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-213Б-23

Студент: Пономарев А.А

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 29.11.24

Москва, 2024

Постановка задачи

Вариант 13.

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Child1 и Child2 можно «соединить» между собой дополнительным каналом. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1. Процесс child1 и child2 производят работу над строками. Child2 пересылает результат своей работы родительскому процессу. Родительский процесс полученный результат выводит в стандартный поток вывода.

Child1 переводит строки в нижний регистр. Child2 превращает все пробельные символы в символ «_».

Общий метод и алгоритм решения

Использованные системные вызовы:

- pid_t fork() - создание дочернего процесса
- int execve(const char *filename, char *const argv[], char *const envp[]) (и другие вариации exec) - замена образа памяти процесса
- int shm_open(const char *name, int oflag, mode_t mode) – создание/открытие именованной области разделенной памяти
- int shm_unlink(const char *name) – уничтожение именованной области памяти
- int ftruncate(int fd, off_t length) – устанавливает размер файла
- mmap - отображает файл или объект общей памяти в виртуальное адресное пространство процесса.
- sem_t *sem_open(const char *, int, ...) – создание/открытие именованного семефора
- int sem_wait(sem_t *) – уменьшить семафор (ожидание если 0)
- int sem_post(sem_t *) – увеличить семафор
- int sem_unlink(const char *) – уничтожить именованный семафор

1. Общая структура:

parent.c — родительский процесс, который получает данные от пользователя, передает их в общую память и управляет взаимодействием с дочерними процессами.

child1.c — первый дочерний процесс, который принимает строку, преобразует её в нижний регистр.

child2.c — второй дочерний процесс, который заменяет пробелы в строке на символы подчеркивания (_).

2. Процесс работы программы:

1. Создание и настройка общей памяти и семафоров (в `parent.c`):

1. Родительский процесс начинает с создания объекта общей памяти с помощью функции `shm_open`. Это объект, который может быть использован для обмена данными между процессами.
2. С помощью `ftruncate` устанавливается размер общей памяти (в данном случае 1024 байта).

3. С помощью `mmap` общая память отображается в адресное пространство родительского процесса.
4. Затем родительский процесс создаёт три семафора:
5. `sem_parent` — используется для синхронизации родительского процесса с дочерними.
6. `sem_child1` и `sem_child2` — для синхронизации между родителем и двумя дочерними процессами.

2. Создание дочерних процессов (в `parent.c`):

1. Родительский процесс создаёт два дочерних процесса с помощью `fork()`:
2. Первый дочерний процесс выполняет программу `child1` — преобразует строку в нижний регистр.
3. Второй дочерний процесс выполняет программу `child2` — заменяет пробелы на символы подчеркивания.
4. Родительский процесс запускает дочерние процессы с помощью `execl`.

3. Взаимодействие с пользователем (в `parent.c`):

1. Родительский процесс ожидает ввода строки от пользователя (с помощью `read_message`).
2. Если строка не пустая, она копируется в общую память, и родительский процесс сигнализирует дочерним процессам (с помощью `sem_post`), что они могут начать обработку данных.
3. После того как дочерние процессы обработают данные, родительский процесс ждёт их завершения, используя `sem_wait`, чтобы синхронизировать выполнение.
4. Результат обработки (строка) выводится на экран.
5. Цикл повторяется, пока не будет введена пустая строка, что завершит программу.

4. Обработка данных дочерними процессами (`child1.c` и `child2.c`):

`child1.c`:

1. Первый дочерний процесс синхронизируется с родительским процессом с помощью семафора `sem_child1`.
2. Когда процесс получает строку, он преобразует все символы в нижний регистр с помощью функции `tolower`.
3. После этого процесс сигнализирует родительскому процессу о завершении своей работы, используя семафор `sem_parent`.

`child2.c`:

1. Второй дочерний процесс также синхронизируется с родительским процессом с помощью семафора `sem_child2`.
2. Этот процесс заменяет все пробелы в строке на символы подчеркивания (_).
3. После обработки строки дочерний процесс сигнализирует родительскому процессу о завершении своей работы с помощью семафора `sem_parent`.

