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-210Б-23

Студент: Шведов Александр Иванович

Преподаватель: Бахарев В.Д. (ФИИТ)

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

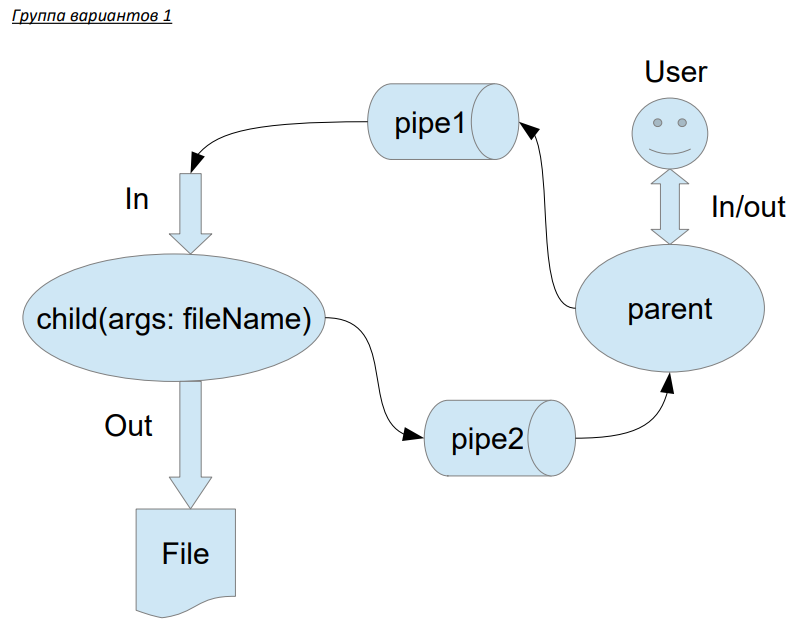
Дата: 12.12.24

Москва, 2024

**Постановка задачи**

**Постановка задачи**

**Вариант 3.**



Пользователь вводит команды вида: «число число число». Далее эти числа передаются от родительского процесса в дочерний. Дочерний процесс производит деление первого числа, на последующие, а результат выводит в файл. 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:

* mmap – отображение файла в память
* fork – создание дочернего процесса
* execv – замена исполняемого кода
* sem\_open – создание/подключение к семафору
* sem\_post – поднятие семафора
* sem\_wait – опускание семафора
* wait – ожидание завершения процесса
* kill – завершение процесса
* sem\_unlink - уничтожает именованный семафор
* shm\_open – открывает объект разделяемой памяти
* ftruncate - укорачивает файл до указанной длины
* sem\_close – закрывает именованый семафор
* munmap – снимает отражение файла

**Код программы**

**parent.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/wait.h>

#include <sys/stat.h>

#include <signal.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <semaphore.h>

#include <string.h>

#include <errno.h>

#define SHM\_NAME "/shared\_memory"

#define SEM\_NAME "/semaphore"

#define SHM\_SIZE 4096

int main()

{

    char filename[100];

    ssize\_t bytes\_read, bytes\_written;

    const char \*msg = "Введите имя файла: ";

    bytes\_written = write(STDOUT\_FILENO, msg, strlen(msg));

    if (bytes\_written == -1)

    {

        const char error\_msg[] = "error: failed to write to stdout\n";

        write(STDERR\_FILENO, error\_msg, sizeof(error\_msg) - 1);

        exit(EXIT\_FAILURE);

    }

    bytes\_read = read(STDIN\_FILENO, filename, sizeof(filename) - 1);

    if (bytes\_read == -1)

    {

        const char error\_msg[] = "error: failed to read from stdin\n";

        write(STDERR\_FILENO, error\_msg, sizeof(error\_msg) - 1);

        exit(EXIT\_FAILURE);

    }

    // Удаление символа новой строки, если он был введен

    if (bytes\_read > 0 && filename[bytes\_read - 1] == '\n')

    {

        filename[bytes\_read - 1] = '\0';

    }

    else

    {

        filename[bytes\_read] = '\0';

    }

    int shm\_fd;

    void \*shm\_ptr;

    pid\_t pid;

    char buffer[SHM\_SIZE]; // Use an array instead of malloc for simplicity

    sem\_t \*sem;

    // Create shared memory

    sem\_unlink(SEM\_NAME);

    shm\_unlink(SHM\_NAME);

    shm\_fd = shm\_open(SHM\_NAME, O\_RDWR | O\_CREAT | O\_EXCL, 0666);

    if (shm\_fd == -1)

    {

        perror("shm\_open");

        exit(1);

    }

    if (ftruncate(shm\_fd, SHM\_SIZE) == -1)

    {

        perror("ftruncate");

        shm\_unlink(SHM\_NAME);

        exit(1);

    }

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

    if (shm\_ptr == MAP\_FAILED)

    {

        perror("mmap");

        shm\_unlink(SHM\_NAME);

        close(shm\_fd);

        exit(1);

