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: Бугренков В.П. Преподаватель: Бахарев В.Д. Оценка:

Дата: 17.12.24

Москва, 2024

# Постановка задачи

Вариант 9. Цель работы

Приобретение практических навыков в:

* Управление процессами в ОС
* Обеспечение обмена данных между процессами посредством shared memory и memory mapping

Задание

В файле записаны команды вида:«число число число<endline>». 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

Использованные системные вызовы:

* pid\_t fork(void); – создает дочерний процесс.
* int shm\_open(const char \*\_\_name, int \_\_oflag, mode\_t \_\_mode) — открывает сегмент shm
* void \*mmap(void \*\_\_addr, size\_t \_\_len, int \_\_prot, int \_\_flags, int \_\_fd, off\_t \_\_offset) — создает новый маппинг в виртуальном адресном пространстве
* sem\_t \*sem\_open (const char \*\_\_name, int \_\_oflag, …) - открывает именнованный семафор
* int sem\_unlink (const char \*\_\_name) — удаляет именованный семафор
* int sem\_wait(sem\_t \*sem) - уменьшает (блокирует) семафо
* int sem\_post(sem\_t \*sem) - увеличивает (разблокирует) семафор
* int open(const char \*pathname, int flags, mode\_t mode) - открытие\создание файла
* int close(int fd) - закрыть файл
* void exit(int status) - завершения выполнения процесса и возвращение статуса
* int execv(const char \*filename, char \*const argv[]) - замена образа памяти процесса
* pid\_t getpid(void) — получение ID процесса
* ssize\_t read(int \_\_fd, void\* \_\_buf, size\_t \_\_nbytes) — чтение из fd в буфер
* ssize\_t write(int \_\_fd, const void\* \_\_buf, size\_t \_\_n) — запись байтов в буфер

В решении задачи взаимодействие между родительским и дочерним процессами осуществляется через общую память и синхронизацию с помощью семафоров. Родительский процесс открывает файл, создает объект общей памяти (shm\_open), задает его размер (ftruncate), мапирует в адресное пространство (mmap) и создает семафоры для управления доступом. Затем родительский процесс читает строки из файла, записывает их в общую память и сигнализирует дочернему процессу с помощью sem\_post, что данные готовы для обработки.

Дочерний процесс мапирует общую память, ожидает данных с помощью sem\_wait, выполняет их обработку (разбиение строки, вычисления с проверкой деления на ноль) и выводит результат с помощью системных вызовов. После обработки дочерний процесс сигнализирует родителю о завершении работы. Завершив цикл обработки, оба процесса очищают ресурсы, освобождая память (munmap, shm\_unlink) и закрывая семафоры (sem\_close, sem\_unlink), что обеспечивает корректное завершение программы

# Код программы

**parent.c**

*#include* <stdio.h>  
*#include* <stdlib.h>  
*#include* <unistd.h>  
*#include* <fcntl.h>  
*#include* <string.h>  
*#include* <sys/wait.h>  
*#include* <sys/mman.h>  
*#include* <semaphore.h>  
*#include* <sys/stat.h>  
*#include* <signal.h>  
  
*#define* BUFFER\_SIZE 512  
*#define* SHM\_NAME "/shared\_mem"  
*#define* SEM\_PARENT\_WRITE "/sem\_parent\_write"  
*#define* SEM\_CHILD\_READ "/sem\_child\_read"  
*#define* END\_MARKER "END"  
  
*// Очистка ресурсов*void *cleanup\_resources*(*sem\_t* \**sem\_parent\_write*, *sem\_t* \**sem\_child\_read*, char \**shared\_memory*, int *shm\_fd*) {  
 if (*sem\_parent\_write* != NULL) {  
 *sem\_close*(*sem\_parent\_write*);  
 *sem\_unlink*(SEM\_PARENT\_WRITE);  
 }  
 if (*sem\_child\_read* != NULL) {  
 *sem\_close*(*sem\_child\_read*);  
 *sem\_unlink*(SEM\_CHILD\_READ);  
 }  
 if (*shared\_memory* != NULL) {  
 *munmap*(*shared\_memory*, BUFFER\_SIZE);  
 }  
 if (*shm\_fd* != -1) {  
 *close*(*shm\_fd*);  
 *shm\_unlink*(SHM\_NAME);  
 }  
}  
  
