Procesy

Systemy Operacyjne 2

Piotr Zierhoffer

15 października 2011

Procesy

Procesy:

- własna przestrzeń adresowa
  - kod, dane
  - stos i sterta
- rejestry systemowe
- zasoby
- niezależne struktury w systemie operacyjnym...
- ... ale pamięć utrzymywana w trybie Copy–on–write

Tworzenie procesów

- Funkcja:
  `pid_t fork(void)`
- Wartość zwracana:
  - u rodzica: PID procesu potomnego; u procesu potomnego: 0
  - w wypadku błędu: -1 u rodzica (brak procesu potomnego)
- Przykład:

```
main(){
    printf (“Poczatek\n”);
    fork();
    printf (“Koniec\n”);
}
```

Proces macierzysty / potomny

```
main(){
     if (fork())
          printf (“Parent\n”);
     else
          printf (“Child\n”);
}
```

Identyfikatory procesów

- Funkcje:

```
pid_t getpid(void);
pid_t getppid(void);
```

- Wartość zwracana:
  - zawsze poprawne wykonanie — identyfikator procesu / rodzica
- Zadanie: stworzyć dwa procesy, wydrukować dla każdego PID i PPID.

Przetwarzanie współbieżne

```
main(){
     fork();
     printf (“Hi\n”);
     fork();
     printf (“Ha\n”);
     fork();
     printf (“Ho\n”);
}
```

Ilość stworzonych procesów

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
main (){
     fork ();
     fork ();
          if (!fork ())
           fork ();
     fork ();
}
```

Ilość stworzonych procesów

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
main (){
     fork ();
     fork ();
          if ( fork () )
           exit ();
     fork ();
}
```

Ilość stworzonych procesów

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
main (){

     fork ();
     fork ();

          if ( fork () )
          if ( !fork () )

           fork();

          else

           exit();
          fork ();

     }
```

Zakończenie procesu

- Funkcja:
  `void exit(int status)`
- Parametry:
  - `status` — kod wyjścia przekazywany procesowi macierzystemu

- Funkcja:
  `int kill(pid_t pid, int signum)`
- Parametry:
  - `pid` — identyfikator procesu, do którego adresowany jest sygnał
  - `signum` — numer przesyłanego sygnału

Efekt działania potomka

- Funkcje:
  `pid_t wait(int *status)`
  `pid_t waitpid(pid_t pid, int *status, int options)`
- Parametry:
  - `status` — adres w pamięci, w którym **między innymi** umieszczony zostanie status zakończenia
    - może być NULL
  - `pid` — PID procesu, na którego status oczekujemy
  - `options` — opcje :) Np. `WNOHANG`, etc., lub 0
- Wartość zwracana:
  - identyfikator zakończonego procesu lub -1 w przypadku błędu
  - 0, jeżeli użyto flagi `WNOHANG` i nie było żadnego dostępnego potomka

Analiza wartości `status`

Zmienna status, pisana przez funkcję `wait`, zawiera:

- status wyjścia (np. ustawiony przez `exit()`, 8 najstarszych bitów)
- kod sygnału (7 najmłodszych bitów)
- informacja, czy program wykonał zrzut pamięci (1 pozostały bit)

Analiza za pomocą makr:

- `WIFEXITED(status)`
- `WEXITSTATUS(status)`
- `WIFSIGNALED(status)`
- `WTERMSIG(status)`
- `WCOREDUMP(status)`

Zombie i sierotki

- Sierota — proces potomny, którego przodek się już zakończył
- Zombi — proces potomny, który zakończył swoje działanie i czeka na przekazanie statusu zakończenia przodkowi

- Zadanie: Stwórz proces sierotę i proces zombie
  - sprawdzanie stanu procesów:
    `ps aux | grep nazwaprogramu`

Uruchamianie programów

- Grupa funkcji exec służy **podmianie** obrazu aktualnego procesu na nowy.
- Funkcje:
  - `execl`, `execlp`, `execle`, `execv`, `execvp`
- Wszystkie te funkcje są nakładkami na :
  `int execve(const char *filename, char *const argv [], char *const envp[])`
- Parametry:
  - `filename` — nazwa pliku
  - `argv` — lista argumentów zakończona `NULL` (pierwszy argument — nazwa programu!!)
  - `envp` — tablica zmiennych środowiskowych w postaci `klucz=wartosc`, zakończona `NULL`
- Wartość zwracana — brak lub -1 w wypadku błędu