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: Фоменко А.С. Преподаватель: Бахарев В.Д. Оценка:

Дата: 26.12.24

Москва, 2024

# Постановка задачи

**Вариант 8.**

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы

программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через shared memory. Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы. Для синхронизации чтения и записи из shared memory используется семафор.

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 int. Количество чисел может быть произвольным.

# Общий метод

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.

# Код программы

**parent.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <semaphore.h>

#define MAX\_BUFFER 8192

int main() {

    int fd = shm\_open("/my\_shm", O\_CREAT | O\_RDWR, 0666);

    ftruncate(fd, MAX\_BUFFER);

    char \*shared\_memory = mmap(0, MAX\_BUFFER, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

    if (shared\_memory == MAP\_FAILED) {

        perror("mmap failed");

        exit(EXIT\_FAILURE);

    }

    sem\_t \*sem = sem\_open("/my\_semaphore1", O\_CREAT | O\_EXCL, 0644, 0);

    if (sem == SEM\_FAILED) {

        perror("sem\_open failed 2");

        exit(EXIT\_FAILURE);

    }

    pid\_t pid = fork();

    if (pid < 0) {

        perror("fork failed");

        return 1;

    }

    if (pid != 0) {

        sem\_wait(sem);

        printf("The result from the child process:\n%s", shared\_memory);

        munmap(shared\_memory, MAX\_BUFFER);

        shm\_unlink("/my\_shm");

        sem\_close(sem);

        sem\_destroy(sem);

        sem\_unlink("/my\_semaphore1");

        wait(NULL);

    } else {

        char filename[256];

        printf("Input file name: ");

        fgets(filename, sizeof(filename), stdin);

        filename[strcspn(filename, "\n")] = 0;

        char \*args[] = {"./child", filename, NULL};

        execv(args[0], args);

        perror("execv failed");

        exit(EXIT\_FAILURE);

    }

    return 0;

}

**client.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <errno.h>

#include <limits.h>

#include <sys/shm.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <semaphore.h>

#define MAX\_BUFFER 8192

int check\_overflow(const char \*str\_number) {

    if (!str\_number) return 1;

    char \*endptr;

    errno = 0;

    long result = strtol(str\_number, &endptr, 10);

    if (endptr == str\_number) return 1;

    if ((result == LONG\_MAX || result == LONG\_MIN) && errno == ERANGE) return 1;

    if (\*endptr != '\0' || result > INT\_MAX || result < INT\_MIN)  return 1;

    return 0;

}

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

    if (argc < 2) {

        fprintf(stderr, "Error: no file name\n");

        return 1;

    }

    int fd = shm\_open("/my\_shm", O\_CREAT | O\_RDWR, 0666);

    char \*shared\_memory = mmap(0, MAX\_BUFFER, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

    shared\_memory[0] = '\0';

    char buffer[128];

    char string[MAX\_BUFFER];

    string[0] = '\0';

    sem\_t \*sem = sem\_open("/my\_semaphore1", 0);

    if (sem == SEM\_FAILED) {

        perror("sem\_open failed 3");

        exit(EXIT\_FAILURE);

    }

    const char \*filename = argv[1];

    FILE \*file = fopen(filename, "r");

    if (file == NULL) {

        perror("Error: file can't be open");

        return 1;

    }

    char line[MAX\_BUFFER];

    while (fgets(line, sizeof(line), file)) {

        int num1, num2;

        char \*token = strtok(line, " ");

        if (token != NULL) {

            if (check\_overflow(token)) {

                    snprintf(buffer, 128, "Error: int overflow in token = %s\n", token);

                    strcat(string, buffer);

            }

            else {

                num1 = atoi(token);

                while ((token = strtok(NULL, " ")) != NULL) {

                    if (check\_overflow(token)) {

                        snprintf(buffer, 128, "Error: int overflow in token = %s\n", token);

                        strcat(string, buffer);

                    }

                    else {

                        num2 = atoi(token);

                        if (num2 == 0) {

                            snprintf(buffer, 128, "Error: division by zero, exit();\n");

                            strcat(string, buffer);

                            strcpy(shared\_memory, string);

                            sem\_post(sem);

                            munmap(shared\_memory, MAX\_BUFFER);

                            shm\_unlink("/my\_semaphore1");

                            sem\_close(sem);

                            exit(EXIT\_FAILURE);

                        }

                        int result = num1 / num2;

                        snprintf(buffer, 128, "%d / %d = %d\n", num1, num2, result);

                        strcat(string, buffer);

                    }

                }

            }

        }

    }

    strcpy(shared\_memory, string);

    sem\_post(sem);

    munmap(shared\_memory, MAX\_BUFFER);

    shm\_unlink("/my\_semaphore1");

    sem\_close(sem);

