**Mateusz Cąkała**

**Przemysław Czuba**

Treść zadania:

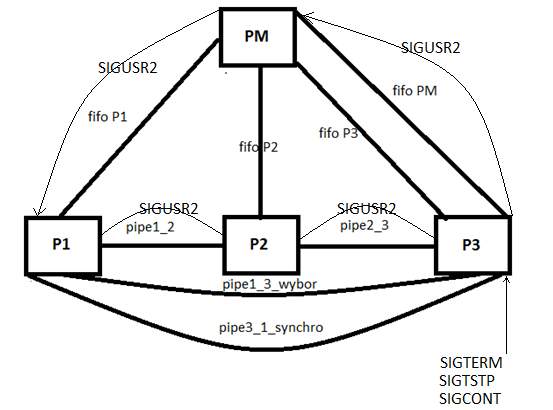
Opracować zestaw programów typu producent-konsument realizujących przy wykorzystaniu mechanizmu łączy komunikacyjnych pipes, następujący schemat komunikacji międzyprocesowej:

Proces 1: czyta dane pojedyncze wiersze ze standardowego strumienia wejściowego i przekazuje je w niezmienionej formie do procesu 2.

Proces 2: pobiera dane przesyłane przez proces 1. Oblicza ilość znaków w każdej linii i wyznaczona liczbę przekazuje do procesu 3.

Proces 3: pobiera dane wyprodukowane przez proces 2 i umieszcza je w standardowym strumieniu wyjściowym. Każda odebrana jednostka danych powinna zostać wprowadzona w osobnym wejściu.

Należy zaproponować i zaimplementować mechanizm informowania się procesów o swoim stanie. Należy wykorzystać do tego dostępny mechanizm sygnałów i łączy nazwanych (kolejek fifo). Scenariusz powiadamiania się procesów o swoim stanie wygląda następująco: do procesu 3 wysyłane są sygnały. Proces 3 przesyła otrzymany sygnał do procesu macierzystego. Proces macierzysty zapisuje wartość sygnału do kolejek fifo oraz wysyła powiadomienie do procesu 1 o odczytaniu zawartości kolejki fifo. Proces 1 po odczytaniu sygnału wysyła powiadomienie do procesu 2 o odczytanie kolejki fifo. Proces 2 powiadamia proces 3 o konieczności odczytu kolejki fifo. Wszystkie trzy procesy powinny być powoływane automatycznie z jednego procesu inicjującego.



**Kod programu:**

#include <stdio.h>

#include <sys/wait.h>

#include <unistd.h>

#include <sys/types.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <signal.h>

#include <fcntl.h>

#define SIZE 512

#define KROPKA 46

//ZMIENNE

FILE \*plik;

int wybor = 0;

char bufor\_tekstowy[SIZE], \*eof = bufor\_tekstowy, wynik, tmp, znak;

int pfd1\_2[2], pfd2\_3[2], pfd3\_1\_synchro[2], pfd1\_3\_wybor[2];

int synchro = 0, ilosc\_znakow, i;

int fd\_write, fd\_read;

char fifo\_name[100];

int pid\_reset;

int stop = 0;

//STRUKTURA KOMUNIKATU

struct msg {

pid\_t pid;

int sygnal;

}msg;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*KOLEJKI FIFO\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int mk\_fifo\_name(pid\_t pid, char\* name, size\_t name\_max) //funkcja tworzaca nazwe kolejki fifo "fifoPID"

{

snprintf(name, name\_max, "fifo%ld", (long)pid);

return 1;

}

void fifo\_send\_id(int id) //funkcja wysylajaca do kolejki fifo o nazwie fifoid, id jest argumentem funkcji

{

mk\_fifo\_name(id, fifo\_name, sizeof(fifo\_name)); //tworzenie nazwy kolejki

fd\_write = open(fifo\_name, O\_WRONLY); //otwarcie do zapisu kolejki fifo

write(fd\_write, &msg, sizeof(msg)); //zapisanie numeru sygnalu do kolejki

close(fd\_write); //zamkniecie fifo

}

void fifo\_send\_pm()

