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: Кудаева В.А.

Преподаватель: Бахарев В.Д.

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

Дата: 24.12.24

Москва, 2024

**Постановка задачи**

**Вариант 16:**

Родительский процесс создает дочерний процесс. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия file с таким именем на запись. Родительский процесс принимает от пользователя строки произвольной длины. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в консоль выводится информация об ошибке. Правило проверки: строка должна оканчиваться на «.» или «;».

**Общий метод и алгоритм решения**

Использованные системные вызовы:

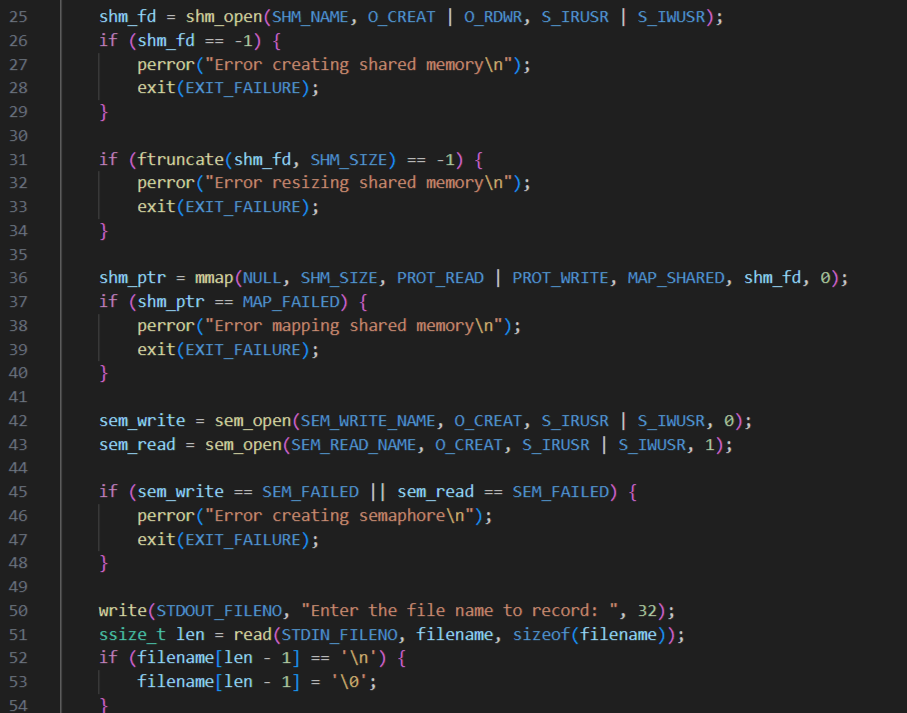
* pid\_t fork(void); – создает дочерний процесс;
* int shm\_open(const char \*name, int oflag, mode\_t mode) - создает или открывает объект разделяемой памяти;
* int write(int fd, const void\* buff, int count) – записывает по дескриптору fd count байт из buff;
* void exit(int number) – вызывает нормальное завершение программы с кодом number;
* int ftruncate(int fd, off\_t length) - устанавливает размер объекта разделяемой памяти;
* void \*mmap(void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset) - отображает объект разделяемой памяти в адресное пространство процесса;
* sem\_t \*sem\_open(const char \*name, int oflag, mode\_t mode, unsigned int value) - создает или открывает именованный семафор;
* int sem\_wait(sem\_t \*sem) - уменьшает значение семафора на 1. Если семафор уже равен 0, процесс блокируется;
* int sem\_post(sem\_t \*sem) - увеличивает значение семафора на 1. Разблокирует процесс, ожидающий семафора;
* int sem\_close(sem\_t \*sem) - закрывает именованный семафор в текущем процессе;
* int sem\_unlink(const char \*name) - удаляет именованный семафор из системы;
* int execl(char\* path, const char\* argc); – заменяет текущий процесс на процесс path, с аргументами argc;
* int close(int fd); – закрывает дескриптор fd;
* pid\_t wait(int status) — функция, которая приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится.

