МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 "Компьютерные науки и прикладная математика"

Кафедра 806 "Вычислительная математика и программирование"

Лабораторная работа №1

По курсу «Операционные системы»

Студент: Снетков Н.С.

Группа: М8О-203Б-23

Вариант: 15

Преподаватель: Миронов Е. С.

Дата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2024

**Содержание**

1. Репозиторий

2. Постановка задачи

3. Общие сведения о программе

4. Общий метод и алгоритм решения

5. Исходный код

6. Сборка программы

7. Демонстрация работы программы

8. Выводы

**Репозиторий**

https://github.com/mxdesta/osLabs/tree/main/lab4

**Постановка задачи**

**Цель работы**

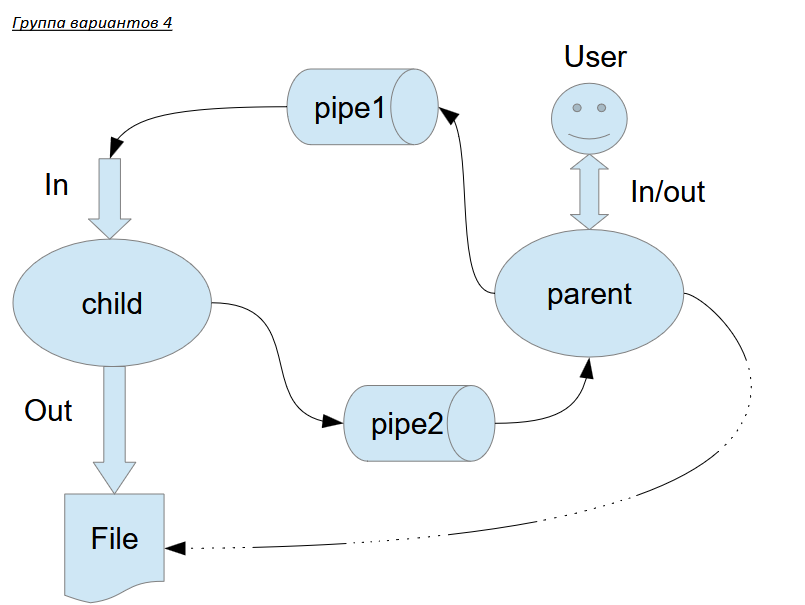
Приобретение практических навыков в:

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.



Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами.Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода.

Вариант 15) Правило проверки: строка должна начинаться с заглавной буквы

**Общие сведения о программе**

Программа компилируется из файла main.cpp. В программе используются следующие системные вызовы:

1. pipe() - существует для передачи информации между различными процессами.

2. fork() - создает новый процесс.

3. execl() - передает процесс на исполнение другой программе.

4. read() - читает данные из файла.

5. write() - записывает данные в файл.

6. close() - закрывает файл.

**Общий метод и алгоритм решения**

1. Родительский процесс (parent.cpp)

Родительский процесс выполняет следующие шаги:

Создание каналов (pipes):

Создаются два канала: pipe1 и pipe2. pipe1 будет использоваться для передачи данных от родительского процесса к дочернему, а pipe2 — для передачи данных от дочернего процесса к родительскому.

Создание дочернего процесса:

Родительский процесс создает дочерний процесс с помощью системного вызова fork(). В дочернем процессе выполняется программа child, которая обрабатывает данные, переданные через канал.

Передача данных в дочерний процесс:

Родительский процесс запрашивает у пользователя имя файла, в который будут записываться результаты, и строки текста. Каждая строка передается в дочерний процесс через канал pipe1.

Получение данных от дочернего процесса:

Родительский процесс читает данные из канала pipe2, который используется для получения результатов обработки от дочернего процесса. Если дочерний процесс возвращает ошибку (строка начинается с "Error:"), она выводится на экран. В противном случае результат записывается в файл.

Завершение работы:

После завершения ввода данных (пользователь вводит "exit"), родительский процесс закрывает каналы и ожидает завершения дочернего процесса с помощью wait().

2. Дочерний процесс (child.cpp)

Дочерний процесс выполняет следующие шаги:

Получение дескрипторов каналов:

Дочерний процесс получает дескрипторы каналов pipe1 и pipe2 через аргументы командной строки. pipe1 используется для чтения данных от родительского процесса, а pipe2 — для отправки результатов обратно.

Чтение данных из канала:

Дочерний процесс читает данные из канала pipe1. Каждая строка проверяется на то, начинается ли она с заглавной буквы.

Проверка строки:

Если строка начинается с заглавной буквы, она отправляется обратно в родительский процесс через канал pipe2.

Если строка не начинается с заглавной буквы, дочерний процесс отправляет сообщение об ошибке в родительский процесс.

Завершение работы:

Дочерний процесс продолжает чтение и обработку данных до тех пор, пока родительский процесс не закроет канал. После этого дочерний процесс закрывает свои каналы и завершает работу.

3. Взаимодействие между процессами

Родительский процесс передает строки текста в дочерний процесс через канал pipe1.

Дочерний процесс проверяет каждую строку и отправляет результат проверки (либо саму строку, либо сообщение об ошибке) обратно в родительский процесс через канал pipe2.

Родительский процесс записывает результаты в файл или выводит сообщения об ошибке на экран.

