Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Курсовой проект по курсу
«Операционные системы»

Студент: Харьков Павел Александрович
Группа: М8О–206Б–19
Вариант: на удовлетворительно
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021.

Постановка задачи

Цель работы

- Приобретение практических навыков в использовании знаний, полученных в течении курса.
- Проведение исследования в выбранной предметной области

Задание

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

Вариант: Необходимо написать 3-и программы. Далее будем обозначать эти программы A, B, C. Программа A принимает из стандартного потока ввода строки, а далее их отправляет программе C. Отправка строк должна производиться построчно. Программа C печатает в стандартный вывод, полученную строку от программы A. После получения программа C отправляет программе A сообщение о том, что строка получена. До тех пор, пока программа A не примет «сообщение о получение строки» от программы C, она не может отправлять следующую строку программе C. Программа B пишет в стандартный вывод количество отправленных символов программой A и количество принятых символов программой C. Данную информацию программа B получает от программ A и C соответственно.

Взаимодействие между программами будет происходить с помощью однонаправленных каналов.

Общие сведения о программе.

Программы компилируются из файлов a.c, b.c, c.c, get_line.c . Также используется заголовочные файлы: unistd.h, stdlib.h, stdio.h, string.h, signal.h. В программе используются следующие системные вызовы:

- fork – создает копию процесса.
- pipe – создает канал данных для взаимодействия между процессами.
- read – читает в память из файлового дескриптора определенное количество байт.
- write – пишет в файловый дескриптор из памяти определенное количество байт.
- close – закрывает файловый дескриптор.
- kill – посылает сигнал по pid процесса.
- execl – заменяет текущий образ процесса новым образом процесса.
- perror – выводит в стандартный поток ошибки сообщения.

Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Первый программа A создаст два дочерних процесса B и C. Затем будет считывать строку из стандартного потока ввода и отправлять размер строки, а затем и саму строку программе C с помощью pipe.
2. Программа C считает размер строки, а затем строку, напечатает строку в стандартный поток вывода и отправит программе A цифру 1, что значит, что она прочитала строку.
3. Затем программа A отправит программе B размер строки, которую она отправила программе C, а программа C отправит программе B размер строки, который она получила.
4. Этот алгоритм будет работать, пока будут вводиться строки, затем программа A пошлет сигнал, чтобы завершить процессы B и C.
5. Отладить программу и протестировать на тестах.

Основные файлы программы.

a.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>

#include "get_line/get_line.h"

int id1, id2;
int pipeAC[2];
int pipeAB[2];
int pipeCA[2];
int pipeCB[2];

void sig_handler(int signal) {
    kill(id1, SIGUSR1);
    kill(id2, SIGUSR1);

    close(pipeAC[0]);
    close(pipeAC[1]);
```

```
    close(pipeCA[0]);
    close(pipeCA[1]);
    close(pipeAB[0]);
    close(pipeAB[1]);
    close(pipeCB[0]);
    close(pipeCB[1]);
    exit(0);
}

int main(){

    if (signal(SIGINT, sig_handler) == SIG_ERR) {
        printf("[%d] ", getpid());
        perror("Error signal ");
        return -1;
    }
    pipe(pipeAC);
    pipe(pipeAB);
    pipe(pipeCA);
    pipe(pipeCB);


    id1 = fork();
    if (id1 == -1)
    {
        perror("fork1");
        exit(-1);
    }
    else if(id1 == 0)
    {
        char name[] = "./b";
        char pAB[3] = "";
        sprintf(pAB, "%d", pipeAB[0]);
        char pCB[3] = "";
        sprintf(pCB, "%d", pipeCB[0]);

        close(pipeAC[0]);
        close(pipeAC[1]);
        close(pipeCA[0]);
        close(pipeCA[1]);
        close(pipeAB[1]);
        close(pipeCB[1]);

        execl(name, name, pAB, pCB, NULL);
    }
    else
    {
        id2 = fork();
        if (id2 == -1)
        {
```

```
            perror("fork2");
            exit(-1);
        }
        else if(id2 == 0)
        {
            char name[] = "./c";
            char pAC[3] = "";
            sprintf(pAC, "%d", pipeAC[0]);
            char pCA[3] = "";
            sprintf(pCA, "%d", pipeCA[1]);
            char pCB[3] = "";
            sprintf(pCB, "%d", pipeCB[1]);

            close(pipeAC[1]);
            close(pipeCA[0]);
            close(pipeAB[0]);
            close(pipeAB[1]);
            close(pipeCB[0]);
            execl(name, name, pAC, pCA, pCB, NULL);
        }
        else
        {
            char* line = NULL;
            int size;
            while((size = get_line(&line, STDIN_FILENO)) != 0)
            {
                write(pipeAC[1], &size, sizeof(int));
                write(pipeAC[1], line, size*sizeof(char));

