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: Маркелов Ярослав

Преподаватель: Бахарев В.Д. (ФИИТ)

Оценка: _____

Дата: 07.12.24

Москва, 2024

Постановка задачи

Вариант 15.

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. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child проверяет строки на валидность правилу. Если строка соответствует правилу, то она выводится в стандартный поток вывода дочернего процесса, иначе в pipe2 выводится информация об ошибке. Родительский процесс полученные от child ошибки выводит в стандартный поток вывода.

Вариант 15) Правило проверки: строка должна начинаться с заглавной буквы

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.

Далее описываете то, что вы делали в рамках лабораторной работы, а также то, как работает ваша программа и т.д..

Код программы

child.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <ctype.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <semaphore.h>

#define BUFFER_SIZE 1024
#define SHM_NAME "/my_shared_memory"
#define SEM_NAME_PARENT "/my_semaphore_parent"
#define SEM_NAME_CHILD "/my_semaphore_child"

int main(int argc, char *argv[]) {
    // Открытие разделяемой памяти
    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0666);
    if (shm_fd == -1) {
        write(STDOUT_FILENO, "Не удалось открыть разделяемую память", sizeof("Не удалось открыть разделяемую память"));
        exit(EXIT_FAILURE);
    }

    // Отображение разделяемой памяти
    char *shared_memory = (char *)mmap(0, BUFFER_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (shared_memory == MAP_FAILED) {
        write(STDOUT_FILENO, "Не удалось отобразить разделяемую память", sizeof("Не удалось отобразить разделяемую память"));
        exit(EXIT_FAILURE);
    }

    // Открытие семафоров
    sem_t *sem_parent = sem_open(SEM_NAME_PARENT, 0);
    sem_t *sem_child = sem_open(SEM_NAME_CHILD, 0);
    if (sem_parent == SEM_FAILED || sem_child == SEM_FAILED) {
        write(STDOUT_FILENO, "Не удалось открыть семафоры", sizeof("Не удалось открыть семафоры"));
        exit(EXIT_FAILURE);
    }

    int file_descriptor = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
    if (file_descriptor == -1) {
```

```
        write(STDOUT_FILENO, "Не удалось открыть файл", sizeof("Не удалось открыть файл"));
        exit(EXIT_FAILURE);
    }

    char input[BUFFER_SIZE];
    char error_msg[BUFFER_SIZE];

    while (1) {
        // Ожидание сигнала от родительского процесса
        sem_wait(sem_child);

        // Чтение из разделяемой памяти
        strncpy(input, shared_memory, BUFFER_SIZE);

        if (strlen(input) == 0) {
            break; // Конец ввода
        }

        if (isupper(input[0])) {
            char tmp[1100];
            int ret = snprintf(tmp, sizeof(tmp), "Валидная строка: %s\n", input);
            if (ret < 0) {
                abort();
            }
            write(STDOUT_FILENO, tmp, strlen(tmp));
            write(file_descriptor, tmp, strlen(tmp));
        } else {
            if (strlen(input)) {
                int ret = snprintf(error_msg, sizeof(error_msg), "Не валидная строка: %s\n", input);
                if (ret < 0) {
                    abort();
                }
                write(STDOUT_FILENO, error_msg, strlen(error_msg));
                write(file_descriptor, error_msg, strlen(error_msg));
            }
        }

        // Сигнализируем родительскому процессу
        sem_post(sem_parent);
    }

    // Освобождение ресурсов
    munmap(shared_memory, BUFFER_SIZE);
    close(shm_fd);
    sem_close(sem_parent);
    sem_close(sem_child);
    close(file_descriptor);
    return 0;
}
```

parent.c

```c
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <sys/mman.h>
#include <semaphore.h>
#include <stdio.h>

#define BUFFER_SIZE 1024
#define SHM_NAME "/my_shared_memory"
#define SEM_NAME_PARENT "/my_semaphore_parent"
#define SEM_NAME_CHILD "/my_semaphore_child"
static char CLIENT_PROGRAM_NAME[] = "child";

void write_string(int fd, const char *str) {
    write(fd, str, strlen(str));
}

void read_string(int fd, char *buffer, size_t size) {
    read(fd, buffer, size);
}

int main() {
    char filename[BUFFER_SIZE];
    char progpath[1024];

    ssize_t len = readlink("/proc/self/exe", progpath, sizeof(progpath) - 1);
    if (len == -1) {
        const char msg[] = "error: failed to read full program path\n";
        write(STDERR_FILENO, msg, sizeof(msg));
        exit(EXIT_FAILURE);
    }

