

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-209БВ-24

Студент: Осипов М.Н.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 30.11.25

Москва, 2025

Постановка задачи

Вариант 9.

В файле записаны команды вида: «число число число». Дочерний процесс производит деление первого числа команда, на последующие числа в команде, а результат выводит в стандартный поток вывода. Если происходит деление на 0, то тогда дочерний и родительский процесс завершают свою работу. Проверка деления на 0 должна осуществляться на стороне дочернего процесса. Числа имеют тип float. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

- `shm_open()` - создает/открывает разделяемую область памяти
 - `mmap()` - отображает shared memory в адресное пространство процесса
 - `munmap()` - закрывает отображение памяти
 - `shm_unlink()` - удаляет shared memory объект
 - `fork()` - создает копию процесса (дочерний процесс)
 - `execl()` - заменяет образ процесса на новый исполняемый файл
 - `wait()` - ожидает завершения дочернего процесса
 - `usleep()` - приостанавливает выполнение на микросекунды
 - `fopen()` - открывает файл `commands.txt`
 - `fgets()` - читает команды построчно из файла
 - `fclose()` - закрывает файл
 - `strtok()` - парсит строку на числа по разделителям
 - `atof()` - преобразует строки в числа типа float
1. Родительский процесс создает shared memory через `shm_open()` и читает команды из файла `commands.txt`
 2. Запускается дочерний процесс через `fork()` + `execl()`, который открывает ту же shared memory через `shm_open()`
 3. Обмен данными происходит через общую память (структура `shared_data_t` с полями `stop_flag`, `line`, `result`, `result_ready`)
 4. Дочерний процесс выполняет вычисления (деление первого числа на все последующие) и выводит результаты в формате "Calculation: X / Y / Z = Result"

5. При делении на 0 дочерний процесс устанавливает stop_flag = 1 и оба процесса корректно завершаются

Код программы

parent.c

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/mman.h>

#include <sys/wait.h>

#include <fcntl.h>

#include <errno.h>


#define SHM_NAME "/lab3_shm"

#define MAX_LINE_LENGTH 1024


typedef struct {

    int stop_flag;

    char line[MAX_LINE_LENGTH];

    float result;

    int result_ready;

} shared_data_t;


void handle_error(const char *msg) {

    perror(msg);

    exit(EXIT_FAILURE);

}


int main(int argc, char *argv[]) {

    if (argc != 2) {
```

```
    fprintf(stderr, "Usage: %s <filename>\n", argv[0]);

    exit(EXIT_FAILURE);

}

FILE *file = fopen(argv[1], "r");

if (file == NULL) {

    handle_error("Error opening file");

}


// Создаем shared memory

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

if (shm_fd == -1) {

    handle_error("Error creating shared memory");

}


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

    handle_error("Error setting shared memory size");

}


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) {

    handle_error("Error mapping shared memory");

}


// Инициализируем shared data

shared_data->stop_flag = 0;

shared_data->line[0] = '\0';

shared_data->result = 0.0f;
```

```
shared_data->result_ready = 0;

// Создаем дочерний процесс
pid_t pid = fork();

if (pid == -1) {
    handle_error("Error creating process");
}

if (pid == 0) {
    // Дочерний процесс
    execl("./a.out", "a.out", NULL);
    handle_error("Error starting child process");
} else {
    // Родительский процесс
    char line[MAX_LINE_LENGTH];

    while (fgets(line, sizeof(line), file) != NULL) {
        if (shared_data->stop_flag) {
            break;
        }

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

        if (strlen(line) == 0) {
            continue;
        }

        strcpy(shared_data->line, line);
    }
}
```

```

while (!shared_data->result_ready && !shared_data->stop_flag) {

    usleep(10000);

}

if (shared_data->stop_flag) {

    break;

}

shared_data->result_ready = 0;

}

fclose(file);

if (!shared_data->stop_flag) {

    shared_data->stop_flag = 1;

    wait(NULL);

}

munmap(shared_data, sizeof(shared_data_t));

close(shm_fd);

shm_unlink(SHM_NAME);

}

return 0;

}

```

child.c

```

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

```

```
#include <sys/mman.h>

#include <fcntl.h>

#include <errno.h>


#define SHM_NAME "/lab3_shm"

#define MAX_LINE_LENGTH 1024

#define MAX_NUMBERS 100


typedef struct {
    int stop_flag;

    char line[MAX_LINE_LENGTH];

    float result;

    int result_ready;
} shared_data_t;


int parse_numbers(const char *line, float *numbers, int max_count) {
    char *token;

    char line_copy[MAX_LINE_LENGTH];

    int count = 0;

    strcpy(line_copy, line);

    token = strtok(line_copy, " \\t\\n");

    while (token != NULL && count < max_count) {
        numbers[count++] = atof(token);

        token = strtok(NULL, " \\t\\n");
    }

    return count;
}
```

```
int main() {

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

    if (shm_fd == -1) {

        perror("shm_open failed");

        return 1;

