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

# Лабораторная работа №4 по курсу
# «Операционные системы»

Студентка: Шатунова Юлия Викторовна
Группа: М8О-208Б-20
Вариант: 13
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2021

**Содержание**

## Репозиторий

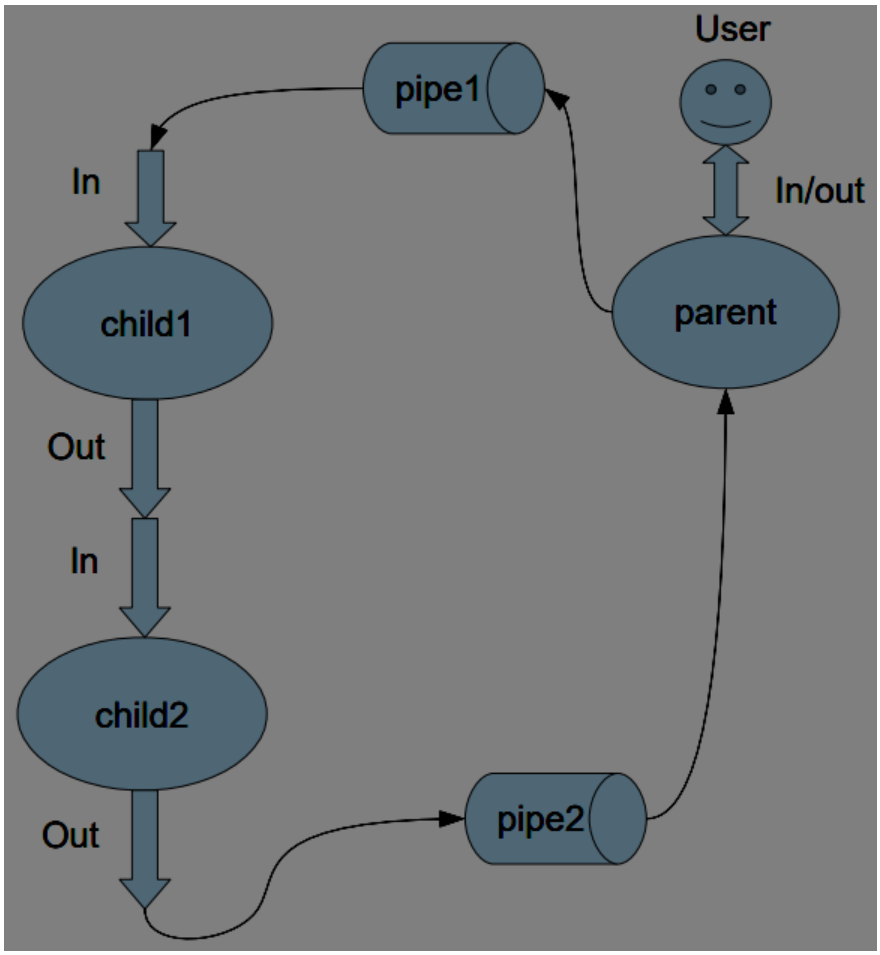
https://github.com/s0bakkaa/OS/tree/main/lab4

## 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.

13 

вариант)
Child1 переводит строки в нижний регистр. Child2 превращает все пробельные символы в символ «_»

## 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.

## Общий метод и алгоритм решения

Для реализации поставленной задачи необходимо выполнить нижеуказанные шаги.

1. Изучить работу с отображением файла в память (mmap и munmap).
2. Изучить работу с процессами (fork).
3. Создать 2 дочерних и 1 родительский процесс.
4. В каждом процессе отобразить файл в память, преобразовать в соответствии с вариантом и снять отображение (mmap, munmap).

## Исходный код

main.c

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

#include "memwork.h"
```

```
int main() {
	size_t map_size = 0;
	char *str_in = (char *) calloc(1, sizeof(char));
    char symbol;
    while ((symbol = getchar()) != EOF) {
	str_in[map_size] = symbol;
		str_in = (char *) realloc(str_in, (++map_size + 1) * sizeof(char));
    }
    str_in[map_size++] = '\0';

    //  определяем обственно создаваемый объект разделяемой памяти для создания или открытия
    int fd = shm_open(BACKING_FILE, O_RDWR | O_CREAT, FILE_MODE);
    if (fd == -1) { //  в случае успеха: возврат неотрицательного дескриптора
	perror("MEM_OPEN");
	exit(EXIT_FAILURE);
    }

    sem_t *sem_ptr;
    //  создаем новый семафор
    //  O_CREAT - флаг, управляющий работой вызова (здесь - создание сема, если тот еще не существует)
    if ((sem_ptr = sem_open(SEMAPHORE_NAME, O_CREAT, FILE_MODE, 2)) == SEM_FAILED) {
	perror ("SEM_OPEN");
	exit(EXIT_FAILURE);
    }

    //  устанавливаем длину обычного файла с файловым дескриптором fd (файл должен быть открыт для записи)
    if (ftruncate(fd, (off_t)map_size) == -1) { //  в случае успеха: возвращает 0
	perror("FTRUNCATE");
	exit(EXIT_FAILURE);
    }
```

```
    caddr_t mem_ptr;
    //  отображение map_size байтов, начиная с позиции 0 файла, определяемого fd, в память, начиная с адреса 0
    //  prot описывает желаемый режим работы памяти
    //      map_shared - тип отражаемого объекта, в данном случае -
    //    разделение использования отображения с ругими процессами
    if ((mem_ptr = mmap(NULL, map_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0)) == MAP_FAILED) {
            perror("MMAP"); //  в случае успеха: указатель отображаемой области
            exit(EXIT_FAILURE);
    }

    int value;
    memset(mem_ptr, '\0', map_size);
    sprintf(mem_ptr, "%s", str_in);
    free(str_in);
    if (sem_getvalue(sem_ptr, &value) != 0) {
            perror("SEM_GETVALUE");
            exit(EXIT_FAILURE);
    }

    while (value++ < 2) {
            if (sem_post(sem_ptr) != 0) { // разлочивает семафор
                    perror("SEM_POST");
                    exit(EXIT_FAILURE);
            }
    }

    while (value-- > 3) {
            if (sem_wait(sem_ptr) != 0) { //  блокировка семафора
                    perror("SEM_WAIT");
                    exit(EXIT_FAILURE);
            }
```

