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-211Б-23

Студент: Рожков И.С.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 04.12.24

Москва, 2024

Постановка задачи

Вариант 21.

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в shared_memory_1 или в shared_memory_2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Правило фильтрации: нечетные строки отправляются в shared_memory_1 , четные в shared_memory_2 . Дочерние процессы инвертируют строки.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork(void); – создает дочерний процесс.
- pid_t getpid(void); – возвращает ID вызывающего процесса.
- int open(const char *__file, int __oflag, …); – используется для открытия файла для чтения, записи или и того, и другого.
- ssize_t write(int __fd, const void *__buf, size_t __n); – Записывает N байт из буфер(BUF) в файл (FD). Возвращает количество записанных байт или -1.
- void exit(int __status); – выполняет немедленное завершение программы. Все используемые программой потоки закрываются, и временные файлы удаляются, управление возвращается ОС или другой программе.
- int close(int __fd); – сообщает операционной системе об окончании работы с файловым дескриптором, и закрывает файл(FD).
- int execv(const char *__path, char *const *__argv); – заменяет образ текущего процесса на образ нового процесса, определённого в пути path.
- ssize_t read(int __fd, void *__buf, size_t __nbytes); – считывает указанное количество байт из файла(FD) в буфер(BUF).
- pid_t wait(int *__stat_loc); – используются для ожидания изменения состояния процесса-потомка вызвавшего процесса и получения информации о потомке, чьё состояние изменилось.
- int shm_open(const char *name, int oflag, mode_t mode); – создает и открывает новый (или открывает уже существующий) объект разделяемой памяти POSIX.
- int shm_unlink(const char *name); – удаляется имя объекта разделяемой памяти и, как только все процессы завершили работу с объектом и отменили его распределение, очищают пространство и уничтожают связанную с ним область памяти.
- int ftruncate(int fd, off_t length); – устанавливают длину файла с файловым дескриптором fd в length байт.
- void * mmap(void *start, size_t length, int prot , int flags, int fd, off_t offset); – отражает length байтов, начиная со смещения offset файла (или другого объекта), определенного файловым дескриптором fd, в память, начиная с адреса start.

- int munmap(void *start, size_t length); – удаляет все отражения из заданной области памяти, после чего все ссылки на данную область будут вызывать ошибку "неправильное обращение к памяти".
- sem_t *sem_open(const char *name, int oflag); ИЛИ sem_t *sem_open(const char *name, int oflag, mode_t mode, unsigned int value); – создаёт новый семафор или открывает уже существующий.
- int sem_wait(sem_t *sem); – уменьшает значение семафора на 1. Если семафор в данный момент имеет нулевое значение, то вызов блокируется до тех пор, пока либо не станет возможным выполнить уменьшение.
- int sem_post(sem_t *sem); – увеличивает значение семафора на 1.
- int sem_unlink(const char *name); – удаляет имя семафора из системы. После вызова этой функции другие процессы больше не смогут открыть этот семафор по имени.
- int sem_close(sem_t *sem); – закрывает указанный семафор, освобождая ресурсы, связанные с ним.

Программа server.c получает на вход два аргумента – пути к файлам, в которые требуется записать результат работы. С помощью readlink() сохраняем полный путь до файла. Создаём две общих памяти для общения с двумя дочерними процессами. А также создаём по два семафора для каждой общей памяти, для контроля очерёдности доступа. Далее выполняется fork() для создания первого дочернего процесса, с помощью конструкции switch/case определяем в каком процессе мы находимся.

Если процесс дочерний, с помощью execv() подменяем образ текущего процесса на новый(client), передавая все нужные данные для доступа к памяти и семафорам .

Если процесс – родитель, то делаем ещё один fork(), далее повторяем те же действия, если мы в дочернем процессе. Если же мы родитель, то начинаем читать строки из потока ввода и по очереди передавать то первому дочернему процессу, то второму, ожидания разблокировки семафора на запись. После отправления очередной строки, открывается соответствующий семафор на чтение. После окончания ввода закрываем всю общую память, семафоры и ждём завершения обоих дочерних процессов, программа завершается.

Программа client открывает переданный в качестве аргумента файл, получает доступ к общей памяти и семафорам, после этого считывает строки из общей памяти, каждый раз ожидая открытия семафора на чтение, переворачивает и записывает в открытый файл. При окончании ввода строк файл закрывается, программа завершается.

Таким образом, контроль доступа над общей памятью реализуется путём некого “жонглирования” двумя семафорами.

