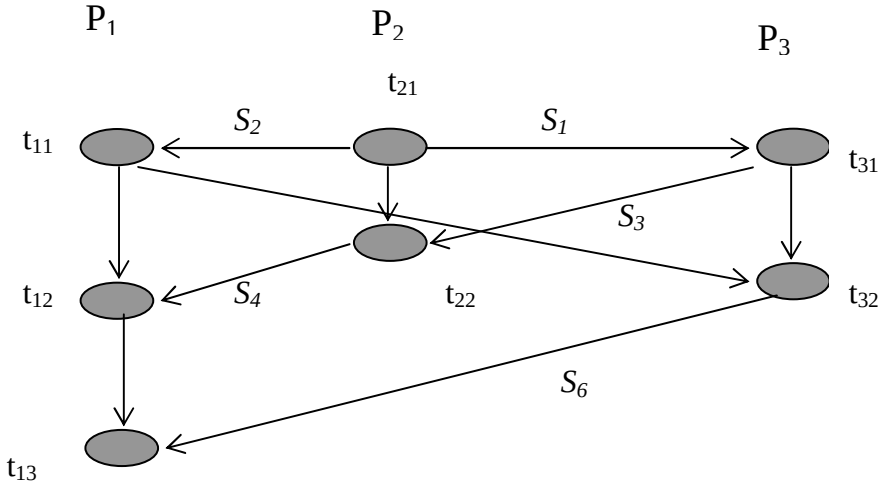
Zadanie, które należy wykonać to zbiór trzech sekwencyjnych procesów: P_1, P_2, P_3.
Proces P_1 składa się z trzech akcji: t_{11} – trwającej 2 jednostki czasu,
t_{12} – trwającej 1 jednostkę czasu,
t_{13} – trwającej 1 jednostkę czasu,
Proces P_2 składa się z dwóch akcji: t_{21} – trwającej 1 jednostkę czasu,
t_{22} – trwającej 4 jednostki czasu,
Proces P_3 składa się z dwóch akcji: t_{31} – trwającej 1 jednostkę czasu,
t_{32} – trwającej 2 jednostki czasu.

Kolejność czasowa akcji zdefiniowana jest zbiorem par G

$$G=\{(t_{21}, t_{31}), (t_{21}, t_{11}), (t_{31}, t_{22}), (t_{22}, t_{12}), (t_{11}, t_{32}), (t_{32}, t_{13})\}.$$

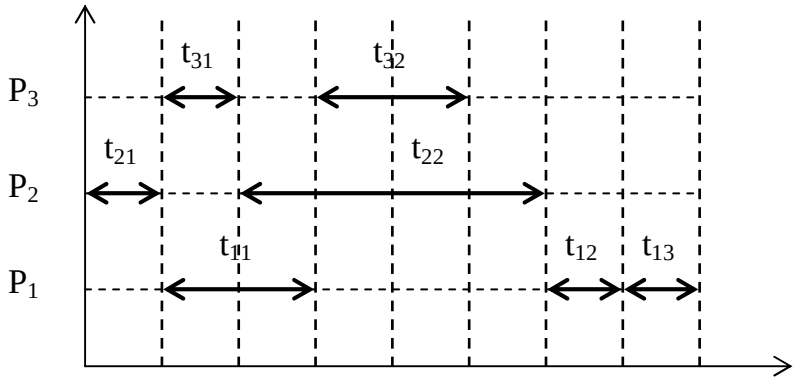
Jeśli para (**a,b**) należy do zbioru, to akcja **b** musi być poprzedzona akcją **a**.

1. Narysuj graf pierwszeństwa oraz wykres czasowy.
2. Przy użyciu semaforów zsynchronizuj procesy P_1, P_2, P_3 . Rozwiązanie przedstaw w postaci pseudokodów dla P_1, P_2, P_3.
3. Ile czasu potrzeba na zakończenie procesów P_1, P_2, P_3?
4. Ile procesorów potrzeba na osiągnięcie takiego czasu?



Wartości początkowe semaforów są równe 0.

