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 №3

По курсу «Операционные системы»

Студент: Теребаев К. Д.

Группа: М8О-203Б-22

Преподаватель: Миронов Е. С.

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

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

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

Москва, 2024

**Цель работы**

Приобретение практических навыков в:

* Освоение принципов работы с файловыми системами
* Обеспечение обмена данных между процессами посредством технологии «File mapping»

**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 через отображаемые файлы (memory-mapped files).

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

2 вариант) Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип float. Количество чисел может быть произвольным.

**Общие сведения о программе**

1. write () - переписывает count байт из буфера в файл. Возвращает количество записанных байт или -1;
2. mmap () – отображает файл на память. Возвращает указатель на начало файла, при ошибке возвращает MAP\_FAILED;
3. munmap () – отменяет отображение файла на память. В случае ошибки возвращает -1;
4. shm\_open () – открывает или создает при необходимости объект разделяемой памяти. Возвращает дескриптор открытого файла или -1;
5. shm\_unlink () – обратная к shm\_open;
6. sem\_open () – инициализирует и открывает именованный семафор;
7. sem\_close () – обратная к sem\_open;
8. sem\_post () – увеличивает (разблокирует) семафор. Возвращает 0 при успехе и -1 при неудаче;
9. sem\_wait () – уменьшает (блокирует) семафор. Возвращает 0 при успехе и -1 при неудаче;
10. ftruncate () – устанавливает файлу заданную длину в байтах.
11. open () - открывает или создаёт файл при необходимости. Возвращает дескриптор открытого файла или -1;
12. close () - закрывает файловый дескриптор, который больше не ссылается ни на один файл, возвращает 0 или -1;
13. fork () - порождается процесс-потомок. Весь код после fork () выполняется дважды, как в процессе-потомке, так и в процессе-родителе. Процесс-потомок и процесс-родитель получают разные коды возврата после вызова fork (). Процесс-родители возвращает идентификатор pid потомка или -1. Процесс-потомок возвращает 0 или -1.

**Общий алгоритм решения**

Используя системный вызов mmap будет отображать файл на память. Теперь наш файл представляет собой массив символов, в него родительский процесс будет записывать, введенные пользователем строчки, а дочерний – читать и проверять на валидность правилу. Чтобы синхронизовать их работу используем семафоры, благодаря им дочерний процесс сможет понимать, что родитель записал в файл строчку, а родительский процесс поймет, что ребенок проверил ее. Если строчка не удовлетворяет условию, то дочерний процесс запишет в конец файла константу, каждый раз родительский процесс проверяет последний элемент файла, в случае если в конце файла оказался символ EOF, то мы ждем завершения работы дочернего процесса и программа останавливает свою работу.

**Основные файлы программы**

main.cpp  
#include <fcntl.h>

#include <semaphore.h>

#include <sys/mman.h>

#include <sys/wait.h>

#include <iostream>

#include "shared\_data.hpp"

using namespace std;

int main() {

    int err;

    string file;

    cout << "Enter filename: ";

    cin >> file;

    cout << "Enter commands:" << endl;

    int shm\_fd = shm\_open(shm\_name, O\_RDWR | O\_CREAT, S\_IRUSR | S\_IWUSR);

    throw\_if(shm\_fd, "Shared memory open error");

    err = ftruncate(shm\_fd, sizeof(SharedData));

    throw\_if(err, "Shared memory truncate error");

    SharedData\* data = (SharedData\*)mmap(NULL, sizeof(SharedData), PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

    if (data == MAP\_FAILED) {

        throw runtime\_error("Error displaying shared memory");

    }

    err = sem\_init(&data->sem1, 1, 0);

    throw\_if(err, "Semaphore init error");

    err = sem\_init(&data->sem2, 1, 0);

    throw\_if(err, "Semaphore init error");

    pid\_t pid = fork();

    throw\_if(pid, "Fork failed");

    if (pid == 0) {

        err = execl("./child\_process", "./child\_process", file.c\_str(), NULL);

        throw\_if(err, "Child file error");

    } else {

        string s, out = "";

        while (getline(cin, s)) {

            if (s.size()) {

                s += '\n';

            }

            out += s;

        }

        data->data[15] = '\0';

        for (size\_t i = 0; i < out.size(); ++i) {

            if (!(i % 15)) {

                err = sem\_post(&data->sem1);

                throw\_if(err, "Semaphore post error");

                err = sem\_wait(&data->sem2);

                throw\_if(err, "Semaphore wait error");

            }

            data->data[i % 15] = out[i];

        }

        if (!out.size() % 15) {

            data->data[out.size() % 15] = '\0';

        }

        data->end = true;

        err = sem\_post(&data->sem1);

        throw\_if(err, "Semaphore post error");

        wait(nullptr);

    }

    err = sem\_destroy(&data->sem1);

    throw\_if(err, "Semaphore destroy error");

    err = sem\_destroy(&data->sem2);

    throw\_if(err, "Semaphore destroy error");

    munmap(data, sizeof(SharedData));

    shm\_unlink(shm\_name);

}

child\_process.cpp

#include <fcntl.h>

#include <semaphore.h>

#include <sys/mman.h>

#include <unistd.h>

#include <iostream>

#include "shared\_data.hpp"

using namespace std;

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

    int file\_d = open(argv[1], O\_CREAT | O\_WRONLY | O\_TRUNC, S\_IRWXU);

    if (file\_d == -1) {

        cerr << "Creating file error" << endl;

        return 1;

    }

    if (dup2(file\_d, fileno(stdout)) == -1) {

        cerr << "Dup2 error" << endl;

        return 1;

    }

    int shm\_fd = shm\_open(shm\_name, O\_RDWR, S\_IRUSR | S\_IWUSR);

    throw\_if(shm\_fd, "Shared memory open error");

    SharedData\* data = (SharedData\*)mmap(NULL, sizeof(SharedData), PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

    if (data == MAP\_FAILED) {

        throw std::runtime\_error("Shared memory map error");

    }

    string in;

    while (1) {

        int err = sem\_wait(&data->sem1);

        throw\_if(err, "Semaphore wait error");

        in += data->data;

        err = sem\_post(&data->sem2);

        throw\_if(err, "Semaphore post error");

        if (data->end) {

            break;

        }

    }

    float res = 0;

    size\_t pos = 0;

    for (size\_t i = 0; i < in.size(); ++i) {

        if (in[i] == '\n' && (i - pos)) {

            res += stof(in.substr(pos, i - pos));

            pos = i + 1;

            cout << res << endl;

            res = 0;

        } else if (isspace(in[i]) && (i - pos)) {

            res += stof(in.substr(pos, i - pos));

            pos = i + 1;

        }

    }

    munmap(data, sizeof(SharedData));

    shm\_unlink(shm\_name);

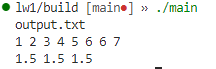
    fclose(stdout);

    return 0;

}

**Пример работы**

Ввод



Вывод (файл output.txt)



**Вывод**

В ходе выполнения лабораторной работы я изучил технологию «File mapping» и применил на практике семафоры. Применение этой технологии передачи данных имеет преимущество над pipe’ом, так как к этим данным будут иметь доступ все процессы. Но тут же возникает основная проблема: синхронизация. Нужно точно понимать, как будет работать программа, чтобы правильно синхронизировать работу процессов.