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-212БВ-24

Студент: Сучков В.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 13.12.25

Москва, 2025

Постановка задачи

Вариант 10.

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Родительский и дочерний процесс должны быть представлены разными программами.

Необходимо использовать shared memory и memory mapping. Для синхронизации чтения и записи из shared memory использовать семафор.

Правило фильтрации: В файле записаны команды вида: «число<endline>». Дочерний процесс производит проверку этого числа на простоту. Если число составное, то дочерний процесс пишет это число в стандартный поток вывода. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- shm_open() и ftruncate(): Создание и установка размера общей памяти.
- mmap(): Отображение SHM в виртуальное адресное пространство процесса.
- sem_open(): Создание и инициализация семафоров.
- sem_wait() и sem_post(): Синхронизация доступа в цикле.
- shm_unlink() и sem_unlink(): Очистка после работы.

Алгоритм решения:

Запуск сервера:

- Получение session_id
- Генерация имен ресурсов
- Создание shared memory
- Инициализация shared memory
- Создание семафоров

Запуск клиента:

- Получение параметров
- Генерация тех же имен
- Ожидание готовности сервера
- Открытие файла
- Работа с файлом

common.h

```
#ifndef COMMON_H

#define COMMON_H
```

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

#define BUFFER_SIZE 4096
#define SHM_NAME_PREFIX "/shm_ipc_"
#define SEM_CLIENT_PREFIX "/sem_client_"
#define SEM_SERVER_PREFIX "/sem_server_"

typedef struct {
    char data[BUFFER_SIZE];
    size_t data_size;
    bool client_ready;
    bool server_ready;
    bool terminate;
} shared_data_t;



static inline void generate_shared_name(char* buffer, size_t
size, const char* prefix, const char* session_id) {
    if (buffer && prefix && session_id) {
        int written = snprintf(buffer, size, "%s%s", prefix,
session_id);
        if (written >= (int)size || written < 0) {
            buffer[size - 1] = '\0';
```

```
        }
    }
}


static inline bool is_valid_session_id(const char* session_id) {
    if (!session_id || strlen(session_id) == 0) {
        return false;
    }

    for (size_t i = 0; session_id[i] != '\0'; i++) {
        char c = session_id[i];
        if (!((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')
|| (c >= '0' && c <= '9') || c == '_' || c == '-')) {
            return false;
        }
    }

    return true;
}


static inline void init_shared_data(shared_data_t* shared_data)
{
    if (shared_data) {
        memset(shared_data, 0, sizeof(shared_data_t));
        shared_data->server_ready = true;
    }
}


static inline void create_shm_name(char* buffer, size_t size,
const char* session_id) {
    generate_shared_name(buffer, size, SHM_NAME_PREFIX,
session_id);
}
```

```
static inline void create_sem_client_name(char* buffer, size_t
size, const char* session_id) {

    generate_shared_name(buffer, size, SEM_CLIENT_PREFIX,
session_id);

}



static inline void create_sem_server_name(char* buffer, size_t
size, const char* session_id) {

    generate_shared_name(buffer, size, SEM_SERVER_PREFIX,
session_id);

}



static inline void cleanup_resources(const char* shm_name, const
char* sem_client_name, const char* sem_server_name, void*
mapped_data,

                                     int shm_fd, void* sem_client,
void* sem_server) {

    if (sem_client) {

        sem_close((sem_t*)sem_client);

    }

    if (sem_server) {

        sem_close((sem_t*)sem_server);

    }



    if (sem_client_name) {

        sem_unlink(sem_client_name);

    }

    if (sem_server_name) {

        sem_unlink(sem_server_name);

    }



    if (mapped_data && mapped_data != MAP_FAILED) {

        munmap(mapped_data, sizeof(shared_data_t));

    }
```

```
    if (shm_fd >= 0) {
        close(shm_fd);
    }

    if (shm_name) {
        shm_unlink(shm_name);
    }
}

#endif
```

server.c

```
#include "common.h"


int is_composite(int n) {
    if (n <= 3) return 0;
    if (n % 2 == 0 || n % 3 == 0) return 1;

    for (int i = 5; i * i <= n; i += 6) {
        if (n % i == 0 || n % (i + 2) == 0)
            return 1;
    }
    return 0;
}


void process_data(const char* input, size_t input_size, char*
output, size_t* output_size) {
    int number = 0;
    int is_negative = 0;
    int has_digits = 0;
    *output_size = 0;
```

