# Лабораторная работа №3 по курсу «Операционные системы»

Группа: М80-210Б-23

Студент: Жаворонков Никита Дмитриевич

Преподаватель: Бахарев В.Д. (ФИИТ)

27.12.2024

Постановка задачи

**Вариант 10.**

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.

# Общий метод и алгоритм решения

1. **Чтение файла:** Родительский процесс считывает числа из файла построчно.
2. **Создание дочернего процесса:** Для каждого прочитанного числа родительский процесс создает дочерний процесс с помощью системного вызова fork().
3. **Проверка на простоту:** Дочерний процесс проверяет число на простоту. Алгоритм проверки:
   * Если число меньше 2, то оно не является простым.
   * Если число равно 2, то оно является простым.
   * Если число больше 2, то необходимо проверить, делится ли оно на числа от 2 до квадратного корня из числа. Если делится, то число составное.
4. **Вывод результата:**
   * Если число составное, то дочерний процесс выводит его в стандартный поток вывода. o Если число отрицательное или простое, то дочерний процесс завершается.
5. **Завершение родительского процесса:** Родительский процесс ожидает завершения всех дочерних процессов с помощью системного вызова wait(). После завершения всех дочерних процессов родительский процесс также завершается.

Общий метод и алгоритм решения

Использованные системные вызовы:

* mmap – отображение файла в память
* fork – создание дочернего процесса
* execv – замена исполняемого кода
* sem\_open – создание/подключение к семафору
* sem\_post – поднятие семафора
* sem\_wait – опускание семафора
* wait – ожидание завершения процесса
* kill – завершение процесса
* sem\_unlink - уничтожает именованный семафор
* shm\_open – открывает объект разделяемой памяти
* ftruncate - укорачивает файл до указанной длины

#include <math.h>

#include <stdbool.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <semaphore.h>

#include <string.h>

#include "commandline.h"

#include "utils.h"

#include "number.h"

#define SHM\_NAME "/memA"

#define SEM\_NAMEA "/semA"

#define SEM\_NAMEB "/semB"

#define ERR\_CANT\_OPEN\_SEM 999

#define ERR\_CANT\_OPEN\_SHM 1111

int isPrime(double a) {

for (size\_t i = 2; i <= sqrt(a); i++)

if ((size\_t)a % i == 0) return 0;

return 1;

}

static inline int readchar( int file, char\* trg ) {

return read( file, trg, 1 );

}

static inline bool strempty( char\* buf ) {

return buf[0] == '\0';

}

int main(int argc, char\*\* argv) {

sem\_t\* semA = sem\_open(SEM\_NAMEA, 0);

if (semA == SEM\_FAILED) return error(ERR\_CANT\_OPEN\_SEM);

sem\_t\* semB = sem\_open(SEM\_NAMEB, 0);

if (semB == SEM\_FAILED) return error(ERR\_CANT\_OPEN\_SEM);

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

if (shm\_fd == -1) return error(ERR\_CANT\_OPEN\_SHM);

char\* shared\_mem = mmap(NULL, MAX\_NUM\_LEN, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

if (shared\_mem == MAP\_FAILED) return error(ERR\_CANT\_OPEN\_SHM);

char buffer[MAX\_NUM\_LEN + 1];

while (1) {

sem\_wait(semA);

strcpy(buffer, shared\_mem);

if( cstrtod(buffer) < 0 || isPrime(cstrtod(buffer)) ) {

shared\_mem[0] = 1;

sem\_post(semB);

break;

}else{

shared\_mem[0] = 0;

sem\_post(semB);

}

if (strempty(buffer)) break;

}

sem\_close(semA);

sem\_close(semB);

munmap(shared\_mem, MAX\_NUM\_LEN);

close(shm\_fd);

return 0;

}

#define \_GNU\_SOURCE

#include <stdbool.h>

#include <unistd.h>

#include <sys/wait.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <semaphore.h>

#include <string.h>

#include "commandline.h"

#include "utils.h"

#define CLIENT\_NAME "/client.out"

#define SHM\_NAME "/memA"

#define SEM\_NAMEA "/semA"

#define SEM\_NAMEB "/semB"

#define ERR\_CANT\_INIT\_SEM 777

#define ERR\_CANT\_INIT\_SHM 888

#define ERR\_CANT\_OPEN\_SEM 999

#define ERR\_CANT\_OPEN\_SHM 1111

struct ProgramState {

int dummy;

};

char\* getexecpath() {

char\* path = malloc(MAX\_PATH\_LEN \* 2);

size\_t len = readlink("/proc/self/exe", path, MAX\_PATH\_LEN);

if (!len) {

free(path);

return NULL;

}

while (path[len] != '/') len--;

path[len] = '\0';

return path;

}

char\* readpath(int argc, char\*\* argv) {

CMD\* command = initCMD(argc, argv);

if (!command) return NULL;

char\* path = expectCMD(command, String);

freeCMD(command);

return path;

}

static inline int readchar( int file, char\* trg ) {

return read( file, trg, 1 );

}

static inline bool strempty( char\* buf ) {

return buf[0] == '\0';

