Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

Группа:М8О-215Б-23

Студент: Венгер Ирина Витальевна

Преподаватель: Миронов Е.С.

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

Дата: \_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2024.

# Содержание

1. Постановка задачи.
2. Общие сведения о программе.
3. Общий метод и алгоритм решения.
4. Код программы.
5. Демонстрация работы программы.
6. Вывод.

## 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.

3 вариант) Пользователь вводит команды вида: «число число число». 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 int. Количество чисел может быть произвольным.

## Общие сведения о программе

Также, как и в 1 лабораторной работе, нужно реализовать 2 процесса, родительский и дочерний, логика такая же, как и в первой лабораторной, пользователь вводит имя файла, в который он хочет произвести запись результата, а затем вводит 2 числа: делимое и делитель. Программа выводит в консоль результат деления внутри дочернего процесса, а также записывает результат в файл. Главное отличие от лабораторной работы номер 1 – реализация, здесь используется для передачи данных в родительский процесс не пайп, а память(shared memory), такая технология называется File mapping, т.е. с помощью семафоров оба процесса могут обращаться к одному участку памяти. Внутри самой программы определена структура того участка памяти, с которым мы работаем: command и filename, command постоянно поддаётся изменению со стороны обоих процессов, так как здесь передаётся не только команда, но и результат работы дочернего процесса. А осуществить контроль над разделяемой памятью, а точнее над временем считывания из неё помогут семафоры. Всего используется 2 файла: main.c – для родительского процесса – пользователь запускает именно его, child.c – реализация родительского процесса.

## Общий метод и алгоритм решения

Сначала мы создаём структуру нашей, разделяемой памяти, в ней будет всего 2 поля, имя файла, с которым работает программа и команда, которая по совместительству будет ещё и возвращать результат процессу-родителю. Далее идёт функция main, в которой и происходит действие программы. В main.c и child.c внутри функций main начало похоже, мы обьявляем структуру shared memory, только если в main.c мы создаём этот раздел, то внутри child.c мы уже подключаемся к нему и ведём работу с ним. Функция ftruncate устанавливает размер выделенной памяти, который равен размеру структуры, которую мы обьявили в самом начале. Далее используем функцию mmap, которая отображает этот участок памяти в адресное пространство процессора. Далее с помощью функции sem\_open() создаём семафоры, а затем уже используем знакомую функцию fork(), создавая дочерний процесс. Если вернулся 0 – то мы находимся внутри дочернего процесса. Далее происходит та же логика, что и в первой лабораторной работе, запускается дочерний процесс и начинается его работа. Далее идёт уже работа пользователя с родительским процессом. Происходит считывание имени файла, а затем и команды, которую ввёл пользователь. Затем происходит запись этой команды в разделённую память. Со стороны child.c происходит считывание имени файла и команды, далее происходит обработка команды, и запись результата в файл. В случае деления на 0 выбрасывается Zero Division сообщение. Благодаря функции sem\_wait() мы можем наладить логику взаимодействия между процессами, чтобы они не опережали друг друга и не было такой ситуации, что ребёнок обращается к shared\_memory, а там ничего нет. Функция sem\_post() увеличивает счётчик семафора на 1 выше, что сигнализирует ребёнку о том, что тот может начать работу. В конце работы все семафоры закрываются, а функция munmap() освобождает занятую память.

## Код программы

Код программы смотрите в приложении 1.

## Использование утилиты strace

Скриншоты strace представлены в приложении 2. По результатам strace видно, какие системные вызовы использует ОС во время работы нашей программы: mmap – выделение памяти под какую-то структуру или переменную. ftruncate – устанавливает размер выделенной памяти. Clone – создала процесс и вернула его PID. Futex – примитив синхронизации, который используется для реализации семафоров, мьютексов, в этой программе мы используем семафоры, поэтому его появление вполне логично.

