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: Александров М.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 10.12.25

Москва, 2025

Постановка задачи

Вариант 2.

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса пишет имя файла, которое будет передано при создании дочернего процесса. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс передает команды пользователя через pipe1, который связан с стандартным входным потоком дочернего процесса. Дочерний процесс при необходимости передает данные в родительский процесс через pipe2. Результаты своей работы дочерний процесс пишет в созданный им файл. Допускается просто открыть файл и писать туда, не перенаправляя стандартный поток вывода.

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

Общий метод и алгоритм решения

Использованные системные вызовы:

- *pid_t fork(void)*; – создает дочерний процесс.
- *int shm_open(const char *name, int oflag, mode_t mode)*; – создает или открывает объект разделяемой памяти.
- *int ftruncate(int fd, off_t length)*; – устанавливает размер объекта разделяемой памяти;
- *int execv(const char *path, char *const argv[])*; – заменяет текущий образ процесса на новый исполняемый файл.
- *void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);* – отображает разделяемую память в адресное пространство процесса.
- *sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value);* – создает или открывает именованный семафор.
- *int sem_wait(sem_t *sem);* – уменьшает значение семафора (операция P).
- *int sem_post(sem_t *sem);* – увеличивает значение семафора (операция V).
- *int sem_close(sem_t *sem);* – закрывает семафор.
- *int sem_unlink(const char *name);* – удаляет именованный семафор из системы.
- int munmap(void *addr, size_t length); – удаляет отображение разделяемой памяти.
- *int shm_unlink(const char *name);* – удаляет объект разделяемой памяти из системы.
- *pid_t waitpid(pid_t pid, int *status, int options);* – ожидает изменения состояния указанного дочернего процесса.

В рамках лабораторной работы была реализована программа, состоящая из двух исполняемых файлов: родительского (*parent*) и дочернего (*child*). Родительский процесс создаёт дочерний с помощью системного вызова *fork*(), после чего заменяет его образ на отдельную программу с помощью *execv*(). Для передачи данных между процессами используется именованная разделяемая память *(shm_open, mmap*) и именованный семафор (*sem_open, sem_wait, sem_post*) для синхронизации доступа к общей области памяти.

Родительский процесс запрашивает у пользователя последовательности чисел в формате: <число> <число> … <число> <*Enter*>. Каждая введённая строка передаётся в дочерний процесс через разделяемую память.

Дочерний процесс (*child.c*) получает имя выходного файла как аргумент командной строки, считывает строки из разделяемой памяти, парсит числа (типа *float*), вычисляет их сумму и

записывает результат в указанный файл. После обработки строки дочерний процесс сбрасывает флаг занятости в разделяемой памяти, сигнализируя родителю о готовности к приёму следующей порции данных.

Синхронизация обеспечивается семафором: только один процесс в каждый момент времени имеет доступ к данным в разделяемой памяти, что исключает гонки и повреждение данных.

Код программы

parent.c

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

#define SHM_SIZE 4096

const char SHM_NAME[] = "example_sh_memory";
const char SEM_NAME[] = "example_semaphore";

static char SERVER_PROGRAM_NAME[] = "child";



int main(int argc, char *argv[]) {
    // проверка аргументов
    if (argc != 2) {
        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);
```

```
}

// путь до директории
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';
   }

// shared memory

int shm = shm_open(SHM_NAME, O_RDWR, 0600);
if (shm == -1 && errno != ENOENT) {
   const char msg[] = "error: failed to open SHM\n";
   write(STDERR_FILENO, msg, sizeof(msg));
   _exit(EXIT_FAILURE);
}

shm = shm_open(SHM_NAME, O_RDWR | O_CREAT | O_TRUNC, 0600);
if (shm == -1) {
   const char msg[] = "error: failed to create SHM\n";
   write(STDERR_FILENO, msg, sizeof(msg));
   _exit(EXIT_FAILURE);
```

```
    }

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

    // semaphore

    sem_t *sem = sem_open(SEM_NAME, O_RDWR | O_CREAT | O_TRUNC, 0600, 1);
    if (sem == SEM_FAILED) {
        const char msg[] = "error: failed to create semaphore\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        _exit(EXIT_FAILURE);
    }