}

int readuint(const int file, char\* buf, size\_t n) {

char c;

bool foundint = false;

int readstatus;

while ((readstatus = readchar(file, &c))) {

if (readstatus == -1) return ERR\_INVALID\_INPUT;

if (IS\_BLANK(c))

if (foundint) break; else continue;

if (!IS\_DIGIT(c)) return ERR\_INVALID\_INPUT;

foundint = true;

\*(buf++) = c;

n--;

if (!n) return ERR\_INVALID\_INPUT;

}

\*(buf++) = '\0';

return NO\_ERR;

}

int main(int argc, char\*\* argv) {

char\* path;

if (!(path = getexecpath())) return error(ERR\_PATH\_READ);

char\* target;

if (!(target = readpath(argc, argv))) return error(ERR\_INVALID\_INPUT);

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

if (shm\_fd == -1) return error(ERR\_CANT\_INIT\_SHM);

if (ftruncate(shm\_fd, MAX\_NUM\_LEN) == -1) return error(ERR\_CANT\_INIT\_SHM);

char\* shared\_mem = mmap(NULL, MAX\_NUM\_LEN, PROT\_READ | PROT\_WRITE, MAP\_SHARED, shm\_fd, 0);

if (shared\_mem == MAP\_FAILED) return error(ERR\_CANT\_INIT\_SHM);

sem\_unlink(SEM\_NAMEA);

sem\_unlink(SEM\_NAMEB);

sem\_t\* semA = sem\_open(SEM\_NAMEA, O\_CREAT | O\_EXCL, S\_IRUSR | S\_IWUSR, 0);

if (semA == SEM\_FAILED) return error(ERR\_CANT\_INIT\_SEM);

sem\_t\* semB = sem\_open(SEM\_NAMEB, O\_CREAT | O\_EXCL, S\_IRUSR | S\_IWUSR, 0);

if (semB == SEM\_FAILED) return error(ERR\_CANT\_INIT\_SEM);

int exitcode = 0;

switch (fork()) {

case -1:

exitcode = ERR\_CANT\_INIT\_CHILD;

break;

case 0: {

char \*const args[] = { CLIENT\_NAME, NULL };

strcpy(&path[strlen(path)], CLIENT\_NAME);

exitcode = execv(path, args);

break;

}

default: {

free(path);

close(shm\_fd);

int file;

if ((file = open(target, O\_RDONLY)) == -1) return error(ERR\_NO\_SUCH\_FILE);

char buf[MAX\_NUM\_LEN];

int readerr;

while (!(readerr = readuint(file, buf, MAX\_NUM\_LEN)) && !strempty(buf)) {

strcpy( shared\_mem, buf );

sem\_post(semA);

sem\_wait(semB);

while( shared\_mem[0] != 0 && shared\_mem[0] != 1 );

if( shared\_mem[0] == 1 ) {

break;

}else{

write( stdout, buf, strlen(buf) );

write( stdout, "\n", 1 );

}

}

sem\_post(semA);

close(file);

int childStatus;

wait(&childStatus);

if (!WIFEXITED(childStatus)) exitcode = childStatus;

exitcode |= readerr;

break;

}

}

sem\_close(semA);

sem\_close(semB);

sem\_unlink(SEM\_NAMEA);

sem\_unlink(SEM\_NAMEB);

munmap(shared\_mem, MAX\_NUM\_LEN);

shm\_unlink(SHM\_NAME);

if (exitcode) return error(exitcode);

}

# Протокол работы программы

execve("./server.out", ["./server.out", "tests/a.txt"], 0x7ffd7e6a97f8 /\* 24 vars \*/) = 0

brk(NULL) = 0x55f8s60d05000

arch\_prctl(0x3001 /\* ARCH\_??? \*/, 0x7ffef2471e70) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f0e9b829000

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

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=17571, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 17571, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f0e9b824000

close(3) = 0

openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libm.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\0\0\0\0\0\0\0\0"..., 832) = 832

newfstatat(3, "", {st\_mode=S\_IFREG|0644, st\_size=940560, ...}, AT\_EMPTY\_PATH) = 0

mmap(NULL, 942344, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f0e9b73d000

mmap(0x7f0e9b74b000, 507904, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xe000) = 0x7f0e9b74b000

mmap(0x7f0e9b7c7000, 372736, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x8a000) = 0x7f0e9b7c7000

mmap(0x7f0e9b822000, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0xe4000) = 0x7f0e9b822000

