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: Чувилов А.А.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 27.12.24

Москва, 2024

Постановка задачи

Вариант 21.

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами. Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы инвертируют строки.

Общий метод и алгоритм решения

Использованные системные вызовы:

- **write(int fd, const void *buf, size_t count)** - аписывает данные в файловый дескриптор (например, выводит текст на экран).
- **read(int fd, void *buf, size_t count)** - считывает данные из файлового дескриптора (например, ввод пользователя с клавиатуры).
- **shm_open(const char *name, int oflag, mode_t mode)** - открывает или создаёт объект разделяемой памяти.
- **ftruncate(int fd, off_t length)** - задаёт размер объекта разделяемой памяти.
- **mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset)** - отображает объект разделяемой памяти в адресное пространство процесса.
- **munmap(void *addr, size_t length)** - удаляет отображение разделяемой памяти.
- **shm_unlink(const char *name)** - удаляет объект разделяемой памяти.
- **sem_open(const char *name, int oflag, mode_t mode, unsigned int value)** - создаёт или открывает именованный семафор.
- **sem_post(sem_t *sem)** - увеличивает значение семафора (разблокирует ожидающий процесс).
- **sem_unlink(const char *name)** - даляет именованный семафор.
- **fork()** - создаёт новый процесс (дочерний).
- **execlp(const char *file, const char *arg, ..., NULL)** - заменяет текущий процесс новым (запускает другую программу).
- **time(time_t *tloc)** - возвращает текущее время.
- **exit(int status)** - завершает выполнение программы

Данный алгоритм организует взаимодействие между родительским процессом и двумя дочерними процессами через разделяемую память (shared memory) и семафоры. Сначала пользователь вводит имена файлов для двух дочерних процессов. Родительский процесс создает разделяемую память и семафоры для синхронизации. Затем запускаются два дочерних процесса, которые будут читать данные из памяти, основываясь на сигналах от родителя через семафоры. Родительский процесс в бесконечном цикле запрашивает ввод строки от пользователя, записывает ее в разделяемую память и активирует соответствующий семафор, чередуя доступ между дочерними процессами. При вводе "exit" процесс завершает свою работу, освобождая все созданные ресурсы, включая память, семафоры и файлы.

Код программы

main.c

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

#define SHM_NAME "/shm_example"
#define SEM_CHILD1 "/sem_child1"
#define SEM_CHILD2 "/sem_child2"

#define SHM_SIZE 1024

void HandleError(const char *msg) {
    write(STDERR_FILENO, msg, strlen(msg));
    write(STDERR_FILENO, "\n", 1);
```

```
    exit(1);
}

void Print(const char *msg) {
    write(STDOUT_FILENO, msg, strlen(msg));
}

ssize_t Getline(char **lineptr, size_t *n, int fd) {
    if (*lineptr == NULL) {
        *lineptr = malloc(128);
        *n = 128;
    }

    size_t pos = 0;
    char c;
    while (read(fd, &c, 1) == 1) {
        if (pos >= *n - 1) {
            *n *= 2;
            *lineptr = realloc(*lineptr, *n);
        }
        (*lineptr)[pos++] = c;
        if (c == '\n') {
            break;
        }
    }

    if (pos == 0) {
        return -1;
    }

    (*lineptr)[pos] = '\0';
    return pos;
}
```

```
int main() {
    char *file1 = NULL, *file2 = NULL;
    size_t file_len = 0;
    char *input = NULL;
    size_t len = 0;
    ssize_t nread;

    Print("Введите имя файла для дочернего процесса 1: ");
    Getline(&file1, &file_len, STDIN_FILENO);
    file1[strcspn(file1, "\n")] = 0;

    Print("Введите имя файла для дочернего процесса 2: ");
    Getline(&file2, &file_len, STDIN_FILENO);
    file2[strcspn(file2, "\n")] = 0;

    // Создаем shared memory
    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0644);
    if (shm_fd == -1) {
        HandleError("Ошибка создания разделяемой памяти");
    }

    if (ftruncate(shm_fd, SHM_SIZE) == -1) {
        HandleError("Ошибка установки размера разделяемой памяти");
    }

    char *shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        HandleError("Ошибка отображения разделяемой памяти");
    }

