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: Шандрюк Пётр Николаевич

Группа: М8О-208Б-20

Вариант: 11

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/Peter1811/OS/tree/main/lab5

**Постановка задачи**

**Цель работы**

Приобретение практических навыков в:

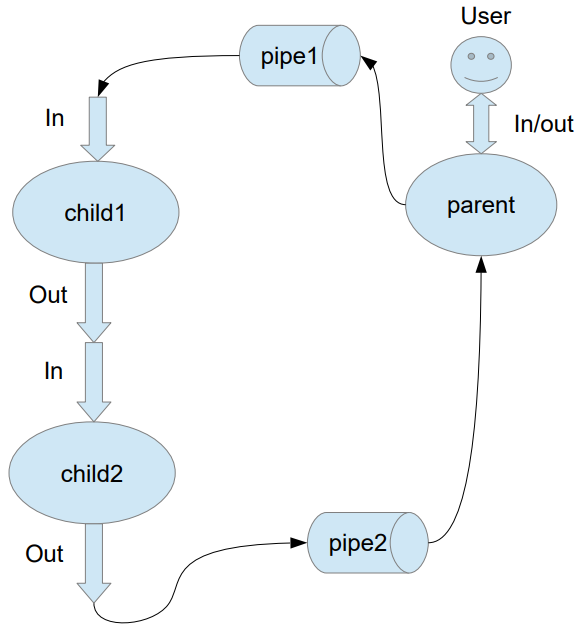
· Освоение принципов работы с файловыми системами

· 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.



12 вариант) Child1 переводит строки в верхний регистр. Child2 преобразует все пробелы в знаки нижнего подчеркивания

**Общие сведения о программе**

Программа компилируется из файла main.cpp. 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 работу с отображением файла в память(mmap и munmap).
2. Изучить работу с процессами(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**

main.cpp

#include <unistd.h>

#include <iostream>

#include <string>

#include <stdlib.h>

#include <errno.h>

#include <fcntl.h>

#include <sys/mman.h>

#include <sys/stat.h>

#include <string.h>

#include <stdbool.h>

#include <ctype.h>

#include <sys/wait.h>

#include <semaphore.h>

using namespace std;

void change\_spaces(char\* src, int size) {

int j = 0;

for(int i = 0; i < size ; ++i) {

if(src[i] == ' ')

src[i] = '\_';

}

}

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

if(argc != 2) {

printf("INVALID COUNT OF ARGS\nUSAGE: %s <file>\n", argv[0]);

exit(-1);

}

int fd\_0 = -1;

int fd\_1 = -1;

int fd\_2 = -1;

int fd\_3 = -1;

char\* src;

char\* src2;

char\* src3;

char\* src4;

char\* src5;

struct stat statbuf;

string s;

FILE\* f1 = fopen("file.txt", "r");

fd\_0 = fileno(f1);

// printf("%d", fd\_1);

char c;

if((fd\_1 = open("file1.txt", O\_CREAT | O\_RDWR, S\_IRUSR | S\_IWUSR)) < 0) {

printf("OPEN ERROR\n");

exit(-1);

}

if((fd\_2 = open("file2.txt", O\_CREAT | O\_RDWR, S\_IRUSR | S\_IWUSR)) < 0) {

printf("OPEN ERROR\n");

exit(-1);

}

if((fd\_3 = open("file3.txt", O\_CREAT | O\_RDWR, S\_IRUSR | S\_IWUSR)) < 0) {

printf("OPEN ERROR\n");

exit(-1);

}

if(fstat(fd\_0, &statbuf) < 0) {

printf("FSTAT ERROR\n");

exit(-1);

}

if(ftruncate(fd\_1, statbuf.st\_size) < 0) {

printf("FTRUNCATE ERROR\n");

exit(-1);

}

if(ftruncate(fd\_2, statbuf.st\_size) < 0) {

printf("FTRUNCATE ERROR\n");

exit(-1);

}

if(ftruncate(fd\_3, statbuf.st\_size) < 0) {

printf("FTRUNCATE ERROR\n");

exit(-1);

}

// char buff[statbuf.st\_size];

char\* buff = (char\*) malloc(sizeof(char)\*statbuf.st\_size);

if(read(fd\_0, buff, statbuf.st\_size) != statbuf.st\_size) {

printf("READ ERROR1\n");

exit(-1);

