

INSTYTUT TELEINFORMATYKI I AUTOMATYKI

Wydział Cybernetyki WAT

Przedmiot: SYSTEMY OPERACYJNE

SPRAWOZDANIE Z PROJEKTU ZALICZENIOWEGO

Temat: Projekt zaliczeniowy

Wykonał:

Marcin Kozłowski
WCY21IY2S1

Data wykonania ćwiczenia:

10.01.2023 r.

1. Treść zadania projektowego

Zadanie projektowe - 12 Opracować zestaw programów typu producent - konsument realizujących następujący schemat synchronicznej komunikacji międzyprocesowej: • Proces 1: czyta dane wprowadzane przez użytkownika z klawiatury lub poprzez plik tekstowy i przekazuje je w niezmięnionej formie do procesu 2 poprzez mechanizm komunikacyjny K1. • Proces 2: pobiera dane przesłane przez proces 1. Konwertuje dane otrzymane z procesu 1 do postaci heksadecymalnej, wypisuje zarówno postać źródłową jak i heksadecymalną na ekranie, a następnie dane w postaci heksadecymalnej przekazuje do procesu 3 poprzez mechanizm komunikacyjny K2. Ponadto proces ten zlicza w tle wszystkie otrzymane znaki, aby wynik wypisać na ekranie podczas zakończenia działania. • Proces 3: pobiera dane wyprodukowane przez proces 2 i wypisuje je na standardowym strumieniu diagnostycznym. Ponadto należy zaimplementować oddzielną aplikację, która umożliwi odwrotną konwersję pozwalając tym samym na weryfikację poprawności przesyłania danych przez przygotowane aplikacje.

Wszystkie trzy procesy powinny być powoływane automatycznie z jednego procesu inicjującego (jeśli wykorzystane mechanizmy komunikacji to umożliwiają). Po powołaniu procesów potomnych proces inicjujący wstrzymuje pracę. Proces inicjujący wznowia pracę w momencie kończenia pracy programu (o czym niżej), jego zadaniem jest „posprzątać” po programie przed zakończeniem działania. Ponadto należy zaimplementować mechanizm asynchronicznego przekazywania informacji pomiędzy operatorem a procesami oraz pomiędzy procesami. Należy wykorzystać do tego dostępny mechanizm sygnałów. Operator może wysłać do dowolnego procesu sygnał zakończenia działania (S1), sygnał wstrzymania działania (S2) i sygnał wznowienia działania (S3). Sygnał S2 powoduje wstrzymanie wymiany danych pomiędzy procesami. Sygnał S3 powoduje wznowienie tej wymiany. Sygnał S1 powoduje zakończenie działania oraz zwolnienie wszelkich wykorzystywanych przez procesy zasobów (zasoby zwalnia proces macierzysty). Każdy z sygnałów przekazywany jest przez operatora tylko do jednego, dowolnego procesu. O tym, do którego procesu wysłać sygnał, decyduje operator, a nie programista. Każdy z sygnałów operator może wysłać do innego procesu. Mimo, że operator kieruje sygnał do jednego procesu, to pożądane przez operatora działanie musi zostać zrealizowane przez wszystkie trzy procesy. W związku z tym, proces odbierający sygnał od operatora musi powiadomić o przyjętym żądaniu pozostałe dwa procesy. Powinien wobec tego przekazać do nich odpowiedni sygnał informując o tym jakiego działania wymaga operator. Procesy odbierające sygnał, powinny zachować się adekwatnie do otrzymanego sygnału. Wszystkie trzy procesy powinny zareagować zgodnie z żądaniem operatora. Sygnały oznaczone w opisie zadania symbolami S1 ÷ S3 należy wybrać samodzielnie spośród dostępnych w systemie.

2. Opis rozwiązania, komentarze, wnioski

A) Proces Macierzysty

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/msg.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <string.h>
#include <time.h>
#include <signal.h>
#include <errno.h>
#include <signal.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#define FIFO "my_fifo"
int notStop = 1;
//1.Funkcja do przesyłania innym sygnału stopu
void sendStopToOthers(int sig){
    printf("%d wysyła sygnał wstrzymania do pozostałych procesów: %d %d %d\n",
getpid(), getpid()+1, getpid()+2, getpid()+3);
    kill(getpid()+1, SIGUSR1);
    kill(getpid()+2, SIGUSR1);
    kill(getpid()+3, SIGUSR1);
    signal(sig, SIG_IGN);
}
//2.Funkcja do przesyłania innym sygnału wznowienia
void sendResumeToOthers(int sig){
    printf("%d wysyła sygnał wznowienia do pozostałych procesów: %d %d %d\n",
getpid(), getpid()+1, getpid()+2, getpid()+3);
    kill(getpid()+1, SIGUSR2);
    kill(getpid()+2, SIGUSR2);
    kill(getpid()+3, SIGUSR2);
    signal(sig, SIG_IGN);
}
//3.Funkcja do przesyłania innym sygnału zakończenia działania
void sendKillToOthers(int sig){
    printf("%d wysyła sygnał zabicia do pozostałych procesów: %d %d %d\n",
getpid(), getpid()+1, getpid()+2, getpid()+3);
    signal(sig, SIG_IGN);
    kill(getpid()+1, SIGPROF);
    kill(getpid()+2, SIGPROF);
    kill(getpid()+3, SIGPROF);
    kill(getpid(), SIGPROF);
}
```