    // Создаем семафоры
```

```
sem_t *sem_child1 = sem_open(SEM_CHILD1, O_CREAT, 0644, 0);
sem_t *sem_child2 = sem_open(SEM_CHILD2, O_CREAT, 0644, 0);
if (sem_child1 == SEM_FAILED || sem_child2 == SEM_FAILED) {
    HandleError("Ошибка создания семафоров");
}

pid_t child1, child2;

if ((child1 = fork()) == 0) {
    execlp("./child1", "./child1", file1, NULL);
    HandleError("Ошибка запуска дочернего процесса 1");
}

if ((child2 = fork()) == 0) {
    execlp("./child2", "./child2", file2, NULL);
    HandleError("Ошибка запуска дочернего процесса 2");
}

srand(time(NULL));
int k = 1;
while (1) {
    Print("Введите строку (или 'exit' для завершения): ");
    nread = Getline(&input, &len, STDIN_FILENO);
    input[strcspn(input, "\n")] = 0;

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

    strncpy(shm_ptr, input, SHM_SIZE - 1);

    if (k % 2 == 0) {
        sem_post(sem_child2);
```

```
        } else {
            sem_post(sem_child1);
        }
        k+=1;
    }

    Print("Работа завершена.\n");

    // Удаляем ресурсы
    munmap(shm_ptr, SHM_SIZE);
    shm_unlink(SHM_NAME);
    sem_unlink(SEM_CHILD1);
    sem_unlink(SEM_CHILD2);



    free(file1);
    free(file2);
free(inout);
return 0;
}
```

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

#define SHM_NAME "/shm_example"
#define SEM_CHILD1 "/sem_child1"
#define SHM_SIZE 1024

void HandleError(const char *msg) {
    write(STDERR_FILENO, msg, strlen(msg));
    write(STDERR_FILENO, "\n", 1);
    exit(1);
}
```

```
void ReverseString(char *str) {
    size_t len = strlen(str);
    for (size_t i = 0; i < len / 2; ++i) {
        char temp = str[i];
        str[i] = str[len - i - 1];
        str[len - i - 1] = temp;
    }
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        HandleError("Usage: <program> <output_file>");
    }

    int fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd == -1) {
        HandleError("Cannot open file");
    }

    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0644);
    if (shm_fd == -1) {
        HandleError("Ошибка доступа к разделяемой памяти");
    }

    char *shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED,
shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        HandleError("Ошибка отображения разделяемой памяти");
    }

    sem_t *sem_child1 = sem_open(SEM_CHILD1, 0);
    if (sem_child1 == SEM_FAILED) {
        HandleError("Ошибка доступа к семафору");
    }

    while (1) {
        sem_wait(sem_child1);

        char buffer[SHM_SIZE];
        strncpy(buffer, shm_ptr, SHM_SIZE - 1);
        buffer[SHM_SIZE - 1] = '\0';

        ReverseString(buffer);
        write(fd, buffer, strlen(buffer));
        write(fd, "\n", 1);
    }

    munmap(shm_ptr, SHM_SIZE);
    close(fd);

    return 0;
```

```
}
```

child2.c

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

#define SHM_NAME "/shm_example"
#define SEM_CHILD2 "/sem_child2"
#define SHM_SIZE 1024

void HandleError(const char *msg) {
    write(STDERR_FILENO, msg, strlen(msg));
    write(STDERR_FILENO, "\n", 1);
    exit(1);
}

void ToUpperCase(char *str) {
    while (*str) {
        *str = toupper((unsigned char)*str);
        str++;
    }
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        HandleError("Usage: <program> <output_file>");
    }

    int fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0644);
    if (fd == -1) {
        HandleError("Cannot open file");
    }

    int shm_fd = shm_open(SHM_NAME, O_RDWR, 0644);
    if (shm_fd == -1) {
        HandleError("Ошибка доступа к разделяемой памяти");
    }

    char *shm_ptr = mmap(NULL, SHM_SIZE, PROT_READ | PROT_WRITE,
MAP_SHARED, shm_fd, 0);
    if (shm_ptr == MAP_FAILED) {
        HandleError("Ошибка отображения разделяемой памяти");
    }

    sem_t *sem_child2 = sem_open(SEM_CHILD2, 0);
    if (sem_child2 == SEM_FAILED) {
```

