Московский Авиационный Институт
(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу
«Операционные системы»

Работа с общей памятью

Студент: Мирошников Дмитрий Евгеньевич
Группа: М8О-210Б-22
Вариант: 4
Преподаватель: Соколов Андрей Алексеевич
Оценка:_____
Дата:_____
Подпись:_____

Москва, 2023

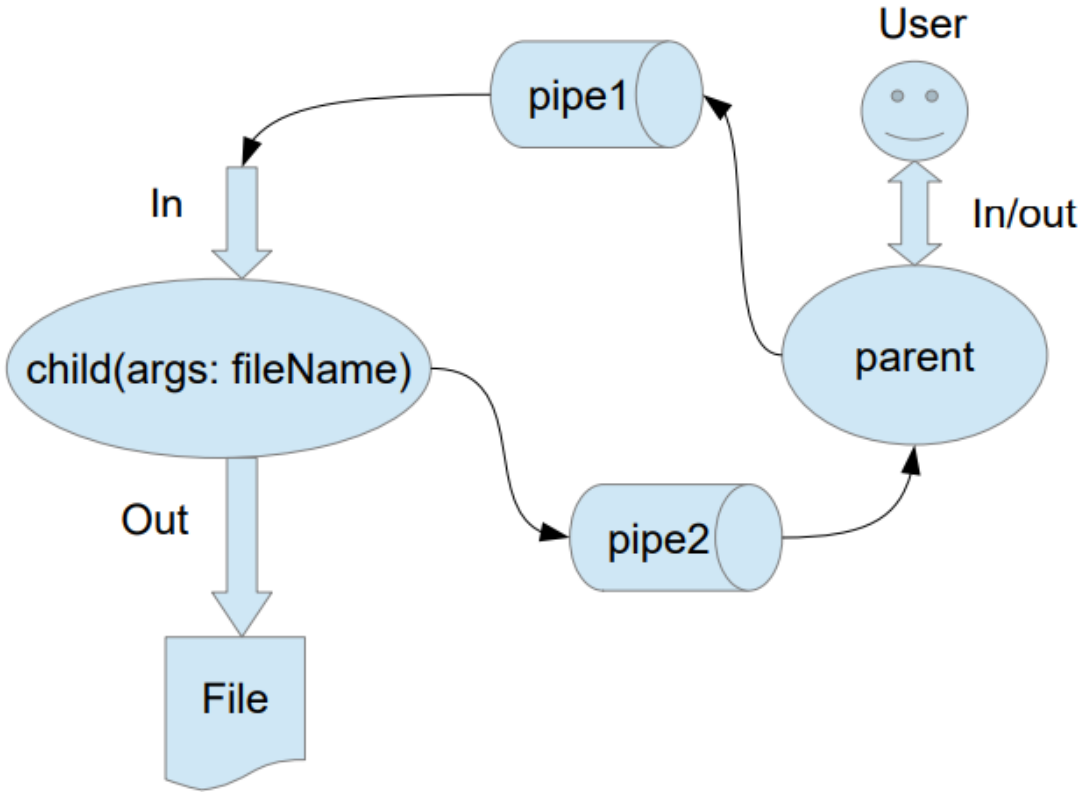
Постановка задачи

Цель работы
Целью работы является приобретение практических навыков в:

1) Освоении принципов работы с файловыми системами
2) 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 решения 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.

Группа вариантов 1



Пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа на

последующие, а результат выводит в файл. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип float. Количество чисел может быть произвольным.

Общие сведение о программе

Программа компилируется из файла parent.cpp. Также используются заголовочные файлы: iostream, stdio.h, fcntl.h, unistd.h, sys/wait.h, sys/mman.h, vector.

В программе используются следующие системные вызовы:

1) mmap - создаёт новое сопоставление в виртуальном адресном пространстве вызывающий процесс. Начальный адрес нового сопоставления: указан в addr. Аргументы функции: **void* addr** - желаемый адрес участка отображаемой памяти, передаём NULL - тогда ядро само выберет этот адрес. **size_t len -** количество байт, которое нужно отобразить в память, **int prot -** число, определяющее степень защищённости отображённого участка памяти(только чтение, только запись, исполнение, область недоступна). Обычные значения - PROT_READ, PROT_WRITE(можно комбинировать через |), **int flag -** описывает атрибуты области. Обычное значение - MAP_SHARED, **int filedes -** дескриптор файла, который нужно отобразить, **off_t off -** смещение отображённого участка от начала файла.

