Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №2 по курсу**

**«Операционные системы»**

**Межпроцессное взаимодействие**

Студент: Шараковский Юрий Дмитриевич

Группа: М8О–206Б–20

Вариант: 21

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

* Управление процессами в ОС
* Обеспечение обмена данных между процессами посредством каналов

## Задание

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или каналы (pipe).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Вариант 21: Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы инвертируют строки.

**Общие сведения о программе**

Программа компилируется из файла main.c. Также используется заголовочные файлы: unistd.h, errno.h, signal.h, fcntl.h, sys/types.h, sys/stat.h, sys/syscall.h. В программе используются следующие системные вызовы:

1. **fork** – для создания дочерних процессов-копий родительского и дальнейшего запуска их кода.
2. **pipe** – для создания неименованного канала, используемого при общении с дочерними процессами (возвращает пару дескрипторов – в один из которых пишет родитель, а из другого читает ребенок.
3. **read** – системный вызов чтения n байт из файла по его дескриптору, используется в родительском процессе внутри реализации функции readString, а также используется дочерними процессами для чтения из канала.
4. **write** – системный вызов записи n байт в файл по его дескриптору, используется родительским процессом для записи в канал, также используется дочерними процессами для вывода в файл.
5. **open** –системный вызов открытия файла по его имени, возвращает дескриптор, используется для открытия текстовых файлов, по указанному пользователем имени, и записи в них дочерними процессами.
6. **close –** завершить текущие операции с файлом и закрыть его по указанному файловому дескриптору, используется для закрытия каналов всеми процессами и для закрытия выходного файла дочерними процессами.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы основных системных вызовов (fork, pipe, read, write, open, close).
2. Написать функцию считывания строки с потока ввода и функцию реверса строки (используя системные вызовы).
3. Написать логику подпрограммы родителя и ребенка: родитель в цикле считывает строки и отправляет их детям, ребенок в цикле считывает строки из канала, переворачивает их, и записывает в исходный файл (до возвращения значения EOF из вызова read).
4. Отладить программу и протестировать на основных тестах.

**Основные файлы программы**

**main.c**

#include <errno.h>

#include <signal.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/syscall.h>

#include <stdio.h>

#define BUFSIZE 256

#define SYSEOF 0

#define FAILURE -1

#define SUCCESS 0

#define TRUE 1

#define FALSE 0

int readString(int fd, char\* str, int maxsize) {

char ch;

int rbytes = 0;

int ret;

if ((ret = read(fd, &ch, 1)) < 1)

return SYSEOF;

while ((ch == ' ') || (ch == '\n') || (ch == '\t'))

if (read(fd, &ch, 1) < 1)

return SYSEOF;

int i = 0;

for (; i < maxsize - 1; i++) {

str[i] = ch;

rbytes++;

if (read(fd, &ch, 1) < 1)

return rbytes;

if (ch == '\r')

continue;

if (ch == '\n')

break;

}

str[++i] = '\0';

return rbytes;

}

static inline void reverseString(char\* dst, char\* src, int size) {

for (int i = size - 1; i >= 0; --i)

dst[size - i - 1] = src[i];

return;

}

int parent(int pipe1, int pipe2) {

char buf[BUFSIZE];

int size;

int fd[] = { pipe1, pipe2 };

int lever = 0;

int nbytes;

while ((size = readString(STDIN\_FILENO, buf, BUFSIZE)) != SYSEOF) {

if ((nbytes = write(fd[lever], &size, sizeof(size))) != sizeof(size))

return FAILURE;

int p = 0;

while ((nbytes = write(fd[lever], buf + p, size - p)) != size - p) {

p += nbytes;

if (p == size)

break;

if (nbytes < 1)

return FAILURE;

}

lever ^= 1;

}

return SUCCESS;

}

int child(int input, char filename[BUFSIZE]) {

int output = open(filename, O\_CREAT | O\_TRUNC | O\_WRONLY, S\_IRUSR | S\_IWUSR);

if (output < 0)

return FAILURE;

char buf[BUFSIZE];

char rev[BUFSIZE];

char nl[1];

nl[0] = '\n';

int size = 0;

int nbytes = 0;

while (TRUE) {

if ((nbytes = read(input, &size, sizeof(size))) != sizeof(size)) {

if (nbytes == SYSEOF)

break;

return FAILURE;

}

if (read(input, buf, size) != size)

return FAILURE;

reverseString(rev, buf, size);

if (write(output, rev, size) != size)

return FAILURE;

if (write(output, nl, sizeof(nl)) != sizeof(nl))

return FAILURE;

}

close(output);

return SUCCESS;

}

int main() {

int ret;

char file1[BUFSIZE], file2[BUFSIZE];

readString(STDIN\_FILENO, file1, BUFSIZE);

readString(STDIN\_FILENO, file2, BUFSIZE);

int pipe1[2], pipe2[2];

if (pipe(pipe1) != 0)

return FAILURE;

if (pipe(pipe2) != 0)

return FAILURE;

int pid1 = fork();

if (pid1 < 0) {

return FAILURE;

}

else if (pid1 == 0) {

close(pipe2[1]);

close(pipe2[0]);

close(pipe1[1]);

ret = child(pipe1[0], file1);

close(pipe1[0]);

}

else {

int pid2 = fork();

if (pid2 < 0) {

return FAILURE;

}

else if (pid2 == 0) {

close(pipe1[0]);

close(pipe1[1]);

close(pipe2[1]);

ret = child(pipe2[0], file2);

close(pipe2[0]);

}

else {

close(pipe1[0]);

close(pipe2[0]);

ret = parent(pipe1[1], pipe2[1]);

close(pipe1[1]);

close(pipe2[1]);

}

}

return ret;

}

**Пример работы**

$ make

gcc main.c -o main

$ cat testfile02

t02\_f1

t02\_f2

test2: fprintf fwrite fputs fputc

test2: fscanf fread fgets fgetc

test2: ungetcyuroksov@LAPTOP-

$ ./main < testfile02

$ cat t02\_f1

ctupf stupf etirwf ftnirpf :2tset

ctegnu :2tset

$ cat t02\_f2

ctegf stegf daerf fnacsf :2tset

$ ./main

f1

f2

test1 reversed

test2 asdasdasdasda

asdasdasdaa

dsdsaaaa

beginasdasdasdasdasdasdend

$ cat ./f1

desrever 1tset

aadsadsadsa

dnedsadsadsadsadsadsanigeb

$ cat ./f2

adsadsadsadsa 2tset

aaaasdsd

**Репозиторий**

https://github.com/YurokSov/os\_lab\_2

**Вывод**

Выполнив эту лабораторную работу, я приобрел навыки создания простых многопроцессных программ, и применил на практике такой способ межпроцессного взаимодействия, как каналы (pipe). Каналы удобно применять в тех случаях, когда некоторая дочерняя группа процессов требует ввода данных из родительского процесса. Однако такое общение может быть и двухсторонним.