Код программы

server.c

```c
#include "lib.h"


static char CLIENT_PROGRAM_NAME[] = "client";


int main(int argc, char **argv)
```

```
{
    int shm_fd1, shm_fd2;
    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';
    }

    if ((shm_fd1 = shm_open(SHM_NAME_1, O_CREAT | O_RDWR, 0666)) == -1)
    {
        const char msg[] = "error: failed to open shared memory 1";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
    if (ftruncate(shm_fd1, BUFFER_SIZE) == -1)
```

```
    {
        const char msg[] = "error: failed to set size for shared memory 1";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
    if ((shm_fd2 = shm_open(SHM_NAME_2, O_CREAT | O_RDWR, 0666)) == -1)
    {
        const char msg[] = "error: failed to open shared memory 2";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
    if (ftruncate(shm_fd2, BUFFER_SIZE) == -1)
    {
        const char msg[] = "error: failed to set size for shared memory 1";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    char *shared_memory1 = mmap(0, BUFFER_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd1, 0);
    char *shared_memory2 = mmap(0, BUFFER_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd2, 0);
    if (shared_memory1 == MAP_FAILED || shared_memory2 == MAP_FAILED)
    {
        const char msg[] = "error: failed to map shared memory";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    sem_t *sem_write1 = sem_open(SEM_WRITE_1, O_CREAT | O_EXCL, 0644, 1);
    sem_t *sem_read1 = sem_open(SEM_READ_1, O_CREAT | O_EXCL, 0644, 0);

    sem_t *sem_write2 = sem_open(SEM_WRITE_2, O_CREAT | O_EXCL, 0644, 1);
    sem_t *sem_read2 = sem_open(SEM_READ_2, O_CREAT | O_EXCL, 0644, 0);
```

```
    if (sem_write1 == SEM_FAILED || sem_read1 == SEM_FAILED || sem_write1 ==
SEM_FAILED || sem_write2 == SEM_FAILED)
    {
        const char msg[] = "error: failed to open semaphore";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    const pid_t child_1 = fork();
    switch (child_1)
    {
    case -1:
    {
        const char msg[] = "error: failed to spawn new process\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
    break;

    case 0:
    {
        pid_t pid = getpid();

        {
            char msg[64];
            const int32_t length = snprintf(msg, sizeof(msg),
                                            "%d: I'm a child1\n", pid);
            write(STDOUT_FILENO, msg, length);
        }

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

```
            char *const args[] = {CLIENT_PROGRAM_NAME, argv[1], SEM_WRITE_1,
SEM_READ_1, SHM_NAME_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:
    {
        const pid_t child_2 = fork();


        switch (child_2)
        {
        case -1:
        {
            const char msg[] = "error: failed to spawn new process\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }
        break;


        case 0:
        {
            pid_t pid = getpid();


            {
```

```
                char msg[64];
                const int32_t length = snprintf(msg, sizeof(msg),
                                                "%d: I'm a child2\n", pid);
                write(STDOUT_FILENO, msg, length);
            }

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

                char *const args[] = {CLIENT_PROGRAM_NAME, argv[2], SEM_WRITE_2,
SEM_READ_2, SHM_NAME_2, 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:
        {
            pid_t pid = getpid();

            {
                char msg[128];
                const int32_t length = snprintf(msg, sizeof(msg),
                                                "%d: I'm a parent, my child1 &
child2 has PID %d %d\n", pid, child_1, child_2);
                write(STDOUT_FILENO, msg, length);
```

```
        }

        char buf[BUFFER_SIZE];
        ssize_t bytes;
        int odd = 1;
        {
            sleep(1);
            const char msg[] = "Input strings:\n";
            write(STDOUT_FILENO, msg, sizeof(msg));
        }
        while (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);
            }
            else if (buf[0] == '\n')
            {
                break;
            }
            {
                buf[bytes - 1] = '\0';
                int32_t written;
                if (odd)
                {
                    sem_wait(sem_write1);
                    snprintf(shared_memory1, BUFFER_SIZE, buf);
                    sem_post(sem_read1);
                }

                else
                {
```

```
                sem_wait(sem_write2);
                snprintf(shared_memory2, BUFFER_SIZE, buf);
                sem_post(sem_read2);
            }

            odd = abs(odd - 1);
        }
    }
    buf[0] = '\n';
    sem_wait(sem_write1);
    snprintf(shared_memory1, BUFFER_SIZE, buf);
    sem_post(sem_read1);
    sem_wait(sem_write2);
    snprintf(shared_memory2, BUFFER_SIZE, buf);
    sem_post(sem_read2);

    if (sem_close(sem_write1) == -1 ||
        sem_close(sem_write2) == -1 ||
        sem_close(sem_read1) == -1 ||
        sem_close(sem_read2) == -1)
    {
        const char msg[] = "error: failed to sem_close";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

    if (sem_unlink(SEM_WRITE_1) == -1 ||
        sem_unlink(SEM_WRITE_2) == -1 ||
        sem_unlink(SEM_READ_1) == -1 ||
        sem_unlink(SEM_READ_2) == -1)
    {
        const char msg[] = "error: failed to sem_unlink";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
```

