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-215Б-23

Студент: Дехтеренко Д.С.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 13.03.25

Москва, 2025

Постановка задачи

Вариант 10.

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». Дочерний процесс производит проверку этого числа на простоту. Если число составное, то дочерний процесс пишет это число в стандартный поток вывода. Если число отрицательное или простое, то тогда дочерний и родительский процессы завершаются. Количество чисел может быть произвольным.

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программы (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork() - создание дочернего процесса
- execl(const char *path, const char *arg, ...) – замена памяти процесса
- pid_t waitpid(pid_t pid, int *status, int options)- ожидание завершения дочернего процесса
- int dup2(int oldfd, int newfd) - переназначение файлового дескриптора
- int open(const char *pathname, int flags, mode_t mode) - открытие\создание файла
- int close(int fd) - закрыть файл
- size_t read (int fd, void* buf, size_t cnt) – чтение из файла
- size_t write (int fd, void* buf, size_t cnt) – запись в файл
- int unlink(const char *pathname) - удаляет имя из файловой системы
- void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset) - создает новое отображение в виртуальном адресном пространстве вызывающего процесса
- int munmap(void *addr, size_t length) - удаляет отображения для указанного диапазона адресов

Создаём отображение временного файла. В программе родительского процесса создаём массив символов, в который с клавиатуры пользователем считывается строка – имя файла с числами. Далее этот файл открываем на чтение. Создаём дочерний процесс. В дочернем процессе поток ввода

переопределяем файлом, далее закрываем в неё дескриптор открытого файла. Используем execl для замены памяти процесса на программу, написанную для дочернего процесса. В программе дочернего процесса отображаем тот же файл.

В родительском процессе закрываем дескриптор открытого файла с числами. Считываем в цикле из отображаемой общей памяти числа.

В программе дочернего процесса считываем из входного потока числа и проверяем на простоту. В случае успех процесс завершается. В случае неуспеха числа записываются в отображаемую общую память.

Для синхронизации используются семафоры, чтобы родительский процесс читал только те данные, которые уже записал дочерний.

При использовании системных вызовов обрабатываются ошибки и освобождаются используемы ресурсы.

Код программы

parent.c

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


int main()
{

    char file_name[101];
    scanf("%100s", file_name);
    int d = open(file_name, O_RDONLY);

    if (d == -1)
    {
        perror("open failed");
        return 1;
    }
```

```
int pipe1[2];

if (pipe(pipe1) == -1)
{
   close(d);
   perror("pipe failed");
   return 2;
}

pid_t pid = fork();

if (-1 == pid)
{
   close(d), close(pipe1[0]), close(pipe1[1]);
   perror("fork");
   return 6;
}


if (pid == 0)
{
   close(pipe1[0]);

   if (dup2(d, STDIN_FILENO) == -1)
   {
      close(d), close(pipe1[1]);
      perror("dup2 failed");
      return 3;
   }

   close(d);

   if (dup2(pipe1[1], STDOUT_FILENO) == -1)
   {
      close(pipe1[1]);
      perror("dup2 failed");
```

```
        return 3;
    }

    execl("child", "child", NULL);
    close(pipe1[1]);
    perror("execl() failed");
    return 4;
}
else
{
    close(d);
    close(pipe1[1]);
    char number [1000];
    char symbol;
    int i = 0;
    size_t status;

    while ((status = read(pipe1[0], &symbol, sizeof(symbol))) > 0)
    {
        if (symbol == '\n')
        {
            number[i] = 0;
            write(STDOUT_FILENO, number, strlen(number));
            write(STDOUT_FILENO, "\n", 1);
            i = 0;
        }
        else
        {
            number[i++] = symbol;
        }
    }


    close(pipe1[0]);
    wait(NULL);

    if (status == -1)
```

```
        {
            perror("read failed");
            return 5;
        }
    }
    return 0;
}
```

child.c

```
#include "stdio.h"
#include "stdlib.h"
#include "unistd.h"
#include <string.h>
#include <stdbool.h>
#define MAX_SIZE 1000



bool prime(int n)
{
    for(int i = 2; i * i <= n; i++)
        if(n % i == 0)
            return false;
    return true;
}

int main()
{
    char number[MAX_SIZE];
    char symbol;
    int i = 0;
    while ((read(STDIN_FILENO, &symbol, sizeof(symbol))) > 0)
    {
        if (symbol == '\n')
        {
            number[i] = 0;
            if (prime(atoi(number)))
```

```
            {
                return 0;
            }
            write(STDOUT_FILENO, number, strlen(number));
            write(STDOUT_FILENO, "\n", 1);
            i = 0;
        }
        else
        {
            number[i++] = symbol;
        }
    }
    return 0;
}
```

Протокол работы программы

Тест 1:

