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: Юрков Евгений Юрьевич

Группа: М8О–212Б–22

Вариант: 14

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2023.

**Постановка задачи**

## Цель работы

Целью является 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 14:** Child1 переводит строки в нижний регистр. Child2 убирает все задвоенные пробелы.

**Общие сведения о программе**

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

1. **fork** – создание процесса.
2. **ftruncate** – обрезает файл доя заданного размера.
3. **open** – открывает и создает файл.
4. **mmap –** отражеает файлы или устройства в памяти.
5. **sem\_open –** открывает и создает семафор.
6. **sem\_post** – увеличивает семафор.
7. **sem\_wait –** ждет возможности уменьшить семафор.
8. **sem\_close –** закрывает семафор.
9. **munmap** – снимает отражение файла в памяти.
10. **close** – закрывает файл.

**Общий метод и алгоритм решения**.

Для реализации поставленной задачи необходимо:

1. Изучить принципы работы mmap, munmap, sem\_open, sem\_post, sem\_wait, sem\_close.
2. Написать child1.c и child2.c, которые будут открывать mapped file и обрабатывать данные в нем.
3. Использовать семафор для корректной работы программы
4. Написать файл main.cpp, в котором будет создаваться mapped file и вводиться строка.

**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\_process(int\* a, int\* b) {   
    \*a = fork();   
    if (\*a > 0) {   
        \*b = fork();   
    }   
    if (\*a == -1 || \*b == -1) {   
        perror("fork error");   
        exit(-1);   
    }   
    return;   
}   
   
int main(int argc, char \*argv[]) {   
    if (argc < 2) {   
        printf("Using {executed file} {name of the shared file}");   
        exit(-1);   
    }   
   
    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");   
        exit(-1);   
    }   
    ftruncate(fd, 1024);   
    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);   
    }   
   
    // Reading of the string s   
   
    int i = 0;   
    do {   
        char c;   
        c = getchar();   
        mapped[i] = c;   
        ++i;   
    } while (mapped[i - 1] != EOF && mapped[i - 1] != '\n');   
   
    // creating a semaphore   
   
    sem\_t\* sem = sem\_open("/sem\_lab3", O\_CREAT, 0777, 0);   
    if (sem ==SEM\_FAILED) {   
        perror("Semaphore failed");   
        exit(-1);   
    }   
    sem\_post(sem);   
   
    create\_two\_process(&pid1, &pid2); // create child1 & child2   
   
    if (pid1 > 0 && pid2 > 0) { // Parent process   
        waitpid(pid1, NULL, WUNTRACED);   
        waitpid(pid2, NULL, WUNTRACED);   
   
        ftruncate(fd, strlen(mapped));   
        printf("%s\n", mapped);   
    }   
    else if (pid1 == 0) { // Child1 process   
        execl("./child1", "./child1", argv[1], "/sem\_lab3");   
    }   
    else { // Child2 process   
        execl("./child2", "./child2", argv[1], "/sem\_lab3");   
    }   
   
    sem\_close(sem);   
    sem\_unlink("/sem\_lab3");   
   
    printf("The result was written to %s\n", argv[1]);   
   
    munmap(mapped, strlen(mapped));   
    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("Using {executed file} {name of the shared file} {sem  
aphore name}");   
        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);   
   
    for (int i = 0; mapped[i] != '\0' && mapped[i] != EOF && mapped[  
i] != '\n'; ++i) {   
        mapped[i] = tolower(mapped[i]);   
    }   
   
    sem\_post(sem);    
   
    // printf("%s\n", mapped);   
   
    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("Using {executed file} {name of the shared file} {sem  
aphore name}");   
        exit(-1);   
    }   
   
    int fd = open(argv[1], O\_RDWR, 0644);   
    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 2 failed");   
        exit(-1);   
    }   
   
    sem\_wait(sem);   
   
    int j = 0;   
    for (int i = 0; mapped[i] != '\0' && mapped[i] != EOF && mapped[  
i] != '\n'; ++i) {   
        if (!(mapped[i] == ' ' && mapped[j - 1] == ' ')) {   
            mapped[j++] = mapped[i];   
        }   
    }   
    mapped[j] = '\0';   
       
    sem\_post(sem);   
   
    // printf("%s\n", mapped);   
   
    sem\_close(sem);   
    munmap(mapped, file\_size);   
    close(fd);   
   
    return 0;   
}

**Тесты:**

[kruyneg@matebook14 build]$ cat ../lab3/tests/test1.txt   
ABCDef   ghI   
[kruyneg@matebook14 build]$ cat ../lab3/tests/test2.txt   
ThE QuiCk   BrowN FoX JuMpS  OVER   the  LaZy dOg   
[kruyneg@matebook14 build]$ cat ../lab3/tests/test3.txt   
I  LOVE TO EAT PIZZA    AND  play video games.

**Результат**:  
[kruyneg@matebook14 build]$ ./main3 ../lab3/tests/output.txt < ../lab3/tests/test1.txt   
abcdef ghi   
The result was written to ../lab3/tests/output.txt   
[kruyneg@matebook14 build]$ ./main3 ../lab3/tests/output.txt < ../lab3/tests/test2.txt   
the quick brown fox jumps over the lazy dog   
The result was written to ../lab3/tests/output.txt   
[kruyneg@matebook14 build]$ ./main3 ../lab3/tests/output.txt < ../lab3/tests/test3.txt   
i love to eat pizza and play video games.   
The result was written to ../lab3/tests/output.txt

**Вывод**

Проделав работу, я успешно реализовал передачу строки 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 необходимость синхронизации доступа к общим данным. Чтобы решить её я использовал семафор.

Отображемые файлы используются для установления соответствия между адресным пространством процесса и содержимым файла. Они позволяют программе работать с файлом, как если бы он находился в памяти, обеспечивая прозрачный доступ к данным. Преимуществом mapped file также является удобство работы с большими файлами, благодаря доступу по индексу. Они дают возможность обмениваться данными между процессами без необходимости копирования их содержимого.