Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Курсовой проект по курсу**

**«Операционные системы»**

Студент: Шевлякова София Сергеевна

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 pipeA\_to\_C. 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 <stdlib.h>

#include <fcntl.h>

#include <string.h>

#include <sys/wait.h>

#define READ 0

#define WRITE 1

void send\_error\_and\_stop(char \*massage, int code) {

int i = strlen(massage);

write(STDERR\_FILENO, massage, sizeof(char) \* i);

exit(code);

}

int main() {

int file\_descriptorsA\_to\_C[2];

int file\_descriptorsA\_to\_B[2];

int file\_descriptorsC\_to\_B[2];

int file\_descriptorsC\_to\_A[2];

int pipeA\_to\_C = pipe(file\_descriptorsA\_to\_C);

int pipeA\_to\_B = pipe(file\_descriptorsA\_to\_B);

int pipeC\_to\_B = pipe(file\_descriptorsC\_to\_B);

int pipeC\_to\_A = pipe(file\_descriptorsC\_to\_A);

if (pipeA\_to\_C == -1 || pipeA\_to\_B == -1 || pipeC\_to\_B == -1 || pipeC\_to\_A == -1) {

send\_error\_and\_stop("Pipe error\n", 1);

}

switch (fork()) {

case -1:

send\_error\_and\_stop("Fork error\n", 2);

case 0:

close(file\_descriptorsA\_to\_C[WRITE]);

close(file\_descriptorsA\_to\_C[READ]);

close(file\_descriptorsC\_to\_B[WRITE]);

close(file\_descriptorsA\_to\_B[WRITE]);

close(file\_descriptorsC\_to\_A[WRITE]);

close(file\_descriptorsC\_to\_A[READ]);

char file\_descriptor\_read\_A[10];

char file\_descriptor\_read\_C[10];

sprintf(file\_descriptor\_read\_A, "%d", file\_descriptorsA\_to\_B[READ]);

sprintf(file\_descriptor\_read\_C, "%d", file\_descriptorsC\_to\_B[READ]);

char \*B\_argv[] = {"b", file\_descriptor\_read\_A, file\_descriptor\_read\_C, NULL};

if (execv("b", B\_argv) == -1) {

send\_error\_and\_stop("Exec B error\n", 3);

}

break;

default:

switch (fork()) {

case -1:

send\_error\_and\_stop("Fork error\n", 2);

case 0:

close(file\_descriptorsA\_to\_C[WRITE]);

close(file\_descriptorsA\_to\_B[WRITE]);

close(file\_descriptorsA\_to\_B[READ]);

close(file\_descriptorsC\_to\_B[READ]);

close(file\_descriptorsC\_to\_A[READ]);

char file\_descriptor\_read\_A[10];

char file\_descriptor\_write\_A[10];

char file\_descriptor\_write\_B[10];

sprintf(file\_descriptor\_read\_A, "%d", file\_descriptorsA\_to\_C[READ]);

sprintf(file\_descriptor\_write\_A, "%d", file\_descriptorsC\_to\_A[WRITE]);

sprintf(file\_descriptor\_write\_B, "%d", file\_descriptorsC\_to\_B[WRITE]);

char \*C\_argv[] = {"c", file\_descriptor\_read\_A, file\_descriptor\_write\_A, file\_descriptor\_write\_B, NULL};

if (execv("c", C\_argv) == -1) {

send\_error\_and\_stop("Exec C error\n", 4);

}

break;

default:

close(file\_descriptorsA\_to\_C[READ]);

close(file\_descriptorsA\_to\_B[READ]);

close(file\_descriptorsC\_to\_B[WRITE]);

close(file\_descriptorsC\_to\_B[READ]);

close(file\_descriptorsC\_to\_A[WRITE]);

char string[100];

while (1) {

if(gets(string) == NULL) {

send\_error\_and\_stop("Gets error\n", 5);

}

int length = strlen(string);

if(length == 0) {

if(write(file\_descriptorsA\_to\_C[WRITE], &length, sizeof(length)) == -1) {

send\_error\_and\_stop("Write in fdA\_to\_C error\n", 6);

}

if(write(file\_descriptorsA\_to\_B[WRITE], &length, sizeof(length)) == -1) {

send\_error\_and\_stop("Write in fdA\_to\_B error\n", 7);