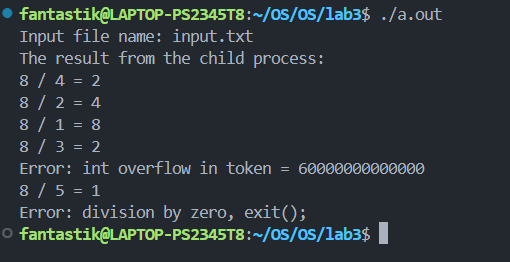
    fclose(file);

    return 0;

}

# Протокол работы программы

**Тестирование:**



**Strace:**

fantastik@LAPTOP-PS2345T8:~/OS/OS/lab3$ strace ./a.out

execve("./a.out", ["./a.out"], 0x7ffcf80861e0 /\* 37 vars \*/) = 0

brk(NULL) = 0x56119ea16000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fd841aea000

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

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=19791, ...}) = 0

mmap(NULL, 19791, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fd841ae5000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/librt.so.1", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=14624, ...}) = 0

mmap(NULL, 16400, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fd841ae0000

mmap(0x7fd841ae1000, 4096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1000) = 0x7fd841ae1000

mmap(0x7fd841ae2000, 4096, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7fd841ae2000

mmap(0x7fd841ae3000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7fd841ae3000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libpthread.so.0", O\_RDONLY|O\_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0\0"..., 832) = 832

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=14408, ...}) = 0

mmap(NULL, 16400, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fd841adb000

mmap(0x7fd841adc000, 4096, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1000) = 0x7fd841adc000

mmap(0x7fd841add000, 4096, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7fd841add000

mmap(0x7fd841ade000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x2000) = 0x7fd841ade000

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\0\220\243\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

fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2125328, ...}) = 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, 2170256, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7fd8418c9000

mmap(0x7fd8418f1000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7fd8418f1000

mmap(0x7fd841a79000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) = 0x7fd841a79000

mmap(0x7fd841ac8000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x7fd841ac8000

mmap(0x7fd841ace000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fd841ace000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fd8418c6000

arch\_prctl(ARCH\_SET\_FS, 0x7fd8418c6740) = 0

set\_tid\_address(0x7fd8418c6a10) = 401726

set\_robust\_list(0x7fd8418c6a20, 24) = 0

rseq(0x7fd8418c7060, 0x20, 0, 0x53053053) = 0

mprotect(0x7fd841ac8000, 16384, PROT\_READ) = 0

mprotect(0x7fd841ade000, 4096, PROT\_READ) = 0

mprotect(0x7fd841ae3000, 4096, PROT\_READ) = 0

mprotect(0x56119e201000, 4096, PROT\_READ) = 0

mprotect(0x7fd841b22000, 8192, PROT\_READ) = 0

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

munmap(0x7fd841ae5000, 19791) = 0

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

ftruncate(3, 8192) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7fd841ae8000

getrandom("\x4b\xe7\xb5\x0a\x75\x94\x23\xf5", 8, GRND\_NONBLOCK) = 8

getrandom("\x9e\xef\xaf\x14\x14\x9a\x62\x2a", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.817miW", 0x7ffd9c794e70, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/dev/shm/sem.817miW", O\_RDWR|O\_CREAT|O\_EXCL|O\_NOFOLLOW|O\_CLOEXEC, 0644) = 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) = 0x7fd841ae7000

link("/dev/shm/sem.817miW", "/dev/shm/sem.my\_semaphore1") = 0

fstat(4, {st\_mode=S\_IFREG|0644, st\_size=32, ...}) = 0

getrandom("\xcb\xe1\x4f\x67\x66\x3f\x63\xfe", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x56119ea16000

brk(0x56119ea37000) = 0x56119ea37000

unlink("/dev/shm/sem.817miW") = 0

close(4) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7fd8418c6a10) = 401727

Input file name: futex(0x7fd841ae7000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL, FUTEX\_BITSET\_MATCH\_ANYinput.txt

) = 0

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=401727, si\_uid=1000, si\_status=1, si\_utime=0, si\_stime=0} ---

fstat(1, {st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0x7), ...}) = 0

write(1, "The result from the child proces"..., 35The result from the child process:

) = 35

write(1, "8 / 4 = 2\n", 108 / 4 = 2

) = 10

write(1, "8 / 2 = 4\n", 108 / 2 = 4

) = 10

write(1, "8 / 1 = 8\n", 108 / 1 = 8

) = 10

write(1, "8 / 3 = 2\n", 108 / 3 = 2

) = 10

write(1, "Error: int overflow in token = 6"..., 46Error: int overflow in token = 60000000000000

) = 46

write(1, "8 / 5 = 1\nError: division by zer"..., 438 / 5 = 1

Error: division by zero, exit();

) = 43

munmap(0x7fd841ae8000, 8192) = 0

unlink("/dev/shm/my\_shm") = 0

munmap(0x7fd841ae7000, 32) = 0

unlink("/dev/shm/sem.my\_semaphore1") = 0

wait4(-1, NULL, 0, NULL) = 401727

exit\_group(0) = ?

+++ exited with 0 +++

# 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.