P1	**P2**	**P3**
begin	begin	begin
repeat	repeat	repeat
wait(*S2*)	do t_{21}	wait(*S1*)
do t_{11}	signal(*S1*)	do t_{31}
signal(*S5*)	signal(*S2*)	signal(*S3*)
wait(*S4*)	wait(*S3*)	wait(*S5*)
do t_{12}	do t_{22}	do t_{32}
wait(*S6*)	signal(*S4*)	signal(*S6*)
do t_{13}	end	end
end		



Na zakończenie procesu P_1 potrzeba 8 jednostek; procesu P_2 – 6 jednostek; procesu P_3 – 5 jednostek. Dla osiągnięcia takiego czasu konieczne są dwa procesory.

9. POWŁOKI

9.1. WSTĘP

Strukturę systemu Unix, a w tym także Linux, można przedstawić za pomocą modelu warstwowego. Najbardziej wewnętrzną częścią jest jądro systemu, które ma dostęp do wszystkich zasobów komputera. Zewnętrzna warstwa, za pomocą której użytkownik komunikuje się z systemem to powłoka (*shell*). W systemach Unix funkcjonują trzy różne rodzaje powłok: shell Bourne'a (sh), shell Korna (ksh)

oraz C shell (csh). Linux udostępnia rozszerzone warianty wymienionych interpretatorów poleceń. Są to:

- BASH (Bourne Again Shell),
- PDKSH (Public Domain Korn Shell),
- TCSH.

Po zalogowaniu się do systemu Linux zostaje uruchomiona powłoka – dla każdego użytkownika uruchomiany jest jego własny egzemplarz interpretera komend (login shell). Domyślną powłoką dla użytkownika jest BASH, jednak wszystkie są w pełni dostępne. Można w dowolnym momencie je wywołać za pomocą poleceń: bash, tcsh, ksh (kolejna powłoka jest zagnieżdżona w poprzedniej). Aby zakończyć pracę z daną powłoką, należy wykonać polecenie **`exit`** lub (CTRL+D). Wszystkie powłoki mają wspólny zestaw podstawowych operacji, opisany w poprzednich rozdziałach. Różnią się między sobą sposobem definiowania zmiennych lokalnych i środowiskowych, aliasów, odwołaniem do historii zdarzeń, skryptami inicjacyjnymi oraz językiem skryptowym.

9.2. POWŁOKA BASH

9.2.1. ZMIENNE ŚRODOWISKOWE I POWŁOKI

Pracując z daną powłoką można wydawać polecenia, deklarować zmienne, tworzyć i uruchamiać skrypty. Zmienne zdefiniowane w danej powłoce są dla niej lokalne i nie można się do nich odwoływać w innych powłokach. Można jednak zdefiniować zmienne środowiskowe, które są dostępne w każdej nowej powłoce. Sposób ich przekazywania jest analogiczny do przekazywania parametrów przez wartość – nowa powłoka dysponuje kopiami zmiennych środowiskowych. Zmienne środowiskowe są zwykle używane do podawania globalnych ustawień systemu (ścieżka dostępu do poleceń, postać znaku zachęty). Nazwa zmiennej może się składać z dowolnych liter, cyfr oraz ze znaku ”_”, nie może jednak rozpoczynać się cyfrą.

Zmienne lokalne można definiować poleceniem: **`zmienna=wartosc`** (obok operatora przypisania nie mogą znajdować się spacje), a należy się do nich odwoływać poprzez symbol **$zmienna**. Aby usunąć zmienną, stosujemy polecenie **`unset zmienna`**. Listę wszystkich zdefiniowanych zmiennych można wyświetlić poleceniem **`set`**. Wartości przypisywane zmiennym mogą zawierać znaki, które mają szczególne znaczenie:

”.... ” – maskuje znaki specjalne (spacja, *, ?, [,], ., <, >, &, |),
‘.... ’ – działają jak podwójne cudzysłowy, dodatkowo maskują znak $,

`.....` – wymuszają wykonanie polecenia,
\ – maskuje jeden znak.