    }

    sem = sem\_open(SEM\_NAME, O\_CREAT | O\_EXCL, 0666, 0);

    if (sem == SEM\_FAILED)

    {

        perror("sem\_open");

        munmap(shm\_ptr, SHM\_SIZE);

        shm\_unlink(SHM\_NAME);

        close(shm\_fd);

        exit(1);

    }

    pid = fork();

    if (pid < 0)

    {

        perror("fork");

        exit(1);

    }

    else if (pid == 0)

    {

        execlp("./child", "child", filename, NULL);

        perror("execl");

        exit(1);

    }

    else

    {

        float num;

        int i = 0;

        while ((bytes\_read = read(STDIN\_FILENO, buffer, sizeof(buffer))) > 0)

        {

            memcpy(shm\_ptr, buffer, bytes\_read);

            if (sem\_post(sem) == -1)

            {

                perror("sem\_post");

                exit(1);

            }

            memset(buffer, 0, sizeof(buffer));

        }

        if (bytes\_read == -1)

        {

            perror("read");

            exit(1);

        }

        // Signal end of input to the child.

        memset(shm\_ptr, 0, SHM\_SIZE); // clear the shared memory before exiting

        if (sem\_post(sem) == -1)

        {

            perror("sem\_post");

            exit(1);

        }

        wait(NULL); // Wait for the child to finish

        munmap(shm\_ptr, SHM\_SIZE);

        close(shm\_fd);

        shm\_unlink(SHM\_NAME);

        sem\_close(sem);

        sem\_unlink(SEM\_NAME);

        printf("Parent finished\n");

    }

    return 0;

}

**child.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <signal.h>

#include <ctype.h>

#include <semaphore.h>

#include <string.h>

#include <errno.h>

#define SHM\_NAME "/shared\_memory"

#define SEM\_NAME "/semaphore"

#define SHM\_SIZE 4096

#define MAX\_NUMBERS 100

int convertStringToIntArray(const char \*str, int array[], int \*size)

{

    char \*token;

    char \*str\_copy = strdup(str);

    \*size = 0;

    token = strtok(str\_copy, " ");

    while (token != NULL && \*size < MAX\_NUMBERS)

    {

        array[(\*size)++] = atoi(token);

        token = strtok(NULL, " ");

    }

    free(str\_copy);

    return \*size;

}

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

{

    if (argc != 2)

    {

        const char msg[] = "Usage: <program> <filename>\n";

        write(STDERR\_FILENO, msg, sizeof(msg) - 1);

        exit(EXIT\_FAILURE);

    }

    char \*filename = argv[1];

    int file = open(filename, O\_WRONLY | O\_CREAT | O\_TRUNC, 0644);

    if (file == -1)

    {

        perror("open");

        exit(EXIT\_FAILURE);

    }

    int shm\_fd;

    void \*shm\_ptr;

    sem\_t \*sem;

    char buffer[SHM\_SIZE];

    shm\_fd = shm\_open(SHM\_NAME, O\_RDONLY, 0666);

    if (shm\_fd == -1)

    {

        perror("shm\_open");

        exit(1);

    }

    shm\_ptr = mmap(NULL, SHM\_SIZE, PROT\_READ, MAP\_SHARED, shm\_fd, 0);

    if (shm\_ptr == MAP\_FAILED)

    {

        perror("mmap");

        close(shm\_fd);

        exit(1);

    }

    sem = sem\_open(SEM\_NAME, 0);

    if (sem == SEM\_FAILED)

    {

        perror("sem\_open");

        munmap(shm\_ptr, SHM\_SIZE);

        close(shm\_fd);

        exit(1);

    }

    struct sigaction sa;

    sa.sa\_handler = SIG\_IGN;

    sigemptyset(&sa.sa\_mask);

    sa.sa\_flags = 0;

    if (sigaction(SIGTERM, &sa, NULL) == -1)

    {

        perror("sigaction");

        exit(1);

    }

    while (1)

    {

        if (sem\_wait(sem) == -1)

        {

            if (errno == EINTR)

                continue;

            perror("sem\_wait");

            munmap(shm\_ptr, SHM\_SIZE);

            close(shm\_fd);

            exit(1);

        }

        memcpy(buffer, shm\_ptr, SHM\_SIZE);

        if (strlen(buffer) == 0)

            break; // check for empty buffer.

        if (buffer[0] == '\n')

        {

            kill(getppid(), SIGTERM);

            break;

        }

        int array[MAX\_NUMBERS];

        int size = 0;

        convertStringToIntArray(buffer, array, &size);

        if (size < 2)

        {

            const char msg[] = "You have written few numbers\n";

            int32\_t written = write(STDOUT\_FILENO, msg, sizeof(msg));

            if (written != sizeof(msg))

            {

                const char msg[] = "error: failed to write to file\n";

                write(STDERR\_FILENO, msg, sizeof(msg));

                exit(EXIT\_FAILURE);

            }

        }

        int divisible = array[0];

        for (int i = 1; i != size; ++i)

        {

            if (array[i] == 0)

