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-214Б-23

Студент: Заворотный А.А.

Преподаватель: Бахарев В.Д. (ФИИТ)

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

Дата: 19.10.24

Москва, 2024

**Постановка задачи Вариант 5.**

**Пользователь вводит команды вида: «число<endline>». Далее это число передается от родительского процесса в дочерний. Дочерний процесс производит проверку на простоту. Если число составное, то в это число записывается в файл. Если число отрицательное или простое, то тогда дочерний процесс завершается.**

**Общий метод и алгоритм решения**

Использованные системные вызовы:

1. \*\*`pid\_t fork(void);`\*\* – создает дочерний процесс.

2. \*\*`ssize\_t readlink(const char \*path, char \*buf, size\_t bufsiz);`\*\* – читает символическую ссылку.

3. \*\*`ssize\_t write(int fd, const void \*buf, size\_t count);`\*\* – записывает данные в файл.

4. \*\*`ssize\_t read(int fd, void \*buf, size\_t count);`\*\* – читает данные из файла.

5. \*\*`int open(const char \*path, int oflag, ...);`\*\* – открывает файл.

6. \*\*`int close(int fd);`\*\* – закрывает файл.

7. \*\*`int shm\_open(const char \*name, int oflag, mode\_t mode);`\*\* – создает или открывает объект

разделяемой памяти.

8. \*\*`int shm\_unlink(const char \*name);`\*\* – удаляет объект разделяемой памяти.

9. \*\*`int ftruncate(int fd, off\_t length);`\*\* – устанавливает размер файла.

10. \*\*`void \*mmap(void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset);`\*\* – отображает

файл или устройство в память.

11. \*\*`int munmap(void \*addr, size\_t length);`\*\* – отменяет отображение файла или устройства в

память.

12. \*\*`sem\_t \*sem\_open(const char \*name, int oflag, mode\_t mode, unsigned int value);`\*\* – создает

или открывает семафор.

13. \*\*`int sem\_close(sem\_t \*sem);`\*\* – закрывает семафор.

14. \*\*`int sem\_unlink(const char \*name);`\*\* – удаляет семафор.

15. \*\*`int sem\_wait(sem\_t \*sem);`\*\* – ожидает, пока значение семафора не станет больше нуля, и

уменьшает его.

16. \*\*`int sem\_post(sem\_t \*sem);`\*\* – увеличивает значение семафора.

17. \*\*`pid\_t wait(int \*wstatus);`\*\* – ожидает завершения дочернего процесса.

18. \*\*`int execv(const char \*path, char \*const argv[]);`\*\* – заменяет текущий процесс новым

процессом.

19. \*\*`void exit(int status);`\*\* – завершает выполнение программы.

Эти системные вызовы используются для управления процессами, файлами, разделяемой памятью и синхронизации между процессами с помощью семафоров.

**Код программы**

**server.c**

#include <stdio.h>

#include <stdlib.h>

#include <stdint.h>

#include <stdbool.h>

#include <string.h>

#include <unistd.h>

#include <sys/wait.h>

#include <sys/stat.h>

#include <sys/mman.h>

#include <fcntl.h>

#include <semaphore.h>

#define SHM\_NAME "/Data"

#define SEM\_PARENT "/parent\_semaphore"

#define SEM\_CHILD "/child\_semaphore"

**static** **char** CLIENT\_PROGRAM\_NAME[] = "client";

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

**if** (argc == 1) {

fprintf(stderr, "Usage: %s filename\n", argv[0]);

exit(EXIT\_SUCCESS);

}

**char** progpath[1024];

ssize\_t len = readlink("/proc/self/exe", progpath, **sizeof**(progpath) - 1);

**if** (len == -1) {

perror("readlink");

exit(EXIT\_FAILURE);

}

**while** (progpath[len] != '/') {

--len;

}

progpath[len] = '\0';

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

**if** (fd == -1) {

perror("shm\_open");

exit(EXIT\_FAILURE);

}

**if** (ftruncate(fd, BUFSIZ) == -1) {

perror("ftruncate");

close(fd);

exit(EXIT\_FAILURE);

