Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

Студент: Армишев Кирилл Константинович

Группа: М8О–208Б–21

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023

**Постановка задачи**

## Задание

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 С. Отправка строк должна производится построчно. Программа C печатает в стандартный вывод, полученную строку от программы A. После получения программа C отправляет программе А сообщение о том, что строка получена. До тех пор пока программа А не примет «сообщение о получение строки» от программы С, она не может отправлять следующую строку программе С. Программа B пишет в стандартный вывод количество отправленных символов программой А и количество принятых символов программой С. Данную информацию программа B получает от программ A и C соответственно. Способ организация межпроцессорного взаимодействия выбирает студент.

**Общие сведения о программе**

Программа состоит из трех файлов: a.c, b.c, c.c, используются заголовочные файлы: stdio.h, stdlib.h, unistd.h, sys/wait.h, fcntl.h, string.h. В ходе работы были применены следующие системные вызовы:

1. **write () -** переписывает count байт из буффера в файл. Возвращает количество записанных байт или -1;
2. **read () -** считывает count байт из файла в буффер. Возвращает количество считаных байт (оно может быть меньше count) или -1;
3. **pipe () -** создаёт канал между двумя процессами. Создаёт и помещает в массив 2 файловых дескриптора для чтения и для записи. Возвращает 0 или -1;
4. **close() -** закрывает файловый дескриптор, который больше не ссылается ни на один файл, возвращает 0 или -1;
5. **fork () -** порождается процесс-потомок. Весь код после fork() выполняется дважды, как в процессе-потомке, так и в процессе-родителе. Процесс-потомок и процесс-родитель получают разные коды возврата после вызова fork(). Процесс-родители возвращает идентификатор pid потомка или -1. Процесс-потомок возвращает 0 или -1;
6. **execv () -** заменяет текущий образ процесса новым образом процесса. Новая программа наследует от вызывавшего процесса идентификатор и открытые файловые дескрипторы;

**Общий метод и алгоритм решения**

С помощью двух вызовов fork(), создаются два дочерних процесса В и С. А получает от пользователя строку, длинна которой ограничивается 100 символами, и передает ее в С через pipe. С получает строку и выводит ее в стандартный поток вывода, после этого отправляет программе А через канал длину строки (т.к. по условию, пока программа А не примет «сообщение о получение строки» от программы С, она не может отправлять следующую строку программе С). А и С отправляют через каналы в программу В размеры строк и В выводит их. Программа прекращает свою работу в случае введения пустой строки.

**Основные файлы программы**

======================== a.c ========================

#include <stdio.h>  
#include <unistd.h>  
#include <string.h>  
  
int main() {  
 int A\_C[2];  
 if (pipe(A\_C)==-1){  
 perror("error pipe1");  
 }  
 int A\_B[2];  
 if (pipe(A\_B)==-1){  
 perror("error pipe2");  
 }  
 int C\_B[2];  
 if (pipe(C\_B)==-1){  
 perror("error pipe3");  
 }  
 int C\_A[2];  
 if (pipe(C\_A)==-1){  
 perror("error pipe3");  
 }  
 int id = fork();  
 if (id == -1)  
 {  
 perror("fork error");  
 return 0;  
 }else if(id == 0) {  
 close(A\_C[1]); close(A\_C[0]); close(C\_B[1]); close(A\_B[1]); close(C\_A[1]); close(C\_A[0]);  
 char A\_read[10];  
 char C\_read[10];  
 sprintf(A\_read, "%d", A\_B[0]);  
 sprintf(C\_read, "%d", C\_B[0]);  
 char \*B\_argv[] = {"b", A\_read, C\_read, NULL};  
 execv("b", B\_argv);  
 }  
 int id2;  
 if(id > 0) {  
 id2 = fork();  
 if (id2 == -1)  
 {  
 perror("fork2 error");  
 return 0;  
 }else if(id2 == 0) {  
 close(A\_C[1]); close(C\_B[0]); close(A\_B[1]); close(C\_A[0]); close(A\_B[0]);  
 char A\_read[10];  
 char A\_write[10];  
 char B\_write[10];  
 sprintf(A\_read, "%d", A\_C[0]);  
 sprintf(A\_write, "%d", C\_A[1]);  
 sprintf(B\_write, "%d", C\_B[1]);  
  
 char \*C\_argv[] = {"c", A\_read, A\_write, B\_write, NULL};  
  
 execv("c", C\_argv);  
 }  
 }  
 if(id != 0 && id2 != 0) {  
 close(A\_C[0]); close(A\_B[0]); close(C\_B[0]); close(C\_B[1]); close(C\_A[1]);  
 char string[100];  
 while (1) {  
 if(gets(string) == NULL) {  
 perror("gets error");  
 return 0;  
 }  
 int length = strlen(string);  
  
 if(length == 0) {  
 if(write(A\_C[1], &length, sizeof(length)) == -1) {  
 perror("Write in A\_C error");  
 return 0;  
 }  
  
 if(write(A\_B[1], &length, sizeof(length)) == -1) {  
 perror("Write in A\_B error");  
 return 0;  
 }  
 break;  
 }  
 if(write(A\_C[1], &length, sizeof(length)) == -1) {  
 perror("Write in A\_C error");  
 return 0;  
 }  
 if(write(A\_C[1], &string, sizeof(char) \* length) == -1) {  
 perror("Write in A\_C error");  
 return 0;  
 }  
  
 if(write(A\_B[1], &length, sizeof(length)) == -1) {  
 perror("Write in A\_B error");  
 return 0;  
 }  
  
 int check = 0;  
 if(read(C\_A[0], &check, sizeof(check)) == -1) {  
 perror("Read from C\_A error");  
 return 1;  
 }  
 }  
 }  
 return 0;  
}

