8. PRAWA DO KATALOGÓW I PLIKÓW

Właściciel zasobów może ograniczyć innym użytkownikom systemu dostęp do plików i katalogów dzięki możliwości nadawania i odbierania *praw dostępu*. Organizacja praw dostępu wygląda następująco:

- każdy plik i katalog posiada atrybuty określające prawo do:
  - odczytania (r - ang. *reading*),
  - zapisania (w - ang. *writing*),
  - wykonania (x - ang. *executing*).
- prawa dostępu określane są niezależnie dla:
  - właściciela zasobu (u - ang. *user*),
  - członków grupy, do której należy właściciel zasobu (g - ang. *group*),
  - pozostałych użytkowników (o - ang. *others*).

Wyświetlenie praw dostępu - polecenia *ls* z opcją *-l*, np.

```
$ ls -al
drwxrwxr-x 6 nowakj infor 755 Feb 22 17:23 .
drwxrwxr-x 2 nowakj infor 222 Jan 10 10:22 ..
-rwxr-x--- 1 nowakj infor 1560 Feb 26 13:01 lista
```

Odczytanie wyświetlonych w taki sposób informacji o zasobie i prawach dostępu do niego, umożliwia analiza 10 pierwszych znaków każdego wiersza (np. *drwxr-x--x*). Znaczenie poszczególnych znaków (pozycji) jest następujące:

- 1. znak - rodzaj zasobu: katalog (d), plik (-), link symboliczny (l)
- 2. - 4. znak - prawa właściciela zasobu,
- 5. - 7. znak - prawa grupy, do której należy właściciel,
- 8. - 10. znak - prawa pozostałych użytkowników.

-	r	w	x	r	W	x	r	w	X
plik (-), katalog (d), symlink (l)	**u** (user) prawa właściciela zasobu			**g** (group) prawa grupy, do której należy właściciel zasobu			**o** (other) prawa pozostałych użytkowników		

Dziewięć znaków, które mówią o prawach dostępu (znaki od 2. do 10.) określa się mianem *maski praw* (np. *rwxr-x--x*).

Liczbowa maska praw:

r	w	x	r	-	x	-	-	x
1	1	1	1	0	1	0	0	1
111 binarnie = 7 dziesiętnie user			101 binarnie = 5 dziesiętnie group			001 binarnie = 1 dziesiętnie other		

Przykładowe maski praw należy interpretować:

```
751 = rwxr-x--x
711 = rwx--x--x
530 = r-x-wx---
```

Do zmiany praw dostępu służy polecenie *chmod*, które można uruchomić zarówno z maską liczbową, jak i prawami poszczególnych użytkowników (u, g, o, a) zapisanymi odpowiednimi symbolami (r, w, x), np.

```
$ chmod 751 lista.txt
$ chmod g+wx *
$ chmod a-rwx tajne.txt
```

Schemat zmiany praw przy wykorzystaniu symbolicznych oznaczeń:

$$\$ \text{ chmod } \begin{bmatrix} u \\ g \\ o \\ a \end{bmatrix} \begin{bmatrix} + \\ - \end{bmatrix} \begin{bmatrix} r \\ w \\ x \end{bmatrix} \text{ plik}$$

9. ZARZĄDZANIE PROCESAMI

9.1. Charakterystyka procesów

Proces jest to program realizowany przez system operacyjny. Wyróżniamy:

- *procesy użytkownika* (*zadania użytkownika*) - programy i polecenia uruchomione przez użytkownika,
- *procesy systemowe* - programy uruchomione przez system operacyjny.

Systemy operacyjne umożliwiające współbieżną realizację procesów to:

- *systemy wielozadaniowe* - ponieważ w tym samym czasie system operacyjny realizuje więcej niż jedno zadanie,
- *systemy z podziałem czasu* - ponieważ czas pracy procesora dzielony jest pomiędzy poszczególne procesy.

Przeciwieństwem systemów wielozadaniowych są *systemy jednozadaniowe* (np. MS DOS).

9.2. Zarządzanie procesami użytkownika w systemie Linux

Nowe zadanie użytkownika - np. program, polecenie - uruchamiane jest domyślnie *na pierwszym planie*, pozostałe zadania znajdują się *w tle*.
Użytkownik ma możliwość zarządzania uruchomionymi zadaniami - może:

- uruchomić zadanie w tle - poprzez dodanie znaku ampersand (&), np.

```
$ vi praca.txt &
[1] 7835
```

po uruchomieniu zadania w tle, wyświetlane są dwa numery - pierwszy, w nawiasach kwadratowych, oznacza numer zadania użytkownika (tutaj: 1), a drugi numer procesu systemowego (tutaj: 7835).

```
$
```