}

**char** \*ptr = (**char** \*)mmap(NULL, BUFSIZ, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

**if** (ptr == MAP\_FAILED) {

perror("mmap");

close(fd);

exit(EXIT\_FAILURE);

}

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** (sem\_parent == SEM\_FAILED || sem\_child == SEM\_FAILED) {

perror("sem\_open");

close(fd);

exit(EXIT\_FAILURE);

}

pid\_t child = fork();

**if** (child == -1) {

perror("fork");

exit(EXIT\_FAILURE);

}

**switch** (child) {

**case** -1: {

**const** **char** msg[] = "error: failed to spawn new process\n";

write(STDERR\_FILENO, msg, **sizeof**(msg));

exit(EXIT\_FAILURE);

}

**case** 0: {

pid\_t pid = getpid();

printf("%d: I'm a child\n", pid);

**char** path[1024];

snprintf(path, **sizeof**(path) - 1, "%s/%s", progpath, CLIENT\_PROGRAM\_NAME);

execl(path, CLIENT\_PROGRAM\_NAME, argv[1], NULL);

perror("execl");

exit(EXIT\_FAILURE);

}

**default**: {

pid\_t pid = getpid();

printf("%d: I'm a parent, my child has PID %d\n", pid, child);

**char** buf[BUFSIZ];

ssize\_t bytes;

**while** ((bytes = read(STDIN\_FILENO, buf, **sizeof**(buf) - 1)) > 0) {

buf[strcspn(buf, "\n")] = '\0';

strncpy(ptr, buf, BUFSIZ);

sem\_post(sem\_child);

sem\_wait(sem\_parent);

}

strncpy(ptr, "", BUFSIZ);

sem\_post(sem\_child);

wait(NULL);

munmap(ptr, BUFSIZ);

shm\_unlink(SHM\_NAME);

sem\_close(sem\_parent);

sem\_close(sem\_child);

sem\_unlink(SEM\_PARENT);

sem\_unlink(SEM\_CHILD);

close(fd);

}

}

**return** 0;

}

**client.c**

#include <stdio.h>

#include <string.h>

#include <stdint.h>

#include <stdbool.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <ctype.h>

#include <sys/mman.h>

#include <semaphore.h>

#define SHM\_NAME "/Data"

#define SEM\_PARENT "/parent\_semaphore"

#define SEM\_CHILD "/child\_semaphore"

**int** isPrime(**int** N) {

**if** (N == 1) {

**return** 0;

}

**if** (N == 2) {

**return** 1;

}

**if** (N % 2 == 0){

**return** 0;

}

**for** (**int** i = 3; i \* i <= N; i++) {

**if** (N % i == 0) {

**return** 0;

}

i++;

}

**return** 1;

}

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

**if** (argc != 2) {

fprintf(stderr, "Usage: %s output\_file\n", argv[0]);

**return** EXIT\_FAILURE;

}

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

**if** (fd == -1) {

perror("shm\_open");

**return** EXIT\_FAILURE;

}

**char** \*ptr = (**char** \*)mmap(NULL, BUFSIZ, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

**if** (ptr == MAP\_FAILED) {

perror("mmap");

close(fd);

**return** EXIT\_FAILURE;

}

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** (sem\_parent == SEM\_FAILED || sem\_child == SEM\_FAILED) {

perror("sem\_open");

munmap(ptr, BUFSIZ);

close(fd);

**return** EXIT\_FAILURE;

}

pid\_t pid = getpid();

int32\_t file = open(argv[1], O\_WRONLY | O\_CREAT | O\_TRUNC, 0600);

**if** (file == -1) {

perror("open");

sem\_close(sem\_parent);

sem\_close(sem\_child);

munmap(ptr, BUFSIZ);

close(fd);

**return** EXIT\_FAILURE;

}

{

**char** msg[128];

int32\_t len = snprintf(msg, **sizeof**(msg) - 1,

"%d: Start typing lines of text. Press 'Ctrl-D' with no input to exit\n", pid);

write(STDOUT\_FILENO, msg, len);