}

if(write(fd\_1, buff, statbuf.st\_size) != statbuf.st\_size) {

printf("READ ERROR\n");

exit(-1);

}

// for (int i = 0; i < statbuf.st\_size; i++) printf("%c", buff[i]);

int pid\_0 = 0;

int pid\_1 = 0;

int status\_0 = 0;

int status\_1 = 0;

sem\_t semaphore;

sem\_init(&semaphore, 0, 1);

if((pid\_0 = fork()) > 0) { //Parent

if((pid\_1 = fork()) > 0) { //Parent

// sem\_wait(&semaphore);

sleep(2);

waitpid(pid\_1, &status\_1, WNOHANG);

waitpid(pid\_0, &status\_0, WNOHANG);

src5 = (char\*)mmap(0, statbuf.st\_size, PROT\_READ, MAP\_SHARED, fd\_3, 0);

if(src5 == MAP\_FAILED) {

printf("MMAP ERROR5\n");

exit(-1);

}

for(int i = 0; i < statbuf.st\_size; ++i) { printf("%c", src5[i]); }

printf("\n");

if(munmap(src5, statbuf.st\_size) != 0) {

printf("MUNMAP ERROR\n");

exit(-1);

}

// sleep(2);

// sem\_post(&semaphore);

}

else if(pid\_1 == 0) { //Child2

sem\_wait(&semaphore);

// sleep(1);

src3 = (char\*)mmap(0, statbuf.st\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd\_2, 0);

if(src3 == MAP\_FAILED) {

printf("MMAP ERROR3\n");

exit(-1);

}

src4 = (char\*)mmap(0, statbuf.st\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd\_3, 0);

if(src4 == MAP\_FAILED) {

printf("MMAP ERROR4\n");

exit(-1);

}

change\_spaces(src3, statbuf.st\_size);

for (int i = 0; i < statbuf.st\_size; i++) src4[i] = src3[i];

if(munmap(src3, statbuf.st\_size) != 0) {

printf("MUNMAP ERROR\n");

exit(-1);

}

if(munmap(src4, statbuf.st\_size) != 0) {

printf("MUNMAP ERROR3\n");

exit(-1);

}

// sleep(1);

sem\_post(&semaphore);

}

else {

printf("FORK ERROR 1\n");

exit(-1);

}

}

else if (pid\_0 == 0) { //Child1

sem\_wait(&semaphore);

// sleep(1);

src = (char\*)mmap(0, statbuf.st\_size, PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd\_1, 0);

if(src == MAP\_FAILED) {

printf("MMAP ERROR1\n");

exit(-1);

}

src2 = (char\*)mmap(0, statbuf.st\_size, PROT\_WRITE, MAP\_SHARED, fd\_2, 0);

if(src2 == MAP\_FAILED) {

printf("MMAP ERROR2\n");

exit(-1);

}

for(int i = 0; i < statbuf.st\_size; ++i) { src2[i] = toupper(src[i]); }

if(munmap(src, statbuf.st\_size) != 0) {

printf("MUNMAP ERROR\n");

exit(-1);

}

if(munmap(src2, statbuf.st\_size) != 0) {

printf("MUNMAP ERROR\n");

exit(-1);

}

// sleep(1);

sem\_post(&semaphore);

}

else {

printf("FORK ERROR 2\n");

exit(-1);

}

remove("file1.txt");

remove("file2.txt");

remove("file3.txt");

sem\_destroy(&semaphore);

close(fd\_0);

close(fd\_1);

close(fd\_2);

close(fd\_3);

free(buff);

return 0;

}

**Демонстрация работы программы**

Для удобства исходные строки записываются в файл file.txt

**peter@DESKTOP-V53N291:$ cat test.txt**

**hhadghfdf fdgfdgf**

**sdfgsdg sdgfsg**

**sdfgsgsg sdfgdsg**

**peter@DESKTOP-V53N291:$ gcc** -pthread-lrt **main.cpp**

**peter@DESKTOP-V53N291:$ ./a.out**

**HHADGHFDF\_FDGFDGF**

**SDFGSDG\_SDGFSG\_\_\_\_**

**SDFGSGSG\_SDFGDSG\_\_\_\_\_\_\_\_\_\_**

**Выводы**

В С++ 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.