```
        HandleError("Ошибка доступа к семафору");
    }

    while (1) {
        sem_wait(sem_child2);

        char buffer[SHM_SIZE];
        strncpy(buffer, shm_ptr, SHM_SIZE - 1);
        buffer[SHM_SIZE - 1] = '\0';

        ToUpperCase(buffer);
        write(fd, buffer, strlen(buffer));
        write(fd, "\n", 1);
    }

    munmap(shm_ptr, SHM_SIZE);
    close(fd);

    return 0;
}
```

Протокол работы программы

Тестирование:

$ ./a.out

Enter filename for child 1: 1.txt

Enter filename for child 2: 2.txt

Enter a line: hello

Enter a line: lol

Enter a line: world

Enter a line: how are you?

Enter a line: ddd

Enter a line: exit

Вывод в файле 1.txt:

olleh

dlrow

ddd

Вывод в файле 2.txt:

LOL

HOW ARE YOU?

Strace:

strace -f ./p

execve("./p", ["./p"], 0x7ffcdaf78058 /* 46 vars */) = 0

brk(NULL) = 0x599f3787e000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffdf0ecf8a0) = -1 EINVAL (Недопустимый аргумент)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x72c4ad48a000

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=58047, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 58047, PROT_READ, MAP_PRIVATE, 3, 0) = 0x72c4ad47b000

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) = 0x72c4ad200000

mprotect(0x72c4ad228000, 2023424, PROT_NONE) = 0

**mmap(0x72c4ad228000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x72c4ad228000**

**mmap(0x72c4ad3bd000, 360448, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x72c4ad3bd000**

**mmap(0x72c4ad416000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x72c4ad416000**

**mmap(0x72c4ad41c000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x72c4ad41c000**

close(3)   = 0

**mmap(NULL, 12288, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x72c4ad478000**

arch_prctl(ARCH_SET_FS, 0x72c4ad478740) = 0

set_tid_address(0x72c4ad478a10)   = 4452

set_robust_list(0x72c4ad478a20, 24)   = 0

rseq(0x72c4ad4790e0, 0x20, 0, 0x53053053) = 0

mprotect(0x72c4ad416000, 16384, PROT_READ) = 0

mprotect(0x599f377ef000, 4096, PROT_READ) = 0

mprotect(0x72c4ad4c4000, 8192, PROT_READ) = 0

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

munmap(0x72c4ad47b000, 58047)   = 0

**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"..., 79Введите имя
файла для дочернего процесса 1:) = 79**

getrandom("\xf4\xa6\xd3\x64\x8a\x16\xed\x82", 8, GRND_NONBLOCK) = 8

brk(NULL)   = 0x599f3787e000

brk(0x599f3789f000)   = 0x599f3789f000

read(0, f1

"f", 1)   = 1

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

read(0, "\n", 1)   = 1

**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"..., 79Введите имя
файла для дочернего процесса 2:) = 79**

read(0, f2

"f", 1) = 1

read(0, "2", 1) = 1

read(0, "\n", 1) = 1

openat(AT_FDCWD, "/dev/shm/shm_example",
O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0644) = 3

ftruncate(3, 1024) = 0

mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) =
0x72c4ad4c3000

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

getrandom("\xd8\xef\xc8\x7d\x83\xc5\x74\xf5", 8, GRND_NONBLOCK) = 8

getrandom("\x3f\x14\x95\xd4\xb3\x9a\xc0\xee", 8, GRND_NONBLOCK) = 8

newfstatat(AT_FDCWD, "/dev/shm/sem.LDbYQg", 0x7ffdf0ecf580,
AT_SYMLINK_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)

openat(AT_FDCWD, "/dev/shm/sem.LDbYQg", O_RDWR|O_CREAT|O_EXCL, 0644) = 4

write(4, "\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, 4, 0) = 0x72c4ad489000

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

newfstatat(4, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0

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

