Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

Студент: Соколов Арсений Игоревич

Группа: М8О-207Б-21

Вариант: 17

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/valerasaray/os

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в управлении 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).

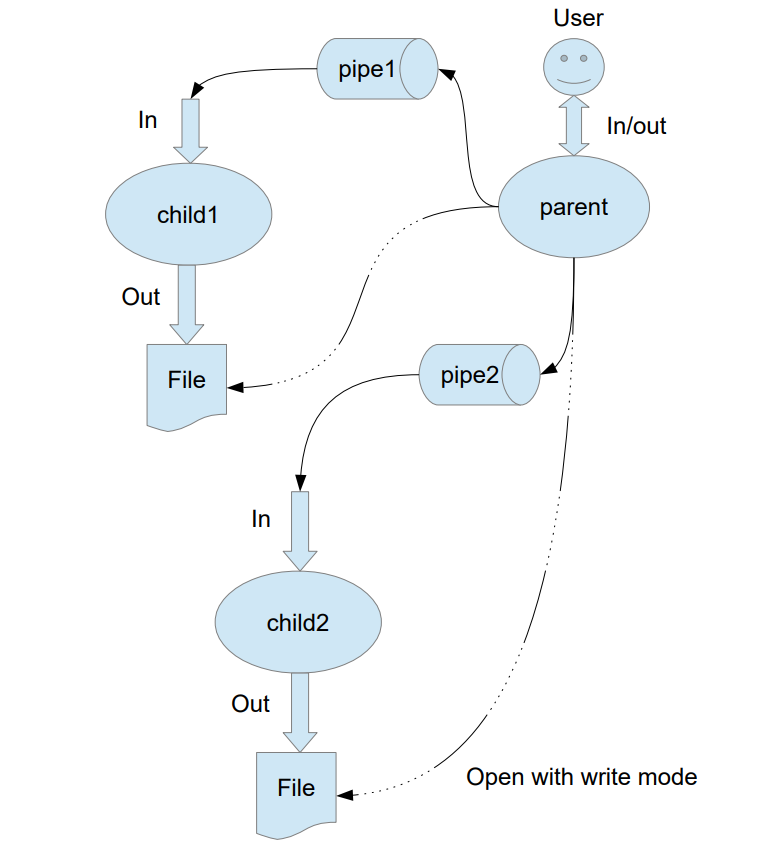
Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

**Группа вариантов 5**

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 17**

Правило фильтрации: строки длины больше 10 символов отправляются в pipe2, иначе в pipe1. Дочерние процессы удаляют все гласные из строк.



**Общие сведения о программе**

Программа компилируется из файла main.cpp, child.cpp.

**Общий метод и алгоритм решения**

Пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на запись для дочернего процесса. Аналогично для второй строки и второго дочернего процесса. Родительский процесс принимает от пользователя строки произвольной длины и пересылает строки длины больше 10 символов в pipe2, иначе в pipe1. Дочерние процессы удаляют все гласные из строк и пишут результаты своей работы в стандартный вывод. Управление доступам к общим ресурсам между несколькими ресурсами реализуем через механизм синхронизации с использованием семафора, изменяя его значение.

**Исходный код**

**main.cpp**

**#include <iostream>**

**#include <string>**

**#include <mutex>**

**#include <algorithm>**

**#include <unistd.h>**

**#include <sys/mman.h>**

**#include <sys/stat.h>**

**#include <semaphore.h>**

**#include <fcntl.h>**

**#include <errno.h>**

**#include <fstream>**

**#include <string.h>**

**using namespace std;**

**int main()**

**{**

**string current\_str;**

**int child\_tag;**

**fstream res\_file;**

**string child1, child2;**

**cout << "Введите имя для первого дочернего файла: ";**

**cin >> child1;**

**cout << "Введите имя для второго дочернего файла: ";**

**cin >> child2;**

**shm\_unlink("1.back");**

**shm\_unlink("2.back");**

**sem\_unlink("\_sem1");**

**sem\_unlink("\_sem2");**

**sem\_t \*sem1 = sem\_open("\_sem1", O\_CREAT, S\_IWUSR | S\_IRUSR | S\_IRGRP | S\_IROTH, 0);**

**sem\_t \*sem2 = sem\_open("\_sem2", O\_CREAT, S\_IWUSR | S\_IRUSR | S\_IRGRP | S\_IROTH, 0);**

**int state = 0;**

**std::mutex mx;**

**mx.lock();**

**while (++state < 2) {**

**sem\_post(sem1);**

**}**

**while (--state > 1) {**

**sem\_wait(sem1);**

**}**

**mx.unlock();**

**state = 0;**

**mx.lock();**

**while (++state < 2) {**

**sem\_post(sem2);**

**}**

**while (--state > 1) {**

**sem\_wait(sem2);**

**}**

**mx.unlock();**

**pid\_t f\_id1 = fork();**

**if (f\_id1 == -1)**

**{**

**cout << "Ошибка fork с кодом -1, возвращенным в родительском процессе, child1 не создан" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**else if (f\_id1 == 0)**

