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-210БВ-24

Студент: Лабутин М.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 02.11.25

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

server.c

```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>

#include <stdio.h>



#define SHM_SIZE 4096

#define BUFFER_SIZE 1024

char SHM_NAME[1024];

char SEM_PARENT[1024];

char SEM_CHILD1[1024];

char SEM_CHILD2[1024];

int main() {
```

```
    char unique_suffix[64];
    snprintf(unique_suffix, sizeof(unique_suffix), "%d", getpid());

    snprintf(SHM_NAME, sizeof(SHM_NAME), "shm-%s", unique_suffix);
    snprintf(SEM_PARENT, sizeof(SEM_PARENT), "sem-parent-%s", unique_suffix);
    snprintf(SEM_CHILD1, sizeof(SEM_CHILD1), "sem-child1-%s", unique_suffix);
    snprintf(SEM_CHILD2, sizeof(SEM_CHILD2), "sem-child2-%s", unique_suffix);

    int shm_fd = shm_open(SHM_NAME, O_RDWR | O_CREAT | O_EXCL, 0600);
    if (shm_fd == -1) {
        perror("shm_open");
        exit(EXIT_FAILURE);
    }

    if (ftruncate(shm_fd, SHM_SIZE) == -1) {
        perror("ftruncate");
        exit(EXIT_FAILURE);
    }

    char *shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        perror("mmap");
        exit(EXIT_FAILURE);
    }

    uint32_t *data_size = (uint32_t *)shm_ptr;
    *data_size = 0;

    sem_t *sem_parent = sem_open(SEM_PARENT, O_CREAT | O_EXCL, 0600, 1);
    if (sem_parent == SEM_FAILED) {
        perror("sem_open (parent)");
        exit(EXIT_FAILURE);
    }

    sem_t *sem_child1 = sem_open(SEM_CHILD1, O_CREAT | O_EXCL, 0600, 0);
    if (sem_child1 == SEM_FAILED) {
```

```
        perror("sem_open (child1)");
        exit(EXIT_FAILURE);
    }

    sem_t *sem_child2 = sem_open(SEM_CHILD2, O_CREAT | O_EXCL, 0600, 0);
    if (sem_child2 == SEM_FAILED) {
        perror("sem_open (child2)");
        exit(EXIT_FAILURE);
    }

    pid_t pid1 = fork();
    if (pid1 == 0) {
        char *args[] = {"./child1", SHM_NAME, SEM_PARENT, SEM_CHILD1, SEM_CHILD2, NULL};
        execv("./child1", args);
        perror("execv child1");
        exit(EXIT_FAILURE);
    } else if (pid1 < 0) {
        perror("fork child1");
        exit(EXIT_FAILURE);
    }

    pid_t pid2 = fork();
    if (pid2 == 0) {
        char *args[] = {"./child2", SHM_NAME, SEM_PARENT, SEM_CHILD1, SEM_CHILD2, NULL};
        execv("./child2", args);
        perror("execv child2");
        exit(EXIT_FAILURE);
    } else if (pid2 < 0) {
        perror("fork child2");
        exit(EXIT_FAILURE);
    }

    bool running = true;
    char buffer[BUFFER_SIZE];

    printf("Enter text (Ctrl+D for exit): ");
```

```
        fflush(stdout);


        while (running) {
            sem_wait(sem_parent);


            data_size = (uint32_t *)shm_ptr;
            char *data = shm_ptr + sizeof(uint32_t);


            if (*data_size == UINT32_MAX) {
                running = false;
                sem_post(sem_child1);
            } else if (*data_size > 0) {
                printf("Result: %.*s\n\n", *data_size, data);
                *data_size = 0;


                printf("Enter text (Ctrl+D for exit): ");
                fflush(stdout);


                if (fgets(buffer, BUFFER_SIZE, stdin) != NULL) {
                    size_t len = strlen(buffer);
                    if (len > 0 && buffer[len-1] == '\n') {
                        buffer[len-1] = '\0';
                        len--;
                    }
                    *data_size = len;
                    memcpy(data, buffer, len);
                    sem_post(sem_child1);
                } else {
                    *data_size = UINT32_MAX;
                    running = false;
                    sem_post(sem_child1);
                }
            } else {
                if (fgets(buffer, BUFFER_SIZE, stdin) != NULL) {
                    size_t len = strlen(buffer);
                    if (len > 0 && buffer[len-1] == '\n') {
```

```
                        buffer[len-1] = '\0';
                        len--;
                    }
                    *data_size = len;
                    memcpy(data, buffer, len);
                    sem_post(sem_child1);
                } else {
                    *data_size = UINT32_MAX;
                    running = false;
                    sem_post(sem_child1);
                }
            }
        }
        sleep(1);

        waitpid(pid1, NULL, 0);
        waitpid(pid2, NULL, 0);

        sem_close(sem_parent);
        sem_close(sem_child1);
        sem_close(sem_child2);

        sem_unlink(SEM_PARENT);
        sem_unlink(SEM_CHILD1);
        sem_unlink(SEM_CHILD2);

        munmap(shm_ptr, SHM_SIZE);
        shm_unlink(SHM_NAME);
        close(shm_fd);

        return EXIT_SUCCESS;
    }
```

child_1.c

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

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


#define SHM_SIZE 4096
int main(int argc, char *argv[]) {
    if (argc != 5) {
        return EXIT_FAILURE;
    }

    const char *SHM_NAME = argv[1];
    const char *SEM_PARENT = argv[2];
    const char *SEM_CHILD1 = argv[3];
    const char *SEM_CHILD2 = argv[4];

