Московский авиационный институт

(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Операционные системы»

**Межпроцессорное взаимодействие**

Студент: Сафуанов Р. А.

Преподаватель: Соколов А. А.

Группа: М8О-206Б-20

Москва, 2022

**Постановка задачи**

Цель работы:

Приобретение практических навыков в:

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

Задание (вариант 18):

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 (основной̆ 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.

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 второй̆ строки и процесса child2. родительский̆ и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной̆ 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 удаляют все гласные из строк.

**Общие сведения о программе**

Программа компилируется из файла main.c. Подключены заголовочные файлы: unistd.h, fcntl.h, stdlib.h. В программе используются следующие системные вызовы:

1. pipe –– принимает массив из двух целых чисел, в случае успеха массив будет содержать два файловых дескриптора, которые будут использоваться для конвейера, первое число в массиве предназначено для чтения, второе для записи, а также вернется 0. В случае неуспеха вернется -1.
2. fork –– создает новый процесс, который является копией родительского процесса, за исключением разных process ID и parent process ID. В случае успеха fork() возвращает 0 для ребенка, число больше 0 для родителя – child ID, в случае ошибки возвращает -1.
3. open –– создает или открывает файл, если он был создан. В качестве аргументов принимает путь до файла, режим доступа (запись, чтение и т.п.), модификатор доступа (при создании можно указать права для файла). Возвращает в случае успеха файловый дескриптор – положительное число, иначе возвращает -1.
4. close –– принимает файловый дескриптор в качестве аргумента, удаляет файловый дескриптор из таблицы дескрипторов, в случае успеха вернет 0, в случае неуспеха вернет -1.
5. read –– предназначена для чтения какого-то числа байт из файла, принимает в качестве аргументов файловый дескриптор, буфер, в который будут записаны данные и число байт. В случае успеха вернет число прочитанных байт, иначе -1.
6. write –– предназначена для записи какого-то числа байт в файл, принимает в качестве аргументов файловый дескриптор, буфер, из которого будут считаны данные для записи и число байт. В случае успеха вернет число записанных байт, иначе -1.

**Общий метод и алгоритм решения**

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы pipe и fork.  
2. Написать функцию считывания имён выходных файлов  
3. Создать каналы связи для каждого из дочерних процессов 4. Создать функцию обработки ввода  
5. Создать функцию фильтрации в родительском процессе  
6. Создать функцию фильтрации в дочерних процессах  
7. Написать обработку ошибок  
8. Написать тесты

**Исходный код**

main.c

#include <fcntl.h>

#include <stdlib.h>

#include <unistd.h>

void child\_work(int from, int to) {

char buf[1];

while (read(from, buf, 1) > 0) {

char c = buf[0];

if (c != ’a’ && c != ’e’ && c != ’i’ && c != ’o’ && c != ’u’ && c != ’y’ &&

c != ’A’ && c != ’E’ && c != ’I’ && c != ’O’ && c != ’U’ && c != ’Y’) {

write(to, buf, 1);

}

}

close(to);

close(from);

}

void parrent\_work(int child1, int child2) {

char buf[1];

int is\_even = 0;

while (read(STDIN\_FILENO, buf, 1) > 0) {

if (!is\_even) {

write(child1, buf, 1);

} else {

write(child2, buf, 1);

}

if (buf[0] == ’\n’) {

is\_even = !is\_even;

}

}

close(child1);

close(child2);

}

int read\_name\_and\_open\_file() {

const size\_t FILE\_NAME\_SIZE = 64;

char f\_name[FILE\_NAME\_SIZE];

char buf[1];

int idx = 0;

while (idx < FILE\_NAME\_SIZE && read(STDIN\_FILENO, buf, 1) > 0) {

if (buf[0] == ’\n’) {

break;

}

f\_name[idx++] = buf[0];

}

f\_name[idx++] = ’\0’;

return open(f\_name, O\_WRONLY | O\_TRUNC);

}

void error(char\* buf, size\_t size) { write(STDERR\_FILENO, buf, size); }

void check\_file\_id(int id) {

if (id == -1) {

error("File not found\n", 15);

exit(-1);

} }

void check\_pipe\_creation(int\* pipefd) {

if (pipe(pipefd) == -1) {

error("Cannot create pipe\n", 19);

exit(-2); }

}

int check\_fork() {

int fd = fork();

if (fd == -1) {

error("Cannot create process\n", 22);

exit(-3); }

return fd; }

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

int f1 = read\_name\_and\_open\_file();

check\_file\_id(f1);

int f2 = read\_name\_and\_open\_file();

check\_file\_id(f2);

int pipefd1[2];

check\_pipe\_creation(pipefd1);

int child1 = check\_fork();

if (child1 == 0) {

close(pipefd1[1]);

child\_work(pipefd1[0], f1);

return 0;

}

close(pipefd1[0]);

int pipefd2[2];

check\_pipe\_creation(pipefd2);

int child2 = check\_fork();

if (child2 == 0) {

close(pipefd1[1]);

close(pipefd2[1]);

child\_work(pipefd2[0], f2);

return 0;

}  
close(pipefd2[0]);

parrent\_work(pipefd1[1], pipefd2[1]);

return 0;

}

**Пример работы**

Первый тест - проверка на обработку случая отсутсвия первого файла

cat test1

f

./a.out <test1

File not found

Второй тест - проверка на обработку случая отсутсвия второго файла

cat test2

f1

f

./a.out <test2

File not found

Третий и четвёртый тесты - проверка работоспособности программы на корректных данных

cat test3  
f1  
f2  
a b c d e f g h i j k l

m n o p q r s t u v w x y z

a.out <test3

cat f1

b c d f g h j k l

cat f2

m n p q r s t v w x z

cat test4

f1  
f2

m n o p q r s t u v w x y z

a b c d e f g h i j k l

m n o p q r s t u v w x y z

a b c d e f g h i j k l

m n o p q r s t u v w x y z

a b c d e f g h i j k l

m n o p q r s t u v w x y z

a b c d e f g h I j k l

./a.out <test4

cat f1

m n p q r s t v w x z

m n p q r s t v w x z

m n p q r s t v w x z

m n p q r s t v w x z

cat f2

b c d f g h j k l

b c d f g h j k l

b c d f g h j k l

b c d f g h j k l

Пятый тест - проверка работоспособности программы на пустых данных

cat test5

f1

f2

./a.out <test5

cat f1

cat f2

**Вывод**

В процессе работы над лабораторной я научился основам работы с конвейерами и процессами в Си. Процессы занимают важную роль в разработке ПО, так как программы зачастую состоят из нескольких, относительно обособленных, подпрограмм, то есть процессов. Конвейеры как один из способов обмена данными между процессами также играют немаловажную роль.