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: Резинкин Д.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 20.09.25

Москва, 2025

Постановка задачи

Вариант 1.

Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс считает их сумму и выводит её в файл. Числа имеют тип int. Количество чисел может быть произвольным.

В данной лабораторной работе программа из первой лабораторной работы была переработана с использованием:

- Разделяемой памяти (shared memory) для межпроцессного взаимодействия
- Отображения памяти (memory mapping)
- Семафоров для синхронизации процессов

Общий метод и алгоритм решения

1. Родительский процесс создаёт объекты:
   - Именованную разделяемую память для обмена данными
   - Два именованных семафора для синхронизации:
     - `sem_parent`: сигнализирует, что данные готовы для чтения ребёнком
     - `sem_child`: сигнализирует, что ответ готов для чтения родителем
2. Имена объектов генерируются уникально на основе PID родительского процесса
3. Родительский процесс создаёт дочерний процесс через fork() + execve()
4. Дочерний процесс подключается к уже созданным объектам разделяемой памяти и семафорам
5. Рабочий цикл:
   - Родитель читает строку от пользователя, копирует в разделяемую память, сигнализирует семафором
   - Дочерний процесс ждёт сигнал, обрабатывает строку, вычисляет сумму, записывает результат в файл и разделяемую память, сигнализирует обратно
   - Родитель получает сигнал, выводит результат пользователю
6. Завершение происходит при вводе пустой строки

Использованные системные вызовы:

- pid_t fork(void); - создание дочернего процесса
- int shm_open(const char *name, int oflag, mode_t mode); - создание/открытие разделяемой памяти
- int ftruncate(int fd, off_t length); - установка размера разделяемой памяти
- void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset); - отображение памяти
- int munmap(void *addr, size_t length); - удаление отображения памяти
- sem_t *sem_open(const char *name, int oflag, ...); - создание/открытие семафора
- int sem_wait(sem_t *sem); - ожидание семафора
- int sem_post(sem_t *sem); - увеличение семафора
- int sem_close(sem_t *sem); - закрытие семафора
- int sem_unlink(const char *name); - удаление семафора
- int shm_unlink(const char *name); - удаление разделяемой памяти
- ssize_t read(int fd, void *buf, size_t count); - чтение данных
- ssize_t write(int fd, const void *buf, size_t count); - запись данных
- int open(const char *pathname, int flags, mode_t mode); - открытие/создание файла
- pid_t waitpid(pid_t pid, int *status, int options); - ожидание завершения дочернего процесса