    // дочерний процесс
    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));
            _exit(EXIT_FAILURE);
```

```
    } break;

case 0: { // дочерний
    {
            char path[2048];
            snprintf(path, sizeof(path) - 1, "%s/%s", progpath, SERVER_PROGRAM_NAME);

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

            int32_t status = execv(path, args);

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

default: { // PARENT

    bool running = 1;
    while (running) {
        char buf[SHM_SIZE - sizeof(uint32_t)];
        ssize_t bytes = read(STDIN_FILENO, buf, sizeof(buf));

        if (bytes < 0) {
            const char msg[] = "error: failed to read from stdin\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            _exit(EXIT_FAILURE);
        }

        sem_wait(sem);
```

```
            uint32_t *length = (uint32_t *)shm_buf;
            char *text = shm_buf + sizeof(uint32_t);

            if (bytes > 0) {
                *length = (uint32_t)bytes;
                memcpy(text, buf, bytes);
            } else {
                *length = UINT32_MAX;
                running = false;
            }
            sem_post(sem);
        }
    } break;
    }

    waitpid(child, NULL, 0);

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

    exit(EXIT_SUCCESS);
}
```

child.c

```
#include <stdint.h>
#include <stdbool.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
#include <string.h>
```

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

#define SHM_SIZE 4096

const char SHM_NAME[] = "example_sh_memory";
const char SEM_NAME[] = "example_semaphore";

int main(int argc, char *argv[]) {
    if (argc != 2) {
        char msg[1024];
        uint32_t len = snprintf(msg, sizeof(msg) - 1, "error: need filename\n");
        write(STDERR_FILENO, msg, len);
        _exit(EXIT_FAILURE);
    }

        int shm = shm_open(SHM_NAME, O_RDWR, 0);
    if (shm == -1 && errno != ENOENT) {
        const char msg[] = "error: failed to open SHM\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        _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));
        _exit(EXIT_FAILURE);
```

```
}

sem_t *sem = sem_open(SEM_NAME, O_RDWR);
if (sem == SEM_FAILED) {
    const char msg[] = "error: failed to open semaphore\n";
    write(STDERR_FILENO, msg, sizeof(msg));
    _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 file";
    write(STDERR_FILENO, msg, sizeof(msg));
    _exit(EXIT_FAILURE);
}

bool running = true;
while (running) {
    sem_wait(sem);

    uint32_t *length = (uint32_t *)shm_buf;
    char *text = shm_buf + sizeof(uint32_t);

    if (*length == UINT32_MAX) {
        running = false;
    } else if (*length > 0 && *length < SHM_SIZE - sizeof(uint32_t)) {
        char local_buf[4096];
        memcpy(local_buf, text, *length);
        local_buf[*length] = '\0';
        *length = 0;

        sem_post(sem);
```

```
            float sum = 0.0f;
            char *token = strtok(local_buf, " \t");
            while (token) {
                char *endptr;
                float f = strtof(token, &endptr);
                if (endptr != token && (*endptr == '\0' || isspace(*endptr))) {
                    sum += f;
                }
                token = strtok(NULL, " \t");
            }

            char output[100];
            int len = snprintf(output, sizeof(output), "%.3f\n", sum);
            if (len > 0) {
                write(file, output, len);
            }

            continue;
        }

        sem_post(sem);
    }

    close(file);
    sem_close(sem);
    munmap(shm_buf, SHM_SIZE);
    exit(EXIT_SUCCESS);
}
```

Протокол работы программы

Тестирование:

```
maks-alex@DESKTOP-QFPFVP1:~/OS/lab3/src$ cc -o parent parent.c && cc -o child child.c
maks-alex@DESKTOP-QFPFVP1:~/OS/lab3/src$ ./parent test1.txt
1.5 2.5
-1.0 3.0 2.0
10.0
maks-alex@DESKTOP-QFPFVP1:~/OS/lab3/src$ cat test1.txt
4.000
4.000
10.000
maks-alex@DESKTOP-QFPFVP1:~/OS/lab3/src$ ./parent test2.txt
maks-alex@DESKTOP-QFPFVP1:~/OS/lab3/src$ cat test2.txt
maks-alex@DESKTOP-QFPFVP1:~/OS/lab3/src$ ./parent test3.txt
abc xyz
-1.543 1.543
maks-alex@DESKTOP-QFPFVP1:~/OS/lab3/src$ cat test3.txt
0.000
0.000
maks-alex@DESKTOP-QFPFVP1:~/OS/lab3/src$ ./parent test1.txt
124 4234 2325 654 32 1.2
1.1 2.1 3.1 4.7
maks-alex@DESKTOP-QFPFVP1:~/OS/lab3/src$ cat test1.txt
7370.200
11.000
```