{

fd\_write = open("fifo\_PM", O\_WRONLY); //otwarcie do zapisu kolejki fifo procesu macierzystego

write(fd\_write, &msg, sizeof(msg)); //zapisanie numeru sygnalu do kolejki

close(fd\_write); //zamkniecie fifo

}

void fifo\_read\_pm()

{

fd\_read = open("fifo\_PM", O\_RDONLY); //otwarcie do odczytu kolejki fifo procesu macierzystego

read(fd\_read, &msg, sizeof(msg)); //odczytanie numeru sygnalu z kolejki

close(fd\_read); //zamkniecie fifo

}

void fifo\_name\_read()

{

fd\_read = open(fifo\_name, O\_RDONLY); //otwarcie do odczytu kolejki fifo

read(fd\_read, &msg, sizeof(msg)); //odczytanie numeru sygnalu z kolejki

close(fd\_read); //zamkniecie fifo

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SYGNALY\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*///

void sig\_term\_P1(int nr\_syg, siginfo\_t \*info, void \*context)//funkcja obslugi sygnalu 18 dla procesu p1

{

pid\_t P1 = getpid();//pobranie pid procesu 1

if (info->si\_pid == P1)//weryfikacja pid procesu wysylajacego syngal

{

unlink(fifo\_name); //usniecie kolejki

\_exit(2);//zakonczenie procesu 1

}

}

void sig\_term\_P2(int nr\_syg, siginfo\_t \*info, void \*context)//funkcja obslugi sygnalu 18 dla procesu p1

{

pid\_t P2 = getpid();//pobranie pid procesu 2

if (info->si\_pid == P2)//weryfikacja pid procesu wysylajacego syngal

{

unlink(fifo\_name); //usniecie kolejki

\_exit(2);//zakonczenie procesu 2

}

}

void sig\_term\_PM(int nr\_syg, siginfo\_t \*info, void \*context)//funkcja obslugi sygnalu 18 dla procesu p1

{

pid\_t PM = getpid(); //pobranie pid procesu macierzystego

if (info->si\_pid == PM)//weryfikacja pid procesu wysylajacego syngal

{

unlink("fifo\_PM"); //usniecie kolejki

\_exit(2);//zakonczenie procesu macierzystego

}

}

void sig\_tstp\_P1(int nr\_syg, siginfo\_t \*info, void \*context)//funkcja obslugi sygnalu 18 dla procesu p1

{

pid\_t PM = getppid();//pobranie pid procesu macierzystego

if (info->si\_pid == PM || info->si\_pid == getpid())//weryfikacja pid procesu wysylajacego syngal

{

stop = 1;//zmienna sygnalizujaca otrzymanie sygnalu 20

while (stop == 1){ ; }//petla nieskonczona dopoki zmienna stop == 1

}

}

void sig\_tstp\_P2(int nr\_syg, siginfo\_t \*info, void \*context)//funkcja obslugi sygnalu 18 dla procesu p1

{

pid\_t P1 = getppid() + 1;//pobranie pid procesu 1

if (info->si\_pid == P1 || info->si\_pid == getpid())//weryfikacja pid procesu wysylajacego syngal

{

stop = 1;//zmienna sygnalizujaca otrzymanie sygnalu 20

while (stop == 1){ ; }//petla nieskonczona dopoki zmienna stop == 1

}

}

void sig\_cont\_P1(int nr\_syg, siginfo\_t \*info, void \*context)//funkcja obslugi sygnalu 18 dla procesu p1

{

pid\_t PM = getppid();//pobranie pid procesu macierzystego

if (info->si\_pid == PM)//weryfikacja pid procesu wysylajacego syngal

{

stop = 0;//odblokowanie procesu macierzystego

}

}

void sig\_cont\_P2(int nr\_syg, siginfo\_t \*info, void \*context)//funkcja obslugi sygnalu 18 dla procesu p1

