Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

Группа: М8О-211БВ-24

Студент: Рыбин В.В.

Преподаватель: Бахарев В.Д.

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

Дата: 23.10.25

Москва, 2025

**Постановка задачи**

**Вариант 12.**

Child1 переводит строки в верхний регистр. Child2 убирает все задвоенные пробелы.

**Общий метод и алгоритм решения**

Использованные системные вызовы:

**Системные вызовы для работы с разделяемой памятью**

* int shm\_open(const char \*name, int oflag, mode\_t mode)  
  Создает или открывает объект разделяемой памяти. Возвращает файловый дескриптор или -1 при ошибке.
* int ftruncate(int fd, off\_t length)  
  Изменяет размер файла или разделяемой памяти. Возвращает 0 при успехе, -1 при ошибке.
* void \*mmap(void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset)  
  Отображает файл или разделяемую память в адресное пространство процесса. Возвращает указатель на отображенную область или MAP\_FAILED.
* int munmap(void \*addr, size\_t length)  
  Удаляет отображение памяти. Возвращает 0 при успехе, -1 при ошибке.
* int shm\_unlink(const char \*name)  
  Удаляет объект разделяемой памяти. Возвращает 0 при успехе, -1 при ошибке.

**Системные вызовы для работы с семафорами**

* sem\_t \*sem\_open(const char \*name, int oflag, ...)  
  Открывает или создает именованный семафор. Возвращает указатель на семафор или SEM\_FAILED.
* int sem\_wait(sem\_t \*sem)  
  Ожидает семафор (уменьшает значение на 1). Возвращает 0 при успехе, -1 при ошибке.
* int sem\_post(sem\_t \*sem)  
  Освобождает семафор (увеличивает значение на 1). Возвращает 0 при успехе, -1 при ошибке.
* int sem\_close(sem\_t \*sem)  
  Закрывает семафор. Возвращает 0 при успехе, -1 при ошибке.
* int sem\_unlink(const char \*name)  
  Удаляет именованный семафор. Возвращает 0 при успехе, -1 при ошибке.

**Системные вызовы для управления процессами**

* pid\_t fork(void)  
  Создает новый процесс-потомок. Возвращает 0 в потомке, PID потомка в родителе, -1 при ошибке.
* int execv(const char \*path, char \*const argv[])  
  Заменяет текущий процесс новым процессом. Возвращает -1 только при ошибке.
* pid\_t waitpid(pid\_t pid, int \*wstatus, int options)  
  Ожидает завершения указанного процесса. Возвращает PID завершенного процесса или -1.

**Функции для работы с файлами и вводом-выводом**

* ssize\_t write(int fd, const void \*buf, size\_t count)  
  Записывает данные в файловый дескриптор. Возвращает количество записанных байт или -1.
* ssize\_t read(int fd, void \*buf, size\_t count)  
  Читает данные из файлового дескриптора. Возвращает количество прочитанных байт или -1.
* int close(int fd)  
  Закрывает файловый дескриптор. Возвращает 0 при успехе, -1 при ошибке.

В ходе лабораторной работы я создавал объекты разделяемой памяти, отображал их в адресное пространство процессов. Также я создавал несколько процессов, чтобы они могли общаться между собой благодаря той самой разделяемой памяти для обработки входных данных. Чтобы не было гонки данных, я использовал семафор.

**Код программы**

**main.c**

#include <fcntl.h>

#include <stdint.h>

#include <stdbool.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <unistd.h>

#include <sys/fcntl.h>

#include <sys/mman.h>

#include <wait.h>

#include <semaphore.h>

#define SHM\_SIZE 4096

const char SHM\_NAME[] = "shm-name";

const char SEM\_NAME\_SERVER[] = "example-1-sem-server";

const char SEM\_NAME\_CHILD\_1[] = "example-1-sem-child-1";

const char SEM\_NAME\_CHILD\_2[] = "example-1-sem-child-2";

int main() {

    int shm = shm\_open(SHM\_NAME, O\_RDWR, 0600);

    if (shm == -1 && errno != ENOENT) {

        // enoent - ошибка: нет такого файла или каталога

        const char msg[] = "error: failed to open SHM\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    shm = shm\_open(SHM\_NAME, O\_RDWR | O\_CREAT | O\_TRUNC, 0600);

    // O\_RDWR - открыть на запись и на чтение

    // O\_CREAT - создать, если не существует

    // O\_TRUNC - если существовует, то стереть

    // 0600 - права доступа, 6 = 110

    // владелец может читать и писать

    if (shm == -1) {

        // возникла ошибка при создании разделяемой памяти

        const char msg[] = "error: failed to open SHM\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    if (ftruncate(shm, SHM\_SIZE) == -1) {

        // shm\_open просто создал разделяемую память нулевой длины

        // функция ftruncate изменяет размер этой памяти до SHM\_SIZE

        // возвращает -1, если возникла ошибка

        const char msg[] = "error: failed to resize SHM\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    char \*shm\_buf = mmap(NULL, SHM\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm, 0);