```

}
//4.Funkcja kończąca program
void receiveKill(int sig){
    printf("%d odebrał sygnał zabicia: %d\n",getpid(), sig);
    signal(sig, SIG_IGN);
    int qid;
    key_t msgkey;
    msgkey=ftok(".", 'm');
    if((qid = msgget(msgkey, IPC_CREAT | 0660)) == -1){
        perror("Błąd tworzenia kolejki komunikatów\n");
        exit(1);
    }
    printf("Delete!\n");
    //Zamknięcie kolejki komunikatów
    if(msgctl(qid, IPC_RMID, 0) == -1){
        printf("Błąd usuwania kolejki komunikatów\n");
        exit(1);
    }
    printf("Kolejka zamknięta\n");
    notStop = 0;
}

int main(int args, char* argv[])
{
    //5.Stworzenie kolejki
    umask(0);
    mkfifo(FIFO, 0666);
    printf("Parent PID: %d \n", getpid());
    int p1, p2, p3;
    int true = 1;
    //6.Powołanie procesów potomnych
    if((p1=fork())==0){
        execlp("./p1","./p1", NULL);
    }
    else if((p2=fork())==0){
        execlp("./p2","./p2", NULL);
    }
    else if((p3=fork())==0){
        execlp("./p3","./p3", NULL);
    }
    else{
        //7.Nasłuchiwanie odpowiednicy sygnałów
        printf("Children PID: %d %d %d\n", p1, p2, p3);
        while(notStop){
            signal(SIGINT, sendStopToOthers);
            signal(SIGQUIT, sendResumeToOthers);
            signal(SIGILL, sendKillToOthers);
            signal(SIGPROF, receiveKill);

```

```
    }  
}  
printf("Koniec programu\n");  
return 0;  
}
```

Działanie:

1. Proces wysyła pozostałym procesom sygnał wstrzymania (jest to sygnał SIGUSR1) używając funkcji kill z PID'em procesu oraz odpowiednim sygnałem. Domyślna akcja po otrzymaniu sygnału zostaje zignorowana.
2. Proces wysyła pozostałym procesom sygnał wznowienia (jest to sygnał SIGUSR2) używając funkcji kill z PID'em procesu oraz odpowiednim sygnałem. Domyślna akcja po otrzymaniu sygnału zostaje zignorowana.
3. Proces wysyła pozostałym procesom sygnał zakończenia działania (jest to sygnał SIGPROF) używając funkcji kill z PID'em procesu oraz odpowiednim sygnałem. Domyślna akcja po otrzymaniu sygnału zostaje zignorowana. Proces wysyła również sam sobie sygnał o zakończeniu działania.
4. Proces odbiera sygnał zakończenia działania. Zamyka kolejkę komunikatów i kończy swoje działanie.
5. Tworzona jest kolejka do dalszej komunikacji między procesami potomnymi p2 i p3.
6. Powoływane są procesy potomne.
7. Program wstrzymuje swoje działanie i nasłuchuje odpowiednich sygnałów które przekazuje handlerom.

B) Proces potomny p1

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <errno.h>
int running = 1;
int killFor = 0;
//1.Funkcja do przesyłania innym sygnału stopu
void sendStopToOthers(int sig){
    printf("%d wysyła sygnał wstrzymania do pozostałych procesów: %d %d\n",
getpid(), getpid()+1, getpid()+2);
    kill(getpid()+1, SIGUSR1);
    kill(getpid()+2, SIGUSR1);
    signal(sig, SIG_IGN);
    kill(getpid(), SIGUSR1);
}
//2.Funkcja wykonująca wstrzymanie przesyłania danych
void receiveStop(int sig){
    printf("%d odebrał sygnał wstrzymania: %d\n",getpid(), sig);
    signal(sig, SIG_IGN);
}
//3.Funkcja do przesyłania innym sygnału wznowienia
void sendResumeToOthers(int sig){
    printf("%d wysyła sygnał wznowienia do pozostałych procesów: %d %d\n",
getpid(), getpid()+1, getpid()+2);
    kill(getpid()+1, SIGUSR2);
    kill(getpid()+2, SIGUSR2);
    signal(sig, SIG_IGN);
    kill(getpid(), SIGUSR2);
}
//4.Funkcja wykonująca wznowienie przesyłania danych
void receiveResume(int sig){
    printf("%d odebrał sygnał wznowienia: %d\n",getpid(), sig);
    signal(sig, SIG_IGN);
}
//5.Funkcja do przesyłania innym sygnału zakończenia działania
void sendKillToOthers(int sig){
    printf("%d wysyła sygnał zabicia do pozostałych procesów: %d %d\n",
getpid(), getpid()+1, getpid()+2);
    kill(getpid()+1, SIGPROF);
    kill(getpid()+2, SIGPROF);
    kill(getpid()-1, SIGPROF);
    kill(getpid(), SIGPROF);
}
```

```

//6.Funkcja kończąca program
void receiveKill(int sig){
    printf("%d odebrał sygnał zabicia: %d\n",getpid(), sig);
    signal(sig, SIG_IGN);
    running = 0;
    kill(getpid(), SIGKILL);
}

//7. Struktura służąca do wymiany wysyłania i odbierania informacji
//w kolejce komunikatów
struct mymsgbuf
{
    long mtype;
    char i[1];
}queue;

struct mymsgbuf readbuffer;

int main()
{
    printf("P1 running!\n");
    FILE *fp;
    int wybor;
    int qid;
    key_t msgkey;
    char c;
    //8. Utworzenie kolejki komunikatów
    msgkey=ftok(".", 'm');
    if((qid=msgget(msgkey,IPC_CREAT|0660))==-1)
    {
        perror("Bład otwierania kolejki komunikatow");
        exit(1);
    }
    printf("ODBIERANIE DANYCH ZNAKOWYCH\n");
    printf("1. Z klawiatury\n");
    printf("2. Z pliku\n");
    printf("Wybierz opcję: ");
    scanf("%d",&(wybor));
    switch(wybor)
    {
        case 1:
            sleep(6);
            fp=fopen("tekst.txt","w+");
            if(fp==NULL)
            {
                perror("Bład otwierania pliku1");
                exit(1);
            }
            printf("Wpisz znaki z klawiatury: \n");
            scanf("%c",&c);