Strace:

execve("./parent", ["./parent", "test1.txt"], 0x7ffffc2c8660 /* 29 vars */) = 0

brk(NULL) = 0x563f8a79f000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f50bfd2c000

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

fstat(3, {st_mode=S_IFREG|0644, st_size=20003, ...}) = 0

mmap(NULL, 20003, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f50bfd27000

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\0\220\243\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"..., 784, 64) = 784

fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 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"..., 784, 64) = 784

mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f50bfb15000

mmap(0x7f50bfb3d000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f50bfb3d000

mmap(0x7f50bfcc5000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7f50bfcc5000

mmap(0x7f50bfd14000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f50bfd14000

mmap(0x7f50bfd1a000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f50bfd1a000
close(3) = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f50bfb12000
arch_prctl(ARCH_SET_FS, 0x7f50bfb12740) = 0
set_tid_address(0x7f50bfb12a10) = 12242
set_robust_list(0x7f50bfb12a20, 24) = 0
rseq(0x7f50bfb13060, 0x20, 0, 0x53053053) = 0
mprotect(0x7f50bfd14000, 16384, PROT_READ) = 0
mprotect(0x563f586a5000, 4096, PROT_READ) = 0
mprotect(0x7f50bfd64000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f50bfd27000, 20003) = 0
readlink("/proc/self/exe", "/home/maks-alex/OS/lab3/src/pare"..., 1023) = 34
openat(AT_FDCWD, "/dev/shm/example_sh_memory",
O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/example_sh_memory",
O_RDWR|O_CREAT|O_TRUNC|O_NOFOLLOW|O_CLOEXEC, 0600) = 3
ftruncate(3, 4096) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f50bfd2b000
openat(AT_FDCWD, "/dev/shm/sem.example_semaphore",
O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file or directory)
getrandom("\xf2\xdc\x7f\xc7\xbb\x65\x6a\xb5", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.ChDWlK", 0x7fff90572f80, AT_SYMLINK_NOFOLLOW)
= -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.ChDWlK",
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) = 0x7f50bfd2a000
link("/dev/shm/sem.ChDWlK", "/dev/shm/sem.example_semaphore") = 0
fstat(4, {st_mode=S_IFREG|0600, st_size=32, ...}) = 0
getrandom("\x9b\x3e\x59\xf3\x91\x09\xb6\xff", 8, GRND_NONBLOCK) = 8
brk(NULL) = 0x563f8a79f000
brk(0x563f8a7c0000) = 0x563f8a7c0000
unlink("/dev/shm/sem.ChDWlK") = 0
close(4) = 0
clone(child_stack=NULL,

flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process 12243 attached

, child_tidptr=0x7f50bfb12a10) = 12243

[pid 12243] set_robust_list(0x7f50bfb12a20, 24 <unfinished ...>

[pid 12242] read(0, <unfinished ...>

[pid 12243] <... set_robust_list resumed>) = 0

[pid 12243] execve("/home/maks-alex/OS/lab3/src/child", ["child", "test1.txt"], 0x7fff90574910 /* 29 vars */) = 0

[pid 12243] brk(NULL)   = 0x56466ba1e000

[pid 12243] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f17944dc000

[pid 12243] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

[pid 12243] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

[pid 12243] fstat(3, {st_mode=S_IFREG|0644, st_size=20003, ...}) = 0

[pid 12243] mmap(NULL, 20003, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f17944d7000

[pid 12243] close(3)   = 0

[pid 12243] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

[pid 12243] read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\0\0\0\0"..., 832) = 832

[pid 12243] 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"..., 784, 64) = 784

[pid 12243] fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0

[pid 12243] 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"..., 784, 64) = 784