**Исходный код**

**common.h:**

#ifndef COMMON\_H

#define COMMON\_H

#include <cstddef>

#include <csignal>

extern const char\* SHARED\_FILE;

extern const size\_t BUFFER\_SIZE;

extern volatile sig\_atomic\_t child\_ready;

#endif // COMMON\_H

**parent.h:**

#ifndef PARENT\_H

#define PARENT\_H

#include <string>

void processInput(const std::string& filename, const std::string& input);

void parentSignalHandler(int sig);

void handleError(const std::string& msg);

#endif

**child.cpp:**

#include "../include/common.h"

#include <iostream>

#include <sys/mman.h>

#include <fcntl.h>

#include <unistd.h>

#include <csignal>

#include <cstring>

#include <cctype>

#include <string>

void handleError(const std::string& msg) {

perror(msg.c\_str());

exit(EXIT\_FAILURE);

}

int main() {

// Подключение к разделяемой памяти

int fd = shm\_open(SHARED\_FILE, O\_RDWR, 0666);

if (fd == -1) handleError("shm\_open failed");

void\* shared\_mem = mmap(0, BUFFER\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (shared\_mem == MAP\_FAILED) handleError("mmap failed");

close(fd);

char\* shared\_data = static\_cast<char\*>(shared\_mem);

signal(SIGUSR1, [](int) {});

while (true) {

pause();

if (std::string(shared\_data) == "/exit") {

break;

}

}

munmap(shared\_mem, BUFFER\_SIZE);

shm\_unlink(SHARED\_FILE);

return 0;

}

**common.cpp**

#include "../include/common.h"

const char\* SHARED\_FILE = "/shared\_memory";

const size\_t BUFFER\_SIZE = 1024;

volatilesig\_atomic\_t child\_ready=0;

**parent.cpp:**

#include <iostream>

#include <fstream>

#include <sys/mman.h>

#include <fcntl.h>

#include <unistd.h>

#include <csignal>

#include <cstring>

#include <string>

#include <sys/wait.h>

#include "../include/common.h"

#include "../include/parent.h"

// Обработчик ошибок

void handleError(const std::string& msg) {

perror(msg.c\_str());

exit(EXIT\_FAILURE);

}

// Обработчик сигналов для родителя

void parentSignalHandler(int sig) {

if (sig == SIGUSR1) {

child\_ready = 1;

}

}

bool isValid(const std::string& str) {

return !str.empty() && std::isupper(str[0]);

}

void processInput(const std::string& filename, const std::string& input) {

static pid\_t child\_pid = -1;

if (child\_pid == -1) {

int fd = shm\_open(SHARED\_FILE, O\_CREAT | O\_RDWR, 0666);

if (fd == -1) handleError("shm\_open failed");

if (ftruncate(fd, BUFFER\_SIZE) == -1) handleError("ftruncate failed");

void\* shared\_mem = mmap(0, BUFFER\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (shared\_mem == MAP\_FAILED) handleError("mmap failed");

close(fd);

child\_pid = fork();

if (child\_pid < 0) {

handleError("fork failed");

} else if (child\_pid == 0) {

execl("/home/unix/labs/osLabs/build/lab4/child3", "/home/unix/labs/osLabs/build/lab4/child3", nullptr);

handleError("execl failed");

}

signal(SIGUSR1, parentSignalHandler);

}

if (input == "/exit") {

if (child\_pid > 0) {

int fd = shm\_open(SHARED\_FILE, O\_RDWR, 0666);

if (fd == -1) handleError("shm\_open failed");

void\* shared\_mem = mmap(0, BUFFER\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (shared\_mem == MAP\_FAILED) handleError("mmap failed");

char\* shared\_data = static\_cast<char\*>(shared\_mem);

strncpy(shared\_data, "/exit", BUFFER\_SIZE);

munmap(shared\_mem, BUFFER\_SIZE);

kill(child\_pid, SIGUSR1);

waitpid(child\_pid, nullptr, 0); // Ожидаем завершения дочернего процесса

shm\_unlink(SHARED\_FILE); // Удаляем разделяемую память

}

exit(0);

}

std::string result;

if (isValid(input)) {

result = input;

} else {

result = "Error: строка должна начинаться с заглавной буквы";

}

std::ofstream output\_file(filename, std::ios::app);

if (!output\_file.is\_open()) {

handleError("Failed to open file");

}

if (result.rfind("Error:", 0) == 0) {

std::cerr << result << std::endl;

} else {

output\_file << result << std::endl; // Записываем только корректные строки

}

output\_file.close();

}

**main.c:**  
  
#include "include/parent.h"

#include <iostream>

#include <string>

int main() {

std::string filename = "output.txt";

std::string input;

while (true) {

std::cout << "Введите строку (/exit для выхода): ";

std::getline(std::cin, input);

processInput(filename, input);

if (input == "/exit") {

break;

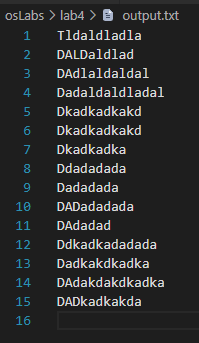
}

}

return 0;

}

**Демонстрация работы программы**



**Выводы**

Программа демонстрирует использование разделяемой памяти и сигналов для межпроцессного взаимодействия. Реализована проверка корректности ввода данных (строка должна начинаться с заглавной буквы). Программа корректно завершает работу, освобождая все ресурсы (память, файловые дескрипторы). Приложение может быть расширено для выполнения более сложных задач, таких как обработка больших объемов данных или параллельные вычисления.