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: Жаворонков Н. Д.

Преподаватель: Бахарев В. Д.

Оценка: _____

Дата: 27.12.24

Москва
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.

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

Программа состоит из двух частей: родительской и дочерней. Родительский процесс открывает файл, читает из него числа и пересылает их через специальный канал на вход дочернему процессу. По завершении пересылки канал закрывается. Дочерний процесс считывает данные из потока ввода и определяет, составное ли поданное на вход число. Если число не составное – программа завершается. Сверка чисел происходит пока не будут прочитаны все числа на входе.

Код программы

client.c

```
#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;
}
```

server.c

```
#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

Тестирование:

supertos@DESKTOP-77RBNCB:~/oslab$ ./server.out tests/a.txt

18

36

96

48

7

55

supertos@DESKTOP-77RBNCB:~/oslab$ **Strace**

```
execve("./server.out", ["./server.out", "tests/a.txt"], 0x7ffe7644f1f8 /* 23 vars */) = 0
brk(NULL)                               = 0x55962d019000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffce59d3e80) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f668ee48000
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) = 0x7f668ee43000
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"..., 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) = 0x7f668ed5c000
mmap(0x7f668ed6a000, 507904, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0xe000) = 0x7f668ed6a000
mmap(0x7f668ede6000, 372736, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x8a000) =
0x7f668ede6000
mmap(0x7f668ee41000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0xe4000) = 0x7f668ee41000
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) = 0x7f668eb33000
mprotect(0x7f668eb5b000, 2023424, PROT_NONE) = 0
mmap(0x7f668eb5b000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x7f668eb5b000
mmap(0x7f668ecf0000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) =
0x7f668ecf0000
mmap(0x7f668ed49000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x215000) = 0x7f668ed49000
mmap(0x7f668ed4f000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -
1, 0) = 0x7f668ed4f000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f668eb30000
arch_prctl(ARCH_SET_FS, 0x7f668eb30740) = 0
set_tid_address(0x7f668eb30a10)         = 1009
set_robust_list(0x7f668eb30a20, 24)     = 0
rseq(0x7f668eb310e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7f668ed49000, 16384, PROT_READ) = 0
mprotect(0x7f668ee41000, 4096, PROT_READ) = 0
mprotect(0x559627d1c000, 4096, PROT_READ) = 0
```

```
mprotect(0x7f668ee82000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f668ee43000, 17571)           = 0
getrandom("\xd3\x3b\x93\xc2\x35\xdc\xb2\xef", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x55962d019000
brk(0x55962d03a000)                     = 0x55962d03a000
readlink("/proc/self/exe", "/home/supertos/oslabb/lab3/serve"..., 512) = 37
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) = 0x7f668ee81000
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("\x22\x8c\xc4\xbf\x6b\x6f\xcf\xd0", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.0lbSOL", 0x7ffce59d3960, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No
such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.0lbSOL", 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) = 0x7f668ee47000
link("/dev/shm/sem.0lbSOL", "/dev/shm/sem.semA") = 0
newfstatat(4, "", {st_mode=S_IFREG|0600, st_size=32, ...}, AT_EMPTY_PATH) = 0
unlink("/dev/shm/sem.0lbSOL")           = 0
close(4)                                = 0
getrandom("\x64\x44\xb6\x1e\x6e\x34\x23\x0c", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.0wR4vF", 0x7ffce59d3960, AT_SYMLINK_NOFOLLOW) = -1 ENOENT (No
such file or directory)
openat(AT_FDCWD, "/dev/shm/sem.0wR4vF", 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) = 0x7f668ee46000
link("/dev/shm/sem.0wR4vF", "/dev/shm/sem.semB") = 0
newfstatat(4, "", {st_mode=S_IFREG|0600, st_size=32, ...}, AT_EMPTY_PATH) = 0
unlink("/dev/shm/sem.0wR4vF")           = 0
close(4)                                = 0
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f668eb30a10) = 1010
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(0x7f668ee47000, 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(0x7f668ee47000, 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(0x7f668ee47000, 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(0x7f668ee47000, 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(0x7f668ee47000, 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(0x7f668ee47000, FUTEX_WAKE, 1)    = 1
close(3)                                = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=1010, si_uid=1000, si_status=0, si_utime=0,
si_stime=0} ---
wait4(-1, [{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 1010
munmap(0x7f668ee47000, 32)              = 0
munmap(0x7f668ee46000, 32)              = 0
unlink("/dev/shm/sem.semA")             = 0
unlink("/dev/shm/sem.semB")             = 0
munmap(0x7f668ee81000, 512)             = 0
unlink("/dev/shm/memA")                 = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

В ходе выполнения лабораторной работы была реализована программа, которая считывает числа из файла, создает дочерние процессы для проверки каждого числа на простоту и выводит составные числа в стандартный поток вывода. Программа успешно выполняет поставленную задачу и демонстрирует применение межпроцессного взаимодействия с использованием системного вызова fork().

В процессе реализации возникла трудность с корректным завершением родительского процесса после завершения всех дочерних процессов. Необходимо было использовать системный вызов wait(), чтобы родительский процесс ожидал завершения каждого дочернего процесса, прежде чем завершиться сам.

В целом, лабораторная работа была интересной и позволила закрепить знания о работе с системными вызовами и межпроцессном взаимодействии. Желательно было бы добавить возможность работы с большим количеством чисел и более эффективных алгоритмов проверки на простоту.