    // отображает разделяемую память в адресное пространство процесса

    if (shm\_buf == MAP\_FAILED) {

        const char msg[] = "error: failed to map SHM\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    sem\_t \*sem\_server = sem\_open(SEM\_NAME\_SERVER, O\_RDWR | O\_CREAT | O\_TRUNC, 0600, 1);

    if (sem\_server == SEM\_FAILED) {

        const char msg[] = "error: failed to create semaphore\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    sem\_t \*sem\_client\_1 = sem\_open(SEM\_NAME\_CHILD\_1, O\_RDWR | O\_CREAT | O\_TRUNC, 0600, 0);

    if (sem\_client\_1 == SEM\_FAILED) {

        const char msg[] = "error: failed to create semaphore\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    sem\_t \*sem\_client\_2 = sem\_open(SEM\_NAME\_CHILD\_2, O\_RDWR | O\_CREAT | O\_TRUNC, 0600, 0);

    if (sem\_client\_2 == SEM\_FAILED) {

        const char msg[] = "error: failed to create semaphore\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    pid\_t child\_1 = fork();

    if (child\_1 == 0) {

        // значит мы находимся в дочернем процессе

        char \*args[] = {"child\_1", NULL};

        execv("./child\_1", args);

        const char msg[] = "error: failed to exec\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    else if (child\_1 == -1) {

        const char msg[] = "error: failed to fork\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    pid\_t child\_2 = fork();

    if (child\_2 == 0) {

        // находимся в дочернем процессе

        char \*args[] = {"child\_2", NULL};

        execv("./child\_2", args);

        const char msg[] = "error: failed to exec\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    else if (child\_2 == -1) {

        // ошибка при создании процесса

        const char msg[] = "error: failed to fork\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    // находимся в родительском процессе

    bool running = true;

    while (running) {

        sem\_wait(sem\_server);

        uint32\_t \*length = (uint32\_t \*)shm\_buf;

        char \*text = shm\_buf + sizeof(uint32\_t);

        if (\*length == UINT32\_MAX) {

            // завершение программы

            running = false;

        }

        else if (\*length > 0) {

            // значит данные получены от child\_2

            const char msg[] = "Result: ";

            write(STDOUT\_FILENO, msg, sizeof(msg) - 1);

            write(STDOUT\_FILENO, text, \*length);

            \*length = 0;

            sem\_post(sem\_server);

        }

        else {

            // length == 0

            // значит обработанных данных нет, надо получить данные от пользователя

            const char msg[] = "Enter text (Ctrl+D for exit): ";

            write(STDOUT\_FILENO, msg, sizeof(msg) - 1);

            char buf[SHM\_SIZE - sizeof(uint32\_t)];

            ssize\_t bytes = read(STDIN\_FILENO, buf, sizeof(buf));

            if (bytes == -1) {

                const char error[] = "error: failed to read from standart input\n";

                write(STDERR\_FILENO, error, sizeof(error));

                \_exit(EXIT\_FAILURE);

            }

            if (bytes > 0) {

                \*length = bytes;

                memcpy(text, buf, bytes);

                sem\_post(sem\_client\_1);

            }

            else {

                \*length = UINT32\_MAX;

                running = false;

                sem\_post(sem\_client\_1);

            }

        }

    }

    waitpid(child\_1, NULL, 0);

    waitpid(child\_2, NULL, 0);

    sem\_unlink(SEM\_NAME\_SERVER);

    sem\_unlink(SEM\_NAME\_CHILD\_1);

    sem\_unlink(SEM\_NAME\_CHILD\_2);

    sem\_close(sem\_server);

    sem\_close(sem\_client\_1);

    sem\_close(sem\_client\_2);

    munmap(shm\_buf, SHM\_SIZE);

    shm\_unlink(SHM\_NAME);

    close(shm);

    return 0;

}

**child\_1.c**

#include <fcntl.h>

#include <stdint.h>

#include <stdbool.h>

#include <ctype.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <unistd.h>

#include <sys/fcntl.h>

#include <sys/mman.h>

#include <wait.h>

#include <semaphore.h>

#define SHM\_SIZE 4096

const char SHM\_NAME[] = "shm-name";

const char SEM\_NAME\_SERVER[] = "example-1-sem-server";

const char SEM\_NAME\_CHILD\_1[] = "example-1-sem-child-1";

const char SEM\_NAME\_CHILD\_2[] = "example-1-sem-child-2";

int main() {

    int shm = shm\_open(SHM\_NAME, O\_RDWR, 0600);

    if (shm == -1) {

        const char msg[] = "error: failed to open SHM\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    char \*shm\_buf = mmap(NULL, SHM\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm, 0);

    if (shm\_buf == MAP\_FAILED) {

        const char msg[] = "error: failed to map\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    sem\_t \*sem\_client\_1 = sem\_open(SEM\_NAME\_CHILD\_1, O\_RDWR);

    if (sem\_client\_1 == SEM\_FAILED) {

        const char msg[] = "error: failed to open semaphore\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    sem\_t \*sem\_client\_2 = sem\_open(SEM\_NAME\_CHILD\_2, O\_RDWR);

    if (sem\_client\_2 == SEM\_FAILED) {

        const char msg[] = "error: failed to open semaphore\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    bool running = true;

    while (running) {

        sem\_wait(sem\_client\_1);

        uint32\_t \*length = (uint32\_t \*)shm\_buf;

        char \*text = shm\_buf + sizeof(uint32\_t);

        if (\*length == UINT32\_MAX) {

            running = false;

