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-216БВ-24

Студент: Гуськов А.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 05.11.25

Москва, 2025

Постановка задачи

Вариант 8.

Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe. Родительский процесс читает из pipe и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами.

В файле записаны команды вида:«число число число<endline>». Дочерний процесс производит деление первого числа команды, 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 int. 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.
- *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.
- *int execl(const char *path, const char *arg, ...)*; – заменяет образ текущей программы на указанную, принимая аргументы в качестве списка.
- *pid_t waitpid(pid_t pid, int *status, int options)*; – ожидает изменения состояния указанного дочернего процесса.

В рамках лабораторной работы создавалась программа, которая использует механизмы разделяемой памяти и семафоров для межпроцессного взаимодействия. Родительский процесс *parent.c* создает дочерний процесс (с помощью системного вызова *fork()*), который заменяет свой образ на программу *child.c* (с помощью системного вызова *execl()*).

Для организации взаимодействия между процессами используется разделяемая память (*shared memory*) и семафоры (*semaphores*). Родительский процесс создает область разделяемой памяти и семафор для синхронизации, запрашивает у пользователя имя файла с данными для обработки. Дочерний процесс читает указанный файл, построчно обрабатывает данные (выполняет деление

первого числа на последующие числа в строке) и записывает результаты в разделяемую память. Родительский процесс читает результаты из разделяемой памяти и выводит их в стандартный вывод.

Код программы

parent.c

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

#define SHM_SIZE 4096

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

int main(int argc, char** argv)
{
  int shared_mem = shm_open(SHM_NAME, O_RDWR, 0666);
  if (shared_mem == -1 && errno != ENOENT)
  {
     const char message[] = "Error: Unable to open shared_mem\n";
     write(STDERR_FILENO, message, sizeof(message));
     exit(EXIT_FAILURE);
  }

  shared_mem = shm_open(SHM_NAME, O_RDWR | O_CREAT | O_TRUNC, 0666);
  if (shared_mem == -1)
  {
```

```
        const char message[] = "Error: Unable to create shared_mem\n";
        write(STDERR_FILENO, message, sizeof(message));
        exit(EXIT_FAILURE);
    }

    if (ftruncate(shared_mem, SHM_SIZE) != 0)
    {
        const char message[] = "Error: Unable to resize shared_mem\n";
        write(STDERR_FILENO, message, sizeof(message));
        exit(EXIT_FAILURE);
    }

    char* const shared_mem_buffer = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shared_mem, 0);
    if (shared_mem_buffer == MAP_FAILED)
    {
        const char message[] = "Error: Unable to map shared_mem\n";
        write(STDERR_FILENO, message, sizeof(message));
        exit(EXIT_FAILURE);
    }



    sem_t* semaphore = sem_open(SEM_NAME, O_RDWR | O_CREAT, 0666, 1);
    if (semaphore == SEM_FAILED)
    {
        const char message[] = "Error: Unable to open semaphore\n";
        write(STDERR_FILENO, message, sizeof(message));
        exit(EXIT_FAILURE);
    }

    char file_path[128];
    {
        const char message[] = "Input filename: ";
```

```
    write(STDOUT_FILENO, message, sizeof(message));

    int result = read(STDIN_FILENO, file_path, sizeof(file_path) - 1);
    if (result <= 0)
    {
        const char error_message[] = "Error: Unable to read from standard input\n";
        write(STDERR_FILENO, error_message, sizeof(error_message));
        exit(EXIT_FAILURE);
    }

    file_path[result - 1] = 0;
}



pid_t child = fork();

if (child == 0)
{
    execl("./child", "client", file_path, NULL);

    const char message[] = "Error: Unable to execv\n";
    write(STDERR_FILENO, message, sizeof(message));
    exit(EXIT_FAILURE);
}
else if (child == -1)
{
    const char message[] = "Error: Unable to fork";
    write(STDERR_FILENO, message, sizeof(message));
    exit(EXIT_FAILURE);
}
```

```
int running = 1;
while(running)
{
    sem_wait(semaphore);

    int* length = (int*)shared_mem_buffer;
    char* data = shared_mem_buffer + sizeof(int);

    if (*length == INT_MAX)
    {
        running = 0;
    }
    else if (*length > 0)
    {
        write(STDOUT_FILENO, data, *length);
        *length = 0;
    }

    sem_post(semaphore);
}


waitpid(child, NULL, 0);

sem_unlink(SEM_NAME);
sem_close(semaphore);

munmap((void*)shared_mem_buffer, SHM_SIZE);

shm_unlink(SHM_NAME);
close(shared_mem);
```

```
  return 0;
}
```

child.c

```
#include <ctype.h>
#include <fcntl.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <unistd.h>
#include <semaphore.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)
  {
    const char message[] = "Error: No filename provided\n";
    write(STDERR_FILENO, message, sizeof(message));
    _exit(EXIT_FAILURE);
```