```
luckyabatur@Luckyabatur:~/projects/OS_labs/lab3/test$ cat numbers.txt
212
32
43221242
34
42
18
8
98888888
124
3
7
88
156
11
luckyabatur@Luckyabatur:~/projects/OS_labs/lab3/test$ cd ../src
luckyabatur@Luckyabatur:~/projects/OS_labs/lab3/src$ ./main
```

../test/numbers.txt

212

32

43221242

34

42

18

8

98888888

124

Тест 2:

luckyabatur@Luckyabatur:~/projects/OS_labs/lab3/test$ cat numbers.txt

17

212

32

43221242

34luckyabatur@Luckyabatur:~/projects/OS_labs/lab3/test$ cd ../src

luckyabatur@Luckyabatur:~/projects/OS_labs/lab3/src$ ./main

../test/numbers.txt

Тест 3:

luckyabatur@Luckyabatur:~/projects/OS_labs/lab3/test$ cat numbers.txt

luckyabatur@Luckyabatur:~/projects/OS_labs/lab3/test$ cd ../src

luckyabatur@Luckyabatur:~/projects/OS_labs/lab3/src$ ./main

../test/numbers.txt

Strace:

luckyabatur@Luckyabatur:~/projects/OS_labs/lab3/src$ echo ../test/numbers.txt | strace -f ./main

execve("./main", ["./main"], 0x7fffae8d6458 /* 36 vars */) = 0

brk(NULL) = 0x55b73c3c8000

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe2691d2000

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=24775, ...}) = 0

mmap(NULL, 24775, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fe2691cb000

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) = 0x7fe268fb9000
mmap(0x7fe268fe1000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fe268fe1000
mmap(0x7fe269169000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7fe269169000
mmap(0x7fe2691b8000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7fe2691b8000
mmap(0x7fe2691be000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fe2691be000
close(3) = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fe268fb6000
arch_prctl(ARCH_SET_FS, 0x7fe268fb6740) = 0
set_tid_address(0x7fe268fb6a10) = 4732
set_robust_list(0x7fe268fb6a20, 24) = 0
rseq(0x7fe268fb7060, 0x20, 0, 0x53053053) = 0
mprotect(0x7fe2691b8000, 16384, PROT_READ) = 0
mprotect(0x55b73b4ea000, 4096, PROT_READ) = 0
mprotect(0x7fe26920a000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7fe2691cb000, 24775) = 0
unlink("/dev/shm/sem.mysem") = -1 ENOENT (No such file or directory)
getrandom("\x68\x6b\x27\xab\x3b\x7a\x88\xe4", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.ITeM2P", 0x7ffc0d625210, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.ITeM2P", O_RDWR|O_CREAT|O_EXCL|O_NOFOLLOW|O_CLOEXEC, 0666) = 3
write(3, "\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, 3, 0) = 0x7fe2691d1000
link("/dev/shm/sem.ITeM2P", "/dev/shm/sem.mysem") = 0
fstat(3, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0

getrandom("\x41\x53\x0a\xd2\xfc\x0e\x83\x80", 8, GRND_NONBLOCK) = 8

brk(NULL) = 0x55b73c3c8000

brk(0x55b73c3e9000) = 0x55b73c3e9000

unlink("/dev/shm/sem.ITeM2P") = 0

close(3) = 0

openat(AT_FDCWD, "temp", O_RDWR|O_CREAT|O_TRUNC, 0666) = 3

ftruncate(3, 4096) = 0

mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7fe2691d0000

fstat(0, {st_mode=S_IFIFO|0600, st_size=0, ...}) = 0

read(0, "../test/numbers.txt\n", 4096) = 20

openat(AT_FDCWD, "../test/numbers.txt", O_RDONLY) = 4

clone(child_stack=NULL,

flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process 4733 attached

, child_tidptr=0x7fe268fb6a10) = 4733

[pid 4733] set_robust_list(0x7fe268fb6a20, 24 <unfinished ...>

[pid 4732] close(4 <unfinished ...>

[pid 4733] <... set_robust_list resumed>) = 0

[pid 4732] <... close resumed>) = 0

[pid 4732] futex(0x7fe2691d1000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 4733] dup2(4, 0) = 0

[pid 4733] close(4) = 0

[pid 4733] execve("child", ["child"], 0x7ffc0d625768 /* 36 vars */) = 0

[pid 4733] brk(NULL) = 0x555ad330c000

[pid 4733] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fa8737bc000

[pid 4733] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

[pid 4733] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 4

[pid 4733] fstat(4, {st_mode=S_IFREG|0644, st_size=24775, ...}) = 0

[pid 4733] mmap(NULL, 24775, PROT_READ, MAP_PRIVATE, 4, 0) = 0x7fa8737b5000

[pid 4733] close(4) = 0

[pid 4733] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 4

[pid 4733] read(4, "\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"..., 832) = 832

[pid 4733] pread64(4, "\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

[pid 4733] fstat(4, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0

[pid 4733] pread64(4, "\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

[pid 4733] mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 4, 0) =

0x7fa8735a3000

[pid 4733] mmap(0x7fa8735cb000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x28000) = 0x7fa8735cb000

[pid 4733] mmap(0x7fa873753000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x1b0000) = 0x7fa873753000

[pid 4733] mmap(0x7fa8737a2000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 4, 0x1fe000) = 0x7fa8737a2000

[pid 4733] mmap(0x7fa8737a8000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fa8737a8000

[pid 4733] close(4) = 0

[pid 4733] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fa8735a0000

[pid 4733] arch_prctl(ARCH_SET_FS, 0x7fa8735a0740) = 0

[pid 4733] set_tid_address(0x7fa8735a0a10) = 4733

[pid 4733] set_robust_list(0x7fa8735a0a20, 24) = 0

[pid 4733] rseq(0x7fa8735a1060, 0x20, 0, 0x53053053) = 0

[pid 4733] mprotect(0x7fa8737a2000, 16384, PROT_READ) = 0

[pid 4733] mprotect(0x555ad2e75000, 4096, PROT_READ) = 0

[pid 4733] mprotect(0x7fa8737f4000, 8192, PROT_READ) = 0

[pid 4733] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

[pid 4733] munmap(0x7fa8737b5000, 24775) = 0

[pid 4733] openat(AT_FDCWD, "temp", O_RDWR) = 4

[pid 4733] mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7fa8737bb000

[pid 4733] openat(AT_FDCWD, "/dev/shm/sem.mysem", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 5

[pid 4733] fstat(5, {st_mode=S_IFREG|0644, st_size=32, ...}) = 0

[pid 4733] getrandom("\x21\xd6\x72\x16\x8d\xc8\xb1\x3a", 8, GRND_NONBLOCK) = 8

[pid 4733] brk(NULL) = 0x555ad330c000

[pid 4733] brk(0x555ad332d000) = 0x555ad332d000

[pid 4733] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x7fa8737ba000

[pid 4733] close(5) = 0

[pid 4733] read(0, "212\n32\n43221242\n34\n42\n18\n8\n98888"..., 10000) = 53

[pid 4733] futex(0x7fa8737ba000, FUTEX_WAKE, 1 <unfinished ...>

[pid 4732] <... futex resumed>) = 0

[pid 4733] <... futex resumed>) = 1

[pid 4732] fstat(1, <unfinished ...>

