**Wojskowa Akademia Techniczna**

**Wydział Elektroniki**

**Instytut Telekomunikacji**

**Studia II°**

**Projektowanie aplikacji sieciowych**

Materiały pomocnicze do zajęć

Cz.1.

**dr inż. Jarosław Krygier**

Warszawa 2017

Spis treści

[1 Procesy potomne (1) 3](#_Toc495309016)

[2 Procesy potomne (2) 5](#_Toc495309017)

[3 Wielkorotne tworzenie procesów potomnych 7](#_Toc495309018)

[4 Kończenie procesów potomnych 9](#_Toc495309019)

# Procesy potomne (1)

/\*

============================================================================

Name : PAS\_1.c

Author :

Version :

Copyright : J. Krygier

Description : Procesy potomne

============================================================================

\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h> //fork(), getpid()

#include <sys/types.h> //pid\_t

int main(void) {

//deklaracja zmiennych

pid\_t pid; // zmianna na identyfikator procesu (tak naprawde to int)

int a = 1; //zmianna manipulacyjna

//pobierz identyfikator procesu glownego (macierzystego)

printf ("Identyfikator procesu glownego (macierzystego): %d \n", getpid());

sleep (5);

printf ("Utworz proces potomny (fork()) \n \n");

//Uwaga pamiec procesu macierzystego zostala skopiowana do potomnego

pid = fork ();

if (pid == -1) {

//jesteśmy w rodzicu, ale blad w utworzeniu nowego procesu

printf ("Blad w utworzeniu procesu \n");

exit (0); // przerywam program

} else if (pid == 0){

///////////////////PROCES POTOMNY/////////////////////////////

printf ("-------------------\n");

printf ("Identyfikator procesu potomnego: %d \n", getpid());

//Jestem w procesie potomnym

//Jaka jest wartosc zmiennej a?

printf ("Proces potomny -> a: %d \n", a);

//Zmieniam wartosc a w procesie macierzystym

a = 2;

printf ("Proces potomny (zmienilem a) -> a: %d \n", a);

int b = 200;

printf ("Dostep do zmiennej zainicjowanej rowniez w procesie mac. -> b: %d \n", b);

///////////////////PROCES POTOMNY/////////////////////////////

} else {

///////////////////PROCES MACIERZYSTY/////////////////////////

printf ("-------------------\n");

printf ("Identyfikator procesu macierzystego: %d \n", getpid());

//Jestem w procesie macierzystym

//Jaka jest wartosc zmiennej a?

printf ("Proces macierzysty -> a: %d \n", a);

//Zmieniam wartosc a w procesie potomnym

a = 3;

printf ("Proces macierzysty (zmienilem a) -> a: %d \n", a);

int b = 100; //druga deklaracja

printf ("Dostep do zmiennej zainicjowanej rowniez w procesie pot. -> b: %d \n", b);

////////////////////PROCES MACIERZYSTY/////////////////////////

}

sleep (5);

printf ("==================\n");

//Teraz wykonuje sie to samo w pbu procesach

printf ("Odczyt w obu procesach (pid: %d) -> a: %d \n",getpid (), a);

//Zmieniam wartosc a w pamieci obu procesow:

a = 10;

printf ("Zmiana w obu proceach (pid: %d) teraz -> a: %d \n",getpid (), a);

while (1) {

sleep (10); // petla nieskonczona zeby zobaczyc pid w termianlu

}

return EXIT\_SUCCESS;

}

# Procesy potomne (2)

/\*

============================================================================

Name : PAS\_1\_A.c

Author :

Version :

Copyright : Your copyright notice

Description : Procesy potomne \_ case zamiast if

============================================================================

\*/

// INNY SPOSOB NA STRUKTUTE PROGRAMU Z FORK()

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h> //fork(), getpid()