```
            }

            if (munmap(shared_memory1, BUFFER_SIZE) || munmap(shared_memory2,
BUFFER_SIZE))
            {
                const char msg[] = "error: failed to munmap";
                write(STDERR_FILENO, msg, sizeof(msg));
                exit(EXIT_FAILURE);
            }

            if (shm_unlink(SHM_NAME_1) == -1 || shm_unlink(SHM_NAME_2) == -1)
            {
                const char msg[] = "error: failed to shm_unlink";
                write(STDERR_FILENO, msg, sizeof(msg));
                exit(EXIT_FAILURE);
            }

            int child_status;
            pid_t wpid;
            while ((wpid = wait(&child_status)) > 0)
            {
                if (child_status != EXIT_SUCCESS)
                {
                    const char msg[] = "error: child exited with error\n";
                    write(STDERR_FILENO, msg, sizeof(msg));
                    exit(child_status);
                }
            }
        }
        break;
        }
        break;
    }
    }
    return 0;
```

```
}
```

client.c

```
#include "lib.h"
#include "string.h"


void str_reverse(char *str)
{
    int len = strlen(str);
    for (int i = 0; i < len / 2; ++i)
    {
        char temp = str[i];
        str[i] = str[len - 1 - i];
        str[len - 1 - i] = temp;
    }
}


int main(int argc, char **argv)
{
    char buf[BUFFER_SIZE];
    ssize_t bytes;


    pid_t pid = getpid();


    // NOTE: `O_WRONLY` only enables file for writing
    // NOTE: `O_CREAT` creates the requested file if absent
    // NOTE: `O_TRUNC` empties the file prior to opening
    // NOTE: `O_APPEND` subsequent writes are being appended instead of
overwritten
    int32_t file = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC | O_APPEND, 0600);
    if (file == -1)
    {
        const char msg[] = "error: failed to open requested file\n";
        write(STDERR_FILENO, msg, sizeof(msg));
```

```
    exit(EXIT_FAILURE);
}

int shm_fd = shm_open(argv[4], O_RDONLY, 0666);
if (shm_fd == -1)
{
    const char msg[] = "error: failed to open shared memory";
    write(STDERR_FILENO, msg, sizeof(msg));
    exit(EXIT_FAILURE);
}
char *shared_memory = mmap(0, BUFFER_SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);
if (shared_memory == MAP_FAILED)
{
    const char msg[] = "error: failed to map shared memory";
    write(STDERR_FILENO, msg, sizeof(msg));
    exit(EXIT_FAILURE);
}

sem_t *sem_write = sem_open(argv[2], 0);
sem_t *sem_read = sem_open(argv[3], 0);
if (sem_write == SEM_FAILED || sem_read == SEM_FAILED)
{
    const char msg[] = "error: client failed to open semaphore";
    write(STDERR_FILENO, msg, sizeof(msg));
    exit(EXIT_FAILURE);
}

char flag = 1;
do
{
    sem_wait(sem_read);
    if (*shared_memory == '\n')
    {
        flag = 0;
```

```
        }
        else
        {

            bytes = strlen(shared_memory) + 1;
            strcpy(buf, shared_memory);

            sem_post(sem_write);

            buf[bytes - 1] = '\0';
            str_reverse(buf);
            buf[bytes - 1] = '\n';

            int32_t written = write(file, buf, bytes);
            if (written != bytes)
            {
                const char msg[] = "error: client failed to write to file\n";
                write(STDERR_FILENO, msg, sizeof(msg));
                exit(EXIT_FAILURE);
            }
        }
    } while (flag);

    if (close(file) == -1)
    {
        const char msg[] = "error: client failed to close file\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
    if (sem_close(sem_write) == -1 || sem_close(sem_read) == -1)
    {
        const char msg[] = "error: client failed to sem_close\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
```

```
    }

    if (munmap(shared_memory, BUFFER_SIZE) == -1)
    {
        const char msg[] = "error: client failed to munmap";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }
    return 0;
}
```

lib.h

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

#define SHM_NAME_1 "/shm1"
#define SHM_NAME_2 "/shm2"

#define SEM_WRITE_1 "/sw1"
#define SEM_WRITE_2 "/sw2"

#define SEM_READ_1 "/sr1"
#define SEM_READ_2 "/sr2"

#define BUFFER_SIZE 1024
```