Код программы

parent_shm.c

```
#define _GNU_SOURCE
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <semaphore.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>

#define BUF_IN_SIZE  2048
#define BUF_OUT_SIZE 2048

// ======= утилиты I/O (как в ЛР1) =======
static void die(const char* msg) {
    ssize_t _ __attribute__((unused)) = write(2, msg, strlen(msg));
    ssize_t __ __attribute__((unused)) = write(2, "\n", 1);
    _exit(1);
}

static ssize_t write_all(int fd, const void* buf, size_t n) {
    const char* p = (const char*)buf;
    size_t left = n;
    while (left) {
        ssize_t w = write(fd, p, left);
        if (w < 0) {
            if (errno == EINTR) continue;
            return -1;
        }
        left -= (size_t)w; p += w;
    }
    return (ssize_t)n;
}

// читает одну строку (заканчивается '\n' или EOF). Возвращает длину, 0 при EOF.
static ssize_t read_line(int fd, char* buf, size_t cap) {
    size_t i = 0;
    while (i + 1 < cap) {
        char c;
        ssize_t r = read(fd, &c, 1);
        if (r == 0) break;          // EOF
        if (r < 0) {
            if (errno == EINTR) continue;
            return -1;
        }
        buf[i++] = c;
        if (c == '\n') break;
    }
```

```
    buf[i] = '\0';
    return (ssize_t)i;
}

static void chomp(char* s) {
    size_t n = strlen(s);
    if (n && s[n-1] == '\n') s[n-1] = '\0';
}

// ======= структура обмена через shared memory =======
struct shm_data {
    // Родитель пишет в in_buf, ребенок отвечает в out_buf.
    // Семафоры обеспечивают порядок, поэтому отдельные флаги не нужны.
    char in_buf[BUF_IN_SIZE];
    char out_buf[BUF_OUT_SIZE];
};

// ======= main =======
int main(int argc, char** argv, char** envp) {
    (void)argc; (void)argv;

    // 1) спросить имя выходного файла (как в ЛР1)
    const char* prompt1 = "Введите имя файла: ";
    write_all(1, prompt1, strlen(prompt1));
    char fileName[512];
    ssize_t fnlen = read_line(0, fileName, sizeof(fileName));
    if (fnlen < 0) die("не удалось выполнить чтение (fileName)");
    if (fnlen == 0) die("имя файла не указано");
    chomp(fileName);

    // 2) придумать уникальные имена объектов (обязательно по условию)
    pid_t pid_self = getpid();
    char shm_name[128], sem_p_name[128], sem_c_name[128];
    // POSIX требует, чтобы имя shm/sem начиналось с '/'
    // Пример: /shm_sum_12345, /sem_p_12345, /sem_c_12345
    {
        // простое формирование строк без snprintf из stdio.h
        char pidbuf[32]; int k = 0;
        // int -> строка
        {
            long long v = pid_self; char tmp[32]; int n = 0;
            if (v == 0) tmp[n++] = '0';
            while (v > 0) { tmp[n++] = (char)('0' + (v % 10)); v /= 10; }
            // reverse
            if (n == 0) { pidbuf[k++] = '0'; }
            else { for (int i = n - 1; i >= 0; --i) pidbuf[k++] = tmp[i]; }
            pidbuf[k] = '\0';
        }
        const char* a = "/shm_sum_";   size_t na = strlen(a);
        const char* b = "/sem_p_";     size_t nb = strlen(b);
        const char* c = "/sem_c_";     size_t nc = strlen(c);

        memcpy(shm_name, a, na); memcpy(shm_name + na, pidbuf, k); shm_name[na + k] = '\0';
```

```
    memcpy(sem_p_name, b, nb); memcpy(sem_p_name + nb, pidbuf, k); sem_p_name[nb + k] = '\0';
    memcpy(sem_c_name, c, nc); memcpy(sem_c_name + nc, pidbuf, k); sem_c_name[nc + k] = '\0';
}

// 3) создать shared memory
int shm_fd = shm_open(shm_name, O_CREAT | O_RDWR, 0666);
if (shm_fd < 0) die("shm_open failed");
if (ftruncate(shm_fd, sizeof(struct shm_data)) < 0) die("ftruncate failed");

void* map = mmap(NULL, sizeof(struct shm_data), PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
if (map == MAP_FAILED) die("mmap failed");
struct shm_data* shm = (struct shm_data*)map;

// 4) создать два семафора:
//    - sem_parent: родитель -> ребенок (данные готовы к чтению)
//    - sem_child : ребенок -> родитель (ответ готов)
sem_t* sem_parent = sem_open(sem_p_name, O_CREAT, 0666, 0);
if (sem_parent == SEM_FAILED) die("sem_open(sem_parent) failed");
sem_t* sem_child  = sem_open(sem_c_name, O_CREAT, 0666, 0);
if (sem_child == SEM_FAILED) die("sem_open(sem_child) failed");

// 5) запустить child через execve и передать имена объектов + имя файла
pid_t pid = fork();
if (pid < 0) die("fork failed");
if (pid == 0) {
    // argv: child_shm <fileName> <shm_name> <sem_p_name> <sem_c_name>
    char* args[6];
    args[0] = (char*)"child_shm";
    args[1] = fileName;
    args[2] = shm_name;
    args[3] = sem_p_name;
    args[4] = sem_c_name;
    args[5] = NULL;
    execve("./child_shm", args, envp);
    die("execve ./child_shm failed");
}

// 6) основной цикл: читать у пользователя -> в shared memory -> сигнал ребенку -> ждать ответ -> печатать
const char* prompt2 =
    "Введите строку, например: \"12 -3 7\" и нажмите Ентер.\n"
    "Пустая строка для завершения.\n";
write_all(1, prompt2, strlen(prompt2));

for (;;) {
    write_all(1, "> ", 2);
    ssize_t n = read_line(0, shm->in_buf, sizeof(shm->in_buf));
    if (n < 0) die("read(user line) failed");

    // EOF: посылаем пустую строку как сигнал завершения
    if (n == 0 || shm->in_buf[0] == '\n' || shm->in_buf[0] == '\0') {
        shm->in_buf[0] = '\0';
        sem_post(sem_parent); // дать ребёнку понять, что пора завершаться
        break;
```

```
        }

        // отдать строку ребенку
        if (sem_post(sem_parent) < 0) die("sem_post(parent) failed");

        // ждать ответа
        if (sem_wait(sem_child) < 0) die("sem_wait(child) failed");

        // вывести ответ
        write_all(1, shm->out_buf, strlen(shm->out_buf));
    }

    // дочитать финальные сообщения, если ребенок что-то допишет (здесь не требуется)

    int status = 0;
    waitpid(pid, &status, 0);


    // 7) уборка
    munmap(shm, sizeof(struct shm_data));
    close(shm_fd);
    sem_close(sem_parent);
    sem_close(sem_child);
    sem_unlink(sem_p_name);
    sem_unlink(sem_c_name);
    shm_unlink(shm_name);


    return WIFEXITED(status) ? WEXITSTATUS(status) : 1;
}
```

