Imię: Sylwia
Nazwisko:Pytko
Grupa:I4Z6S1

Projekt.
Numer przydzielonego zadania: v4, nr 17 (file i pipe)

Data oddania projektu: 18.01.16

Data wykonania sprawozdania: 16.01.16

Opis zadania

Opracować zestaw programów typu producent - konsument realizujących następujący schemat synchronicznej komunikacji międzyprocesowej:

- ***Proces 1: czyta dane ze standardowego strumienia wejściowego i przekazuje je w niezmienionej formie do procesu 2 poprzez mechanizm komunikacyjny K1.***
- ***Proces 2: pobiera dane przesłane przez proces 1. Oblicza liczbę liter „o” w każdej linii i wyznaczoną liczbę przekazuje do procesu 3 poprzez mechanizm komunikacyjny K2.***
- ***Proces 3: pobiera dane wyprodukowane przez proces 2 i wypisuje je na standardowym strumieniu diagnostycznym. Jednostki danych powinny zostać wyprowadzone po maksymalnie 15 w pojedynczym wierszu i oddzielone spacjami.***

Wszystkie trzy procesy powinny być powoływane automatycznie z jednego procesu inicjującego. Po powołaniu procesów potomnych proces inicjujący wstrzymuje pracę. Proces inicjujący wznawia 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. 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 synchronicznej 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 (np. SIGUSR1, SIGUSR2, SIGINT, SIGCONT, ...).

Program ma umożliwiać uruchmienie:

- ***w trybie interaktywnym – operator wprowadza dane z klawiatury,***
- ***w trybie odczytu danych z określonego pliku,***
- ***w trybie odczytu danych z pliku /dev/urandom.***

Mechanizmy komunikacji: K1 – file, K2 – pipe

Kod źródłowy

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <fcntl.h>
#include <sys/stat.h>

	int pid[4]={0};
	int dzialanie=1;

void zakoncz(int sygnal)
{
	printf("zakoncz %d\n", getpid());
	fflush(stdout);
	kill(0,sygnal);

		if(getpid()==pid[0])
	{
		remove("plik.txt");
	}
	exit(0);
}
void wstrzymaj(int sygnal)
{
	if(dzialanie==1)
	{
	printf("wstrzymaj %d\n", getpid());
	fflush(stdout);

	dzialanie=0;

	kill(0, sygnal);
	}

}
void wznow(int sygnal)
{
	if(dzialanie==0)
	{
	printf("wznow %d\n", getpid());
	fflush(stdout);

	dzialanie=1;

	kill(0, sygnal);
	}

}
FILE *pobierzplik()
{
	char nazwapliku[20];
	FILE *plikwej;
	printf("\npodaj nazwę pliku:");
	scanf("%s", &nazwapliku);

	plikwej=fopen(nazwapliku, "r");
	if(plikwej==NULL)
	{
		printf("Blad otwarcia pliku wejsciowego wpisz inny plik\n");
```

```
                plikwej=pobierzplik();
        }
        return plikwej;
}
int main()
{
        char bufor[200];
        char bufor1[200];
        int czym=1;

        FILE *plik1;
        FILE *plikwej;

        int o=0;
        char o1[10];
        int pipe1[2];
        pipe(pipe1);
        int do15=0;

        pid[0]=getpid();



        if(fork()==0) //proces 1
        {
                pid[1]=getpid();
                printf("\nproces 1 pid %d\n", pid[1]);
                fflush(stdout);

                        signal(4,zakoncz);
                        signal(5, wstrzymaj);
                        signal(6, wznow);

                sleep(1);
                printf("Obsluga sygnalow:\n");
                printf("S1 - sygnal zakonczenia: 4 SIGILL\n");
                printf("S2 - sygnal wstrzymania: 5 SIGTRAP\n");
                printf("S3 - sygnal zakonczenia: 6 SIGABORT\n\n");

                printf("jakim sposobem chcesz podać dane?\n");
                printf("1. Ręcznie z konsoli\n");
                printf("2. Z pliku \n");
                printf("3. /dev/urandom \n");
                fflush(stdout);

                scanf("%d", &czym);

                getchar();
                fflush(stdin);