```
    }

    int shared_mem = shm_open(SHM_NAME, O_RDWR, 0600);
    if (shared_mem == -1)
    {
        const char message[] = "Error: Unable to open shared_memory\n";
        write(STDERR_FILENO, message, sizeof(message));
        _exit(EXIT_FAILURE);
    }

    char* shared_mem_buffer = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shared_mem, 0);
    if (shared_mem_buffer == MAP_FAILED)
    {
        const char message[] = "Error: Unable to map shared_memory\n";
        write(STDERR_FILENO, message, sizeof(message));
        _exit(EXIT_FAILURE);
    }

    sem_t* semaphore = sem_open(SEM_NAME, O_RDWR);
    if (semaphore == SEM_FAILED)
    {
        const char message[] = "Error: Unable to open semaphore\n";
        write(STDERR_FILENO, message, sizeof(message));
        _exit(EXIT_FAILURE);
    }

    int file = open(argv[1], O_RDONLY);
    if (file == -1)
    {
```

```
    const char message[] = "Error: Unable to open file";
    write(STDERR_FILENO, message, sizeof(message));
    _exit(EXIT_FAILURE);
}

int bytes;
const int BUFFER_SIZE = 4096 - sizeof(int);
char buffer[BUFFER_SIZE];

int running = 1;
while ((bytes = read(file, buffer, BUFFER_SIZE - 1)) > 0)
{
    buffer[bytes] = 0;
    int numbers[128];
    int counter = 0;
    char* ptr = buffer;

    while(*ptr != 0)
    {
        if (*ptr == '\n')
        {
            ptr++;

            if (counter >= 2)
            {
                int result = numbers[0];

                int valid = 1;
                for (int i = 1; i < counter; ++i)
                {
```

```
            if (numbers[i] == 0)
            {
                const char err_msg[] = "error: Division by zero\n";
                write(STDERR_FILENO, err_msg, sizeof(err_msg));
                valid = 0;
                break;
            }

            result /= numbers[i];
        }

        if (valid)
        {
            char output[50];
            int size = snprintf(output, sizeof(output), "%d\n", result);

            usleep(10000);

            sem_wait(semaphore);

            int* length = (int*)shared_mem_buffer;
            char* text = shared_mem_buffer + sizeof(int);
            *length = size;
            memcpy(text, output, size);

            sem_post(semaphore);
        }
    }
    else if (counter > 0)
    {
```

```
            const char message[] = "error: Not enough arguments passed\n";
            write(STDERR_FILENO, message, sizeof(message));
        }

        counter = 0;
    }

    if (isdigit(*ptr))
    {
        if (counter < 128)
        {
            if (sscanf(ptr, "%d", &numbers[counter]) > 0)
            {
                counter++;
            }
        }

        while (isdigit((unsigned char)*ptr))
        {
            ptr++;
        }
    }
    else
    {
        ptr++;
    }

  }

}
```

```
    sem_wait(semaphore);

    int* length = (int*)shared_mem_buffer;
    *length = INT_MAX;

    sem_post(semaphore);

    close(file);
    sem_close(semaphore);
    munmap(shared_mem_buffer, SHM_SIZE);
    close(shared_mem);

    return 0;
}
```

Протокол работы программы

Тестирование:

Данные входного файла:

12 3 4

20 5 2

100 10 2 5

8 2

15 3 5

```
❯ ./parent
Input filename: ./test_normal.txt
1
2
1
4
1
```

Данные входного файла:

10 2 5

15 0 3

20 4 5

```
❯ ./parent
Input filename: ./test_division_by_zero.txt
1
error: Division by zero
1
```

Данные входного файла:

-12 3 -4

20 -5 2

-100 10 -2