}

break;

}

if(write(file\_descriptorsA\_to\_C[WRITE], &length, sizeof(length)) == -1) {

send\_error\_and\_stop("Write in fdA\_to\_C error\n", 6);

}

if(write(file\_descriptorsA\_to\_C[WRITE], &string, sizeof(char) \* length) == -1) {

send\_error\_and\_stop("Write in fdA\_to\_C error\n", 6);

}

if(write(file\_descriptorsA\_to\_B[WRITE], &length, sizeof(length)) == -1) {

send\_error\_and\_stop("Write in fdA\_to\_B error\n", 7);

}

int check = 0;

if(read(file\_descriptorsC\_to\_A[READ], &check, sizeof(check)) == -1) {

send\_error\_and\_stop("Read from fdA\_to\_C error\n", 8);

return 1;

}

}

break;

}

}

return 0;

}

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

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <fcntl.h>

#include <string.h>

#include <errno.h>

#define READ\_C 1

#define READ\_A 0

void send\_error\_and\_stop(char \*massage, int code) {

int i = strlen(massage);

write(STDERR\_FILENO, massage, sizeof(char) \* i);

exit(code);

}

int main(int argc, const char \*argv[]) {

int length;

int file\_descriptors[2];

file\_descriptors[READ\_A] = atoi(argv[1]);

file\_descriptors[READ\_C] = atoi(argv[2]);

while (1){

if (read(file\_descriptors[READ\_A], &length, sizeof(int)) == -1) {

send\_error\_and\_stop("Read from A error", 1);

}

if(length == 0) {

break;

}

printf("from A = %d\n", length);

if(read(file\_descriptors[READ\_C], &length, sizeof(int)) == -1) {

send\_error\_and\_stop("Read from C error", 2);

return 1;

}

printf("from C = %d\n", length);

}

return 0;

}

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

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <fcntl.h>

#include <string.h>

#include <errno.h>

#define WRITE\_B 2

#define WRITE\_A 1

#define READ\_A 0

void send\_error\_and\_stop(char \*massage, int code) {

int i = strlen(massage);

write(STDERR\_FILENO, massage, sizeof(char) \* i);

exit(code);

}

int main(int argc, const char \*argv[]) {

int file\_descriptors[3];

file\_descriptors[READ\_A] = atoi(argv[1]);

file\_descriptors[WRITE\_A] = atoi(argv[2]);

file\_descriptors[WRITE\_B] = atoi(argv[3]);

int length;

char string[100];

memset(string, 0, 100);

while (1) {

if(read(file\_descriptors[READ\_A], &length, sizeof(int)) == -1) {

send\_error\_and\_stop("Read from A error", 1);

}

if(length == 0) {

break;

}

if(read(file\_descriptors[READ\_A], &string, sizeof(char) \* length) == -1) {

send\_error\_and\_stop("Read from A error", 1);

}

printf("C: %s\n", string);

if(write(file\_descriptors[WRITE\_A], &length, sizeof(int)) == -1) {

send\_error\_and\_stop("Write to A error", 2);

}

if(write(file\_descriptors[WRITE\_B], &length, sizeof(int)) == -1) {

send\_error\_and\_stop("Write to B error", 3);

}

memset(string, 0, 100);

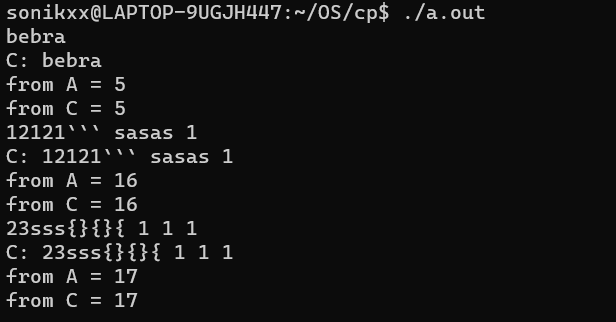
length = 0;

}

return 0;

}

**Пример работы**

****

**Вывод**

В ходе данного курсового проекта я попрактиковалась 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 (трубу) 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 закрепила свои знания и навыки, полученные во время прохождения курса операционных систем.