

Московский Авиационный Институт
(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа № 2
по курсу «Операционные системы»

Студент:	Чурсина Н. А.
Группа:	М8О-208Б-18
Вариант:	8
Преподаватель:	Миронов Е. С.
Оценка:	
Дата:	

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:
 - Управление процессами в ОС
 - Обеспечение обмена данных между процессами посредством каналов

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 каналы (pipe). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

2. Решение задачи

Используемые системные вызовы:

- **write(int fd, const void *buf, size_t count)** - пишет до *count* байт из буфера, на который указывает *buf*, в файл, на который ссылается файловый дескриптор *fd*. (записывает данные в канал обмена)

- **read(int fd, void *buf, size_t count)** - пытается прочитать *count* байт из файлового дескриптора *fd* в буфер, начинающийся по адресу *buf*. Для файлов, поддерживающих смещения, операция чтения начинается с текущего файлового смещения, и файловое смещение увеличивается на количество прочитанных байт. Если текущее файловое смещение находится за концом файла, то ничего не читается и **read()** возвращает ноль. Если значение *count* равно 0, то **read()** может обнаружить ошибки. При отсутствии ошибок, или если **read()** не выполняет проверки, то **read()** с *count* равным 0 возвращает 0 и ничего не меняет. (читает данные канала обмена)

- **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.

- **pipe()** - создаёт собственный поток обмена данными. При этом в передаваемый ей массив записываются числовые идентификаторы (дескрипторы) двух "концов" потока: один на чтение, другой на запись. Поток (pipe) работает по принципу "положенное первым - первым будет считано".

- **_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.

ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ:

Make sure all files are in the same folder.

Enter the name of the first file

file1.txt

Enter the name of the second file

file2.txt

file1.txt

file2.txt

exit normally? true

child exitcode = 0

file1->0

file2->4

file1->1

file2->6

file1->2

file2->7

file1->3

file2->8

file1->5

file2->9

3. Руководство по использованию программы

Компиляция и запуск программного кода в *MacOs Mojave* :

```
gcc -o main main.c
./main
```

4. Листинг программы

```
//вариант 8

#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <stdlib.h>

#include <string.h>

void sortBooble(char*massIn)

{

//сортировка пузырьком

    int tmp;

    int noSwap;

    int i;

    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

    char fname1[50];

    char fname2[50];

    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"

    char *filename;// указатель на путь к нужному файлу

    int file_pipes[2];//массив для хранения файловых дескрипторов

    char buffer[BUFSIZ + 1];//массив для чтения данных из канала

    char dataFile[100];//массив для хранения данных

    if (pipe(file_pipes) == -1)//если канал обмена не создан

    {

        perror("pipe");

        exit(EXIT_FAILURE);

    }

    pid = fork();//создаем новый поток


    if (pid == -1)//поток не создан

    {

        perror("fork");

        exit(EXIT_FAILURE);

    }

    if(pid == 0)//ребенок

    {

```

```

        filename = fname2;//будем читать данные со второго файла
    }

    else//главный поток

    {

        filename =fname1;//читаем данные из первого файла
    }

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

    //-----читаем данные из файла

    FILE * myFile;

    myFile = fopen(filename,"r");

    int i=0;

    fscanf(myFile, "%s", dataFile);

    //-----

    if(pid==0)//если ребенок

    {

        close(file_pipes[0]);//закрываем выход для чтения

        sortBooble(dataFile);//сортируем данные

        write(file_pipes[1], dataFile, strlen(dataFile));//записываем данные в канал обмена, dataFile -буффер

        _exit(EXIT_SUCCESS);//выходим из потока
    }

    else

    {

        close(file_pipes[1]);//закрываем выход для записи

        //-----выводим данные о завершении дочернего потока

        int status;

        waitpid(pid, &status, 0);

        printf("exit normally? %s\n", (WIFEXITED(status) ? "true" : "false")); //не равно нулю, если дочерний процесс успешно
        завершился

        printf("child exitcode = %i\n", WEXITSTATUS(status));

        //-----

        read(file_pipes[0], buffer, BUFSIZ);//читаем данные из канала обмена
    }

```

```

sortBooble(dataFile);//сортируем текущие данные родительского потока

int s_data= strlen(dataFile);

int s_buf = strlen(buffer);


int p;

if(s_data>s_buf){

    p=s_data;

} else {

    p=s_buf;

}

for( int i=0; i<p; i++)

{

    if(i<s_data)

        printf("file1->%c\n",dataFile[i]);

    if(i<s_buf)

        printf("file2->%c\n",buffer[i]);

}

wait(NULL);          /* Ожидание потомка */

fclose(myFile);

_exit(EXIT_SUCCESS); // все дескрипторы файлов, принадлежащие процессу, закрываются

}

}

```

5. Вывод

Современные программы редко работают в одном процессе или потоке. Довольно частая ситуация: нам необходимо запустить какую-то программу из нашей. Также многие программы создают дочерние процессы не для запуска другой программы, а для выполнения параллельной задачи. Например, так поступают простые сетевые серверы — при подсоединении клиента, сервер создаёт свою копию (дочерний процесс), которая обслуживает клиентское соединение и завершается по его закрытию. Родительский же процесс продолжает ожидать новых соединений.

При работе с потоками и процессами необходимо быть весьма осторожным, так как возможно допустить различные ошибки, которые затем будет сложно отловить и исправить.