2) munmap - функция должна удалить сопоставления для всех страниц, содержащих любую часть адресного пространства процесса, начиная с addr и продолжая len байт. Аргументы функции: **void* add**r - указатель на виртуальное адресное пространство, **size_t len** - его размер в байтах.

3) fork - вызов создаёт новый процесс посредством копирования вызывающего процесса. Новый процесс считается дочерним процессом. Вызывающий процесс считается родительским процессом.

4) close - закрывает файловый дескриптор, который после этого не ссылается ни на один и файл и может быть использован повторно. Все блокировки, находящиеся на соответствующем файле, снимаются (независимо от того, был ли использован для установки блокировки именно этот файловый дескриптор).

Общий алгоритм и метод решения

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы mmap.
2. Переписать вариант первой лабораторной работы, работающей на pipe, используя mmap
3. Реализовать ввод и вывод результата
4. Путём двух процессов работать со строками, сообщая между собой информацию
5. В созданный по ходу работы с программой файл, записать результаты вычислений

Основные файлы программы

shared_memory.hpp

```cpp
#pragma once

#include <semaphore.h>
#include <string>
const char* shared_memory_name = "mySharedMemory";
const int prots = PROT_READ | PROT_WRITE;
const int flags = MAP_SHARED;
const int DataSize = 5;
struct SharedMemory {
    char data[DataSize];
    float value;
    int flag;
    sem_t semaphore1;
    sem_t semaphore2;
    sem_t semaphore3;
};
```

parent.cpp

```cpp
#include <iostream>
#include <unistd.h>
#include <string.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <vector>

#include "shahed_memory.hpp"
int Min(int a, int b) {
    return (a <= b) ? a : b;
}
int main() {
    std::cout.precision(6);
    std::cout << std::fixed;
    int temp;
    std::vector<float> results;
    std::string name, str;
```

```
    std::cout << "Enter the name of file: ";
    std::getline(std::cin, name);
    //Открываю область shared memory
    int shared_memory_fd = shm_open(shared_memory_name, O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
    if (shared_memory_fd == -1) {
        std::cerr << "Ошибка при открытии shared memory" << '\n';
        exit(EXIT_FAILURE);
    }
    //Изменяю размер shared_memory до размера структуры
    if (ftruncate(shared_memory_fd, sizeof(SharedMemory)) == -1) {
        std::cerr << "Ошибка при изменении размкера shared memory" << '\n';
        exit(EXIT_FAILURE);
    }
    //Отображаю участок памяти
    SharedMemory* shared_memory = (SharedMemory*)mmap(NULL, sizeof(SharedMemory), prots,
flags,shared_memory_fd, 0);
    if (shared_memory == MAP_FAILED) {
        std::cerr << "Ошибка при отображении shared_memory" << '\n';
        exit(EXIT_FAILURE);
    }
    //Инициализация семафоров
    if (sem_init(&shared_memory->semaphore1, 1, 0) == -1) {
        std::cerr << "Ошибка при инициализации семафора 1" << '\n';
        exit(EXIT_FAILURE);
    }
    if (sem_init(&shared_memory->semaphore2, 1, 0) == -1) {
        std::cerr << "Ошибка при инициализации семафора 2" << '\n';
        exit(EXIT_FAILURE);
    }
    if (sem_init(&shared_memory->semaphore3, 1, 0) == -1) {
        std::cerr << "Ошибка при инициализации семафора 3" << '\n';
        exit(EXIT_FAILURE);
    }
    //Создаём дочерний процесс
    pid_t pid = fork();
    if (pid == -1) {
        std::cout << "Ошибка при создании дочернего процесса" << '\n';
        exit(EXIT_FAILURE);
    }
    //Дочерний процесс
    if (pid == 0) {
        temp = execl("child", "child", name.c_str(), NULL);
        if (temp == -1) {
            std::cerr << "Ошибка при запуске программы в дочернем процессе" << '\n';
            exit(EXIT_FAILURE);
        }
    }
    shared_memory->flag = 1;
    //Родительский процесс
    if (pid > 0) {
        bool first_number_flag = true;
        int start, finish;
        while (std::getline(std::cin, str)) {
            if (!shared_memory->flag) {
                exit(EXIT_FAILURE);
            }
            if (!first_number_flag) {
                results.push_back(shared_memory->value);
            }
            start = 0;
            finish = str.size();
```

```
            while(1) {
                if (DataSize == finish - start) {
                    shared_memory->flag = 0;
                }
                strncpy(shared_memory->data, str.substr(start, Min(DataSize, finish -
start)).c_str(), sizeof(shared_memory->data));
                start += Min(DataSize, finish - start);
                sem_post(&shared_memory->semaphore1);
                sem_wait(&shared_memory->semaphore2);
                if (start == finish) {
                    break;
                }
            }
            first_number_flag = false;
            sem_wait(&shared_memory->semaphore3);
        }
        if (shared_memory->flag) {
            results.push_back(shared_memory->value);
        }
        for (float elem : results) {
            std::cout << elem << '\n';
        }
        sem_destroy(&shared_memory->semaphore1);
        sem_destroy(&shared_memory->semaphore2);
        sem_destroy(&shared_memory->semaphore3);
        munmap(shared_memory, sizeof(SharedMemory));
        shm_unlink(shared_memory_name);
    }
    return 0;
}
```

