

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: Примаченко Александр Александрович

Группа: М8О-208Б-20

Вариант: 3

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_

Дата: \_\_\_\_\_

Подпись: \_\_\_\_\_

Москва, 2021

## **Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Исходный код
5. Выводы

### Постановка задачи

#### Цель работы

Приобретение практических навыков в:

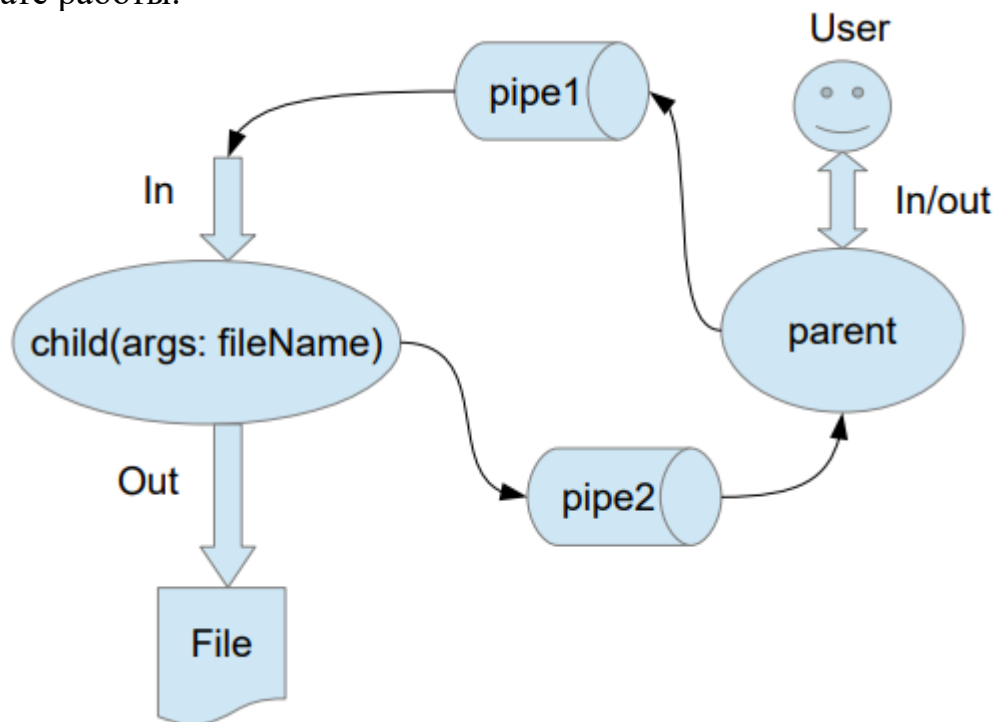
- Освоение принципов работы с файловыми системами
- Обеспечение обмена данными между процессами посредством технологии «File mapping»

#### 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.



## **Общие сведения о программе**

Программа компилируется из файла main.c. Также используется заголовочные файлы: unistd.h, stdio.h, stdlib.h, fcntl.h, errno.h, sys/mman.h, sys/stat.h, string.h, stdbool.h, ctype.h, sys/wait.h, semaphore.h. В программе используются следующие системные вызовы:

1. shm\_open - создаёт/открывает объекты общей памяти POSIX.
2. sem\_open - инициализирует и открывает именованный семафор.
3. ftruncate - обрезает файл до заданного размера.
4. mmap, munmap - отображает файлы или устройства в памяти, или удаляет их отображение.
5. memset - заполнение памяти значением определённого байта.
6. sem\_getvalue - возвращает значение семафора.
7. close - закрывает файловый дескриптор.
8. sem\_close - закрывает именованный семафор.
9. execl - запуск файла на исполнение.
10. sem\_getvalue - возвращает значение семафора.
11. sem\_wait - блокирует семафор.
12. sem\_post - разблокирует семафор.

## Исходный код

### Main.cpp

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

#include "shrmem.h"

using namespace std;

int main(int argc, char **argv) {
    int map_fd = shm_open(BackingFile, O_RDWR, AccessPerms);
    if (map_fd < 0) {
        perror("SHM_OPEN");
        exit(EXIT_FAILURE);
    }
    struct stat statbuf;
    fstat(map_fd, &statbuf);
    const size_t map_size = statbuf.st_size;
    caddr_t memptr = static_cast<char*>(mmap(
        NULL,
        map_size,
        PROT_READ | PROT_WRITE,
        MAP_SHARED,
        map_fd,
        0));
    if (memptr == MAP_FAILED) {
        perror("MMAP");
        exit(EXIT_FAILURE);
    }
    sem_t *semptr = sem_open(SemaphoreName, O_CREAT, AccessPerms, 2);
    if (semptr == SEM_FAILED) {
        perror("SEM_OPEN");
        exit(EXIT_FAILURE);
    }
    if (sem_wait(semptr) != 0) {
        perror("SEM_WAIT");
        exit(EXIT_FAILURE);
    }
}
```