```
❯ ./parent
Input filename: ./test_negative.txt
1
2
5
```

Strace:

❯ strace -f ./parent

execve("./parent", ["./parent"], 0x7ffc713a97a8 /* 61 vars */) = 0

brk(NULL) = 0x55e18da10000

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=162095, ...}) = 0

mmap(NULL, 162095, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f82ab6e1000

close(3) = 0

openat(AT_FDCWD, "/usr/lib/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\0000x\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"..., 896, 64) = 896

fstat(3, {st_mode=S_IFREG|0755, st_size=2149728, ...}) = 0

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f82ab6df000

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"..., 896, 64) = 896

mmap(NULL, 2174000, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f82ab400000

mmap(0x7f82ab424000, 1515520, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x24000) = 0x7f82ab424000

mmap(0x7f82ab596000, 454656, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x196000) = 0x7f82ab596000

mmap(0x7f82ab605000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x204000) = 0x7f82ab605000

mmap(0x7f82ab60b000, 31792, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f82ab60b000

close(3) = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f82ab6dc000

arch_prctl(ARCH_SET_FS, 0x7f82ab6dc740) = 0

set_tid_address(0x7f82ab6dca10) = 6299

set_robust_list(0x7f82ab6dca20, 24) = 0

rseq(0x7f82ab6dc680, 0x20, 0, 0x53053053) = 0

mprotect(0x7f82ab605000, 16384, PROT_READ) = 0

mprotect(0x55e160d25000, 4096, PROT_READ) = 0

mprotect(0x7f82ab748000, 8192, PROT_READ) = 0

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

getrandom("\xc3\xc8\xce\x1f\xf7\xab\xea\x02", 8, GRND_NONBLOCK) = 8

munmap(0x7f82ab6e1000, 162095) = 0

**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, 0666) = 3**

ftruncate(3, 4096) = 0

mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f82ab708000

**openat(AT_FDCWD, "/dev/shm/sem.example_semaphore",
O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file or directory)**

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

mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_DROPPABLE|MAP_ANONYMOUS, -1, 0)
= 0x7f82ab707000

mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f82ab706000

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

getrandom("\x51\xa5\x76\xce\x88\xc6\xc5\x4e\x2d\x9c\xed\xcf\xb2\xbd\xcd\x6a\xcb\x6f\xfe\xa9\xcb\x
99\x6a\x42\x80\xda\xdf\x14\x97\xf6\x73\xef", 32, 0) = 32

newfstatat(AT_FDCWD, "/dev/shm/sem.CbF2am", 0x7ffdf167ff00, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (No such file or directory)

**openat(AT_FDCWD, "/dev/shm/sem.CbF2am",
O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 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) = 0x7f82ab705000

link("/dev/shm/sem.CbF2am", "/dev/shm/sem.example_semaphore") = 0

fstat(4, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0

brk(NULL) = 0x55e18da10000

brk(0x55e18da31000) = 0x55e18da31000

unlink("/dev/shm/sem.CbF2am") = 0

close(4) = 0

write(1, "Input filename: \0", 17Input filename:) = 17

read(0./test_normal.txt

, "./test_normal.txt\n", 127) = 18

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

clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f82ab6dca10) = 6302

rt_sigprocmask(SIG_SETMASK, []strace: Process 6302 attached

, NULL, 8) = 0

[pid 6302] set_robust_list(0x7f82ab6dca20, 24) = 0

[pid 6302] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

[pid 6302] execve("./child", ["client", "./test_normal.txt"], 0x7ffdf16804d8 /* 61 vars */) = 0

[pid 6302] brk(NULL) = 0x5605362f2000

[pid 6302] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

[pid 6302] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

[pid 6302] fstat(3, {st_mode=S_IFREG|0644, st_size=162095, ...}) = 0

[pid 6302] mmap(NULL, 162095, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fe938a1e000

[pid 6302] close(3) = 0

[pid 6302] openat(AT_FDCWD, "/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

[pid 6302] read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0000x\2\0\0\0\0\0"..., 832) = 832

[pid 6302] 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"..., 896, 64) = 896

[pid 6302] fstat(3, {st_mode=S_IFREG|0755, st_size=2149728, ...}) = 0

[pid 6302] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe938a1c000

[pid 6302] 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"..., 896, 64) = 896

[pid 6302] mmap(NULL, 2174000, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fe938800000

[pid 6302] mmap(0x7fe938824000, 1515520, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x24000) = 0x7fe938824000

[pid 6302] mmap(0x7fe938996000, 454656, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x196000) = 0x7fe938996000

[pid 6302] mmap(0x7fe938a05000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x204000) = 0x7fe938a05000

[pid 6302] mmap(0x7fe938a0b000, 31792, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fe938a0b000