5. Завершение работы (в `parent.c`):

1. Когда пользователь вводит пустую строку, родительский процесс завершает цикл и очищает общую память, используя функции `shmdt`, `shmctl`, и `shm_unlink`.
2. Родительский процесс также закрывает и удаляет семафоры с помощью `sem_close` и `sem_unlink`.
3. В конце родительский процесс ждёт завершения дочерних процессов с помощью `wait(NULL)` и завершает выполнение программы.

Пример работы программы:

```
Введите строку (или пустую строку для выхода): Hello WORLD
Результат обработки: hello_world
Введите строку (или пустую строку для выхода): Top LABBBB
Результат обработки: top_labbbb
Введите строку (или пустую строку для выхода):
```

В результате выполнения программы, каждый вводимый текст преобразуется сначала в нижний регистр, а затем пробелы заменяются на подчеркивания.

Код программы

parent.c

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


#define SHM_SIZE 1024

void handle_error(const char *msg) {
    const char *error_message = ": Ошибка\n";
    write(2, msg, strlen(msg));
    write(2, error_message, strlen(error_message));
    _exit(EXIT_FAILURE);
}

void write_message(const char *msg) {
    write(1, msg, strlen(msg));
}

void read_message(char *buffer, size_t size) {
    ssize_t bytes_read = read(0, buffer, size - 1);
    if (bytes_read <= 0) handle_error("Ошибка чтения");
    buffer[bytes_read - 1] = '\0';
}

int main() {
    // Create and open shared memory using shm_open
    int fd = shm_open("/shared_memory", O_CREAT | O_RDWR, 0666);
    if (fd == -1) handle_error("Ошибка создания файла");

    // Set the size of the shared memory object
    if (ftruncate(fd, SHM_SIZE) == -1) handle_error("ftruncate");

    // Map shared memory into process's address space
    char *shared_memory = (char *)mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0);
    if (shared_memory == MAP_FAILED) handle_error("mmap");

    // Create semaphores
    sem_t *sem_parent = sem_open("/sem_parent", O_CREAT, 0666, 0);
    sem_t *sem_child1 = sem_open("/sem_child1", O_CREAT, 0666, 0);
    sem_t *sem_child2 = sem_open("/sem_child2", O_CREAT, 0666, 0);

    if (sem_parent == SEM_FAILED || sem_child1 == SEM_FAILED || sem_child2 ==
SEM_FAILED)
```

```
        handle_error("sem_open");

    pid_t pid1 = fork();
    if (pid1 == -1) handle_error("fork");

    if (pid1 == 0) {
        execl("./child1", "./child1", NULL);
        handle_error("execl (child1)");
    }

    pid_t pid2 = fork();
    if (pid2 == -1) handle_error("fork");

    if (pid2 == 0) {
        execl("./child2", "./child2", NULL);
        handle_error("execl (child2)");
    }

    write_message("Введите строку (или пустую строку для выхода): ");
    char input_buffer[SHM_SIZE];

    while (1) {
        read_message(input_buffer, SHM_SIZE);

        if (strcmp(input_buffer, "") == 0) {
            break;
        }

        strcpy(shared_memory, input_buffer);

        sem_post(sem_child1);
        sem_wait(sem_parent);

        sem_post(sem_child2);
        sem_wait(sem_parent);

        write_message("Результат обработки: ");
        write_message(shared_memory);
        write_message("\nВведите строку (или пустую строку для выхода): ");
    }

    strcpy(shared_memory, "");
    sem_post(sem_child1);
    sem_post(sem_child2);

    wait(NULL);
    wait(NULL);

    // Cleanup
    if (munmap(shared_memory, SHM_SIZE) == -1) handle_error("munmap");
    if (shm_unlink("/shared_memory") == -1) handle_error("shm_unlink");

    if (sem_close(sem_parent) == -1) handle_error("sem_close sem_parent");
    if (sem_close(sem_child1) == -1) handle_error("sem_close sem_child1");
    if (sem_close(sem_child2) == -1) handle_error("sem_close sem_child2");

    if (sem_unlink("/sem_parent") == -1) handle_error("sem_unlink sem_parent");
    if (sem_unlink("/sem_child1") == -1) handle_error("sem_unlink sem_child1");
    if (sem_unlink("/sem_child2") == -1) handle_error("sem_unlink sem_child2");

    return 0;
}
```

child1.c

```
#include <unistd.h>
#include <stdlib.h>
#include <ctype.h>
```

```c
#include <sys/mman.h>
#include <errno.h>
#include <string.h>
#include <semaphore.h>
#include <fcntl.h>