```

```

        //9. Zmienna 'running' odpowiedzialna za dzialanie wysylania
danych do kolejki do momentu
        //wyslania sygnalu zakonczenia dzialania
        while(running&& c!=' ')
        {
            if(c!='\n')
            {

                fputc(c,fp);
                queue.i[0]=c;
                queue.mtype=1;
                //10. Wyslanie wiadomosci za pomoca kolejki
                if((msgsnd(qid,&queue,sizeof(struct mymsgbuf)-
sizeof(long),0))== -1 )
                {
                    perror("Blad wysylania komunikatu");
                    exit(1);
                }
                //11. Obsluga odbieranych sygnalow
                signal(SIGINT, sendStopToOthers);
                signal(SIGUSR1, receiveStop);
                signal(SIGQUIT, sendResumeToOthers);
                signal(SIGUSR2, receiveResume);
                signal(SIGILL, sendKillToOthers);
                signal(SIGPROF, receiveKill);
            }
            scanf("%c",&c);
        }
        queue.i[0]=' ';
        msgsnd(qid,&queue,sizeof(struct mymsgbuf)-sizeof(long),0);
        fclose(fp);
        break;
    case 2:
        //12. Czytanie z pliku
        fp=fopen("tekst.txt","r");
        if(fp==NULL)
        {
            perror("Blad otwierania pliku1");
            exit(1);
        }
        while((queue.i[0]=fgetc(fp))!=EOF)
        {
            queue.mtype=1;
            if((msgsnd(qid,&queue,sizeof(struct mymsgbuf)-
sizeof(long),0))== -1)
            {
                perror("Blad wysylania komunikatu");
                exit(1);
            }
        }
    }
}

```



```

        //11. Obsługa odbieranych sygnałów
        signal(SIGINT, sendStopToOthers);
        signal(SIGUSR1, receiveStop);
        signal(SIGQUIT, sendResumeToOthers);
        signal(SIGUSR2, receiveResume);
        signal(SIGILL, sendKillToOthers);
        signal(SIGPROF, receiveKill);
    }
    fclose(fp);
    break;
default:
    printf("Bledny wybor!\n");
    return 0;
}
}

```

Działanie:

Punkty 1,3,5,6 zbliżone do handler'ów (funkcji dla sygnałów) w procesie macierzystym.

2. Pusta funkcja z uwagi, że działanie polegające na wstrzymaniu przesyłania danych następuje w dalszych procesach.

4. Pusta funkcja z uwagi, że działanie polegające na wznowieniu przesyłania danych następuje w dalszych procesach.

7. Struktura potrzebna do kolejki komunikatów. Zakodowana jest w niej wiadomość przesyłana za pomocą kolejki jak i do kogo ta wiadomość ma trafić.

8. Stworzenie/Otwarcie kolejki komunikatów

9. Jeśli program otrzyma sygnał zakończenia zmienna running zmienia wartość i program zostanie zakończony.

10. Wysłanie wiadomości do kolejki.

11. Nasłuchiwanie sygnałów i przekazywanie ich wykonania do określonych funkcji.

12. Czytanie z pliku po zaznaczeniu odpowiedniej opcji.

C) Proces potomny p2

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/msg.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <string.h>
#include <time.h>
#include <signal.h>
#include <errno.h>
#include <signal.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>

#define FIFO "my_fifo"
int waitFor = 1;
int running = 1;
int fifo;

//1.Funkcja do przesyłania innym sygnału stopu
void sendStopToOthers(int sig){
    printf("%d wysyła sygnał wstrzymania do pozostałych procesów: %d %d\n",
getpid(), getpid()-1, getpid()+1);
    kill(getpid()-1, SIGUSR1);
    kill(getpid()+1, SIGUSR1);
    signal(sig, SIG_IGN);
    kill(getpid(), SIGUSR1);
}

//2.Funkcja wykonująca wstrzymanie przesyłania danych
void receiveStop(int sig){
    printf("%d odebrał sygnał wstrzymania: %d\n",getpid(), sig);
    signal(sig, SIG_IGN);
    waitFor = 0;
}

//3.Funkcja do przesyłania innym sygnału wznowienia
void sendResumeToOthers(int sig){
    printf("%d wysyła sygnał wznowienia do pozostałych procesów: %d %d\n",
getpid(), getpid()-1, getpid()+1);
    kill(getpid()-1, SIGUSR2);
    kill(getpid()+1, SIGUSR2);
    signal(sig, SIG_IGN);
    kill(getpid(), SIGUSR2);
}

//4.Funkcja wykonująca wznowienie przesyłania danych
```

```

void receiveResume(int sig){
    printf("%d odebrał sygnał wznowienia: %d\n",getpid(), sig);
    signal(sig, SIG_IGN);
    waitFor = 1;
}

//5.Funkcja do przesyłania innym sygnału zakończenia działania
void sendKillToOthers(int sig){
    printf("%d wysyła sygnał zabicia do pozostałych procesów: %d %d\n",
getpid(), getpid()-1, getpid()+1);
    kill(getpid()-1, SIGPROF);
    kill(getpid()+1, SIGPROF);
    kill(getpid()-2, SIGPROF);
    kill(getpid(), SIGPROF);
}

//6.Funkcja kończąca program
void receiveKill(int sig){
    printf("%d odebrał sygnał zabicia: %d\n",getpid(), sig);
    signal(sig, SIG_IGN);
    running = 0;
    //kill(getpid(), SIGKILL);
    printf("Kolejka zamknięta\n");
}

