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: Савков И.И. Преподаватель: Бахарев В.Д. Оценка:

Дата: 10.12.24

Москва, 2024

**Вариант 22.**

# Постановка задачи

**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: с вероятностью 80% строки отправляются в shared\_memory\_1 , иначе 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 `parent.c` получает на вход два аргумента — пути к файлам, в которые требуется записать результат работы. Эти файлы открываются на запись с помощью `open()`.

Создается разделяемая память размером 4096 байт с помощью функции `mmap()`. Эта память будет использоваться для передачи данных между родительским процессом и двумя дочерними процессами. Для синхронизации доступа создаются два именованных семафора (`/sem1` и `/sem2`) с помощью `sem\_open()`. Эти семафоры будут использоваться для управления доступом к общей памяти.

Далее выполняются два вызова `fork()`, которые создают два дочерних процесса. В каждом дочернем процессе выполняется подмена образа текущего процесса на исполняемый файл `child` с помощью

`execv()`. Вызов `execv()` передает информацию о номере дочернего процесса, чтобы определить, с каким семафором и какой частью логики он будет работать.

Родительский процесс считывает строки из стандартного ввода. Каждая строка записывается в разделяемую память, после чего открывается соответствующий семафор с помощью `sem\_post()`. Первый дочерний процесс ожидает сигнал от семафора `/sem1`, считывает данные из памяти, переворачивает строку и записывает результат в первый файл. Второй дочерний процесс выполняет аналогичные действия, используя семафор `/sem2` и второй файл.

После окончания ввода (определяется через `CTRL+D`), родительский процесс передает специальный символ конца файла (EOF) в разделяемую память и разблокирует оба семафора, чтобы уведомить дочерние процессы о завершении работы. Затем родительский процесс закрывает разделяемую память, удаляет семафоры и ожидает завершения обоих дочерних процессов.

Программа `child.c` открывает соответствующий семафор (`/sem1` или `/sem2`) и получает доступ к общей памяти. Она ожидает сигнала от семафора, считывает строку из разделяемой памяти, переворачивает ее и записывает в файл. Если строка содержит символ EOF, процесс завершает работу, закрывает семафор и освобождает память.

Таким образом, синхронизация между процессами осуществляется с помощью семафоров, которые обеспечивают контроль последовательности выполнения операций чтения и записи в общую память.

# Код программы

parent**.c**

#include "pool.h"

char \*get\_row(char \*symbol);

int main(int argc, char \*argv[]) { if (argc != 3) {

const char \*msg\_error = "[PARENT] ERROR: INVALID\_INPUT.\n"; write(STDERR\_FILENO, msg\_error, strlen(msg\_error)); exit(INVALID\_INPUT);

}

// Создаем на запись файл для Дочернего процесса 1 char \*input\_path1 = argv[1];

int32\_t file1 = open(input\_path1, O\_WRONLY | O\_TRUNC | 0600); if (file1 == -1) {

const char msg[] = "[PARENT] ERROR: failed to open requested file\n"; write(STDERR\_FILENO, msg, sizeof(msg));

exit(INVALID\_FILES);

}

// Создаем на запись файл для Дочернего процесса 2 char \*input\_path2 = argv[2];

int32\_t file2 = open(input\_path2, O\_WRONLY | O\_TRUNC | 0600); if (file2 == -1) {

const char msg[] = "[PARENT] ERROR: failed to open requested file\n"; write(STDERR\_FILENO, msg, sizeof(msg));

exit(INVALID\_FILES);

}

// Создаем общую память

const size\_t SHM\_SIZE = BUFSIZE;

// Используем именованную разделяемую память const char \*shm\_name = SHARED\_MEMORY\_NAME;

int shm\_fd = shm\_open(shm\_name, O\_CREAT | O\_RDWR, 0666); ftruncate(shm\_fd, SHM\_SIZE);

char \*shm = mmap(NULL, SHM\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

if (shm == MAP\_FAILED) {

const char msg[] = "[PARENT] ERROR: MEMORY\_ERROR.\n"; write(STDERR\_FILENO, msg, sizeof(msg)); exit(MEMORY\_ERROR);

