|  |  |
| --- | --- |
| Laboratorium **Programowanie współbieżne**  Wydział Elektrotechniki Automatyki i Informatyki  Politechnika Świętokrzyska | |
| Studia: **Stacjonarne I stopnia** | Kierunek: **Informatyka** |
| Data wykonania:  **15.10.2019** | Grupa: 3**ID11B** |
| Numer laboratorium:  **2** | Autor: Karol Skorek |
| Temat laboratorium: |
| **Forki i sygnały. Poznajemy finkcje fork(), wait(), signal()** |

1. Wstęp teoretyczny

- fork() – jedna z podstawowych funkcji systemowych. Dzięki niej jest możliwość powołania do życia nowego procesu w relacji rodzic – dziecko. Proces, w którego kodzie znajdzie się wywołanie tej funkcji, staje sie ojcem nowego procesu. W celu utworzenia nowego procesu posługujemy się składnią int pid = fork(). Spowoduje ta linia, że od tej chwili będą żyły dwa osobne procesy o indywidualnych identyfikatorach, czyli liczbie jaką zwróci funkcja fork() - procesowi macierzystemu zwrócony zostanie identyfikator syna, procesowi potomnemu - zero.  Gdy system uzna, że nowego procesu nie da się utworzyć (na przykład z powodu braku pamięci lub przekroczenia limitów na liczbę istniejących procesów), funkcja fork() zwróci wartość -1 i, oczywiście, nie powstanie żaden nowy proces.

- wait() - wstrzymuje proces wywołujący do momentu, aż jeden z procesów potomnych [przestanie dzia](http://students.mimuw.edu.pl/SO/Linux/Temat02/mw.html#PBPU)łać. Jeżeli zakończony potomek juz jest, to funkcja wraca natychmiast.  
Implementacja funkcji wait() jest funkcją systemową [sys\_wait4](http://students.mimuw.edu.pl/SO/Linux/Kod/kernel/exit.c.html#SYS_WAIT4)(), Funkcje ta wywołuje inna funkcja systemowa - sys\_waitpid(), a ta zaś funkcja sys\_wait().  
Parametrem funkcji wait() jest wskaźnik do zmiennej typu int.

Funkcja wait() poszukujac procesu potomnego, ktory przestal dzialac rozpatruje procesy ZOMBIE i STOPPED.

* ZOMBIE

 wpis w tablicy [procesów](https://pl.wikipedia.org/wiki/Proces_(informatyka)) opisujący program, którego wykonanie w [systemie operacyjnym](https://pl.wikipedia.org/wiki/System_operacyjny) zostało zakończone, ale którego zamknięcie nie zostało jeszcze obsłużone przez proces rodzica. W celu uniknięcia takiej sytuacji wykorzystujemy funkcje wait() lub pokrewnych. Nie można procesów zombie wyeliminować poleceniem kill. Zjawisko jest niepożądane, gdyż prowadzi do destabilizacji pracy programu

* STOPPED

Proces przechodzi w stan STOPPED po otrzymaniu sygnalu SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU (2 ostatnie nie wystepuja w systemie V). Uprzednio numer sygnalu, ktory zatrzymal proces jest wpisywany do pola exit\_code. Ojciec przy przegladaniu swoich synow w funkcji wait w przypadku procesu STOPPED sprawdza, czy w tym polu jest zero. Jesli nie, to wait zwraca identyfikator znalezionego procesu wczesniej wyzerowawszy to pole0. Jesli tak, to znaczy, ze ojciec juz sie dowiedzial o zatrzymaniu procesu potomnego i nie ma sensu informowac go o tym powtornie. Wtedy przeszukiwanie jest kontynuowane.

- signal() –

W języku C jest możliwość przechwycenia różnych sygnałów systemowych. Działa to podobnie jak obsługa wyjątków w innych językach.

Każdy sygnał ma swój konkretny numer w systemie. Np. gdy kończymy program wciskając Ctrl+C, to zostaje wysłane przerwanie SIGINT o numerze 2. W samym systemie mamy do dyspozycji kilka zdefiniowanych przerwań, które wykonują coś innego.