```
    for (size_t i = 0; i < input_size; i++) {
        char c = input[i];
        if (c >= '0' && c <= '9') {
            number = number * 10 + (c - '0');
            has_digits = 1;
        } else if (c == '-' && !has_digits) {
            is_negative = 1;
        } else if ((c == '\n' || i == input_size - 1) &&
has_digits) {

            if (is_negative) {
                number = -number;
            }

            if (!is_composite(number)) {
                break;
            } else {
                char num_str[32];
                int len = 0;
                int n = number;
                char temp[32];
                int temp_len = 0;
                while (n > 0) {
                    temp[temp_len++] = '0' + (n % 10);
                    n /= 10;
                }
                for (int j = temp_len - 1; j >= 0; j--) {
                    num_str[len++] = temp[j];
                }
                num_str[len++] = '\n';

                if (*output_size + len <= BUFFER_SIZE) {
```

```
                    memcpy(output + *output_size, num_str, len);
                    *output_size += len;
                } else {
                    break;
                }
            }

            number = 0;
            is_negative = 0;
            has_digits = 0;
        }
    }
}

int main(int argc, char** argv) {
    if (argc != 2) {
        const char msg[] = "Ввод должен быть вида: server
<session_id>\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    const char* session_id = argv[1];

    if (!is_valid_session_id(session_id)) {
        const char msg[] = "Некорректный ввод. Можно
использовать буквы, цифры, ниж подчеркивание и дефис.\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    char shm_name[256];
    char sem_client_name[256];
```

```
    char sem_server_name[256];



    create_shm_name(shm_name, sizeof(shm_name), session_id);

    create_sem_client_name(sem_client_name,
sizeof(sem_client_name), session_id);

    create_sem_server_name(sem_server_name,
sizeof(sem_server_name), session_id);



    int shm_fd = shm_open(shm_name, O_CREAT | O_RDWR, 0666);

    if (shm_fd == -1) {

        perror("shm_open failed");

        exit(EXIT_FAILURE);

    }



    if (ftruncate(shm_fd, sizeof(shared_data_t)) == -1) {

        perror("ftruncate failed");

        cleanup_resources(shm_name, sem_client_name,
sem_server_name, NULL, shm_fd, NULL, NULL);

        exit(EXIT_FAILURE);

    }



    shared_data_t* shared_data = mmap(NULL,
sizeof(shared_data_t), PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);

    if (shared_data == MAP_FAILED) {

        perror("mmap failed");

        cleanup_resources(shm_name, sem_client_name,
sem_server_name, NULL, shm_fd, NULL, NULL);

        exit(EXIT_FAILURE);

    }



    init_shared_data(shared_data);



    sem_t* sem_client = sem_open(sem_client_name, O_CREAT, 0666,
0);
```

```
    sem_t* sem_server = sem_open(sem_server_name, O_CREAT, 0666,
0);



    if (sem_client == SEM_FAILED || sem_server == SEM_FAILED) {

        perror("sem_open failed");

        cleanup_resources(shm_name, sem_client_name,
sem_server_name, shared_data, shm_fd, sem_client, sem_server);

        exit(EXIT_FAILURE);

    }



    char info_msg[256];

    int info_len = snprintf(info_msg, sizeof(info_msg),
"Session_id: %s\n" "Shared memory: %s\n" "Ожидание
клиента...\n", session_id, shm_name);

    write(STDOUT_FILENO, info_msg, info_len);



    while (!shared_data->terminate) {

        if (sem_wait(sem_client) == -1) {

            if (errno == EINTR) continue;

            perror("sem_wait failed");

            break;

        }



        if (shared_data->terminate) {

            break;

        }



        if (!shared_data->client_ready) {

            if (sem_post(sem_server) == -1) {

                perror("sem_post failed");

                break;

            }

            continue;

        }
```

```
        if (shared_data->data_size > 0) {
            char output[BUFFER_SIZE];
            size_t output_size;

            process_data(shared_data->data, shared_data->data_size, output, &output_size);

            if (output_size > 0) {
                memcpy(shared_data->data, output, output_size);
                shared_data->data_size = output_size;
            } else {
                shared_data->data_size = 0;
            }
        }

        shared_data->client_ready = false;

        if (sem_post(sem_server) == -1) {
            perror("sem_post failed");
            break;
        }
    }

    cleanup_resources(shm_name, sem_client_name, sem_server_name, shared_data, shm_fd, sem_client, sem_server);

    write(STDOUT_FILENO, "Server terminated\n", 19);
    return EXIT_SUCCESS;
}
```

client.c