}

// Создаем семафоры

sem\_t \*sem1 = sem\_open(SEM\_NAME1, O\_CREAT, 0600, 1); sem\_t \*sem2 = sem\_open(SEM\_NAME2, O\_CREAT, 0600, 1); if (sem1 == SEM\_FAILED || sem2 == SEM\_FAILED) {

const char msg[] = "[PARENT] ERROR: SEMAPHORE\_ERROR.\n"; write(STDERR\_FILENO, msg, sizeof(msg)); exit(MEMORY\_ERROR);

}

// Создаем дочерний процесс 1 const pid\_t child1 = fork(); if (child1 == -1) {

const char \*msg\_error = "[PARENT] ERROR: INVALID\_FORK.\n"; write(STDERR\_FILENO, msg\_error, strlen(msg\_error)); close(file1);

close(file2); exit(ERROR\_FORK);

}

// Дочерний процесс 1 if (child1 == 0) {

dup2(file1, STDOUT\_FILENO);

// Дочерний процесс 1

char \*const args[] = {"child1", "1", NULL}; sem\_wait(sem1); // Ждем семафор

execv("./child1", args);

const char \*msg\_error = "[PARENT] ERROR: ERROR\_EXECV1\n"; write(STDERR\_FILENO, msg\_error, strlen(msg\_error)); exit(ERROR\_EXECV);

}

// Создаем дочерний процесс 2 pid\_t child2 = fork();

if (child2 == -1) {

const char \*msg\_error = "[PARENT] ERROR: INVALID\_FORK.\n"; write(STDERR\_FILENO, msg\_error, strlen(msg\_error)); close(file1);

close(file2); exit(ERROR\_FORK);

}

// Дочерний процесс 2 if (child2 == 0) {

dup2(file2, STDOUT\_FILENO);

// Дочерний процесс 2

char \*const args[] = {"child2", "2", NULL}; sem\_wait(sem2); // Ждем семафор execv("./child2", args);

const char \*msg\_error = "[PARENT] ERROR: ERROR\_EXECV2\n"; write(STDERR\_FILENO, msg\_error, strlen(msg\_error)); exit(ERROR\_EXECV);

}

// Считываем из буфера ввода, пока не встретим EOF

char \*msg = "Please enter the lines you want to invert. Press 'CTRL + D' to exit.\n"; write(STDOUT\_FILENO, msg, strlen(msg));

srand(time(NULL)); char symbol = '0'; while (symbol != EOF) {

int random\_number = rand() % 100; char \*buf = get\_row(&symbol);

if (buf == NULL) {

const char \*msg\_error = "ERROR: MEMORY\_ERROR\n"; write(STDERR\_FILENO, msg\_error, strlen(msg\_error)); free(buf);

close(file1); close(file2); exit(MEMORY\_ERROR);

}

if (symbol == EOF) { break;

}

char msg\_sem[512];

if (random\_number < 80) { snprintf(shm, SHM\_SIZE, "%s", buf);

sem\_post(sem1); // Отправляем сигнал дочернему процессу 1 uint32\_t len\_msg = snprintf(msg\_sem, sizeof(msg\_sem) - 1,

"[PARENT] Sent to child1: %s\n", buf); write(STDOUT\_FILENO, msg\_sem, len\_msg);

} else {

snprintf(shm, SHM\_SIZE, "%s", buf);

sem\_post(sem2); // Отправляем сигнал дочернему процессу 2 uint32\_t len\_msg = snprintf(msg\_sem, sizeof(msg\_sem) - 1,

"[PARENT] Sent to child2: %s\n", buf); write(STDOUT\_FILENO, msg\_sem, len\_msg);

}

free(buf);

}

symbol = EOF; strcpy(shm, &symbol);

sem\_post(sem1); sem\_post(sem2);

// Закрываем семафоры sem\_close(sem1); sem\_close(sem2);

sem\_unlink(SEM\_NAME1); sem\_unlink(SEM\_NAME2);

// Освобождаем общую память munmap(shm, SHM\_SIZE); shm\_unlink(shm\_name);

close(file1); close(file2);

return OK;

}

char \*get\_row(char \*symbol) { int size = 0;

int capacity = 2;

char \*buf = (char \*) malloc(sizeof(char) \* capacity); if (buf == NULL) {

return NULL;

}

\*symbol = (char) getchar();

while (\*symbol != '\n' && \*symbol != EOF) { if (size == capacity) {

capacity \*= 2;

char \*buffer\_realloc = (char \*) realloc(buf, sizeof(char) \* capacity); if (buffer\_realloc == NULL) {

free(buf); return NULL;