Протокол работы программы

```
$ ./server f1.txt f2.txt
295436: I'm a parent, my child1 & child2 has PID 295437 295438
Input strings:
295437: I'm a child1
295438: I'm a child2
string 1
string 2
string 3
string 4
string 5
string 6
last string

$ cat f1.txt
1 gnirts
3 gnirts
5 gnirts
gnirts tsal

$ cat f2.txt
2 gnirts
4 gnirts
6 gnirts

$ ./server f1.txt f2.txt
30233: I'm a parent, my child1 & child2 has PID 30234 30235
30234: I'm a child1
30235: I'm a child2
Input strings:
vaaaaaaaaaaaaaa vaaaaaaaaaaaa
goooooooooooooooooooooooooool gol ggg       gooooool
one two three
1 2 3
1               9                 8                     3
3 3 3 3 3 5656565656
ggggggggggggggggggggggggggggggggggggggggggggggggggooooooooooooooooooooooooooooooooooooo
jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL

$ cat f1.txt
aaaaaaaaaaaav aaaaaaaaaaaaaaav
eerht owt eno
3                     8                 9               1
ooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooggggggggggggggggggggggggggggggg
$cat f2.txt
loooooog       ggg log looooooooooooooooooooooooooog
3 2 1
```

```
6565656565 3 3 3 3 3
LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj


$ strace ./server f1.txt f2.txt

execve("./server", ["./server", "f1.txt", "f2.txt"], 0x7ffda1244bd0 /* 35 vars */) = 0

brk(NULL)                               = 0x55cd1200a000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffdea481d50) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fc5763da000

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=18103, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 18103, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fc5763d5000

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\237\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

pread64(3, "\4\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\0"..., 48,
848) = 48

pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 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\0"..., 784, 64)
= 784

mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fc5761ac000

mprotect(0x7fc5761d4000, 2023424, PROT_NONE) = 0

mmap(0x7fc5761d4000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x7fc5761d4000

mmap(0x7fc576369000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1bd000) = 0x7fc576369000

mmap(0x7fc5763c2000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x215000) = 0x7fc5763c2000

mmap(0x7fc5763c8000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7fc5763c8000

close(3)                                = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fc5761a9000

arch_prctl(ARCH_SET_FS, 0x7fc5761a9740) = 0

set_tid_address(0x7fc5761a9a10)         = 31496
```

```
set_robust_list(0x7fc5761a9a20, 24)     = 0
rseq(0x7fc5761aa0e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7fc5763c2000, 16384, PROT_READ) = 0
mprotect(0x55ccecf88000, 4096, PROT_READ) = 0
mprotect(0x7fc576414000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7fc5763d5000, 18103)           = 0
readlink("/proc/self/exe", "/home/empress/OS_labs/lab_3/src/"..., 1023) = 38
openat(AT_FDCWD, "/dev/shm/shm112", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 3
ftruncate(3, 1024)                      = 0
openat(AT_FDCWD, "/dev/shm/shm222", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 4
ftruncate(4, 1024)                      = 0
mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7fc576413000
mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7fc5763d9000
getrandom("\x95\xae\xcd\xe0\x8c\x2a\xfa\xe6", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.PqvlrU", 0x7ffdea481190, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.PqvlrU", O_RDWR|O_CREAT|O_EXCL, 0644) = 5
write(5, "\1\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", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7fc5763d8000
link("/dev/shm/sem.PqvlrU", "/dev/shm/sem.sw112") = 0
newfstatat(5, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0
getrandom("\xe9\x5b\x62\x32\x6a\x2c\x46\x02", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x55cd1200a000
brk(0x55cd1202b000)                     = 0x55cd1202b000
unlink("/dev/shm/sem.PqvlrU")           = 0
close(5)                                = 0
getrandom("\x89\x9a\x9c\x5a\xdd\xb9\x13\xf5", 8, GRND_NONBLOCK) = 8
getrandom("\x53\xab\xbe\x3b\xbf\xd3\xa2\x8a", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.vBpbwB", 0x7ffdea481190, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.vBpbwB", O_RDWR|O_CREAT|O_EXCL, 0644) = 5
