Laboratorium 6 – Obsługa procesów w systemie

Proces (zwany też zadaniem) jest jednostką aktywną, kontrolowaną przez system operacyjny i związaną z wykonywanym programem. Proces ma przydzielone zasoby typu pamięć (segment kodu, segment danych, segment stosu, segment danych systemowych), procesor, urządzenia zewnętrzne itp. Część przydzielonych zasobów jest do wyłącznej dyspozycji procesu (np. segment danych, segment stosu), część jest współdzielona z innymi procesami (np. procesor, segment kodu w przypadku współbieżnego wykonywania tego samego programu w ramach kilku procesów).

Każdy proces, z wyjątkiem procesu systemowego *init* o identyfikatorze 1, tworzony jest przez inny proces, który staje się jego przodkiem zwanym też procesem rodzicielskim lub po prostu rodzicem. Nowo utworzony proces nazywany jest potomkiem lub procesem potomnym. Procesy tworzą drzewiastą strukturę hierarchiczną, podobnie jak katalogi.

Potomek tworzony jest w wyniku wywołania przez przodka funkcji systemowej **`fork`**.

`int fork(void)` – tworzy proces potomny, funkcja zdefiniowana w pliku `unistd.h`.

`pid` – identyfikator procesu potomnego, zwracany w procesie macierzystym,

`0` – proces potomny,

`-1` – błąd.

W momencie wywołania funkcji (przez proces który właśnie staje się przodkiem) tworzony jest proces potomny, który wykonuje współbieżnie ze swoim przodkiem ten sam program. Potomek rozpoczyna wykonywanie programu od wyjścia z funkcji **`fork`** i kontynuuje wykonując kolejną instrukcję, znajdującą się w programie po wywołaniu funkcji **`fork`**. Do funkcji **`fork`** wchodzi zatem tylko proces macierzysty, a wychodzą z niej dwa procesy: macierzysty i potomny, przy czym każdy z nich otrzymuję inną wartość zwrotną funkcji **`fork`**. Wartością zwrotną funkcji **`fork`** w procesie macierzystym jest identyfikator (pid) potomka, a w procesie potomnym wartość 0. W przypadku błędnego wykonania funkcji **`fork`** potomek nie zostanie utworzony, a proces wywołujący otrzyma wartość -1.

`int getpid(void)` – zwraca identyfikator procesu, funkcja zdefiniowana w pliku `unistd.h`.

`pid` – identyfikator procesu własnego,

`-1` – błąd.

`int getppid(void)` – zwraca identyfikator procesu macierzystego, funkcja zdefiniowana w pliku `unistd.h`.

`pid` – identyfikator procesu macierzystego,

`-1` – błąd.

void exit(int status) – kończy działanie procesu, który ją wykonał i powoduje przekazanie w odpowiednie miejsce tablicy procesów wartości status, która może zostać odebrana i zinterpretowana przez proces macierzysty. Jeśli wartość `status=0` oznacza to, że proces zakończył się poprawnie. Jeśli proces macierzysty został zakończony, a istnieją jego procesy potomne to wykonanie ich nie jest zakłócone, ale zmienia się ich identyfikator procesu macierzystego, otrzymuje on wartość 1 będącą identyfikatorem procesu systemowego *init*. Funkcja zdefiniowana w pliku `sys/types.h`

int wait(int *status) – oczekiwanie na zakończenie procesu potomnego. Funkcja zwraca identyfikator (pid) procesu, który się zakończył.
`status` – status zakończenia procesu w przypadku zakończenia w sposób normalny lub numer sygnału w przypadku zabicia potomka lub wartość `NULL`, w przypadku, gdy informacja o stanie zakończenia procesu nie jest istotna. Gdy działa para procesów potomnych zakończenie jednego z nich powoduje powrót z funkcji **wait**. Jeżeli funkcja **wait** zostanie wywołana w procesie macierzystym przed zakończeniem procesu potomnego, wykonywanie procesu macierzystego zostanie zawieszone do momentu zakończenia potomka. Jeżeli proces potomny zakończył działanie przed wywołaniem funkcji **wait**, powrót z funkcji **wait** nastąpi natychmiast. Funkcja zdefiniowana w plikach `sys/types.h, sys/wait.h`.

Rodzina funkcji **exec** wywołuje program podany jako parameter, funkcje zdefiniowane są w pliku `unistd.h`.

```
int execl (char *path, char *arg0,…, char *argn, char *null)
int execlp(char *file, char *arg0,…, char *argn, char *null)
int execv (char *path, char * argv[])
int execvp(char *file, char * argv[])
...
```

W ramach istniejącego procesu może nastąpić uruchomienie innego programu w wyniku wywołania jednej z funkcji systemowych `execl`, `execlp`, `execle`, `execv`, `execvp`, `execve`. Funkcje te określane są ogólną nazwą **exec**. Bezbłędne wykonanie funkcji **exec** oznacza bezpowrotne zaprzestanie wykonywania bieżącego programu i rozpoczęcie wykonywania programu, którego nazwa jest przekazana przez argument. Funkcje `execlp` oraz `execvp` szukają pliku wykonywalnego na podstawie ścieżki przeszukiwania podanej w zmiennej środowiskowej PATH. Jeśli zmienna ta nie istnieje, przyjmowana jest domyślna ścieżka `:/bin:/usr/bin`.
Np.: `execl("/bin/ls", "ls", "-l", null)` – powoduje zaprzestanie wykonywania programu i wyświetlenie listy plików w bieżącym katalogu.
Funkcje **exec** nie tworzą nowego procesu, tak jak w przypadku funkcji **fork**!!!