                                if(czym==2)
                                {
                                        plikwej=pobierzplik();
                                }
                                if(czym==3)
                                {
                                        plikwej=fopen("/dev/urandom", "r");
                                                if(plikwej==NULL)
                                                {
                                                        perror("Blad otwarcia /dev/urandom\n");
                                                }
                                }
```

```
		while(1)
		{
		while(dzialanie)
		{


			if(czym==2||czym==3)
			{
				int i;
				for(i=0;i<200; i++)
					bufor[i]=0;

				if(fgets(bufor, 200, plikwej)==0)
				{
					printf("koniec pliku, nic wiecej do pracy");
					fflush(stdout);
					while(1)
					{
					}
				}

			}
			else
			{
					printf("\nwpisz tekst:");
					fflush(stdout);
					fgets(bufor, 200, stdin);
			}


		plik1=fopen("/home/student/plik.txt", "w+");
		if(plik1==NULL)
		{
			perror("Blad otwarcia plik do zapisu\n");
		}

		int i=0;
		while(i<200)
		{

		fprintf(plik1,"%c", bufor[i]);
		if(bufor[i]=='\n')
		{
			break;
		}
		i++;
		}

		fclose(plik1);

	while(access("/home/student/plik.txt",F_OK)==0)
	{
	}



		}
		}
}
else
{
```

```
	if(fork()==0)// proces 2
	{
		pid[2]=getpid();
		printf("proces 2 pid %d\n", pid[2]);
		fflush(stdout);

				signal(4,zakoncz);
				signal(5, wstrzymaj);
				signal(6, wznow);

		close(pipe1[0]);

		while(1)
		{
		while(dzialanie)
		{

			plik1=fopen("/home/student/plik.txt", "r");

			int i;
			for(i=0;i<200; i++)
			{
				bufor1[i]=0;
			}

			if(plik1!=NULL)
			{
			fgets(bufor1, 200, plik1);

			int n=0;

			while(bufor1[n]!='\n')
			{
				if(n<200)
				{
					if(bufor1[n]=='o')
					{
						o++;
					}
					n++;
				}
				else
				{
					break;
				}
			}
			if(bufor1[n]=='\n')
			{
			sprintf(o1,"%i",o);// int na chara(stringa)


			write(pipe1[1], o1, 10);
			o=0;
			}
			fclose(plik1);
			remove("plik.txt");
			}
		}
		}

	}
else
```

```
		{
			if(fork()==0)// proces 3
			{
				pid[3]=getpid();
				printf("proces 3 pid %d\n", pid[3]);
				fflush(stdout);

					signal(4,zakoncz);
					signal(5, wstrzymaj);
					signal(6, wznow);

				close(pipe1[1]);

				while(1)
				{
				while(dzialanie)
				{


					read(pipe1[0], o1, 10);

					if(do15<14)
					{

					fprintf(stderr,"%s ", o1);
					do15++;
					}
					else
					{
						fprintf(stderr,"%s \n", o1);
						do15=0;
					}

				}
				}

			}
			else
			{

				signal(4,zakoncz);
				signal(5, SIG_IGN);
				signal(6, SIG_IGN);
				printf("proces maciezysty pid %d\n\n", pid[0]);

				while(1)
				{}
			}
		}

	}
	return 0;
}
```

Opis rozwiązania

Proces macierzysty

```
	int pid[4]={0};
	int dzialanie=1;

int main()
{
	char bufor[200];
	char bufor1[200];
	int czym=1;