## Демонстрация работы программы

(base) boopie@MacBook-Air-Irina src % ./main

Введите имя файла: file.txt

Введите числа, разделенные пробелом или 'выход' для завершения: 3 4 5

Результат деления: 0

Введите числа, разделенные пробелом или 'выход' для завершения: 0

Результат деления: 0

Введите числа, разделенные пробелом или 'выход' для завершения: 0 0

Результат деления: Ошибка деления на 0

Введите числа, разделенные пробелом или 'выход' для завершения: выход

Выход из child...

Как мы видим, программа работает корректно, а в файле file.txt произошла запись результата, по команде ‘выход’ дочерний процесс завершает свою работу, а затем и родительский процесс также завершает свою работу.

## Вывод

В данной лабораторной работе я познакомился с ещё одним способом передачи сообщений и информации между процессами: file mapping. У этого способа есть свои преимущества по сравнению с pipe, например, сразу несколько процессов могут обращаться к одному адресу в памяти и взаимодействовать с ним, в пайпах пришлось бы знать количество процессов, чтобы посчитать количество каналов, с точки зрения написания кода это было бы не очень удобно. Недостатком file mapping по сравнению с pipe является тяжеловесность shared\_memory, всё таки пайпы легковеснее и на более простых задачах, где количество процессов небольшое, в плане памяти использовать пайпы было бы эффективнее. Оба способа имеют свои плюсы и минусы, я рад, что у меня получилось познакомиться с ними и понять, как взаимодействовать между процессами.

## Приложения

Приложение 1 – код программы:

Main.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <semaphore.h>

#define MAX\_COMMAND\_LEN 256

#define SHM\_NAME "/shared\_memory"

#define SEM\_PARENT "/sem\_parent"

#define SEM\_CHILD "/sem\_child"

typedef struct {

    char filename[MAX\_COMMAND\_LEN];

    char command[MAX\_COMMAND\_LEN];

} shared\_data\_t;

int main() {

    //создаём разделяемую память с правами доступа 0666, где 0 - отсутстиве спец флагов, а 6 = 4 + 2, т.е. чтение плюс запись для владельца, группы и остальных

    int shm\_fd = shm\_open(SHM\_NAME, O\_CREAT | O\_RDWR, 0666);

    if (shm\_fd == -1) {

        perror("Ошибка shm\_open");

        exit(1);

    }

    //устанавливаем размер разделяемой памяти равный размеру созданной структуры

    ftruncate(shm\_fd, sizeof(shared\_data\_t));

    //mmap отображает эту область памяти в адресное пространство процесса, shared\_mem указывает на область памяти, которую видят и родительский и дочерний процессы

    shared\_data\_t \*shared\_mem = (shared\_data\_t \*)mmap(0, sizeof(shared\_data\_t), PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

    if (shared\_mem == MAP\_FAILED) {

        perror("Ошибка mmap");

        exit(1);

    }

    //создаём 2 семафора для уведомления дочернего процесса, что команда введенаб а второй семафор, что результат готов

    sem\_t \*sem\_parent = sem\_open(SEM\_PARENT, O\_CREAT, 0666, 0);

    sem\_t \*sem\_child = sem\_open(SEM\_CHILD, O\_CREAT, 0666, 0);

    //делаем fork и создаём дочерний процесс

    pid\_t pid = fork();

    if (pid < 0) {

        perror("Ошибка fork");

        exit(1);

    }

    //находимся в дочернем

    if (pid == 0) {

        execl("./child", "child", NULL);

        perror("Ошибка execl");

        exit(1);

    } else {

        //внутри родительского

        printf("Введите имя файла: ");

        fgets(shared\_mem->filename, sizeof(shared\_mem->filename), stdin);

        shared\_mem->filename[strcspn(shared\_mem->filename, "\n")] = 0;

        char command[MAX\_COMMAND\_LEN];

        while (1) {

            printf("Введите числа через пробел или ‘выход’ для завершения: ");

