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

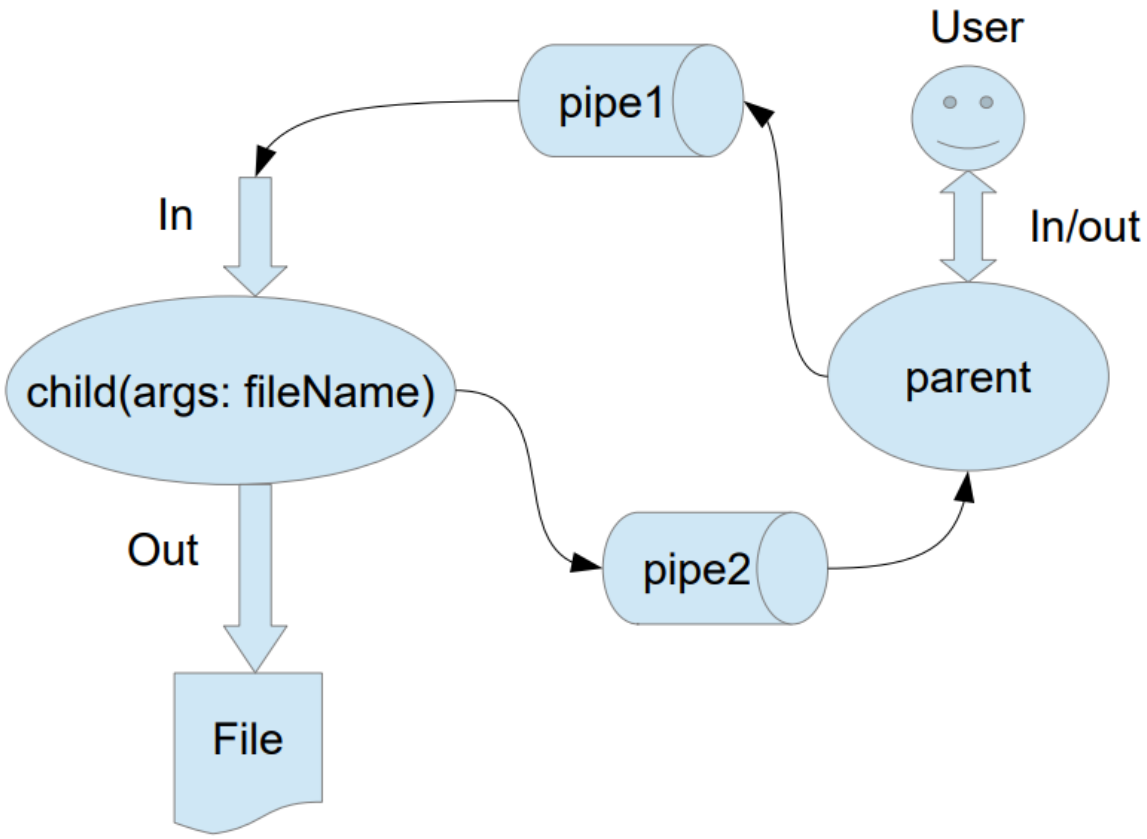
Москва, 2024

Постановка задачи

Постановка задачи

Вариант 3.

Группа вариантов 1



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

```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>
```

```c
#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:

```
Введите имя файла: 1.txt
10 2 4 6
15 3 5
7 9 3
Child finished
Parent finished
```

```
10 : 2 = 5.000000
10 : 4 = 2.500000
10 : 6 = 1.666667
15 : 3 = 5.000000
15 : 5 = 3.000000
7 : 9 = 0.777778
7 : 3 = 2.333333
```

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"..., 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\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, 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.