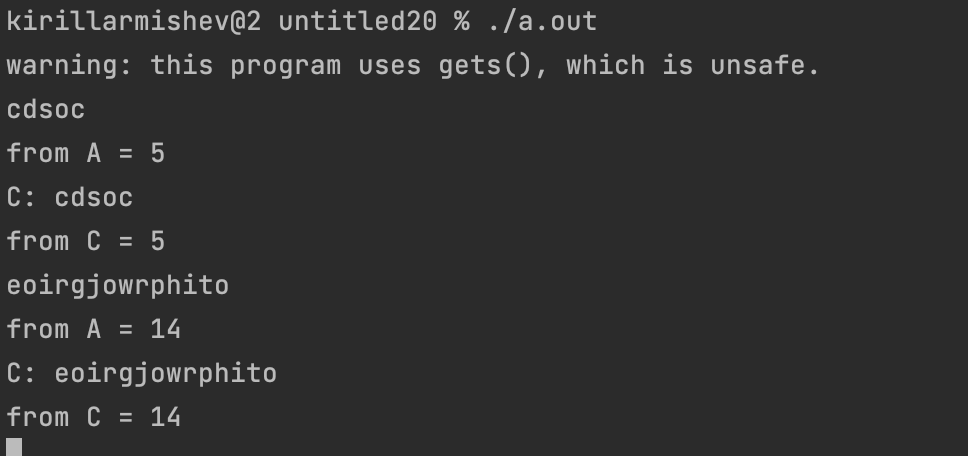
======================== b.c ========================

#include <stdio.h>  
#include <unistd.h>  
#include <stdlib.h>  
  
  
  
int main(int argc, const char \*argv[]) {  
 int length;  
 int file\_descriptors[2];  
 file\_descriptors[0] = atoi(argv[1]); //read from A  
 file\_descriptors[1] = atoi(argv[2]); //read from C  
  
 while (1){  
 if (read(file\_descriptors[0], &length, sizeof(int)) == -1) {  
 perror("Read from A error");  
 }  
  
 if(length == 0) {  
 break;  
 }  
  
 printf("from A = %d\n", length);  
  
 if(read(file\_descriptors[1], &length, sizeof(int)) == -1) {  
 perror("Read from C error");  
 return 1;  
 }  
 printf("from C = %d\n", length);  
 }  
  
 return 0;  
}

======================== c.c ========================

#include <stdio.h>  
#include <unistd.h>  
#include <stdlib.h>  
#include <string.h>  
  
  
int main(int argc, const char \*argv[]) {  
 int file\_descriptors[3];  
 file\_descriptors[0] = atoi(argv[1]); //read from A  
 file\_descriptors[1] = atoi(argv[2]); //write to A  
 file\_descriptors[2] = atoi(argv[3]); //write to B  
  
 int length;  
 char string[100];  
 memset(string, 0, 100);  
  
 while (1) {  
 if(read(file\_descriptors[0], &length, sizeof(int)) == -1) {  
 perror("Read from A error");  
 }  
  
 if(length == 0) {  
 break;  
 }  
  
 if(read(file\_descriptors[0], &string, sizeof(char) \* length) == -1) {  
 perror("Read from A error");  
 }  
  
 printf("C: %s\n", string);  
  
 if(write(file\_descriptors[1], &length, sizeof(int)) == -1) {  
 perror("Write to A error");  
 }  
  
 if(write(file\_descriptors[2], &length, sizeof(int)) == -1) {  
 perror("Write to B error");  
 }  
 memset(string, 0, 100);  
 length = 0;  
 }  
 return 0;  
}

**Пример работы**

****

**Вывод**

В ходе данного курсового проекта я попрактиковался в управлении процессами при помощи системных вызовов и обеспечении обмена данными между процессами при помощи неименованных каналов.

Системные вызовы необходимы для управления процессами, файлами и каталогами, а также каналами ввода и вывода данных. Одним из способов создания дочернего процесса является системный вызов fork(), он создает точную копию исходного процесса, включая все дескрипторы файлов, регистры и т. п. После выполнения вызова fork() исходный процесс и его копия (родительский и дочерний процессы) выполняются независимо друг от друга. Благодаря систему вызову pipe можно создать канал между двумя процессами, в которой один процесс сможем писать поток байтов, а другой процесс сможет его читать, так мы переопределяем потоки ввода-вывода.

Благодаря системным вызовам можно упростить программу или выполнить действия, запрещенные в пользовательском режиме.

При написании курсового проекта я закрепил свои знания и навыки, полученные во время прохождения курса операционных систем.