*// Обработчик сигналов*void *signal\_handler*(int signum) {  
 *write*(STDERR\_FILENO, "Программа прервана.\n", 21);  
 *exit*(EXIT\_FAILURE);  
}  
  
void *write\_error*(const char \**message*) {  
 if (*message* != NULL) {  
 *write*(STDERR\_FILENO, *message*, *strlen*(*message*));  
 }  
}  
  
int *main*(int *argc*, char \**argv*[]) {  
 *signal*(SIGINT, *signal\_handler*);  
 *signal*(SIGTERM, *signal\_handler*);  
  
 if (*argc* != 2) {  
 *write\_error*("Ошибка: Укажите имя файла в качестве аргумента.\n");  
 return EXIT\_FAILURE;  
 }  
  
 *FILE* \*input\_file = *fopen*(*argv*[1], "r");  
 if (input\_file == NULL) {  
 *write\_error*("Ошибка: Не удалось открыть файл.\n");  
 return EXIT\_FAILURE;  
 }  
  
 int shm\_fd = *shm\_open*(SHM\_NAME, O\_CREAT | O\_RDWR, 0666);  
 if (shm\_fd == -1) {  
 *fclose*(input\_file);  
 *write\_error*("Ошибка: Не удалось создать объект общей памяти.\n");  
 return EXIT\_FAILURE;  
 }  
  
 if (*ftruncate*(shm\_fd, BUFFER\_SIZE) == -1) {  
 *fclose*(input\_file);  
 *cleanup\_resources*(NULL, NULL, NULL, shm\_fd);  
 *write\_error*("Ошибка: Не удалось установить размер общей памяти.\n");  
 return EXIT\_FAILURE;  
 }  
  
 char \*shared\_memory = *mmap*(0, BUFFER\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);  
 if (shared\_memory == MAP\_FAILED) {  
 *fclose*(input\_file);  
 *cleanup\_resources*(NULL, NULL, NULL, shm\_fd);  
 *write\_error*("Ошибка: Не удалось отобразить общую память.\n");  
 return EXIT\_FAILURE;  
 }  
  
 *sem\_t* \*sem\_parent\_write = *sem\_open*(SEM\_PARENT\_WRITE, O\_CREAT, 0666, 0);  
 *sem\_t* \*sem\_child\_read = *sem\_open*(SEM\_CHILD\_READ, O\_CREAT, 0666, 0);  
 if (sem\_parent\_write == SEM\_FAILED || sem\_child\_read == SEM\_FAILED) {  
 *fclose*(input\_file);  
 *cleanup\_resources*(sem\_parent\_write, sem\_child\_read, shared\_memory, shm\_fd);  
 *write\_error*("Ошибка: Не удалось создать семафоры.\n");  
 return EXIT\_FAILURE;  
 }  
  
 *pid\_t* pid = *fork*();  
 if (pid < 0) {  
 *fclose*(input\_file);  
 *cleanup\_resources*(sem\_parent\_write, sem\_child\_read, shared\_memory, shm\_fd);  
 *write\_error*("Ошибка: Не удалось создать дочерний процесс.\n");  
 return EXIT\_FAILURE;  
 }  
  
 if (pid == 0) {  
 *execl*("./child", "./child", NULL);  
 *write\_error*("Ошибка: Не удалось выполнить дочерний процесс.\n");  
 *exit*(EXIT\_FAILURE);  
 } else {  
 char buffer[BUFFER\_SIZE];  
 while (*fgets*(buffer, sizeof(buffer), input\_file) != NULL) {  
 *strncpy*(shared\_memory, buffer, BUFFER\_SIZE - 1);  
 shared\_memory[BUFFER\_SIZE - 1] = '\0';  
 *sem\_post*(sem\_parent\_write);  
 *sem\_wait*(sem\_child\_read);  
 }  
  