            {

                kill(getppid(), SIGTERM);

                const char ermsg[] = "Division by zero\n";

                write(STDERR\_FILENO, ermsg, sizeof(ermsg));

                exit(EXIT\_FAILURE);

            }

            char msg[32];

            int32\_t len = snprintf(msg, sizeof(msg) - 1, "%d : %d = %lf\n", divisible, array[i],

                                   ((float)divisible / array[i]));

            int32\_t written = write(file, msg, len);

            if (written != len)

            {

                const char msg[] = "error: failed to write to file\n";

                write(STDERR\_FILENO, msg, sizeof(msg));

                exit(EXIT\_FAILURE);

            }

        }

    }

    sem\_close(sem);

    munmap(shm\_ptr, SHM\_SIZE);

    close(shm\_fd);

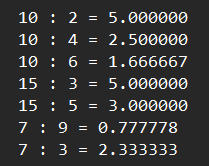
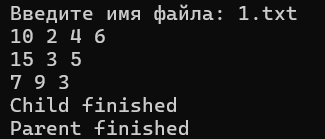
    printf("Child finished\n");

    return 0;

}

**Протокол работы программы**

**Тестирование:**

****

**Strace:**

execve("./parent", ["./parent"], 0x7ffcf762ac80 /\* 27 vars \*/) = 0

brk(NULL) = 0x555fed568000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffeeba09f10) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f54c15f6000

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=37379, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 37379, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f54c15ec000

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) = 0x7f54c13c3000

mprotect(0x7f54c13eb000, 2023424, PROT\_NONE) = 0

mmap(0x7f54c13eb000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f54c13eb000

mmap(0x7f54c1580000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7f54c1580000

mmap(0x7f54c15d9000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x7f54c15d9000

mmap(0x7f54c15df000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f54c15df000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f54c13c0000

arch\_prctl(ARCH\_SET\_FS, 0x7f54c13c0740) = 0

set\_tid\_address(0x7f54c13c0a10) = 1643

set\_robust\_list(0x7f54c13c0a20, 24) = 0

rseq(0x7f54c13c10e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f54c15d9000, 16384, PROT\_READ) = 0

mprotect(0x555fca25a000, 4096, PROT\_READ) = 0

mprotect(0x7f54c1630000, 8192, PROT\_READ) = 0

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

munmap(0x7f54c15ec000, 37379) = 0

write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\270\320\274\321\217 \321\204\320\260\320\271\320\273\320\260"..., 34Введите имя файла: ) = 34

read(0, 1.txt

"1.txt\n", 99) = 6

unlink("/dev/shm/sem.semaphore") = -1 ENOENT (No such file or directory)

unlink("/dev/shm/shared\_memory") = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/dev/shm/shared\_memory", O\_RDWR|O\_CREAT|O\_EXCL|O\_NOFOLLOW|O\_CLOEXEC, 0666) = 3

ftruncate(3, 4096) = 0

mmap(NULL, 4096, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7f54c162f000

getrandom("\x40\x3b\x32\x94\x84\x4b\x00\x06", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.2psWY6", 0x7ffeeba08ba0, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/dev/shm/sem.2psWY6", O\_RDWR|O\_CREAT|O\_EXCL, 0666) = 4

write(4, "\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, 4, 0) = 0x7f54c15f5000

link("/dev/shm/sem.2psWY6", "/dev/shm/sem.semaphore") = 0

newfstatat(4, "", {st\_mode=S\_IFREG|0644, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0

getrandom("\x1c\xe3\x53\xd4\xc3\xdb\x6b\xf0", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x555fed568000

brk(0x555fed589000) = 0x555fed589000

unlink("/dev/shm/sem.2psWY6") = 0

close(4) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f54c13c0a10) = 1671

read(0, 10 2 6

"10 2 6\n", 4096) = 7

futex(0x7f54c15f5000, FUTEX\_WAKE, 1) = 1

read(0, 15 3 9

"15 3 9\n", 4096) = 7

futex(0x7f54c15f5000, FUTEX\_WAKE, 1) = 1

read(0, 7 2 4

"7 2 4\n", 4096) = 6

futex(0x7f54c15f5000, FUTEX\_WAKE, 1) = 1

read(0, "", 4096) = 0

futex(0x7f54c15f5000, FUTEX\_WAKE, 1) = 1

Child finished

wait4(-1, NULL, 0, NULL) = 1671

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=1671, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

munmap(0x7f54c162f000, 4096) = 0

close(3) = 0

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

munmap(0x7f54c15f5000, 32) = 0

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

newfstatat(1, "", {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}, AT\_EMPTY\_PATH) = 0

write(1, "Parent finished\n", 16Parent finished

) = 16

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

В ходе лабораторной работе я приобрел базовые навыки по работе с разделяемой памятью в си. Я научился создавать объект разделяемой памяти, записывать в него данные и читать их из него. Также я узнал о работе с семафорами, научился использовать их для синхронизации при работе с разделяемой памятью. Помимо этого, я узнал о файловых системах и памяти в целом.