В файле parent.c создается родительский и дочерний процессы. Пользователь вводит строку в консоль. Если строка равна "exit", родитель записывает это слово в разделяемую память, уведомляет дочерний процесс через sem\_post(sem\_read) и завершает цикл. Если строка не равна "exit", родитель копирует строку в разделяемую память. Затем он вызывает sem\_post(sem\_read), чтобы разрешить дочернему процессу прочитать данные. Дочерний процесс ждет (sem\_wait(sem\_read)), пока родитель не заполнит память, и только после этого читает данные. После этого дочерний процесс проверяет строку на валидность и если строка корректна, то выводит её в файл, в противном случае выводит сообщение об ошибке в консоль. Затем уведомляет (sem\_post(sem\_write)) родителя о том, что данные обработаны. Родительский процесс ждет завершения обработки данных, прежде чем записать новые. Таким образом, семафор sem\_read обеспечивает, что дочерний процесс не будет читать данные из разделяемой памяти до тех пор, пока родительский процесс не заполнит их, а семафор sem\_write обеспечивает, что родительский процесс не запишет новые данные в разделяемую память, пока дочерний процесс не обработает текущие данные.

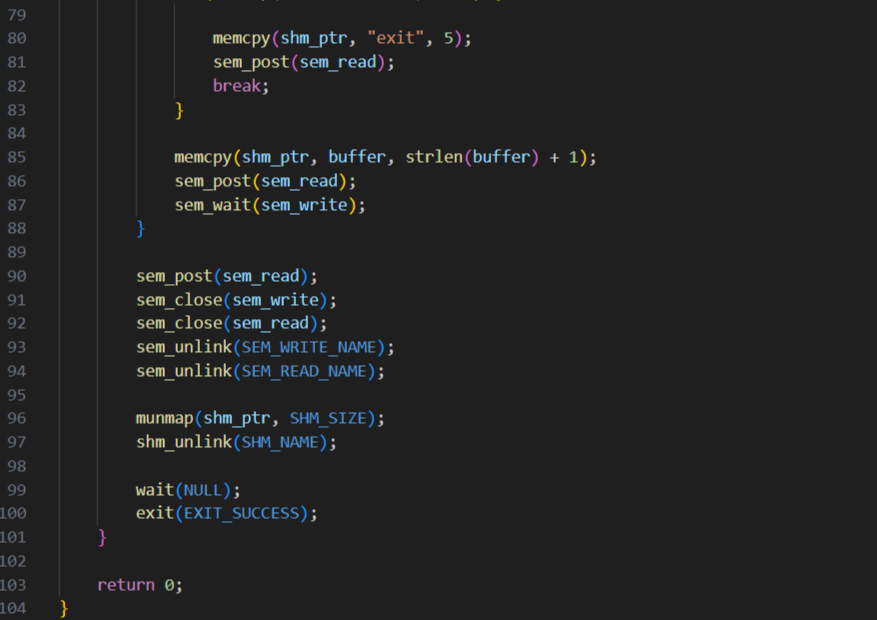
**Код программы**

**parent.c**

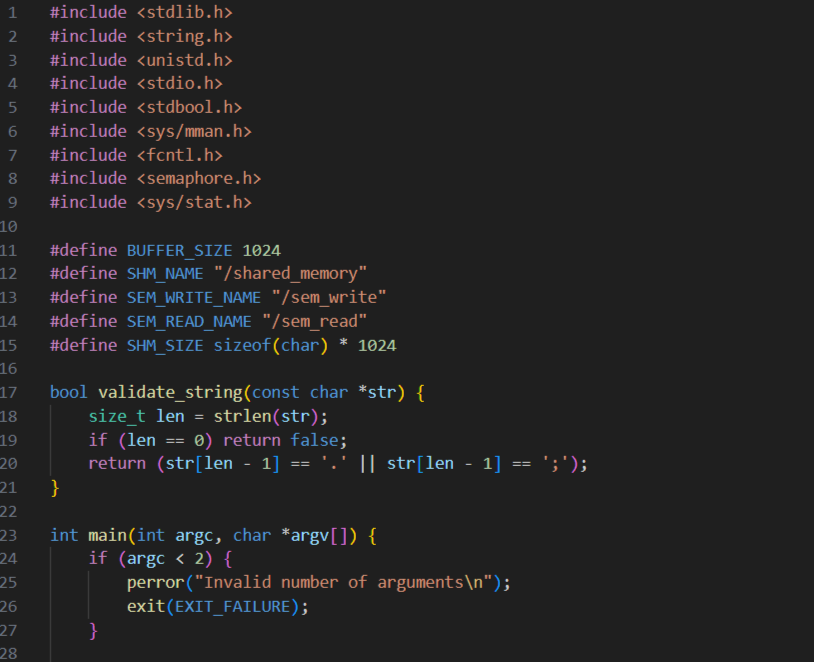
****

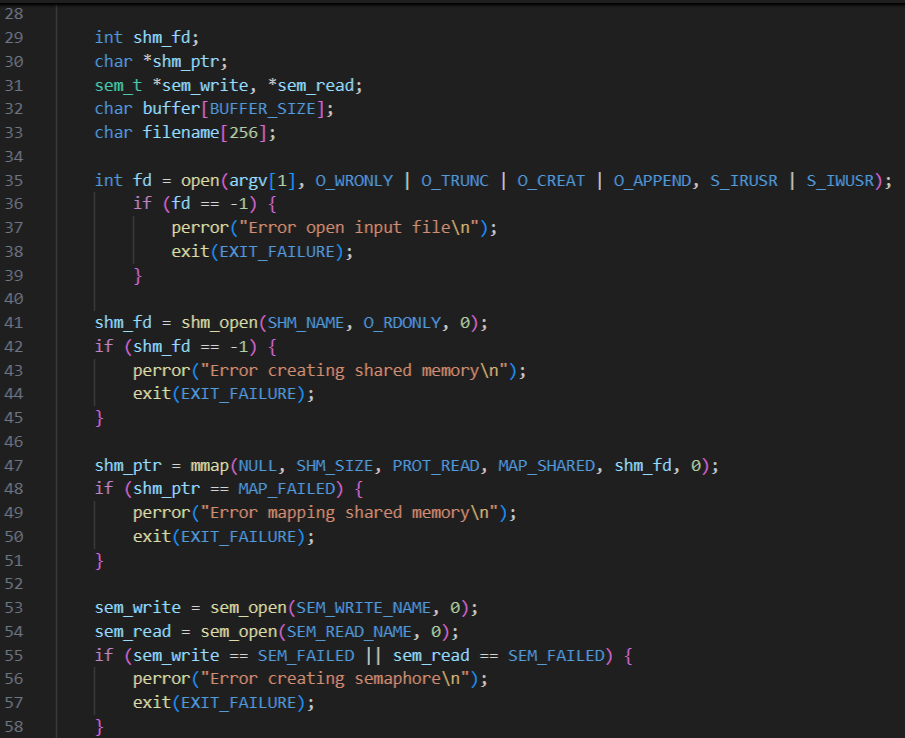
****

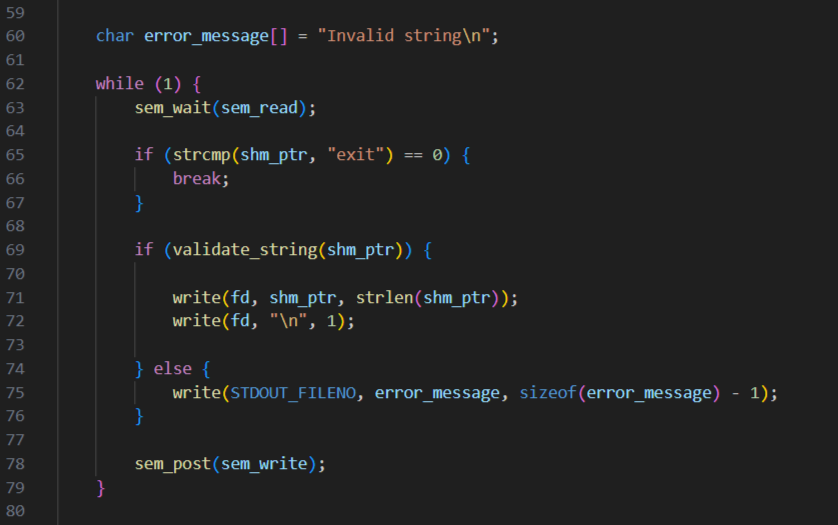
****

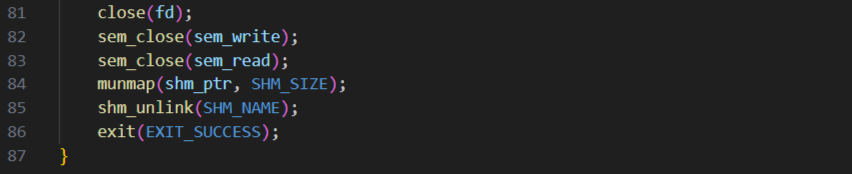
****

**child.c**

****

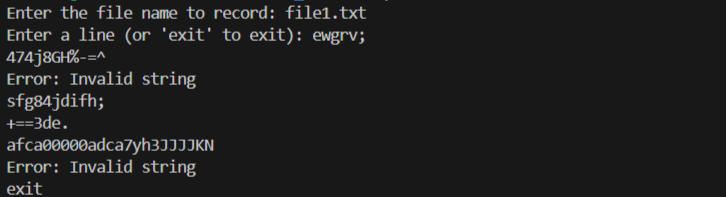
****

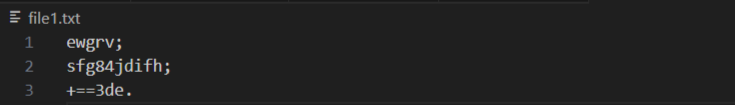
****

****

**Протокол работы программы**

**Тестирование**

****

****

**Вывод strace**

execve("./parent", ["./parent"], 0x7ffee35fd0c0 /\* 73 vars \*/) = 0