    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0600);
    if (shm_fd == -1) {
        return EXIT_FAILURE;
    }

    char *shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        return EXIT_FAILURE;
    }

    sem_t *sem_parent = sem_open(SEM_PARENT, O_RDWR);
    if (sem_parent == SEM_FAILED) {
        return EXIT_FAILURE;
    }

    sem_t *sem_child1 = sem_open(SEM_CHILD1, O_RDWR);
```

```
    if (sem_child1 == SEM_FAILED) {
        return EXIT_FAILURE;
    }

    sem_t *sem_child2 = sem_open(SEM_CHILD2, O_RDWR);
    if (sem_child2 == SEM_FAILED) {
        return EXIT_FAILURE;
    }

    bool running = true;

    while (running) {
        sem_wait(sem_child1);

        uint32_t *data_size = (uint32_t *)shm_ptr;
        char *data = shm_ptr + sizeof(uint32_t);

        if (*data_size == UINT32_MAX) {
            running = false;
            sem_post(sem_child2);
        } else if (*data_size > 0) {
            for (uint32_t i = 0; i < *data_size; i++) {
                data[i] = toupper(data[i]);
            }
            sem_post(sem_child2);
        }
    }

    sem_close(sem_parent);
    sem_close(sem_child1);
    sem_close(sem_child2);

    munmap(shm_ptr, SHM_SIZE);
    close(shm_fd);

    return EXIT_SUCCESS;
```

```
}
```

child_2.c

```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 <semaphore.h>


#define SHM_SIZE 4096
int main(int argc, char *argv[]) {
    if (argc != 5) {
        return EXIT_FAILURE;
    }

    const char *SHM_NAME = argv[1];
    const char *SEM_PARENT = argv[2];
    const char *SEM_CHILD1 = argv[3];
    const char *SEM_CHILD2 = argv[4];

    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0600);
    if (shm_fd == -1) {
        return EXIT_FAILURE;
    }

    char *shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        return EXIT_FAILURE;
    }

    sem_t *sem_parent = sem_open(SEM_PARENT, O_RDWR);
```

```
    if (sem_parent == SEM_FAILED) {
        return EXIT_FAILURE;
    }

    sem_t *sem_child1 = sem_open(SEM_CHILD1, O_RDWR);
    if (sem_child1 == SEM_FAILED) {
        return EXIT_FAILURE;
    }

    sem_t *sem_child2 = sem_open(SEM_CHILD2, O_RDWR);
    if (sem_child2 == SEM_FAILED) {
        return EXIT_FAILURE;
    }

    bool running = true;

    while (running) {
        sem_wait(sem_child2);

        uint32_t *data_size = (uint32_t *)shm_ptr;
        char *data = shm_ptr + sizeof(uint32_t);

        if (*data_size == UINT32_MAX) {
            running = false;
            sem_post(sem_parent);
        } else if (*data_size > 0) {
            char *src = data;
            char *dst = data;
            int space_count = 0;
            int in_space_sequence = 0;

            for (uint32_t i = 0; i < *data_size; i++) {
                if (isspace((unsigned char)src[i])) {
                    if (!in_space_sequence) {
                        in_space_sequence = 1;
                        space_count = 1;
```

```
                    } else {
                        space_count++;
                    }
                } else {
                    if (in_space_sequence) {
                        if (space_count % 2 != 0) {
                            *dst++ = ' ';
                        }
                        in_space_sequence = 0;
                        space_count = 0;
                    }
                    *dst++ = src[i];
                }
            }

            if (in_space_sequence && space_count % 2 != 0) {
                *dst++ = ' ';
            }

            *data_size = dst - data;
            sem_post(sem_parent);
        }
    }

    sem_close(sem_parent);
    sem_close(sem_child1);
    sem_close(sem_child2);

    munmap(shm_ptr, SHM_SIZE);
    close(shm_fd);

    return EXIT_SUCCESS;
}
```

Протокол работы программы

```
mxtv1x@WIN-EMVSNHTNF8D:/mnt/c/Users/User/MAI-LABS/OS-25/lab3$ ./server
Enter text (Ctrl+D for exit): hi  hi  hi  ha
Result: HIHIHIHA

Enter text (Ctrl+D for exit): qwerty 12345 lalala
Result: QWERTY 12345 LALALA
```

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.