#define SHM_SIZE 1024

void handle_error(const char *msg) {
    write(2, msg, strlen(msg));
    _exit(EXIT_FAILURE);
}

int main() {
    // Open shared memory using shm_open
    int fd = shm_open("/shared_memory", O_RDWR, 0666);
    if (fd == -1) handle_error("shm_open");

    // Map shared memory into process's address space
    char *shared_memory = (char *)mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0);
    if (shared_memory == MAP_FAILED) handle_error("mmap");

    // Open semaphores
    sem_t *sem_parent = sem_open("/sem_parent", 0);
    sem_t *sem_child1 = sem_open("/sem_child1", 0);

    if (sem_parent == SEM_FAILED || sem_child1 == SEM_FAILED)
        handle_error("sem_open");

    while (1) {
        sem_wait(sem_child1);

        if (strcmp(shared_memory, "") == 0) break;

        for (int i = 0; shared_memory[i] != '\0'; i++) {
            shared_memory[i] = tolower(shared_memory[i]);
        }
        sem_post(sem_parent);
    }

    if (sem_close(sem_parent) == -1) handle_error("sem_close sem_parent");
    if (sem_close(sem_child1) == -1) handle_error("sem_close sem_child1");

    if (munmap(shared_memory, SHM_SIZE) == -1) handle_error("munmap");

    return 0;
}
```

child2.c

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

#define SHM_SIZE 1024

void handle_error(const char *msg) {
    write(2, msg, strlen(msg));
    _exit(EXIT_FAILURE);
}

int main() {
    // Open shared memory using shm_open
    int fd = shm_open("/shared_memory", O_RDWR, 0666);
```

```
    if (fd == -1) handle_error("shm_open");

    // Map shared memory into process's address space
    char *shared_memory = (char *)mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0);
    if (shared_memory == MAP_FAILED) handle_error("mmap");

    // Open semaphores
    sem_t *sem_parent = sem_open("/sem_parent", 0);
    sem_t *sem_child2 = sem_open("/sem_child2", 0);

    if (sem_parent == SEM_FAILED || sem_child2 == SEM_FAILED)
        handle_error("sem_open");

    while (1) {
        sem_wait(sem_child2);

        if (strcmp(shared_memory, "") == 0) break;

        for (int i = 0; shared_memory[i] != '\0'; i++) {
            if (shared_memory[i] == ' ')
                shared_memory[i] = '_';
        }

        sem_post(sem_parent);
    }

    if (sem_close(sem_parent) == -1) handle_error("sem_close sem_parent");
    if (sem_close(sem_child2) == -1) handle_error("sem_close sem_child2");

    if (munmap(shared_memory, SHM_SIZE) == -1) handle_error("munmap");

    return 0;
}
```

Протокол работы программы

Тестирование:

./final

Введите строку (или пустую строку для выхода): Hello Why

Результат обработки: hello___why

Введите строку (или пустую строку для выхода): I am MARK

Результат обработки: i_am_mark

Введите строку (или пустую строку для выхода): Nope

Результат обработки: nope

Введите строку (или пустую строку для выхода):

Strace:

execve("./final", ["./final"], 0x7ffe6abbd4f0 /* 49 vars */) = 0

brk(NULL)   = 0x55721e431000

arch_prctl(0x3001 /* ARCH_??? */, 0x7fff4ae9fae0) = -1 EINVAL (Недопустимый аргумент)

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

fstat(3, {st_mode=S_IFREG|0644, st_size=70284, ...}) = 0

mmap(NULL, 70284, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f9508618000

close(3)   = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220q\0\0\0\0\0\0"..., 832) = 832

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\232e\273F\236E\241\306\373\317\372\345\270*/\327"..., 68, 824) = 68

fstat(3, {st_mode=S_IFREG|0755, st_size=157224, ...}) = 0

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f9508616000

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\232e\273F\236E\241\306\373\317\372\345\270*/\327"..., 68, 824) = 68

mmap(NULL, 140408, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f95085f3000

mmap(0x7f95085f9000, 69632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x6000) = 0x7f95085f9000

mmap(0x7f950860a000, 24576, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x17000) = 0x7f950860a000

mmap(0x7f9508610000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1c000) = 0x7f9508610000

mmap(0x7f9508612000, 13432, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f9508612000