{

pid\_t PM = getppid() + 1;//pobranie pid procesu 1

if (info->si\_pid == PM)//weryfikacja pid procesu wysylajacego syngal

{

stop = 0;//odblokowanie procesu 2

}

}

void sig\_kons\_P3(int nr\_syg)//funkcja obslugi sygnalu 18 dla procesu p1

{

msg.sygnal = nr\_syg;

kill(getppid(), 12); //wyslanie sygnalu SIGUSR2 do procesu macierzystego

fifo\_send\_pm();//zapis do kolejki fifo procesu macierzystego odebranego numeru sygnalu

}

void sig\_usr2\_PM(int nr\_syg, siginfo\_t \*info, void \*context)//funkcja obslugi sygnalu 18 dla procesu p1

{

pid\_t PM = getpid() + 3;//pobranie pid procesu 3

if (info->si\_pid == PM)//weryfikacja pid procesu wysylajacego syngal

{

pid\_t PM = getpid(); //pobranie pid procesu macierzystego

int P[3] = { PM + 1, PM + 2, PM + 3 };//tablica identyfikatorow procesow

fifo\_read\_pm(); //odebranie komunikatu z fifo PM

if (msg.sygnal == 18)

kill(P[0], 18); //wzbudzenie procesu pierwszego (SIGCONT)

kill(P[0], 12); //"kopniecie" P1 zeby odebral nr sygnalu

fifo\_send\_id(P[0]); //zapis numeru sygnalu do fifo procesu1

fifo\_send\_id(P[1]); //zapis numeru sygnalu do fifo procesu2

fifo\_send\_id(P[2]); //zapis numeru sygnalu do fifo procesu3

if (msg.sygnal == 15)

raise(15); //wyslanie sygnalu SIGTERM do biezacego procesu

}

}

void sig\_usr2\_P1(int nr\_syg, siginfo\_t \*info, void \*context)//funkcja obslugi sygnalu 18 dla procesu p1

{

pid\_t PM = getppid();//pobranie pid procesu macierzystego

if (info->si\_pid == PM)//weryfikacja pid procesu wysylajacego syngal

{

int pid = pid\_reset; //pobranie pid procesu1

fifo\_name\_read(); //odczytanie numeru sygnalu z fifo procesu1

if (msg.sygnal == 15)

{

if (wybor == 2)

{

while (eof != NULL) //dokonczenie odczytywania pliku

{

read(pfd3\_1\_synchro[0], &synchro, sizeof(synchro)); //synchronizacja z P3, oczekiwanie na obliczenie ilosci znakow przez proces3

write(pfd1\_2[1], bufor\_tekstowy, sizeof(bufor\_tekstowy)); //zapis pobranej linii do pipe'a

printf("%s", bufor\_tekstowy); //wypisanie zawartosci bufora

eof = fgets(bufor\_tekstowy, sizeof(bufor\_tekstowy), plik); //pobranie linii z pliku

}

read(pfd3\_1\_synchro[0], &synchro, sizeof(synchro)); //synchronizacja z P3, oczekiwanie na obliczenie ilosci znakow przez proces3

}

}

else if (msg.sygnal == 18)

kill(++pid, 18); //wyslanie sygnalu SIGCONT do procesu2

pid = pid\_reset;

kill(++pid, 12); //wyslanie sygnalu SIGUSR2 do procesu2

if (msg.sygnal == 15) raise(15); //wyslanie sygnalu SIGTERM do biezacego procesu

else if (msg.sygnal == 20) raise(20); //wyslanie sygnalu SIGTSTP do biezacego procesu

}

}

void sig\_usr2\_P2(int nr\_syg, siginfo\_t \*info, void \*context)//funkcja obslugi sygnalu 18 dla procesu p1

{

pid\_t P1 = getppid() + 1;//pobranie pid procesu 1

if (info->si\_pid == P1)//weryfikacja pid procesu wysylajacego syngal

{

int pid = pid\_reset; //pobranie pid procesu2

fifo\_name\_read(); //odczytanie numeru sygnalu z fifo procesu2

kill(++pid, 12); //wyslanie sygnalu SIGUSR2 do procesu3

if (msg.sygnal == 15) raise(15); ////wyslanie sygnalu SIGTERM do biezacego procesu

else if (msg.sygnal == 20) raise(20); //wyslanie sygnalu SIGTSTP do biezacego procesu

