«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И.Ульянова (Ленина)»
СПбГЭТУ «ЛЭТИ»

Кафедра вычислительной техники

Отчет по по лабораторной работе №7 по дисциплине «Организация процессов и программирование в среде Linux»
Тема: «ОБМЕН ДАННЫМИ ЧЕРЕЗ КАНАЛ»

Студент гр.8306 _____ Слепов А.Э.

Преподаватель _____ Разумосвский Г.В.

Санкт-Петербург

2021

Цель работы

Знакомство с механизмом обмена данными через программный канал и системными вызовами, обеспечивающими такой обмен.

Задание

1. Написать программу, которая обменивается данными через канал с двумя потомками. Программа открывает входной файл, построчно читает из него данные и записывает их в канал. Потомки выполняют свои программы и поочередно читают символы из канала и записывают их в свои выходные файлы: первый потомок – нечетные символы, а второй – четные. Синхронизация работы потомков должна осуществляться напрямую с использованием сигналов SIGUSR1 и SIGUSR2. Об окончании записи файла в канал программа оповещает потомков сигналом SIGQUIT и ожидает завершения работы потомков. Когда они заканчивают работу, программа закрывает канал.

2. Откомпилировать все программы и запустить их

Порядок выполнения работы

На рисунке 1 представлен фрагмент вывода программы для входного файла рис. 2. Можно наблюдать, что для синхронизации процессы обмениваются сигналами SIGUSR1 и SIGUSR2(потомок 1 посылает SIGUSR1, потомок 2 – SIGUSR2). Если процесс получает сигнал синхронизации, но при этом канал пуст и родитель уже закончил писать в канал(то есть послал SIGQUIT), процесс завершает свою работу и посылает второму процессу сигнал синхронизации. Получив этот сигнал, второй процесс также завершает работу. Родитель, дождавшись завершения обоих потомков, тоже завершает работу.

```
Потомок 1: 1
Потомок 2 получил SIGUSR1 и продолжает работать
Потомок 2: 2
Потомок 1 получил SIGUSR2 и продолжает работать
Потомок 1: 1
Потомок 2 получил SIGUSR1 и продолжает работать
Потомок 2: 2
Потомок 1 получил SIGUSR2 и продолжает работать
Потомок 1: 1
Потомок 2 получил SIGUSR1 и продолжает работать
Потомок 2: 2
Потомок 1 получил SIGUSR2 и продолжает работать
Потомок 1: 1
Потомок 2 получил SIGUSR1 и продолжает работать
Потомок 2: 2
Потомок 1 получил SIGUSR2 и продолжает работать
Потомок 1: 1
Потомок 2 получил SIGUSR1 и продолжает работать
Потомок 2:

Потомок 1 получил SIGUSR2 и продолжает работать
Потомок 1: канал пуст, предок закончил писать
Происходит выход из потомка 1
Потомок 2 получил SIGUSR1 и продолжает работать
Потомок 2: канал пуст, предок закончил писать
Происходит выход из потомка 2
Родитель дождался 1 потомка и завершает работу
Родитель дождался 2 потомка и завершает работу
```

Рисунок 1. Результаты работы программы

На рисунке 2 представлен входной файл.

```
artem@legion:~/Документы/linux-labs/lab7$ cat input.txt
1)12121212121212121212121212121212121212121212121212121212121212121212121212
2)21212121212121212121212121212121212121212121212121212121212121212121212121
3)21212121212121212121212121212121212121212121212121212121212121212121212121
4)21212121212121212121212121212121212121212121212121212121212121212121212121
5)21212121212121212121212121212121212121212121212121212121212121212121212121
6)21212121212121212121212121212121212121212121212121212121212121212121212121
```

Рисунок 2. Входной файл для родителя

Выходные файлы двух потомков представлены на рисунках 3 и 4 соответственно. Первый потомок читал символы перевода строки, поэтому в нем получилось несколько строк, второй потомок с «\n» уже не сталкивался, поэтому в нем получилась 1 строка.

```
artem@legion:~/Документы/linux-labs/lab7$ cat 1.txt
1111111111111111111111111111111111111111
)1111111111111111111111111111111111111111
)1111111111111111111111111111111111111111
)1111111111111111111111111111111111111111
)1111111111111111111111111111111111111111
)11111111111111111111111111111111111111artem@legion:~/Документы/linux-labs/lab7$
```

Рисунок 3. Выходной файл первого потомка

```
artem@legion:~/Документы/linux-labs/lab7$ cat 2.txt
)22222222222222222222222222222222222222222222222222222222222222222222222232222222222222222
22222222222222222222222
```

Рисунок 4. Выходной файл второго потомка

Выводы

В ходе работы были изучены механизмы обмена данными через программный канал и системными вызовами, обеспечивающими такой обмен, в операционной системе Ubuntu.

