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: Акимов К.К. Преподаватель: Бахарев В.Д. Оценка:

Дата: 19.12.24

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); – создает дочерний процесс.
* int pipe(int \*fd); – создает канал и помещает дескрипторы файла для чтения и записи в fd[0] и fd[1].
* 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 dup2(int fd, int fd2); – копирует FD в FD2, закрыв FD2 если это требуется.
* 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.
* 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 ftruncate(int fd, off\_t length); – устанавливают длину файла с файловым дескриптором fd в length байт.
* 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\_destroy( sem\_t \*sem ); - уничтожает 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 с числами типа int в качестве аргумента командной строки. Открывается файл с помощью fopen() в режиме чтения, создается область разделяемой памяти с помощью shm\_open() и ftruncate(), а также два семафора для синхронизации: sem\_write (управляет записью данных) и sem\_read (управляет чтением).

С помощью fork() создается дочерний процесс. Если это родитель, он читает строки из файла с помощью fgets(), копирует их в разделяемую память и сигнализирует дочернему процессу через sem\_read. После завершения ввода родитель записывает пустую строку в память для сигнала об окончании и ждет завершения дочернего процесса через wait().

Дочерний процесс открывает разделяемую память и подключается к семафорам. Он ждет данных через sem\_read, обрабатывает строки, суммируя числа, и выводит результат. Невалидные строки или числа вызывают сообщение об ошибке, но программа продолжает свое выполнение. После получения пустой строки дочерний процесс завершает работу, закрывая разделяемую память и семафоры.

Все системные вызовы проверяются на ошибки, ресурсы корректно освобождаются с помощью shm\_unlink() и sem\_unlink() после завершения программы.

# Код программы

## Parent.c

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

#define SHM\_NAME "/shared\_memory" #define SEM\_WRITE "/sem\_write" #define SEM\_READ "/sem\_read" #define BUF\_SIZE 1024

int main(int argc, char \*argv[]) { if (argc != 2) {

write(STDERR\_FILENO, "Error, there should be 2 arguments here\n", 40); exit(EXIT\_FAILURE);

}

FILE \*file = fopen(argv[1], "r"); if (!file) {

write(STDERR\_FILENO, "Error with open file\n", 21); exit(EXIT\_FAILURE);

}

int shm\_fd = shm\_open(SHM\_NAME, O\_CREAT | O\_RDWR, 0666); if (shm\_fd == -1) {

write(STDERR\_FILENO, "Error creating shared memory\n", 29); fclose(file);

exit(EXIT\_FAILURE);

}

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

write(STDERR\_FILENO, "Error setting shared memory size\n", 33); 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(STDERR\_FILENO, "Error mapping shared memory\n", 28); 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(STDERR\_FILENO, "Error creating semaphores\n", 26); fclose(file);

shm\_unlink(SHM\_NAME); exit(EXIT\_FAILURE);

}

pid\_t pid = fork(); if (pid == 0) {

// Child

execl("./child", "./child", NULL);

write(STDERR\_FILENO, "Error executing child process\n", 30); exit(EXIT\_FAILURE);

} else if (pid > 0) {

// Parent

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);

}

// End of input sem\_wait(sem\_write); shm\_ptr[0] = '\0';

sem\_post(sem\_read);

wait(NULL); fclose(file);

} else {

write(STDERR\_FILENO, "Error with fork\n", 16); fclose(file);

shm\_unlink(SHM\_NAME); sem\_unlink(SEM\_WRITE); sem\_unlink(SEM\_READ); exit(EXIT\_FAILURE);

}

munmap(shm\_ptr, BUF\_SIZE); shm\_unlink(SHM\_NAME); 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> #include <limits.h>

#define SHM\_NAME "/shared\_memory" #define SEM\_WRITE "/sem\_write" #define SEM\_READ "/sem\_read" #define BUF\_SIZE 1024

enum Errors {

*OK*, *ERROR*

};

enum Errors Str\_to\_int(char \*str, int \*answer, int line) { char \*endptr;

long number = strtol(str, &endptr, 10);

if (number > INT\_MAX || number < INT\_MIN || \*endptr != '\0') { fprintf(stderr, "Error, incorrect value in %d line\n", line); return *ERROR*;

}

\*answer = (int) number; return *OK*;

}

void remove\_carriage\_return(char \*str) { char \*ptr = strchr(str, '\r');

if (ptr) \*ptr = '\0';

}

int main() {

int shm\_fd = shm\_open(SHM\_NAME, O\_RDWR, 0666); if (shm\_fd == -1) {

perror("Error opening shared memory"); exit(EXIT\_FAILURE);

}

char \*shm\_ptr = mmap(0, BUF\_SIZE, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0); if (shm\_ptr == MAP\_FAILED) {

perror("Error mapping shared memory"); 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) { perror("Error opening semaphores"); exit(EXIT\_FAILURE);

}

int line = 1; while (1) {

sem\_wait(sem\_read);

if (shm\_ptr[0] == '\0') { break;

}

remove\_carriage\_return(shm\_ptr); // Удаляем '\r' перед обработкой

char \*token = strtok(shm\_ptr, " "); int line\_sum = 0;

int valid\_line = 1;

while (token != NULL) { int num;

if (Str\_to\_int(token, &num, line) != *OK*) { valid\_line = 0;

break;

}

line\_sum += num;

token = strtok(NULL, " ");

}

if (valid\_line) {

printf("Sum in %d line = %d\n", line, line\_sum);

}

line++; sem\_post(sem\_write);

}

munmap(shm\_ptr, BUF\_SIZE); sem\_close(sem\_write); sem\_close(sem\_read); return 0;