}

}

void sig\_usr2\_P3(int nr\_syg, siginfo\_t \*info, void \*context)//funkcja obslugi sygnalu 18 dla procesu p1

{

pid\_t P2 = getppid() + 2;//pobranie pid procesu 2

if (info->si\_pid == P2)//weryfikacja pid procesu wysylajacego syngal

{

fifo\_name\_read(); //odczytanie numeru sygnalu z fifo procesu3

if (msg.sygnal == 15)

{

unlink(fifo\_name); //usniecie kolejki

\_exit(2); //zakonczenie procesu3

}

else if (msg.sygnal == 20) raise(19); //wyslanie sygnalu SIGSTP do biezacego procesu

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*////\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//RESET BUFORA

void reset\_bufor(void)

{

memset(bufor\_tekstowy, 0, sizeof(bufor\_tekstowy));

}

//MENU PROGRAMU

void menu(void)

{

printf("Menu programu: \n");

printf("1. Rozpocznij wprowadzanie ze standardowego wejscia.\n");

printf("2. Rozpocznij odczyt z pliku.\n");

printf("Podaj liczbe: ");

scanf("%d", &wybor); //pobranie z konsoli wyboru

write(pfd1\_3\_wybor[1], &wybor, sizeof(wybor)); //przeslanie wartosci wyboru do P3

getchar(); //czyszczenie bufora klawiatury

}

//FUNKCJA OBSLUGI STDIN (1)

void obsluga\_1\_P1(void)

{

while (bufor\_tekstowy[0] != KROPKA)//pobieranie linii z konsoli dopoki nie wprowadzono znaku kropki

{

fgets(bufor\_tekstowy, sizeof(bufor\_tekstowy), stdin);//pobranie linii i zapis do bufora

write(pfd1\_2[1], bufor\_tekstowy, sizeof(bufor\_tekstowy));//zapis bufora do pipe'a

}

}

//FUNKCJA OBSLUGI PLIKU (2)

void obsluga\_2\_P1(void)

{

eof = fgets(bufor\_tekstowy, sizeof(bufor\_tekstowy), plik);//pobranie linii z pliku

while (eof != NULL)//pobieranie kolejnych linni z pliku

{

write(pfd1\_2[1], bufor\_tekstowy, sizeof(bufor\_tekstowy));//zapis pobranej linii do pipe'a

printf("%s", bufor\_tekstowy); //wypisanie zawartosci bufora

read(pfd3\_1\_synchro[0], &synchro, sizeof(synchro)); //synchronizacja z P3, oczekiwanie na obliczenie ilosci znakow przez proces3

eof = fgets(bufor\_tekstowy, sizeof(bufor\_tekstowy), plik); //pobranie linii z pliku

}

bufor\_tekstowy[0] = KROPKA;

write(pfd1\_2[1], bufor\_tekstowy, sizeof(bufor\_tekstowy)); //po odczycie pliku zapis '.' do pipe'a P2, przerwanie petli procesu2

}

//MAIN

int main(void)

{

pid\_t proces1, proces2, proces3;