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

    progpath[len] = '\0';

    const char *prompt = "Введите имя файла: ";
    write_string(STDOUT_FILENO, prompt);
    read_string(STDIN_FILENO, filename, sizeof(filename));
    filename[strcspn(filename, "\n")] = '\0'; // Удаляем символ новой строки

//     int file_descriptor = open(filename, O_WRONLY | O_CREAT | O_TRUNC,
S_IRUSR | S_IWUSR);
//     if (file_descriptor == -1) {
//         write_string(STDOUT_FILENO, "Не удалось открыть файл");
//         exit(EXIT_FAILURE);
```

```
//     }

    // Создание разделяемой памяти
    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        write_string(STDOUT_FILENO, "Не удалось создать разделяемую память");
        exit(EXIT_FAILURE);
    }
    ftruncate(shm_fd, BUFFER_SIZE);

    // Отображение разделяемой памяти
    char *shared_memory = (char *)mmap(0, BUFFER_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);
    if (shared_memory == MAP_FAILED) {
        write_string(STDOUT_FILENO, "Не удалось отобразить разделяемую память");
        exit(EXIT_FAILURE);
    }

    // Создание семафоров
    sem_t *sem_parent = sem_open(SEM_NAME_PARENT, O_CREAT, 0666, 0);
    sem_t *sem_child = sem_open(SEM_NAME_CHILD, O_CREAT, 0666, 0);
    if (sem_parent == SEM_FAILED || sem_child == SEM_FAILED) {
        write_string(STDOUT_FILENO, "Не удалось создать семафоры");
        exit(EXIT_FAILURE);
    }

    pid_t child_pid = fork();
    if (child_pid == -1) {
        write_string(STDOUT_FILENO, "Не удалось создать процесс");
        exit(EXIT_FAILURE);
    }

    if (child_pid == 0) {
        char path[2024];
        snprintf(path, sizeof(path) - 1, "%s/%s", progpath, CLIENT_PROGRAM_NAME);

        char *const args[] = {CLIENT_PROGRAM_NAME, filename, NULL};

        int32_t status = execv(path, args);

        if (status == -1) {
            const char msg[] = "error: failed to exec into new executable
image\n";
            write(STDERR_FILENO, msg, sizeof(msg));
            exit(EXIT_FAILURE);
        }
//         // Дочерний процесс
//         char *args[] = {"./child", NULL};
//         execvp(args[0], args);
//         write_string(STDOUT_FILENO, "Не удалось запустить дочерний процесс");
//         exit(EXIT_FAILURE);
```

```
    } else {
        // Родительский процесс
        char input[BUFFER_SIZE];
        const char *input_prompt = "Введите строки (CTRL+D для завершения):\n";
        write_string(STDOUT_FILENO, input_prompt);

        while (read(STDIN_FILENO, input, sizeof(input)) > 0) {
            input[strcspn(input, "\n")] = '\0'; // Удаляем символ новой строки

            // Запись в разделяемую память
            strncpy(shared_memory, input, BUFFER_SIZE);

            // Сигнализируем дочернему процессу
            sem_post(sem_child);

            // Ожидаем подтверждения от дочернего процесса
            sem_wait(sem_parent);
        }

        // Сигнализируем о завершении ввода
        strncpy(shared_memory, "", BUFFER_SIZE);
        sem_post(sem_child);

        // Ожидание завершения дочернего процесса
        wait(NULL);

