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

openat(AT_FDCWD, "/dev/shm/shared_mem", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 4

ftruncate(4, 512) = 0 //parent

mmap(NULL, 512, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f40c9f41000

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

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\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) = 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

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.