Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работ №4 по курсу**

**«Операционные системы»**

Студент: Кузьмичев А. Н.

Группа: М80 – 206Б-18

Вариант: 8

Преподаватель: Миронов Е. С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2019

**1. Постановка задачи**

На вход программе подается название 2-ух файлов. Необходимо отсортировать оба файла

(каждый в отдельном процессе) произвольной сортировкой (на усмотрение студента).

Родительским процессом вывести отсортированные файлы чередованием строк первого и

второго файлов.

Операционная система: MacOS.

**Целью лабораторной работы 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.

**2. Решение задачи**

**Используемые системные вызовы:**

**• fork(void)** - создаёт новый процесс посредством копирования вызывающего

процесса. Новый процесс считается дочерним процессом. Вызывающий процесс

считается родительским процессом. Дочерний и родительский процессы находятся в

отдельных пространствах памяти. Сразу после **fork()** эти пространства имеют одинаковое

содержимое.

• **int close()** - закрывает файловый дескриптор.

• \_**exit(int status)** - все дескрипторы файлов, принадлежащие процессу, закрываются

pid = fork(); Начиная с этого момента, процессов становится два. У каждого своя память. в

процессе-родителе pid хранит идентификатор ребёнка. в ребёнке в этой же переменной

лежит 0. Далее в каждом случае надо закрыть "лишние" концы потоков. поскольку сама

программа теперь существует в двух экземплярах, то фактически у каждого потока

появляются вторые дескрипторы.

**• ptr1 = mmap(NULL, sizeof(char), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd1,**

**0)** - вызов mmap позволяет отобразить открытый файл в адресное пространство

процесса. Первый аргумент является нулевым указателем, при этом система сама

выбирает адрес начала отображаемого сегмента. Длина файла совпадает с размером

char. Устанавливается доступ на чтение и запись. Четвертый аргумент имеет

значение MAP\_SHARED, что позволяет процессам «видеть» изменения, вносимые

друг другом. Функция возвращает адрес начала участка разделяемой памяти, мы

сохраняем его в переменной ptr1.

Одним из способов добиться **совместного использования памяти** родительским и

дочерним процессами является вызов mmap с флагом MAP\_SHARED перед вызовом fork.

В этом случае все отображения памяти, установленные родительским процессом, будут

унаследованы дочерним. Более того, изменения в содержимом объекта, вносимые

родительским процессом, будут видны дочернему, и наоборот.

**ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ:**

Make sure all files are in the same folder.

Enter the name of the first file

warning: this program uses gets(), which is unsafe.

file1.txt

Enter the name of the second file

file2.txt

exit normally? true

child exitcode = 0

file1->

file2->

file1->1

file2->1

file1->4

file2->1

file1->5

file2->3

file1->5

file2->5

file1->6

file2->5

file1->6

file2->6

**3. Руководство по использованию программы**

Компиляция и запуск программного кода в MacOs High Sierra :

gcc -o main main.c

./main

**4. Листинг программы**

|  |
| --- |
|  |
|  |
| #include <stdio.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/wait.h>  #include <stdlib.h>  #include <sys/stat.h>  #include <fcntl.h>  #include <sys/mman.h>  void sortBooble(char\*massIn)//сортировка пузырьком  {  int tmp;  int noSwap;  int i=0;  int N =strlen(massIn);  for ( i= N - 1; i >= 0; i--)  {  noSwap = 1;  for (int j = 0; j < i; j++)  {  if (massIn[j] > massIn[j + 1])  {  tmp = massIn[j];  massIn[j] = massIn[j + 1];  massIn[j + 1] = tmp;  noSwap = 0;  }  }  if (noSwap == 1)  break;  }  }  int main()  {  pid\_t pid;//идентификатор потока  int fd1,fd2, i;  char fname1[50];  char fname2[50];  int fsize=0;  char \*ptr1,\*ptr2;  printf("Make sure all files are in the same folder.\nEnter the name of the first file\n");  gets(fname1);  // fname1="/Users/macbook/Desktop/OS\_2/file1.txt"  printf("\nEnter the name of the second file\n");  gets(fname2);  // fname2="/Users/macbook/Desktop/OS\_2/file2.txt"  //-------------------------------------------------------------------------отображение в память  первого файла  fd1 = open(fname1, O\_RDWR | O\_CREAT, DEFFILEMODE);  ptr1= mmap(NULL, sizeof(char), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd1, 0);//настройки для записи  и чтения ,доступ для всех потоков  //-------------------------------------------------------------------------------------------------  ----------  //-------------------------------------------------------------------------отображение в память  второго файла  fd2 = open(fname2, O\_RDWR | O\_CREAT, DEFFILEMODE);  ptr2 = mmap(NULL, sizeof(char), PROT\_READ | PROT\_WRITE, MAP\_SHARED, fd2, 0);//настройки для записи  и чтения ,доступ для всех потоков  //-------------------------------------------------------------------------------------------------  ----------  close(fd1);  close(fd2);  pid = fork();//создаем новый поток  if (pid == -1)//поток не создан  {  perror("fork");  exit(EXIT\_FAILURE);  }  if(pid==0)//если ребенок  {  sortBooble(ptr2);//сортируем данные  \_exit(EXIT\_SUCCESS);//выходим из потока  }  else  {  //-------------------------------------выводим данные о завершении дочернего потока  int status;  waitpid(pid, &status, 0);  printf("exit normally? %s\n", (WIFEXITED(status) ? "true" : "false"));  printf("child exitcode = %i\n", WEXITSTATUS(status));  //---------------------------------------------------------------------------------  sortBooble(ptr1);//сортируем текущие даннные родительского потока  int s\_data1= strlen(ptr1); //количество символов в файлах  int s\_data2 = strlen(ptr2);  int p;  if(s\_data1>s\_data2)  p=s\_data1;  else  p=s\_data2;  for( int i=0; i<p; i++)  {  if(i<s\_data1)  printf("file1->%c\n",\*(ptr1+i));  if(i<s\_data2)  printf("file2->%c\n",\*(ptr2+i));  }  wait(NULL); /\* Ожидание потомка \*/  \_exit(EXIT\_SUCCESS);  }  } |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**5. Вывод**

Отображение в память содержимого файла, который сначала открывается вызовом open, а

затем отображается вызовом mmap удобно тем, что все операции ввода-вывода

осуществляются ядром и скрыты от программиста, а он просто пишет код, считывающий и записывающий данные в некоторую область памяти. Ему не приходится вызывать read,

write . Часто это заметно упрощает код.