Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**РАБОТА С ОБЩЕЙ ПАМЯТЬЮ**

Студент: Комбаров Владислав Александрович

Группа: М8О–212Б–22

Вариант: 11

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

Целью является приобретение практических навыков в:

* Освоении принципов работы с файловыми системами
* Обеспечении 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 решения 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 11:** Child1 переводит строки в верхний регистр. Child2 превращает все пробельные символы в символ «\_».

**Общие сведения о программе**

Основной файл программы - main.c; child1.c и child2.c – файлы дочерних процессов, в которых осуществляется обработка строк из входного файла. Также используется заголовочные файлы: stdio.h, stdlib.h, string.h, sys/wait.h, unistd.h, fcntl.h, sys/mman.h, semaphore.h, ctype.h. В программе используются следующие системные вызовы:

1. **fork** – создание процесса
2. **ftruncate** – изменяет размер файла до заданного значения
3. **open** – открывает или создает файл в зависимости от его наличия
4. **mmap –** отображает файлы в память
5. **sem\_open –** открывает или создает семафор в зависимости от контекста
6. **sem\_post** – увеличивает семафор на 1
7. **sem\_wait –** ждет возможности уменьшить семафор или уменьшает его на 1 с последующей блокировкой для других потоков в зависимости от контекста
8. **sem\_close –** закрывает семафор
9. **munmap** – снимает отображение файла в память
10. **close( )** – закрывает то, что указано в скобках

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы таких явлений, как отображение файла в память и семафоры; установить, каким образом должны быть связаны процессы (дочерние и родительский) в условиях memory mapped files.
2. Написать child1.c и child2.c, которые будут открывать mapped file и обрабатывать данные в нём.
3. Использовать семафор для корректной работы программы, в том числе в файлах child1.c и child2.c
4. Написать файл main.c, в котором будет создаваться mapped file и где содержимое файла с исходными данными будет вводиться, обрабатываться и выводиться на экран, а также в специальный файл output.

**Основные файлы программы**

**main.c:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/wait.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <semaphore.h>

void create\_two\_processes(int\* a, int\* b) {

\*a = fork();

if (\*a > 0) {

\*b = fork();

}

if (\*a == -1 || \*b == -1) {

perror("fork error\n");

exit(-1);

}

}

int main(int argc, char \*argv[]) {

if (argc != 2) {

printf("Wrong number of args\n");

exit(-1);

}

FILE \*f = fopen("output.txt","w");

fclose(f);

int parent\_pid = getpid(), pid1, pid2;

// Opening of the mapped file

int fd = open(argv[1], O\_RDWR | O\_CREAT | O\_APPEND, 0644);

if (fd == -1) {

perror("Can't open file\n");

exit(-1);

}

ftruncate(fd, 1000);

size\_t file\_size = lseek(fd, 0, SEEK\_END);

char\* mapped = mmap(NULL, file\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (mapped == MAP\_FAILED) {

perror("Map failed\n");

exit(-1);

}

// Reading test file

char c;

int i = 0;

while ((c = getchar()) != EOF) {

mapped[i] = c;

i++;

}

// Semaphore creating

sem\_t\* sem = sem\_open("s\_laba3", O\_CREAT, 0777, 0);

if (sem == SEM\_FAILED) {

perror("Semaphore failed\n");

exit(-1);

}

sem\_post(sem);

create\_two\_processes(&pid1, &pid2); // Create Child1 & Child2 processes

// Parent process

if (pid1 > 0 && pid2 > 0) {

waitpid(pid1, NULL, WUNTRACED);

waitpid(pid2, NULL, WUNTRACED);

ftruncate(fd, file\_size);

printf("%s\n", mapped);

}

// Child1 process

else if (pid1 == 0) {

execl("./child1", "./child1", argv[1], "/s\_laba3", NULL);

}

// Child2 process

else {

execl("./child2", "./child2", argv[1], "/s\_laba3", NULL);

}

sem\_close(sem);

sem\_unlink("/s\_laba3");

printf("The result was written to %s\n", argv[1]);

munmap(mapped, file\_size);

close(fd);

return 0;