}

**char** buf[BUFSIZ];

ssize\_t bytes;

**char** ans[BUFSIZ];

**while** (1) {

sem\_wait(sem\_child);

strncpy(buf, ptr, BUFSIZ);

**if** (strlen(buf) == 0) {

**break**;

}

**if** (!isPrime(atoi(buf))) {

snprintf(ans, **sizeof**(ans), "%s\n", buf);

write(file, ans, strlen(ans));

} **else** {

**if** (strlen(buf)) {

snprintf(ans, **sizeof**(ans), "Prime number or invalid str: %s\n", buf);

write(STDOUT\_FILENO, ans, strlen(ans));

sem\_post(sem\_parent);

**break**;

}

}

sem\_post(sem\_parent);

}

munmap(ptr, BUFSIZ);

close(fd);

sem\_close(sem\_parent);

sem\_close(sem\_child);

close(file);

}

**Протокол работы программы**

Тестирование:

$ ./server file\_test.txt

1031: I'm a parent, my child has PID 1032

1032: I'm a child

1032: Start typing lines of text. Press 'Ctrl-D' with no input to exit

2335

234

45

45

23

Prime number or invalid str: 23

Strace:

$ strace -f ./server file\_test.txt

execve("./server", ["./server", "file\_test.txt"], 0x7ffc38abf770 /\* 27 vars \*/) = 0

brk(NULL) = 0x563ef719a000

mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f1f96cae000

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=20071, ...}) = 0

mmap(NULL, 20071, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f1f96ca9000

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) = 0x7f1f96a97000

mmap(0x7f1f96abf000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f1f96abf000

mmap(0x7f1f96c47000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) = 0x7f1f96c47000

mmap(0x7f1f96c96000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x7f1f96c96000

mmap(0x7f1f96c9c000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f1f96c9c000

close(3) = 0

mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f1f96a94000

arch\_prctl(ARCH\_SET\_FS, 0x7f1f96a94740) = 0

set\_tid\_address(0x7f1f96a94a10) = 2298

set\_robust\_list(0x7f1f96a94a20, 24) = 0

rseq(0x7f1f96a95060, 0x20, 0, 0x53053053) = 0

mprotect(0x7f1f96c96000, 16384, PROT\_READ) = 0

mprotect(0x563eb9d81000, 4096, PROT\_READ) = 0

mprotect(0x7f1f96ce6000, 8192, PROT\_READ) = 0

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

munmap(0x7f1f96ca9000, 20071) = 0

readlink("/proc/self/exe", "/mnt/c/Users/\320\220\320\273\320\265\320\272\321\201\320\265\320\271/CLio"..., 1023) = 87

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

**ftruncate(3, 8192) = 0**

**mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7f1f96cac000**

**openat(AT\_FDCWD, "/dev/shm/sem.parent\_semaphore", O\_RDWR|O\_NOFOLLOW|O\_CLOEXEC) = -1 ENOENT (No such file or directory)**

getrandom("\xfa\x4e\x03\xc4\xb2\x14\xb5\x84", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.mrOAKe", 0x7ffdba1c00c0, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (No such file or directory)

**openat(AT\_FDCWD, "/dev/shm/sem.mrOAKe", O\_RDWR|O\_CREAT|O\_EXCL|O\_NOFOLLOW|O\_CLOEXEC, 0666) = 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) = 0x7f1f96cab000**

link("/dev/shm/sem.mrOAKe", "/dev/shm/sem.parent\_semaphore") = 0

fstat(4, {st\_mode=S\_IFREG|0644, st\_size=32, ...}) = 0

getrandom("\xde\x81\x89\x0c\x70\xfe\x67\xee", 8, GRND\_NONBLOCK) = 8

brk(NULL) = 0x563ef719a000

brk(0x563ef71bb000) = 0x563ef71bb000

**unlink("/dev/shm/sem.mrOAKe") = 0**

close(4) = 0

openat(AT\_FDCWD, "/dev/shm/sem.child\_semaphore", O\_RDWR|O\_NOFOLLOW|O\_CLOEXEC) = -1 ENOENT (No such file or directory)