//STWORZENIE LACZY

pipe(pfd1\_2); //PIPE P1 -> P2

pipe(pfd2\_3); //PIPE P2 -> P3

pipe(pfd3\_1\_synchro); //PIPE P3->P1

pipe(pfd1\_3\_wybor); //PIPE P1->P3

//FIFO PM

mkfifo("fifo\_PM", 0666);//stworzenie fifo procesu macierzystego

//SYGNALY - DEF STRUKTUR

sigset\_t P1\_set, P2\_set, P3\_set, PM\_set;//stworzenie zbiorow masek

// zbiory sygnalow syg

sigfillset(&P1\_set); //wypelnienie maski wszystkimi sygnalami

sigdelset(&P1\_set, 12); //usuniecie z maski sygnalu 12

sigdelset(&P1\_set, 18); //usuniecie z maski sygnalu 18

sigdelset(&P1\_set, 15); //usuniecie z maski sygnalu 15

sigdelset(&P1\_set, 20); //usuniecie z maski sygnalu 20

sigfillset(&P2\_set); //wypelnienie maski wszystkimi sygnalami

sigdelset(&P2\_set, 12); //usuniecie z maski sygnalu 12

sigdelset(&P2\_set, 18); //usuniecie z maski sygnalu 18

sigdelset(&P2\_set, 15); //usuniecie z maski sygnalu 15

sigdelset(&P2\_set, 20); //usuniecie z maski sygnalu 20

sigfillset(&P3\_set); //wypelnienie maski wszystkimi sygnalami

sigdelset(&P3\_set, 12); //usuniecie z maski sygnalu 12

sigdelset(&P3\_set, 18); //usuniecie z maski sygnalu 18

sigdelset(&P3\_set, 15); //usuniecie z maski sygnalu 15

sigdelset(&P3\_set, 20); //usuniecie z maski sygnalu 20

sigfillset(&PM\_set); //wypelnienie maski wszystkimi sygnalami

sigdelset(&PM\_set, 12); //usuniecie z maski sygnalu 12

sigdelset(&PM\_set, 15); //usuniecie z maski sygnalu 15

// SYGNALY - OBSLUGA

static struct sigaction P3\_KONS;//utworzenie struktury obslugi sygnalow procesu3 wysylanych z konsoli

P3\_KONS.sa\_handler = sig\_kons\_P3;//przypisanie nowej funkcji obslugi sygnalow

P3\_KONS.sa\_flags = SA\_RESTART; //ustawienie flagi wznawiajacej dzialanie procesu od momentu otrzymania sygnalu

static struct sigaction PM\_USR2;//utworzenie struktury obslugi sygnalow procesu3 wysylanych z konsoli

PM\_USR2.sa\_sigaction = sig\_usr2\_PM;//przypisanie nowej funkcji obslugi sygnalow

PM\_USR2.sa\_flags = SA\_SIGINFO;//ustawienie flagi pozwalajacej na uzyskanie pid procesu wysylajacego syg

static struct sigaction P1\_USR2;//utworzenie struktury obslugi sygnalu USR2 procesu1

P1\_USR2.sa\_sigaction = sig\_usr2\_P1;//przypisanie nowej funkcji obslugi sygnalow

P1\_USR2.sa\_flags = SA\_RESTART + SA\_SIGINFO; //ustawienie flagi wznawiajacej dzialanie procesu od momentu otrzymania sygnalu oraz

static struct sigaction P2\_USR2;//utworzenie struktury obslugi sygnalu USR2 procesu2

P2\_USR2.sa\_sigaction = sig\_usr2\_P2;//przypisanie nowej funkcji obslugi sygnalow

P2\_USR2.sa\_flags = SA\_RESTART + SA\_SIGINFO;//ustawienie flagi wznawiajacej dzialanie procesu od momentu otrzymania sygnalu

static struct sigaction P3\_USR2;//utworzenie struktury obslugi sygnalu USR2 procesu3

P3\_USR2.sa\_sigaction = sig\_usr2\_P3;//przypisanie nowej funkcji obslugi sygnalow

P3\_USR2.sa\_flags = SA\_RESTART + SA\_SIGINFO;//ustawienie flagi wznawiajacej dzialanie procesu od momentu otrzymania sygnalu

///SIGTERM

static struct sigaction P1\_TERM;//utworzenie struktury obslugi sygnalu TERM procesu1

P1\_TERM.sa\_sigaction = sig\_term\_P1;//przypisanie nowej funkcji obslugi sygnalow

P1\_TERM.sa\_flags = SA\_RESTART + SA\_SIGINFO; //ustawienie flagi wznawiajacej dzialanie procesu od momentu otrzymania sygnalu

static struct sigaction P2\_TERM;//utworzenie struktury obslugi sygnalu TERM procesu2