        // Освобождение ресурсов
        munmap(shared_memory, BUFFER_SIZE);
        shm_unlink(SHM_NAME);
        sem_close(sem_parent);
        sem_close(sem_child);
        sem_unlink(SEM_NAME_PARENT);
        sem_unlink(SEM_NAME_CHILD);
        //close(file_descriptor);
    }

    return 0;
}
```

Протокол работы программы

(родительский процесс)
Чтение пути к текущему исполняемому файлу:

Используется системный вызов readlink для получения пути к текущему исполняемому файлу.

Путь сохраняется в переменной progpath.

Ввод имени файла:

Программа запрашивает у пользователя имя файла, в который будут записываться строки.

Имя файла считывается с помощью `read_string` и сохраняется в переменной `filename`.

Создание разделяемой памяти:

Используется системный вызов `shm_open` для создания объекта разделяемой памяти с именем `SHM_NAME`.

Размер разделяемой памяти устанавливается с помощью `ftruncate`.

Память отображается в адресное пространство процесса с помощью `mmap`.

Создание семафоров:

Используется `sem_open` для создания двух семафоров: `SEM_NAME_PARENT` и `SEM_NAME_CHILD`.

Эти семафоры будут использоваться для синхронизации между родительским и дочерним процессами.

Создание дочернего процесса:

Используется `fork` для создания дочернего процесса.

Если процесс является дочерним (`child_pid == 0`), то выполняется запуск другой программы (дочернего процесса) с помощью `execv`.

Если процесс является родительским, то продолжается выполнение основной логики.

Ввод строк и запись в разделяемую память:

Родительский процесс запрашивает у пользователя строки для записи в файл.

Введенные строки записываются в разделяемую память с помощью `strncpy`.

После записи строки в разделяемую память родительский процесс сигнализирует дочернему процессу с помощью `sem_post`.

Родительский процесс ожидает подтверждения от дочернего процесса с помощью `sem_wait`.

Завершение ввода и ожидание дочернего процесса:

Когда пользователь завершает ввод (например, нажимает `CTRL+D`), родительский процесс записывает пустую строку в разделяемую память и сигнализирует дочернему процессу.

Родительский процесс ожидает завершения дочернего процесса с помощью `wait`.

Освобождение ресурсов:

После завершения работы дочернего процесса родительский процесс освобождает разделяемую память с помощью `munmap` и `shm_unlink`.

Семафоры закрываются с помощью `sem_close` и удаляются с помощью `sem_unlink`.

Дочерний процесс
Открытие разделяемой памяти:

Дочерний процесс открывает разделяемую память с помощью `shm_open`.

Память отображается в адресное пространство процесса с помощью `mmap`.

Открытие семафоров:

Дочерний процесс открывает семафоры `SEM_NAME_PARENT` и `SEM_NAME_CHILD` с помощью `sem_open`.

Открытие файла для записи:

Дочерний процесс открывает файл, указанный пользователем, с помощью `open`.

Чтение строк из разделяемой памяти и запись в файл:

Дочерний процесс ожидает сигнала от родительского процесса с помощью `sem_wait`.

После получения сигнала дочерний процесс читает строку из разделяемой памяти с помощью `strncpy`.

Если строка пустая, это означает конец ввода, и дочерний процесс завершает работу.

Если строка начинается с заглавной буквы, она считается валидной и записывается в файл и выводится на экран.

Если строка начинается со строчной буквы, она считается невалидной и записывается в файл и выводится на экран с соответствующим сообщением.

После обработки строки дочерний процесс сигнализирует родительскому процессу с помощью `sem_post`.

Освобождение ресурсов:

После завершения работы дочерний процесс освобождает разделяемую память с помощью `munmap` и закрывает файл с помощью `close`.

Семафоры закрываются с помощью `sem_close`.

Тестирование

```
$ ./parent
Введите имя файла: 123.txt
Введите строки (CTRL+D для завершения):
123456
Не валидная строка: 123456
12345
```

```
Не валидная строка: 12345
QWert
Валидная строка: QWert
qwe
Не валидная строка: qwe
```