getrandom("\xdc\xb5\x6f\xfd\xdd\x41\xa2\x2f", 8, GRND\_NONBLOCK) = 8

newfstatat(AT\_FDCWD, "/dev/shm/sem.a4vJxJ", 0x7ffdba1c00c0, AT\_SYMLINK\_NOFOLLOW) = -1 ENOENT (No such file or directory)

openat(AT\_FDCWD, "/dev/shm/sem.a4vJxJ", O\_RDWR|O\_CREAT|O\_EXCL|O\_NOFOLLOW|O\_CLOEXEC, 0666) = 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) = 0x7f1f96caa000

link("/dev/shm/sem.a4vJxJ", "/dev/shm/sem.child\_semaphore") = 0

fstat(4, {st\_mode=S\_IFREG|0644, st\_size=32, ...}) = 0

unlink("/dev/shm/sem.a4vJxJ") = 0

close(4) = 0

clone(child\_stack=NULL, flags=CLONE\_CHILD\_CLEARTID|CLONE\_CHILD\_SETTID|SIGCHLDstrace: Process 2299 attached

, child\_tidptr=0x7f1f96a94a10) = 2299

[pid 2299] set\_robust\_list(0x7f1f96a94a20, 24 <unfinished ...>

[pid 2298] getpid( <unfinished ...>

[pid 2299] <... set\_robust\_list resumed>) = 0

[pid 2298] <... getpid resumed>) = 2298

[pid 2299] getpid( <unfinished ...>

[pid 2298] fstat(1, <unfinished ...>

[pid 2299] <... getpid resumed>) = 2299

[pid 2298] <... fstat resumed>{st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

[pid 2299] fstat(1, <unfinished ...>

[pid 2298] write(1, "2298: I'm a parent, my child has"..., 42 <unfinished ...>

2298: I'm a parent, my child has PID 2299

[pid 2299] <... fstat resumed>{st\_mode=S\_IFCHR|0620, st\_rdev=makedev(0x88, 0), ...}) = 0

[pid 2298] <... write resumed>) = 42

[pid 2299] write(1, "2299: I'm a child\n", 18 <unfinished ...>

2299: I'm a child

[pid 2298] read(0, <unfinished ...>

[pid 2299] <... write resumed>) = 18

[pid 2299] execve("/mnt/c/Users/\320\220\320\273\320\265\320\272\321\201\320\265\320\271/CLionProjects/Labs\_osi\_Zavorotnyi\_A\_M8O-214--23/Lab3/client", ["client", "file\_test.txt"], 0x7ffdba1c29e0 /\* 27 vars \*/) = 0

[pid 2299] brk(NULL) = 0x55b780814000

[pid 2299] mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f4085f68000

[pid 2299] access("/etc/ld.so.preload", R\_OK) = -1 ENOENT (No such file or directory)

[pid 2299] openat(AT\_FDCWD, "/etc/ld.so.cache", O\_RDONLY|O\_CLOEXEC) = 3

[pid 2299] fstat(3, {st\_mode=S\_IFREG|0644, st\_size=20071, ...}) = 0

[pid 2299] mmap(NULL, 20071, PROT\_READ, MAP\_PRIVATE, 3, 0) = 0x7f4085f63000

[pid 2299] close(3) = 0

[pid 2299] openat(AT\_FDCWD, "/lib/x86\_64-linux-gnu/libc.so.6", O\_RDONLY|O\_CLOEXEC) = 3

[pid 2299] 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

[pid 2299] 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

[pid 2299] fstat(3, {st\_mode=S\_IFREG|0755, st\_size=2125328, ...}) = 0

[pid 2299] 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

[pid 2299] mmap(NULL, 2170256, PROT\_READ, MAP\_PRIVATE|MAP\_DENYWRITE, 3, 0) = 0x7f4085d51000

[pid 2299] mmap(0x7f4085d79000, 1605632, PROT\_READ|PROT\_EXEC, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x28000) = 0x7f4085d79000

