Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт № 8 «Компьютерные науки и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

Студент: Яруллин А.Р.

Группа: М8О-201Б-21

Вариант: №13

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

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

Приобретение практических навыков в:

* Освоение принципов работы с файловыми системами
* Обеспечение обмена данных между процессами посредством технологии «File mapping»
  + - 1. **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 отображаемые файлы (memory-mapped files).

**Вариант №13:** Child1 переводит строки в нижний регистр. Child2 превращает все пробельные символы в символ «\_».

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

* + - 1. **Общие сведения о программе**

Программа представляет из себя файл lab4.c.

В программе используются такие команды, как:

**int sscanf(const char \****str***, const char \****format***, ...***) -* считывает информацию из символьной строки, на которую указывает *str*.

**open(const char \****pathname***, int***flags***) -**  возвращает файловый дескриптор - небольшое, неотрицательное значение - для использования в последующих системных вызовах.

**int fstat(int***filedes***, struct stat \****buf***) –** функция, которая возвращает информацию об файле.

**void \* mmap(void \****start***, size\_t***length***, int***prot* **, int***flags***, int***fd***, off\_t***offset***) -** Функция отражает *length* байтов, начиная со смещения *offset*файла (или другого объекта), определенного файловым описателем *fd*, в память, начиная с адреса *start*. Последний параметр (адрес) необязателен, и обычно бывает равен 0. Настоящее местоположение отраженных данных возвращается самой функцией, и никогда не бывает равным 0.

**int process\_id fork(void) –** создание дочернего процесса, в переменной process\_id будет лежать «специальный код» процесса(-1 - ошибка, 0 - дочерний процесс, >0 - родительский).

**close(int***fd***) -** закрытие файлового дескриптора, который после этого не ссылается ни на один и файл и может быть использован повторно

**int munmap(void \****start***, size\_t***length***) -** Системный вызов удаляет все отражения из заданной области памяти, после чего все ссылки на данную область будут вызывать ошибку "неправильное обращение к памяти" (invalid memory reference). Отражение удаляется автоматически при завершении процесса. С другой стороны, закрытие файла не приведет к снятию отражения.

* + - 1. **Общий метод и алгоритм решения**

C самого начала программа получает название файла, который впоследствии открывается на чтение, затем с помощью вызова mmap этот файл отображается в память, после с помощью вызова fork создаются родительский и дочерний процессы. Дочерний процесс, обращаясь к файлу в памяти, производит вычисления согласно заданию и выводит их в стандартный поток вывода.

* + - 1. **Исходный код**

**main.cpp**

*#include "parent.h"*

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <vector>*

*int main(void) {*

*std::vector <std::string> input;*

*std::string s;*

*while (getline(std::cin, s)) {*

*input.push\_back(std::move(s));*

*}*

*std::vector <std::string> output = ParentRoutine(input);*

*for (const auto &res : output){*

*std::cout << res << std::endl;*

*}*

*return 0;*

*}*

**errorlib.cpp**

*#include "errorlib.h"*

*int Oerror(const char \* error, int id) {*

*write(STDERR\_FILENO, error, strlen(error));*

*exit(id);*

*}*

**parent.cpp**

*#include <unistd.h>*

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <vector>*

*#include <sys/wait.h>*

*#include <fcntl.h>*

*#include <semaphore.h>*

*#include <sys/signal.h>*

*#include <sys/mman.h>*

*#include "parent.h"*

*#include "errorlib.h"*

*std::vector<std::string> ParentRoutine(const std::vector<std::string> &input)*

*{*

*sem\_t \*sem1, \*sem2, \*sem3, \*sem4;*

*char readChar;*

*std::string str;*

*std::vector<std::string> output;*

*int SIZE = 0;*

*std::ofstream inFile;*

*inFile.open("file1");*

*for (const auto &line : input)*

*{*

*inFile << line << '\n';*

*SIZE += line.size() + 1;*

*}*

*inFile.close();*

*unlink("file2");*

*unlink("file3");*

*int file1 = open("file1", O\_RDWR, S\_IRUSR);*

*int file2 = open("file2", O\_RDWR | O\_CREAT, S\_IRUSR | S\_IWUSR);*

*int file3 = open("file3", O\_RDWR | O\_CREAT, S\_IRUSR | S\_IWUSR);*

*if (file1 == -1 || file2 == -1 || file3 == -1)*

*{*

*Oerror("open error", -1);*

*}*

*if (ftruncate(file2, SIZE) == -1 || ftruncate(file2, SIZE) == -1 || ftruncate(file3, SIZE) == -1)*