ПРИЛОЖЕНИЕ А
Текст программы-родителя

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

#define BUF_SIZE 128


void signal_handler(int sig){
      if (sig == SIGQUIT) {
        printf("Родитель получил свой же SIGQUIT\n");
    }A
    if (sig == SIGPIPE) {
        printf("Труба еще не открыта на чтение!\n");
    }
}

int main(int argc, char** argv){
      if(argc == 4){
            char buf[BUF_SIZE];
            FILE *input = NULL;
            int fd[2];
            pid_t pid_1, pid_2;

           signal(SIGQUIT, signal_handler);
            signal(SIGUSR1, SIG_IGN);
            signal(SIGUSR2, SIG_IGN);
            signal(SIGPIPE, signal_handler);
            if (pipe(fd) == -1) {
            printf("Ошибка создания канала\n");
            exit(EXIT_FAILURE);
      }
      if ( (input = fopen(argv[1], "r")) == NULL) {
            printf("Ошибка открытия входного файла\n");
            exit(EXIT_FAILURE);
      }

      if (!(pid_1 = fork())) {
            execl("proc1", "proc1", &fd[0], argv[2], NULL);
      }

      if (!(pid_2 = fork())) {
            execl("proc2", "proc2", &fd[0], argv[3], NULL);
      }
      sleep(1);
      fcntl(*fd, F_SETFL, O_NONBLOCK);


      while (fgets(buf, BUF_SIZE, input) != NULL) {
            sleep(2);
            printf("Прочитали: %s", buf);
              if (write(fd[1], buf, strlen(buf)) == -1) {
                  printf("Ошибка записи в канал\n");
                  exit(EXIT_FAILURE);
              }
          }

      printf("Родитель закончил писать в канал и послал SIGQUIT\n");
```

```
	kill(0, SIGQUIT);

	waitpid(pid_1, NULL, 0);
	printf("Родитель дождался 1 потомка и завершает работу\n");
	waitpid(pid_2, NULL, 0);
	printf("Родитель дождался 2 потомка и завершает работу\n");

	close(fd[0]);
	close(fd[1]);
	fclose(input);
	exit(EXIT_SUCCESS);
	}
	else
		printf("Недостаточно аргументов для запуска\n");
	return 0;
}
```

ПРИЛОЖЕНИЕ Б
Текст программы-потомка 1

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


FILE *output = NULL;
int read_pipe_fd;
char ch;
char readFlag = 1;
int readReturn;
char sigquitFlag = 0;

void signal_handler(int sig){
      if (sig == SIGQUIT) {
        printf("Потомок 1 получил SIGQUIT родителя\n");
        sigquitFlag = 1;

    }
    if (sig == SIGUSR2){
      printf("Потомок 1 получил SIGUSR2 и продолжает работать\n");
      readReturn = read(read_pipe_fd, &ch, 1);
      if(readReturn > 0){
                  printf("Потомок 1: %c\n", ch);
                  fprintf(output, "%c", ch);
                  readFlag = 1;
            kill(0, SIGUSR1);
      }
      else{
            if(sigquitFlag == 0){
                  printf("Потомок 1: канал пуст, предок не закончил писать\n");
                  kill(0, SIGUSR2);
            }
            if(sigquitFlag == 1){
                  printf("Потомок 1: канал пуст, предок закончил писать\n");
                  kill(0, SIGUSR1);
                  printf("Происходит выход из потомка 1\n");
                  fclose(output);
                  exit(EXIT_SUCCESS);
            }
      }
    }

    if (sig == SIGUSR1){
      //printf("Потомок 1 получил свой же SIGUSR1\n");
    }
}

int main(int argc, char** argv){
      if(argc == 3){
            read_pipe_fd = *argv[1];

            signal(SIGQUIT, signal_handler);
            signal(SIGUSR1, signal_handler);
            signal(SIGUSR2, signal_handler);
            output = fopen(argv[2], "a");
```

```
            read(read_pipe_fd, &ch, 1);
            fputc(ch, output);
        printf("Потомок 1: %c\n", ch);
        kill(0, SIGUSR1);

            while(readFlag || !sigquitFlag)
                  pause();

      }
      else
            printf("Недостаточно аргументов для запуска\n");
      return 0;
}
```

ПРИЛОЖЕНИЕ В
Текст программы-потомка 2

```
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include <fcntl.h>
#include <stdbool.h>

FILE *output = NULL;
int read_pipe_fd;
char ch;
char readFlag = 1;
char sigquitFlag = 0;

int readReturn;

void signal_handler(int sig){
      if (sig == SIGQUIT) {
        printf("Потомок 2 получил SIGQUIT родителя\n");
        sigquitFlag = 1;
    }
    if (sig == SIGUSR1){
      printf("Потомок 2 получил SIGUSR1 и продолжает работать\n");
      readReturn = read(read_pipe_fd, &ch, 1);
      if(readReturn > 0){
                  printf("Потомок 2: %c\n", ch);
                  fprintf(output, "%c", ch);
                  readFlag = 1;
            kill(0, SIGUSR2);
      }
      else{
            readFlag = 0;
            if(sigquitFlag == 0){
                  printf("Потомок 2: канал пуст, предок не закончил писать\n");
                  kill(0, SIGUSR1);
            }
            if(sigquitFlag == 1){
                  printf("Потомок 2: канал пуст, предок закончил писать\n");
                  kill(0, SIGUSR2);
                  printf("Происходит выход из потомка 2\n");
                  fclose(output);
                  exit(EXIT_SUCCESS);
            }
      }

    }

    if (sig == SIGUSR2){
      //printf("Потомок 2 получил свой же SIGUSR2\n");
    }
}


int main(int argc, char** argv){
      if(argc == 3){
            read_pipe_fd = *argv[1];
            output = fopen(argv[2], "a");

            signal(SIGQUIT, signal_handler);
            signal(SIGUSR1, signal_handler);
```

```
		signal(SIGUSR2, signal_handler);

		while(readFlag || !sigquitFlag)
			pause();
	}
	else
		printf("Недостаточно аргументов для запуска\n");
	return 0;
}
```

ПРИЛОЖЕНИЕ Г
Текст скрипта для запуска всех программ

```
#!/bin/sh
rm 1.txt
rm 2.txt
gcc proc1.c -o proc1
gcc proc2.c -o proc2
gcc lab7.c
./a.out input.txt 1.txt 2.txt
```