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\300A\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\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0", 32, 848) = 32

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\7\2C\n\357_\243\335\2449\206V>\237\374\304"..., 68, 880) = 68

fstat(3, {st_mode=S_IFREG|0755, st_size=2029592, ...}) = 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

pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\7\2C\n\357_\243\335\2449\206V>\237\374\304"..., 68, 880) = 68

mmap(NULL, 2037344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f9508401000

mmap(0x7f9508423000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x7f9508423000

mmap(0x7f950859b000, 319488, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19a000) = 0x7f950859b000

mmap(0x7f95085e9000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f95085e9000

mmap(0x7f95085ef000, 13920, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f95085ef000

close(3)   = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f95083fe000

arch_prctl(ARCH_SET_FS, 0x7f95083fe740) = 0

mprotect(0x7f95085e9000, 16384, PROT_READ) = 0

mprotect(0x7f9508610000, 4096, PROT_READ) = 0

mprotect(0x557203b27000, 4096, PROT_READ) = 0

mprotect(0x7f9508657000, 4096, PROT_READ) = 0

munmap(0x7f9508618000, 70284)   = 0

set_tid_address(0x7f95083fea10)   = 2771

set_robust_list(0x7f95083fea20, 24)   = 0

rt_sigaction(SIGRTMIN, {sa_handler=0x7f95085f9bf0, sa_mask=[], sa_flags=SA_RESTORER|SA_SIGINFO, sa_restorer=0x7f9508607420}, NULL, 8) = 0

rt_sigaction(SIGRT_1, {sa_handler=0x7f95085f9c90, sa_mask=[], sa_flags=SA_RESTORER|SA_RESTART|SA_SIGINFO, sa_restorer=0x7f9508607420}, NULL, 8) = 0

rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0

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

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

close(3) = 0

stat("shared_memory", {st_mode=S_IFREG|0664, st_size=0, ...}) = 0

shmget(0x41052c36, 1024, IPC_CREAT|0666) = 18

shmat(18, NULL, 0) = 0x7f9508656000

statfs("/dev/shm/", {f_type=TMPFS_MAGIC, f_bsize=4096, f_blocks=500620, f_bfree=491706, f_bavail=491706, f_files=500620, f_ffree=500478, f_fsid={val=[2994540076, 841147858]}, f_namelen=255, f_frsize=4096, f_flags=ST_VALID|ST_NOSUID|ST_NODEV}) = 0

futex(0x7f9508615390, FUTEX_WAKE_PRIVATE, 2147483647) = 0

openat(AT_FDCWD, "/dev/shm/sem.sem_parent", O_RDWR|O_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)

getpid() = 2771

lstat("/dev/shm/8ICaKH", 0x7fff4ae9f440) = -1 ENOENT (Нет такого файла или каталога)

openat(AT_FDCWD, "/dev/shm/8ICaKH", O_RDWR|O_CREAT|O_EXCL, 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) = 0x7f9508629000

link("/dev/shm/8ICaKH", "/dev/shm/sem.sem_parent") = 0

fstat(3, {st_mode=S_IFREG|0664, st_size=32, ...}) = 0

brk(NULL) = 0x55721e431000

brk(0x55721e452000) = 0x55721e452000

unlink("/dev/shm/8ICaKH") = 0

close(3) = 0

openat(AT_FDCWD, "/dev/shm/sem.sem_child1", O_RDWR|O_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)

getpid() = 2771

lstat("/dev/shm/lWcdeK", 0x7fff4ae9f440) = -1 ENOENT (Нет такого файла или каталога)

openat(AT_FDCWD, "/dev/shm/lWcdeK", O_RDWR|O_CREAT|O_EXCL, 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) = 0x7f9508628000

link("/dev/shm/lWcdeK", "/dev/shm/sem.sem_child1") = 0

fstat(3, {st_mode=S_IFREG|0664, st_size=32, ...}) = 0

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

close(3)   = 0

openat(AT_FDCWD, "/dev/shm/sem.sem_child2", O_RDWR|O_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)