SIGINT - Proces został przerwany;

SIGQUIT - Ktoś poprosił proces o zatrzymanie i zrobienie zrzutu pamięci do pliku systemowego;

SIGFPE - Błąd operacji zmiennoprzecinkowej;

SIGTRAP - Debugger pyta, gdzie jest proces;

SIGSEGV - Proces próbował uzyskać dostęp do niewłaściwego miejsca w pamięci;

SIGWINCH - Zmieniono wielkość okna terminala;

SIGTERM - Ktoś własnie poprosił jądro systemu o skasowanie procesu;

SIGPIPE - Proces zapisał coś do potoku, którego nikt nie odczytuje.

Do wyświetlenia listy sygnałów wystarczy wpisać kill –l

W bibliotece signal.h można znaleźć numery dostępnych sygnałów oraz które z nich są UNIX-owe

#define NSIG 23

#define SIGINT 2

#define SIGILL 4

#define SIGABRT\_COMPAT 6

#define SIGFPE 8

#define SIGSEGV 11

#define SIGTERM 15

#define SIGBREAK 21

#define SIGABRT 22       /\* used by abort, replace SIGIOT in the future \*/

#define SIGABRT2 22

#ifdef \_POSIX

#define    SIGHUP    1    /\* hangup \*/

#define    SIGQUIT    3    /\* quit \*/

#define    SIGTRAP    5    /\* trace trap (not reset when caught) \*/

#define SIGIOT  6       /\* IOT instruction \*/

#define    SIGEMT    7    /\* EMT instruction \*/

#define    SIGKILL    9    /\* kill (cannot be caught or ignored) \*/

#define    SIGBUS    10    /\* bus error \*/

#define    SIGSYS    12    /\* bad argument to system call \*/

#define    SIGPIPE    13    /\* write on a pipe with no one to read it \*/

#ifdef \_\_USE\_MINGW\_ALARM

#define    SIGALRM    14    /\* alarm clock \*/

#endif

#endif

W celu wywołania wystarczy skorzystać z funkcji signal(SIGINT,SIG\_IGN).

1. Praca na laboratorium

Podczas zajęć napisałem następujące kody do zajęć

- zad 2

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

int main()

{

int childpid;

printf("startuje proces macierzysty pid %d\n",getpid());

if ((childpid = fork()) == -1)

{

perror("nie moge forknac");

exit(1);

}

else

if (childpid ==0 )

{

printf("Proces potomny o pidzie %d z rodzica %d\n",getpid(),getppid());

}

else

{

printf("Proces macierzysty o pidzie %d i dziecku %d\n",getpid(),childpid);

}

return 0;

}

– wyniki

startuje proces macierzysty pid 1864

Proces macierzysty o pidzie 1864 i dziecku 1865

Proces potomny o pidzie 1865 z rodzica 1864

- zad3a

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

int main()

{

int childpid;

printf("startuje proces macierzysty pid %d\n",getpid());

if ((childpid = fork()) == -1)

{

perror("nie moge forknac");

exit(1);

}

else

if (childpid ==0 )

{

sleep(2);

printf("Proces potomny o pidzie %d z rodzica %d\n",getpid(),getppid());

}

else

{

printf("Proces macierzysty o pidzie %d i dziecku %d\n",getpid(),childpid);

exit(2);

}

return 0;

}

startuje proces macierzysty pid 1867

Proces macierzysty o pidzie 1867 i dziecku 1868

Proces potomny o pidzie 1868 z rodzica 1

-zad3b

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

int main()

{

int childpid;

printf("startuje proces macierzysty pid %d\n",getpid());

if ((childpid = fork()) == -1)

{

perror("nie moge forknac");

exit(1);

}

else

if (childpid ==0 )

{

printf("Proces potomny o pidzie %d z rodzica %d\n",getpid(),getppid());

exit(2);

}

else

{

sleep(2);

printf("Proces macierzysty o pidzie %d i dziecku %d\n",getpid(),childpid);

}

return 0;

}

startuje proces macierzysty pid 1872

Proces potomny o pidzie 1873 z rodzica 1872

Proces macierzysty o pidzie 1872 i dziecku 1873

-zad4

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

int main()

{

int childpid;

int status = 1;

printf("startuje proces macierzysty pid %d\n",getpid());

if ((childpid = fork()) == -1)

{

perror("nie moge forknac");

exit(-1);

}

else

if (childpid ==0 )

{

printf("Proces potomny o pidzie %d z rodzica %d\n",getpid(),getppid());

exit(2);

