Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

Студент: Ибрагимов Далгат Магомедалиевич

Группа: М8О-208Б-22

Вариант: 14

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2023

**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/L0ckR/OS_LABS>

**Постановка задачи**

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 через отображаемые файлы (memory-mapped files).

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

**Общие сведения о программе**

Программа состоит из двух частей – главная программа, родительский процесс, описанный в main.c, и программы дочерних прочессов, описанные в child1.c и child2.c. Программы написаны под операционную систему Linux.

**Общий метод и алгоритм решения**

Родительский процесс получает на вход потоки и создает дочерние процессы, передавая им данные на вход. Перед этим родительский процесс создает два shared-memory файла, по которым он будет передавать дочерним процессам строки, согласно правилу фильтрации. Дочерние процессы уменьшают заглавные буквы и удаляют задвоенные пробелы и записывают результат в соответствующие файлы.

**Исходный код**

**========================== parent.hpp ==========================**

**#pragma once**

**#include <utils.hpp>**

**void ParentProcess(const char \* pathToChild1, const char \* pathToChild2, std::istream & streamIn, std::ostream & streamOut);**

**========================== utils.h ==========================**

**#pragma once**

**#include <iostream>**

**#include <algorithm>**

**#include <fstream>**

**#include <string>**

**#include <unistd.h>**

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

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

**#include <fcntl.h>**

**#include <string.h>**

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

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

**#include <fcntl.h>**

**#include <unistd.h>**

**#include <semaphore.h>**

**sem\_t\* CreateSemaphore(const char \*name, int value);**

**int CreateShm(const char\* name);**

**char\* MapSharedMemory(const int size, int fd);**

**int CreateFork();**

**constexpr const char \*SEM\_1 = "SEM\_1";**

**constexpr const char \*SEM\_2 = "SEM\_2";**

**constexpr const char \*SEM\_3 = "SEM\_3";**

**const int FILE\_SIZE = 1024;**

**========================== utils.cpp ==========================**

**#include "utils.hpp"**

**sem\_t\* CreateSemaphore(const char \*name, int value) {**

**sem\_t \*semptr = sem\_open(name, O\_CREAT, S\_IRUSR | S\_IWUSR, value);**

**if (semptr == SEM\_FAILED){**

**perror("Couldn't open the semaphore");**

**exit(EXIT\_FAILURE);**

**}**

**return semptr;**

**}**

**int CreateShm(const char\* name) {**

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

**if (fd == -1) {**

**std::cerr << "Failed shm\_open\n";**

**exit(-1);**

**}**

**ftruncate(fd, 1024);**

**return fd;**

**}**

**char\* MapSharedMemory(const int size, int fd) {**

**char \*memptr = (char\*)mmap(nullptr, size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);**

**if (memptr == MAP\_FAILED) {**

**perror("Error with file mapping");**

**exit(EXIT\_FAILURE);**

**}**

**return memptr;**

**}**

**int CreateFork() {**

**int pid;**

**pid = fork();**

**if (pid == -1) {**

**std::cerr << "Failed fork()\n";**

**exit(-2);**

**}**

**return pid;**

**}**

**========================== child1.cpp ==========================**

**#include "utils.hpp"**

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

**if (argc != 3) {**

**std::cerr << "Wrong argc in child1\n";**

**exit(-6);**

**}**

**const char\* fileName = argv[1];**

**const char\* fileName2 = argv[2];**

**sem\_t \*semptr1 = CreateSemaphore(SEM\_1, 0);**

**int shd\_fd1 = CreateShm(fileName);**

**char \*memptr1 = MapSharedMemory(FILE\_SIZE, shd\_fd1);**

**sem\_t \*semptr2 = CreateSemaphore(SEM\_2, 0);**

**int shd\_fd2 = CreateShm(fileName2);**

**char \*memptr2 = MapSharedMemory(FILE\_SIZE, shd\_fd2);**

**while(true) {**

**sem\_wait(semptr1);**

**std::string\_view st(memptr1);**

**std::string s = {st.begin(), st.end()};**

**transform(s.begin(), s.end(), s.begin(), ::tolower);**

**strcpy(memptr2, s.c\_str());**

**strcpy(memptr1, "\n");**

**sem\_post(semptr2);**

**if (s == "\0") {**

**break;**

**}**

**}**

**sem\_close(semptr1);**

**sem\_unlink(SEM\_1);**

**shm\_unlink(fileName);**

**munmap(memptr1, 1024);**

**close(shd\_fd1);**

**sem\_close(semptr2);**

**sem\_unlink(SEM\_2);**

**shm\_unlink(fileName2);**

**munmap(memptr2, 1024);**

**close(shd\_fd2);**

**}**

**========================== child2.cpp ==========================**

**#include "utils.hpp"**

**bool BothAreSpaces(char lhs, char rhs){**

**return (lhs == rhs) && (lhs == ' ');**

**};**

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