	FILE *plik1;
	FILE *plikwej;

	int o=0;
	char o1[10];
	int pipe1[2];
	pipe(pipe1);
	int do15=0;
```

Zadeklarowanie **zmiennych globalnych**:
Pid – tablica przechowująca pidy procesow
Dzialanie – zmienna informująca czy proces nie jest wstrzymany

Zadeklarowanie w **procesie macierzystym** potrzebnych zmiennych.
Bufor – bufor potrzebny do pobierania wartości wejściowych.
Bufor1 – bufor potrzebny do pobrania z pliku który był narzędziem komunikacyjnym.
Plik1 – wskaźnik na plik bedacy narzedziem komunikacyjnym
Plikwej – wskaźnik na plik wejściowy
O – liczba znaków „o”
O1 – string przechowujący informacje o liczbie znaków „o”, potrzebny do pipe
Pipe1[2] – tablica przechowująca uchwyty na pliki odczytu i zapisu
Pipe(pipe1)- utworzenie pipe
Do15 – zmienna pilnująca aby w linii nie wyswietlono wiecej niż 15 znakow

```
                else
                {

                        signal(4,zakoncz);
                        signal(5, SIG_IGN);
                        signal(6, SIG_IGN);
                        printf("proces maciezysty pid %d\n\n", pid[0]);

                        while(1)
                        {}
                }
```

W procesie macierzystym:
Przechwycenie sygnału 4 i wykonanie funkcji zakoncz.
Zignorowanie sygnałów 5 i 6.
Wejście w stan wstrzymania (nicnierobienia).

Proces 1

```
if(fork()==0) //proces 1
        {
                pid[1]=getpid();
                printf("\nproces 1 pid %d\n", pid[1]);
                fflush(stdout);

                        signal(4,zakoncz);
                        signal(5, wstrzymaj);
                        signal(6, wznow);

                sleep(1);
                printf("Obsluga sygnalow:\n");
                printf("S1 - sygnal zakonczenia: 4 SIGILL\n");
                printf("S2 - sygnal wstrzymania: 5 SIGTRAP\n");
                printf("S3 - sygnal zakonczenia: 6 SIGABORT\n\n");

                printf("jakim sposobem chcesz podać dane?\n");
                printf("1. Ręcznie z konsoli\n");
                printf("2. Z pliku \n");
                printf("3. /dev/urandom \n");
                fflush(stdout);

                scanf("%d", &czym);

                getchar();
                fflush(stdin);
```

W **procesie 1:**
Stworzenie procesu pierwszego.
Przypisanie jego pidu do tablicy.
Przechwycenie sygnałów 4, 5 i 6 aby zostały obsłużone funkcjami kolejno: zakończ, wstrzymaj i wznów.
Wyświetlenie menu i pobranie od użytkownika jednej z opcji. Jeśli użytkownik wpisze cokolwiek innego dostanie opcje wpisywania z klawiatury.

```
if(czym==2)
{
	plikwej=pobierzplik();
}
if(czym==3)
{
	plikwej=fopen("/dev/urandom", "r");
		if(plikwej==NULL)
		{
			perror("Blad otwarcia /dev/urandom\n");
		}
}
```

Jeśli użytkownik wybrał opcje 2 – wywołanie funkcji pobierz plik. Przypisanie wskaźnika na plik zmiennej plikwej.
Jeśli użytkownik wybrał opcje 3 – przypisanie do plikwej wskaźnika na randomowy plik.

```
FILE *pobierzplik()
{
	char nazwapliku[20];
	FILE *plikwej;
	printf("\npodaj nazwę pliku:");
	scanf("%s", &nazwapliku);

	plikwej=fopen(nazwapliku, "r");
	if(plikwej==NULL)
	{
		printf("Blad otwarcia pliku wejsciowego wpisz inny plik\n");
		plikwej=pobierzplik();
	}
	return plikwej;
}
```

Funkcja **pobierzplik**:
Prosi użytkownika o nazwę pliku. W razie podania nieistniejącego pliku prosi jeszcze raz.
Zwraca wskaźnik na plik.

```
while(1)
                        {

        while(dzialanie)
                        {
```

Nieskończona pętla wykonująca zadanie procesu 1. Kod będzie się wykonywał tylko wtedy gdy proces nie będzie wstrzymany. W innym wypadku będzie nic nie robił do czasu wznowienia działania programu.