- zatrzymać proces na pierwszym planie i przenieść go w tło - służy do tego kombinacja [Ctrl]+[Z], np.

```
$ vi referat.txt
...
```

uruchomienie i praca z edytorem vi

```
[Ctrl]+[Z]
```

zatrzymanie edytora i przeniesienie go w tło

```
 [3]+ Stopped        vi
```

informacje o zatrzymanym edytorze, m.in. [3] - nadany numer zadania, „+" - znacznik bieżącego zadania (tzn. „+" oznacza, że jest to bieżące zadanie, czyli ostatnie przeniesione w tło)

- wyświetlić informacje o stanie zadań w tle - polecenie *jobs*, np.

```
$ jobs
[1]   Stopped                 vi praca.txt
[2]-  Running                 sleep 1000 &
[3]+  Stopped                 vi referat.txt
[4]   Done                    ls

$ jobs -l
[1]-  7835 Stopped            vi praca.txt
[2]   7841 Running            sleep 10000 &
[3]+  7902 Stopped            vi referat.txt
```

- wyświetlić informacje o procesach systemowych - polecenie *ps*, np.

```
$ ps
PID  TTY      TIME     CMD
7679 pts/1    00:00:00 bash
7835 pts/1    00:00:00 vi
7841 pts/1    00:00:00 sleep
7902 pts/1    00:00:00 vi
8103 pts/1    00:00:00 ps
```

- przesunąć zadanie z tła na pierwszy plan - służy do tego polecenie *fg* wywoływane z numerem zadania, nazwą procesu, znacznikiem zadania bieżącego (+) lub poprzedniego (-), np.

```
$ fg 1
$ fg +
$ fg "vi referat.txt"
```

- przesunąć zatrzymane zadanie z pierwszego planu w tło - polecenie *bg*, np.

```
$ bg 1
```

- zakończyć wykonywane zadanie przed czasem - do zakończenia zadania pierwszoplanowego służy kombinacja [Ctrl]+[C], do zakończenie zadania w tle służy polecenie *kill* wywoływane z numerem procesu systemowego lub numerem zadania, znacznikiem bieżącego (+) lub poprzedniego (-) zadania. Przy poleceniu *kill*, numer zadania użytkownika i znaczniki zadania muszą być poprzedzone znakiem procent (%), np.

```
$ kill 7835
$ kill %2
$ kill %+
```

9.3. Przykład zarządzania procesami

Napisane zostały dwa skrypty - *skrypt1.sh* i *skrypt2.sh*

Plik *skrypt1.sh*:

```
echo "Poczatek 1" > plik1
sleep 45
echo "Koniec 1" >> plik1
```


Plik *skrypt2.sh*:

```
echo "Poczatek 2" > plik2
sleep 45
echo "Koniec 2" >> plik2
```

Uruchomienie zadania na pierwszym planie:

```
$ . skrypt1.sh
```

znak zachęty pojawia się dopiero po 45 sekundach

```
$
```

Uruchomienie zadania w tle:

```
$ . skrypt1.sh&
[1] 783
```

znak zachęty pojawia się natychmiast

```
$ . skrypt2. sh&
[2] 794
```

Wyświetlenie stanu realizowanych zadań:

```
$ jobs -l
[1]-  783 Running        skrypt1.sh&
[2]+  794 Running        skrypt2.sh&
```

Usunięcie zadań:

```
$ kill 783
$ kill %+
```

10. BEZPOŚREDNIA KOMUNIKACJA Z UŻYTKOWNIKAMI SYSTEMU

Użytkownik w systemie Linux może komunikować się bezpośrednio z innymi użytkownikami. Przydatne do tego celu są polecenia:

- *finger* - wyświetlenie informacji o zalogowanych użytkownikach, np.

```
$ finger
```

- *write* - wysłanie komunikatu do zalogowanego użytkownika, np.

```
$ write nowakj
Czesc Janek
Co u Ciebie slychac?
^D
```

- *talk* - prowadzenie „rozmów” z innym zalogowanym użytkownikiem, np.

```
$ talk nowakj
```

użytkownik nowakj dostanie komunikat:

```
Message from Talk_Daemon@ie.uek.krakow.pl at 12:21...
talk: connection requested by kowalj@ie.uek.krakow.pl
talk: respond with:  talk kowalj@ie.uek.krakow.pl
```

aby przyjąć rozmowę użytkownik nowakj musi wydać polecenie:

```
$ talk kowalj
```

aby odrzucić rozmowę lub ją zakończyć nowakj musi skorzystać z kombinacji klawiszy [Ctrl]+[C]

- polecenie *mesg* - wyłączenie (*mesg n*) lub włączenie (*mesg y*) przyjmowania wiadomości i rozmów przez użytkownika, np.

```
$ mesg n
```