            //считываем команду из stdin

            fgets(command, sizeof(command), stdin);

            command[strcspn(command, "\n")] = 0;

            //копируем команду в shared memory

            strcpy(shared\_mem->command, command);

            //увеличиваем счётчик семафора и дочерний процесс может быть выполнен

            sem\_post(sem\_parent);

            //проверка на exit

            if (strcmp(command, "выход") == 0) {

                break;

            }

            //ждём выполнение дочернего процесса,

            sem\_wait(sem\_child);

            printf("Результат деления: %s\n", shared\_mem->command);

        }

        wait(NULL);

        //закрываем все семафоры и освобождаем ресурсы

        sem\_close(sem\_parent);

        sem\_close(sem\_child);

        sem\_unlink(SEM\_PARENT);

        sem\_unlink(SEM\_CHILD);

        munmap(shared\_mem, sizeof(shared\_data\_t));

        shm\_unlink(SHM\_NAME);

        close(shm\_fd);

    }

    return 0;

}

Child.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <semaphore.h>

#define MAX\_COMMAND\_LEN 256

//область разделяемой памяти

#define SHM\_NAME "/shared\_memory"

//семафоры для синхронизации между процессами

#define SEM\_PARENT "/sem\_parent"

#define SEM\_CHILD "/sem\_child"

typedef struct {

    char filename[MAX\_COMMAND\_LEN];

    char command[MAX\_COMMAND\_LEN];

} shared\_data\_t;

int main() {

    int shm\_fd = shm\_open(SHM\_NAME, O\_RDWR, 0666);

    if (shm\_fd == -1) {

        perror("Ошибка shm\_open");

        exit(1);

    }

    //устанавливаем размер разделяемой памяти равный размеру созданной структуры

    ftruncate(shm\_fd, sizeof(shared\_data\_t));

    //mmap отображает эту область памяти в адресное пространство процесса, shared\_mem указывает на область памяти, которую видят и родительский и дочерний процессы

    shared\_data\_t \*shared\_mem = (shared\_data\_t \*)mmap(0, sizeof(shared\_data\_t), PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

    if (shared\_mem == MAP\_FAILED) {

        perror("Ошибка mmap");

        exit(1);

    }

    //создаём 2 семафора для уведомления дочернего процесса, что команда введенаб а второй семафор, что результат готов

    sem\_t \*sem\_parent = sem\_open(SEM\_PARENT, 0);

    sem\_t \*sem\_child = sem\_open(SEM\_CHILD, 0);

    while (1) {

        sem\_wait(sem\_parent);

        if (strcmp(shared\_mem->command, "выход") == 0) {

            printf("Выход из child\n");

            break;

        }

        //открываем файл, имя которого было передано через shared mem

        FILE \*file = fopen(shared\_mem->filename, "a");

        if (file == NULL) {

            perror("Ошибка открытия файла");

            exit(1);

        }

        int num1, num2, result;

        char \*token = strtok(shared\_mem->command, " ");

        num1 = atoi(token);

        result = num1;

        int division\_by\_zero = 0;

        while ((token = strtok(NULL, " ")) != NULL) {

            num2 = atoi(token);

            if (num2 == 0) {

                strcpy(shared\_mem->command, "Ошибка деления на 0");

                fprintf(file, "Ошибка деления на 0\n");

                division\_by\_zero = 1;

                break;

            }

            result /= num2;

        }

        if (!division\_by\_zero) {

            sprintf(shared\_mem->command, "%d", result);

            fprintf(file, "Результат деления: %d\n", result);

        }

        fflush(file);

        fclose(file);

        //говорим родительскому процессу, что ребёнок завершил работу

        sem\_post(sem\_child);

    }

    munmap(shared\_mem, sizeof(shared\_data\_t));

    close(shm\_fd);

    sem\_close(sem\_parent);

    sem\_close(sem\_child);

    return 0;

}

Приложение 2 – strace:





