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: Мутагиров Т.Р.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 23.10.25

Москва, 2025

Постановка задачи

Вариант 14.

Child1 переводит строки в нижний 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

client.c

```c
#include <fcntl.h>
#include <stdint.h>
#include <stdio.h>
#include <stdbool.h>

#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include <unistd.h>
#include <sys/fcntl.h>
#include <wait.h>
#include <semaphore.h>
#include <sys/mman.h>

#define SHM_SIZE 4096

char CHILD_PROGRAM_NAME[] = "child";

int main() {
    char shm_name[256], sem_name[256];
    snprintf(shm_name, 256, "client_shm%d", getpid());
    snprintf(sem_name, 256, "client_sem%d", getpid());
```

```
int shm = shm_open(shm_name, O_RDWR | O_CREAT | O_TRUNC, 0600);
if(shm == -1) {
    char msg[] = "error:failed to create/open SHM\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    exit(EXIT_FAILURE);
}

if(ftruncate(shm, SHM_SIZE) == -1) {
    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_WRITE, MAP_SHARED, shm, 0);
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 = sem_open(sem_name, O_RDWR | O_CREAT | O_TRUNC, 0600, 1);
if (sem == SEM_FAILED) {
    const char msg[] = "error: failed to create semaphore\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    exit(EXIT_FAILURE);
}

int *stage = (int *)(shm_buf + sizeof(uint32_t));
uint32_t *len = (uint32_t *)shm_buf;
char *text = shm_buf + sizeof(uint32_t) + sizeof(int);
*stage = 0; *len = 0;

const pid_t child1 = fork();
if(child1 == -1) {
    const char msg[] = "error: failed to spawn new process\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    exit(EXIT_FAILURE);
} else if(child1 == 0) {
    char *argv[] = {"child", "1", shm_name, sem_name, NULL};
    execv("./child", argv);
    const char msg[] = "error: failed to exec\n";
```

```
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    const pid_t child2 = fork();
    if(child2 == -1) {
        const char msg[] = "error: failed to spawn new process\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    } else if(child2 == 0) {
        char *argv[] = {"child", "0", shm_name, sem_name, NULL};
        execv("./child", argv);
        const char msg[] = "error: failed to exec\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    // client
    pid_t pid = getpid();
    size_t bytes;
    while(true) {
        char buf[SHM_SIZE - sizeof(uint32_t)];
        bytes = read(STDIN_FILENO, buf, sizeof(buf));
        if(*len == -1) {
            const char msg[] = "error: failed to read from standard input\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            _exit(EXIT_FAILURE);
        }
        sem_wait(sem);
        *len = bytes;
        if(*len > 0) {
            memcpy(text, buf, *len);
        } else {
            *len = UINT32_MAX;
            sem_post(sem);
            break;
        }
        *stage = 1; // дальше обрабатывать будет child1
        sem_post(sem);

        // вывод после child2
        // ждем пока child2 не закончит работу
```

```
        while (true) {
            sem_wait(sem);
            if(*stage == 3) {
                // child2 законил обрабоотку
                write(STDOUT_FILENO, text, *len);
                *stage = 0;
                sem_post(sem);
                break;
            }
            sem_post(sem);
        }
    }
    waitpid(child1, NULL, 0);
    waitpid(child2, NULL, 0);

    sem_unlink(sem_name);
    sem_close(sem);
    munmap(shm_buf, SHM_SIZE);
    shm_unlink(shm_name);
    close(shm);
    return 0;
}
```

child.c

```
#include <fcntl.h>
#include <stdint.h>
#include <stdbool.h>
#include <ctype.h>

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>

#include <unistd.h>
#include <sys/fcntl.h>
#include <wait.h>
#include <semaphore.h>
#include <sys/mman.h>

#define SHM_SIZE 4096

int main(int argc, const char **argv) {
```

```
    bool first = argv[1][0] - '0';
    int shm = shm_open(argv[2], O_RDWR, 0);
    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 SHM\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        _exit(EXIT_FAILURE);
    }

    sem_t *sem = sem_open(argv[3], O_RDWR);
    if (sem == SEM_FAILED) {
        const char msg[] = "error: failed to open semaphore\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        _exit(EXIT_FAILURE);
    }

    int *stage = (int *)(shm_buf + sizeof(uint32_t));
    uint32_t *len = (uint32_t *)shm_buf;
    char *text = shm_buf + sizeof(uint32_t) + sizeof(int);

    char nbuf[4096];
    while(true) {
        // преобразить buf
        sem_wait(sem);
        if(*len == UINT32_MAX) {
            sem_post(sem);
            break;
        }
        if(first) { // child 1
            if(*stage == 1) {
                for(size_t i = 0; i < *len; ++i) text[i] = tolower(text[i]);
                *stage = 2; // дальше child2
            }
        } else { // child 2
            if(*stage == 2) {
```

```
                size_t nsz = 0;
                for(size_t i = 0; i < *len; ++i) {
                    if(text[i] != ' ' || i == *len - 1 || text[i + 1] != ' ') {
                        nbuf[nsz++] = text[i];
                    }
                }
                *len = nsz;
                memcpy(text, nbuf, *len);
                *stage = 3;
            }
        }
        sem_post(sem);
    }
    sem_close(sem);
    munmap(shm_buf, SHM_SIZE);
    close(shm);
    return 0;
}
```

Протокол работы программы

```
rinrow@DESKTOP-S9IRGBR:~/2sem/OS/OS_labs/3$ ./client
1026: I'm a Parent
HELLO MAI FIIt                          and World
hello mai fiit and world


 ---------------
 ---------------
Bye
bye
```

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.