*{*

*Oerror("ftruncate", -1);*

*}*

*sem1 = sem\_open("1", O\_CREAT | O\_EXCL, 0777, 0);*

*sem2 = sem\_open("2", O\_CREAT | O\_EXCL, 0777, 0);*

*sem3 = sem\_open("3", O\_CREAT | O\_EXCL, 0777, 0);*

*sem4 = sem\_open("4", O\_CREAT | O\_EXCL, 0777, 0);*

*pid\_t pid1;*

*pid\_t pid2;*

*pid1 = fork();*

*if (pid1 > 0)*

*{*

*pid2 = fork();*

*}*

*if (pid1 == 0)*

*{*

*char \*in = (char \*)mmap(NULL, SIZE, PROT\_READ, MAP\_SHARED, file1, 0);*

*char \*ans = (char \*)mmap(NULL, SIZE, PROT\_WRITE, MAP\_SHARED, file3, 0);*

*sem1 = sem\_open("1", 0);*

*sem2 = sem\_open("2", 0);*

*while (1)*

*{*

*sem\_wait(sem1);*

*for (int i = 0; i < SIZE; ++i)*

*{*

*if (in[i] >= 'A' && in[i] <= 'Z')*

*{*

*ans[i] = std::tolower(in[i]);*

*}*

*else*

*{*

*ans[i] = in[i];*

*}*

*}*

*munmap(in, SIZE);*

*munmap(ans, SIZE);*

*sem\_post(sem2);*

*}*

*}*

*if (pid1 > 0 && pid2 == 0)*

*{*

*char \*in = (char \*)mmap(NULL, SIZE, PROT\_READ, MAP\_SHARED, file3, 0);*

*char \*ans = (char \*)mmap(NULL, SIZE, PROT\_WRITE, MAP\_SHARED, file2, 0);*

*sem3 = sem\_open("3", 0);*

*sem4 = sem\_open("4", 0);*

*while (1)*

*{*

*sem\_wait(sem3);*

*for (int i = 0; i < SIZE; ++i)*

*{*

*if (in[i] == ' ')*

*{*

*ans[i] = '\_';*

*}*

*else*

*{*

*ans[i] = in[i];*

*}*

*}*

*munmap(in, SIZE);*

*munmap(ans, SIZE);*

*sem\_post(sem4);*

*}*

*}*

*if (pid1 == -1 || pid2 == -1)*

*{*

*Oerror("can't create processes child:\n", -1);*

*}*

*if (pid1 != 0 && pid2 != 0)*

*{*

*char \*ans1 = (char \*)mmap(NULL, SIZE, PROT\_WRITE, MAP\_SHARED, file2, 0);*

*sem1 = sem\_open("1", 0);*

*sem2 = sem\_open("2", 0);*

*sem3 = sem\_open("3", 0);*

*sem4 = sem\_open("4", 0);*

*sem\_post(sem1);*

*sem\_wait(sem2);*

*str.clear();*

*sem\_post(sem3);*

*sem\_wait(sem4);*

*for (int i = 0; i < SIZE; ++i)*

*{*

*readChar = ans1[i];*

*if (readChar == '\n')*

*{*

*output.push\_back(std::move(str));*

*str.clear();*

*}*

*else*

*{*

*str += readChar;*

*}*

*}*

*munmap(ans1, SIZE);*

*sem\_close(sem1);*

*sem\_close(sem2);*

*sem\_close(sem3);*

*sem\_close(sem4);*

*sem\_unlink("1");*

*sem\_unlink("2");*

*sem\_unlink("3");*

*sem\_unlink("4");*

*kill(pid1, SIGKILL);*

*kill(pid2, SIGKILL);*

*}*

*return output;*

*}*

**errorlib.h**

*#ifndef ERRORLIB\_H*

*#define ERRORLIB\_H*

*#include <stdlib.h>*

*#include <stdio.h>*

*#include <string.h>*

*#include <unistd.h>*

*int Oerror(const char \* error, int id);*

*#endif*

**parent.h**

*#ifndef PARENT\_H*

*#define PARENT\_H*

*#include <sys/wait.h>*

*#include <iostream>*

*#include <fstream>*

*#include <vector>*

*std::vector<std::string> ParentRoutine(const std::vector<std::string> &input);*

*#endif //PARENT\_H*

* + - 1. **Демонстрация работы программы**

**Input**

TThE quick browNNN

f0x jumps .OvEr. the l@zy dog

**Output**

tthe\_quick\_brownnn\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_

f0x\_jumps\_.over.\_the\_l@zy\_dog

**7. Выводы**

Проделав лабораторную работу, я еще потренировался в работе с процессами в ОС UNIX, узнал и освоил технологию file-maping, которая иногда может дать значительный прирост производительности, если сравнивать ее с обычной буферизированной работой с файлом. Эта технология позволяет отображать файлы на участок памяти, доступ к которой имеет как родительский, так и дочерний процесс. Однако необходимо быть крайне острожным при работе с этой технологией, поскольку отображаемая память является общей для всех процессов, поэтому важно знать последовательность обращения к памяти, или использовать специальные средства(семафоры или мьютексы), которые контролируют доступ нескольких процессов до общего ресурса. Поэтому если сравнивать эту лабораторную работу с лабораторной №2, то здесь прослеживается некоторое удобство, по сравнению с работой с pipe, однако, с другой стороны, здесь необходимо контролировать доступ к общей памяти, что заставляет быть всегда начеку.