                int ok;
                read(pipeCA[0], &ok, sizeof(int));

                write(pipeAB[1], &size, sizeof(int));
            }
            free(line);

            kill(id1, SIGUSR1);
            kill(id2, SIGUSR1);
        }
    }

    close(pipeAC[0]);
    close(pipeAC[1]);
    close(pipeCA[0]);
    close(pipeCA[1]);
    close(pipeAB[0]);
    close(pipeAB[1]);
    close(pipeCB[0]);
    close(pipeCB[1]);
}
```

b.c

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

int pipeAB;
int pipeCB;

void sig_handler(int signal) {
   close(pipeAB);
   close(pipeCB);
   exit(0);
}

int main(int argc, char *argv[]){
   if (signal(SIGUSR1, sig_handler) == SIG_ERR) {
      printf("[%d] ", getpid());
      perror("Error signal ");
      return -1;
   }
   pipeAB = atoi(argv[1]);
   pipeCB = atoi(argv[2]);

   int sizeA;
   int sizeB;
   while(read(pipeAB, &sizeA, sizeof(int)) > 0 && read(pipeCB, &sizeB, sizeof(int)) > 0){
      printf("[B] Get A: %d; Get C: %d\n", sizeA, sizeB);
   }

}
```

c.c

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

int pipeAC;
int pipeCA;
int pipeCB;

void sig_handler(int signal) {
   close(pipeAC);
   close(pipeCA);
   close(pipeCB);

   exit(0);
```

```
}

int main(int argc, char *argv[]){
    if (signal(SIGUSR1, sig_handler) == SIG_ERR) {
        perror("[C] Error signal ");
        return -1;
    }
    pipeAC = atoi(argv[1]);
    pipeCA = atoi(argv[2]);
    pipeCB = atoi(argv[3]);

    int sizeA;
    while(read(pipeAC, &sizeA, sizeof(int))> 0){
        char line[sizeA+1];
        line[sizeA] = '\0';
        read(pipeAC, line, sizeA * sizeof(char));
        printf("[C] Get from A: %s\n", line);

        int ok = 1;
        write(pipeCA, &ok, sizeof(int));

        write(pipeCB, &sizeA, sizeof(int));

    }

}
```

get_line.c

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

#include "get_line.h"

int get_char(int fd)
{
    char c;
    if (read(fd, &c, 1) == 1)
        return (unsigned char)c;
    return EOF;
}

int get_line(char** line_, int fd)
{
    free(*line_);
    int c;
    int size = 0;
    int cap = 4;
    char* line = (char*)malloc(cap * sizeof(char));
    while((c = get_char(fd)) != EOF && c != '\n' && c!= '\r')
```

```
    {
        if(size == cap)
        {
            cap *= 2;
            line = (char*)realloc(line, cap * sizeof(char));
            if(line == NULL)
                exit(-1);
        }
        line[size] = c;
        ++size;
    }
    if(size == cap)
    {
        cap += 1;
        line = (char*)realloc(line, cap * sizeof(char));
        if(line == NULL)
            exit(-1);
    }
    line[size] = '\0';
    *line_ = line;
    return size;
}
```

Пример работы программы.

```
pablo$ make
gcc -pedantic -Wall src/get_line/get_line.c -c -o src/get_line/get_line.o
gcc -pedantic -Wall src/a.c src/get_line/get_line.o -o a
gcc -pedantic -Wall src/b.c src/get_line/get_line.o -o b
gcc -pedantic -Wall src/c.c src/get_line/get_line.o -o c
pablo$ cat tests/test1
hello
world
kwfjekejfkfjkeejfkeffkej
pablo$ ./a < tests/test1
[C] Get from A: hello
[B] Get A: 5; Get C: 5
[C] Get from A: world
[B] Get A: 5; Get C: 5
[C] Get from A: kwfjekejfkfjkeejfkeffkej
[B] Get A: 24; Get C: 24
pablo$ ./a
hi
[C] Get from A: hi
[B] Get A: 2; Get C: 2
my name is pasha
[C] Get from A: my name is pasha
[B] Get A: 16; Get C: 16
```

Выводы.

Для написания данного курсового проекта мне пригодились знания, которые я получил во время курса. Я использовал однонаправленные каналы, о которых узнал, делая 2 лабораторную работу; создание процессов, которое я использовал на протяжении всего курса; сигналы, необходимые для 4 и 6 лабораторных работ; системный вызов execl, который я использовал в 6. Я допустил серьезную ошибку, когда передавал в аргументы execl целые числа, а не строки из-за чего моя программа работала некорректно. Также проанализировав, какой метод использовать для межпроцессного взаимодействия – однонаправленные каналы, отображение файлов в память или сервер сообщений ZeroMQ, я пришел к выводу, что для такой задачи я буду использовать простые однонаправленные каналы.