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-214БВ-24

Студент: Шитов Н.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 27.11.25

Москва, 2025

Постановка задачи

Вариант 4.

нужно взять свою первую лабу и переделать её с использованием shared memory и memory mapping. Варианты остаются те же, что и у первой лабораторной. Так как блокирующего чтения из каналов у вас больше не будет, то для синхронизации чтения и записи из shared memory будем использовать семафор. Для тех, кто будет писать под Windows, прилагаю список системных функций, которые вам понадобятся: CreateFileMapping, MapViewOfFile, UnmapViewOfFile, OpenSemaphore, WaitForSingleObject, ReleaseSemaphore. Аналогично, для Linux: shm_open, shm_unlink, ftruncate, mmap, munmap, sem_open, sem_wait, sem_post, sem_unlink, sem_close Отдельно отмечу, что shared memory и memory mapping это не одно и то же. В этой лабораторной работе вам нужно создать именованый shared memory объект, он будет находиться именно в оперативной памяти, а не на вашем диске в виде файлика. Поскольку вам нужно будет создавать именованые shared memory и семафоры, то их название должно отличаться для экземпляров пар программ server и client. Не делайте название именованных shared memory и семафоров константным.

Общий метод и алгоритм решения

Использованные системные вызовы:

- **shm_open()** — создаёт/открывает именованный объект разделяемой памяти.
- **ftruncate()** — устанавливает размер разделяемой памяти.
- **mmap()** — отображает разделяемую память в адресное пространство процесса.
- **munmap()** — отменяет отображение.
- **shm_unlink()** — удаляет объект разделяемой памяти из системы.
- **sem_open()** — создаёт/открывает именованный семафор для синхронизации.
- **sem_wait() / sem_post()** — захват и освобождение семафора.
- **sem_close() / sem_unlink()** — закрытие и удаление семафора.
- **fork()** — создаёт дочерний процесс.
- **open() / write() / close()** — работа с файлом вывода.
- **kill()** — при делении на ноль принудительное завершение родительского процесса из дочернего.

  Родительский процесс создаёт дочерний и разделяемую память, в которой размещает структуру с буфером и флагами. Пользователь вводит строку с числами родитель кладёт её в общую память и сигнализирует дочернему через семафор. Дочерний процесс, дождавшись сигнала, считывает строку, парсит float-числа, последовательно делит первое на остальные, записывает результат в файл, и, при делении на 0 завершает оба процесса. Всё без каналов, только shared memory и синхронизация.