getpid()   = 2771

lstat("/dev/shm/Cmv9SH", 0x7fff4ae9f440) = -1 ENOENT (Нет такого файла или каталога)

openat(AT_FDCWD, "/dev/shm/Cmv9SH", O_RDWR|O_CREAT|O_EXCL, 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) = 0x7f9508627000

link("/dev/shm/Cmv9SH", "/dev/shm/sem.sem_child2") = 0

fstat(3, {st_mode=S_IFREG|0664, st_size=32, ...}) = 0

unlink("/dev/shm/Cmv9SH")   = 0

close(3)   = 0

clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f95083fea10) = 2772

clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f95083fea10) = 2773

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\321\203 (\320\270\320"..., 84Введите строку (или пустую строку для выхода):) = 84

read(0, Hello   WHY

"Hello   WHY\n", 1023)   = 12

futex(0x7f9508628000, FUTEX_WAKE, 1)   = 1

futex(0x7f9508627000, FUTEX_WAKE, 1)   = 1

write(1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202
\320\276\320\261\321\200\320\260\320\261\320\276\321"..., 39Результат обработки:) = 39

write(1, "hello___why", 11hello___why)   = 11

write(1, "\n\320\222\320\262\320\265\320\264\320\270\321\202\320\265
\321\201\321\202\321\200\320\276\320\272\321\203 (\320\270"..., 85

Введите строку (или пустую строку для выхода):) = 85

read(0, I am MARK

"I am MARK\n", 1023)   = 10

futex(0x7f9508628000, FUTEX_WAKE, 1)   = 1

futex(0x7f9508629000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY) = 0

futex(0x7f9508627000, FUTEX_WAKE, 1)   = 1

futex(0x7f9508629000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY) = 0

write(1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202 \320\276\320\261\321\200\320\260\320\261\320\276\321"..., 39Результат обработки:) = 39

write(1, "i_am_mark", 9i_am_mark)   = 9

write(1, "\n\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\201\321\202\321\200\320\276\320\272\321\203 (\320\270"..., 85

Введите строку (или пустую строку для выхода):) = 85

read(0, Nope

"Nope\n", 1023)   = 5

futex(0x7f9508628000, FUTEX_WAKE, 1)   = 1

futex(0x7f9508629000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY) = 0

futex(0x7f9508627000, FUTEX_WAKE, 1)   = 1

futex(0x7f9508629000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY) = -1 EAGAIN (Ресурс временно недоступен)

write(1, "\320\240\320\265\320\267\321\203\320\273\321\214\321\202\320\260\321\202 \320\276\320\261\321\200\320\260\320\261\320\276\321"..., 39Результат обработки:) = 39

write(1, "nope", 4nope)   = 4

write(1, "\n\320\222\320\262\320\265\320\264\320\270\321\202\320\265 \321\201\321\202\321\200\320\276\320\272\321\203 (\320\270"..., 85

Введите строку (или пустую строку для выхода):) = 85

read(0,

"\n", 1023)   = 1

futex(0x7f9508628000, FUTEX_WAKE, 1)   = 1

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=2772, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

futex(0x7f9508627000, FUTEX_WAKE, 1)   = 1

wait4(-1, NULL, 0, NULL)   = 2772

wait4(-1, NULL, 0, NULL) = 2773

--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=2773, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---

shmdt(0x7f9508656000) = 0

shmctl(18, IPC_RMID, NULL) = 0

munmap(0x7f9508629000, 32) = 0

munmap(0x7f9508628000, 32) = 0

munmap(0x7f9508627000, 32) = 0

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

unlink("/dev/shm/sem.sem_child1") = 0

unlink("/dev/shm/sem.sem_child2") = 0

exit_group(0) = ?

+++ exited with 0 +++

Вывод

В ходе лабораторной работы была реализована программа, использующая общую память для межпроцессного взаимодействия. Родительский процесс записывает строку в общую память, а два дочерних процесса последовательно изменяют её, преобразуя символы в нижний регистр и заменяя пробелы на подчеркивания. Работа с системными вызовами, такими как `shmget`, `shmat`, и `shmdt`, а также синхронизация процессов через `wait`, позволили продемонстрировать основы взаимодействия между процессами. Лабораторная работа улучшила понимание механизмов работы с памятью и процессами в Linux.