child_shm.c

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

#define BUF_IN_SIZE  2048
#define BUF_OUT_SIZE 2048

// ===== утилиты I/O (как в ЛР1) =====
static void die(const char* msg) {
    ssize_t _ __attribute__((unused)) = write(2, msg, strlen(msg));
    ssize_t __ __attribute__((unused)) = write(2, "\n", 1);
    _exit(1);
}

static ssize_t write_all(int fd, const void* buf, size_t n) {
```

```
    const char* p = (const char*)buf;
    size_t left = n;
    while (left) {
        ssize_t w = write(fd, p, left);
        if (w < 0) { if (errno == EINTR) continue; return -1; }
        left -= (size_t)w; p += w;
    }
    return (ssize_t)n;
}

// ---- парсер целых и int -> строка (как в твоей child.c) ----
static int parse_ll(const char* s, size_t* i, long long* out) {
    while (s[*i] == ' ' || s[*i] == '\t' || s[*i] == '\r') (*i)++;

    int sign = 1;
    if (s[*i] == '+' || s[*i] == '-') {
        if (s[*i] == '-') sign = -1;
        (*i)++;
    }

    if (s[*i] < '0' || s[*i] > '9') return 0;

    long long val = 0;
    int found_digit = 0;
    while (s[*i] >= '0' && s[*i] <= '9') {
        int d = s[*i] - '0';
        val = val * 10 + d;
        (*i)++;
        found_digit = 1;
    }

    // запрет дробных: если . или , — считаем ошибкой, как в ЛР1
    if (s[*i] == '.' || s[*i] == ',') {
        *out = 0;
        while (s[*i] && s[*i] != ' ' && s[*i] != '\n') (*i)++;
        errno = EINVAL;
        return -1;
    }

    if (!found_digit) return 0;
    *out = val * sign;
    return 1;
}

static int ll_to_buf(long long v, char* buf) {
    char tmp[32];
    int neg = v < 0;
    unsigned long long x = neg ? (unsigned long long)(-(v+1)) + 1ULL : (unsigned long long)v;
    int n = 0;
    do {
        tmp[n++] = (char)('0' + (x % 10ULL));
        x /= 10ULL;
    } while (x);
```

```
    int k = 0;
    if (neg) buf[k++] = '-';
    for (int i = n - 1; i >= 0; --i) buf[k++] = tmp[i];
    return k;
}

// ===== структура shared memory =====
struct shm_data {
    char in_buf[BUF_IN_SIZE];
    char out_buf[BUF_OUT_SIZE];
};

int main(int argc, char** argv) {
    if (argc < 5) die("usage: child_shm <fileName> <shm_name> <sem_p_name> <sem_c_name>");

    const char* fileName   = argv[1];
    const char* shm_name   = argv[2];
    const char* sem_p_name = argv[3];
    const char* sem_c_name = argv[4];

    // открыть файл на дозапись (как в ЛР1)
    int fd = open(fileName, O_WRONLY | O_CREAT | O_APPEND, 0644);
    if (fd < 0) die("child: open(file) failed");

