Powrót (Systemy operacyjne)

Programowanie systemowe - procesy, wątki

ćwiczenia laboratoryjne

Przydatne biblioteki i funkcje

- unistd.h
  - fork - tworzy proces. wartości zwracane
    - -1 - jeżeli wystąpił błąd (nie udało się utworzyć procesu).
    - 0 - jeżeli proces został utworzony (stał się dzieckiem procesu nadrzędnego).
    - PID procesu potomnego (większy niż 0) - jeżeli proces utworzył proces potomny.
  - getpid - zwraca identyfikator bieżącego procesu.
  - getppid - zwraca identyfikator nadrzędnego procesu.
  - sleep - zawiesza wykonanie procesu na pewien czas (sekundy).
  - usleep - zawiesza wykonanie procesu na pewien czas (milisekundy).
- stdlib.h
  - exit - kończy proces.
- sys/wait.h
  - wait - czeka na zakończenie lub zatrzymanie procesów potomnych.
  - waitpid - czeka na zakończenie lub zatrzymanie procesu potomnego.
- signal.h
  - signal - obsługuje sygnał dla procesu.
  - raise - wysyła sygnał do bieżącego procesu.
  - kill - wysyła sygnał do procesu o podanym PID.
  - pthread_kill - wysyła sygnał do wątku.
  - sigemptyset - inicjalizuje zbiór sygnałów.
  - sigaddset - dodaje sygnał do zbioru.
  - sigwait - czeka na sygnał.
- pthread.h
  - pthread_create - tworzy wątek.
  - pthread_join - czeka na zakończenie wątku.
  - pthread_self - zwraca id wątku.
  - pthread_exit - kończy wątek.

Przydatne stałe wartości

- signal.h
  - SIG_DFL - domyślna obsługa sygnału.
  - SIG_IGN - ignorowanie sygnału.

Zadania

Zadanie 1. Przed uruchomieniem postaraj się odgadnąć co wyświetli program w C:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>


int main() {
 printf("pierwszy napis\n");
 fork();
 wait(NULL);
 printf("drugi napis\n");
 fork();
 wait(NULL);
```

```
 printf("trzeci napis\n");
 return 0;
}
```

Następnie skompiluj, uruchom oraz sprawdź, co zostanie wyświetlone. Uzasadnij działanie tego programu oraz sprawdź co się stanie jeśli usuniesz `wait(NULL)` z kodu programu.

Zadanie 2. Utwórz program, który będzie działał w nieskończonej pętli. Jeżeli do procesu uruchamiający skrypt zostanie wysłany sygnał `SIGINT`, powinien wyświetlać napis: *Jestem nieśmiertelny* oraz swój indentyfikator procesu. Jeżeli do procesu uruchamiający skrypt zostanie wysłany sygnał `SIGHUP`, skrypt powinien przestać reagować na sygnał `SIGINT`.
Podpowiedź: Przykładowy program obsługujący sygnał:

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

void handleSignal(int s) {
 write(STDOUT_FILENO, "HELLO", 5);
}

int main() {
 signal(SIGINT, handleSignal);
 return 0;
}
```

Zadanie 3. Napisz program, który utworzy proces potomny.
Proces potomny powinien wyświetlać swój identyfikator oraz identyfikator procesu nadrzędnego oraz "wylosowaną" liczbę z zakresu [1; 50] oraz zakończyć swoje działanie w momencie, gdy wylosowana liczba będzie podzielna przez 5.
Proces nadrzędny, powinien poczekać, aż jego proces potomny zakończy swoje działanie i wypisać komunikat w którym wypisze swoje id, a następnie zakończyć program.
Podpowiedź: W celu losowania liczb możesz użyć biblioteki: `<time.h>`, `<stdlib.h>` oraz funkcji `rand`.

Zadanie 4. Napisz program, który utworzy 2 procesy potomne.
Procesy potomne powinny działać w nieskończonej pętli wyświetlać swój identyfikator oraz identyfikator procesu nadrzędnego.
Proces nadrzędny powinien, co jakiś czas, "losować" liczbę [1; 50] i zakończyć pracę pierwszego jeśli wylosowana zostanie liczba mniejsza niż 10 oraz drugiego procesu jeśli większa niż 40. Po "zabiciu" dwóch procesów powinien skończyć swoje działanie.
Podpowiedź: W celu losowania liczb możesz użyć biblioteki: `<time.h>`, `<stdlib.h>` oraz funkcji `rand`.

Zadanie 5. Napisz program, który będzie zawierał wątek, który, co jakiś czas, będzie wyświetlał swój identyfikator oraz "wylosowaną" liczbę z przedziału [1; 50]. Wątek powinien zakończyć swoje działanie w momencie, gdy wylosowana liczba będzie podzielna przez 5.
Podpowiedź 1: W celu losowania liczb możesz użyć biblioteki: `<time.h>`, `<stdlib.h>` oraz funkcji `rand`.
Podpowiedź 2: Przykładowy program tworzący wątek:

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

void *myThread(void *arg) {
 usleep(600000);
}

int main() {
 pthread_t threadId;

 pthread_create(&threadId, NULL, myThread, NULL);
 pthread_join(threadId, NULL);

 return 0;
}
```