Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №1 по курсу**

**«Операционные системы»**

**ВЗАИМОДЕЙСТВИЕ МЕЖДУ ПРОЦЕССАМИ**

Студент: Сектименко Ирина Владимировна

Группа: М8О–210Б–22

Вариант: 21

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

* управлении процессами в ОС;
* обеспечении обмена данных между процессами посредством каналов.

## Задание

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

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, если они нечетные, и в pipe2, если четные. Процессы child1 и child2 инвертируют строки и пишут результаты своей работы в стандартный вывод.

## Общие сведения о программе

Программа скомпилируется из трех файлов parent.cpp, son1.cpp, son2.cpp. В программе используются следующие системные вызовы:

1. **pid\_t fork()** – создание дочернего процесса.
2. **int execl(const char \*filename, char \*const argv[], char \*const envp[])** – замена образа памяти процесса.
3. **int pipe(int pipefd[2])** – создание неименованного канала для передачи данных между процессами.
4. **int dup2(int oldfd, int newfd)** – переназначение файлового дескриптора.
5. **int open(const char \*pathname, int flags, mode\_t mode)** – открытие/создание файла.
6. **int close(int fd)** – закрыть файл.

## Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, execl, pipe, dup2, open, close.
2. Написать программу родительского процесса parent.cpp, которая:
   1. создает два дочерних процесса командой fork;
   2. принимает строки из стандартного потока ввода и пересылает их в pipe1 или pipe2.
3. Написать программы дочерних процессов son1.cpp и son2.cpp, которые будут инвертировать строки.
4. В файле родительского процесса вызывать дочерние и запускать их программы командой execl.

## Основные файлы программы

**parent.cpp:**

#include "stdio.h"

#include "iostream"

#include "stdlib.h"

#include "unistd.h"

#include "sys/wait.h"

#include "string"

#include "fcntl.h"

int main() {

std::string fileName11 = "", fileName22 = "";

char c;

while (true) {

read(STDIN\_FILENO, &c, sizeof(char));

if (c == '\n' || c == ' ') {

break;

}

fileName11 += c;

}

char fileName1[fileName11.size() + 1];

for (int i = 0; i < fileName11.size(); ++i) {

fileName1[i] = fileName11[i];

}

fileName1[fileName11.size()] = '\0';

while (true) {

read(STDIN\_FILENO, &c, sizeof(char));

if (c == '\n' || c == ' ') {

break;

}

fileName22 += c;

}

char fileName2[fileName22.size() + 1];

for (int i = 0; i < fileName22.size(); ++i) {

fileName2[i] = fileName22[i];

}

fileName2[fileName22.size()] = '\0';

int pipe1\_fd[2]; // 0 - read, 1 - write

int err1 = pipe(pipe1\_fd);

if (err1 == -1) {

perror("pipe");

return -1;

}

int pipe2\_fd[2];

int err2 = pipe(pipe2\_fd);

if (err2 == -1) {

perror("pipe");

return -1;

}

pid\_t pid = fork();

if (pid == -1) {

perror("fork1");

return -1;

}

pid\_t pid2;

if (pid > 0) {

pid2 = fork();

if (pid2 == -1) {

perror("fork2");

return -1;

} else if (pid2 == 0) {

pid = 1;

}

}

if (pid == 0) {

pid2 = 1;

}

std::string s, s1, s2;

char c1, c2;

if (pid > 0 && pid2 > 0) {

while (read(STDIN\_FILENO, &c, sizeof(c))) {

if (c == ' ' || c == '\n' || c == '\t') {

s += '\n';

if (s.size() % 2 == 0) {

s1 += s;

} else {

s2 += s;

}

s = "";

continue;

}

s += c;

}

close(pipe1\_fd[0]);

close(pipe2\_fd[0]);

write(pipe1\_fd[1], s1.c\_str(), s1.length());

write(pipe2\_fd[1], s2.c\_str(), s2.length());

close(pipe1\_fd[1]);

close(pipe2\_fd[1]);

}

if (pid == 0) {

close(pipe1\_fd[1]);

dup2(pipe1\_fd[0], STDIN\_FILENO);

close(pipe1\_fd[0]);

execl("son1.out", "son1.out", fileName1, NULL);

} else if (pid2 == 0) {

close(pipe2\_fd[1]);

dup2(pipe2\_fd[0], STDIN\_FILENO);

close(pipe2\_fd[0]);

execl("son2.out", "son2.out", fileName2, NULL);

}

return 0;

}

**son1.cpp:**

#include "stdio.h"

#include "iostream"

#include "stdlib.h"

#include "unistd.h"

#include "sys/wait.h"

#include "string"

#include "fcntl.h"

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

int file;

file = open(argv[1], O\_WRONLY | O\_CREAT | O\_APPEND, S\_IRWXU);

char c;

while (read(STDIN\_FILENO, &c, sizeof(c)) > 0) {

write(file, &c, sizeof(c));

}

close(file);

return 0;

}

**son2.cpp:**

#include "stdio.h"

#include "iostream"

#include "stdlib.h"

#include "unistd.h"

#include "sys/wait.h"

#include "string"

#include "fcntl.h"

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

int file;

file = open(argv[1], O\_WRONLY | O\_CREAT | O\_APPEND, S\_IRWXU);

char c;

while (read(STDIN\_FILENO, &c, sizeof(c)) > 0) {

write(file, &c, sizeof(c));

}

close(file);

return 0;

}

## Пример работы

В консоль вводится:

text1.txt

text2.txt

asd as

zxc a zxcvb

asdfg

qwerty

В файле text1.txt будет: asd

zxc

a

zxcvb

asdfg

В файле text2.txt:

as

qwerty

## Вывод

Данная лабораторная работа позволила более детально разобраться в создании и работе дочерних процессов (например, при использовании двух системных вызовов fork подряд получается родительский процесс, два дочерних процесса и один дочерний процесс дочернего процесса, о создании которого можно не сразу понять), «общении» между ними с помощью неименованных каналов pipe и во включении другой программы в работу текущей. Были изучены принципы работы сложных функций printf и scanf, в основе которых лежат функции write и read соответсвенно.