Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №4 по курсу**

**«Операционные системы»**

Студент: Садаков А.А.

Группа: М8О–206Б–19

Вариант: 15

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**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.

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.

Правило проверки: строка должна начинаться с заглавной буквы.

**Общие сведения о программе**

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

1. **mmap** – создает отображение файла в память.
2. **munmap** – снимает отображение.
3. **open** – открывает файл.
4. **close –** закрывает файл.
5. **sem\_init –** инициализация семафора.
6. **sem\_wait** – ожидание доступа, если значение семафора отрицательное, то вызывающий поток блокируется до тех пор, пока один из потоков не вызовет sem\_post.
7. **sem\_post** – увеличивает значение семафора и разблокирует ожидающие потоки.
8. **sem\_destroy** – уничтожает семафор.
9. **read** – чтение из файла в буфер.
10. **write** – запись из буфера в файл.
11. **sleep** – переход в режим ожидания на указанное количество секунд.
12. **exit** – завершение работы программы с некоторым статусом.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить работу с отображением файла в память(mmap и munmap).
2. Вспомнить работу с процессами(fork).
3. Создать 1 дочерний и 1 родительский процесс.
4. В каждом процессе отобразить файл в память, преобразовать в соответствии с вариантом и снять отображение(mmap, munmap).

**Основные файлы программы**

**main.c:**

**#include <stdio.h>**

**#include <string.h>**

**#include <unistd.h>**

**#include <sys/wait.h>**

**#include <stdbool.h>**

**#include <fcntl.h>**

**#include <sys/mman.h>**

**#include <errno.h>**

**const int FILE\_LENGHT = 20;**

**const int STR\_LENGHT = 100;**

**//родительский процесс**

**void parent (char\* data, int fd) {**

**char \*errorMess = "ERROR";**

**while (1) {**

**char str[100] = {'\0'};**

**scanf("%s", str);**

**int size = strlen(str);**

**data[0] = size;**

**data[1] = -1;**

**memcpy(data + 2, str, size + 1);**

**if (size == 0) {**

**data[2] = '\n';**

**break;**

**}**

**dup2(fd, 1);**

**while (data[1] != 1 && data[1] != 0) {}**

**if (data[1] == 0) {**

**printf("%s\n", errorMess);**

**} else {**

**printf("%s\n", str);**

**}**

**}**

**wait(NULL);**

**close(fd);**

**}**

**//дочерний процесс**

**void child (char\* data) {**

**while (1) {**

**while (data[2] == '\0'){}**

**bool result;**

**char\* str = data + 2;**

**int size = data[0];**

**if (size == 0) {**

**break;**

**}**

**if(str[0] < 'A' || str[0] > 'Z'){**

**data[1] = 0;**

**} else {**

**data[1] = 1;**

**}**

**data[2] = '\0';**

**}**

**}**

**int main () {**

**char memfile[FILE\_LENGHT];**

**char file[FILE\_LENGHT];**

**printf("Enter name of memory file\n");**

**scanf("%s", memfile);**

**printf("Enter name of out file\n");**

**scanf("%s", file);**

**int out\_fd = open(file, O\_CREAT | O\_WRONLY, S\_IWUSR | S\_IRUSR);**

**if (out\_fd == -1) {**

**fprintf(stderr, "Can't open file\n");**

**return -1;**

**}**

**int mem\_fd = open(memfile, O\_CREAT | O\_RDWR, S\_IWUSR | S\_IRUSR);**

**if (mem\_fd == -1) {**

**fprintf(stderr, "Can't open file\n");**

**return -1;**

**}**

**int pid;**

**if (ftruncate(mem\_fd, STR\_LENGHT)) {**

**perror("ftruncate");**

**return -1;**

**}**

**char\* dataPtr = (char\*) mmap(NULL, STR\_LENGHT + 3, PROT\_READ | PROT\_WRITE, MAP\_SHARED, mem\_fd, 0); //Создаем отображение файла в память**

**dataPtr[2] = '\0';**

**close(mem\_fd);**

**if (dataPtr == MAP\_FAILED) {**

**perror("Map");**

**return -1;**

**}**

**pid = fork();**

**if (pid == -1) {**

**perror("FORK");**

**} else if (pid == 0) {**

**child(dataPtr);**

**} else {**

**parent(dataPtr, out\_fd);**

**}**

**munmap(dataPtr, STR\_LENGHT);**

**close(out\_fd);**

**return 0;**

**}**

**Пример работы**

**Тест 1 (пустой файл):**

aleksandr201@LAPTOP-M1VJISD6:~/OS/lab4$ ./a.out < test1

Enter name of memory file

Enter name of out file

Can't open file

**Тест 2 (обычный файл):**

aleksandr201@LAPTOP-M1VJISD6:~/OS/lab4$ cat test2

memf

out

Aaefaf

aAFAEFEF

AAAAAAA

aaAaefaef

aleksandr201@LAPTOP-M1VJISD6:~/OS/lab4$ ./a.out < test2

Enter name of memory file

Enter name of out file

aleksandr201@LAPTOP-M1VJISD6:~/OS/lab4$ cat out

Aaefaf

ERROR

AAAAAAA

ERROR

**Вывод**

Отображаемая память позволяет различным процессам общаться через общедоступный файл. Отображаемая память может использоваться для взаимодействия процессов или как простой способ для обращения к содержимому файла.