Programowanie Współbieżne

Laboratorium 2

Procesy i sygnały

1. Funkcja fork.

a) Jedynym sposobem utworzenia nowego procesu w Unixie jest **wywołanie funkcji:** `int fork().` Wywołanie to **nie dotyczy** procesu `INIT`.

b) Funkcję tą wywołuje się **w dwóch** przypadkach:
- gdy proces wywołujące chce utworzyć swoją kopię tak, aby jedna z nich mogła wykonać inne zadania
- gdy proces chce wykonać drugi program, wtedy kopia procesu wykonuje polecenie `exec` – działanie jest to typowe dla programów typu shell

c) Proces wywołujący `fork` nazywany jest **procesem macierzystym** lub przodkiem.

d) Proces powstały na skutek wywołania funkcji `fork` nazywany jest **procesem potomnym** lub potomkiem.

e) Wywołanie funkcji `fork` **zwraca wartości**:
- **PID nowego procesu** do procesu macierzystego
- **0** do nowego procesu
- **-1** w przypadku wystąpienia błędu wywołania

f) **Proces potomny kopiuje z procesu macierzystego** następujące wartości:
- rzeczywisty identyfikator użytkownika
- rzeczywisty identyfikator grupy
- obowiązujący identyfikator użytkownika
- obowiązujący identyfikator grupy
- identyfikator grupy procesów
- identyfikator grupy terminali
- roboczy katalog bieżący
- ustalenia dotyczące obsługi sygnałów
- maskę trybów dostępu do plików

g) **Różnice pomiędzy procesem potomnym a procesem macierzystym:**
- proces potomny ma nowy jednoznaczny identyfikator procesu (zazwyczaj większy o 1 względem procesu macierzystego)
- proces potomny posiada inny identyfikator procesu macierzystego względem identyfikatora ppid dla procesu macierzystego
- proces potomny posiada własne kopie deskryptorów plików procesu macierzystego

2. Funkcja exit

a) Funkcja `exit` służy do **zakończenia procesu**. Po jej wykonaniu nigdy nie następuje powrót do procesu, który ją wywołał.

b) Funkcja `exit` **przyjmuje argument, który jest liczbą całkowitą**, która oznacza stan końcowy proces.
- exit(`0`) – poprawne zakończenie wykonania procesu
- `exit(!0)` – oznacza wystąpienie błędu

3. Funkcja wait.

a) Powoduje, że **proces oczekuje aż jeden z jego potomków zakończy działanie**.
b) Wait **zwraca `PID` procesu potomnego**, który został zakończony w wyniku wywołania funkcji `exit`, w wyniku nadania sygnału lub wykonywania go w trybie śledzenia.
c) Jeżeli proces nie posiada żadnego potomka to funkcja **zwraca wartość -1**.
d) Gdy proces macierzysty posiada procesy potomne to oczekuje na zakończenie się jednego z procesów potomnych.
e) Gdy proces potomny kończy działanie to proces macierzysty **dostaje sygnał `SIGCLD`**, który można również zignorować, aby proces macierzysty ignorował zachowanie procesów potomnych (procesy zombie będą automatycznie usuwane).
f) W przypadku, gdy proces macierzysty nie wywołał funkcji `wait` to proces potomny, który ma być zakończony staje się **procesem zombie** – jego stan utrzymywany jest do momentu upomnienia się o niego przez proces macierzysty.
g) **`PPID` dla procesu zombie staje się 1**, ponieważ rodzicem procesu zombie staje się proces `INIT` o `PID` równym 1.
h) Istnieje funkcja **`wait3`**, która ma możliwość nieczekania na zakończenie się procesu potomnego.

4. Funkcja signal.

a) Sygnał jest **informacją dla procesu, że wystąpiło jakieś zdarzenie**. Sygnały inaczej nazywamy **przerwaniami programowymi**.
b) Sygnały są **wysyłane asynchronicznie**.
c) Każdy sygnał ma nazwę opisaną w `signal.h`.
d) Sygnały można wysyłać z jednego procesu do drugiego lub z jądra do procesu.
e) Do wysyłania sygnałów służy **funkcja systemowa `kill`**.
f) Proces może dostarczyć funkcje, które będą wywoływane za każdym razem, gdy pojawi się specjalny rodzaj sygnału – **procedura obsługi sygnału**.
g) Proces może **zignorować wszystkie sygnały oprócz `SIGKILL`** oraz **`SIGSTOP`**.
h) W celu określenia jak sygnał ma być obsłużony to proces **wywołuje funkcję `signal(sygnał, procedura).`** Pierwszym parametrem jest jeden z sygnałów, drugim może być procedura obsługi lub **parametr stały wykonywania czynności domyślnych** (`SIG_DFL`) lub **parametr stały ignorowania sygnału** (`SIG_IGN`).
i) Oznaczenia sygnałów:
   - `SIGALRM` – budzik na określoną liczbę sekund
   - `SIGCLD` – zakończenie procesu potomnego
   - `SIGHUP` – zawieszenie pracy procesu w przypadku zamknięcia terminala sterującego lub wysyłany jest w przypadku zakończenia pracy przez proces przywódczy
   - `SIGINT` – znak przerwania
   - `SIGKILL` – bezwzględne zakończenie procesu
   - `SIGPIPE` – dane nie są odbierane z łącza komunikacyjnego
   - `SIGQUIT` – znak zakończenia
   - `SIGSEGV` – naruszenie segmentacji
   - `SIGSTOP` – bezwzględne zatrzymanie procesu, można reaktywować przez `SIGCONT`
   - `SIGTERM` – programowe zakończenie procesu
   - `SIGUSR1/SIGUSR2` – sygnały zdefiniowane przez użytkownika
j) W celu **zablokowania** jednego lub więcej sygnałów można wywołać funkcję `sigblock`. **Odblokowanie** następuje przez wywołanie `sigsetmask`.

