Московский Авиационный Институт
(Национальный Исследовательский Университет)
Факультет информационных технологий и прикладной математики
Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу
«Операционные системы»

Студент: Скрипачев Фёдор Михайлович
Группа: М8О-209Б-23
Вариант: 16
Преподаватель: Миронов Евгений Сергеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2025

Содержание

Репозиторий

https://github.com/gthcbr25/osi/tree/main/oslab3

Постановка задачи

Цель работы

Приобретение практических навыков в:
· Освоение принципов работы с файловыми системами
· Обеспечение обмена данных между процессами посредством технологии «File mapping»

Задание

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов.
Взаимодействие между процессами осуществляется через системные сигналы/события и/или через отображаемые файлы (memory-mapped files).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.
Вариант 16) Правило проверки: строка должна оканчиваться на «.» или «;»

Общие сведения о программе

Программа компилируется из файла main.c. Также используется заголовочные файлы: unistd.h, stdio.h , stdlib.h, fcntl.h, errno.h, sys/mman.h, sys/stat.h, string.h, stdbool.h, ctype.h, sys/wait.h, semaphore.h. В программе используются следующие системные вызовы:

1. shm_open - создаёт/открывает объекты общей памяти POSIX.
2. sem_open - инициализирует и открывает именованный семафор.
3. ftruncate - обрезает файл до заданного размера.
4. mmap, munmap - отображает файлы или устройства в памяти, или удаляет их отображение.
5. memset - заполнение памяти значением определённого байта.
6. sem_getvalue - возвращает значение семафора.
7. close - закрывает файловый дескриптор.
8. sem_close - закрывает именованный семафор.
9. execl - запуск файла на исполнение.

10. sem_getvalue - возвращает значение семафора.
11. sem_wait - блокирует семафор.
12. sem_post - разблокирует семафор.

Общий метод и алгоритм решения

Для реализации, поставленной задачи необходимо:

1. Изучить принцип работы с mmap.
2. Изучить принцип работы с fork.
3. Создать 2 дочерних и 1 родительский процесс.
4. В каждом процессе отобразить файл в память, преобразовать в соответствии с вариантом и снять отображение (mmap, munmap).

Исходный код

parent.c

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

#define INITIAL_SIZE 128

char *read_lines() {
    char *line = malloc(INITIAL_SIZE);
    if (!line) {
        perror("Allocation error");
```

```
        exit(EXIT_FAILURE);
    }

    size_t size = INITIAL_SIZE;
    size_t length = 0;
    int ch;

    while ((ch = getchar()) != EOF) {
        if (length + 1 >= size) {
            size *= 2;
            char *new_line = realloc(line, size);
            if (!new_line) {
                perror("Reallocation error");
                free(line);
                exit(EXIT_FAILURE);
            }
            line = new_line;
        }
        line[length++] = ch;
    }

    if (length == 0 && ch == EOF) {
        free(line);
        return NULL;
    }

    line[length] = '\0';
    return line;
}
```

```
int main(int argc, char* argv[]) {
    const char* back_name = "Lab3.back";
    unsigned perms = S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH;

    char* input_data = NULL;
    char* file_name = NULL;

    size_t size = 0;
    printf("Enter filename: ");
    getline(&file_name, &size, stdin);

    printf("Enter strings: ");
    input_data = read_lines();

    size_t input_len = strlen(input_data);
    if (input_data[input_len - 1] == '\n') {
        input_data[input_len - 1] = '\0';
        input_len--;
    }

    size_t map_size = strlen(file_name) + 1 + input_len + 1;

    int fd = shm_open(back_name, O_RDWR | O_CREAT, perms);
    if (fd == -1) {
        perror("SHM_OPEN");
        free(input_data);
        exit(EXIT_FAILURE);
    }
```