}

# Протокол работы программы

kirill@DESKTOP-O0B2VHP:/mnt/c/Users/User/OSI/lab3/src$ gcc -o parent parent.c kirill@DESKTOP-O0B2VHP:/mnt/c/Users/User/OSI/lab3/src$ gcc -o child child.c kirill@DESKTOP-O0B2VHP:/mnt/c/Users/User/OSI/lab3/src$ ./parent file.txt

Sum in 1 line = 10

Sum in 2 line = 11

Sum in 3 line = 12

Error, incorrect value in 4 line Sum in 5 line = 14

Sum in 6 line = 15

kirill@DESKTOP-O0B2VHP:/mnt/c/Users/User/OSI/lab3/src$ strace ./parent file.txt execve("./parent", ["./parent", "file.txt"], 0x7fff3f13e248 /\* 28 vars \*/) = 0 brk(NULL) = 0x55df1ddba000

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

0x7fad75da5000

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=20231, ...}) = 0

mmap(NULL, 20231, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fad75da0000

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) = 0x7fad75b8e000

mmap(0x7fad75bb6000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7fad75bb6000 mmap(0x7fad75d3e000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3,

0x1b0000) = 0x7fad75d3e000

mmap(0x7fad75d8d000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x7fad75d8d000 mmap(0x7fad75d93000, 52624, PROT\_READ|PROT\_WRITE,

MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7fad75d93000

close(3) = 0

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

0x7fad75b8b000

arch\_prctl(ARCH\_SET\_FS, 0x7fad75b8b740) = 0 set\_tid\_address(0x7fad75b8ba10) = 614

set\_robust\_list(0x7fad75b8ba20, 24) = 0

rseq(0x7fad75b8c060, 0x20, 0, 0x53053053) = 0

mprotect(0x7fad75d8d000, 16384, PROT\_READ) = 0

mprotect(0x55df1cbf5000, 4096, PROT\_READ) = 0

mprotect(0x7fad75ddd000, 8192, PROT\_READ) = 0

prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0 munmap(0x7fad75da0000, 20231) = 0

getrandom("\x90\xfc\x76\xf4\x31\x22\xd1\x6c", 8, GRND\_NONBLOCK) = 8 brk(NULL) = 0x55df1ddba000

brk(0x55df1dddb000) = 0x55df1dddb000 openat(AT\_FDCWD, "file.txt", O\_RDONLY) = 3

openat(AT\_FDCWD, "/dev/shm/shared\_memory", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0666) = 4

ftruncate(4, 1024) = 0

mmap(NULL, 1024, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 4, 0) = 0x7fad75da4000

openat(AT\_FDCWD, "/dev/shm/sem.sem\_write", O\_RDWR|O\_NOFOLLOW|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

getrandom("\xaf\x08\x9a\x86\xe9\xdb\xb8\x03", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.twWXNJ", 0x7ffe67586070, AT\_SYMLINK\_NOFOLLOW) = -1

ENOENT (No such file or directory) openat(AT\_FDCWD, "/dev/shm/sem.twWXNJ",

O\_RDWR|O\_CREAT|O\_EXCL|O\_NOFOLLOW|O\_CLOEXEC, 0666) = 5

write(5, "\1\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) = 0x7fad75da3000

link("/dev/shm/sem.twWXNJ", "/dev/shm/sem.sem\_write") = 0 fstat(5, {st\_mode=S\_IFREG|0644, st\_size=32, ...}) = 0 unlink("/dev/shm/sem.twWXNJ") = 0

close(5) = 0

openat(AT\_FDCWD, "/dev/shm/sem.sem\_read", O\_RDWR|O\_NOFOLLOW|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

getrandom("\x60\x41\xc2\xf4\xdb\x09\xca\x4a", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.yMpJR2", 0x7ffe67586070, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/dev/shm/sem.yMpJR2",

O\_RDWR|O\_CREAT|O\_EXCL|O\_NOFOLLOW|O\_CLOEXEC, 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) = 0x7fad75da2000

link("/dev/shm/sem.yMpJR2", "/dev/shm/sem.sem\_read") = 0 fstat(5, {st\_mode=S\_IFREG|0644, st\_size=32, ...}) = 0 unlink("/dev/shm/sem.yMpJR2") = 0

close(5) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7fad75b8ba10) = 615

fstat(3, {st\_mode=S\_IFREG|0777, st\_size=66, ...}) = 0 read(3, "2 3 5\r\n1 1 9\r\n12 0 0\r\n16 -2 -1 D"..., 4096) = 66

futex(0x7fad75da3000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL,

FUTEX\_BITSET\_MATCH\_ANYSum in 1 line = 10

) = 0

futex(0x7fad75da2000, FUTEX\_WAKE, 1Sum in 2 line = 11

) = 1

futex(0x7fad75da2000, FUTEX\_WAKE, 1Sum in 3 line = 12

) = 1

futex(0x7fad75da2000, FUTEX\_WAKE, 1Error, incorrect value in 4 line

) = 1

futex(0x7fad75da2000, FUTEX\_WAKE, 1) = 1

Sum in 5 line = 14

read(3, "", 4096) = 0

futex(0x7fad75da2000, FUTEX\_WAKE, 1Sum in 6 line = 15

) = 1

futex(0x7fad75da2000, FUTEX\_WAKE, 1) = 1

wait4(-1, NULL, 0, NULL) = 615

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=615, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

close(3) = 0

munmap(0x7fad75da4000, 1024) = 0

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

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

unlink("/dev/shm/sem.sem\_read") = 0 exit\_group(0) = ?

+++ exited with 0 +++

kirill@DESKTOP-O0B2VHP:/mnt/c/Users/User/OSI/lab3/src$

# 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.