}

buf = buffer\_realloc;

}

buf[size] = \*symbol; size++;

\*symbol = (char) getchar();

}

buf[size] = '\0'; return buf;

}

**child.c**

#include "pool.h"

void reverse\_string(char \*str) { int len = strlen(str);

for (int i = 0; i < len / 2; ++i) { char temp = str[i];

str[i] = str[len - i - 1]; str[len - i - 1] = temp;

}

}

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

if (argc != 2) {

const char \*msg\_error = "[CHILD] ERROR: INVALID\_INPUT.\n"; write(STDERR\_FILENO, msg\_error, strlen(msg\_error)); exit(EXIT\_FAILURE);

}

int child\_number = atoi(argv[1]); sem\_t \*sem;

if (child\_number == 1)

sem = sem\_open(SEM\_NAME1, 1);

else

sem = sem\_open(SEM\_NAME2, 1);

if (sem == SEM\_FAILED) {

const char \*msg\_error = "[CHILD] ERROR: SEMAPHORE\_ERROR.\n"; write(STDERR\_FILENO, msg\_error, strlen(msg\_error)); exit(EXIT\_FAILURE);

}

const size\_t SHM\_SIZE = 4096;

// Подключаемся к именованной разделяемой памяти

int shm\_fd = shm\_open(SHARED\_MEMORY\_NAME, O\_RDWR, 0666);

char \*shm = mmap(NULL, SHM\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0); if (shm == MAP\_FAILED) {

const char \*msg\_error = "[CHILD] ERROR: MEMORY\_ERROR.\n"; write(STDERR\_FILENO, msg\_error, strlen(msg\_error)); exit(EXIT\_FAILURE);

}

while (1) {

sem\_wait(sem); status = shm[0];

if (status == EOF) { break;

}

char \*row = shm;

reverse\_string(row); // Переворачиваем строку

write(STDOUT\_FILENO, row, strlen(row)); write(STDOUT\_FILENO, "\n", 1);

}

// После завершения работы munmap(shm, SHM\_SIZE); sem\_close(sem);

return 0;

}

**pool.h**

#ifndef POOL\_H #define POOL\_H

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

#define SHARED\_MEMORY\_NAME "/shared\_memory"

#define BUFSIZE 4096

#define SEM\_NAME1 "/sem1" #define SEM\_NAME2 "/sem2"

typedef enum { OK, INVALID\_INPUT, INVALID\_FILES, MEMORY\_ERROR, ERROR\_FORK, ERROR\_EXECV,

} state;

#endif //POOL\_H

# Протокол работы программы

goldglaid@GoldGlaid:~/OSLabs/lab3$ ./parent file1.txt file2.txt Please enter the lines you want to invert. Press 'CTRL + D' to exit. test1

[PARENT] Sent to child1: test1 test2

[PARENT] Sent to child1: test2 test3

[PARENT] Sent to child2: test3 adiaweiooqweoixzsudfasdaosdwuqeqidiasodoso1234

[PARENT] Sent to child1: adiaweiooqweoixzsudfasdaosdwuqeqidiasodoso1234 GOAAAAAAL

[PARENT] Sent to child1: GOAAAAAAL 0987654321

[PARENT] Sent to child1: 0987654321

goldglaid@GoldGlaid:~/OSLabs/lab3$ cat file1.txt 1tset

2tset 4321osodosaidiqequwdsoadsafduszxioewqooiewaida LAAAAAAOG

1234567890

goldglaid@GoldGlaid:~/OSLabs/lab3$ cat file2.txt 3tset

goldglaid@GoldGlaid:~/OSLabs/lab3$ strace ./parent file1.txt file2.txt execve("./parent", ["./parent", "file1.txt", "file2.txt"], 0x7ffce9a9c610 /\* 26 vars \*/) = 0 brk(NULL) = 0x560ab04da000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7fff04025b60) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f89ab352000

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=18567, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 18567, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f89ab34d000

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) = 0x7f89ab124000

mprotect(0x7f89ab14c000, 2023424, PROT\_NONE) = 0

mmap(0x7f89ab14c000, 1658880, PROT\_READ|PROT\_EXEC,

MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f89ab14c000

mmap(0x7f89ab2e1000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000)

= 0x7f89ab2e1000

mmap(0x7f89ab33a000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x7f89ab33a000 mmap(0x7f89ab340000, 52816, PROT\_READ|PROT\_WRITE,

MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f89ab340000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f89ab121000

arch\_prctl(ARCH\_SET\_FS, 0x7f89ab121740) = 0 set\_tid\_address(0x7f89ab121a10) = 82144

set\_robust\_list(0x7f89ab121a20, 24) = 0

rseq(0x7f89ab1220e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f89ab33a000, 16384, PROT\_READ) = 0

mprotect(0x560aac382000, 4096, PROT\_READ) = 0

mprotect(0x7f89ab38c000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

munmap(0x7f89ab34d000, 18567) = 0

openat(AT\_FDCWD, "file1.txt", O\_WRONLY|O\_EXCL|O\_NOCTTY|O\_TRUNC) = 3

openat(AT\_FDCWD, "file2.txt", O\_WRONLY|O\_EXCL|O\_NOCTTY|O\_TRUNC) = 4

openat(AT\_FDCWD, "/dev/shm/shared\_memory", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0666) = 5

**ftruncate(5, 4096) = 0**

**mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 5, 0) = 0x7f89ab38b000**

openat(AT\_FDCWD, "/dev/shm/sem.sem1", O\_RDWR|O\_NOFOLLOW) = 6 newfstatat(6, "", {st\_mode=S\_IFREG|0600, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0 getrandom("\xb6\xc4\xaa\x12\x62\x17\x48\x73", 8, GRND\_NONBLOCK) = 8 brk(NULL) = 0x560ab04da000

brk(0x560ab04fb000) = 0x560ab04fb000

**mmap(NULL, 32, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 6, 0) = 0x7f89ab351000**

close(6) = 0

openat(AT\_FDCWD, "/dev/shm/sem.sem2", O\_RDWR|O\_NOFOLLOW) = 6 newfstatat(6, "", {st\_mode=S\_IFREG|0600, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0

**mmap(NULL, 32, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 6, 0) = 0x7f89ab350000**

close(6) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f89ab121a10) = 82145

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f89ab121a10) = 82146

write(1, "Please enter the lines you want "..., 69Please enter the lines you want to invert. Press 'CTRL + D' to exit.

) = 69

newfstatat(0, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0x3), ...}, AT\_EMPTY\_PATH) = 0 read(0, Gooooaaaal

"Gooooaaaal\n", 1024) = 11

**futex(0x7f89ab351000, FUTEX\_WAKE, 1) = 1**

write(1, "[PARENT] Sent to child1: Gooooaa"..., 36[PARENT] Sent to child1: Gooooaaaal

) = 36

read(0, test1

"test1\n", 1024) = 6

**futex(0x7f89ab351000, FUTEX\_WAKE, 1) = 1**

write(1, "[PARENT] Sent to child1: test1\n", 31[PARENT] Sent to child1: test1

) = 31

read(0, test2

"test2\n", 1024) = 6

**futex(0x7f89ab351000, FUTEX\_WAKE, 1) = 1**

write(1, "[PARENT] Sent to child1: test2\n", 31[PARENT] Sent to child1: test2

) = 31

read(0, test3

"test3\n", 1024) = 6

**futex(0x7f89ab351000, FUTEX\_WAKE, 1) = 1**

write(1, "[PARENT] Sent to child1: test3\n", 31[PARENT] Sent to child1: test3

) = 31

read(0, asd

"asd\n", 1024) = 4

**futex(0x7f89ab350000, FUTEX\_WAKE, 1) = 1**

write(1, "[PARENT] Sent to child2: asd\n", 29[PARENT] Sent to child2: asd

) = 29

read(0, "", 1024) = 0

**futex(0x7f89ab351000, FUTEX\_WAKE, 1) = 1**

**futex(0x7f89ab350000, FUTEX\_WAKE, 1) = 1**

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=82145, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=82146, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

**munmap(0x7f89ab351000, 32) = 0**

**munmap(0x7f89ab350000, 32) = 0**

**unlink("/dev/shm/sem.sem1") = 0**

**unlink("/dev/shm/sem.sem2") = 0**

**munmap(0x7f89ab38b000, 4096) = 0**

**unlink("/dev/shm/shared\_memory") = 0**

close(3) = 0

close(4) = 0

exit\_group(0) = ?

+++ exited with 0 +++ goldglaid@GoldGlaid:~/OSLabs/lab3$

# 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.