

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-23

Студент: Богданов М.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 15.11.25

Москва, 2025

Постановка задачи

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.

Общий метод и алгоритм решения

Использованные системные вызовы:

- 1) shm_open - создаёт/открывает объект разделяемой памяти
- 2) shm_unlink - удаляет именованный объект разделяемой памяти
- 3) ftruncate - устанавливает размер файла/разделяемой памяти
- 4) mmap - отображает файл/объект в память процесса
- 5) munmap - удаляет отображение памяти
- 6) sem_open - создаёт/открывает именованный семафор
- 7) sem_wait - уменьшает значение семафора (ожидание)
- 8) sem_post - увеличивает значение семафора (сигнал)
- 9) sem_unlink - удаляет именованный семафор
- 10) sem_close - закрывает открытый семафор

Основная идея была в том что теперь процессы вместо того чтобы обмениваться данными при помощи буфера(в данном случае пайпа) теперь просто обращаются к одной и той же области памяти. Это значительно ускоряет работу кода при большом объеме данных, потому что исчезает необходимость в копировании в пайп.

Код программы

lab3_client.c

```
#include <fcntl.h>
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
#include <semaphore.h>
#include <sys/mman.h>
#include <errno.h>

#define SHM_SIZE 4096

static char SERVER_PROGRAM_NAME[] = "lab3_server";
const char SHM_NAME[] = "lab_shm";
const char SEM_NAME[] = "lab_sem";

int main(int argc, char **argv) {
    shm_unlink(SHM_NAME);
    sem_unlink(SEM_NAME);
    int shm = shm_open(SHM_NAME, O_RDWR | O_CREAT | O_EXCL, 0600);
    if (shm == -1) {
```

```

const char msg[] = "error: failed to create SHM\n";
write(STDERR_FILENO, msg, sizeof(msg) - 1);
exit(EXIT_FAILURE);
}

if (ftruncate(shm, SHM_SIZE) == -1) {
    const char msg[] = "error: failed to resize SHM\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    shm_unlink(SHM_NAME);
    close(shm);
    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) - 1);
    shm_unlink(SHM_NAME);
    close(shm);
    exit(EXIT_FAILURE);
}

uint32_t *length = (uint32_t *)shm_buf;
*length = 0;

sem_t *sem = sem_open(SEM_NAME, O_CREAT | O_EXCL, 0600, 1);
if (sem == SEM_FAILED) {
    const char msg[] = "error: failed to create semaphore\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    munmap(shm_buf, SHM_SIZE);
    shm_unlink(SHM_NAME);
    close(shm);
    exit(EXIT_FAILURE);
}

char filename[256];
ssize_t bytes_read = read(STDIN_FILENO, filename, sizeof(filename) - 1);
if (bytes_read <= 0) {
    const char msg[] = "error: failed to read filename\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    sem_unlink(SEM_NAME);
    sem_close(sem);
    munmap(shm_buf, SHM_SIZE);
    shm_unlink(SHM_NAME);
    close(shm);
    exit(EXIT_FAILURE);
}

filename[bytes_read] = '\0';
filename[strcspn(filename, "\n")] = '\0';

char prospath[1024];
ssize_t len = readlink("/proc/self/exe", prospath, sizeof(prospath) - 1);
if (len == -1) {
    const char msg[] = "error: failed to read full program path\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);
    sem_unlink(SEM_NAME);
    sem_close(sem);
    munmap(shm_buf, SHM_SIZE);
    shm_unlink(SHM_NAME);
    close(shm);
    exit(EXIT_FAILURE);
}

while (len > 0 && prospath[len] != '/')
    --len;

if (len == 0) {
    const char msg[] = "error: invalid program path\n";
    write(STDERR_FILENO, msg, sizeof(msg) - 1);

```

```

sem_unlink(SEM_NAME);
sem_close(sem);
munmap(shm_buf, SHM_SIZE);
shm_unlink(SHM_NAME);
close(shm);
exit(EXIT_FAILURE);
}
progp[1024] = '\0';
const pid_t child = fork();
switch (child) {
    case -1: {
        const char msg[] = "error: failed to spawn new process\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        sem_unlink(SEM_NAME);
        sem_close(sem);
        munmap(shm_buf, SHM_SIZE);
        shm_unlink(SHM_NAME);
        close(shm);
        exit(EXIT_FAILURE);
    } break;

    case 0: {
        char path[1024];
        strcpy(path, progp);
        strcat(path, "/");
        strcat(path, SERVER_PROGRAM_NAME);

        char *const args[] = {SERVER_PROGRAM_NAME, filename, NULL};
        int32_t 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) - 1);
            exit(EXIT_FAILURE);
        }
    } break;

    default: {
        char buf[SHM_SIZE - sizeof(uint32_t) - 1];
        ssize_t bytes;
        while ((bytes = read(STDIN_FILENO, buf, sizeof(buf))) > 0) {
            if (buf[0] == '\n') {
                break;
            }
            sem_wait(sem);
            uint32_t *length = (uint32_t *)shm_buf;
            char *text = shm_buf + sizeof(uint32_t);

            *length = bytes;
            memcpy(text, buf, bytes);
            text[bytes] = '\0';
            sem_post(sem);
        }

        if (bytes < 0) {
            const char msg[] = "error: failed to read from stdin\n";
            write(STDERR_FILENO, msg, sizeof(msg) - 1);
        }
        sem_wait(sem);
        uint32_t *length_final = (uint32_t *)shm_buf;
        *length_final = UINT32_MAX;
        sem_post(sem);
    }
}