brk(NULL) = 0x55e0c4363000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffd47f40af0) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7fb776086000

access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

newfstatat(AT\_FDCWD, "/usr/lib/openfoam/openfoam2312/platforms/linux64GccDPInt32Opt/lib/dummy", {st\_mode=S\_IFDIR|0755, st\_size=4096, ...}, 0) = 0

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

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=27331, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 27331, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7fb77607f000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

mprotect(0x7fb7760c0000, 8192, PROT\_READ) = 0

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

munmap(0x7fb77607f000, 27331) = 0

openat(AT\_FDCWD, "/dev/shm/shared\_memory", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0600) = 3

ftruncate(3, 1024) = 0

mmap(NULL, 1024, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7fb7760bf000

openat(AT\_FDCWD, "/dev/shm/sem.sem\_write", O\_RDWR|O\_NOFOLLOW) = 4

newfstatat(4, "", {st\_mode=S\_IFREG|0600, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0

getrandom("\xa3\xc5\x56\x94\xb6\x05\xf1\x5f", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x55e0c4363000

brk(0x55e0c4384000) = 0x55e0c4384000

mmap(NULL, 32, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 4, 0) = 0x7fb776085000

close(4) = 0

openat(AT\_FDCWD, "/dev/shm/sem.sem\_read", O\_RDWR|O\_NOFOLLOW) = 4