Код программы

main.c

```c
#include <stdint.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <semaphore.h>
#include "shared.h"
#include <string.h>

static char SERVER_PROGRAM_NAME[] = "lab3";

int main(int argc, char **argv) {
    if (argc == 1) {
        char msg[1024];
        uint32_t len = snprintf(msg, sizeof(msg) - 1, "usage: %s filename\n", argv[0]);
        write(STDERR_FILENO, msg, len);
        exit(EXIT_SUCCESS);
    }

    char progpath[1024];
    {
        ssize_t len = readlink("/proc/self/exe", progpath, sizeof(progpath) - 1);
        if (len == -1) {
            const char msg[] = "error: failed to read full program path\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }
        while (progpath[len] != '/')
            --len;
        progpath[len] = '\0';
    }

    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0600);
    if (shm_fd == -1) {
        const char msg[] = "error: failed to create shared memory object\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    if (ftruncate(shm_fd, sizeof(SharedData)) == -1) {
        const char msg[] = "error: failed to set shared memory size\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        close(shm_fd);
        exit(EXIT_FAILURE);
    }
```

```
    SharedData *shared_data = mmap(NULL, sizeof(SharedData), PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);
    if (shared_data == MAP_FAILED) {
        const char msg[] = "error: failed to map shared memory\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        close(shm_fd);
        exit(EXIT_FAILURE);
    }

    sem_t *sem_data_ready = sem_open(SEM_DATA_READY_NAME, O_CREAT | O_EXCL, 0600, 0);
    if (sem_data_ready == SEM_FAILED) {
        const char msg[] = "error: failed to create semaphore data_ready\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        munmap(shared_data, sizeof(SharedData));
        close(shm_fd);
        exit(EXIT_FAILURE);
    }

    sem_t *sem_data_free = sem_open(SEM_DATA_FREE_NAME, O_CREAT | O_EXCL, 0600, 1);
    if (sem_data_free == SEM_FAILED) {
        const char msg[] = "error: failed to create semaphore data_free\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        sem_close(sem_data_ready);
        sem_unlink(SEM_DATA_READY_NAME);
        munmap(shared_data, sizeof(SharedData));
        close(shm_fd);
        exit(EXIT_FAILURE);
    }

    pid_t child = fork();

    switch (child) {
        case -1: {
            const char msg[] = "error: failed to spawn new process\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            sem_close(sem_data_ready);
            sem_close(sem_data_free);
            sem_unlink(SEM_DATA_READY_NAME);
            sem_unlink(SEM_DATA_FREE_NAME);
            munmap(shared_data, sizeof(SharedData));
            close(shm_fd);
            shm_unlink(SHM_NAME);
            exit(EXIT_FAILURE);
        } break;

        case 0: {

            close(shm_fd);

            char path[1024];
            snprintf(path, sizeof(path) - 1, "%s/%s", progpath, SERVER_PROGRAM_NAME);

            char *const args[] = {SERVER_PROGRAM_NAME, argv[1], NULL};
```

```
            int status = execv(path, args);

            if (status == -1) {
                const char msg[] = "error: failed to exec into new executable image\n";
                write(STDERR_FILENO, msg, sizeof(msg));
                exit(EXIT_FAILURE);
            }
        } break;

        default: {
            {
                pid_t pid = getpid();
                char msg[64];
                const int32_t length = snprintf(msg, sizeof(msg),
                    "%d: I'm a parent, my child has PID %d\n", pid, child);
                write(STDOUT_FILENO, msg, length);
            }

            char buf[4096];
            ssize_t bytes;

            while (true) {
                bytes = read(STDIN_FILENO, buf, sizeof(buf) - 1);
                if (bytes < 0) {
                    const char msg[] = "error: failed to read from stdin\n";
                    write(STDERR_FILENO, msg, sizeof(msg));
                    break;
                } else if (buf[0] == '\n') {
                    break;
                }

                if (sem_wait(sem_data_free) == -1) {
                    const char msg[] = "error: failed to wait on semaphore data_free\n";
                    write(STDERR_FILENO, msg, sizeof(msg));
                    break;
                }

                if (bytes >= sizeof(shared_data->data)) {
                    bytes = sizeof(shared_data->data) - 1;
                }
                memcpy(shared_data->data, buf, bytes);
                shared_data->data[bytes] = '\0';
                shared_data->ready = 1;

                if (sem_post(sem_data_ready) == -1) {
                    const char msg[] = "error: failed to post semaphore data_ready\n";
                    write(STDERR_FILENO, msg, sizeof(msg));
                    break;
                }
            }

            wait(NULL);

            sem_close(sem_data_ready);
```

```
            sem_close(sem_data_free);
            sem_unlink(SEM_DATA_READY_NAME);
            sem_unlink(SEM_DATA_FREE_NAME);
            munmap(shared_data, sizeof(SharedData));
            close(shm_fd);
            shm_unlink(SHM_NAME);
        } break;
    }

    return 0;
}
```