//7. Struktura służąca do wymiany wysyłania i odbierania informacji
//w kolejce komunikatów
static struct sembuf buf;

struct mymsgbuf
{
    long mtype;
    char i[1];
}queue;

struct mymsgbuf readbuffer;

int main()
{
    printf("P2 running!\n");
    sleep(3);
    FILE *fp2;
    int s=0;
    int qid;
    char c;
    int wybor2;
    key_t msgkey;
    msgkey=ftok(".", 'm');

    char in[5];
    umask(0);
    //8. Otwarcie fifo

```

```

fifo = open(FIFO, O_WRONLY);

fp2=fopen("hex.txt","w+");
if(fp2==NULL)
{
    perror("Bład otwierania pliku2");
    exit(1);
}
if((qid=msgget(msgkey,IPC_CREAT|0660))==-1)
{
    perror("Bład otwierania kolejki komunikatow");
    exit(1);
}
readbuffer.mtype=1;

printf("Potwierdz swój wybór: ");
scanf("%d",&(wybor2));
switch(wybor2)
{
    case 1:
        //9. Wyświetlanie odczytanych danych z kolejki i przesłanie ich do
fifo
        msgrcv(qid,&readbuffer,sizeof(struct mymsgbuf)-
sizeof(long),readbuffer.mtype,0);
        sprintf(in, "0x%x\n", *readbuffer.i);
        write(fifo, in, 5);
        while((*readbuffer.i)!=' ' && running)
        {
            //10. Jeśli zmienna waitFor zostanie zmieniona w wyniku
sygnału wstrzymania proces nie będzie czytał z kolejki
            if(waitFor){
                printf("Proces %d - odebrałem wartosc: %c      hex:
0x%x\n",getpid(),*readbuffer.i,*readbuffer.i);
                fprintf(fp2,"0x%x\n",*readbuffer.i);
                msgrcv(qid,&readbuffer,sizeof(struct mymsgbuf)-
sizeof(long),readbuffer.mtype,0);

                sprintf(in, "0x%x\n", *readbuffer.i);
                write(fifo, in, 5);

                s++;
            }

            signal(SIGINT, sendStopToOthers);
            signal(SIGUSR1, receiveStop);
            signal(SIGQUIT, sendResumeToOthers);
            signal(SIGUSR2, receiveResume);
            signal(SIGILL, sendKillToOthers);
            signal(SIGPROF, receiveKill);

```

```

    }
    break;
case 2:
    while(msgrcv(qid,&readbuffer,sizeof(struct mymsgbuf)-
sizeof(long),readbuffer.mtype,IPC_NOWAIT)!=-1)
    {
        if(waitFor){
            printf("Proces %d - odebralem wartosc: %c      hex:
0x%x\n",getpid(),*readbuffer.i,*readbuffer.i);
            fprintf(fp2,"0x%x\n",*readbuffer.i);

            sprintf(in, "0x%x\n", *readbuffer.i);
            write(fifo, in, 5);

            s++;
        }
        //11. Obsługa odbieranych sygnałów
        signal(SIGINT, sendStopToOthers);
        signal(SIGUSR1, receiveStop);
        signal(SIGQUIT, sendResumeToOthers);
        signal(SIGUSR2, receiveResume);
        signal(SIGILL, sendKillToOthers);
        signal(SIGPROF, receiveKill);
    }
    break;
default:
    printf("Bledny wybor!\n");
    return 0;
}

rewind(fp2);

sleep(1);
printf("Pobrano %d znakow.\n",s);
fclose(fp2);
kill(getpid()-2, SIGILL);
return 0;
}

```

Działanie:

Punkty 1-7 zbliżone do punktów z procesem p1.

8. Otwarcie kolejki fifo.

9. Odebranie danych z jednego mechanizmu komunikacyjnego i przekazanie go do drugiego

10. Zmienna zmieniająca wartość przy odebraniu określonego sygnału i tym samym wstrzymująca odczytanie (wymianę) danych.

11. Obsługa sygnałów.

D) Proces potomny p3

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/msg.h>
#include <stdlib.h>
#include <sys/shm.h>
#include <string.h>
#include <time.h>
#include <signal.h>
#include <errno.h>
#include <signal.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>

#define FIFO "my_fifo"
int waitFor = 1;
int running = 1;
int fifo;

//1.Funkcja do przesyłania innym sygnału stopu
void sendStopToOthers(int sig){
    printf("%d wysyła sygnał wstrzymania do pozostałych procesów: %d %d\n",
getpid(), getpid()-1, getpid()-2);
    kill(getpid()-1, SIGUSR1);
    kill(getpid()-2, SIGUSR1);
    signal(sig, SIG_IGN);
    kill(getpid(), SIGUSR1);
}

//2.Funkcja wykonująca wstrzymanie przesyłania danych
void receiveStop(int sig){
    printf("%d odebrał sygnał wstrzymania: %d\n",getpid(), sig);
    signal(sig, SIG_IGN);
    waitFor = 0;
}

//3.Funkcja do przesyłania innym sygnału wznowienia
void sendResumeToOthers(int sig){
    printf("%d wysyła sygnał wznowienia do pozostałych procesów: %d %d\n",
getpid(), getpid()-1, getpid()-2);
    kill(getpid()-1, SIGUSR2);
    kill(getpid()-2, SIGUSR2);
    signal(sig, SIG_IGN);
    kill(getpid(), SIGUSR2);
}
```

```

//4.Funkcja wykonująca wznowienie przesyłania danych
void receiveResume(int sig){
    printf("%d odebrał sygnał wznowienia: %d\n",getpid(), sig);
    signal(sig, SIG_IGN);
    waitFor = 1;
}

//5.Funkcja do przesyłania innym sygnału zakończenia działania
void sendKillToOthers(int sig){
    printf("%d wysłał sygnał zabicia do pozostałych procesów: %d %d\n",
getpid(), getpid()-1, getpid()-2);
    kill(getpid()-1, SIGPROF);
    kill(getpid()-2, SIGPROF);
    kill(getpid()-3, SIGPROF);
    kill(getpid(), SIGPROF);
}

//6.Funkcja kończąca program
void receiveKill(int sig){
    printf("%d odebrał sygnał zabicia: %d\n",getpid(), sig);
    signal(sig, SIG_IGN);
    running = 0;
    close(fifo);
    kill(getpid(), SIGKILL);
}

int main()
{
    printf("P3 running!\n");
    sleep(2);
    umask(0);
    mkfifo(FIFO, 0666);
    char out[5];
    //7. Otwarcie kolejki fifo
    fifo = open(FIFO, O_RDONLY);
    if(read(fifo, out, 5)>0 && *out != '\n'){
        printf("Proces %d - odebrałem wartosc: %s\n",getpid(), out);
    }
    //8. Zmienna running odpowiedzialna za działanie programu i zmienna
    waitFor odpowiadająca za czytanie z kolejki fifo
    while(running){
        if(waitFor){
            if(read(fifo, out, 5)>0 && *out != '\n'){
                printf("Proces %d - odebrałem wartosc: %s\n",getpid(), out);
            }
        }
        //9. Odbieranie sygnałów i przekazywanie ich do określonych funkcji
        signal(SIGINT, sendStopToOthers);
        signal(SIGUSR1, receiveStop);
        signal(SIGQUIT, sendResumeToOthers);
        signal(SIGUSR2, receiveResume);
    }
}

