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-209Б-24

Студент: Артонкин В.Н.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 29.11.25

Москва, 2025

Постановка задачи

Вариант 20.

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. Правило фильтрации: строки длины больше 10 символов отправляются в file2, иначе в file1. Дочерние процессы инвертируют строки.

Общий метод и алгоритм решения

Использованные системные вызовы:

- **void pthread_exit(void *retval) -** завершает поток и возвращает retval
- **pid_t fork(void) -** создает дочерний процесс.
- **int shm_open(const char *name, int oflag, mode_t mode) -** создает и открывает объект разделяемой памяти POSIX
- **int ftruncate(int fd, off_t length) -** устанавливает длину файлвого дескриптора в length байт
- **void * mmap(void *start, size_t length, int prot , int flags, int fd, off_t offset) -** Функция mmap отражает length байтов, начиная со смещения offset файла (или другого объекта), определенного файловым описателем fd, в память, начиная с адреса start.
- **void free(void *ptr) -** освобождает память.
- **int munmap(void *start, size_t length) -** освобождает разделенную POSIX память
- **int shm_unlink(const char *name) -** удаляет имя объекта разделенной памяти

Сначала были созданы все необходимые переменные и структуры данных, с учетом области памяти, которая потом использовалась в качестве общей для всех процеесов. Также были считаны названия файлов, в которые дочерние процессы писали строки. После считывались строки, клались в общую память процессов, обрабатывались каждым из них по-отдельности

Код программы

parent.c

```c
#include <string.h>

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>
```

```c
#include <sys/wait.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <sys/stat.h>


#define MAX_CHAR 256
#define SHARED_MEM_SIZE 4096
#define SHM_NAME "/string_processor"
#define MAX_STRINGS 100


typedef struct {
    char data[MAX_STRINGS][SHARED_MEM_SIZE];
    size_t lengths[MAX_STRINGS];
    int processed_by_child1[MAX_STRINGS];
    int processed_by_child2[MAX_STRINGS];
    int count;
    int finished;
} shared_data_t;


int main(){
    char filename1[MAX_CHAR];
    char filename2[MAX_CHAR];


    printf("Введите имя 1-го файла\n");
    if (fgets(filename1, sizeof(filename1), stdin) == NULL) {
        perror("filename1 error");
        exit(1);
    }
    printf("Введите имя 2-го файла\n");
```

```
    if (fgets(filename2, sizeof(filename2), stdin) == NULL) {
        perror("filename2 error");
        exit(1);
    }

    filename1[strcspn(filename1, "\n")] = '\0';
    filename2[strcspn(filename2, "\n")] = '\0';

    // Создаем shared memory объект
    int shm_fd = shm_open(SHM_NAME, O_CREAT | O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("shm_open error");
        exit(1);
    }

    // Устанавливаем размер shared memory
    if (ftruncate(shm_fd, sizeof(shared_data_t)) == -1) {
        perror("ftruncate error");
        exit(1);
    }

    // Отображаем shared memory в адресное пространство
    shared_data_t *shared_data = mmap(NULL, sizeof(shared_data_t),
                                      PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd,
0);
    if (shared_data == MAP_FAILED) {
        perror("mmap error");
        exit(1);
    }
```

```
// Инициализируем shared data
shared_data->count = 0;
shared_data->finished = 0;
for (int i = 0; i < MAX_STRINGS; i++) {
    shared_data->processed_by_child1[i] = 0;
    shared_data->processed_by_child2[i] = 0;
}

pid_t pid1, pid2;

// Первый дочерний процесс
switch(pid1 = fork()) {
    case -1:
        perror("fork 1 error");
        exit(1);
    case 0:
        execl("./child1", "./child1", filename1, SHM_NAME, NULL);
        perror("execl child 1 error");
        exit(1);
}

// Второй дочерний процесс
switch(pid2 = fork()) {
    case -1:
        perror("fork 2 error");
        exit(1);
    case 0:
        execl("./child2", "./child2", filename2, SHM_NAME, NULL);
```

```
        perror("execl child 2 error");
        exit(1);
    }

    // Родительский процесс - ввод данных
    char *buffer = NULL;
    size_t size = 0;

    while (1) {
        printf("Введите строку или 'exit' для выхода:\n");
        fflush(stdout);
        ssize_t len = getline(&buffer, &size, stdin);
        if (len == -1) break;

        if (strcmp(buffer, "exit\n") == 0) {
            break;
        }

        // Добавляем строку в shared memory
        if (shared_data->count < MAX_STRINGS) {
            int idx = shared_data->count;
            strncpy(shared_data->data[idx], buffer, SHARED_MEM_SIZE - 1);
            shared_data->data[idx][SHARED_MEM_SIZE - 1] = '\0';
            shared_data->lengths[idx] = len;
            shared_data->count++;
        } else {
            printf("Превышен лимит строк!\n");
        }
    }
```

```
    // Сигнализируем о завершении
    shared_data->finished = 1;

    free(buffer);

    // Ожидаем завершения дочерних процессов
    wait(NULL);
    wait(NULL);

    // Освобождаем ресурсы
    munmap(shared_data, sizeof(shared_data_t));
    shm_unlink(SHM_NAME);

    return 0;
}
```