    }

    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");

        close(shm_fd);

        return 1;

    }

    float numbers[MAX_NUMBERS];

    while (1) {

        if (shared_data->stop_flag) {

            break;

        }

        if (strlen(shared_data->line) > 0) {

            int num_count = parse_numbers(shared_data->line, numbers, MAX_NUMBERS);

            if (num_count >= 2) {

                float result = numbers[0];

                int error = 0;
```



```
printf("Calculation: %.2f", numbers[0]);

for (int i = 1; i < num_count; i++) {

    printf(" / %.2f", numbers[i]);

    if (numbers[i] == 0.0f) {

        printf(" -> ERROR: division by zero!\n");

        shared_data->stop_flag = 1;

        error = 1;

        break;

    }

    result /= numbers[i];

}

if (!error) {

    printf(" = %.2f\n", result);

    shared_data->result = result;

    shared_data->result_ready = 1;

}

}

shared_data->line[0] = '\0';

}

usleep(10000);

}

munmap(shared_data, sizeof(shared_data_t));

close(shm_fd);

return 0;

}
```

Протокол работы программы

./parent commands.txt

Calculation: $10.00 / 2.00 / 2.50 = 2.00$

Calculation: $0.00 / 3.00 / 2.00 = 0.00$

Calculation: $15.00 / 3.00 / 5.00 = 1.00$

Calculation: $20.00 / 0.00 \rightarrow$ ERROR: division by zero!

strace ./parent commands.txt

execve("./parent", ["/parent", "commands.txt"], 0x7ffc38fda038 /* 27 vars */) = 0

brk(NULL) = 0x635e575d6000

arch_prctl(0x3001 /* ARCH_??? */, 0x7fff97bd2820) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7a7ea3a85000

access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

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

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=61360, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 61360, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7a7ea3a76000

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\3\0>\0\1\0\0\0P\237\2\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"..., 784, 64) = 784

pread64(3, "\4\0\0\0\0\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0"..., 48, 848) = 48

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\00{\f\225\|= \201\327\312\301P\32\$\230\266\235"..., 68, 896) = 68

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 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"..., 784, 64) = 784

mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7a7ea3800000

mprotect(0x7a7ea3828000, 2023424, PROT_NONE) = 0

mmap(0x7a7ea3828000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7a7ea3828000

mmap(0x7a7ea39bd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7a7ea39bd000

mmap(0x7a7ea3a16000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7a7ea3a16000

mmap(0x7a7ea3a1c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7a7ea3a1c000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7a7ea3a73000

arch_prctl(ARCH_SET_FS, 0x7a7ea3a73740) = 0

set_tid_address(0x7a7ea3a73a10) = 985

set_robust_list(0x7a7ea3a73a20, 24) = 0

rseq(0x7a7ea3a740e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7a7ea3a16000, 16384, PROT_READ) = 0

mprotect(0x635e419ad000, 4096, PROT_READ) = 0

mprotect(0x7a7ea3abf000, 8192, PROT_READ) = 0

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

munmap(0x7a7ea3a76000, 61360) = 0

getrandom("\xa3\xfa\x5e\x44\xff\x61\xc1\xe", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x635e575d6000

brk(0x635e575f7000) = 0x635e575f7000

openat(AT_FDCWD, "commands.txt", O_RDONLY) = 3

openat(AT_FDCWD, "/dev/shm/lab3_shm", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 4

ftruncate(4, 1036) = 0

mmap(NULL, 1036, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7a7ea3abe000

clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7a7ea3a73a10) = 986

newfstatat(3, "", {st_mode=S_IFREG|0777, st_size=60, ...}, AT_EMPTY_PATH) = 0

read(3, "10 2 2.5\r\n0 3 2\r\n15 3 5\r\n20 \321\217\320\261"..., 4096) = 60

clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000}, Calculation: 10.00 / 2.00 / 2.50 = 2.00

NULL) = 0

clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000}, Calculation: 0.00 / 3.00 / 2.00 = 0.00

NULL) = 0

clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000}, Calculation: 15.00 / 3.00 / 5.00 = 1.00

NULL) = 0

clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000}, Calculation: 20.00 / 0.00 ->
ERROR: division by zero!

NULL) = 0

close(3) = 0

munmap(0x7a7ea3abe000, 1036) = 0

close(4) = 0

unlink("/dev/shm/lab3_shm") = 0

exit_group(0) = ?

+++ exited with 0 +++

Вывод

Я составил и отладил программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними. В результате работы программа (основной процесс) создаёт один дочерний процесс. Родительский и дочерний процессы успешно обмениваются данными через разделяемую память. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files). Использованы ключевые системные вызовы: shm_open(), mmap(), munmap() - для работы с shared memory; fork(), execl(), wait() - для управления процессами; fopen(), fgets() - для чтения команд из файла. Программа корректно обрабатывает различные сценарии, включая аварийное завершение при делении на ноль.