```



```
    signal(SIGILL, sendKillToOthers);  
    signal(SIGPROF, receiveKill);  
}  
return 0;  
}
```

Działanie:

Punkty 1-7 zbliżone do punktów z procesem p1.

8. Otwarcie kolejki fifo.

9. Jeśli zostaną odebrane właściwe sygnały to zmienna running lub waitFor zmieniają wartość. Powoduje to zakończenie działania programu lub wstrzymanie wymiany danych między procesami.

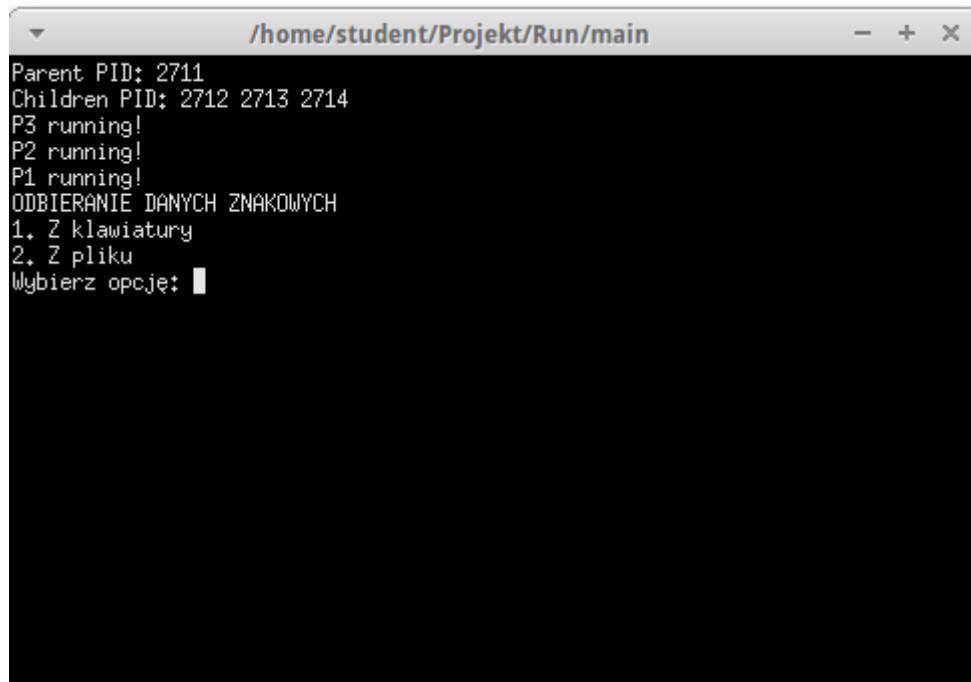
E) Oddzielny program sprawdzający

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    FILE *fp;
    FILE *fp2;
    fp=fopen("./tekst.txt","r");
    fp2=fopen("./hex.txt","r");
    if(fp==NULL)
    {
        perror("Bład otwierania pliku1");
        exit(1);
    }
    if(fp2==NULL)
    {
        perror("Bład otwierania pliku2");
        exit(1);
    }
    int first;
    int second;
    char str2;
    char line[256];
    char c;
    while ((c=fgetc(fp))!=EOF && fgets(line,256, fp2))
    {
        printf("%c = ", c);
        printf("%s", line);
        if(strtol(line, 0, 0) != c){
            printf("Dane nie sa jednakowe!\n");
            return 1;
        }
    }

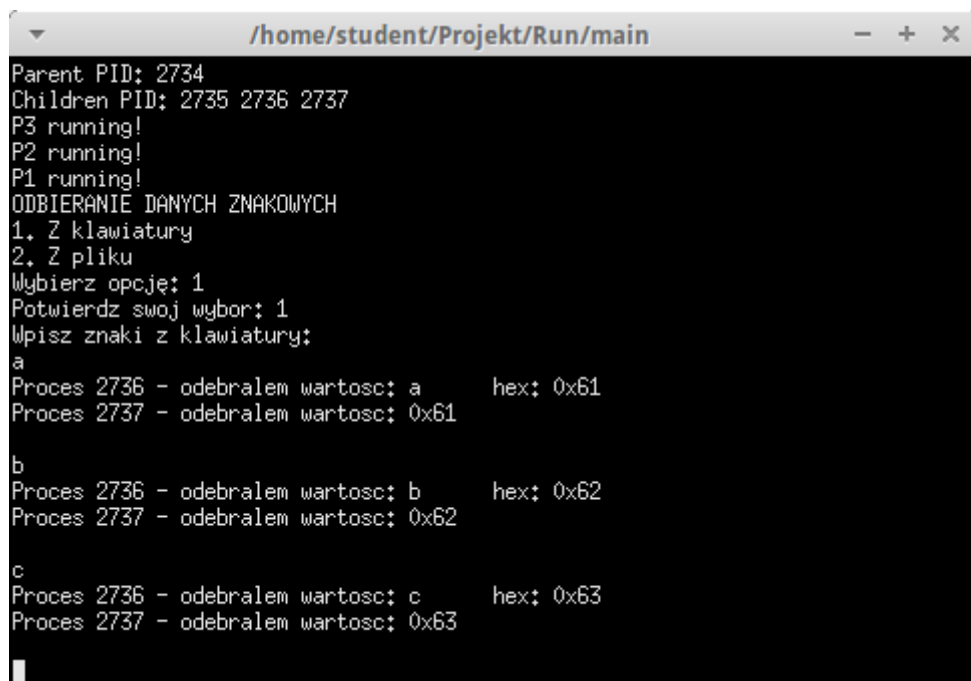
    printf("Dane sa jednakowe!\n");
    fclose(fp2);
    fclose(fp);
    return 0;
}
```

3. Wyniki uruchomienia programu