    // подключиться к shared memory и семафорам
    int shm_fd = shm_open(shm_name, O_RDWR, 0666);
    if (shm_fd < 0) die("child: shm_open failed");

    void* map = mmap(NULL, sizeof(struct shm_data), PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (map == MAP_FAILED) die("child: mmap failed");
    struct shm_data* shm = (struct shm_data*)map;

    sem_t* sem_parent = sem_open(sem_p_name, 0);
    if (sem_parent == SEM_FAILED) die("child: sem_open(parent) failed");
    sem_t* sem_child  = sem_open(sem_c_name, 0);
    if (sem_child == SEM_FAILED) die("child: sem_open(child) failed");

    // рабочий цикл: ждать строку -> посчитать сумму -> отдать ответ
    for (;;) {
        if (sem_wait(sem_parent) < 0) die("child: sem_wait(parent) failed");

        // сигнал на завершение: пустая строка
        if (shm->in_buf[0] == '\0' || shm->in_buf[0] == '\n') {
            break;
        }

        // разбор строки и сумма
        size_t i = 0;
        int found_any = 0;
        long long sum = 0;
        long long val;

        while (1) {
```

```
            int st = parse_ll(shm->in_buf, &i, &val);
            if (st == 1) {
                found_any = 1;
                sum += val;
            } else if (st == -1) {
                const char* msg = "ERR: invalid number format\n";
                // вернуть ответ в shared memory
                size_t L = strlen(msg);
                memcpy(shm->out_buf, msg, L+1);
                // продублировать в файл (как в ЛР1)
                write_all(fd, msg, L);
                // сообщить родителю
                sem_post(sem_child);
                found_any = 0;
                goto next_iter;
            } else {
                break; // конец токенов
            }
        }

        if (!found_any) {
            const char* msg = "Бро, ошибка, тут числа нет либо что-то чужеродное\n";
            size_t L = strlen(msg);
            memcpy(shm->out_buf, msg, L+1);
            write_all(fd, msg, L);
            sem_post(sem_child);
            goto next_iter;
        }

        // подготовить "sum=<value>\n"
        {
            char out[128];
            int k = 0;
            memcpy(out + k, "sum=", 4); k += 4;
            k += ll_to_buf(sum, out + k);
            out[k++] = '\n';

            // в shared memory
            if ((size_t)k >= BUF_OUT_SIZE) k = (int)BUF_OUT_SIZE - 1;
            memcpy(shm->out_buf, out, (size_t)k);
            shm->out_buf[k] = '\0';

            // в файл
            write_all(fd, out, (size_t)k);
        }

        // отдать сигнал родителю: ответ готов
        sem_post(sem_child);

    next_iter:
        ;
    }
```

```
    // финал
    munmap(shm, sizeof(struct shm_data));
    close(shm_fd);
    sem_close(sem_parent);
    sem_close(sem_child);
    close(fd);
    return 0;
}
```

Протокол работы программы

```
diwan@DESKTOP-FVGD4PE:/mnt/e/Учеба/2 курс/oc/lab3$ make

gcc -Wall -Wextra -O2 parent_shm.c -o parent_shm -pthread

gcc -Wall -Wextra -O2 child_shm.c -o child_shm -pthread

diwan@DESKTOP-FVGD4PE:/mnt/e/Учеба/2 курс/oc/lab3$ ./parent_shm

Введите имя файла: output.txt

Введите строку, например: "12 -3 7" и нажмите Ентер.

Пустая строка для завершения.

> 12 32 -12 -30 2

sum=4

> 12.2 2

ERR: invalid number format

> 1 2 3 4 5

sum=15
```

Вывод

В ходе выполнения лабораторной работы были изучены механизмы межпроцессного взаимодействия через разделяемую память и семафоры в операционной системе Linux. Получены практические навыки работы с системными вызовами shm_open, mmap, sem_open, sem_wait, sem_post и другими.

Основные отличия от первой лабораторной работы:

- Вместо каналов (pipe) используется разделяемая память
- Вместо блокирующего чтения/записи используется синхронизация через семафоры
- Объекты имеют уникальные имена на основе PID процесса
- Исключено использование стандартных библиотек ввода-вывода

Проблемы, с которыми пришлось столкнуться:

- Организация корректной синхронизации процессов через семафоры

- Генерация уникальных имён для объектов IPC
- Корректное освобождение ресурсов (память, семафоры) при завершении

В результате была создана корректно работающая программа, демонстрирующая эффективное использование механизмов разделяемой памяти и синхронизации для межпроцессного взаимодействия.