Zmienne lokalne powłoki można wyświetlić poleceniem **`set`**. Aby utworzyć zmienną środowiskową ze zdefiniowanej zmiennej, należy użyć polecenia **`export zmienna`**.
Linux definiuje specjalne zmienne powłoki, które są używane do konfigurowania powłoki. Ich nazwy są zarezerwowanymi słowami kluczowymi. Listę zdefiniowanych zmiennych specjalnych można wyświetlić poleceniem **`env`**. Najważniejsze z nich to:

HOME – ścieżka katalogu macierzystego,

USERNAME – nazwa użytkownika,

SHELL – ścieżka powłoki logowania,

PATH – lista katalogów rozdzielonych dwukropkami, które są przeszukiwane w celu znalezienia polecenia do uruchomienia,

PS1 – monit podstawowy; może zawierać następujące kody:

\! – nr z listy historii,
\d – data,
\s – bieżąca powłoka,
\t – czas,
\u – nazwa użytkownika,
\w – bieżący katalog,

PS2 – monit wtórny (dla drugiego i kolejnych wierszy polecenia),

CDPATH – lista katalogów przeszukiwanych podczas wykonywania komendy cd,

TERM – nazwa terminala,

HISTSIZE – ilość zdarzeń historii,

HISTFILE – nazwa pliku historii zdarzeń,

HISTFILESIZE – rozmiar pliku historii,

MAIL – ścieżka do pliku skrzynki pocztowej,

MAILCHECK – określa odstęp czasu, po jakim użytkownik jest powiadamiany o nowej poczcie,

MAILPATH – nazwy ścieżek plików skrzynek pocztowych.

BASH używa **kilku** dodatkowych zmiennych określających właściwości powłoki, będącymi opcjami, które mogą być włączone lub wyłączone. Są to: **ignoreeof**, **noclobber**, **noglob**. Ustawienie zmiennej dokonuje się poleceniem:

```
set –o zmienna
```

Aby wyłączyć działanie zmiennej, należy użyć polecenia:

```
set +o zmienna
```

Znaczenie poszczególnych zmiennych jest następujące:

ignoreeof – blokuje możliwość wylogowania się za pomocą [CTRL+d] (domyślnie do 10 razy). Można nadać jej dowolną wartość (np. ignoreeof=30),

noclobber – zapobiega nadpisaniu plików podczas przekierowania (>),

noglob – blokuje rozwijanie znaków specjalnych.

9.2.1.1. Ćwiczenia

Zdefiniuj zmienne, a następnie sprawdź ich wartości:

1.
```
a=cos
echo $a
set
```

2.
```
set zm1=”wartosc zm. A wynosi $a”
echo $zm1
```
(” ” – nie maskują znaku $)

3.
```
set zm2=’wartosc zm a wynosi $a’
echo $zm2
```
(‘ ‘ – maskują znak $)

4. Nadaj zmiennej wartość odpowiadającą nazwie katalogu i użyj jej w poleceniu kopiowania
```
katalog=~/proba
set
cp plik $katalog
```

5. Usuń zdefiniowaną zmienną i sprawdź, czy faktycznie jej już nie ma
```
unset katalog
echo $katalog
set
```

6. Sprawdź działanie znaków ` czy powodują potraktowanie wartości zmiennej jako polecenia do wykonania:
```
listc=`ls –l *.c`
echo $listc
set
```

9.2.2. HISTORIA

W powłoce BASH ostatnio wydane polecenia są pamiętane. Polecenie **`history`** wyświetla listę ponumerowanych poleceń (zdarzeń). Można się do nich odwoływać używając poleceń z wykrzyknikiem.

Przykłady

`!3` – wyświetla i wykonuje polecenie historii nr 3,

`!ab` – wyświetla i wykonuje ostatnio wydane polecenie rozpoczynające się ciągiem znaków „ab”,

`!?ab?` – wyświetla i wykonuje ostatnio wydane polecenie zawierające ciąg znaków „ab”,

`!-3` – wyświetla i wykonuje trzecie od tyłu wydane polecenie.