```
/home/student/Projekt/Run/main
Parent PID: 2711
Children PID: 2712 2713 2714
P3 running!
P2 running!
P1 running!
ODBIERANIE DANYCH ZNAKOWYCH
1. Z klawiatury
2. Z pliku
Wybierz opcję: █
```

Obraz 1. Wybór opcji

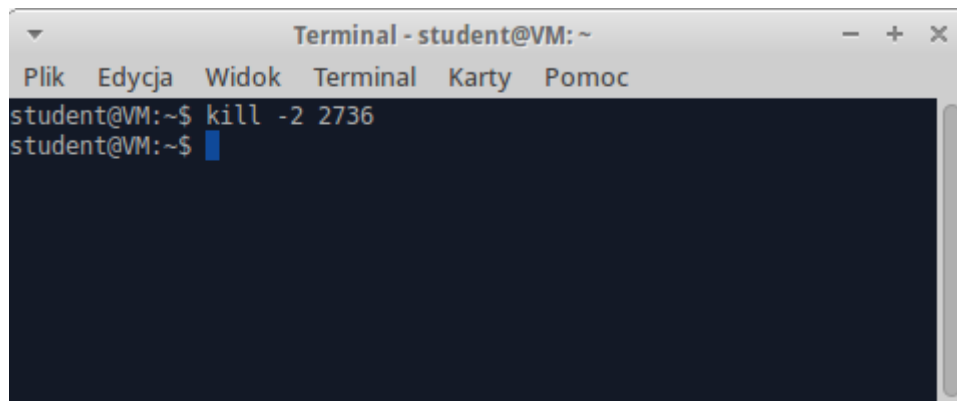


```
/home/student/Projekt/Run/main
Parent PID: 2734
Children PID: 2735 2736 2737
P3 running!
P2 running!
P1 running!
ODBIERANIE DANYCH ZNAKOWYCH
1. Z klawiatury
2. Z pliku
Wybierz opcję: 1
Potwierdz swój wybór: 1
Wpisz znaki z klawiatury:
a
Proces 2736 - odebrałem wartość: a      hex: 0x61
Proces 2737 - odebrałem wartość: 0x61

b
Proces 2736 - odebrałem wartość: b      hex: 0x62
Proces 2737 - odebrałem wartość: 0x62

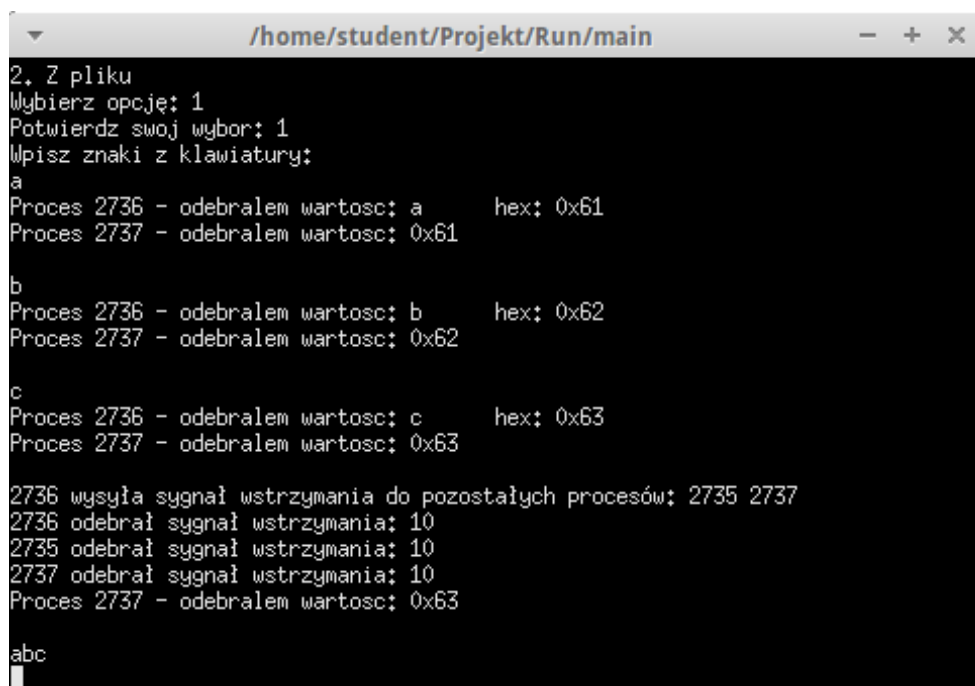
c
Proces 2736 - odebrałem wartość: c      hex: 0x63
Proces 2737 - odebrałem wartość: 0x63
█
```

Obraz 2. Zamiana znaków z klawiatury



```
Terminal - student@VM: ~
Plik  Edycja  Widok  Terminal  Karty  Pomoc
student@VM:~$ kill -2 2736
student@VM:~$
```

Obraz 3. Sygnał wstrzymania do jednego z procesów.