        }

        else if (\*length > 0) {

            // переводим в верхний регистр

            for (uint32\_t i = 0; i < \*length; ++i) {

                text[i] = toupper(text[i]);

            }

        }

        sem\_post(sem\_client\_2);

    }

    sem\_close(sem\_client\_1);

    sem\_close(sem\_client\_2);

    munmap(shm\_buf, SHM\_SIZE);

    close(shm);

    return 0;

}

**child\_2.c**

#include <fcntl.h>

#include <stdint.h>

#include <stdbool.h>

#include <ctype.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include <unistd.h>

#include <sys/fcntl.h>

#include <sys/mman.h>

#include <wait.h>

#include <semaphore.h>

#define SHM\_SIZE 4096

const char SHM\_NAME[] = "shm-name";

const char SEM\_NAME\_SERVER[] = "example-1-sem-server";

const char SEM\_NAME\_CHILD\_1[] = "example-1-sem-child-1";

const char SEM\_NAME\_CHILD\_2[] = "example-1-sem-child-2";

int main() {

    int shm = shm\_open(SHM\_NAME, O\_RDWR, 0600);

    if (shm == -1) {

        const char msg[] = "error: failed to open SHM\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    char \*shm\_buf = mmap(NULL, SHM\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm, 0);

    if (shm\_buf == MAP\_FAILED) {

        const char msg[] = "error: failed to map\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    sem\_t \*sem\_client\_2 = sem\_open(SEM\_NAME\_CHILD\_2, O\_RDWR);

    if (sem\_client\_2 == SEM\_FAILED) {

        const char msg[] = "error: failed to open semaphore\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    sem\_t \*sem\_server = sem\_open(SEM\_NAME\_SERVER, O\_RDWR);

    if (sem\_server == SEM\_FAILED) {

        const char msg[] = "error: failed to open semaphore\n";

        write(STDERR\_FILENO, msg, sizeof(msg));

        \_exit(EXIT\_FAILURE);

    }

    bool running = true;

    while (running) {

        sem\_wait(sem\_client\_2);

        uint32\_t \*length = (uint32\_t \*)shm\_buf;

        char \*text = shm\_buf + sizeof(uint32\_t);

        if (\*length == UINT32\_MAX) {

            running = false;

        }

        else if (\*length > 0) {

            // переводим в верхний регистр

            uint32\_t new\_length = 0;

            bool prev\_symbol\_is\_space = 0;

            for (uint32\_t i = 0; i < \*length; ++i) {

                if (text[i] == ' ') {

                    if (!prev\_symbol\_is\_space) {

                        text[new\_length++] = text[i];

                        prev\_symbol\_is\_space = true;

                    }

                }

                else {

                    text[new\_length++] = text[i];

                    prev\_symbol\_is\_space = false;

                }

            }

            \*length = new\_length;

        }

        sem\_post(sem\_server);

    }

    sem\_close(sem\_client\_2);

    sem\_close(sem\_server);

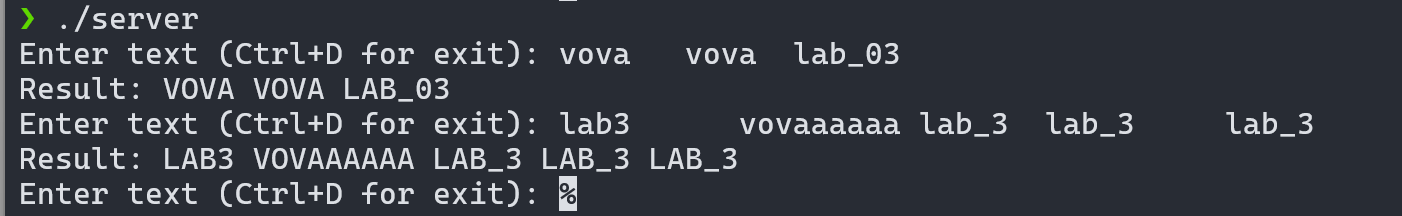
    munmap(shm\_buf, SHM\_SIZE);

    close(shm);

    return 0;

}

**Протокол работы программы**



**Вывод**

В ходе лабораторной работы я приобрел практические навыки при работе с разделяемой памятью, с ее отображением в адресное пространство процесса, а также потренировался в использовании семафоров.