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: Эсмедляев Федор Романович

Группа: М8О–212Б–22

Вариант: 22

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 данными 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.

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.

Вариант 22) Правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Дочерние процессы инвертируют строки.

**Общие сведения о программе**

Программа компилируется из файла main.cpp. Помимо этого, есть программа child.cpp(по заданию их в теории должно быть 2, но они буквально одинаковые, поэтому их можно сократить в 1), которая запускается при помощи exec. Обе программы собираются при помощи системы сборки CMake. В программе используются следующие системные вызовы:

1. **fork** - создает дочерний процесс, повторяющий программу родительского.
2. **exec** - позволяет запустить исполняемый файл изнутри программы.
3. **pipe** - создает пайп для передачи данных между процессами
4. **dup2** - переназначение файлового дескриптора
5. **open** - открытие файла
6. **сlose** - закрытие файла

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы fork, exec, pipe.
2. Написать основную программу процесса main.cpp, которая будет осуществлять вызов системных функций.
3. Написать программу child.cpp, которая будет запускаться из дочернего процесса при помощи exec

**Основные файлы программы**

**main.cpp:**

#include <bits/stdc++.h>

#include <unistd.h>

#include <fstream>

#include <fcntl.h>

#include <random>

#include <chrono>

using namespace std;

mt19937 rng(chrono::steady\_clock::now().time\_since\_epoch().count());

char \*read\_string(size\_t &string\_len)

{

string\_len = 0;

size\_t string\_cap = 1;

char \*string = new char[string\_cap];

char c;

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

if (c == '\n')

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

if (c == EOF)

return nullptr;

while (c != '\n')

{

string[string\_len] = c;

string\_len++;

if (string\_len >= string\_cap)

{

string\_cap \*= 2;

char \*new\_string = new char[string\_cap];

for (size\_t i = 0; i < string\_len; i++)

{

new\_string[i] = string[i];

}

delete[] string;

string = new\_string;

}

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

if (c == EOF)

return nullptr;

}

string[string\_len] = '\0';

return string;

}

int main()

{

string file\_name\_1;

char symbol\_1;

if (read(STDIN\_FILENO, &symbol\_1, sizeof(char)) == -1)

{

return 7;

}

while (symbol\_1 != '\n')

{

file\_name\_1 += symbol\_1;

if (read(STDIN\_FILENO, &symbol\_1, sizeof(char)) == -1)

{

return 7;

}

}

int file\_1 = open(file\_name\_1.c\_str(), O\_WRONLY | O\_CREAT, 0777);

if (file\_1 == -1)

{

return -1;

}

string file\_name\_2;

char symbol\_2;

if (read(STDIN\_FILENO, &symbol\_2, sizeof(char)) == -1)

{

return 7;

}

while (symbol\_2 != '\n')

{

file\_name\_2 += symbol\_2;

if (read(STDIN\_FILENO, &symbol\_2, sizeof(char)) == -1)

{

return 7;

}

}

int file\_2 = open(file\_name\_2.c\_str(), O\_WRONLY | O\_CREAT, 0777);

if (file\_2 == -1)

{

return -1;

}

int fd\_1[2]; // fd[0] - read; fd[1] - write

// Creating pipe and checking that it was created

if (pipe(fd\_1) == -1)

{

return 1;

}

int fd\_2[2]; // fd[0] - read; fd[1] - write

// Creating pipe and checking that it was created

if (pipe(fd\_2) == -1)

{

return 1;

}

int id\_1 = fork();

if (id\_1 == -1) // fork error

{

return 2;

}

else if (id\_1 > 0) // in 1 process

{

// close(fd\_1[0]);

int id\_2 = fork();

if (id\_2 == -1) // fork error

{

return 2;

}

else if (id\_2 > 0)

{

size\_t len = 0;

char \*stroka = read\_string(len);

char p = 's'; // start

while (stroka != nullptr)

{

if (rng() % 100 <= 80)

{

write(fd\_1[1], &p, 1);

write(fd\_1[1], &len, sizeof(len));

write(fd\_1[1], stroka, sizeof(char) \* len);

}

else

{

write(fd\_2[1], &p, 1);

write(fd\_2[1], &len, sizeof(len));

write(fd\_2[1], stroka, sizeof(char) \* len);

}

stroka = read\_string(len);

}

p = 'e'; // end

write(fd\_1[1], &p, 1);

write(fd\_2[1], &p, 1);

close(file\_1);

close(file\_2);

close(fd\_1[1]);

close(fd\_1[0]);

close(fd\_2[1]);

close(fd\_2[0]);

}

else

{

close(fd\_2[1]);

close(fd\_1[1]);

close(fd\_1[0]);

close(file\_1);

if (dup2(file\_2, STDOUT\_FILENO) == -1)

{

return -1;

}

close(file\_2);

if (dup2(fd\_2[0], STDIN\_FILENO) == -1)

{

return -1;

}

close(fd\_2[0]);

execl("./son.out", NULL);

return 0;

}

}

else

{

close(fd\_1[1]);

close(fd\_2[0]);

close(fd\_2[1]);

close(file\_2);

if (dup2(file\_1, STDOUT\_FILENO) == -1)

{

return -1;

}

close(file\_1);

if (dup2(fd\_1[0], STDIN\_FILENO) == -1)

{

return -1;

}

close(fd\_1[0]);

execl("./son.out", NULL);

return 0;

}

return 0;

}

**child.cpp:**

#include <iostream>

#include <cstring>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

void reverse\_str(char str[], size\_t length)

{

char tempo;

for (size\_t i = 0; i < length; i++)

{

tempo = str[length - 1];

str[length - 1] = str[i];

str[i] = tempo;

length--;

}

}

int main()

{

char a = ' ';

size\_t s\_len;

read(0, &a, sizeof(char));

while (a != 'e')

{

read(0, &s\_len, sizeof(s\_len));

char string[s\_len];

read(0, &string, sizeof(char) \* s\_len);

reverse\_str(string, s\_len);

write(1, string, sizeof(char) \* s\_len);

write(1, "\n", 1);

read(0, &a, sizeof(char));

}

return 0;

}

**CMakeLists.txt:**

cmake\_minimum\_required(VERSION 3.10)

project(X)

set(CMAKE\_CXX\_STANDARD 17)

add\_executable(main main.cpp)

add\_executable(son.out child.cpp)

**Пример работы**

1. **Ввод:**

a1.txt

a2.txt

abc

test

gicha

**a1.txt:**

cba

tset

**a2.txt:**

ahcig

1. **Ввод одной строки:**

**a1.txt:**

hsalahs

**a2.txt:**

1. **Ввод(так получилось, это 80%):**

a1.txt

a2.txt

kubik

rubika

sobirayu

za

28

secund

ez

ez

**a1.txt:**

kibuk

akibur

uyaribos

az

82

dnuces

ze

ze

**a2.txt:**

**Вывод**

Мной изучены основные системные вызовы для работы с процессами и обмена данными между ними. Моя программа поочередно открывает/создает файлы, которые вводит пользователь, а после последовательно считывает строки и с вероятностью 80% распределяет их и через pipe1 и pipe2 передает их в дочернюю программу. Кстати, есть смысл использовать функцию рандома, как у меня, поскольку обычную функцию можно теоретически подгадать и взломать. Далее просто через dup2 просто печатаем и получаем результат.