```
        int i;

        for(i=0;i<200; i++)

                bufor[i]=0;
```

Czyszczenie bufora;

```
if(czym==2||czym==3)
{


        if(fgets(bufor, 200, plikwej)==0)
        {
                printf("koniec pliku, nic wiecej do
pracy");
                fflush(stdout);
                while(1)
                {
                }
        }

}
```

Jeśli dane wejściowe były pobierane z pliku (w tym /dev/urandom) funkcją fgets pobieramy dane z pliku do uzyskania znaku końca linni lub do odczytania 200 znaków(zapełnienia bufora). Jeśli napotkamy znak końca pliku proces przechodzi w stan nicnierobiena.

```
        else
        {

        printf("\nwpisz tekst:");

        fflush(stdout);

        fgets(bufor, 200, stdin);
        }
```

W innym przypadku program prosi o użytkownika o wpisanie danych na konsole. Funkcja fgets w taki sam sposób jak w przypadku wyżej pobiera te dane do buforu.

```
plik1=fopen("/home/student/plik.txt", "w+");
if(plik1==NULL)
{
	perror("Blad otwarcia plik do zapisu\n");
}
```

Otworzenie pliku będącego narzędziem komunikacyjnym.

```
int i=0;
while(i<200)
{

		fprintf(plik1,"%c", bufor[i]);
		if(bufor[i]=='\n')
		{
		break;
		}
	i++;
}
```

Wpisanie znak po znaku wartości z bufora do pliku do przekazania procesowi 2.

Nie można przekazywać całego stringa ponieważ one posiadają na koniec \0 więc jeśli było by ono jako znak w pliku to co występuje za nim nie zostałoby przekazane.

```
while(access("/home/student/plik.txt",F_OK)==0)
	{
	}

		}
		}
}
```

Proces wchodzi w stan nicnierobienia do czasu aż nie będzie istniał plik do komunikacji.

Funkcja access z parametrem F_OK zwraca 0 jesli plik istnieje i -1 jesli plik nie istnieje.

Proces 2

```
else
	{
			if(fork()==0)// proces 2
			{
				pid[2]=getpid();
				printf("proces 2 pid %d\n", pid[2]);
				fflush(stdout);

							signal(4,zakoncz);
							signal(5, wstrzymaj);
							signal(6, wznow);

				close(pipe1[0]);
```

Stworzenie procesu nr 2.

Przypisanie jego pid do tablicy.

Przechwycenie sygnałów 4, 5 i 6 aby zostały obsłużone funkcjami kolejno: zakończ, wstrzymaj i wznów.
Zamknięcie wejścia czytającego pipe-a.

```
while(1)
{

	while(dzialanie)
	{
```

Nieskończona pętla wykonująca zadanie procesu 2. Kod będzie się wykonywał tylko wtedy gdy proces nie będzie wstrzymany. W innym wypadku będzie nic nie robił do czasu wznowienia działania programu.

```
int i;

	for(i=0;i<200; i++)

	{

		bufor1[i]=0;

	}
```

Wyczyszczenie buforu.

```
plik1=fopen("/home/student/plik.txt", "r");

if(plik1!=NULL)
{
        fgets(bufor1, 200, plik1);
```

Otworzenie pliku do odczytu. Jeśli otworzony poprawnie to rób dalej to co w kodzie, jeśli nie (nie istnieje), pomiń to i sprawdzaj znowu.

Funkcją fgets odczytanie zzawartości pliku.

```
int n=0;

while(bufor1[n]!='\n')
{
        if(n<200)
        {
                if(bufor1[n]=='o')
                {
                        o++;
                }
                n++;
        }
        else
        {
                break;
        }
}
```

Odczytuj bufor po literce tak długo aż nie napotkasz \n lub skończy się bufor. Zliczaj napotkane literki „o”.

```
if(bufor1[n]=='\n')
{
        sprintf(o1,"%i",o);// int na chara(stringa)


        write(pipe1[1], o1, 10);
        o=0;
}
```

Jeśli przerwanie nastąpliło z powodu znalezienia znaku końca linii:
Przekonwertuj liczbę literek „o” na stringa.
Zapisz(prześlij) go przez pipe. Wyzeruj liczbę o.