5. Funkcja exec.

a) W celu wykonania programu należy **wywołać funkcje `exec`**.
b) **`Exec` zastępuje program bieżącego procesu nowym programem**.
c) Proces, który wywołał funkcję systemową `exec` jest nazywany **procesem wywołującym**.
d) Program, który ma być wykonany nazywany jest **nowym programem**.
e) Nowy program **odziedziczy następujące cechy**:
   - identyfikator procesu
   - identyfikator procesu macierzystego
   - identyfikator grupy procesów
   - identyfikator grupy terminali
   - rzeczywisty identyfikator grupy
   - rzeczywisty identyfikator użytkownika
   - katalog główny
   - roboczy katalog bieżący
   - maskę trybów dostępu do plików
   - pliki zajęte

6. Podstawowe pojęcia związane z procesami.

a) **Proces** – program sekwencyjny w trakcie wykonywania.
b) **Procesy współbieżne** – procesy, których wykonanie może, ale nie musi przebiegać równolegle. Jeden proces zaczął się przed końcem drugiego.
c) **Procesy równoległe** – procesy współbieżne wykonywane w tym samym czasie.
d) **Procesy zależne** – dwa procesy nazywamy zależnymi, jeżeli fakt wykonywania któregokolwiek z nich wpływa na wykonywanie się drugiego.
e) **Zmienna dzielona** – zmienna wspólna, wykorzystywana przez kilka współbieżnych procesów.
f) **Sekcja krytyczna** – fragment procesu, w którym korzysta on ze zmiennej dzielonej.
g) **Synchronizacja** – uporządkowanie akcji poszczególnych procesów w czasie.
h) **Blokada procesów** – każdy proces ze zbioru P czeka na zdarzenie, które może być spowodowane wyłącznie przez inny proces z tego zbioru.
i) **Głodzenie procesów** – sytuacja, w której proces jest nieskończenie wstrzymywane, gdyż zdarzenie, na które oczekuje powoduje wznawianie innych procesów.
j) **Aktywne czekanie** – proces czekający na zdarzenie bez przerwy sprawdza, czy ono już wystąpiło angażując niepotrzebnie czas procesora.
k) **Uczciwość mocna** – jeżeli proces wykonując się co pewien czas sprawdza, czy potrzebny zasób jest już dostępny, to po pewnym czasie otrzyma przydział tego zasobu.
l) **Uczciwość słaba** – w momencie zgłoszenia zapotrzebowania proces zostaje zawieszony i czeka na przydział zasobu.

7. Pytania i odpowiedzi.

a) Czy exec tworzy nowy proces?
b) Czy można przechwycić SIGSTOP?
c) Czy proces potomny posiada taki sam identyfikator procesu macierzystego jak proces macierzysty?
d) Czym różnią się procesy współbieżne od równoległych?
e) Czy można obsłużyć SIGKILL?

f) Jeśli proces macierzysty nie zainteresuje się zakończeniem procesu potomnego, to proces potomny staje się: f.1) demonem f.2) zombie f.3) drakulą
Czy można jakoś zapobiec powyższej sytuacji?
g) Czy po wykonaniu exec() proces dziedziczy id po wykonaniu procesu?
h) Czy po wykonaniu exec() można powrócić do procesu, który go wywołał?
i) Czy po wykonaniu fork() proces potomny dziedziczy id grupy?
j) Czy sygnał SIGSTOP można obsłużyć?
k) Co to jest proces?
l) Czy proces potomny kopiuje z procesu macierzystego maskę trybów dostępu do pliku?
m) Czy sygnały wysyłane są synchronicznie?
n) Jaka kombinacja klawiszy wywoła SIGINT?
o) Jaki sygnał otrzymuje proces macierzysty po zakończeniu procesu potomnego?
p) Wskaż, gdzie opisane są sygnały.
q) Jaki identyfikator posiada proces zombie, gdy jest przejmowany przez proces macierzysty? Jak się nazywa ten proces?
r) Podaj argument funkcji exit, który oznacza poprawne zakończenie procesu.
s) Czy funkcja wait może przyjąć parametr NULL?
t) Co zwraca funkcja wait jeśli proces macierzysty nie posiada procesów potomnych?
u) Czy funkcja wait może zostać wykonana bez parametrów?
v) Jak nazywamy proces wywołujący funkcję exec?
w) Sygnał kończący bezwzględnie proces to?
x) Do czego służy funkcja exit?
y) Nowy program utworzony funkcją exec kopiuje czynności obsługi sygnałów?
z) Podaj funkcję służącą do wysyłania sygnałów.

Odpowiedzi:

Ad a) nie
Ad b) nie
Ad c) nie
Ad d) **Procesy współbieżne** – procesy, których wykonanie może, ale nie musi przebiegać równolegle. Jeden proces zaczął się przed końcem drugiego. **Procesy równoległe** – procesy współbieżne wykonywane w tym samym czasie.
Ad e) nie
Ad f) zombie; ignorując sygnał SIGCLD
Ad g) tak
Ad h) tak, jeśli exec zwróci błąd
Ad i) tak
Ad j) nie
Ad k) Proces to program sekwencyjny w trakcie wykonywania.
Ad l) tak
Ad m) nie
Ad n) CTRL+Z
Ad o) SIGCLD
Ad p) signal.h
Ad q) 1, INIT
Ad r) 0
Ad s) tak
Ad t) -1

Ad u) nie
Ad v) proces wywołujący
Ad w) SIGKILL
Ad x) do zakończenia procesu
Ad y) nie
Ad z) kill

Opracowano dn. 20.10.2017
Bartłomiej Osak, Tomasz Odzimek