```
}

int pid_0 = 0;
if ((pid_0 = fork()) == 0) { //  если потомок
   //  освобождение отображения в адресном пространстве процесса
        if (munmap(mem_ptr, map_size) == -1) {
                perror("MUNMAP");
                exit(EXIT_FAILURE);
        }
        close(fd);
        if (sem_close(sem_ptr) != 0) {
                perror("SEM_CLOSE");
                exit(EXIT_FAILURE);
        }
        execl("child0", "child0", NULL);
        perror("EXECL");
   exit(EXIT_FAILURE);
}
else if (pid_0 < 0) {
        perror("FORK");
        exit(EXIT_FAILURE);
}

while (true) {
        if (sem_getvalue(sem_ptr, &value) != 0) {
                perror("SEM_GETVALUE");
                        exit(EXIT_FAILURE);
        }
        if (value == 0) {
                if (sem_wait(sem_ptr) == -1) {
                        perror("SEM_WAIT");
                                exit(EXIT_FAILURE);
```

```
                }
                printf("%s", mem_ptr);
                return EXIT_SUCCESS;
        }
    }
}
```

child0.c

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

#include "memwork.h"

int main(int argc, char **argv) {
        int map_fd = shm_open(BACKING_FILE, O_RDWR, FILE_MODE);
        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 mem_ptr = mmap(NULL, map_size, PROT_READ | PROT_WRITE, MAP_SHARED, map_fd,
0);
	if (mem_ptr == MAP_FAILED) {
		perror("MMAP");
		exit(EXIT_FAILURE);
	}

	sem_t *sem_ptr = sem_open(SEMAPHORE_NAME, O_CREAT, FILE_MODE, 2);
	if (sem_ptr == SEM_FAILED) {
		perror("SEM_OPEN");
		exit(EXIT_FAILURE);
	}
	if (sem_wait(sem_ptr) != 0) {
		perror("SEM_WAIT");
		exit(EXIT_FAILURE);
	}

	char *str_in = (char *) malloc(map_size * sizeof(char));
	for (int index = 0; index < map_size; ++index) {
		str_in[index] = tolower(mem_ptr[index]);
	}
	memset(mem_ptr, '\0', map_size);
	sprintf(mem_ptr, "%s", str_in);
	free(str_in);
	pid_t pid = fork();
	if (pid == 0) {
		munmap(mem_ptr, map_size);
		close(map_fd);
		sem_close(sem_ptr);
		execl("child1", "child1", NULL);
		perror("EXECL");
		exit(EXIT_FAILURE);
	}
```

```
        else if (pid == -1) {
                perror("FORK");
                exit(EXIT_FAILURE);
        }
        return EXIT_SUCCESS;
}
```

child1.c

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

#include "memwork.h"

int main(int argc, char **argv) {
   int map_fd = shm_open(BACKING_FILE, O_RDWR, FILE_MODE);
   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 mem_ptr = mmap(NULL, map_size, PROT_READ | PROT_WRITE, MAP_SHARED, map_fd, 0);
```

```
if (mem_ptr == MAP_FAILED) {
        perror("MMAP");
          exit(EXIT_FAILURE);
}

        sem_t *sem_ptr = sem_open(SEMAPHORE_NAME, O_CREAT, FILE_MODE, 2);
if (sem_ptr == SEM_FAILED) {
          perror("SEM_OPEN");
          exit(EXIT_FAILURE);
}
if (sem_wait(sem_ptr) != 0) {
          perror("SEM_WAIT");
          exit(EXIT_FAILURE);
}

char *str_out = (char *) calloc(1, sizeof(char));
size_t m_size = 0;
for (int index = 0; index < map_size; ++index) {
                if (mem_ptr[index] == ' ') {
                        str_out[index] = '_';
                }
                else {
                        str_out[index] = mem_ptr[index];
                }
}
memset(mem_ptr, '\0', map_size);
sprintf(mem_ptr, "%s", str_out);
free(str_out);
close(map_fd);
usleep(00150000);
sem_post(sem_ptr);
sem_close(sem_ptr);
```

```
    return EXIT_SUCCESS;
}
```

## Демонстрация работы программы

```
[yulia@andromeda src]$ cat test.txt
Take Care Of Nature
WASH YOUR HANDS
Beware  Of  PEOPLE
[yulia@andromeda src]$ gcc -pthread -lrt main.c
[yulia@andromeda src]$ gcc -pthread -lrt child0.c -o child0
[yulia@andromeda src]$ gcc -pthread -lrt child1.c -o child1
[yulia@andromeda src]$ ./a.out < test.txt
take_care_of_nature
wash_your_hands
beware__of__people
```

## Выводы

В Си, помимо механизма общения между процессами через pipe, также существуют и другие способы взаимодействия, например, отображение файла в память. Последний подход работает быстрее, за счет отсутствия постоянных вызовов read и write и тратит меньше памяти под кэш.

Семафор — наиболее часто употребляемый метод для синхронизации потоков и для контролирования одновременного доступа множеством потоков / процессов к общей памяти. Взаимодействие между процессами заключается в том, что процессы работают с одним и тем же набором данных и корректируют свое поведение в зависимости от этих данных.