[pid 6302] close(3) = 0

[pid 6302] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe938a19000

[pid 6302] arch_prctl(ARCH_SET_FS, 0x7fe938a19740) = 0

[pid 6302] set_tid_address(0x7fe938a19a10) = 6302

[pid 6302] set_robust_list(0x7fe938a19a20, 24) = 0

[pid 6302] rseq(0x7fe938a19680, 0x20, 0, 0x53053053) = 0

[pid 6302] mprotect(0x7fe938a05000, 16384, PROT_READ) = 0

[pid 6302] mprotect(0x560512053000, 4096, PROT_READ) = 0

[pid 6302] mprotect(0x7fe938a85000, 8192, PROT_READ) = 0

[pid 6302] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

[pid 6302] getrandom("\x0d\x01\xa5\xb5\x37\x5b\xd7\xe1", 8, GRND_NONBLOCK) = 8

[pid 6302] munmap(0x7fe938a1e000, 162095) = 0

[pid 6302] openat(AT_FDCWD, "/dev/shm/example_sh_memory", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 3

[pid 6302] mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7fe938a45000

[pid 6302] openat(AT_FDCWD, "/dev/shm/sem.example_semaphore", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 4

[pid 6302] fstat(4, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0

[pid 6302] brk(NULL)   = 0x5605362f2000

[pid 6302] brk(0x560536313000)   = 0x560536313000

[pid 6302] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7fe938a44000

[pid 6302] close(4)   = 0

[pid 6302] openat(AT_FDCWD, "./test_normal.txt", O_RDONLY) = 4

[pid 6302] read(4, "12 3 4\n20 5 2\n100 10 2 5\n8 2\n15 "..., 4091) = 36

[pid 6302] clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000}, NULL) = 0

[pid 6299] futex(0x7f82ab705000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 6302] futex(0x7fe938a44000, FUTEX_WAKE, 1 <unfinished ...>

[pid 6299] <... futex resumed>)   = -1 EAGAIN (Resource temporarily unavailable)

[pid 6302] <... futex resumed>)   = 0

[pid 6299] write(1, "1\n", 21

)   = 2

[pid 6302] clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000}, NULL) = 0

[pid 6299] futex(0x7f82ab705000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 6302] futex(0x7fe938a44000, FUTEX_WAKE, 1 <unfinished ...>

[pid 6299] <... futex resumed>) = -1 EAGAIN (Resource temporarily unavailable)

[pid 6302] <... futex resumed>) = 0

[pid 6299] write(1, "2\n", 22

<unfinished ...>

[pid 6302] clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000} <unfinished ...>

[pid 6299] <... write resumed>) = 2

[pid 6302] <... clock_nanosleep resumed>, NULL) = 0

[pid 6299] futex(0x7f82ab705000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 6302] futex(0x7fe938a44000, FUTEX_WAKE, 1 <unfinished ...>

[pid 6299] <... futex resumed>) = -1 EAGAIN (Resource temporarily unavailable)

[pid 6302] <... futex resumed>) = 0

[pid 6299] write(1, "1\n", 21

<unfinished ...>

[pid 6302] clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000} <unfinished ...>

[pid 6299] <... write resumed>) = 2

[pid 6302] <... clock_nanosleep resumed>, NULL) = 0

[pid 6299] write(1, "4\n", 2 <unfinished ...>

4

[pid 6302] clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=0, tv_nsec=10000000} <unfinished ...>

[pid 6299] <... write resumed>) = 2

[pid 6302] <... clock_nanosleep resumed>, NULL) = 0

[pid 6299] futex(0x7f82ab705000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 6302] futex(0x7fe938a44000, FUTEX_WAKE, 1 <unfinished ...>

[pid 6299] <... futex resumed>) = -1 EAGAIN (Resource temporarily unavailable)

[pid 6302] <... futex resumed>) = 0

[pid 6299] write(1, "1\n", 21

<unfinished ...>

[pid 6302] read(4 <unfinished ...>

[pid 6299] <... write resumed>) = 2

[pid 6302] <... read resumed>, "", 4091) = 0

[pid 6299] futex(0x7f82ab705000, FUTEX_WAKE, 1 <unfinished ...>

[pid 6302] futex(0x7fe938a44000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 6299] <... futex resumed>) = 0

[pid 6302] <... futex resumed>) = -1 EAGAIN (Resource temporarily unavailable)

[pid 6299] futex(0x7f82ab705000, FUTEX_WAKE, 1 <unfinished ...>

[pid 6302] close(4 <unfinished ...>

[pid 6299] <... futex resumed>) = 0

[pid 6302] <... close resumed>) = 0

[pid 6299] wait4(6302 <unfinished ...>

[pid 6302] munmap(0x7fe938a44000, 32) = 0

[pid 6302] munmap(0x7fe938a45000, 4096) = 0

[pid 6302] close(3) = 0

[pid 6302] exit_group(0) = ?

[pid 6302] +++ exited with 0 +++

<... wait4 resumed>, NULL, 0, NULL) = 6302

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=6302, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

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

munmap(0x7f82ab705000, 32) = 0

munmap(0x7f82ab708000, 4096) = 0

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

close(3) = 0

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В ходе выполнения лабораторной работы была успешно реализована программа межпроцессного взаимодействия с использованием механизмов разделяемой памяти и семафоров.

Столкнулся с проблемами синхронизации работы процессов, которая решилась использованием системного вызова *sleep()*, и невозможностью выполнения программы из-за ранее созданного семафора, который необходимо было удалить перед работой программы.