P2\_TERM.sa\_sigaction = sig\_term\_P2;//przypisanie nowej funkcji obslugi sygnalow

P2\_TERM.sa\_flags = SA\_RESTART + SA\_SIGINFO; //ustawienie flagi wznawiajacej dzialanie procesu od momentu otrzymania sygnalu

static struct sigaction PM\_TERM;//utworzenie struktury obslugi sygnalu TERM procesu macierzystego

PM\_TERM.sa\_sigaction = sig\_term\_PM;//przypisanie nowej funkcji obslugi sygnalow

PM\_TERM.sa\_flags = SA\_RESTART + SA\_SIGINFO;//ustawienie flagi wznawiajacej dzialanie procesu od momentu otrzymania sygnalu

//SIGTSTP

static struct sigaction P1\_TSTP;//utworzenie struktury obslugi sygnalu TSTP procesu1

P1\_TSTP.sa\_sigaction = sig\_tstp\_P1;//przypisanie nowej funkcji obslugi sygnalow

P1\_TSTP.sa\_flags = SA\_RESTART + SA\_SIGINFO;//ustawienie flagi wznawiajacej dzialanie procesu od momentu otrzymania sygnalu

static struct sigaction P2\_TSTP;//utworzenie struktury obslugi sygnalu TSTP procesu2

P2\_TSTP.sa\_sigaction = sig\_tstp\_P2;//przypisanie nowej funkcji obslugi sygnalow

P2\_TSTP.sa\_flags = SA\_RESTART + SA\_SIGINFO;//ustawienie flagi wznawiajacej dzialanie procesu od momentu otrzymania sygnalu

//SIGCONT

static struct sigaction P1\_CONT;//utworzenie struktury obslugi sygnalu CONT procesu1

P1\_CONT.sa\_sigaction = sig\_cont\_P1;//przypisanie nowej funkcji obslugi sygnalow

P1\_CONT.sa\_flags = SA\_RESTART + SA\_SIGINFO;//ustawienie flagi wznawiajacej dzialanie procesu od momentu otrzymania sygnalu

static struct sigaction P2\_CONT;//utworzenie struktury obslugi sygnalu CONT procesu2

P2\_CONT.sa\_sigaction = sig\_cont\_P2;//przypisanie nowej funkcji obslugi sygnalow

P2\_CONT.sa\_flags = SA\_RESTART + SA\_SIGINFO;//ustawienie flagi wznawiajacej dzialanie procesu od momentu otrzymania sygnalu

//PROCES 1

proces1 = fork();//utworzenie procesu potomnego p1

switch (proces1)

{

case 0:

//STWORZENIE FIFO

pid\_reset = msg.pid = getpid();//pobranie pid procesu p1

mk\_fifo\_name(msg.pid, fifo\_name, sizeof(fifo\_name)); //tworzenie nazwy fifo P1

mkfifo(fifo\_name, 0666); //tworzenie fifo o nazwie fifoPID

//SYGNALY

sigprocmask(SIG\_BLOCK, &P1\_set, NULL);//zablokowanie sygnalow ze zbioru P1\_set

sigaction(12, &P1\_USR2, NULL); //ustawienie nowej akcji zwiazanej z sygnalem 12

sigaction(15, &P1\_TERM, NULL); //ustawienie nowej akcji zwiazanej z sygnalem 15

sigaction(20, &P1\_TSTP, NULL); //ustawienie nowej akcji zwiazanej z sygnalem 20

sigaction(18, &P1\_CONT, NULL); //ustawienie nowej akcji zwiazanej z sygnalem 18

while (1)

{

menu(); //wyswietlenie menu programu

if (wybor == 1)

obsluga\_1\_P1();//obsluga wczytywania ciagow znakow z STDIN

if (wybor == 2)

{

plik = fopen("plik\_we.txt", "r");//otwarcie pliku zrodlowego

obsluga\_2\_P1();//obsluga wczytywania ciagow znakow z pliku

fclose(plik); //zamkniecie pliku

}

reset\_bufor(); //czyszczenie bufora tekstowego

}

}

//PROCES 2

proces2 = fork();//utworzenie procesu potomnego p2

switch (proces2)