write(5, "\0\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", 32) = 32
mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7fc5763d7000
link("/dev/shm/sem.vBpbwB", "/dev/shm/sem.sr112") = 0
newfstatat(5, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0
```

```
     unlink("/dev/shm/sem.vBpbwB")           = 0

     close(5)                                = 0

     getrandom("\x80\x41\x97\x39\xa3\xc2\x0e\x82", 8, GRND_NONBLOCK) = 8

     newfstatat(AT_FDCWD, "/dev/shm/sem.QDwhjV", 0x7ffdea481190, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (No such file or directory)

     openat(AT_FDCWD, "/dev/shm/sem.QDwhjV", O_RDWR|O_CREAT|O_EXCL, 0644) = 5

     write(5, "\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, 5, 0) = 0x7fc5763d6000

     link("/dev/shm/sem.QDwhjV", "/dev/shm/sem.sw222") = 0

     newfstatat(5, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0

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

     close(5)                                = 0

     getrandom("\x38\x90\x1d\xeb\xc3\xb7\x0a\xb2", 8, GRND_NONBLOCK) = 8

     newfstatat(AT_FDCWD, "/dev/shm/sem.i0UA3e", 0x7ffdea481190, AT_SYMLINK_NOFOLLOW) = -1
ENOENT (No such file or directory)

     openat(AT_FDCWD, "/dev/shm/sem.i0UA3e", O_RDWR|O_CREAT|O_EXCL, 0644) = 5

     write(5, "\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, 5, 0) = 0x7fc5763d5000

     link("/dev/shm/sem.i0UA3e", "/dev/shm/sem.sr222") = 0

     newfstatat(5, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0

     unlink("/dev/shm/sem.i0UA3e")           = 0

     close(5)                                = 0

     clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7fc5761a9a10) = 31497

     31497: I'm a child1

     clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7fc5761a9a10) = 31498

     getpid()                                = 31496

     31498: I'm a child2

     write(1, "31496: I'm a parent, my child1 &"..., 6031496: I'm a parent, my child1 &
child2 has PID 31497 31498

     ) = 60

     clock_nanosleep(CLOCK_REALTIME, 0, {tv_sec=1, tv_nsec=0}, 0x7ffdea4814b0) = 0

     write(1, "Input strings:\n\0", 16Input strings:

     )      = 16

     read(0, 123

     "123\n", 1024)                  = 4

     futex(0x7fc5763d7000, FUTEX_WAKE, 1)    = 1
```

```
read(0, 123
"123\n", 1024)                  = 4
futex(0x7fc5763d5000, FUTEX_WAKE, 1)    = 1
read(0, 456
"456\n", 1024)                  = 4
futex(0x7fc5763d7000, FUTEX_WAKE, 1)    = 1
read(0, 456
"456\n", 1024)                  = 4
futex(0x7fc5763d5000, FUTEX_WAKE, 1)    = 1
read(0, tyu
"tyu\n", 1024)                  = 4
futex(0x7fc5763d7000, FUTEX_WAKE, 1)    = 1
read(0, ghj
"ghj\n", 1024)                  = 4
futex(0x7fc5763d5000, FUTEX_WAKE, 1)    = 1
read(0, cvbb
"cvbb\n", 1024)                 = 5
futex(0x7fc5763d7000, FUTEX_WAKE, 1)    = 1
read(0, fggh
"fggh\n", 1024)                 = 5
futex(0x7fc5763d5000, FUTEX_WAKE, 1)    = 1
read(0,
"\n", 1024)                     = 1
futex(0x7fc5763d7000, FUTEX_WAKE, 1)    = 1
futex(0x7fc5763d5000, FUTEX_WAKE, 1)    = 1
munmap(0x7fc5763d8000, 32)              = 0
munmap(0x7fc5763d6000, 32)              = 0
munmap(0x7fc5763d7000, 32)              = 0
munmap(0x7fc5763d5000, 32)              = 0
unlink("/dev/shm/sem.sw112")            = 0
unlink("/dev/shm/sem.sw222")            = 0
unlink("/dev/shm/sem.sr112")            = 0
unlink("/dev/shm/sem.sr222")            = 0
munmap(0x7fc576413000, 1024)            = 0
munmap(0x7fc5763d9000, 1024)            = 0
```

```
     unlink("/dev/shm/shm112")               = 0

     unlink("/dev/shm/shm222")               = 0

     wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 31497

     --- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=31497, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---

     wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 31498

     wait4(-1, 0x7ffdea481500, 0, NULL)      = -1 ECHILD (No child processes)

     exit_group(0)                           = ?

     +++ exited with 0 +++
```

Вывод

В ходе написания данной лабораторной работы я научился работать с новыми системными вызовами в СИ, которые используются для работы с семафорами и shared memory. Научился передавать данные посредством shared memory и контролировать доступ через семафоры. Проблем во время написания лабораторной работы не возникло.