 *strncpy*(shared\_memory, END\_MARKER, BUFFER\_SIZE - 1);  
 shared\_memory[BUFFER\_SIZE - 1] = '\0';  
 *sem\_post*(sem\_parent\_write);  
 *wait*(NULL);  
 }  
  
 *cleanup\_resources*(sem\_parent\_write, sem\_child\_read, shared\_memory, shm\_fd);  
 *fclose*(input\_file);  
  
 return EXIT\_SUCCESS;  
}

**child.c**

*#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
#include <string.h>  
#include <semaphore.h>  
#include <sys/mman.h>  
#include <fcntl.h>  
#include <signal.h>  
  
#define BUFFER\_SIZE 512  
#define SHM\_NAME "/shared\_mem"  
#define SEM\_PARENT\_WRITE "/sem\_parent\_write"  
#define SEM\_CHILD\_READ "/sem\_child\_read"  
#define END\_MARKER "END"  
  
void cleanup\_resources(sem\_t \*sem\_parent\_write, sem\_t \*sem\_child\_read, char \*shared\_memory, int shm\_fd) {  
 if (sem\_parent\_write != NULL) {  
 sem\_close(sem\_parent\_write);  
 }  
 if (sem\_child\_read != NULL) {  
 sem\_close(sem\_child\_read);  
 }  
 if (shared\_memory != NULL) {  
 munmap(shared\_memory, BUFFER\_SIZE);  
 }  
 if (shm\_fd != -1) {  
 close(shm\_fd);  
 }  
}  
  
void write\_error(const char \*message) {  
 if (message != NULL) {  
 write(STDERR\_FILENO, message, strlen(message));  
 }  
}  
  
void signal\_handler(int signum) {  
 write(STDERR\_FILENO, "Дочерний процесс прерван.\n", 27);  
 exit(EXIT\_FAILURE);  
}  
  
int process\_command(const char \*command) {  
 char buffer[BUFFER\_SIZE];  
 strncpy(buffer, command, BUFFER\_SIZE - 1);  
 buffer[BUFFER\_SIZE - 1] = '\0';  
  
 char \*token = strtok(buffer, " ");  
 float result = 0.0;  
 int is\_first = 1;  
  
 while (token != NULL) {  
 float num = atof(token);  
 if (is\_first) {  
 result = num;  
 is\_first = 0;  
 } else {  
 if (num == 0) {  
 write\_error("Ошибка: Деление на ноль.\n");  
 return -1;  
 }  
 result /= num;  
 }  
 token = strtok(NULL, " ");  
 }  
  
 char output[BUFFER\_SIZE];  
 snprintf(output, BUFFER\_SIZE, "Результат деления: %.6f\n", result);  
 write(STDOUT\_FILENO, output, strlen(output));  
  
 return 0;  
}  
  
int main() {  
 signal(SIGINT, signal\_handler);  
 signal(SIGTERM, signal\_handler);  
  
 int shm\_fd = shm\_open(SHM\_NAME, O\_RDWR, 0666);  
 if (shm\_fd == -1) {  
 write\_error("Ошибка: Не удалось открыть общую память.\n");  
 return EXIT\_FAILURE;  
 }  
  
 char \*shared\_memory = mmap(0, BUFFER\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);  
 if (shared\_memory == MAP\_FAILED) {  
 cleanup\_resources(NULL, NULL, NULL, shm\_fd);  
 write\_error("Ошибка: Не удалось отобразить общую память.\n");  
 return EXIT\_FAILURE;  
 }  
  
