Powrót (Systemy operacyjne)

Zarządzanie procesami

ćwiczenia laboratoryjne

Ważne komendy

- `bg` - wznawia zawieszone procesy pozostawiając je uruchomione w tle.
- `fg` - wznawia zawieszone procesu i uruchamia je w pierwszym planie.
- `jobs` - wyświetla listę zatrzymanych procesów.
- `kill` - wysyła sygnał do procesu wpływając na jego zachowanie lub go przerywając.
- `killall` - wysyła sygnał do procesów o określonej nazwie wpływając na jego zachowanie lub go przerywając.
- `nice` - pozwala na uruchomienie procesu z określonym priorytetem.
- `nohup` - uruchamia procesu, który będzie działał po wyjściu z terminala.
- `ps` - wyświetla listę aktualnie uruchomionych procesów w systemie, do których użytkownik ma dostęp.
- `pstree` - wyświetla drzewo aktualnie uruchomionych procesów w systemie, do których użytkownik ma dostęp (wskazujące relacje między procesami).
- `renice` - pozwala na zmianę priorytetu uruchomionego procesu.
- `top` - interaktywne narzędzie do monitorowania procesów.
  można też stosować nowszą komendę: `htop`
- `trap` - służy do przechwytywania sygnałów wysłanych do procesu skryptu.
  - listę sygnałów można wyświetlić dodając opcję `-l`.

Ważne polecenia

- `exit` *`kod`* - powoduje zakończenie działania procesu i przekazanie jego kodu wyjścia.
  - *`kod`* jest liczbą jednobajtową (tzn. z zakresu 0-255).
- `wait` *`PID`* - powoduje zsynchronizowanie się (zaczekanie) procesu rodzicielskiego z zakończeniem procesu potomnego o podanym identyfikatorze i zwrócenie tego kodu jako wartości zwracanej przez polecenie.

Skróty

- `$$` - PID aktualnego procesu.
- `$PPID` - PID nadrzędnego procesu (rodzica).
- `$!` - PID ostatnio uruchomionego procesu drugoplanowego.
- `$?` - wartość zwrócona przez ostatnio wykonane polecenie pierwszoplanowe.

Przesyłanie sygnału do procesu: `kill -s` *`sygnał PID`*

Skróty klawiszowe

- ctrl-c - przerywa ("zabija") aktualnie uruchomione zadanie.
- ctrl-z - zatrzymuje aktualnie uruchomione zadanie, które można później zakończyć lub wznowić, na pierwszym planie lub w tle.

Uruchomienie polecenia w tle: *`polecenie`* `&`

Zadania

Zadanie 1. Wyświetl listę wszystkich procesów:

- wraz z informacją jaki jest właściciel danego.
- wraz z procesami, które wykonują się w innym terminalu.
- wraz z procesami, które wykonują się bez terminala (zamiast pliku do terminala zawierają `?`).

Zadanie 2. Utwórz skrypt, który w nieskończonej pętli będzie wyświetlał dowolny niepusty napis co 5 sekund oraz uruchom go w tle przy pomocy `&` oraz wykonaj poniższe czynności.

- sprawdź listę procesów, które zostały uruchomione w tle w bieżącym terminalu.
- otwórz nowy terminal oraz wyświetl w nim listę wszystkich procesów, w taki sposób, aby widoczne były również te które zostały uruchomione w poprzednim terminalu.
- sprawdź czy możesz wznowić proces działający w tle na pierwszy plan w pierwszym oraz drugim terminalu.
- sprawdź czy możesz przerwać (zabić) proces działający w tle w pierwszym oraz drugim terminalu.
- wyłącz oba terminale, następnie otwórz ponownie nowy terminal oraz sprawdź czy procesy nadal istnieją.

Zadanie 3. Utwórz skrypt, który w nieskończonej pętli będzie wyświetlał dowolny niepusty napis co 5 sekund oraz uruchom go w taki sposób, aby tworzył proces, który nie jest zależny od terminala oraz wykonaj poniższe czynności.

- wyłącz terminal, następnie otwórz ponownie nowy oraz sprawdź czy proces nadal istnieje.
- wyloguj się następnie zaloguj i sprawdź czy proces nadal istnieje.
- sprawdź czy możesz wznowić proces na pierwszym planie.
- przerwij (zabij) utworzonego w tle proces następnie przejrzyj plik *nohup.out* oraz postaraj się z zrozumieć z czego wynika jego zawartość.
- uruchom proces przekierowując zawartość do pliku i sprawdź co się stanie.