#include <sys/types.h> //pid\_t

int main(void) {

//deklaracja zmiennych

//pid\_t pid; // zmianna na identyfikator procesu (tak naprawde to int)

int a = 1; //zmianna manipulacyjna

//pobierz identyfikator procesu glownego (macierzystego)

printf ("Identyfikator procesu glownego (macierzystego): %d \n", getpid());

sleep (5);

printf ("Utworz proces potomny (fork()) \n \n");

//Uwaga pamiec procesu macierzystego zostala skopiowana do potomnego

switch(fork()) {

case -1:

//jesteśmy w rodzicu, ale blad w utworzeniu nowego procesu

printf ("Blad w utworzeniu procesu \n");

exit (0); // przerywam program (zamiast brake;)

case 0:

//jesteśmy w procesie potomnym

///////////////////PROCES POTOMNY/////////////////////////////

printf ("-------------------\n");

printf ("Identyfikator procesu potomnego: %d \n", getpid());

//Jestem w procesie potomnym

//Jaka jest wartosc zmiennej a?

printf ("Proces potomny -> a: %d \n", a);

//Zmieniam wartosc a w procesie macierzystym

a = 2;

printf ("Proces potomny (zmienilem a) -> a: %d \n", a);

break;

///////////////////PROCES POTOMNY/////////////////////////////

default:

//jesteśmy w procesie macierzystym

///////////////////PROCES MACIERZYSTY/////////////////////////

printf ("-------------------\n");

printf ("Identyfikator procesu macierzystego: %d \n", getpid());

//Jestem w procesie macierzystym

//Jaka jest wartosc zmiennej a?

printf ("Proces macierzysty -> a: %d \n", a);

//Zmieniam wartosc a w procesie potomnym

a = 3;

printf ("Proces macierzysty (zmienilem a) -> a: %d \n", a);

break;

////////////////////PROCES MACIERZYSTY/////////////////////////

}

sleep (5);

printf ("==================\n");

//Teraz wykonuje sie to samo w pbu procesach

printf ("Odczyt w obu procesach (pid: %d) -> a: %d \n",getpid (), a);

//Zmieniam wartosc a w pamieci obu procesow:

a = 10;

printf ("Zmiana w obu proceach (pid: %d) teraz -> a: %d \n",getpid (), a);

while (1) {

sleep (10); // petla nieskonczona zeby zobaczyc pid w termianlu

}

return EXIT\_SUCCESS;

}

# Wielkorotne tworzenie procesów potomnych

/\*

============================================================================

Name : PAS\_2.c

Author : J. Krygier

Version :

Copyright : Do wykorzystania w ramach zajec

Description : Wielkorotne tworzenie procesow potomnych - dziecko ma dziecko

============================================================================

\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h> //fork(), getpid()

#include <sys/types.h> //pid\_t

int main(void) {

//deklaracja zmiennych

pid\_t pid; // zmianna na identyfikator procesu (tak naprawde to int)

sleep (1);

printf ("Utworz proces potomny (fork()) \n \n");

pid = fork ();

if (pid == 0){

///////////////////PROCES POTOMNY/////////////////////////////

printf ("-------------------\n");

printf ("Identyfikator procesu potomnego: %d \n", getpid());

//Jestem w procesie potomnym 1

//.....

///////////////////PROCES POTOMNY/////////////////////////////

} else {

///////////////////PROCES MACIERZYSTY/////////////////////////

//|

printf ("==============\n");

printf ("Identyfikator procesu macierzystego: %d \n", getpid());

//Jestem w procesie macierzystym

//|.....

///////////////////PROCES MACIERZYSTY/////////////////////////

//utworze nowy proces potomy

if (fork()==0) {

//+++++++NOWY PROCES POTOMNY++++++++++++

printf ("+++++++++++++\n");

printf ("Identyfikator nowego procesu potomnego: %d \n", getpid());

//Jestem w procesie potomnym 2

//.....

//+++++++NOWY PROCES POTOMNY++++++++++++

} else {

///////////////////PROCES MACIERZYSTY/////////////////////////

printf ("==============\n");

printf ("Identyfikator procesu macierzystego: %d \n", getpid());

//....