close(4) = 0

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

getrandom("\x61\x24\xbb\x26\xa9\x1d\xc7\x85", 8, GRND_NONBLOCK) = 8

newfstatat(AT_FDCWD, "/dev/shm/sem.3D58Ew", 0x7ffdf0ecf580,
AT_SYMLINK_NOFOLLOW) = -1 ENOENT (Нет такого файла или каталога)

openat(AT_FDCWD, "/dev/shm/sem.3D58Ew", O_RDWR|O_CREAT|O_EXCL, 0644) = 4

write(4, "\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, 4, 0) = 0x72c4ad488000

link("/dev/shm/sem.3D58Ew", "/dev/shm/sem.sem_child2") = 0

newfstatat(4, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0

unlink("/dev/shm/sem.3D58Ew") = 0

close(4)   = 0

clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLDstrace: Process
4453 attached

, child_tidptr=0x72c4ad478a10) = 4453

[pid 4452] clone(child_stack=NULL,
flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD <unfinished ...>

[pid 4453] set_robust_list(0x72c4ad478a20, 24) = 0

strace: Process 4454 attached

[pid 4453] execve("./child1", ["./child1", "f1"], 0x7ffdf0ecfa78 /* 46 vars */ <unfinished ...>

[pid 4454] set_robust_list(0x72c4ad478a20, 24) = 0

[pid 4452] <... clone resumed>, child_tidptr=0x72c4ad478a10) = 4454

[pid 4454] execve("./child2", ["./child2", "f2"], 0x7ffdf0ecfa78 /* 46 vars */ <unfinished ...>

[pid 4452] 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"..., 73Введите строку (или
'exit' для завершения):) = 73

[pid 4452] read(0, <unfinished ...>

[pid 4454] <... execve resumed>)   = 0

[pid 4454] brk(NULL)   = 0x60f0f9047000

[pid 4453] <... execve resumed>)   = 0

[pid 4453] brk(NULL)   = 0x592559052000

[pid 4454] arch_prctl(0x3001 /* ARCH_??? */, 0x7ffdbc779fc0 <unfinished ...>

[pid 4453] arch_prctl(0x3001 /* ARCH_??? */, 0x7fffd3481d10 <unfinished ...>

[pid 4454] <... arch_prctl resumed>)   = -1 EINVAL (Недопустимый аргумент)

[pid 4454] mmap(NULL, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x71d1310e2000

[pid 4454] access("/etc/ld.so.preload", R_OK) = -1 ENOENT (Нет такого файла или
каталога)

[pid 4454] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

[pid 4454] newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=58047, ...},
AT_EMPTY_PATH) = 0

[pid 4454] mmap(NULL, 58047, PROT_READ, MAP_PRIVATE, 3, 0) = 0x71d1310d3000

[pid 4454] close(3)   = 0

[pid 4454] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6",

O_RDONLY|O_CLOEXEC) = 3

[pid 4454] read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0\0"..., 832) = 832

[pid 4454] 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 4454] 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 4454] 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 4454] newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0

[pid 4454] 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 4454] mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x71d130e00000

[pid 4454] mprotect(0x71d130e28000, 2023424, PROT_NONE) = 0

[pid 4454] mmap(0x71d130e28000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x71d130e28000

[pid 4453] <... arch_prctl resumed>) = -1 EINVAL (Недопустимый аргумент)

[pid 4453] mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>

[pid 4454] mmap(0x71d130fbd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x71d130fbd000

[pid 4454] mmap(0x71d131016000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x71d131016000

[pid 4453] <... mmap resumed>) = 0x77651ebb4000

[pid 4454] mmap(0x71d13101c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0 <unfinished ...>

[pid 4453] access("/etc/ld.so.preload", R_OK <unfinished ...>

[pid 4454] <... mmap resumed>) = 0x71d13101c000