Zadanie 4. Napisz dwa skrypty. Pierwszy sprawdza czy długości trzech odcinków pobranych jako argumenty są dodatnie. Jeżeli tak jako kod wyjścia przekazuje 1,w przeciwnym razie 0. Drugi skrypt uruchamia w tle pierwszy podając argumenty, które otrzymał oraz pobiera wynik procesu. Jeżeli długości są dodatnie powinien wyświetlić informację czy z podanych boków możliwe jest zbudowanie trójkąta.

Zadanie 5. Utwórz trzy skrypty. Pierwszy:

```
#!/bin/bash

trap ./napis.sh USR1
trap ./koniec.sh USR2
while true
do
 sleep 5
done
exit 0
```

Drugi powinien wyświetlić dowolny napis, a trzeci przerwać (zabić) proces skryptu. Uruchom pierwszy skrypt. Następnie w drugim terminalu prześlij do procesu uruchamiającego skrypt sygnał `USR1` oraz sprawdź co zostanie wyświetlone w pierwszym terminalu. Następnie prześlij sygnał `USR2`.

Zadanie 6. Utwórz skrypt, który będzie działał w nieskończonej pętli. Jeżeli do procesu uruchamiający skrypt zostanie wysłany sygnał `SIGINT`, powinien wyświetlać napis: *Jestem nieśmiertelny*. Jeżeli do procesu uruchamiający skrypt zostanie wysłany sygnał `SIGHUP`, skrypt powinien przestać reagować na sygnał `SIGINT`.

Zadanie 7. Przy pomocy skryptów utwórz symulator dwóch dźwigów, które przenoszą materiały budowlane z pierwszego miejsca do drugiego. Pierwszy dźwig powinien przenosić z pierwszego miejsca do bufora pomocniczego, o ograniczonej pojemności, natomiast drugi dźwig pobierać materiał z bufora do drugiego miejsca.
Symulator powinien składać się z:

- trzech folderów, które będą symulowały: pierwsze miejsce, drugie miejsce, bufor;
- plików, które będą symulowały materiał budowlany, które będą znajdywały się w pierwszym miejscu;
- trzech procesów, które będą symulowały: pierwszy dźwig, drugi dźwig i nadzorce dźwigów.

Proces nadzorujący po otrzymaniu sygnału `USR1` powinien utworzyć dwa procesy symulujące prace dźwigów oraz uruchomić je w tle, natomiast po otrzymaniu sygnału `SIGINT`, zakończyć pracę dwóch utworzonych procesów i wyświetlić komunikat.
Proces symulujący pracę pierwszego dźwigu powinien przenosić pojedynczo pliki z pierwszego folderu do folderu symulującego bufor, dopóki pierwszy folder nie będzie pusty. Po przeniesieniu wszystkich plików powinien zakończyć swoje działanie zwracając liczbę przeniesionych plików. Należy też zadbać o to by nie przenosił plików, jeżeli bufor przekroczy swoją pojemność (np. bufor może mieć co najwyżej trzy pliki).
Proces symulujący pracę drugiego dźwigu powinien przenosić pojedynczo pliki z bufora do drugiego folderu o ile w buforze jest jakiś plik oraz zakończyć swoje działanie jeżeli pierwszy "dźwig" zakończył pracę oraz bufor jest już pusty. Po zakończeniu pracy proces powinien zwrócić liczbę przeniesionych plików.
Proces nadzorujący (o ile nie zostanie przerwany przez użytkownika) po rozpoczęciu działania, powinien nadzorować pracę dwóch dźwigów i wypisywać odpowiednie komunikaty (odnośnie rozpoczęcia pracy dźwigu, zakończenia pracy dźwigu, liczby przeniesionych materiałów - dane pobrane od innych procesów). Dodatkowo może posłużyć do komunikacji między procesami symulującymi pracę dźwigu oraz po zakończeniu ich pracy powinien zakończyć swoje działanie.
Pamiętaj, że symulacja powinna zostać utworzona z dokładnie trzech procesów, które są odpowiednio zsynchronizowane oraz każdy z tych procesów powinien kończyć swoje działanie w odpowiednim momencie.