 sem\_t \*sem\_parent\_write = sem\_open(SEM\_PARENT\_WRITE, 0);  
 sem\_t \*sem\_child\_read = sem\_open(SEM\_CHILD\_READ, 0);  
 if (sem\_parent\_write == SEM\_FAILED || sem\_child\_read == SEM\_FAILED) {  
 cleanup\_resources(sem\_parent\_write, sem\_child\_read, shared\_memory, shm\_fd);  
 write\_error("Ошибка: Не удалось открыть семафоры.\n");  
 return EXIT\_FAILURE;  
 }  
  
 while (1) {  
 sem\_wait(sem\_parent\_write);  
  
 if (strcmp(shared\_memory, END\_MARKER) == 0) {  
 break;  
 }  
  
 if (process\_command(shared\_memory) < 0) {  
 break;  
 }  
  
 sem\_post(sem\_child\_read);  
 }  
  
 cleanup\_resources(sem\_parent\_write, sem\_child\_read, shared\_memory, shm\_fd);  
  
 return EXIT\_SUCCESS;  
}*

# Протокол работы программы

qwental@DESKTOP-NKF1EUK:~/workspace/OS\_LABS/lab3/src$ cat test.txt

100 2 2 5

50 5 2

120 4 2 3 2

200 2 2 2 2 2

15 3 3

36 6 6

1000 5 5 5 2

8 2 2

90 10 3 3

60 2 3 5

45 5 5

25 5 1

300 3 5 2 2

100 2 4 5

40 4 2

64 8 2

128 8 2

500 10 2 5

100 2 2 2 2

qwental@DESKTOP-NKF1EUK:~/workspace/OS\_LABS/lab3/src$ cat build\_and\_check.sh

#!/bin/bash

# chmod +x build\_and\_check.sh

# ./build\_and\_check.sh

# ls /dev/shm

# Имена файлов и программ

PARENT\_SRC="parent.c"

CHILD\_SRC="child.c"

PARENT\_BIN="parent"

CHILD\_BIN="child"

SHM\_OBJECT="/shared\_mem"

SEM\_PARENT="/sem\_parent\_write"

SEM\_CHILD="/sem\_child\_read"

# Компиляция

echo "===Компиляция parent.c==="

gcc -o $PARENT\_BIN $PARENT\_SRC -lrt -lpthread

if [ $? -ne 0 ]; then

echo "Ошибка компиляции parent.c"

exit 1

fi

echo "===Компиляция child.c==="

gcc -o $CHILD\_BIN $CHILD\_SRC -lrt -lpthread

if [ $? -ne 0 ]; then

echo "Ошибка компиляции child.c"

exit 1

fi

# Проверка наличия ресурсов до запуска

echo "===Проверка наличия объектов в /dev/shm перед запуском==="

ls /dev/shm | grep -E "($SHM\_OBJECT|$SEM\_PARENT|$SEM\_CHILD)"

if [ $? -eq 0 ]; then

echo "Обнаружены остаточные ресурсы в /dev/shm перед запуском. Удалите их вручную!"

exit 1

fi

# Запуск программы

echo "===Запуск программы==="

./$PARENT\_BIN test.txt

# Проверка наличия ресурсов после завершения

echo "===Проверка наличия объектов в /dev/shm после завершения==="

ls /dev/shm | grep -E "($SHM\_OBJECT|$SEM\_PARENT|$SEM\_CHILD)"

if [ $? -eq 0 ]; then

echo "Ошибка: ресурсы не были освобождены!"

echo "Оставшиеся объекты:"

ls /dev/shm | grep -E "($SHM\_OBJECT|$SEM\_PARENT|$SEM\_CHILD)"

exit 1

else

echo "Все ресурсы были успешно освобождены."

fi

# Очистка

echo "===Очистка бинарных файлов==="

rm -f $PARENT\_BIN $CHILD\_BIN

echo "Скрипт завершён успешно."