```
/home/student/Projekt/Run/main
2. Z pliku
Wybierz opcję: 1
Potwierdz swój wybór: 1
Wpisz znaki z klawiatury:
a
Proces 2736 - odebrałem wartosc: a      hex: 0x61
Proces 2737 - odebrałem wartosc: 0x61

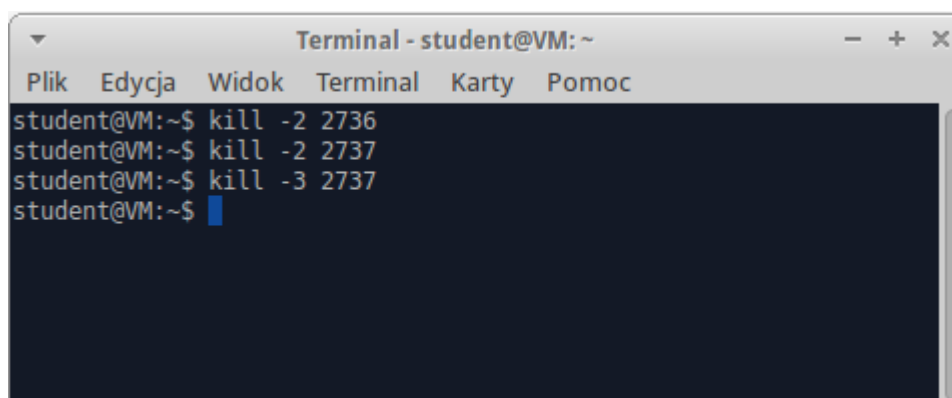
b
Proces 2736 - odebrałem wartosc: b      hex: 0x62
Proces 2737 - odebrałem wartosc: 0x62

c
Proces 2736 - odebrałem wartosc: c      hex: 0x63
Proces 2737 - odebrałem wartosc: 0x63

2736 wysyła sygnał wstrzymania do pozostałych procesów: 2735 2737
2736 odebrał sygnał wstrzymania: 10
2735 odebrał sygnał wstrzymania: 10
2737 odebrał sygnał wstrzymania: 10
Proces 2737 - odebrałem wartosc: 0x63

abc
```

Obraz 4. Wstrzymanie przesyłania danych



```
Terminal - student@VM: ~
Plik  Edycja  Widok  Terminal  Karty  Pomoc
student@VM:~$ kill -2 2736
student@VM:~$ kill -2 2737
student@VM:~$ kill -3 2737
student@VM:~$
```

Obraz 5. Sygnał wznowienia do jednego z procesów.

```
/home/student/Projekt/Run/main
2735 odebrał sygnał wstrzymania: 10
2737 odebrał sygnał wstrzymania: 10
Proces 2737 - odebrałem wartosc: 0x63

abc
2737 wysyła sygnał wstrzymania do pozostałych procesów: 2736 2735
2735 odebrał sygnał wstrzymania: 10
2737 odebrał sygnał wstrzymania: 10
2736 odebrał sygnał wstrzymania: 10
2737 wysyła sygnał wznowienia do pozostałych procesów: 2736 2735
2735 odebrał sygnał wznowienia: 12
2737 odebrał sygnał wznowienia: 12
2736 odebrał sygnał wznowienia: 12
Proces 2736 - odebrałem wartosc: c      hex: 0x63
Proces 2736 - odebrałem wartosc: a      hex: 0x61
Proces 2737 - odebrałem wartosc: 0x61

Proces 2736 - odebrałem wartosc: b      hex: 0x62
Proces 2737 - odebrałem wartosc: 0x62

Proces 2736 - odebrałem wartosc: c      hex: 0x63
Proces 2737 - odebrałem wartosc: 0x63
```

Obraz 6. Wznowienie przesyłania danych.

```
Terminal - student@VM: ~
Plik  Edycja  Widok  Terminal  Karty  Pomoc
student@VM:~$ kill -2 2736
student@VM:~$ kill -2 2737
student@VM:~$ kill -3 2737
student@VM:~$ kill -4 2735
student@VM:~$
```

Obraz 7. Sygnał zakończenia działania do jednego z procesów.

```
/home/student/Projekt/Run/main
Proces 2736 - odebralem wartosc: c      hex: 0x63
Proces 2736 - odebralem wartosc: a      hex: 0x61
Proces 2737 - odebralem wartosc: 0x61

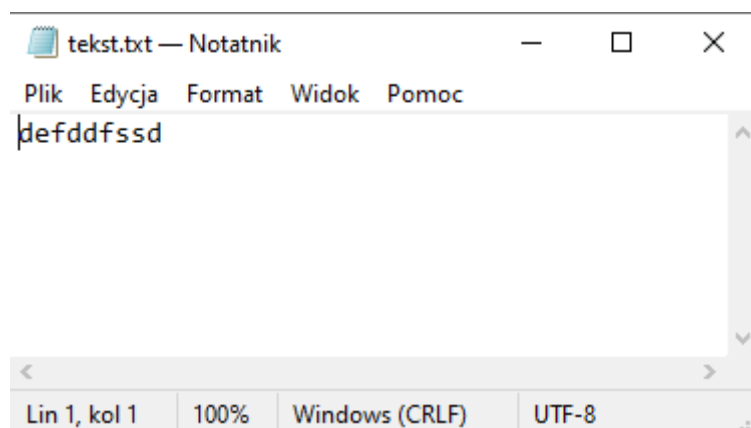
Proces 2736 - odebralem wartosc: b      hex: 0x62
Proces 2737 - odebralem wartosc: 0x62

Proces 2736 - odebralem wartosc: c      hex: 0x63
Proces 2737 - odebralem wartosc: 0x63

2735 wysyla sygnał zabicia do pozostałych procesów: 2736 2737
2735 odebrał sygnał zabicia: 27
2736 odebrał sygnał zabicia: 27
2737 odebrał sygnał zabicia: 27
Kolejka zamknięta
2734 odebrał sygnał zabicia: 27
Delete!
Kolejka zamknięta
Koniec programu

Process returned 0 (0x0)   execution time : 340.080 s
Press ENTER to continue.
Pobrano 7 znakow.
```

Obraz 8. Zakończenie działania programu.



Obraz 9. Początkowe dane dla czytania z pliku.