**{**

**sem\_close(sem1);**

**string child = child1;**

**execlp("./child", child.c\_str(), "1.back", "\_sem1", NULL);**

**perror("Ошибка execlp");**

**}**

**pid\_t f\_id2 = fork();**

**if (f\_id2 == -1)**

**{**

**cout << "Ошибка fork с кодом -1, возвращенным в родительском процессе, child2 не создан" << endl;**

**exit(EXIT\_FAILURE);**

**}**

**else if (f\_id2 == 0)**

**{**

**sem\_close(sem2);**

**string child = child2;**

**execlp("./child", child.c\_str(), "2.back", "\_sem2", NULL);**

**perror("Ошибка execlp");**

**}**

**else**

**{**

**while (getline(std::cin, current\_str))**

**{**

**int s\_size = current\_str.size() + 1;**

**char \*buffer = (char \*) current\_str.c\_str();**

**if (current\_str.size() <= 10)**

**{**

**int fd = shm\_open("1.back", O\_RDWR | O\_CREAT, S\_IWUSR | S\_IRUSR | S\_IRGRP | S\_IROTH);**

**ftruncate(fd, s\_size);**

**char \*mapped = (char \*) mmap(NULL, s\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);**

**memset(mapped, '\0', s\_size);**

**sprintf(mapped, "%s", buffer);**

**munmap(mapped, s\_size);**

**close(fd);**

**sem\_wait(sem1);**

**}**

**else**

**{**

**int fd = shm\_open("2.back", O\_RDWR | O\_CREAT, S\_IWUSR | S\_IRUSR | S\_IRGRP | S\_IROTH);**

**ftruncate(fd, s\_size);**

**char \*mapped = (char \*) mmap(NULL, s\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);**

**memset(mapped, '\0', s\_size);**

**sprintf(mapped, "%s", buffer);**

**munmap(mapped, s\_size);**

**close(fd);**

**sem\_wait(sem2);**

**}**

**}**

**}**

**return 0;**

**}**

**child.cpp**

**#include <iostream>**

**#include <string>**

**#include <algorithm>**

**#include <unistd.h>**

**#include <sys/mman.h>**

**#include <sys/stat.h>**

**#include <semaphore.h>**

**#include <fcntl.h>**

**#include <errno.h>**

**#include <string.h>**

**#include <fstream>**

**#include <set>**

**using namespace std;**

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

**{**

**char \*semFile = (char \*) *argv*[2];**

**sem\_t \*sem = sem\_open(semFile, O\_CREAT, S\_IWUSR | S\_IRUSR | S\_IRGRP | S\_IROTH, 0);**

**std::string vovels = "aoueiy";**

**std::set<char> volSet(vovels.begin(), vovels.end());**

**string filename = *argv*[0];**

**fstream cur\_file;**

**cur\_file.open(filename, fstream::in | fstream::out | fstream::app);**

**char \*backfile = (char \*) *argv*[1];**

**int state;**

**while (1)**

**{**

**sem\_getvalue(sem, &state);**

**if (state == 0) {**

**int fd = shm\_open(backfile, O\_RDWR | O\_CREAT, S\_IWUSR | S\_IRUSR | S\_IRGRP | S\_IROTH);**

**struct stat statBuf;**

**fstat(fd, &statBuf);**

**int size\_of\_str = statBuf.st\_size;**

**ftruncate(fd, size\_of\_str);**

**char \*mapped = (char \*) mmap(NULL, size\_of\_str, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);**

**std::string allocated = mapped;**

**string result\_str;**

**for (int i = 0; i < size\_of\_str; i++) {**

**if (volSet.find(std::tolower(allocated[i])) == volSet.cend()) {**

**result\_str.push\_back(allocated[i]);**

**}**

**}**

**cur\_file << result\_str << endl;**

**close(fd);**

**munmap(mapped, sizeof(int));**

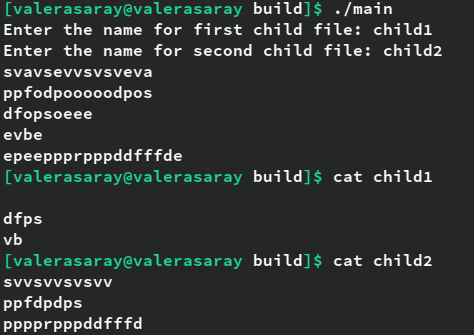
**sem\_post(sem);**

**}**

**}**

**}**

**Демонстрация работы программы**



**Выводы**

В ходе выполнения лабораторной работы №4 я приобрел практические навыки в управлении процессами в ОС, и в обеспечении обмена данных между процессами посредством каналов с использованием семафоров.