qwental@DESKTOP-NKF1EUK:~/workspace/OS\_LABS/lab3/src$ chmod +x build\_and\_check.sh

**qwental@DESKTOP-NKF1EUK:~/workspace/OS\_LABS/lab3/src$ ./build\_and\_check.sh**

===Компиляция parent.c===

===Компиляция child.c===

===Проверка наличия объектов в /dev/shm перед запуском===

===Запуск программы===

Результат деления: 5.000000

Результат деления: 5.000000

Результат деления: 2.500000

Результат деления: 6.250000

Результат деления: 1.666667

Результат деления: 1.000000

Результат деления: 4.000000

Результат деления: 2.000000

Результат деления: 1.000000

Результат деления: 2.000000

Результат деления: 1.800000

Результат деления: 5.000000

Результат деления: 5.000000

Результат деления: 2.500000

Результат деления: 5.000000

Результат деления: 4.000000

Результат деления: 8.000000

Результат деления: 5.000000

Результат деления: 6.250000

===Проверка наличия объектов в /dev/shm после завершения===

Все ресурсы были успешно освобождены.

===Очистка бинарных файлов===

Скрипт завершён успешно.

qwental@DESKTOP-NKF1EUK:~/workspace/OS\_LABS/lab3/src$ ls /dev/shm

qwental@DESKTOP-NKF1EUK:~/workspace/OS\_LABS/lab3/src$

**Strace:**

qwental@DESKTOP-NKF1EUK:~/workspace/OS\_LABS/lab3/src$ strace ./parent test.txt

**execve("./parent", ["./parent", "test.txt"], 0x7ffe68e72a68 /\* 20 vars \*/) = 0**

brk(NULL) = 0x5575fa6eb000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffd79dc60c0) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f40c9f08000

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=31847, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 31847, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f40c9f00000

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) = 0x7f40c9cd7000

mprotect(0x7f40c9cff000, 2023424, PROT\_NONE) = 0

mmap(0x7f40c9cff000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f40c9cff000

mmap(0x7f40c9e94000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7f40c9e94000

mmap(0x7f40c9eed000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x7f40c9eed000

mmap(0x7f40c9ef3000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f40c9ef3000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f40c9cd4000

arch\_prctl(ARCH\_SET\_FS, 0x7f40c9cd4740) = 0

set\_tid\_address(0x7f40c9cd4a10) = 8272

set\_robust\_list(0x7f40c9cd4a20, 24) = 0

rseq(0x7f40c9cd50e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f40c9eed000, 16384, PROT\_READ) = 0

mprotect(0x5575f9124000, 4096, PROT\_READ) = 0

mprotect(0x7f40c9f42000, 8192, PROT\_READ) = 0

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

munmap(0x7f40c9f00000, 31847) = 0

rt\_sigaction(SIGINT, {sa\_handler=0x5575f91224a2, sa\_mask=[INT], sa\_flags=SA\_RESTORER|SA\_RESTART, sa\_restorer=0x7f40c9d19520}, {sa\_handler=SIG\_DFL, sa\_mask=[], sa\_flags=0}, 8) = 0

rt\_sigaction(SIGTERM, {sa\_handler=0x5575f91224a2, sa\_mask=[TERM], sa\_flags=SA\_RESTORER|SA\_RESTART, sa\_restorer=0x7f40c9d19520}, {sa\_handler=SIG\_DFL, sa\_mask=[], sa\_flags=0}, 8) = 0

getrandom("\xaf\x0a\x0d\xe7\x07\xc8\x58\x67", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x5575fa6eb000

brk(0x5575fa70c000) = 0x5575fa70c000

**openat(AT\_FDCWD, "test.txt", O\_RDONLY) = 3 //parent**

**openat(AT\_FDCWD, "/dev/shm/shared\_mem", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0666) = 4 //parent**

**ftruncate(4, 512) = 0 //parent**

**mmap(NULL, 512, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 4, 0) = 0x7f40c9f41000 // child**

**openat(AT\_FDCWD, "/dev/shm/sem.sem\_parent\_write", O\_RDWR|O\_NOFOLLOW) = -1 ENOENT (No such file or directory) //parent**

getrandom("\xae\xbb\xf6\xb2\xa6\x86\xc5\x50", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.QOrI4b", 0x7ffd79dc5bc0, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (No such file or directory)