Polecenia historii można składać. Jeśli np. polecenie o numerze 100 ma postać „**`ls`**”, wówczas **„`!100 *.c`**” jest równoważne poleceniu „**`ls *.c`**”. Możliwe jest także odwołanie do parametrów ostatnio wydanego polecenia poprzez „**!!***” lub „**!***”.

Przykład

Wydane zostało polecenie:

```
ls /home/elektr-a/ea029
```

Wówczas wydanie polecenia:

```
find !* -name szukaj
```

powoduje rozpoczęcie poszukiwania pliku o nazwie **szukaj** od katalogu **/home/elektr-a/ea029**.

Liczba pamiętanych zdarzeń historii przechowywana jest w zmiennej systemowej **HISTSIZE**. Aby ją zmienić, należy użyć polecenia np.: **`HISTSIZE=100`**. Historia przechowywana jest w pliku **.bash_history**, w katalogu domowym użytkownika. Można to zmienić definiując zmienną

HISTFILE i nadając jej wartość, będącą nazwą innego pliku, np. **`HISTFILE=mojahistoria`**. Wówczas po wylogowaniu historia wydanych poleceń trafi do pliku o nazwie mojahistoria.

9.2.3. ALIASY

Aliasy, czyli utożsamienia, to dodatkowe nazwy dla najczęściej wykonywanych poleceń. Polecenie **`alias`** bez parametrów wyświetla listę zdefiniowanych aliasów. Tym samym poleceniem definiuje się także nowe aliasy.

Przykłady

`alias kto=who`	– zamiast polecenia who można używać nazwy **`kto`**,
`alias l='ls -li'`	– **`l`** staje się drugą nazwą dla polecenia **`ls`** z opcjami **`li`**; jeśli w poleceniu występują spacje, należy użyć pojedynczych cudzysłowów,
`alias lc='ls -l *.c'`	– polecenie **`lc`** może być przydatne przy listowaniu plików źródłowych w języku C z danego katalogu,
`alias cp='cp -i'`	– nazwa polecenia również może być aliasem.

Aby usunąć zdefiniowany alias, należy użyć polecenia **`unalias`**, np. **`unalias kto`** – usuwa definicję aliasu **`kto`**.

Aliasy nie są przekazywane do powłok potomnych, istnieją jedynie w bieżącej powłoce, pamiętane są w pamięci operacyjnej .

9.2.4. PLIKI STARTOWE POWŁOKI

Definiowanie zmiennych oraz aliasów służy usprawnieniu pracy z powłoką. Dlatego konieczne jest, aby po zalogowaniu użytkownik miał zdefiniowane swoje środowisko według swoich potrzeb. W tym celu definicje aliasów i zmiennych środowiskowych umieszcza się w plikach startowych powłoki. Plik inicjacyjny logowania powłoki BASH to **.bash_profile**. Jest on wykonywany automatycznie w czasie logowania użytkownika, powłoką logowania którego (login shell) jest BASH. Plik **.bashrc** wykonywany jest przy każdym uruchomieniu nowego shella (BASH). Oba pliki znajdują się w katalogu domowym użytkownika.

W pliku .bash_profile znajdują się m.in. polecenia ustawiające ścieżki przeszukiwania, typ terminala, znak zachęty, inne zmienne środowiskowe. Nowy shell powoływany jest przy wykonywaniu skryptów lub dowolnych poleceń, które nie są wbudowane w interpreter poleceń (komendy wewnętrzne powłoki). Zawsze wtedy wykonywany jest plik .bashrc. Znajdują się w nim polecenia ustawiające zmienne powłoki oraz definicje aliasów.

Istnieją ogólne pliki startowe powłoki, które są wykonywane przed osobistymi. Pliki .bash_profile oraz .bashrc poza swoim standardowym przeznaczeniem mogą być dodatkowo uruchomione w dowolnym momencie przez użytkownika analogicznie jak inne skrypty powłoki poleceniem „**.**” lub „**`source`**”.

