Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

Студент: Дубровин Дмитрий
Группа: М80-207Б-21
Вариант: 1
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2022

**Содержание**

## Репозиторий

https://github.com/1droozd1/os_labs/tree/main/lab_4

## Постановка задачи

### 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.

**Задание по варианту (вариант 1):** пользователь вводит команды вида: «число число число<endline>». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип int. Количество чисел может быть произвольным.

## Общие сведения о программе

Программа представляет из себя один файл main.c или же main2.c

## Общий метод и алгоритм решения

Есть 2 варианта подхода к данной задачи, в процессе решения я не сначала не смог реализовать первый из-за чего перешел ко второму. В процессе разработки второго я понял, какую именно ошибку я допустил при работе с первым вариантом — как результат два варианта решения задачи.

Отличаются они прежде всего методом обработки чисел, вводимых пользователем, а именно: в первом варианте мы сохраняем все полученные числа в строку, чтобы после передать их в дочерний процесс одной строй, а после уже в самом дочернем процессе обработать их и

найти сумму, когда во втором варианте мы передаем в дочерний процесс число за числом, показывая нужные данные состояниями.

## Исходный код

### *Main.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <fcntl.h>

const unsigned int MAX_LENGTH = 1 * 1024 * 1024;  // Не выделять больше мегабайта памяти
const unsigned int CHUNK_SIZE = 100;

typedef struct number{
    int num;
    int result;
    char read_num[1000];
    char filename[20];
} number;

int getting_value(sem_t *semaphore)
{
    int s;
    sem_getvalue(semaphore, &s);
    return s;
}

int sum_from_char(char *s)
{
    int i = 0, sum = 0, n = 0, flag1 = 0;
    int len = strlen(s);
```

```
    for (int i = 0; i < len; ++i) {
        if (s[i] >= '0' && s[i] <= '9' && flag1 == 0) {
            n = n * 10 + (s[i] - '0');
        } else if (flag1 == 1 && s[i] >= '0' && s[i] <= '9') {
            n = n * 10 - (s[i] - '0');
        } else if (s[i] == '-') {
            flag1 = 1;
        } else if (s[i] == ' ' && flag1 == 1) {
            flag1 = 0;
            sum += n;
            n = 0;
            continue;
        } else if (n) {
            sum += n;
            n = 0;
            flag1 = 0;
        }
    }
    return sum + n;
}


int main(int args, char *argv[])
{
    pid_t id;
    unsigned int str_len = CHUNK_SIZE;
    char *str_ptr = malloc(CHUNK_SIZE * sizeof(char));
    char file_name[20];

    number *buffer = mmap(NULL, sizeof(number), PROT_READ | PROT_WRITE,
MAP_SHARED | MAP_ANONYMOUS, 0, 0);
    if (buffer == NULL) {
        perror("Can't mmap");
        return -1;
```

```
    }

    sem_t *semaphore = mmap(NULL, sizeof(sem_t), PROT_READ |PROT_WRITE,
MAP_SHARED|MAP_ANONYMOUS, 0, 0);
    sem_init(semaphore, 1, 0);

    id = fork();

    if (id < 0) {
        perror("fork error");
        return -1;
    }


    // Child Process
    else if (id == 0) {
        sem_wait(semaphore);

        printf("[Child Process,  id=%d]\n", getpid());

        char read_file_name[20];
        strcpy(read_file_name, buffer->filename);

        char* read_sequence_of_numbers;
        read_sequence_of_numbers = (char*)malloc(sizeof(char) * (buffer->num));
        strcpy(read_sequence_of_numbers, buffer->read_num);

        printf("[Child Process,  id=%d]: File name from the pipe: %s\n", getpid(), read_file_name);
        printf("[Child Process,  id=%d]: Numbers from the pipe: %s\n", getpid(),
read_sequence_of_numbers);

        remove(read_file_name); //if we have the same file

        buffer->result = sum_from_char(read_sequence_of_numbers);
```

```
    free(read_sequence_of_numbers);

    FILE *write_res;
    if ((write_res = fopen(read_file_name, "w")) == NULL) {
        printf("Error: can't open file\n");
        exit(1);
    }
    printf("Результат: %d\n", buffer->result);
    fprintf(write_res, "%d", buffer->result);

    sem_post(semaphore);
    fclose(write_res);
    exit(0);

}

//Parent process
else if (id != 0) {
    printf("[Parent Process, id=%d]: Write name of file: ", getpid());
    fgets(file_name, 20, stdin);
    if (file_name[strlen(file_name) - 1] == '\n')
        file_name[strlen(file_name) - 1] = '\0';

    printf("[Parent Process, id=%d]: Write int numbers: ", getpid());
    int c;
    unsigned int i;

    for (i = 0, c = EOF; (c = getchar()) != '\n' && c != EOF; i++) {
        str_ptr[i] = c;

        if (i == MAX_LENGTH) {
            free(str_ptr);
            printf("Слишком много входных данных!\n");
            exit(1);
        }
```