```
strace
$ strace -f ./parent
execve("./parent", ["./parent"], 0x7ffd6c1ed988 /* 65 vars */) = 0
brk(NULL)                               = 0x55a98433e000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffea54ad890) = -1 EINVAL (Недопустимый аргумент)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7f2071f97000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=82523, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 82523, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f2071f82000
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"..., 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) =
0x7f2071d59000
mprotect(0x7f2071d81000, 2023424, PROT_NONE) = 0
mmap(0x7f2071d81000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f2071d81000
mmap(0x7f2071f16000, 360448, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7f2071f16000
mmap(0x7f2071f6f000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f2071f6f000
mmap(0x7f2071f75000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f2071f75000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1,
0) = 0x7f2071d56000
arch_prctl(ARCH_SET_FS, 0x7f2071d56740) = 0
set_tid_address(0x7f2071d56a10)         = 6037
set_robust_list(0x7f2071d56a20, 24)     = 0
rseq(0x7f2071d570e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7f2071f6f000, 16384, PROT_READ) = 0
mprotect(0x55a983388000, 4096, PROT_READ) = 0
mprotect(0x7f2071fd1000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) =
0
munmap(0x7f2071f82000, 82523)           = 0
```

```
readlink("/proc/self/exe", "/home/myar/CLionProjects/OS_Labs"..., 1023) = 39
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \320\270\320\274\321\217
\321\204\320\260\320\271\320\273\320\260"..., 34Введите имя файла: ) = 34
read(0, Qwerty
"Qwerty\n", 1024)               = 7
openat(AT_FDCWD, "/dev/shm/my_shared_memory",
O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 3
ftruncate(3, 1024)                      = 0
mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f2071fd0000
openat(AT_FDCWD, "/dev/shm/sem.my_semaphore_parent", O_RDWR|O_NOFOLLOW) = -1
ENOENT (Нет такого файла или каталога)
getrandom("\x0c\x68\x5d\xd3\xb9\x60\xef\x5c", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.whrqEj", 0x7ffea54ac550,
AT_SYMLINK_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/dev/shm/sem.whrqEj", O_RDWR|O_CREAT|O_EXCL, 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) = 0x7f2071f96000
link("/dev/shm/sem.whrqEj", "/dev/shm/sem.my_semaphore_parent") = 0
newfstatat(4, "", {st_mode=S_IFREG|0664, st_size=32, ...}, AT_EMPTY_PATH) = 0
getrandom("\x01\x18\xdb\x95\x78\x99\xb3\x8d", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x55a98433e000
brk(0x55a98435f000)                     = 0x55a98435f000
unlink("/dev/shm/sem.whrqEj")           = 0
close(4)                                = 0
openat(AT_FDCWD, "/dev/shm/sem.my_semaphore_child", O_RDWR|O_NOFOLLOW) = -1
ENOENT (Нет такого файла или каталога)
getrandom("\x01\x7d\xbf\x06\x5a\x3c\x1b\xbc", 8, GRND_NONBLOCK) = 8
newfstatat(AT_FDCWD, "/dev/shm/sem.lx7llH", 0x7ffea54ac550, AT_SYMLINK_NOFOLLOW)
= -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/dev/shm/sem.lx7llH", O_RDWR|O_CREAT|O_EXCL, 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) = 0x7f2071f95000
link("/dev/shm/sem.lx7llH", "/dev/shm/sem.my_semaphore_child") = 0
newfstatat(4, "", {st_mode=S_IFREG|0664, st_size=32, ...}, AT_EMPTY_PATH) = 0
unlink("/dev/shm/sem.lx7llH")           = 0
close(4)                                = 0
clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f2071d56a10) = 6039
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265
\321\201\321\202\321\200\320\276\320\272\320\270 (CTR"..., 66Введите строки (CTRL+D для
завершения):
strace: Process 6039 attached
) = 66
[pid  6037] read(0,  <unfinished ...>
[pid  6039] set_robust_list(0x7f2071d56a20, 24) = 0
[pid  6039] execve("/home/myar/CLionProjects/OS_Labs/child", ["child", "Qwerty"],
0x7ffea54ada68 /* 65 vars */) = 0
[pid  6039] brk(NULL)                   = 0x559df52a2000
[pid  6039] arch_prctl(0x3001 /* ARCH_??? */, 0x7ffe26229aa0) = -1 EINVAL (Недопустимый
аргумент)
[pid  6039] mmap(NULL, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f3e964d9000
```