Przykładowy plik **.bash_profile**:

```
#.bash_profile
PATH=$PATH:$HOME/bin:$HOME:.
CDPATH=$CDPATH:$HOME
HISTSIZE=100
PS1=[\u\s@\w]$
PS2=@
export PATH CDPATH HISTSIZE PS1 PS2
```

Przykładowy plik **.bashrc**:

```
#.bashrc
set –o ignoreeof
set –o noclobber
alias rm=’rm –i’
alias cp=’cp –i’
alias mv=’mv –i’
alias l=’ls –la’
```

Do plików inicjacyjnych powłoki zaliczany również bywa **.bash_logout**, który jest wykonywany w czasie wylogowywania się użytkownika. Mogą się w nim znaleźć przykładowo polecenia:

```
clear
echo „Do nastepnego razu”
```

9.3. POWŁOKA TCSH

Powłoka TCSH jest kompatybilna ze standardową powłoką C.

9.3.1. ZMIENNE ŚRODOWISKOWE I POWŁOKI

Zmienne zwykłe powłoki TCSH definiuje się za pomocą polecenia **`set`**. Poprawne są następujące przykłady:

```
set zmienna=wartosc
set zmienna = wartosc
```

Uwaga!

Spacje występują po obu stronach znaku „=" lub nie występują w ogóle.

Aby usunąć definicję zmiennej, trzeba użyć polecenia **`unset`**. Podobnie jak w BASH występują zmienne o charakterze przełączników: **ignoreeof**, **noclobber**, **noglob**. Dodatkowo można ustawiać następujące zmienne:

echo – sterującą wyświetlaniem (lub nie) poleceń przed ich wykonaniem,
notify – sterującą informowaniem o zakończeniu procesów tła,
verbose – sterującą wyświetlaniem polecenia po odwołaniu się do historii.

Zmienne te ustawia się poleceniem **`set nazwa`**.

Przykłady

```
set ignoreeof
set notify
unset verbose
```

Zmienne specjalne powłoki TCSH:

prompt – umożliwia definiowanie monitu,

prompt1 – definiuje monit kolejnej linii polecenia,

history – określa ilość pamiętanych zdarzeń historii,

savehist – określa liczbę zdarzeń, które zostaną zapisane i przechowane w pliku .history do następnego zalogowania,

path, cdpath – mają takie samo znaczenie jak PATH i CDPATH w powłoce BASH; lista katalogów objęta jest nawiasami, a poszczególne pozycje oddzielone są spacjami,

user – nazwa użytkownika,

shell – powłoka logowania,

term – terminal,

mail – tablica, której elementy oddzielone są spacjami, łącząca

funkcje zmiennych MAIL, MAILCHECK i MAILPATH z powłoki BASH.

W powłoce TCSH zmienne środowiskowe definiowane są za pomocą polecenia **`setenv`** bez znaku równości. Ich znaczenie i zasięg jest analogiczny jak exportowanych zmiennych powłoki BASH.

Przykład

```
setenv PATH ($PATH /home/abc)
```

Powłoka zachowuje zgodność zmiennych środowiskowych: **HOME**, **USER**, **TERM**, **PATH** ze zdefiniowanymi zmiennymi zwykłymi: **home**, **user**, **term**, **path**. Aby wyświetlić zdefiniowane zmienne zwykłe, należy wywołać polecenie: **set** bez argumentów, natomiast polecenie **`setenv`** pokazuje listę zdefiniowanych zmiennych środowiskowych.

9.3.2. HISTORIA

Sposób odwoływania się do poleceń historii jest analogiczny jak w powłoce BASH. W powłoce TCSH liczba pamiętanych poleceń shella zależy od wartości zmiennej **`history`**. Zmienna **`savehist`** określa ilość ostatnio wydanych poleceń (w obrębie wszystkich sesji), które będą zapisane w pliku **.savehist** w katalogu domowym użytkownika i które będzie można wykorzystać jako historię w kolejnej sesji.