[pid 12243] mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f17942c5000

[pid 12243] mmap(0x7f17942ed000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f17942ed000

[pid 12243] mmap(0x7f1794475000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7f1794475000

[pid 12243] mmap(0x7f17944c4000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f17944c4000

[pid 12243] mmap(0x7f17944ca000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f17944ca000

[pid 12243] close(3)   = 0

[pid 12243] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f17942c2000

[pid 12243] arch_prctl(ARCH_SET_FS, 0x7f17942c2740) = 0

[pid 12243] set_tid_address(0x7f17942c2a10) = 12243

[pid 12243] set_robust_list(0x7f17942c2a20, 24) = 0

[pid 12243] rseq(0x7f17942c3060, 0x20, 0, 0x53053053) = 0

[pid 12243] mprotect(0x7f17944c4000, 16384, PROT_READ) = 0

[pid 12243] mprotect(0x564666807000, 4096, PROT_READ) = 0

[pid 12243] mprotect(0x7f1794514000, 8192, PROT_READ) = 0

[pid 12243] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

[pid 12243] munmap(0x7f17944d7000, 20003) = 0

[pid 12243] openat(AT_FDCWD, "/dev/shm/example_sh_memory", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 3

[pid 12243] mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f17944db000

[pid 12243] openat(AT_FDCWD, "/dev/shm/sem.example_semaphore", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 4

[pid 12243] fstat(4, {st_mode=S_IFREG|0600, st_size=32, ...}) = 0

[pid 12243] getrandom("\x8d\x8a\xd3\xae\x62\xc5\x95\x6e", 8, GRND_NONBLOCK) = 8

[pid 12243] brk(NULL) = 0x56466ba1e000

[pid 12243] brk(0x56466ba3f000) = 0x56466ba3f000

[pid 12243] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f17944da000

[pid 12243] close(4) = 0

[pid 12243] openat(AT_FDCWD, "test1.txt", O_WRONLY|O_CREAT|O_TRUNC, 0600) = 4

1.5 2.5 3.5

[pid 12242] <... read resumed>"1.5 2.5 3.5\n", 4092) = 12

[pid 12242] read(0, <unfinished ...>

[pid 12243] futex(0x7f17944da000, FUTEX_WAKE, 1) = 0

[pid 12243] write(4, "7.500\n", 6) = 6

10.0 35.5

[pid 12242] <... read resumed>"10.0 35.5\n", 4092) = 10

[pid 12242] futex(0x7f50bfd2a000, FUTEX_WAKE, 1 <unfinished ...>

[pid 12243] futex(0x7f17944da000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 12242] <... futex resumed>) = 0

[pid 12243] <... futex resumed>) = -1 EAGAIN (Resource temporarily unavailable)

[pid 12242] read(0, <unfinished ...>

[pid 12243] write(4, "45.500\n", 7) = 7

[pid 12242] <... read resumed>"", 4092) = 0

[pid 12242] futex(0x7f50bfd2a000, FUTEX_WAKE, 1 <unfinished ...>

[pid 12243] close(4 <unfinished ...>

[pid 12242] <... futex resumed>) = 0

[pid 12242] wait4(12243, <unfinished ...>

[pid 12243] <... close resumed>) = 0

[pid 12243] munmap(0x7f17944da000, 32) = 0

[pid 12243] munmap(0x7f17944db000, 4096) = 0

[pid 12243] exit_group(0) = ?

[pid 12243] +++ exited with 0 +++

<... wait4 resumed>NULL, 0, NULL) = 12243

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=12243, si_uid=1000, si_status=0, si_utime=7854 /* 78.54 s */, si_stime=2 /* 0.02 s */} ---

munmap(0x7f50bfd2a000, 32) = 0

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

munmap(0x7f50bfd2b000, 4096) = 0

close(3) = 0

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

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В ходе выполнения лабораторной работы были успешно изучены и применены основные системные вызовы для работы с процессами и межпроцессным взаимодействием в ОС Linux — именованная разделяемая память и именованные семафоры. Была реализована программа, в которой родительский процесс передаёт дочернему последовательности чисел через разделяемую память, а дочерний вычисляет их сумму и записывает результат в файл. Основными сложностями стали корректная синхронизация доступа к общей памяти, безопасное управление ресурсами.