{

case 0:

//STWORZENIE FIFO

pid\_reset = msg.pid = getpid(); //pobranie pid procesu p2

mk\_fifo\_name(msg.pid, fifo\_name, sizeof(fifo\_name)); //tworzenie nazwy fifo P2

mkfifo(fifo\_name, 0666); //tworzenie fifo o nazwie fifoPID

//SYGNALY

sigprocmask(SIG\_BLOCK, &P2\_set, NULL);//zablokowanie sygnalow ze zbioru P2\_set

sigaction(12, &P2\_USR2, NULL);//ustawienie nowej akcji zwiazanej z sygnalem 12

sigaction(15, &P2\_TERM, NULL);//ustawienie nowej akcji zwiazanej z sygnalem 15

sigaction(20, &P2\_TSTP, NULL);//ustawienie nowej akcji zwiazanej z sygnalem 20

sigaction(18, &P2\_CONT, NULL);//ustawienie nowej akcji zwiazanej z sygnalem 18

while (1)

{

while (1)

{

read(pfd1\_2[0], bufor\_tekstowy, sizeof(bufor\_tekstowy));//odczytanie z pipe'a wczytanego ciagu znakow od procesu1

if (bufor\_tekstowy[0] == KROPKA)

ilosc\_znakow = KROPKA;

else

ilosc\_znakow = strlen(bufor\_tekstowy) - 1; //obliczenie ilosci znakow

write(pfd2\_3[1], &ilosc\_znakow, sizeof(ilosc\_znakow));//zapis do pipe'a obliczonej ilosci znakow

if (bufor\_tekstowy[0] == KROPKA) break; // zakonczenie liczenia w przypadku kropki na STDIN lub konca pliku

}

}

}

//PROCES 3

proces3 = fork();//utworzenie procesu potomnego p3

switch (proces3)

{

case 0:

//STWORZENIE FIFO

pid\_reset = msg.pid = getpid(); //pobranie pid procesu p3

mk\_fifo\_name(msg.pid, fifo\_name, sizeof(fifo\_name)); //tworzenie nazwy fifo P3

mkfifo(fifo\_name, 0666); //tworzenie fifo o nazwie fifoPID

//SYGNALY

sigprocmask(SIG\_BLOCK, &P3\_set, NULL);//zablokowanie sygnalow ze zbioru P3\_set

sigaction(20, &P3\_KONS, NULL); //ustawienie nowej akcji zwiazanej z sygnalem 20

sigaction(18, &P3\_KONS, NULL); //ustawienie nowej akcji zwiazanej z sygnalem 18

sigaction(15, &P3\_KONS, NULL); //ustawienie nowej akcji zwiazanej z sygnalem 15

sigaction(12, &P3\_USR2, NULL); //ustawienie nowej akcji zwiazanej z sygnalem 12

while (1)

{

read(pfd1\_3\_wybor[0], &wybor, sizeof(wybor)); //odczyt z pipe'a wybranego trybu pracy programu

while (1)

{

read(pfd2\_3[0], &ilosc\_znakow, sizeof(ilosc\_znakow)); //odczyt z pipe'a ilosci znakow obliczonej przez proces2

if (ilosc\_znakow == KROPKA) break;

printf("%d\n", ilosc\_znakow); //wyswietlenie ilosci znakow w konsoli

if (wybor == 2) //jesli PLIK to synchronizacja z procesem1

write(pfd3\_1\_synchro[1], &synchro, sizeof(synchro));//zezwolenie procesowi1 na pobranie kolejnej linii z pliku

}

}

}

//SYGNALY PM

sigprocmask(SIG\_BLOCK, &PM\_set, NULL);//zablokowanie sygnalow ze zbioru PM\_set

sigaction(12, &PM\_USR2, NULL);//ustawienie nowej akcji zwiazanej z sygnalem 12

sigaction(15, &PM\_TERM, NULL);//ustawienie nowej akcji zwiazanej z sygnalem 15

while (1)

pause();

return 0;

}