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: Григорян А.А.

Преподаватель: Бахарев В.Д.

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

Дата: 22.02.25

Москва, 2025

**Постановка задачи**

**Вариант 6.**

**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.
* ssize\_t write(int fd, const void buf[.count], size\_t count); - пишет count байтов из буфера в файл, на который ссылается файловый дескриптор fd
* ssize\_t read(int fd, void buf[.count], size\_t count) – пытается прочитать count байтов из файлового дескриптора fd в буфер buff
* pid\_t getpid(void); - получить pid текущего или родительского процесса
* int open(const char \*pathname, int flags, mode\_t mode ); - открыть файл с указанными флагами или создать, если указаны специальные флаги, возвращает файловый дескриптор
* int close(int fd); - закрывает файловый дескриптор
* int execv(const char \*pathname, char \*const argv[]); - заменяет образ текущего процесса на новый образ, создается новый стек, кучу и сегменты данных
* pid\_t waidpid (pid\_t pid, int \*stat\_loc, int options); - ждет пока процесс с pid завершится и получается код выхода
* getpid(): Возвращает идентификатор текущего процесса.
* shm\_open(): Открывает или создает объект разделяемой памяти.
* ftruncate(): Изменяет размер файла.
* mmap(): Отображает файл или устройство в память.
* munmap(): Удаляет отображение памяти.
* sem\_open(): Открывает или создает именованный семафор.
* sem\_close(): Закрывает именованный семафор.
* sem\_wait(): Уменьшает значение семафора, блокируя, если значение равно нулю.
* sem\_post(): Увеличивает значение семафора.
* unlink(): Удаляет имя файла.
* waitpid(): Ожидает завершения дочернего процесса.

В данной лабораторной работе реализуется межпроцессное взаимодействие с использованием разделяемой памяти и семафоров. Программа parent.c выполняет чтение строк из входного текстового файла, передает их в разделяемую память, а программа child.c получает эти данные, выполняет обработку и возвращает результат обратно.

Основной принцип работы заключается в том, что parent.c считывает строки из файла, передает их в разделяемую память и уведомляет child.c о готовности данных. В свою очередь, child.c ожидает данные, обрабатывает их, вычисляя сумму чисел в строке, и записывает результат обратно в разделяемую память. Процессы взаимодействуют между собой с помощью семафоров, что обеспечивает синхронизацию и предотвращает состояние гонки.

Работа parent.c начинается с чтения имени файла из стандартного ввода. Затем программа открывает указанный файл и создает объект разделяемой памяти с именем /shared\_memory\_lab, а также два семафора: SEM\_PARENT и SEM\_CHILD, которые управляют доступом к памяти. Далее с помощью вызова fork() создается дочерний процесс, который выполняет программу child.c посредством execl. В основном цикле parent.c считывает содержимое файла, разбивает его на строки, копирует их в разделяемую память и отправляет сигнал child.c о наличии новых данных. Дождавшись обработки строки, parent.c получает результат из разделяемой памяти и выводит его на экран. По завершении работы программа освобождает ресурсы, закрывает файлы, удаляет семафоры и освобождает разделяемую память.

Программа child.c подключается к уже существующему объекту разделяемой памяти и семафорам. В бесконечном цикле она ожидает, когда parent.c отправит новую строку. Получив данные, child.c выполняет их обработку: разбивает строку на числа, вычисляет их сумму и записывает результат обратно в разделяемую память.

После этого программа сигнализирует parent.c о завершении обработки. Если child.c обнаруживает пустую строку в памяти, это означает окончание работы, после чего программа завершает выполнение и освобождает используемые ресурсы.

**Код программы**

**parent.с**

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <semaphore.h>

#include <stdlib.h>

#include <string.h>

#define SHM\_NAME "/shared\_memory\_lab"

#define SEM\_PARENT "/sem\_parent"

#define SEM\_CHILD "/sem\_child"

#define SHM\_SIZE 1024

int main() {

char filename[256];

int len = read(STDIN\_FILENO, filename, sizeof(filename));

if (len <= 0) \_exit(1);

filename[len - 1] = '\0';

int file\_fd = open(filename, O\_RDONLY);

if (file\_fd == -1) \_exit(1);

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

ftruncate(shm\_fd, SHM\_SIZE);

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

sem\_t \*sem\_parent = sem\_open(SEM\_PARENT, O\_CREAT, 0666, 0);

sem\_t \*sem\_child = sem\_open(SEM\_CHILD, O\_CREAT, 0666, 0);

if (fork() == 0) {

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

\_exit(1);

}

char buffer[SHM\_SIZE];

while (1) {

int bytes\_read = read(file\_fd, buffer, sizeof(buffer) - 1);

if (bytes\_read <= 0) break;

buffer[bytes\_read] = '\0';

char \*line = strtok(buffer, "\n");

while (line) {

memcpy(shm\_ptr, line, strlen(line) + 1);

sem\_post(sem\_child);

sem\_wait(sem\_parent);

write(STDOUT\_FILENO, shm\_ptr, strlen((char\*)shm\_ptr));

write(STDOUT\_FILENO, "\n", 1);

line = strtok(NULL, "\n");

}

}

memset(shm\_ptr, 0, SHM\_SIZE);

sem\_post(sem\_child);

close(file\_fd);

wait(NULL);

munmap(shm\_ptr, SHM\_SIZE);

shm\_unlink(SHM\_NAME);

sem\_close(sem\_parent);

sem\_close(sem\_child);

sem\_unlink(SEM\_PARENT);

sem\_unlink(SEM\_CHILD);

return 0;