```
    if (ftruncate(fd, map_size) == -1) {
        perror("FTRUNCATE");
        close(fd);
        free(input_data);
        exit(EXIT_FAILURE);
    }

    char* memptr = mmap(NULL, map_size, PROT_READ | PROT_WRITE,
MAP_SHARED, fd, 0);
    if (memptr == MAP_FAILED) {
        perror("MMAP");
        close(fd);
        free(input_data);
        exit(EXIT_FAILURE);
    }

    snprintf(memptr, map_size, "%s|%s", file_name, input_data);

    pid_t cpid = fork();

    if (cpid == -1) {
        perror("FORK");
        munmap(memptr, map_size);
        close(fd);
        free(input_data);
        exit(EXIT_FAILURE);
    }
```

```
    if (cpid == 0) {
        munmap(memptr, map_size);
        close(fd);
        execl("./build/child", "child", NULL);
        perror("EXECL");
        exit(EXIT_FAILURE);
    } else {
        int status;
        wait(&status);
        free(input_data);
        munmap(memptr, map_size);
        close(fd);

        if (WIFEXITED(status) && WEXITSTATUS(status) == 0) {
            printf("Child process completed successfully.\n");
            exit(EXIT_SUCCESS);
        } else {
            fprintf(stderr, "Child process failed.\n");
            exit(EXIT_FAILURE);
        }
    }
}
```

Child.c

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

```
#include <fcntl.h>
#include <unistd.h>

void write_lines_ending_with_dot(const char *input_data, const char *filename) {
    FILE *file = fopen(filename, "w");
    if (!file) {
        perror("Failed to open file");
        return;
    }

    const char *current_line = input_data;

    while (*current_line) {
        const char *line_end = strchr(current_line, '\n');
        if (!line_end) {
            line_end = current_line + strlen(current_line);
        }

        if ((line_end > current_line && *(line_end - 1) == '.') | (line_end > current_line && *(line_end - 1) == ';')) {
            fwrite(current_line, 1, line_end - current_line, file);
            fputc('\n', file);
        }

        current_line = line_end;
        if (*current_line == '\n') {
            current_line++;
        }
    }
```

```
    fclose(file);
}

int main() {
    const char* back_name = "Lab3.back";
    unsigned perms = S_IWUSR | S_IRUSR | S_IRGRP | S_IROTH;

    int map_fd = shm_open(back_name, O_RDWR, perms);

    if (map_fd < 0) {
        perror("SHM_OPEN");
        exit(EXIT_FAILURE);
    }

    struct stat statbuf;
    if (fstat(map_fd, &statbuf) == -1) {
        perror("FSTAT");
        close(map_fd);
        exit(EXIT_FAILURE);
    }

    size_t map_size = statbuf.st_size;

    char* memptr = mmap(NULL, map_size, PROT_READ | PROT_WRITE,
MAP_SHARED, map_fd, 0);
    if (memptr == MAP_FAILED) {
        perror("MMAP");
```

```
        close(map_fd);
        exit(EXIT_FAILURE);
    }


    close(map_fd);


    char filename[20] = {0};
    size_t readed_data_id = 0;

    // Разделяем имя файла и данные
    for (size_t i = 0; i < map_size; i++) {
        if (memptr[i] != '|') {
            filename[i] = memptr[i];
        } else {
            readed_data_id = i + 1;
            break;
        }
    }

    char* input_data = memptr + readed_data_id;

    write_lines_ending_with_dot(input_data, filename);

    munmap(memptr, map_size);
    return 0;
}
```

Демонстрация работы программы

./parent
Insert file name: 1.txt
Insert strings: qwerty!
Hello;
Hi.
World
^D
1.txt
Hello;
Hi.

Выводы

В Си помимо механизма общения между процессами через pipe, также существуют и другие способы взаимодействия, например отображение файла в память, такой подход работает быстрее, за счет отсутствия постоянных вызовов read, write и тратит меньше памяти под кэш. После отображения возвращается void*, который можно привести к своему указателю на тип и обрабатывать данные как массив, где возвращенный указатель – указатель на первый элемент.