}

else

{

wait(&status);

printf("Proces macierzysty o pidzie %d i dziecku %d\n",getpid(),childpid);

printf("Status:%d\n ",status);

}

return 0;

}

startuje proces macierzysty pid 1875

Proces potomny o pidzie 1876 z rodzica 1875

Proces macierzysty o pidzie 1875 i dziecku 1876

Status:512

- zad 5

- w tym zadaniu miałem błąd (zad 5)

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

int main()

{

int childpid;

for(int i=0;i<5;i++){

if ((childpid = fork()) == -1)

{

perror("nie moge forknac");

exit(-1);

}

else

if (childpid ==0 )

{

printf("Proces potomny o pidzie %d z rodzica %d\n",getpid(),getppid());

exit(1);

}

else

{

printf("Proces macierzysty o pidzie %d i dziecku %d\n",getpid(),childpid);

if(i==4){

}else{

sleep(5);

wait(NULL);

}

}

}

return 0;

}

Proces macierzysty o pidzie 1905 i dziecku 1906

Proces potomny o pidzie 1906 z rodzica 1905

Proces macierzysty o pidzie 1905 i dziecku 1907

Proces potomny o pidzie 1907 z rodzica 1905

Proces macierzysty o pidzie 1905 i dziecku 1909

Proces potomny o pidzie 1909 z rodzica 1905

Proces macierzysty o pidzie 1905 i dziecku 1910

Proces potomny o pidzie 1910 z rodzica 1905

Proces macierzysty o pidzie 1905 i dziecku 1911

Proces potomny o pidzie 1911 z rodzica 1905

-zad 6

#include <stdlib.h>

#include <stdio.h>

#include <unistd.h>

#include <sys/wait.h>

int main()

{

int childpid;

for(int i=0;i<5;i++){

if ((childpid = fork()) == -1)

{

perror("nie moge forknac");

exit(-1);

}

else

if (childpid ==0 )

{

if(i==4){ sleep(5);

}else{

wait(NULL);

}

printf("Proces potomny o pidzie %d z rodzica %d\n",getpid(),getppid());

}

else

{

printf("Proces macierzysty o pidzie %d i dziecku %d\n",getpid(),childpid);

wait(NULL);

exit(1);

}

}

return 0;

}

roces macierzysty o pidzie 1914 i dziecku 1915

Proces potomny o pidzie 1915 z rodzica 1914

Proces macierzysty o pidzie 1915 i dziecku 1916

Proces potomny o pidzie 1916 z rodzica 1915

Proces macierzysty o pidzie 1916 i dziecku 1917

Proces potomny o pidzie 1917 z rodzica 1916

Proces macierzysty o pidzie 1917 i dziecku 1918

Proces potomny o pidzie 1918 z rodzica 1917

Proces macierzysty o pidzie 1918 i dziecku 1919

Proces potomny o pidzie 1919 z rodzica 1918

-zad 7

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

#include <signal.h>

#include <sys/wait.h>

void oblsluga\_zakonczenia\_dziecka(int nr\_sig)

{

printf("Rodzic już się dowiedział o zakończeniu procesu %d\n",wait(NULL));

}

int main()

{

signal(SIGCHLD,oblsluga\_zakonczenia\_dziecka);

int childpid;

printf("startuje proces macierzysty pid %d\n",getpid());

if ((childpid = fork()) == -1)

{

perror("nie moge forknac");

exit(1);

}

else

if (childpid ==0 )

{

printf("Proces potomny o pidzie %d z rodzica %d\n",getpid(),getppid());

}

else

{

sleep(2);

printf("Proces macierzysty o pidzie %d i dziecku %d\n",getpid(),childpid);

}

return 0;

}

Proces macierzysty o pidzie 1925 i dziecku 1926

Proces potomny o pidzie 1926 z rodzica 1925

Rodzic już się dowiedział o zakończeniu procesu 1926

1. Wnioski

Prawie wszystkie zadania przebiegły pomyślnie. Ćwiczenia pozwoliły mi przetestować w praktyce działanie funkcji systemowych omówionych w instrukcji. Ponadto pozwoliło mi to bardziej panować nad programem. Wiedza przypomniana na laboratorium z pewnością przyda się na laboratorium w przyszłości.