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: Тремель Д.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 23.12.24

Москва, 2024

Постановка задачи

Вариант 6.

Постановка задачи Родительский процесс создает дочерний процесс. Первой строчкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия файла с таким именем на чтение. Стандартный поток ввода дочернего процесса переопределяется открытым файлом. Дочерний процесс читает команды из стандартного потока ввода. Стандартный поток вывода дочернего процесса перенаправляется в pipe1. Родительский процесс читает из pipe1 и прочитанное выводит в свой стандартный поток вывода. Родительский и дочерний процесс должны быть представлены разными программами. В файле записаны команды вида: «число число число». Дочерний процесс считает их сумму и выводит результат в стандартный поток вывода. Числа имеют тип int. Количество чисел может быть произвольным.

Общий метод и алгоритм решения

Использованные системные вызовы:

● pid_t fork(void); — создаёт новый дочерний процесс.

● int pipe(int *fd); — создаёт канал, сохраняя дескрипторы для чтения и записи в fd[0] и fd[1].

● pid_t getpid(void); — возвращает идентификатор текущего процесса.

● int open(const char *file, int oflag, ...); — открывает файл для чтения, записи или обоих операций.

● ssize_t write(int fd, const void *buf, size_t n); — записывает n байт из буфера buf в файл с дескриптором fd. Возвращает количество записанных байт или -1 в случае ошибки.

● void exit(int status); — завершает выполнение программы, закрывая все потоки и удаляя временные файлы. Управление передаётся операционной системе или другой программе.

● int close(int fd); — завершает работу с файловым дескриптором fd, закрывая связанный с ним файл.

● int dup2(int fd, int fd2); — копирует дескриптор fd в fd2, предварительно закрыв fd2, если это необходимо.

● int execv(const char *path, char *const *argv); — заменяет текущий процесс на новый, используя образ, указанный в path.

● ssize_t read(int fd, void *buf, size_t nbytes); — считывает nbytes из файла с дескриптором fd в буфер buf.

● pid_t wait(int *stat_loc); — ожидает изменения состояния дочернего процесса, возвращая информацию о завершении или сигнале прерывания.

● int shm_open(const char *name, int oflag, mode_t mode); — создаёт или открывает объект разделяемой памяти.

● int shm_unlink(const char *name); — удаляет имя объекта разделяемой памяти и очищает его при завершении всех связанных процессов.

● void *mmap(void *start, size_t length, int prot, int flags, int fd, off_t offset); — отображает в память участок файла или объекта длиной length, начиная с указанного смещения offset.

● int ftruncate(int fd, off_t length); — устанавливает длину файла, связанного с дескриптором fd, равной length байт.

● int sem_wait(sem_t *sem); — уменьшает значение семафора на 1. Если значение равно нулю, процесс блокируется до возможности выполнения операции.

● int sem_post(sem_t *sem); — увеличивает значение семафора на 1.

● int sem_destroy(sem_t *sem); — удаляет безымянный семафор, расположенный по адресу sem.

Для выполнения данной лабораторной работы я изучил указанные выше системные вызовы, а также пример выполнения подобного задания.

Программа parent.c принимает путь к файлу, содержащему числа типа int, в качестве аргумента командной строки. Файл открывается с использованием fopen() в режиме чтения. Затем создаётся область разделяемой памяти с помощью вызовов shm_open() и ftruncate(). Для синхронизации используются два семафора: sem_write (управляет процессом записи данных) и sem_read (управляет процессом чтения).

С использованием вызова fork() создаётся дочерний процесс. Если это родительский процесс, он построчно читает данные из файла с помощью fgets(), копирует их в разделяемую память и сигнализирует дочернему процессу с помощью семафора sem_read. После завершения передачи данных родительский процесс записывает в разделяемую память пустую строку как сигнал окончания ввода и ожидает завершения работы дочернего процесса через wait().

Дочерний процесс подключается к разделяемой памяти и семафорам. Он ожидает данные, используя семафор sem_read, затем обрабатывает полученные строки, суммируя числа, и выводит результат. Если в строках обнаруживаются некорректные данные, выводится сообщение об ошибке, но выполнение программы продолжается. После получения пустой строки как сигнала об окончании ввода дочерний процесс завершает работу, закрывая разделяемую память и семафоры.

Все системные вызовы проверяются на наличие ошибок. Ресурсы корректно освобождаются: разделяемая память удаляется с использованием shm_unlink(), а семафоры — с помощью sem_unlink() после завершения работы программы.