newfstatat(4, "", {st\_mode=S\_IFREG|0600, st\_size=32, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 32, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 4, 0) = 0x7fb776084000

close(4) = 0

write(1, "Enter the file name to record: \0", 32Enter the file name to record: ) = 32

read(0, file1.txt

"file1.txt\n", 256) = 10

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7fb775e53a10) = 52936

close(3) = 0

write(1, "Enter a line (or 'exit' to exit)"..., 35Enter a line (or 'exit' to exit): ) = 35

read(0, dhdjde

"dhdjde\n", 1024) = 7

futex(0x7fb776084000, FUTEX\_WAKE, 1) = 1

read(0, dnjdjd;

"dnjdjd;\n", 1024) = 8

futex(0x7fb776084000, FUTEX\_WAKE, 1) = 1

read(0, exit

"exit\n", 1024) = 5

futex(0x7fb776084000, FUTEX\_WAKE, 1) = 1

futex(0x7fb776084000, FUTEX\_WAKE, 1) = 0

munmap(0x7fb776085000, 32) = 0

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=52936, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

munmap(0x7fb776084000, 32) = 0

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

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

munmap(0x7fb7760bf000, 1024) = 0

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

wait4(-1, NULL, 0, NULL) = 52936

exit\_group(0) = ?

+++ exited with 0 +++

**Вывод**

В ходе лабораторной работы мне удалось реализовать взаимодействие между родительским и дочерним процессами с использованием разделяемой памяти и семафоров для организации межпроцессного взаимодействия. Мною были изучены механизмы управления доступом к разделяемым ресурсам для предотвращения гонок данных. В данной работе потребовалось использование двух семафоров для обеспечения корректного порядка выполнения операций чтения и записи между процессами.