[pid 4453] <... access resumed>) = -1 ENOENT (Нет такого файла или каталога)

[pid 4453] openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

[pid 4453] newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=58047, ...}, AT_EMPTY_PATH) = 0

[pid 4453] mmap(NULL, 58047, PROT_READ, MAP_PRIVATE, 3, 0) = 0x77651eba5000

[pid 4453] close(3) = 0

[pid 4454] close(3 <unfinished ...>

[pid 4453] openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

[pid 4454] <... close resumed>) = 0

[pid 4454] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>

[pid 4453] 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 4453] pread64(3, <unfinished ...>

[pid 4454] <... mmap resumed>) = 0x71d1310d0000

[pid 4453] <... pread64 resumed>"\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 4453] 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 4454] arch_prctl(ARCH_SET_FS, 0x71d1310d0740 <unfinished ...>

[pid 4453] 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 4453] newfstatat(3, "", <unfinished ...>

[pid 4454] <... arch_prctl resumed>) = 0

[pid 4453] <... newfstatat resumed>{st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0

[pid 4454] set_tid_address(0x71d1310d0a10 <unfinished ...>

[pid 4453] pread64(3, <unfinished ...>

[pid 4454] <... set_tid_address resumed>) = 4454

[pid 4453] <... pread64 resumed>"\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 4454] set_robust_list(0x71d1310d0a20, 24) = 0

[pid 4453] mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0 <unfinished ...>

[pid 4454] rseq(0x71d1310d10e0, 0x20, 0, 0x53053053) = 0

[pid 4453] <... mmap resumed>) = 0x77651e800000

[pid 4454] mprotect(0x71d131016000, 16384, PROT_READ <unfinished ...>

[pid 4453] mprotect(0x77651e828000, 2023424, PROT_NONE) = 0

[pid 4453] mmap(0x77651e828000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x77651e828000

[pid 4453] mmap(0x77651e9bd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1bd000) = 0x77651e9bd000

[pid 4453] mmap(0x77651ea16000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x77651ea16000

[pid 4454] <... mprotect resumed>) = 0

[pid 4453] mmap(0x77651ea1c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0 <unfinished ...>

[pid 4454] mprotect(0x60f0f8f61000, 4096, PROT_READ <unfinished ...>

[pid 4453] <... mmap resumed>) = 0x77651ea1c000

[pid 4454] <... mprotect resumed>) = 0

[pid 4453] close(3 <unfinished ...>

[pid 4454] mprotect(0x71d13111c000, 8192, PROT_READ <unfinished ...>

[pid 4453] <... close resumed>) = 0

[pid 4454] <... mprotect resumed>) = 0

[pid 4454] prlimit64(0, RLIMIT_STACK, NULL, <unfinished ...>

[pid 4453] mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>

[pid 4454] <... prlimit64 resumed>{rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

[pid 4453] <... mmap resumed>) = 0x77651eba2000

[pid 4454] munmap(0x71d1310d3000, 58047 <unfinished ...>

[pid 4453] arch_prctl(ARCH_SET_FS, 0x77651eba2740 <unfinished ...>

[pid 4454] <... munmap resumed>) = 0

[pid 4453] <... arch_prctl resumed>) = 0

[pid 4453] set_tid_address(0x77651eba2a10) = 4453

[pid 4453] set_robust_list(0x77651eba2a20, 24 <unfinished ...>

[pid 4454] openat(AT_FDCWD, "f2", O_WRONLY|O_CREAT|O_TRUNC, 0644 <unfinished ...>

[pid 4453] <... set_robust_list resumed>) = 0

[pid 4453] rseq(0x77651eba30e0, 0x20, 0, 0x53053053) = 0

[pid 4454] <... openat resumed>) = 3

[pid 4453] mprotect(0x77651ea16000, 16384, PROT_READ) = 0

[pid 4454] openat(AT_FDCWD, "/dev/shm/shm_example", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 4

[pid 4453] mprotect(0x5925581c2000, 4096, PROT_READ <unfinished ...>