**openat(AT\_FDCWD, "/dev/shm/sem.QOrI4b", O\_RDWR|O\_CREAT|O\_EXCL, 0666) = 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) = 0x7f40c9f07000

link("/dev/shm/sem.QOrI4b", "/dev/shm/sem.sem\_parent\_write") = 0

newfstatat(5, "", {st\_mode=S\_IFREG|0644, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0

unlink("/dev/shm/sem.QOrI4b") = 0

close(5) = 0

**openat(AT\_FDCWD, "/dev/shm/sem.sem\_child\_read", O\_RDWR|O\_NOFOLLOW) = -1 ENOENT (No such file or directory)**

getrandom("\x17\x58\xb8\x19\xcc\x26\x9f\xec", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.bljHZB", 0x7ffd79dc5bc0, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/dev/shm/sem.bljHZB", O\_RDWR|O\_CREAT|O\_EXCL, 0666) = 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) = 0x7f40c9f06000

link("/dev/shm/sem.bljHZB", "/dev/shm/sem.sem\_child\_read") = 0

newfstatat(5, "", {st\_mode=S\_IFREG|0644, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0

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

close(5) = 0

**clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f40c9cd4a10) = 8273**

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=193, ...}, AT\_EMPTY\_PATH) = 0

read(3, "100 2 2 5\r\n50 5 2\r\n120 4 2 3 2\r\n"..., 4096) = 193

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1Результат деления: 5.000000

) = 1

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1Результат деления: 5.000000

) = 1

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1Результат деления: 2.500000

) = 1

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1Результат деления: 6.250000

) = 1

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1) = 1

Результат деления: 1.666667

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1Результат деления: 1.000000

) = 1

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1Результат деления: 4.000000

) = 1

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1Результат деления: 2.000000

) = 1

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1Результат деления: 1.000000

) = 1

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1Результат деления: 2.000000

) = 1

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1Результат деления: 1.800000

) = 1

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1Результат деления: 5.000000

) = 1

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1) = 1

Результат деления: 5.000000

futex(0x7f40c9f06000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL, FUTEX\_BITSET\_MATCH\_ANY) = -1 EAGAIN (Resource temporarily unavailable)

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1Результат деления: 2.500000

) = 1

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1Результат деления: 5.000000

) = 1

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1Результат деления: 4.000000

) = 1

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1Результат деления: 8.000000

) = 1

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1Результат деления: 5.000000

) = 1

read(3, "", 4096) = 0

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1Результат деления: 6.250000

) = 1

**futex**(0x7f40c9f07000, FUTEX\_WAKE, 1) = 1

**wait4(-1, NULL, 0, NULL) = 8273**

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=8273, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

munmap(0x7f40c9f07000, 32) = 0

**unlink("/dev/shm/sem.sem\_parent\_write") = 0**

munmap(0x7f40c9f06000, 32) = 0

**unlink("/dev/shm/sem.sem\_child\_read") = 0**

**munmap(0x7f40c9f41000, 512) = 0 //child**

**close(4) = 0 //child**

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

**close(3) = 0 //parent**

**exit\_group(0) = ? //parent**

+++ exited with 0 +++

# Вывод

В ходе лабораторной работы я приобрел практические навыки в управлении процессами ОС и обеспечении обмена данных между процессами посредством shared memory и mmap. Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. Проблем в ходе выполнения не возникло.