child.cpp

```
#include <iostream>
#include <unistd.h>
#include <vector>
#include <fcntl.h>
#include <string>
#include <sys/wait.h>
#include <sys/mman.h>

#include "shahed_memory.hpp"
const float EPS = 1e-10;
int main(int argc, char* argv[]) {
    int fd = open(argv[1], O_CREAT | O_WRONLY, S_IRWXU);
    if (fd == -1) {
        std::cerr << "Ошибка при открытии файла для записи ответов" << '\n';
        exit(EXIT_FAILURE);
    }
    if (ftruncate(fd, 0) == -1) {
        std::cerr << "Ошибка при очистке файла" << '\n';
        exit(EXIT_FAILURE);
    }
    int shared_memory_fd = shm_open(shared_memory_name, O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
    if (shared_memory_fd == -1) {
        std::cerr << "Ошибка при открытии shared memory" << '\n';
        exit(EXIT_FAILURE);
    }
```

```
    SharedMemory* shared_memory = (SharedMemory*)mmap(NULL, sizeof(SharedMemory), prots, flags,
shared_memory_fd, 0);
    if (shared_memory == MAP_FAILED) {
        std::cerr << "Ошибка при отображении shared_memory" << '\n';
        exit(EXIT_FAILURE);
    }
    std::string s, temp;
    int first_gap_index;
    float first_number;
    std::vector<float> numbers;
    while(1) {
        s = "";
        while(1) {
            sem_wait(&shared_memory->semaphore1);;
            s += std::string(shared_memory->data);
            sem_post(&shared_memory->semaphore2);
            if (std::string(shared_memory->data).size() < DataSize || shared_memory->flag == 0)
{
                shared_memory->flag = 1;
                break;
            }
        }
        s += '\n';
        for (int i = 0; i < s.size(); ++i) {
            if (s[i] == ' ' || s[i] == '\n') {
                first_gap_index = i;
                break;
            }
        }
        temp = "";
        first_number = std::stof(s.substr(0, first_gap_index + 1));
        for (int i = first_gap_index + 1; i < s.size(); ++i) {
            if (s[i] == ' ' || s[i] == '\n') {
                numbers.push_back(std::stof(temp));
                temp = "";
                continue;
            }
            temp += s[i];
        }
        for (int i = 0; i < numbers.size(); ++i) {
            if (numbers[i] < EPS && numbers[i] > -EPS) {
                std::cerr << "Ошибка при попытке деления на 0" << '\n';
                shared_memory->flag = 0;
                sem_post(&shared_memory->semaphore3);
                exit(EXIT_FAILURE);
            }
            first_number /= numbers[i];
        }
        numbers.clear();

        shared_memory->value = first_number;
        dprintf(fd, "%f\n", first_number);
        sem_post(&shared_memory->semaphore3);
    }
    munmap(shared_memory, sizeof(SharedMemory));
    close(fd);
    return 0;
}
```

Пример работы

```
dmitrijmrsh@LAPTOP-7SMT8REA:~/Labs-
git/os_lab_3/src/build$ ./parent
Enter the name of file: out
1
1 2 3 4 5
3 2 1
4.5 1.5 2
4.5 1.5
2222222
0 1
1.000000
0.008333
1.500000
1.500000
3.000000
2222222.000000
0.000000

dmitrijmrsh@LAPTOP-7SMT8REA:~/Labs-
git/os_lab_3/src/build$ ./parent
Enter the name of file: out
1 2 3 4 5
2 3 4
3 2 1
1 0
Ошибка при попытке деления на 0
0.008333
0.166667
1.500000
```

Вывод

Выполнив данную лабораторную работу, я изучил приципы работы виртуальной памяти, оценил удобство её использования. Переписав лабораторную работу 1 на mmap, я понял, что можно работать с передачей данных по-разному. В будущем мне может пригодится умение работать с mmap, так как эта технология актуальна в низкоуровневой разработке.