```

/home/student/Projekt/Run/main
Parent PID: 3010
Children PID: 3011 3012 3013
P3 running!
P2 running!
P1 running!
ODBIERANIE DANYCH ZNAKOWYCH
1. Z klawiatury
2. Z pliku
Wybierz opcję: 2
Potwierdź swój wybór: 2
Proces 3012 - odebrałem wartosc: d      hex: 0x64
Proces 3013 - odebrałem wartosc: 0x64

Proces 3012 - odebrałem wartosc: e      hex: 0x65
Proces 3012 - odebrałem wartosc: f      hex: 0x66
Proces 3013 - odebrałem wartosc: 0x65

Proces 3012 - odebrałem wartosc: d      hex: 0x64
Proces 3013 - odebrałem wartosc: 0x66

Proces 3012 - odebrałem wartosc: d      hex: 0x64
Proces 3013 - odebrałem wartosc: 0x64

Proces 3012 - odebrałem wartosc: f      hex: 0x66
Proces 3013 - odebrałem wartosc: 0x64

Proces 3012 - odebrałem wartosc: s      hex: 0x73
Proces 3013 - odebrałem wartosc: 0x66

Proces 3012 - odebrałem wartosc: s      hex: 0x73
Proces 3013 - odebrałem wartosc: 0x73

Proces 3012 - odebrałem wartosc: d      hex: 0x64
Proces 3013 - odebrałem wartosc: 0x73

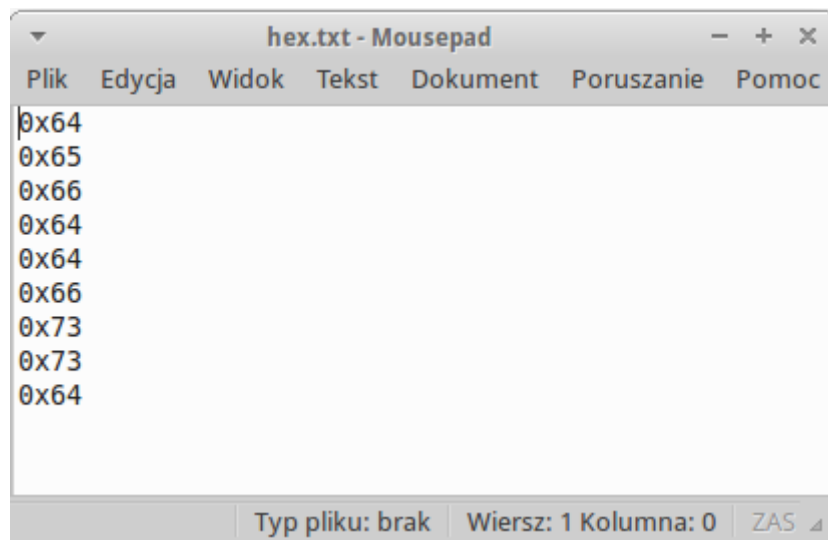
Proces 3013 - odebrałem wartosc: 0x64

Pobrano 9 znakow.
3010 wysyła sygnał zabicia do pozostałych procesów: 3011 3012 3013
3010 odebrał sygnał zabicia: 27
Delete!
Kolejka zamknięta
Koniec programu
3013 odebrał sygnał zabicia: 27

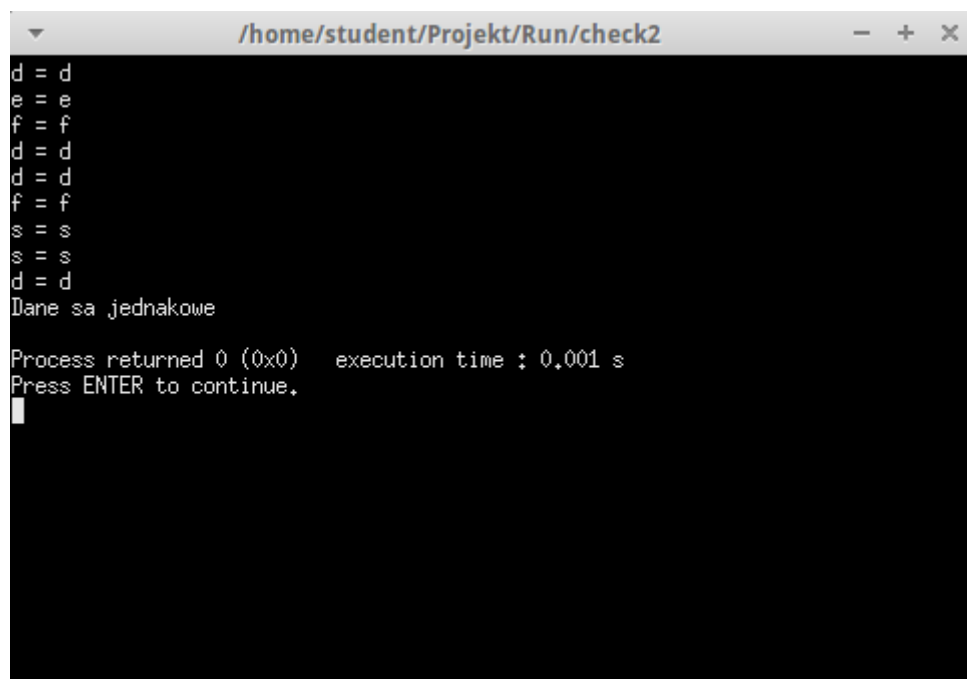
Process returned 0 (0x0)   execution time : 4.553 s
Press ENTER to continue.

```

Obraz 10. Rezultat czytania z pliku.



Obraz 11. Rezultat czytania z pliku i zapis do innego pliku.



Obraz 12. Rezultat działania programu sprawdzającego.