[pid 4454] mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x71d13111b000

[pid 4453] <... mprotect resumed>) = 0

[pid 4454] openat(AT_FDCWD, "/dev/shm/sem.sem_child2", O_RDWR|O_NOFOLLOW) = 5

[pid 4454] newfstatat(5, "", <unfinished ...>

[pid 4453] mprotect(0x77651ebee000, 8192, PROT_READ <unfinished ...>

[pid 4454] <... newfstatat resumed>{st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0

[pid 4454] getrandom(<unfinished ...>

[pid 4453] <... mprotect resumed>) = 0

[pid 4454] <... getrandom resumed>"\xa4\x77\x6f\x5b\x1f\xbc\x9f\xa8", 8, GRND_NONBLOCK) = 8

[pid 4454] brk(NULL <unfinished ...>

[pid 4453] prlimit64(0, RLIMIT_STACK, NULL, <unfinished ...>

[pid 4454] <... brk resumed>) = 0x60f0f9047000

[pid 4454] brk(0x60f0f9068000) = 0x60f0f9068000

[pid 4454] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0 <unfinished ...>

[pid 4453] <... prlimit64 resumed>{rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

[pid 4454] <... mmap resumed>) = 0x71d1310e1000

[pid 4453] munmap(0x77651eba5000, 58047 <unfinished ...>

[pid 4454] close(5) = 0

[pid 4453] <... munmap resumed>) = 0

[pid 4454] futex(0x71d1310e1000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>

[pid 4453] openat(AT_FDCWD, "f1", O_WRONLY|O_CREAT|O_TRUNC, 0644) = 3

[pid 4453] openat(AT_FDCWD, "/dev/shm/shm_example",

O_RDWR|O_NOFOLLOW|O_CLOEXEC) = 4

[pid 4453] mmap(NULL, 1024, PROT_READ|PROT_WRITE, MAP_SHARED, 4, 0) = 0x77651ebed000

[pid 4453] openat(AT_FDCWD, "/dev/shm/sem.sem_child1", O_RDWR|O_NOFOLLOW) = 5

[pid 4453] newfstatat(5, "", {st_mode=S_IFREG|0644, st_size=32, ...}, AT_EMPTY_PATH) = 0

[pid 4453] getrandom("\xbd\xa0\xce\xdf\x07\x8f\xed\x4b", 8, GRND_NONBLOCK) = 8

[pid 4453] brk(NULL) = 0x592559052000

[pid 4453] brk(0x592559073000) = 0x592559073000

[pid 4453] mmap(NULL, 32, PROT_READ|PROT_WRITE, MAP_SHARED, 5, 0) = 0x77651ebb3000

[pid 4453] close(5) = 0

[pid 4453] futex(0x77651ebb3000, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 0, NULL, FUTEX_BITSET_MATCH_ANYexit

<unfinished ...>

[pid 4452] <... read resumed>"e", 1) = 1

[pid 4452] read(0, "x", 1) = 1

[pid 4452] read(0, "i", 1) = 1

[pid 4452] read(0, "t", 1) = 1

[pid 4452] read(0, "\n", 1) = 1

[pid 4452] write(1, "\320\240\320\260\320\261\320\276\321\202\320\260 \320\267\320\260\320\262\320\265\321\200\321\210\320\265\320\275\320\260."..., 33Работа завершена.

) = 33

[pid 4452] munmap(0x72c4ad4c3000, 1024) = 0

[pid 4452] unlink("/dev/shm/shm_example") = 0

[pid 4452] unlink("/dev/shm/sem.sem_child1") = 0

[pid 4452] unlink("/dev/shm/sem.sem_child2") = 0

[pid 4452] exit_group(0) = ?

[pid 4452] +++ exited with 0 +++

Вывод

В ходе выполнения лабораторной работы была реализована

межпроцессная коммуникация с использованием разделяемой памяти и семафоров. Основная задача заключалась в организации взаимодействия родительского процесса с двумя дочерними процессами, которые обрабатывают данные, передаваемые через общую память.

В процессе выполнения работы были изучены и применены следующие механизмы и инструменты:

- Разделяемая память (shared memory), которая обеспечила совместный доступ к данным между процессами.
- Семафоры, которые использовались для синхронизации работы процессов, исключая состояние гонки.
- Системные вызовы для создания процессов (fork) и передачи управления выполняемым программам (execlp).

В ходе работы была организована корректная обработка ошибок, что позволило избежать некорректного поведения программы в случае сбоев. Все созданные ресурсы (память, семафоры) освобождаются по завершении работы программы, что предотвращает их утечку.