```
        fclose(plik1);
        remove("plik.txt");
        }
}
}
```

Zamknij używany plik. Usuń go. (W tym momencie wznowi się proces 1)

Proces 3

```
else
{
        if(fork()==0)// proces 3
        {
                pid[3]=getpid();
                printf("proces 3 pid %d\n", pid[3]);
                fflush(stdout);

                        signal(4,zakoncz);
                        signal(5, wstrzymaj);
                        signal(6, wznow);

                close(pipe1[1]);
```

Stworzenie procesu nr 3.

Przypisanie jego pid do tablicy.

Przechwycenie sygnałów 4, 5 i 6 aby zostały obsłużone funkcjami kolejno: zakończ, wstrzymaj i wznów.

Zamknięcie wejścia piszącego pipe-a.

```
while(1)
{

        while(dzialanie)
        {
```

Nieskończona pętla wykonująca zadanie procesu 3. Kod będzie się wykonywał tylko wtedy gdy proces nie będzie wstrzymany. W innym wypadku będzie nic nie robił do czasu wznowienia działania programu.

```
read(pipe1[0], o1, 10);

if(do15<14)
{

	fprintf(stderr,"%s ", o1);
	do15++;
}
else
{
	fprintf(stderr,"%s \n", o1);
	do15=0;
}
```

Zczytanie zawartości przesyłanej przez pipe.
Wypisywanie przeczytanej liczby oznaczającej ilość „o” w linii na standardowe wyjście diagnostyczne.

Obsługa sygnałów

```
void zakoncz(int sygnal)
{
	printf("zakoncz %d\n", getpid());
	fflush(stdout);
	kill(0,sygnal);

	if(getpid()==pid[0])
	{
		remove("plik.txt");
	}
	exit(0);
}
```

Sygnał 4 jest sygnałem kończącym działanie programu. Gdy zostanie on wyzłany do któregoś z procesów 1, 2 lub 3 zostaje on obsłużony przez funkcje zakończ.

Funkcją kill z pierwszym paramerem 0 jest on wysyłany do grupy procesów procesu który go odebrał. Oznacza to że sygnał ten jest wysyłany do matki oraz wszystkich trzech procesów potomnych.

Jeśli proces odbierający nie jest procesem macierzystym kończy swoje działanie funkcją exit, i staje się procesem zombie.

Gdy ten sygnał odbierze matka przed zakończeniem swojego działania usuwa plik do komunikacji jeśli taki jeszcze istnieje. I kończy swoje działanie zabijając przy tym swoje dzieci.

```
void wstrzymaj(int sygnal)
{
	if(dzialanie==1)
	{
		printf("wstrzymaj %d\n", getpid());
		fflush(stdout);

		dzialanie=0;

		kill(0, sygnal);
	}

}
```

Sygnał 5 jest sygnałem wstrzymującym prace procesów. Gdy zostanie on wyzłany do któregoś z procesów 1, 2 lub 3 zostaje on obsłużony przez funkcje wstrzymaj.

Jeśli program aktualnie nie jest wstrzymany, zmienna dzialanie zostanie ustawiona na 0 (w danym procesie) a sygnał zostanie przesłany do grupy procesów. Inne procesy po odebraniu go zrobią sobie to samo. Pocesy zostaną wstrzymane. Nie będą ragować też na ponowne wysyłanie tego sygnału.

Proces macierzysty ignoruje ten sygnał.

```
void wznow(int sygnal)
{
	if(dzialanie==0)
	{
		printf("wznow %d\n", getpid());
		fflush(stdout);

		dzialanie=1;