```
[pid  4733] munmap(0x7fa8737ba000, 32 <unfinished ...>
[pid  4732] <... fstat resumed>{st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x2), ...}) = 0
[pid  4733] <... munmap resumed>)       = 0
[pid  4732] write(1, "212\n", 4 <unfinished ...>
212
[pid  4733] munmap(0x7fa8737bb000, 4096 <unfinished ...>
[pid  4732] <... write resumed>)        = 4
[pid  4733] <... munmap resumed>)       = 0
[pid  4732] write(1, "32\n", 3 <unfinished ...>
32
[pid  4733] close(4 <unfinished ...>
[pid  4732] <... write resumed>)        = 3
[pid  4733] <... close resumed>)        = 0
[pid  4732] write(1, "43221242\n", 943221242
 <unfinished ...>
[pid  4733] exit_group(0 <unfinished ...>
[pid  4732] <... write resumed>)        = 9
[pid  4733] <... exit_group resumed>)   = ?
[pid  4732] write(1, "34\n", 334
)        = 3
[pid  4733] +++ exited with 0 +++
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=4733, si_uid=1000, si_status=0,
si_utime=0, si_stime=0} ---
write(1, "42\n", 342
)                = 3
write(1, "18\n", 318
)                = 3
write(1, "8\n", 28
)                 = 2
write(1, "98888888\n", 998888888
)             = 9
write(1, "124\n", 4124
)                = 4
wait4(4733, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 4733
munmap(0x7fe2691d0000, 4096)          = 0
close(3)                          = 0
munmap(0x7fe2691d1000, 32)            = 0
```

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

unlink("temp") = 0

lseek(0, -1, SEEK_CUR) = -1 ESPIPE (Illegal seek)

exit_group(0) = ?

+++ exited with 0 +++

Вывод

При выполнении работы познакомился с технологией memory mapping. Пытался работать с пустым файлом, из-за чего получал SIGBUS, не сразу понял, как правильно сделать синхронизацию.