```

```

    sem_unlink(SEM_NAME);
    sem_close(sem);
    munmap(shm_buf, SHM_SIZE);
    shm_unlink(SHM_NAME);
    close(shm);
} break;
}

return 0;
}

```

lab3_server.c

```

#include <fcntl.h>
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <semaphore.h>
#include <sys/mman.h>
#include <errno.h>
#include <ctype.h>

#define SHM_SIZE 4096
#define MIN_LEN 256

const char SHM_NAME[] = "lab_shm";
const char SEM_NAME[] = "lab_sem";
float buf_to_numbers(char * buf) {
    float n = 10, num2 = 0, result;
    int num_path, i = 0, flag = 0, min_flag = 0, first_flag = 0;
    while (true){
        if (buf[i] != '1' && buf[i] != '2' && buf[i] != '3' && buf[i] != '4' && buf[i] != '5' \
            && buf[i] != '6' && buf[i] != '7' && buf[i] != '8' && buf[i] != '9' && buf[i] != '0' && buf[i] != '.' && buf[i] != ' ' \
            && buf[i] != '-' && buf[i] != '\n'){
            return -1;
        }
        if (buf[i] == ' '){
            n = 10;
            flag = 0;
            if (min_flag == 1){
                num2 *= -1;
            }
            min_flag = 0;
            if (first_flag == 0){
                result = num2;
                first_flag = 1;
            }
            else if (num2 != 0 && first_flag == 1){
                result /= num2;
            }
            else if (num2 == 0 && first_flag == 1){
                return -1;
            }
            num2 = 0;

```

```

}
else if (buf[i] == '\n' || buf[i] == '\0'){
    n = 10;
    flag = 0;
    if (min_flag == 1){
        num2 *= -1;
    }
    min_flag = 0;
    if (first_flag == 0){
        result = num2;
        first_flag = 1;
    }
    else if (num2 != 0 && first_flag == 1){
        result /= num2;
    }
    else if (num2 == 0 && first_flag == 1){
        return -1;
    }
    num2 = 0;
    break;
}
else if (buf[i] == '.'){
    flag = 1;
    n = 0.1;
}
else if (buf[i] == '-'){
    min_flag = 1;
}
else if (flag == 0) {
    num2 = (buf[i] - '0') + num2 * n;
}
else if (flag == 1) {
    num2 += (buf[i] - '0') * n;
    n /= 10;
}
++i;
}
return result;
}

```

```

int main(int argc, char **argv) {
    if (argc < 2) {
        const char msg[] = "error: no filename provided\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }
    int shm = shm_open(SHM_NAME, O_RDWR, 0);
    if (shm == -1) {
        const char msg[] = "error: failed to open SHM\n";
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        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) - 1);
        close(shm);
        exit(EXIT_FAILURE);
    }
    sem_t *sem = sem_open(SEM_NAME, 0);
    if (sem == SEM_FAILED) {

```

```

const char msg[] = "error: failed to open semaphore\n";
write(STDERR_FILENO, msg, sizeof(msg) - 1);
munmap(shm_buf, SHM_SIZE);
close(shm);
exit(EXIT_FAILURE);
}
int32_t 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(STDERR_FILENO, msg, sizeof(msg) - 1);
    sem_close(sem);
    munmap(shm_buf, SHM_SIZE);
    close(shm);
    exit(EXIT_FAILURE);
}
bool running = true;
float res = 0;
char output_buf[MIN_LEN];
while (running) {
    sem_wait(sem);
    uint32_t *length = (uint32_t *)shm_buf;
    char *text = shm_buf + sizeof(uint32_t);
    if (*length == UINT32_MAX) {
        sem_post(sem);
        running = false;
    } else if (*length > 0) {
        res = buf_to_numbers(text);
        if (res != -1) {
            int len = snprintf(output_buf, MIN_LEN, "%f\n", res);
            write(file, output_buf, len);
        } else {
            const char error_msg[] = "error: invalid input or division by zero\n";
            write(file, error_msg, sizeof(error_msg) - 1);
        }

        *length = 0;
        sem_post(sem);
    } else {
        sem_post(sem);
    }
}
sem_close(sem);
munmap(shm_buf, SHM_SIZE);
close(shm);
close(file);

return 0;
}

```

Протокол работы программы

Тестирование:

#1

bogdanoff@arch ~/s/O/l/src (main)> strace -o test1.log ./lab3_client

test.txt

12 12 12 12

1 1 1 1 1

121 1212 1212 1212

13313 13 1 1 1 3131

1333 333

1311 1

1311 0

0 1

Файл:

0.006944

error: invalid input or division by zero

0.000000

0.327077

4.003003

1311.000000

error: invalid input or division by zero

0.000000

#2

bogdanoff@arch ~/s/O/l/src (main)> strace -o test2.log ./lab3_client

test.txt

0 1 1

1 1 1212

1222 2222

1 1 1 1 1 1 1

Файл:

0.000000

0.000825

0.549955

1.000000

#3

bogdanoff@arch ~/s/O/l/src (main)> strace -o test3.log ./lab3_client

test.txt

1 0

0 1

0 0 0 0

Файл:

error: invalid input or division by zero

0.000000

error: invalid input or division by zero

Stace log будет находится в отдельных файлах в папке trace т.к. тут он будет занимать очень много места

Вывод

Использование shared memory значительно ускоряет работу кода, потому что позволяет процессам напрямую общаться между собой. Но при этом нужно работать с ней очень аккуратно потому что неправильная работа с ней может полностью нарушить работу программы (например, гонка данных).