child1.c

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

#define SHARED_MEM_SIZE 4096
#define SHM_NAME "/string_processor"
#define MAX_STRINGS 100
```

```c
typedef struct {
    char data[MAX_STRINGS][SHARED_MEM_SIZE];
    size_t lengths[MAX_STRINGS];
    int processed_by_child1[MAX_STRINGS];
    int processed_by_child2[MAX_STRINGS];
    int count;
    int finished;
} shared_data_t;

int main(int argc, char *argv[]) {
    if (argc < 3) {
        perror("Not enough arguments\n");
        exit(1);
    }

    FILE *f = fopen(argv[1], "w");
    if (!f) {
        perror("fopen error");
        exit(1);
    }

    // Открываем shared memory
    int shm_fd = shm_open(argv[2], O_RDWR, 0666);
    if (shm_fd == -1) {
        perror("shm_open error in child");
        exit(1);
    }
```

```
    // Отображаем shared memory
    shared_data_t *shared_data = mmap(NULL, sizeof(shared_data_t),
                                      PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd, 0);
    if (shared_data == MAP_FAILED) {
        perror("mmap error in child");
        exit(1);
    }

    while (1) {
        int processed_something = 0;

        // Проверяем все строки
        for (int i = 0; i < shared_data->count; i++) {
            // Обрабатываем только короткие строки (<= 11 символов) и которые еще не обработаны
            if (shared_data->lengths[i] <= 11 && !shared_data->processed_by_child1[i]) {
                // Реверсируем строку
                for (int j = shared_data->lengths[i] - 2; j >= 0; j--) {
                    fputc(shared_data->data[i][j], f);
                }
                fputc('\n', f);
                fflush(f);

                // Помечаем как обработанную
                shared_data->processed_by_child1[i] = 1;
                processed_something = 1;
            }
        }
```

```
        // Если все завершено и больше нечего обрабатывать - выходим
        if (shared_data->finished) {
            int all_processed = 1;
            for (int i = 0; i < shared_data->count; i++) {
                if (shared_data->lengths[i] <= 11 &&
!shared_data->processed_by_child1[i]) {
                    all_processed = 0;
                    break;
                }
            }
            if (all_processed) break;
        }

        // Если ничего не обработали в этой итерации - небольшая пауза
        if (!processed_something) {
            usleep(10000); // 10ms
        }
    }

    fclose(f);
    munmap(shared_data, sizeof(shared_data_t));
    return 0;
}
```

child2.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```c
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <sys/stat.h>

#define SHARED_MEM_SIZE 4096
#define SHM_NAME "/string_processor"
#define MAX_STRINGS 100

typedef struct {
    char data[MAX_STRINGS][SHARED_MEM_SIZE];
    size_t lengths[MAX_STRINGS];
    int processed_by_child1[MAX_STRINGS];
    int processed_by_child2[MAX_STRINGS];
    int count;
    int finished;
} shared_data_t;

int main(int argc, char *argv[]) {
    if (argc < 3) {
        perror("Not enough arguments\n");
        exit(1);
    }

    FILE *f = fopen(argv[1], "w");
    if (!f) {
        perror("fopen error");
        exit(1);
    }
```

```
        // Открываем shared memory
        int shm_fd = shm_open(argv[2], O_RDWR, 0666);
        if (shm_fd == -1) {
            perror("shm_open error in child");
            exit(1);
        }

        // Отображаем shared memory
        shared_data_t *shared_data = mmap(NULL, sizeof(shared_data_t),
                                          PROT_READ | PROT_WRITE, MAP_SHARED, shm_fd,
0);
        if (shared_data == MAP_FAILED) {
            perror("mmap error in child");
            exit(1);
        }

        while (1) {
            int processed_something = 0;

            // Проверяем все строки
            for (int i = 0; i < shared_data->count; i++) {
                // Обрабатываем только длинные строки (> 11 символов) и которые еще
не обработаны
                if (shared_data->lengths[i] > 11 &&
!shared_data->processed_by_child2[i]) {
                    // Реверсируем строку
                    for (int j = shared_data->lengths[i] - 2; j >= 0; j--) {
                        fputc(shared_data->data[i][j], f);
                    }
```

```
                    fputc('\n', f);
                    fflush(f);

                    // Помечаем как обработанную
                    shared_data->processed_by_child2[i] = 1;
                    processed_something = 1;
                }
            }

            // Если все завершено и больше нечего обрабатывать - выходим
            if (shared_data->finished) {
                int all_processed = 1;
                for (int i = 0; i < shared_data->count; i++) {
                    if (shared_data->lengths[i] > 11 &&
!shared_data->processed_by_child2[i]) {
                        all_processed = 0;
                        break;
                    }
                }
                if (all_processed) break;
            }

            // Если ничего не обработали в этой итерации - небольшая пауза
            if (!processed_something) {
                usleep(10000); // 10ms
            }
        }

        fclose(f);
```

```
    munmap(shared_data, sizeof(shared_data_t));

    return 0;

}
```