server.c

```
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/mman.h>
#include <semaphore.h>
#include "shared.h"
#include <string.h>

float bober(char buf[]) {
    float numbers[100];
    int count = 0;
    int i = 0;

    while (buf[i] != '\0' && buf[i] != '\n') {
        while (buf[i] == ' ') i++;

        if (buf[i] == '\0' || buf[i] == '\n') break;

        float num = 0;
        int is_negative = 0;
        int has_decimal = 0;
        float decimal_place = 0.1f;

        if (buf[i] == '-') {
            is_negative = 1;
            i++;
        }

        while ((buf[i] >= '0' && buf[i] <= '9') || buf[i] == '.') {
            if (buf[i] == '.') {
                has_decimal = 1;
            } else {
                if (!has_decimal) {
                    num = num * 10 + (buf[i] - '0');
                } else {
                    num += (buf[i] - '0') * decimal_place;
```

```
                    decimal_place *= 0.1f;
                }
            }
            i++;
        }

        if (is_negative) {
            num = -num;
        }

        numbers[count++] = num;
        while (buf[i] == ' ') i++;
    }

    if (count == 0) {
        return 0;
    }

    if (count == 1) {
        return numbers[0];
    }

    float result = numbers[0];
    for (int j = 1; j < count; j++) {
        if (numbers[j] == 0) {
            const char msg[] = "error: division by zero\n";
            write(STDOUT_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }
        result /= numbers[j];
    }

    return result;
}

int main(int argc, char **argv) {
    if (argc != 2) {
        const char msg[] = "error: filename argument required\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0);
    if (shm_fd == -1) {
        const char msg[] = "error: failed to open shared memory object\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    SharedData *shared_data = mmap(NULL, sizeof(SharedData), PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);
    if (shared_data == MAP_FAILED) {
        const char msg[] = "error: failed to map shared memory\n";
        write(STDERR_FILENO, msg, sizeof(msg));
```

```
        close(shm_fd);
        exit(EXIT_FAILURE);
    }

    sem_t *sem_data_ready = sem_open(SEM_DATA_READY_NAME, 0);
    if (sem_data_ready == SEM_FAILED) {
        const char msg[] = "error: failed to open semaphore data_ready\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        munmap(shared_data, sizeof(SharedData));
        close(shm_fd);
        exit(EXIT_FAILURE);
    }

    sem_t *sem_data_free = sem_open(SEM_DATA_FREE_NAME, 0);
    if (sem_data_free == SEM_FAILED) {
        const char msg[] = "error: failed to open semaphore data_free\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        sem_close(sem_data_ready);
        munmap(shared_data, sizeof(SharedData));
        close(shm_fd);
        exit(EXIT_FAILURE);
    }

    int file = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0600);
    if (file == -1) {
        const char msg[] = "error: failed to open requested file\n";
        write(STDOUT_FILENO, msg, sizeof(msg));
        sem_close(sem_data_ready);
        sem_close(sem_data_free);
        munmap(shared_data, sizeof(SharedData));
        close(shm_fd);
        exit(EXIT_FAILURE);
    }

    while (true) {
        if (sem_wait(sem_data_ready) == -1) {
            const char msg[] = "error: failed to wait on semaphore data_ready\n";
            write(STDOUT_FILENO, msg, sizeof(msg));
            break;
        }

        if (shared_data->ready) {
            char local_buf[4096];
            strncpy(local_buf, shared_data->data, sizeof(local_buf) - 1);
            local_buf[sizeof(local_buf) - 1] = '\0';

            shared_data->ready = 0;

            if (sem_post(sem_data_free) == -1) {
                const char msg[] = "error: failed to post semaphore data_free\n";
                write(STDOUT_FILENO, msg, sizeof(msg));
                break;
            }
```

```
            float result = bober(local_buf);
            char result_str[128];
            int bytes = snprintf(result_str, sizeof(result_str), "%.4f\n", result);

            if (write(file, result_str, bytes) != bytes) {
                const char msg[] = "error: failed to write to file\n";
                write(STDOUT_FILENO, msg, sizeof(msg));
                break;
            }

            const char success_msg[] = "success\n";
            write(STDOUT_FILENO, success_msg, sizeof(success_msg));
        }
    }

    sem_close(sem_data_ready);
    sem_close(sem_data_free);
    munmap(shared_data, sizeof(SharedData));
    close(shm_fd);
    close(file);

    return 0;
}
```

shared.h

```
#ifndef SHARED_H
#define SHARED_H

#include <sys/mman.h>
#include <fcntl.h>
#include <semaphore.h>
#include <stdio.h>

#define SHM_NAME "/lab3_shm"
#define SEM_DATA_READY_NAME "/lab3_sem_data_ready"
#define SEM_DATA_FREE_NAME "/lab3_sem_data_free"

typedef struct {
    char data[4096];
    int ready;
} SharedData;

#endif
```

Протокол работы программы

Тестирование:

```
┌──(flow㉿kali)-[~/Documents/vsc/oc/lab3]
└─$ ./main output.txt
70481: I'm a parent, my child has PID 70482
1000 2 2 2
success
12 2.5 1.1
success
1 1 1 1 1 1
success
0 123 12 1
success
5 1 2 3 4 5
success
1.1 2
success
11 0
error: division by zero
^C

┌──(flow㉿kali)-[~/Documents/vsc/oc/lab3]
└─$ cat output.txt
125.0000
4.3636
1.0000
0.0000
0.0417
0.5500
```

Strace:

execve("./main", ["./main", "output.txt"], 0x7ffc79a23498 /* 55 vars */) = 0

brk(NULL) = 0x55877eb98000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7feec3bea000

access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

fstat(3, {st_mode=S_IFREG|0644, st_size=122447, ...}) = 0

mmap(NULL, 122447, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7feec3bcc000

close(3) = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0p\236\2\0\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 840, 64) = 840

fstat(3, {st_mode=S_IFREG|0755, st_size=2003408, ...}) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 840, 64) = 840

mmap(NULL, 2055800, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7feec39d6000

mmap(0x7feec39fe000, 1462272, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7feec39fe000

mmap(0x7feec3b63000, 352256, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x18d000) = 0x7feec3b63000