```
            if (i == str_len) {                    // Блок заполнен
                str_len = i + CHUNK_SIZE;
                str_ptr = realloc(str_ptr, str_len);    // Расширяем блок на ещё один килобайт
            }
        }
        str_ptr[i] = '\0';                    // Признак конца строки

        buffer->num = str_len;
        strcpy(buffer->filename, file_name);
        strcpy(buffer->read_num, str_ptr);

        printf("[Parent Process, id=%d] file name is: %s\n", getpid(), buffer->filename);
        sem_post(semaphore);
    }

    free(str_ptr);

    if (munmap(buffer, sizeof(number))!= 0) {
        printf("UnMapping failed\n");
        return 1;
    }

    sem_destroy(semaphore);

    if (munmap(semaphore, sizeof(semaphore))!= 0) {
        printf("UnMapping of semaphore failed\n");
        return 1;
    }

    return 0;
}
```

## *Main2.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/mman.h>
#include <semaphore.h>

typedef struct number{

    int num;
    int st;
    char filename[20];

} number;

int getting_value(sem_t *semaphore)
{
    int s;
    sem_getvalue(semaphore, &s);
    return s;
}

void setting_value(sem_t *semaphore, int n)
{
    while (getting_value(semaphore) < n)
    {
        sem_post(semaphore);
    }
    while (getting_value(semaphore) > n)
    {
        sem_wait(semaphore);
    }
}

int main(int args, char *argv[])
```

```
{
    pid_t id;
    char file_name[20];

    number *buffer = mmap(NULL, sizeof(number), PROT_READ | PROT_WRITE,
MAP_SHARED | MAP_ANONYMOUS, 0, 0);
    if (buffer == NULL) {
        perror("Can't mmap");
        return -1;
    }

    sem_t *semaphore = mmap(NULL, sizeof(sem_t), PROT_READ |PROT_WRITE,
MAP_SHARED|MAP_ANONYMOUS, 0, 0);
    sem_init(semaphore, 1, 2);

    id = fork();

    if (id < 0) {
        perror("fork error");
        return -1;
    }
    // Child Process
    else if (id == 0) {
        int sum = 0;
        while(1)
        {
            while(getting_value(semaphore) == 2)
            {
                continue;
            }
            if (buffer->st == 0)
            {
                sum += buffer->num;
                setting_value(semaphore, 2);
            }
```

```
        else if (buffer->st == 1)
        {
            sum += buffer->num;

            remove(buffer->filename);

            FILE *write_res;
            if ((write_res = fopen(buffer->filename, "w")) == NULL) {
                printf("Error: can't open file\n");
                exit(1);
            }
            fprintf(write_res, "%d", sum);
            buffer->num = sum;
            fclose(write_res);

            setting_value(semaphore, 0);
            exit(0);
        }
    }

}
//Parent process
else {
    printf("[Parent Process, id=%d]: Write name of file: ", getpid());
    fgets(file_name, 20, stdin);
    if (file_name[strlen(file_name) - 1] == '\n')
        file_name[strlen(file_name) - 1] = '\0';

    strcpy(buffer->filename, file_name);

    printf("[Parent Process, id=%d]: Write int numbers: ", getpid());

    while(getting_value(semaphore) != 0)
    {
        char c;
```

```
            scanf("%d%c", &buffer->num, &c);
            if (c == ' ')
            {
                buffer->st = 0;
            }
            if (c == '\n')
            {
                buffer->st = 1;
            }
            setting_value(semaphore, 1);
            while(getting_value(semaphore) == 1)
            {
                continue;
            }
        }
        printf("[Parent Process, id=%d]: Result: %d\n", getpid(), buffer->num);
    }

    if (munmap(buffer, sizeof(number))!= 0) {
        printf("UnMapping Failed\n");
        return 1;
    }
    sem_destroy(semaphore);

    if (munmap(semaphore, sizeof(semaphore))!= 0) {
        printf("UnMapping of semaphore failed\n");
        return 1;
    }
    return 0;
}
```

## Демонстрация работы программы

```
dr0ozd1@Dmitry-Nitro-AN515-45:~/Coding/os_labs/lab_4$ cmake src/
-- The C compiler identification is GNU 11.3.0
```

```
-- The CXX compiler identification is GNU 11.3.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/dr0ozd1/Coding/os_labs/lab_4
dr0ozd1@Dmitry-Nitro-AN515-45:~/Coding/os_labs/lab_4$ make
[ 50%] Building C object CMakeFiles/main2.dir/main2.c.o
[100%] Linking C executable main2
[100%] Built target main2
dr0ozd1@Dmitry-Nitro-AN515-45:~/Coding/os_labs/lab_4$ ./main2
[Parent Process, id=118483]: Write name of file: fileNAME
[Parent Process, id=118483]: Write int numbers: 1 2 3 4 5 6
[Parent Process, id=118483]: Result: 21
dr0ozd1@Dmitry-Nitro-AN515-45:~/Coding/os_labs/lab_4$ ./main2
[Parent Process, id=118531]: Write name of file: fileNAME
[Parent Process, id=118531]: Write int numbers: 99 +1
[Parent Process, id=118531]: Result: 100
dr0ozd1@Dmitry-Nitro-AN515-45:~/Coding/os_labs/lab_4$ ./main2
[Parent Process, id=118606]: Write name of file: fileNAME
[Parent Process, id=118606]: Write int numbers: 100 -1
[Parent Process, id=118606]: Result: 99
dr0ozd1@Dmitry-Nitro-AN515-45:~/Coding/os_labs/lab_4$ ./main2
[Parent Process, id=118690]: Write name of file: fileNAME
[Parent Process, id=118690]: Write int numbers: 0
[Parent Process, id=118690]: Result: 0
```

# Выводы

В ходе проведения данной лабораторной я работы я познакомился с технологией обмена данных file mapping, а также узнал и прочувствовал трудности при работе с ней.