Код программы

parent.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <semaphore.h>
#include <unistd.h>
#include <sys/wait.h>

#define SHM_NAME "/shared_memory"
#define SEM_WRITE "/sem_write"
#define SEM_READ "/sem_read"
#define BUF_SIZE 256

void write_error(const char *msg) {
    write(STDERR_FILENO, msg, strlen(msg));
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        const char msg[] = "Usage: ./parent <filename>\n";
```

```
        write(STDERR_FILENO, msg, sizeof(msg) - 1);
        exit(EXIT_FAILURE);
    }

    FILE *file = fopen(argv[1], "r");
    if (!file) {
        write_error("Error opening file\n");
        exit(EXIT_FAILURE);
    }

    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        write_error("Error creating shared memory\n");
        fclose(file);
        exit(EXIT_FAILURE);
    }

    if (ftruncate(shm_fd, BUF_SIZE) == -1) {
        write_error("Error setting shared memory size\n");
        fclose(file);
        shm_unlink(SHM_NAME);
        exit(EXIT_FAILURE);
    }

    char *shm_ptr = mmap(0, BUF_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd,
0);
    if (shm_ptr == MAP_FAILED) {
        write_error("Error mapping shared memory\n");
        fclose(file);
        shm_unlink(SHM_NAME);
        exit(EXIT_FAILURE);
    }

    sem_t *sem_write = sem_open(SEM_WRITE, O_CREAT, 0666, 1);
    sem_t *sem_read = sem_open(SEM_READ, O_CREAT, 0666, 0);
    if (sem_write == SEM_FAILED || sem_read == SEM_FAILED) {
        write_error("Error creating semaphores\n");
        fclose(file);
        shm_unlink(SHM_NAME);
        exit(EXIT_FAILURE);
    }

    pid_t pid = fork();
    if (pid == 0) {
        execl("./child", "./child", NULL);
        write_error("Error executing child process\n");
        exit(EXIT_FAILURE);
    } else if (pid > 0) {
        char buffer[BUF_SIZE];
        while (fgets(buffer, sizeof(buffer), file) != NULL) {
```

```
            sem_wait(sem_write);
            strncpy(shm_ptr, buffer, BUF_SIZE);
            sem_post(sem_read);
        }

        sem_wait(sem_write);
        shm_ptr[0] = '\0';
        sem_post(sem_read);

        wait(NULL);

        fclose(file);
        munmap(shm_ptr, BUF_SIZE);
        shm_unlink(SHM_NAME);
        sem_close(sem_write);
        sem_close(sem_read);
        sem_unlink(SEM_WRITE);
        sem_unlink(SEM_READ);
    }

    return 0;
}
```

child.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <semaphore.h>
#include <unistd.h>

#define SHM_NAME "/shared_memory"
#define SEM_WRITE "/sem_write"
#define SEM_READ "/sem_read"
#define BUF_SIZE 256

void write_error(const char *msg) {
    write(STDERR_FILENO, msg, strlen(msg));
}

void write_message(const char *msg, int num) {
    char buffer[BUF_SIZE];
    int len = snprintf(buffer, BUF_SIZE, msg, num);
    write(STDOUT_FILENO, buffer, len);
}
```

```
int str_to_int(const char *str, int *num) {
    char *endptr;
    long val = strtol(str, &endptr, 10);

    if (str == endptr || *endptr != '\0') {
        return 0;
    }

    *num = (int)val;
    return 1;
}

void remove_carriage_return(char *str) {
    char *pos;
    if ((pos = strchr(str, '\n')) != NULL) {
        *pos = '\0';
    }
    if ((pos = strchr(str, '\r')) != NULL) {
        *pos = '\0';
    }
}

int main() {
    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
    if (shm_fd == -1) {
        write_error("Error opening shared memory\n");
        exit(EXIT_FAILURE);
    }

    char *shm_ptr = mmap(0, BUF_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd,
0);
    if (shm_ptr == MAP_FAILED) {
        write_error("Error mapping shared memory\n");
        exit(EXIT_FAILURE);
    }

    sem_t *sem_write = sem_open(SEM_WRITE, 0);
    sem_t *sem_read = sem_open(SEM_READ, 0);
    if (sem_write == SEM_FAILED || sem_read == SEM_FAILED) {
        write_error("Error opening semaphores\n");
        exit(EXIT_FAILURE);
    }

    int line = 1;
    while (1) {
        sem_wait(sem_read);

        if (shm_ptr[0] == '\0') {
            break;
```

```
        }

        remove_carriage_return(shm_ptr);

        char *token = strtok(shm_ptr, " ");
        int line_sum = 0;
        int valid_line = 1;

        while (token != NULL) {
            int num;
            if (!str_to_int(token, &num)) {
                char buffer[BUF_SIZE];
                int len = snprintf(buffer, BUF_SIZE, "Invalid number in line %d:
%s\n", line, token);
                write(STDERR_FILENO, buffer, len);
                valid_line = 0;
                break;
            }
            line_sum += num;
            token = strtok(NULL, " ");
        }

        if (valid_line) {
            write_message("Sum in line %d = %d\n", line);
        }

        line++;
        sem_post(sem_write);
    }

    munmap(shm_ptr, BUF_SIZE);
    sem_close(sem_write);
    sem_close(sem_read);

    return 0;
}
```