		kill(0, sygnal);
	}

}
```

Sygnał 5 jest sygnałem wznawiającym prace procesów. Gdy zostanie on wyzłany do któregoś z procesów 1, 2 lub 3 zostaje on obsłużony przez funkcje wznow.

Jeśli program aktualnie jest wstrzymany, zmienna dzialanie zostanie ustawiona na 1 (w danym procesie) a sygnał zostanie przesłany do grupy procesów. Inne procesy po odebraniu go zrobią sobie to samo. Pocesy zostaną wznowione. Gdy działają nie ragują na ten sygnał.

Proces macierzysty ignoruje ten sygnał.

Opis ogólny

W zadaniu do komunikacji między procesami użyto tak jak podano w treści zadania metody komunikacji fifo oraz pipe.

Pierwszy proces po zapisaniu odebranych danych do pliku musiał czekać aż stworzony plik nie będzie usunięty po odczytaniu z niego przez proces drugi. W innym wypadku proces pierwszy odczytał by dużo więcej danych w pewnym czasie niż zasoby zostały by przydzielone procesowi drugiemu (i trzeciemu) i te mogły by cokolwiek zrobić z otrzymanymi danymi.

Z komunikacją między procesem 2 a procesem 3 nie było juz takiego problemu ponieważ wysłana dana przez pipe może być odrazu odebrana. Nie ma potrzeby zamykania a potem ponownego otwierania pliku.

Wszystkie procesy wykonują pętle nieskończoną aby nie zakończyły się przed takim żądaniem użytkownika. Jednak gdy plik wejściowy zostanie odczytany w całości program już nic więcej nie zrobi. Oczekuje tylko na odpowiedni sygnał.

Wysłanie sygnału do dowolnego procesu powoduje, że zostaje on przeslany do całej grupy procesów i wszystkie (oprócz macierzystego który ignoruje sygnały wstrzymania i wznowienia) wykonują dane polecenie. Zostało wykorzystane takie rozwiązanie bo pocesy nie posiadają watości pid swoich braci.

Wstrzymanie procesu polega na wykonywaniu nieskończonej pętli nie posiadającej żadnych poleceń. Po wznowieniu dzialania udostępnia się pętla wykonująca pożądane działanie procesu.

Sygnał zakończenia powoduje zakończenie procesów oraz posprzątanie przez matkę używanego pliku.

Prezentacja wyników

```
Laboratorium:/home/student# gcc -o p p23.c
Laboratorium:/home/student# ./p

proces 1 pid 2657
proces 2 pid 2658
proces 3 pid 2659
proces maciezysty pid 2656

Obsluga sygnalow:
S1 - sygnal zakonczenia: 4 SIGILL
S2 - sygnal wstrzymania: 5 SIGTRAP
S3 - sygnal zakonczenia: 6 SIGABORT

jakim sposobem chcesz podać dane?
1. Ręcznie z konsoli
2. Z pliku
3. /dev/urandom
1

wpisz tekst:ooo
3
wpisz tekst:abc
0
wpisz tekst:ola
1
wpisz tekst:wstrzymaj 2657
wstrzymaj 2659
wstrzymaj 2658
wznow 2659
wznow 2657
wznow 2658
ooko
3
wpisz tekst:zakoncz 2659
zakoncz 2657
zakoncz 2658
zakoncz 2656
Laboratorium:/home/student#
```