}

**child1.с:**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <semaphore.h>

int main(int argc, char\*\* argv) {

if (argc != 3) {

printf("Wrong number of args");

exit(-1);

}

int fd = open(argv[1], O\_RDWR, 0644);

if (fd == -1) {

perror("Can't open file");

exit(-1);

}

size\_t file\_size = lseek(fd, 0, SEEK\_END);

char\* mapped = mmap(NULL, file\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (mapped == MAP\_FAILED) {

perror("Map failed");

exit(-1);

}

sem\_t\* sem = sem\_open(argv[2], 0);

if (sem == SEM\_FAILED) {

perror("Semaphore 1 failed");

exit(-1);

}

sem\_wait(sem);

int i = 0;

while (i < file\_size) {

if (mapped[i] != EOF && mapped[i] != '\n' && mapped[i] != '\0') {

mapped[i] = toupper(mapped[i]);

}

i++;

}

sem\_post(sem);

sem\_close(sem);

munmap(mapped, file\_size);

close(fd);

return 0;

}

**child2.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <semaphore.h>

int main(int argc, char\*\* argv) {

if (argc != 3) {

printf("Wrong number of args");

exit(-1);

}

int fd = open(argv[1], O\_RDWR, 0644);

if (fd == -1) {

perror("Can't open file");

exit(-1);

}

size\_t file\_size = lseek(fd, 0, SEEK\_END);

char\* mapped = mmap(NULL, file\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd, 0);

if (mapped == MAP\_FAILED) {

perror("Map failed\n");

exit(-1);

}

sem\_t\* sem = sem\_open(argv[2], 0);

if (sem == SEM\_FAILED) {

perror("Semaphore 2 failed\n");

exit(-1);

}

sem\_wait(sem);

int i = 0;

while (i < file\_size) {

if (mapped[i] == ' ') {

mapped[i] = '\_';

}

i++;

}

sem\_post(sem);

sem\_close(sem);

munmap(mapped, file\_size);

close(fd);

return 0;

}

**Тесты:**

**File test31.txt:**

edtfyghj g

guhu

gbgyu fg

uubn

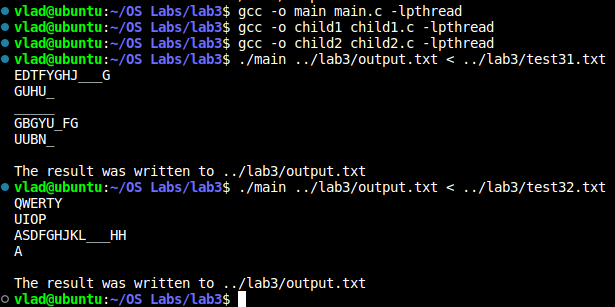
**File test32.txt:**

qwerty

uiop

asdfghjkl hh

a

**Результаты**:  


**Вывод**

Проделав работу, я успешно реализовал передачу строк между процессами через mapped file. Я получил практические навыки работы с общей памятью в ОС UNIX, а также понял, как использовать mapped files для обеспечения эффективного доступа к данным. Одной из основных проблем являлась инициализация дочерних процессов, так как в 1 лабораторной я успешно сделал это с помощью функции fork(), а здесь этот приём не получился. Я узнал об инициализации с помощью указателей и ссылок, когда данные о процессе меняются прямо в памяти, а идентификаторы получаются путём разыменовывания указателей. Было не совсем понятно, когда менять значение семафора и в скольких файлах использовать munmap в конце. Небольшие проблемы были с определением размера файла (функцией lseek). Сначала я пытался использовать только один файл при запуске программы, но это было безуспешно – программа зацикливалась, начинала работать только после нажатия пробела или выводила о записи результата (в тот же файл, что и начальный) по нескольку раз. Потом я увидел метод, связанный с перенаправлением ввода из файла в программу и записью обработанных данных в другой файл. Таким образом, данные из файла отправляются в программу main, обрабатываются и записываются в файл output. Сначала было важно перед каждым запуском программы очищать файл output, чтобы он не засорялся данными из предыдущих запусков, но потом программа была переделана: в main функции добавились открытие на запись и сразу же закрытие файла output.txt, так мы добиваемся его очищения.