```
#include "common.h"
```

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

    if (argc != 3) {

        const char msg[] = "Ввод должен быть вида: client
<session_id> <filename>\n";

        write(STDERR_FILENO, msg, sizeof(msg) - 1);

        exit(EXIT_FAILURE);

    }



    const char* session_id = argv[1];

    const char* filename = argv[2];



    if (!is_valid_session_id(session_id)) {

        const char msg[] = "Некорректный ввод. Можно
использовать буквы, цифры, ниж подчеркивание и дефис.\n";

        write(STDERR_FILENO, msg, sizeof(msg) - 1);

        exit(EXIT_FAILURE);

    }



    char shm_name[256];

    char sem_client_name[256];

    char sem_server_name[256];



    create_shm_name(shm_name, sizeof(shm_name), session_id);

    create_sem_client_name(sem_client_name,
sizeof(sem_client_name), session_id);

    create_sem_server_name(sem_server_name,
sizeof(sem_server_name), session_id);



    int shm_fd = shm_open(shm_name, O_RDWR, 0666);

    if (shm_fd == -1) {

        perror("shm_open failed");

        exit(EXIT_FAILURE);

    }
```

```
    shared_data_t* shared_data = mmap(NULL,
sizeof(shared_data_t), PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);

    if (shared_data == MAP_FAILED) {

        perror("mmap failed");

        cleanup_resources(NULL, NULL, NULL, NULL, shm_fd, NULL,
NULL);

        exit(EXIT_FAILURE);

    }



    sem_t* sem_client = sem_open(sem_client_name, 0);

    sem_t* sem_server = sem_open(sem_server_name, 0);



    if (sem_client == SEM_FAILED || sem_server == SEM_FAILED) {

        perror("sem_open failed");

        cleanup_resources(shm_name, sem_client_name,
sem_server_name, shared_data, shm_fd, sem_client, sem_server);

        exit(EXIT_FAILURE);

    }



    int max_wait = 100; // 100 * 0.1s = 10 секунд

    while (!shared_data->server_ready && max_wait-- > 0) {

        usleep(100000); // 100ms

    }



    if (!shared_data->server_ready) {

        const char msg[] = "Ошибка. Сервер не готов спустя 10
секунд\n";

        write(STDERR_FILENO, msg, sizeof(msg) - 1);

        cleanup_resources(NULL, NULL, NULL, shared_data, shm_fd,
sem_client, sem_server);

        exit(EXIT_FAILURE);

    }



    shared_data->client_ready = false;
```

```
    if (sem_post(sem_client) == -1) {
        perror("sem_post failed");
        cleanup_resources(NULL, NULL, NULL, shared_data, shm_fd,
sem_client, sem_server);
        exit(EXIT_FAILURE);
    }

    if (sem_wait(sem_server) == -1) {
        perror("sem_wait failed");
        cleanup_resources(NULL, NULL, NULL, shared_data, shm_fd,
sem_client, sem_server);
        exit(EXIT_FAILURE);
    }

    int file_fd = open(filename, O_RDONLY);
    if (file_fd == -1) {
        char error_msg[256];
        int len = snprintf(error_msg, sizeof(error_msg), "Error:
Failed to open file '%s'\n", filename);
        write(STDERR_FILENO, error_msg, len);

        shared_data->terminate = true;
        sem_post(sem_client);

        cleanup_resources(NULL, NULL, NULL, shared_data, shm_fd,
sem_client, sem_server);
        exit(EXIT_FAILURE);
    }

    char buffer[BUFFER_SIZE];
    ssize_t bytes_read;
    int processed_any = 0;
```

```
    while ((bytes_read = read(file_fd, buffer, sizeof(buffer)))
> 0) {

        size_t to_copy = bytes_read < BUFFER_SIZE ? bytes_read :
BUFFER_SIZE - 1;

        memcpy(shared_data->data, buffer, to_copy);

        shared_data->data_size = to_copy;

        shared_data->client_ready = true;


        if (sem_post(sem_client) == -1) {

            perror("sem_post failed");

            break;

        }


        if (sem_wait(sem_server) == -1) {

            if (errno == EINTR) continue;

            perror("sem_wait failed");

            break;

        }


        if (shared_data->data_size > 0) {

            write(STDOUT_FILENO, shared_data->data, shared_data-
>data_size);

            processed_any = 1;

        }


        shared_data->client_ready = false;

    }


    if (!processed_any && bytes_read == 0) {

        const char msg[] = "Нет составных чисел либо файл
содержит только завершающие числа.\n";

        write(STDOUT_FILENO, msg, sizeof(msg) - 1);

    }
```

```
    shared_data->terminate = true;

    if (sem_post(sem_client) == -1) {

        perror("sem_post failed");

    }


    close(file_fd);


    cleanup_resources(NULL, NULL, NULL, shared_data, shm_fd,
sem_client, sem_server);


    return EXIT_SUCCESS;

}
```

Протокол работы программы

```
goida@zov:~/OS/lab3$ ./client main test.txt
4
26
42
```

Файл 1:

```
1   4
2   26
3   42
4   -6
5   4
6
```

Вывод

В процессе выполнения лабораторной работы я составил программу, подобную программе из лабораторной №1, но при этом осуществляющую работу с shared memory и семафорами. Я приобрел базовые навыки использования shared memory и memory mapping. Одной из основных сложностей была в правильной регулировке выполнения процессов с помощью семафоров.