////////////////////PROCES MACIERZYSTY/////////////////////////

}

}

while (1);

return EXIT\_SUCCESS;

}

# Kończenie procesów potomnych

/\*

============================================================================

Name : PAS\_2.c

Author :

Version :

Copyright : Your copyright notice

Description : konczenie procesow potomnych

============================================================================

\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h> //fork(), getpid()

#include <sys/types.h> //pid\_t

#include <sys/wait.h> //wait()

int main(void) {

// ZAMYKANIE (KONCZENIE) PROCESOW - synchronizacja zakonczenia

// !!!! W pierwszej kolejnosci koncz procesy potomne a potem macierzyste

//deklaracja zmiennych

pid\_t pid, id; // zmianna na identyfikator procesu (tak naprawde to int)

sleep (1);

int i; // zmienna pomocnicza

int status; //zmienna statusu zakoczenia procesu

printf ("Utworz proces potomny (fork()) \n \n");

pid = fork ();

if (pid == 0){

///////////////////PROCES POTOMNY/////////////////////////////

printf ("-------------------\n");

printf ("Identyfikator procesu potomnego: %d \n", getpid());

//Jestem w procesie potomnym 1

printf ("pid%d: Proces potomny wykonuje sie powoli.\n", getpid());

//wywolanie instruckji systemowej: wyswietli procesy zaczynajace sie od "PAS"

system ("ps -A | grep PAS");

for (i=0; i<10; i++) {

printf (".\n");

sleep (2);

}

printf ("pid%d: Koncze proces potomny z kodem wyjscia %d\n", getpid(), 0xaa);

exit (0xaa); //kod powrtu 0xaa = 170

///////////////////PROCES POTOMNY/////////////////////////////

} else {

///////////////////PROCES MACIERZYSTY/////////////////////////

//|

printf ("==============\n");

printf ("Identyfikator procesu macierzystego: %d \n", getpid());

//Jestem w procesie macierzystym

//|.....

// gdyby proces potomny skonczyl sie zanim proces macierzysty dotrze do 'wait()',

// proces potomny staje sie tzw. procesem zombi (nieczynnym), ktory powinien sie zakonczyc

// w momencie wywolania fukcji 'wait()'

#if 0

sleep (20); // mozna to sprawdzic dodajac opoznienie proceu macierzystego

#endif

printf ("pid%d: Wstrzymuje proces macierzysty do zakonczenia procesu potomnego\n", getpid());

id = wait (&status);

//wywolanie instruckji systemowej: wyswietli procesy zaczynajace sie od "PAS"

system ("ps -A | grep PAS");

// pod zmienna ststus ukryty jest kod powrotu exit() z proc. potomnego w przedostatnim bajcie ('0xaa')

// oststni bajt dodaje wystem operacyjny: przyczyna zakonczenia procesu potomnego

printf ("pid%d: Koncze proc. macierzysty po zakonczeniu procesu potomnego, ktorego pid byl %d (status:0x%x)\n", getpid(),id,status);

// istnieja makra (w stdlib.h) do sprawdzenia przyczyny zakonczenia procesu potomnego

if (WIFEXITED(status)) {

printf("Proces potomny zakonczony exitem, status=%d\n", WEXITSTATUS(status));

} else if (WIFSIGNALED(status)) {

printf("Proces potomny zabity sygnalem %d\n", WTERMSIG(status));

} else if (WIFSTOPPED(status)) {

printf("Proces potomny zakonczony sygnalem %d\n", WSTOPSIG(status));

} else if (WIFCONTINUED(status)) {

printf("Proces potomny dziala dalej\n");

}

exit (0);

////////////////////PROCES MACIERZYSTY/////////////////////////

}

printf ("to juz sie nie wyswietli\n");

return EXIT\_SUCCESS; // to tez nie bedzie wykorzystane

}