```

char *out = (char *)calloc(1, sizeof(char));
size_t m_size = 0;
int flag = 0;
string first;
string second;

FILE *filename = fopen(argv[1], "w");
if (filename == NULL) cout << "fopen error";

for (int i = 0; i + 1 < map_size; ++i) { // преобразование
    if (flag == 0) {
        first.push_back(memptr[i]);
    }
    else if (flag == 1) {
        second.push_back(memptr[i]);
    }
    if (memptr[i] == ' ' && flag == 0) {
        flag = 1;
    }
    else if ((memptr[i] == ' ' || memptr[i] == '\n') && flag == 1) {
        if (atoi(second.c_str()) == 0) {
            perror("division by zero error\n");
            break;
        }
        first = to_string(atoi(first.c_str())/atoi(second.c_str()));
        second = "";
        if (memptr[i] == '\n') {
            fprintf(filename, "%s\n", first.c_str());
            flag = 0;
            //cout << first << endl;
            first = "";
            second = "";
        }
    }
    //out[m_size] = memptr[i];
    //out = (char *)realloc(out, (++m_size + 1) * sizeof(char));
}
fclose(filename);

out[m_size++] = '\0';
ftruncate(map_fd, (off_t)m_size);
memset(memptr, '\0', m_size);
sprintf(memptr, "%s", out);
free(out);
close(map_fd);
usleep(00150000);
sem_post(semprtr);
sem_close(semprtr);
return EXIT_SUCCESS;
}

```

## Child.cpp

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

#include "shrmem.h"

using namespace std;

int main(int argc, char **argv) {
    int map_fd = shm_open(BackingFile, O_RDWR, AccessPerms);
    if (map_fd < 0) {
        perror("SHM_OPEN");
        exit(EXIT_FAILURE);
    }
    struct stat statbuf;
    fstat(map_fd, &statbuf);
    const size_t map_size = statbuf.st_size;
    caddr_t memptr = static_cast<char*>(mmap(
        NULL,
        map_size,
        PROT_READ | PROT_WRITE,
        MAP_SHARED,
        map_fd,
        0));
    if (memptr == MAP_FAILED) {
        perror("MMAP");
        exit(EXIT_FAILURE);
    }
    sem_t *semptr = sem_open(SemaphoreName, O_CREAT, AccessPerms, 2);
    if (semptr == SEM_FAILED) {
        perror("SEM_OPEN");
        exit(EXIT_FAILURE);
    }
    if (sem_wait(semptr) != 0) {
        perror("SEM_WAIT");
        exit(EXIT_FAILURE);
    }
    char *out = (char *)calloc(1, sizeof(char));
    size_t m_size = 0;
    int flag = 0;
    string first;
    string second;
```

```

FILE *filename = fopen(argv[1], "w");
if (filename == NULL) cout << "fopen error";

for (int i = 0; i + 1 < map_size; ++i) { // преобразование
    if (flag == 0) {
        first.push_back(memptr[i]);
    }
    else if (flag == 1) {
        second.push_back(memptr[i]);
    }
    if (memptr[i] == ' ' && flag == 0) {
        flag = 1;
    }
    else if ((memptr[i] == ' ' || memptr[i] == '\n') && flag == 1) {
        if (atoi(second.c_str()) == 0) {
            perror("division by zero error\n");
            break;
        }
        first = to_string(atoi(first.c_str())/atoi(second.c_str()));
        second = "";
        if (memptr[i] == '\n') {
            fprintf(filename, "%s\n", first.c_str());
            flag = 0;
            //cout << first << endl;
            first = "";
            second = "";
        }
    }
    //out[m_size] = memptr[i];
    //out = (char *)realloc(out, (++m_size + 1) * sizeof(char));
}
fclose(filename);

out[m_size++] = '\0';
ftruncate(map_fd, (off_t)m_size);
memset(memptr, '\0', m_size);
sprintf(memptr, "%s", out);
free(out);
close(map_fd);
usleep(00150000);
sem_post(sempr);
sem_close(sempr);
return EXIT_SUCCESS;
}

```

## Shrmem.h

```

#ifndef SRC__SHRMEM_H_
#define SRC__SHRMEM_H_

```

```

#include <fcntl.h>

```



```
const char *BackingFile = "os_lab4.back";  
const char *SemaphoreName = "os_lab4.semaphore";  
unsigned AccessPerms = S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH;  
#endif//SRC__SHRMEM_H_
```

## **Выводы**

В Си помимо механизма общения между процессами через pipe, также существуют и другие способы взаимодействия, например отображение файла в память, такой подход работает быстрее, за счет отсутствия постоянных вызовов read, write и тратит меньше памяти под кэш. После отображения возвращается void\*, который можно привести к своему указателю на тип и обрабатывать данные как массив, где возвращенный указатель – указатель на первый элемент.