Протокол работы программы

```
execve("./parent", ["./parent"], 0x7ffef32d6410 /* 34 vars */) = 0
brk(NULL)                               = 0x557788396000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f2656d49000
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=11939, ...}) = 0
mmap(NULL, 11939, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f2656d46000
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) = 0x7f2656b34000
mmap(0x7f2656b5c000, 1605632, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x7f2656b5c000
mmap(0x7f2656ce4000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7f2656ce4000
mmap(0x7f2656d33000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1fe000) = 0x7f2656d33000
mmap(0x7f2656d39000, 52624, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
-1, 0) = 0x7f2656d39000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f2656b31000
arch_prctl(ARCH_SET_FS, 0x7f2656b31740) = 0
set_tid_address(0x7f2656b31a10)         = 8278
set_robust_list(0x7f2656b31a20, 24)     = 0
```

```
rseq(0x7f2656b32060, 0x20, 0, 0x53053053) = 0
mprotect(0x7f2656d33000, 16384, PROT_READ) = 0
mprotect(0x5577683ca000, 4096, PROT_READ) = 0
mprotect(0x7f2656d81000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f2656d46000, 11939)           = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x1), ...}) = 0
getrandom("\x64\x07\xc4\xf9\x16\x25\xe3\x02", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x557788396000
brk(0x5577883b7000)                     = 0x5577883b7000
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265\320\270\320\274\321\217 1-\320\263\320\276 \321\204\320"..., 40) = 40
fstat(0, {st_mode=S_IFREG|0644, st_size=104, ...}) = 0
read(0, "file1.txt\nfile2.txt\n1234567890\n1"..., 4096) = 104
write(1, "\320\222\320\262\320\265\320\264\320\270\321\202\320\265\320\270\320\274\321\217 2-\320\263\320\276 \321\204\320"..., 40) = 40
openat(AT_FDCWD, "/dev/shm/string_processor", O_RDWR|O_CREAT|O_NOFOLLOW|O_CLOEXEC, 0666) = 3
ftruncate(3, 411208)                    = 0
mmap(NULL, 411208, PROT_READ|PROT_WRITE, MAP_SHARED, 3, 0) = 0x7f2656acc000
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f2656b31a10) = 8279
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0x7f2656b31a10) = 8280
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\273"..., 63) = 63
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\273"..., 63) = 63
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\273"..., 63) = 63
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\273"..., 63) = 63
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\273"..., 63) = 63
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\273"..., 63) = 63
wait4(-1, NULL, 0, NULL)                = 8279
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=8279, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
wait4(-1, NULL, 0, NULL)                = 8280
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=8280, si_uid=1000, si_status=0, si_utime=0, si_stime=0} ---
```

```
munmap(0x7f2656acc000, 411208)          = 0
unlink("/dev/shm/string_processor")     = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

Я научился организовывать общую память для межпроцессного взаимодействия. Понял, как взаимодействуют между собой процессы, как обеспечить их корректную взаимосвязанную работу. Было неочевидно, как организовать структуру памяти, к которой обращаются процессы, чтобы обеспечить безопасную обработку информации. Такой подход позволяет быстрее производить обработку информации, она не копируется для передачи, в отличии от pipe.