Przykład

```
set history = 200
set savehist = 200
```

W powłoce można odwoływać się do ostatnio wydanych poleceń, korzystając z symboli {**tekst**} lub {**nr**}, co oznacza ostatnie polecenie rozpoczynające się tekstem lub polecenie o numerze **nr**.

Przykłady

`!{ls} -la` – zostanie powtórzone ostatnio wydane polecenie **`ls`** z dodaniem opcji **`la`**,

`!{cd} /` – jeśli ostatnie polecenie **cd** było bez parametrów, wykonane zostanie **cd /**

`!{c} /` – wykonane zostanie ostatnio wydane polecenie rozpoczynające

się na **c** z parametrem /,

!{2}1 – zostanie wykonane polecenie będące złożeniem polecenia drugiego historii i symbolu „1”,

Podobnie jak w BASH w TCSH można korzystać z parametrów ostatnio wydanego polecenia stosując symbol: **!***

9.3.3. ALIASY

W powłoce TCSH aliasy definiuje się poleceniem:

```
alias ‘tekst polecenia’
```

natomiast usuwanie aliasów dokonuje się poleceniem :

```
unalias polecenie
```

Przykłady

```
alias dir ‘ls’
alias rm ‘rm –i’
alias sdir ‘ls –la|sort’
```

W tym przypadku wywołanie polecenia „**sdir** /” nie dałoby zamierzonego efektu. Aby uzyskać posortowaną listę zawartości katalogu głównego, należy alias zdefiniować w sposób następujący:

```
alias sdir ‘ls-l \!*|sort’
```

a następnie skorzystać z niego: **sdir /**

Ciąg symboli „!*” oznacza parametry ostatnio wydanego polecenia, czyli w naszym przypadku „/”.Poprzedzający je symbol „\” ma za zadanie maskowanie specjalnego symbolu ”!”.

9.3.4. PLIKI STARTOWE POWŁOKI

W powłoce TCSH stosuje się trzy pliki inicjacyjne: **.login**, .**logout**, .**tcshrc**. Ich znaczenie jest takie jak plików startowych powłoki BASH: **.bash_profile**, **.bash_logout**, **.bashrc**. Plik .login wykonywany jest przy każdym logowaniu użytkownika, .logout przy wylogowywaniu się, natomiast .tcshrc uruchamiany jest przy każdym wywołaniu powłoki TCSH. Ponadto każdy z nich może być wykonany w dowolnym momencie za pomocą polecenia **source** lub **.**.Wówczas do wykonania danego skryptu nie jest wywoływana powłoka.

LITERATURA

[1] E. Frisch: *UNIX – Administracja systemu*. Wyd. RM, Warszawa 1997.

[2] W.Iszkowski, M. Maniecki: *Programowanie współbieżne.* WNT, Warszawa 1981.

[3] Z. Królikowski, M. Sajkowski: *System operacyjny UNIX dla początkujących i zaawansowanych.* Wyd. Nakom, Poznań 1998.

[4] A. M. Lister, R. D. Eager: *Wprowadzenie do systemów operacyjnych.* WNT, Warszawa 1994.

[5] M. Ludwiński: *Podsystem plików*. LinuxPlus nr 7/99, s.25-28.

[6] M. Ludwiński: *Podsystem sterujący procesami*. LinuxPlus nr 8/99, s.19-22.

[7] L. Madeja: *Ćwiczenia z systemu Linux.* Wyd. MIKON, Warszawa 1999.

[8] J. Olszewski: Projektowanie struktur systemów operacyjnych. WNT, Warszawa 1981.

[9] R. Petersen: *Arkana – Linux*. Wyd. RM, Warszawa 1997.

[10] M. J. Roclikind: *Programowanie w systemie UNIX dla zaawansowanych*. WNT, Warszawa 1993.

[11] A. Show: *Projektowanie logiczne systemów operacyjnych*. WNT, Warszawa 1980.

[12] A. Silberschatz, J. Peterson, P. Galvin: *Podstawy systemów operacyjnych*. WNT, Warszawa 1999.

[13] S. Strobel, T. Uhl: *Linux*. WNT, Warszawa 1993.