```
[pid  6039] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или каталога)
[pid  6039] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
[pid  6039] newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=82523, ...}, AT_EMPTY_PATH) = 0
[pid  6039] mmap(NULL, 82523, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f3e964c4000
[pid  6039] close(3)                    = 0
[pid  6039] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
[pid  6039] 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
[pid  6039] 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  6039] 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
[pid  6039] 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
[pid  6039] newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
[pid  6039] 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  6039] mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f3e9629b000
[pid  6039] mprotect(0x7f3e962c3000, 2023424, PROT_NONE) = 0
[pid  6039] mmap(0x7f3e962c3000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f3e962c3000
[pid  6039] mmap(0x7f3e96458000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x7f3e96458000
[pid  6039] mmap(0x7f3e964b1000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7f3e964b1000
[pid  6039] mmap(0x7f3e964b7000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f3e964b7000
[pid  6039] close(3)                    = 0
[pid  6039] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f3e96298000
[pid  6039] arch_prctl(ARCH_SET_FS, 0x7f3e96298740) = 0
[pid  6039] set_tid_address(0x7f3e96298a10) = 6039
[pid  6039] set_robust_list(0x7f3e96298a20, 24) = 0
[pid  6039] rseq(0x7f3e962990e0, 0x20, 0, 0x53053053) = 0
[pid  6039] mprotect(0x7f3e964b1000, 16384, PROT_READ) = 0
[pid  6039] mprotect(0x559df3668000, 4096, PROT_READ) = 0
[pid  6039] mprotect(0x7f3e96513000, 8192, PROT_READ) = 0
[pid  6039] prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
[pid  6039] munmap(0x7f3e964c4000, 82523) = 0
[pid  6039] openat(AT_FDCWD, "/dev/shm/my_shared_memory", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 3
[pid  6039] mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f3e96512000
[pid  6039] openat(AT_FDCWD, "/dev/shm/sem.my_semaphore_parent", O_RDWR|O_NOFOLLOW) = 4
[pid  6039] newfstatat(4, "", {st_mode=S_IFREG|0664, st_size=32, ...}, AT_EMPTY_PATH) = 0
[pid  6039] getrandom("\x17\x36\xc3\x6c\x8b\x27\x97\x2b", 8, GRND_NONBLOCK) = 8
[pid  6039] brk(NULL)                   = 0x559df52a2000
[pid  6039] brk(0x559df52c3000)         = 0x559df52c3000
[pid  6039] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x7f3e964d8000
[pid  6039] close(4)                    = 0
```

```
[pid  6039] openat(AT_FDCWD, "/dev/shm/sem.my_semaphore_child",
O_RDWR|O_NOFOLLOW) = 4
[pid  6039] newfstatat(4, "", {st_mode=S_IFREG|0664, st_size=32, ...}, AT_EMPTY_PATH) = 0
[pid  6039] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) =
0x7f3e964d7000
[pid  6039] close(4)                    = 0
[pid  6039] openat(AT_FDCWD, "Qwerty", O_WRONLY|O_CREAT|O_TRUNC, 0600) = 4
[pid  6039] futex(0x7f3e964d7000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0,
NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>
[pid  6037] <... read resumed>"", 1024) = 0
[pid  6037] futex(0x7f2071f95000, FUTEX_WAKE, 1) = 1
[pid  6039] <... futex resumed>)        = 0
[pid  6037] wait4(-1,  <unfinished ...>
[pid  6039] munmap(0x7f3e96512000, 1024) = 0
[pid  6039] close(3)                    = 0
[pid  6039] munmap(0x7f3e964d8000, 32)  = 0
[pid  6039] munmap(0x7f3e964d7000, 32)  = 0
[pid  6039] close(4)                    = 0
[pid  6039] exit_group(0)               = ?
[pid  6039] +++ exited with 0 +++
<... wait4 resumed>NULL, 0, NULL)       = 6039
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=6039, si_uid=1000,
si_status=0, si_utime=0, si_stime=0} ---
munmap(0x7f2071fd0000, 1024)            = 0
unlink("/dev/shm/my_shared_memory")     = 0
munmap(0x7f2071f96000, 32)              = 0
munmap(0x7f2071f95000, 32)              = 0
unlink("/dev/shm/sem.my_semaphore_parent") = 0
unlink("/dev/shm/sem.my_semaphore_child") = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

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.