mmap(0x7feec3bb9000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e2000) = 0x7feec3bb9000

mmap(0x7feec3bbf000, 52856, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7feec3bbf000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7feec39d3000

arch_prctl(ARCH_SET_FS, 0x7feec39d3740) = 0

set_tid_address(0x7feec39d3a10) = 22670

set_robust_list(0x7feec39d3a20, 24)   = 0

rseq(0x7feec39d3680, 0x20, 0, 0x53053053) = 0

mprotect(0x7feec3bb9000, 16384, PROT_READ) = 0

mprotect(0x5587718f5000, 4096, PROT_READ) = 0

mprotect(0x7feec3c28000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7feec3bcc000, 122447)   = 0

readlink("/proc/self/exe", "/home/flow/Documents/vsc/oc/lab3"..., 1023) = 37

openat(AT_FDCWD, "/dev/shm/lab3_shm", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0600) = 3

ftruncate(3, 4100)   = 0

mmap(NULL, 4100, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7feec3be8000

rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_DROPPABLE|MAP_ANONYMOUS, -1, 0) = 0x7feec3be7000

mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7feec3be6000

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

getrandom("\x6e\x69\x61\x44\x8e\x8c\x68\x86\xf8\x75\x52\xde\x61\x14\xec\xc5\x47\xd2\x1e\x3a\x69\xaf\xf5\x56\xf2\xe1\xdd\x0a\x80\x5e\x06\x68", 32, 0) = 32

newfstatat(AT_FDCWD, "/dev/shm/sem.H5VfAJ", 0x7ffcebdf09a0, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)

openat(AT_FDCWD, "/dev/shm/sem.H5VfAJ", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0600) = 4

write(4, "\0\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32

mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7feec3be5000

link("/dev/shm/sem.H5VfAJ", "/dev/shm/sem.lab3_sem_data_ready") = 0

fstat(4, {st_mode=S_IFREG|0600, st_size=32, ...}) = 0

getrandom("\x6b\xab\x65\x55\x34\x6f\x5f\x3f", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x55877eb98000

brk(0x55877ebb9000) = 0x55877ebb9000

unlink("/dev/shm/sem.H5VfAJ") = 0

close(4) = 0

newfstatat(AT_FDCWD, "/dev/shm/sem.S104P3", 0x7ffcebdf09a0, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)

openat(AT_FDCWD, "/dev/shm/sem.S104P3", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0600) = 4

write(4, "\1\0\0\0\0\0\0\200\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0", 32) = 32

mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7feec3be4000

link("/dev/shm/sem.S104P3", "/dev/shm/sem.lab3_sem_data_free") = 0

fstat(4, {st_mode=S_IFREG|0600, st_size=32, ...}) = 0

unlink("/dev/shm/sem.S104P3") = 0

close(4) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7feec39d3a10) = 22671

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

getpid() = 22670

write(1, "22670: I'm a parent, my child ha"..., 44) = 44

read(0, "100 2 2\n", 4095) = 8

futex(0x7feec3be5000, FUTEX_WAKE, 1) = 1

read(0, "5 2.5\n", 4095) = 6

futex(0x7feec3be5000, FUTEX_WAKE, 1) = 1

read(0, "1 1 1 1\n", 4095) = 8

futex(0x7feec3be5000, FUTEX_WAKE, 1) = 1

```
read(0, "11 3 4 5 6\n", 4095)          = 11

futex(0x7feec3be5000, FUTEX_WAKE, 1)    = 1

read(0, "0 1\n", 4095)                 = 4

futex(0x7feec3be5000, FUTEX_WAKE, 1)    = 1

read(0, "1 0\n", 4095)                 = 4

futex(0x7feec3be5000, FUTEX_WAKE, 1)    = 1

read(0, 0x7ffcebdf0d50, 4095)          = ? ERESTARTSYS (To be restarted if SA_RESTART is set)

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=22671, si_uid=1000, si_status=1,
si_utime=0, si_stime=0} ---

read(0, 0x7ffcebdf0d50, 4095)          = ? ERESTARTSYS (To be restarted if SA_RESTART is set)

--- SIGINT {si_signo=SIGINT, si_code=SI_KERNEL} ---

  +++ killed by SIGINT +++
```

Вывод

В лабораторной работе №3 реализован обмен данными между процессами через POSIX shared memory с синхронизацией на именованных семафорах, что позволило заменить каналы из ЛР №1 на более эффективный механизм IPC. Программа корректно обрабатывает последовательное деление чисел, записывает результат в файл и завершает оба процесса при попытке деления на ноль.