**if (argc != 3) {**

**std::cerr << "Wrong argc in child2\n";**

**exit(-6);**

**}**

**const char\* fileName1 = argv[1];**

**const char\* fileName2 = argv[2];**

**sem\_t \*semptr2 = CreateSemaphore(SEM\_2, 0);**

**int shd\_fd2 = CreateShm(fileName1);**

**char \*memptr2 = MapSharedMemory(FILE\_SIZE, shd\_fd2);**

**sem\_t \*semptr3 = CreateSemaphore(SEM\_3, 0);**

**int shd\_fd3 = CreateShm(fileName2);**

**char \*memptr3 = MapSharedMemory(FILE\_SIZE, shd\_fd3);**

**while(true) {**

**sem\_wait(semptr2);**

**std::string\_view st(memptr2);**

**std::string s = {st.begin(), st.end()};**

**std::string::iterator new\_end = std::unique(s.begin(), s.end(), BothAreSpaces);**

**s.erase(new\_end, s.end());**

**strcpy(memptr3, s.c\_str());**

**strcpy(memptr2, "\n");**

**sem\_post(semptr3);**

**if (s == "\0") {**

**break;**

**}**

**}**

**sem\_close(semptr2);**

**sem\_unlink(SEM\_2);**

**shm\_unlink(fileName1);**

**munmap(memptr2, FILE\_SIZE);**

**close(shd\_fd2);**

**sem\_close(semptr3);**

**sem\_unlink(SEM\_3);**

**shm\_unlink(fileName2);**

**munmap(memptr3, FILE\_SIZE);**

**close(shd\_fd3);**

**}**

**========================== parent.cpp ==========================**

**#include "parent.hpp"**

**#include "utils.hpp"**

**void ParentProcess(const char \* pathToChild1, const char \* pathToChild2, std::istream & streamIn, std::ostream & streamOut){**

**const char\* name1 = "/shm1";**

**const char\* name2 = "/shm2";**

**const char\* name3 = "/shm3";**

**pid\_t pid = CreateFork();**

**if (pid == 0) { // child 1**

**if (execlp(pathToChild1, "child1", name1, name2, nullptr) == -1) {**

**std::cerr << "Failed execlp()\n";**

**exit(-5);**

**}**

**}else {**

**pid = CreateFork();**

**if (pid == 0) { // child 2**

**if (execlp(pathToChild2, "child2", name2, name3, nullptr) == -1) {**

**std::cerr << "Failed execlp()\n";**

**exit(-5);**

**}**

**} else { //parent**

**sem\_t \*semptr1 = CreateSemaphore(SEM\_1, 0);**

**int shd\_fd1 = CreateShm(name1);**

**char \*memptr1 = MapSharedMemory(FILE\_SIZE, shd\_fd1);**

**sem\_t \*semptr2 = CreateSemaphore(SEM\_2, 0);**

**int shd\_fd2 = CreateShm(name2);**

**char \*memptr2 = MapSharedMemory(FILE\_SIZE, shd\_fd2);**

**sem\_t \*semptr3 = CreateSemaphore(SEM\_3, 0);**

**int shd\_fd3 = CreateShm(name3);**

**char \*memptr3 = MapSharedMemory(FILE\_SIZE, shd\_fd3);**

**int status;**

**std::string line;**

**while (std::getline(streamIn, line)) {**

**line += '\n';**

**strcpy(memptr1, line.c\_str());**

**sem\_post(semptr1);**

**sem\_wait(semptr3);**

**std::string\_view st(memptr3);**

**std::string s = {st.begin(), st.end()};**

**streamOut << s;**

**strcpy(memptr3, "\n");**

**}**

**strcpy(memptr1, "\0");**

**sem\_post(semptr1);**

**streamOut << "";**

**waitpid(-1, &status, 0);**

**waitpid(-1, &status, 0);**

**sem\_close(semptr1);**

**sem\_unlink(SEM\_1);**

**shm\_unlink("/shm1");**

**munmap(memptr1, FILE\_SIZE);**

**close(shd\_fd1);**

**sem\_close(semptr2);**

**sem\_unlink(SEM\_2);**

**shm\_unlink("/shm2");**

**munmap(memptr2, FILE\_SIZE);**

**close(shd\_fd2);**

**sem\_close(semptr3);**

**sem\_unlink(SEM\_3);**

**shm\_unlink("/shm3");**

**munmap(memptr3, FILE\_SIZE);**

**close(shd\_fd3);**

**}**

**}**

**return;**

**}**

**========================== main.cpp ==========================**

**#include "parent.hpp"**

**int main(void) {**

**// export PATH\_TO\_FM\_CHILD1="/home/lockr/projects/OS\_LABS/build/lab3/fm\_child1"**

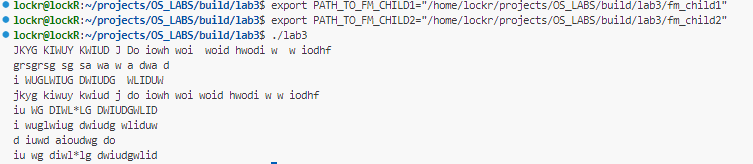
**// export PATH\_TO\_FM\_CHILD2="/home/lockr/projects/OS\_LABS/build/lab3/fm\_child2"**

**ParentProcess(getenv("PATH\_TO\_FM\_CHILD1"), getenv("PATH\_TO\_FM\_CHILD2"), std::cin, std::cout);**

**exit(EXIT\_SUCCESS);**

**}**

**Демонстрация работы программы**

****

**Выводы**

В ходе выполнения лабораторной работы я закрепил навыки работы с системными вызовами Linux. Выяснилось, что системный интерфейс Linux намного более дружелюбен к разработчику чем Windows.