Протокол работы программы

```
u@DESKTOP-3U3OER0:/mnt/c/Users/u/CLionProjects/OS/lab_3$ gcc -o parent
parent.c -lrt -pthread

u@DESKTOP-3U3OER0:/mnt/c/Users/u/CLionProjects/OS/lab_3$ gcc -o child
child.c -lrt -pthread

u@DESKTOP-3U3OER0:/mnt/c/Users/u/CLionProjects/OS/lab_3$ ./parent
commands.txt

Sum in line 1 = 6

Sum in line 2 = 15

Sum in line 3 = 24

Sum in line 4 = 7

Sum in line 5 = -1

Sum in line 6 = 0

u@DESKTOP-3U3OER0:/mnt/c/Users/u/CLionProjects/OS/lab_3$ strace ./parent
commands.txt

execve("./parent", ["./parent", "commands.txt"], 0x7ffce16d4408 /* 26
vars */) = 0

brk(NULL)                               = 0x55c41b831000

arch_prctl(0x3001 /* ARCH_??? */, 0x7fff66c922b0) = -1 EINVAL (Invalid
argument)

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=61562, ...}) = 0

mmap(NULL, 61562, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f3905220000

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"..., 832) = 832

fstat(3, {st_mode=S_IFREG|0644, st_size=35960, ...}) = 0

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7f390521e000

mmap(NULL, 39904, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f3905214000
```

```
mmap(0x7f3905216000, 16384, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x200

0) = 0x7f3905216000

mmap(0x7f390521a000, 8192, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x6000) = 0x7f39

0521a000

mmap(0x7f390521c000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x700

0) = 0x7f390521c000

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\220q\0\0\0\0\0\0"...,
832) = 832

pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\232e\273F\236E\241\306\373\317\372\345\2
70*/\327"..

., 68, 824) = 68

fstat(3, {st_mode=S_IFREG|0755, st_size=157224, ...}) = 0

pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\232e\273F\236E\241\306\373\317\372\345\2
70*/\327"..

., 68, 824) = 68

mmap(NULL, 140408, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f39051f1000

mmap(0x7f39051f7000, 69632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x600

0) = 0x7f39051f7000

mmap(0x7f3905208000, 24576, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x17000) = 0x7f

3905208000

mmap(0x7f390520e000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1c0

00) = 0x7f390520e000

mmap(0x7f3905210000, 13432, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0)

= 0x7f3905210000
```

```
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\300A\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\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32,
848) = 32

pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\7\2C\n\357_\243\335\2449\206V>\237\374\3
04"..., 68,

 880) = 68

fstat(3, {st_mode=S_IFREG|0755, st_size=2029592, ...}) = 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

pread64(3,
"\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32,
848) = 32

pread64(3,
"\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\7\2C\n\357_\243\335\2449\206V>\237\374\3
04"..., 68,

 880) = 68

mmap(NULL, 2037344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f3904fff000

mmap(0x7f3905021000, 1540096, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2

2000) = 0x7f3905021000

mmap(0x7f3905199000, 319488, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19a000) = 0x

7f3905199000

mmap(0x7f39051e7000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e

7000) = 0x7f39051e7000

mmap(0x7f39051ed000, 13920, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0)
```

```
= 0x7f39051ed000

close(3)                                = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7f3904ffc000

arch_prctl(ARCH_SET_FS, 0x7f3904ffc740) = 0

mprotect(0x7f39051e7000, 16384, PROT_READ) = 0

mprotect(0x7f390520e000, 4096, PROT_READ) = 0

mprotect(0x7f390521c000, 4096, PROT_READ) = 0

mprotect(0x55c3dba96000, 4096, PROT_READ) = 0

mprotect(0x7f390525d000, 4096, PROT_READ) = 0

munmap(0x7f3905220000, 61562)           = 0

set_tid_address(0x7f3904ffca10)         = 91

set_robust_list(0x7f3904ffca20, 24)     = 0

rt_sigaction(SIGRTMIN, {sa_handler=0x7f39051f7bf0, sa_mask=[],
sa_flags=SA_RESTORER|SA_SIGINFO

, sa_restorer=0x7f3905205420}, NULL, 8) = 0

rt_sigaction(SIGRT_1, {sa_handler=0x7f39051f7c90, sa_mask=[],
sa_flags=SA_RESTORER|SA_RESTART|

SA_SIGINFO, sa_restorer=0x7f3905205420}, NULL, 8) = 0

rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024,
rlim_max=RLIM64_INFINITY}) = 0

brk(NULL)                               = 0x55c41b831000

brk(0x55c41b852000)                     = 0x55c41b852000

openat(AT_FDCWD, "commands.txt", O_RDONLY) = 3

statfs("/dev/shm/", {f_type=TMPFS_MAGIC, f_bsize=4096, f_blocks=996906,
f_bfree=996906, f_bava

il=996906, f_files=996906, f_ffree=996905, f_fsid={val=[2874711782,
382598001]}, f_namelen=255

, f_frsize=4096, f_flags=ST_VALID|ST_NOSUID|ST_NODEV|ST_NOATIME}) = 0

futex(0x7f3905213390, FUTEX_WAKE_PRIVATE, 2147483647) = 0

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

ftruncate(4, 256)                       = 0

mmap(NULL, 256, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f390525c000
```