```
Laboratorium:/home/student# kill -5 2657
Laboratorium:/home/student# kill -5 2658
Laboratorium:/home/student# kill -6 2659
Laboratorium:/home/student# kill -4 2659
Laboratorium:/home/student#
```

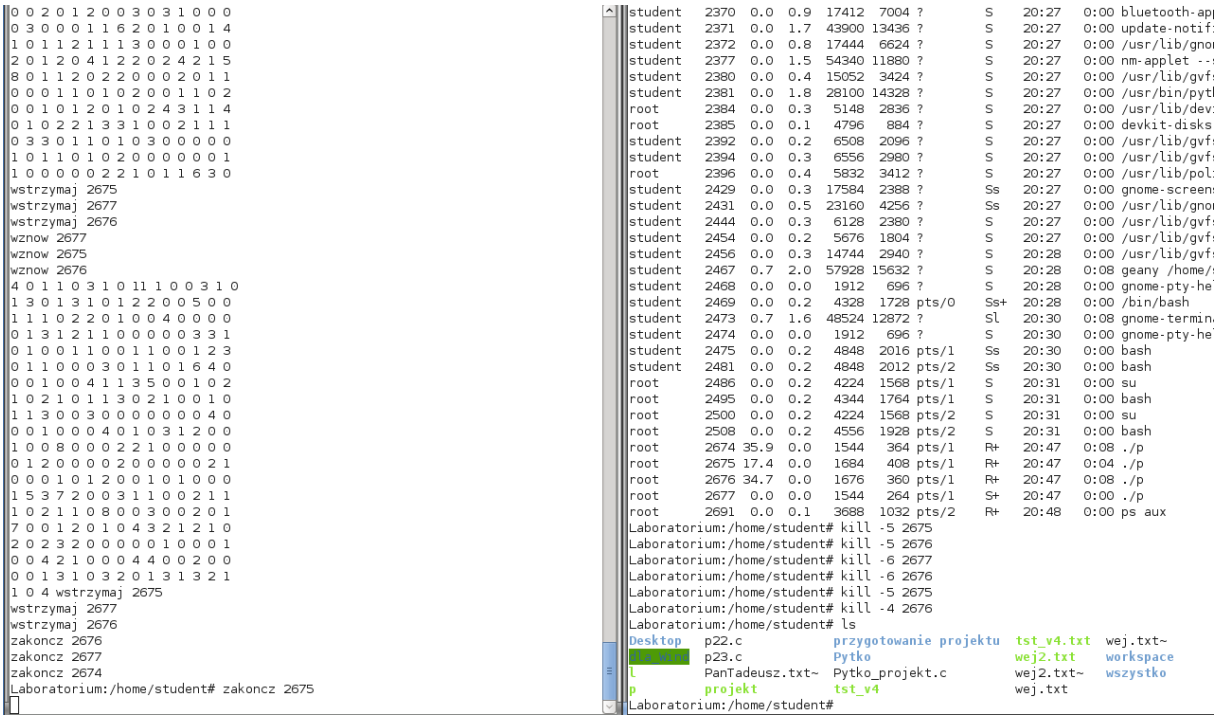
Wpisywanie z konsoli.

Pierwszy sygnał wstrzymania wstrzymuje. Drugi nic nie zmienia. Wznowienie powoduje przywrócenie dzialania programu. I ostatni sygnał zakańczający.

```
Laboratorium:/home/student# ./p

proces 1 pid 2666
proces 2 pid 2667
proces 3 pid 2668
proces maciezysty pid 2665

Obsluga sygnalow:
S1 - sygnal zakonczenia: 4 SIGILL
S2 - sygnal wstrzymania: 5 SIGTRAP
S3 - sygnal zakonczenia: 6 SIGABORT

jakim sposobem chcesz podać dane?
1. Ręcznie z konsoli
2. Z pliku
3. /dev/urandom
2

podaj nazwę pliku:tst_v4
1 3 0 0 2 2 koniec pliku, nic wiecej do pracyzakoncz 2666
zakoncz 2668
zakoncz 2665
Laboratorium:/home/student# zakoncz 2667
```

```
Laboratorium:/home/student# cat tst_v4
Jakiś tekst z polskimi znakami, zażółć gęślą jaźń.
Mała pułapka ooo

        oko
większa pułapka                 ß oOo
Laboratorium:/home/student# kill -4 2666
Laboratorium:/home/student#
```

Odczytywanie z pliku.



/dev/urandom.

Sygnał 5 wstrzymał wszystkie procesy. Kolejny sygnał 5 nic nie zmienił.
Sygnał 6 zwnowił wszystkie procesy. Kolejny sygnał 6 nic nie zmienił.
Zamknięcie działania programu działa również w czasie wstrzymania.
Program usuwa po sobie plik plik.txt.

Wnioski

Program poprawnie zlicza ilość liter „o” zarówno z klawiatury jak i pliku. Znak NULL w pliku traktuje jako zwykły znak i odczytuje plik do końca. Na wysyłane do różnych procesów sygnały reaguje zgodnie z oczekiwaniami, albo cały sie wstrzymuje albo wznawia. Nie reaguje na sygnały wywołujące stan w którym się obecnie znajduje.

Po sygnale zakończenia działania wyłącza się nie pozostawiając po sobie używanego pliku.

Wszystkie założenia projektu zostały więc spelnione a program działa poprawnie.