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\0P\237\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

pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) = 68

newfstatat(3, "", {st\_mode=S\_IFREG|0755, st\_size=2220400, ...}, AT\_EMPTY\_PATH) = 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, 2264656, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f0e9b514000

mprotect(0x7f0e9b53c000, 2023424, PROT\_NONE) = 0

mmap(0x7f0e9b53c000, 1658880, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f0e9b53c000

mmap(0x7f0e9b6d1000, 360448, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1bd000) = 0x7f0e9b6d1000

mmap(0x7f0e9b72a000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x215000) = 0x7f0e9b72a000

mmap(0x7f0e9b730000, 52816, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f0e9b730000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f0e9b511000

arch\_prctl(ARCH\_SET\_FS, 0x7f0e9b511740) = 0

set\_tid\_address(0x7f0e9b511a10) = 66391

set\_robust\_list(0x7f0e9b511a20, 24) = 0

rseq(0x7f0e9b5120e0, 0x20, 0, 0x53053053) = 0

mprotect(0x7f0e9b72a000, 16384, PROT\_READ) = 0

mprotect(0x7f0e9b822000, 4096, PROT\_READ) = 0

mprotect(0x55f831362000, 4096, PROT\_READ) = 0

mprotect(0x7f0e9b863000, 8192, PROT\_READ) = 0

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

munmap(0x7f0e9b824000, 17571) = 0

getrandom("\x78\x4e\x1f\x07\xf9\x30\x39\xcb", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x55f860d05000

brk(0x55f860d26000) = 0x55f860d26000

readlink("/proc/self/exe", "/home/supertos/mai\_os\_lab\_full-m"..., 512) = 51

openat(AT\_FDCWD, "/dev/shm/memA", O\_RDWR|O\_CREAT|O\_NOFOLLOW|O\_CLOEXEC, 0666) = 3

ftruncate(3, 512) = 0

mmap(NULL, 512, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7f0e9b862000

unlink("/dev/shm/sem.semA") = -1 ENOENT (No such file or directory)

unlink("/dev/shm/sem.semB") = -1 ENOENT (No such file or directory)

getrandom("\xb7\x25\xc5\x93\xa2\x56\x16\x84", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.7GrHea", 0x7ffef2471950, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/dev/shm/sem.7GrHea", O\_RDWR|O\_CREAT|O\_EXCL, 0600) = 4

write(4, "\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, 4, 0) = 0x7f0e9b828000

link("/dev/shm/sem.7GrHea", "/dev/shm/sem.semA") = 0

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

unlink("/dev/shm/sem.7GrHea") = 0

close(4) = 0

getrandom("\x0b\xea\x41\xc6\x23\x3f\xb7\x6d", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.D7RhbL", 0x7ffef2471950, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/dev/shm/sem.D7RhbL", O\_RDWR|O\_CREAT|O\_EXCL, 0600) = 4

write(4, "\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, 4, 0) = 0x7f0e9b827000

link("/dev/shm/sem.D7RhbL", "/dev/shm/sem.semB") = 0

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

unlink("/dev/shm/sem.D7RhbL") = 0

close(4) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLD, child\_tidptr=0x7f0e9b511a10) = 66392

close(3) = 0

openat(AT\_FDCWD, "tests/a.txt", O\_RDONLY) = 3

read(3, "1", 1) = 1

read(3, "8", 1) = 1

read(3, "\r", 1) = 1

futex(0x7f0e9b828000, FUTEX\_WAKE, 1) = 1

write(1, "18", 218) = 2

write(1, "\n", 1

) = 1

read(3, "\n", 1) = 1

read(3, "3", 1) = 1

read(3, "6", 1) = 1

read(3, "\r", 1) = 1

futex(0x7f0e9b828000, FUTEX\_WAKE, 1) = 1

write(1, "36", 236) = 2

write(1, "\n", 1

) = 1

read(3, "\n", 1) = 1

read(3, "9", 1) = 1

read(3, "6", 1) = 1

read(3, "\r", 1) = 1

futex(0x7f0e9b828000, FUTEX\_WAKE, 1) = 1

write(1, "96", 296) = 2

write(1, "\n", 1

) = 1

read(3, "\n", 1) = 1

read(3, "4", 1) = 1

read(3, "8", 1) = 1

read(3, "\r", 1) = 1

futex(0x7f0e9b828000, FUTEX\_WAKE, 1) = 1

write(1, "48", 248) = 2

write(1, "\n", 1

) = 1

read(3, "\n", 1) = 1

read(3, "3", 1) = 1

read(3, "0", 1) = 1

read(3, "\r", 1) = 1

futex(0x7f0e9b828000, FUTEX\_WAKE, 1) = 1

write(1, "30", 230) = 2

write(1, "\n", 1

) = 1

read(3, "\n", 1) = 1

read(3, "7", 1) = 1

read(3, "\r", 1) = 1

futex(0x7f0e9b828000, FUTEX\_WAKE, 1) = 1

close(3) = 0

--- SIGCHLD {si\_signo=SIGCHLD, si\_code=CLD\_EXITED, si\_pid=66392, si\_uid=1000, si\_status=0, si\_utime=0, si\_stime=0} ---

wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 66392

munmap(0x7f0e9b828000, 32) = 0

munmap(0x7f0e9b827000, 32) = 0

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

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

munmap(0x7f0e9b862000, 512) = 0

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

exit\_group(0) = ?

+++ exited with 0 +++

# Вывод

В ходе лабораторной работе я приобрел базовые навыки по работе с разделяемой памятью в си. Я научился создавать объект разделяемой памяти, записывать в него данные и читать их из него. Также я узнал о работе с семафорами, научился использовать их для синхронизации при работе с разделяемой памятью. Помимо этого, я узнал о файловых системах и памяти в целом.