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

ectory)

getpid()                                = 91

lstat("/dev/shm/h0lSe6", 0x7fff66c91f10) = -1 ENOENT (No such file or
directory)

openat(AT_FDCWD, "/dev/shm/h0lSe6", O_RDWR|O_CREAT|O_EXCL, 0666) = 5

write(5,
"\1\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) = 0x7f390522f000

link("/dev/shm/h0lSe6", "/dev/shm/sem.sem_write") = 0

fstat(5, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0

unlink("/dev/shm/h0lSe6")               = 0

close(5)                                = 0

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

ctory)

getpid()                                = 91

lstat("/dev/shm/3GbFp7", 0x7fff66c91f10) = -1 ENOENT (No such file or
directory)

openat(AT_FDCWD, "/dev/shm/3GbFp7", 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) = 0x7f390522e000

link("/dev/shm/3GbFp7", "/dev/shm/sem.sem_read") = 0

fstat(5, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0

unlink("/dev/shm/3GbFp7")               = 0

close(5)                                = 0

clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x

7f3904ffca10) = 92

fstat(3, {st_mode=S_IFREG|0777, st_size=42, ...}) = 0

read(3, "1 2 3\n4 5 6\n7 8 9\n1 1 1 1 1 1 1\n"..., 512) = 42
```

```
futex(0x7f390522f000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL,
FUTEX_BITSET_MATCH_ANYS

um in line 1 = 6

) = 0

futex(0x7f390522e000, FUTEX_WAKE, 1Sum in line 2 = 15

)    = 1

futex(0x7f390522e000, FUTEX_WAKE, 1Sum in line 3 = 24

)    = 1

futex(0x7f390522e000, FUTEX_WAKE, 1Sum in line 4 = 7

)    = 1

futex(0x7f390522e000, FUTEX_WAKE, 1Sum in line 5 = -1

)    = 1

read(3, "", 512)                        = 0

futex(0x7f390522e000, FUTEX_WAKE, 1Sum in line 6 = 0

)    = 1

futex(0x7f390522e000, FUTEX_WAKE, 1)    = 1

wait4(-1, NULL, 0, NULL)                = 92

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=92,
si_uid=1000, si_status=0, si_uti

me=0, si_stime=1} ---

close(3)                                = 0

munmap(0x7f390525c000, 256)             = 0

unlink("/dev/shm/shared_memory")        = 0

munmap(0x7f390522f000, 32)              = 0

munmap(0x7f390522e000, 32)              = 0

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

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

exit_group(0)                           = ?

+++ exited with 0 +++
```

Вывод

В процессе выполнения данной лабораторной работы я изучил новые системные вызовы на языке Си, которые позволяют эффективно работать с разделяемой памятью и семафорами. Освоил передачу данных между процессами через shared memory и управление доступом с использованием семафоров.