[pid 2299] mmap(0x7f4085f01000, 323584, PROT\_READ, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1b0000) = 0x7f4085f01000

[pid 2299] mmap(0x7f4085f50000, 24576, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_DENYWRITE, 3, 0x1fe000) = 0x7f4085f50000

[pid 2299] mmap(0x7f4085f56000, 52624, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_FIXED|MAP\_ANONYMOUS, -1, 0) = 0x7f4085f56000

[pid 2299] close(3) = 0

[pid 2299] mmap(NULL, 12288, PROT\_READ|PROT\_WRITE, MAP\_PRIVATE|MAP\_ANONYMOUS, -1, 0) = 0x7f4085d4e000

[pid 2299] arch\_prctl(ARCH\_SET\_FS, 0x7f4085d4e740) = 0

[pid 2299] set\_tid\_address(0x7f4085d4ea10) = 2299

[pid 2299] set\_robust\_list(0x7f4085d4ea20, 24) = 0

[pid 2299] rseq(0x7f4085d4f060, 0x20, 0, 0x53053053) = 0

[pid 2299] mprotect(0x7f4085f50000, 16384, PROT\_READ) = 0

[pid 2299] mprotect(0x55b779d18000, 4096, PROT\_READ) = 0

[pid 2299] mprotect(0x7f4085fa0000, 8192, PROT\_READ) = 0

[pid 2299] prlimit64(0, RLIMIT\_STACK, NULL, {rlim\_cur=8192\*1024, rlim\_max=RLIM64\_INFINITY}) = 0

[pid 2299] munmap(0x7f4085f63000, 20071) = 0

**[pid 2299] openat(AT\_FDCWD, "/dev/shm/Data", O\_RDWR|O\_NOFOLLOW|O\_CLOEXEC) = 3**

**[pid 2299] mmap(NULL, 8192, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 3, 0) = 0x7f4085f66000**

**[pid 2299] openat(AT\_FDCWD, "/dev/shm/sem.parent\_semaphore", O\_RDWR|O\_NOFOLLOW|O\_CLOEXEC) = 4**

[pid 2299] fstat(4, {st\_mode=S\_IFREG|0644, st\_size=32, ...}) = 0

[pid 2299] getrandom("\x98\x58\x7e\x68\x46\x7c\x93\x63", 8, GRND\_NONBLOCK) = 8

[pid 2299] brk(NULL) = 0x55b780814000

[pid 2299] brk(0x55b780835000) = 0x55b780835000

[pid 2299] mmap(NULL, 32, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 4, 0) = 0x7f4085f65000

[pid 2299] close(4) = 0

**[pid 2299] openat(AT\_FDCWD, "/dev/shm/sem.child\_semaphore", O\_RDWR|O\_NOFOLLOW|O\_CLOEXEC) = 4**

**[pid 2299] fstat(4, {st\_mode=S\_IFREG|0644, st\_size=32, ...}) = 0**

**[pid 2299] mmap(NULL, 32, PROT\_READ|PROT\_WRITE, MAP\_SHARED, 4, 0) = 0x7f4085f64000**

[pid 2299] close(4) = 0

[pid 2299] getpid() = 2299

[pid 2299] openat(AT\_FDCWD, "file\_test.txt", O\_WRONLY|O\_CREAT|O\_TRUNC, 0600) = 4

[pid 2299] write(1, "2299: Start typing lines of text"..., 712299: Start typing lines of text. Press 'Ctrl-D' with no input to exit

) = 71

[pid 2299] futex(0x7f4085f64000, FUTEX\_WAIT\_BITSET|FUTEX\_CLOCK\_REALTIME, 0, NULL, FUTEX\_BITSET\_MATCH\_ANY

**Вывод**

**Программа демонстрирует взаимодействие между родительским и дочерним процессами с использованием разделяемой памяти и семафоров для синхронизации. Родительский процесс считывает строки от пользователя и передает их дочернему процессу через разделяемую память. Дочерний процесс обрабатывает строки и записывает их в файл, а также выводит соответствующие сообщения на экран.**