}

**Протокол работы программы**

[arcsenius@ars-nbdewxx9 src]$ strace ./parent

execve("./parent", ["./parent"], 0x7ffd854e1520 /\* 66 vars \*/) = 0

brk(NULL) = 0x63101ca85000

openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

fstat(3, {st\_mode=S\_IFREG|0644, st\_size=241615, ...}) = 0

mmap(NULL, 241615, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7a975d337000

close(3) = 0

openat(AT\_FDCWD, "/usr/lib/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\340\_\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=2014520, ...}) = 0

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7a975d335000

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, 2034616, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7a975d144000

mmap(0x7a975d168000, 1511424, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x24000) = 0x7a975d168000

mmap(0x7a975d2d9000, 319488, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x195000) = 0x7a975d2d9000

mmap(0x7a975d327000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1e3000) = 0x7a975d327000

mmap(0x7a975d32d000, 31672, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7a975d32d000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7a975d141000

arch\_prctl(ARCH\_SET\_FS, 0x7a975d141740) = 0

set\_tid\_address(0x7a975d141a10) = 15277

set\_robust\_list(0x7a975d141a20, 24) = 0

rseq(0x7a975d142060, 0x20, 0, 0x53053053) = 0

mprotect(0x7a975d327000, 16384, PROT\_READ) = 0

mprotect(0x6310092d8000, 4096, PROT\_READ) = 0

mprotect(0x7a975d3ac000, 8192, PROT\_READ) = 0

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

munmap(0x7a975d337000, 241615) = 0

read(0, input.txt

"input.txt\n", 256) = 10

openat(AT\_FDCWD, "input.txt", O\_RDONLY) = 3

openat(AT\_FDCWD, "/dev/shm/shared\_memory\_lab", 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) = 0x7a975d371000

openat(AT\_FDCWD, "/dev/shm/sem.sem\_parent", O\_RDWR|O\_NOFOLLOW|O\_CLOEXEC) = -1 ENOENT (Нет такого файла или каталога)

getrandom("\xc6\x6c\x26\x4e\xef\x25\xb0\xb0", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.yn6cXJ", 0x7ffdf9319700, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)

openat(AT\_FDCWD, "/dev/shm/sem.yn6cXJ", 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) = 0x7a975d370000

link("/dev/shm/sem.yn6cXJ", "/dev/shm/sem.sem\_parent") = 0

fstat(5, {st\_mode=S\_IFREG|0644, st\_size=32, ...}) = 0

getrandom("\x06\xc6\xbd\xd3\x19\x9a\x29\xf0", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x63101ca85000

brk(0x63101caa6000) = 0x63101caa6000

unlink("/dev/shm/sem.yn6cXJ") = 0

close(5) = 0

openat(AT\_FDCWD, "/dev/shm/sem.sem\_child", O\_RDWR|O\_NOFOLLOW|O\_CLOEXEC) = -1 ENOENT (Нет такого файла или каталога)

getrandom("\x5a\x75\xc8\xe4\xd5\xf5\xa7\xd6", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.EQe0YO", 0x7ffdf9319700, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)

openat(AT\_FDCWD, "/dev/shm/sem.EQe0YO", 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) = 0x7a975d36f000

link("/dev/shm/sem.EQe0YO", "/dev/shm/sem.sem\_child") = 0

fstat(5, {st\_mode=S\_IFREG|0644, st\_size=32, ...}) = 0

unlink("/dev/shm/sem.EQe0YO") = 0

close(5) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7a975d141a10) = 15279

read(3, "10 20 30\n5 15\n100 200 300\n", 1023) = 26

futex(0x7a975d370000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL, FUTEX\_BITSET\_MATCH\_ANYChild received: 10 20 30

Parsed number: 10

Parsed number: 20

Parsed number: 30

Child computed: 60

) = 0

write(1, "60", 260) = 2

write(1, "\n", 1

) = 1

futex(0x7a975d36f000, FUTEX\_WAKE, 1Child received: 5 15) = 1

Parsed number: 5

Parsed number: 15

Child computed: 20

futex(0x7a975d370000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL, FUTEX\_BITSET\_MATCH\_ANY) = -1 EAGAIN (Ресурс временно недоступен)

write(1, "20", 220) = 2

write(1, "\n", 1

) = 1

futex(0x7a975d36f000, FUTEX\_WAKE, 1Child received: 100 200 300

Parsed number: 100) = 1

Parsed number: 200

Parsed number: 300

Child computed: 600

futex(0x7a975d370000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL, FUTEX\_BITSET\_MATCH\_ANY) = -1 EAGAIN (Ресурс временно недоступен)

write(1, "600", 3600) = 3

write(1, "\n", 1

) = 1

read(3, "", 1023) = 0

futex(0x7a975d36f000, FUTEX\_WAKE, 1) = 1

close(3) = 0

wait4(-1, NULL, 0, NULL) = 15279

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=15279, si\_uid=1001, si\_status=0, si\_utime=0, si\_stime=0} ---

munmap(0x7a975d371000, 1024) = 0

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

munmap(0x7a975d370000, 32) = 0

munmap(0x7a975d36f000, 32) = 0

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

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

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

В ходе данной работы я научился создавать процессы, налаживать общение между ними с помощью shared memory. Столкнулся с проблемами при синхронизации с помощью семафоров.