ACS. OS. HW7

Горбачева Маргарита Валерьевна | БПИ-245

Задание:

Разработать программу использующую для работы с текстовыми файлами только системные вызовы. Программа на языке C (или C++ в стиле C) должна прочитать, используя буфер, размер которого превышает читаемые файлы и записать прочитанный файл в файл с другим именем. Имена файлов для чтения и записи задавать с использованием аргументов командной строки.

Вместо большого буфера использовать для работы с файлами буфер ограниченного размера равный 32 байт, требующий циклического использования как для чтения, так и для записи (+1 балл).

Читать и переписывать не только текстовые, но и исполняемые файлы (текстовые и бинарные, содержащие признак исполнимости), включая скрипты, которые сохраняют режим доступа исходных файлов, обеспечивающий их запуск. При этом обычные текстовые файлы запускаться не должны. Для них режим доступа должен оставаться прежним (+1 балл).

Решение на 10 баллов:

Код и отчёт на GitHub:
https://github.com/Misss-Lacoste/HSE-FCS-SE-2-course/tree/main/ACS/linux/Homeworks/HW7

Как работает программа:

1. **Проверяет, что в аргументы переданы названия файлов.** Программа проверяет: пришло ли 2 аргумента (имена файлов). Если нет — выводит ошибку и завершается.
2. **Открывает исходный файл** (открывает первый файл (из argv[1]) в режиме только для чтения (O_RDONLY).
3. **Получаем информацию о файле, чтобы знать его режим доступа** (читает метаданные исходного файла (включая права доступа) с помощью fstat()).
4. **Открываем файл для записи** (второй файл (из argv[2]) в режиме только для записи, с такими же правами, как у исходного файла).
5. **В цикле читает в буфер данные из первого файла и записываем не более, чем size символов во второй.**

6. **Закрывает оба файла** (закрывает дескрипторы файлов, проверяет на успешность закрытия). Завершение программы.

```
#include <stdio.h> //заголовочный файл для ввода-вывода(как iostream в C++)
#include <stdlib.h> //заголовочный файл для работы с памятью
#include <fcntl.h> //для функций open, readonly, writeonly
#include <sys/stat.h> //для системных вызовов read, write
#include <unistd.h> //для функций read(), write()

/*#define BUFFER 32*/
const int bufferSize = 32; //задаём константой буфера размер 32байта

/*#define BUFFER 32*/

int main(int argc, char* argv[]) //1) число command-line arguments; 2)сам vector of command-line arguments
{
   if(argc != 3) { //кол-во аргументов: программа + 2 файла = 3 штуки
      fprintf(stderr, "Wrong arguments entered.\n" /*, argv[0]*/);
      exit(1);
   }

   const char *FirstFile = argv[1]; //имя файла для чтения сохр. в 1й арг
   const char *SecondFile = argv[2]; //для записи файл, это уже 2й арг

   char buffer[bufferSize]; //временное хранение данных. создали буфер
   ssize_t read_bytes; //signed data type для представления байт объекта

   /*int src_fd = open(source_file, O_RDONLY);*/

   int file;
   if((file = open(FirstFile, O_RDONLY)) < 0) //открыли файл только для чтения, если не открыли,то ошибка
   {
      printf("Unable to open file in reading mode.\n");
      exit(1);
```

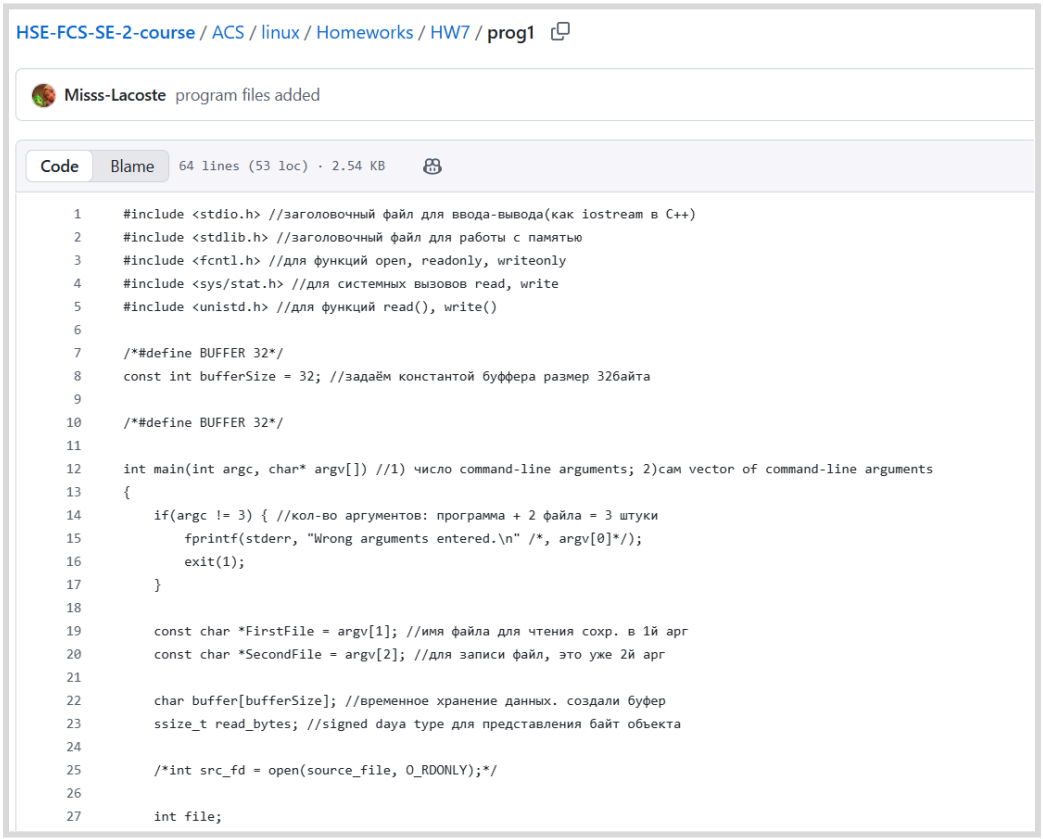
```
    }


    struct stat fileInfo;              //структура для хранения информации о файле
    if (fstat(file, &fileInfo) == -1) //информация о файле через дескриптор
    {
        printf("Unable to get file information.\n");
        close(file);
        exit(-1);
    }

    int newFile;
    if ((newFile = open(SecondFile, O_WRONLY | O_CREAT, fileInfo.st_mode)) < 0) {
        printf("Unable to open file for editing - writing.\n");
        close(file);
        exit(1);
    }

    read_bytes = read(file, buffer, bufferSize);
    while(read_bytes > 0) { //читаем, пока буфер не закончится
        if (read_bytes == 1) {
            printf("Unable to read file.\n");
            close(file);
            exit(1);
        }
        write(newFile, buffer, read_bytes);
        read_bytes = read(file, buffer, bufferSize);
    }
    if(close(newFile) < 0) {
        printf("Unable to close file.\n");
    }
    return 0;
}
```

Результат: запустим код и посмотрим на результат.

```
МАРГАРИТА@HP15-EH2064CI MINGW64 /c/HSE/HSE-FCS-SE-2-course/ACS/linux/Home
works/HW7 (main)
$ gcc code.c -o a.out

МАРГАРИТА@HP15-EH2064CI MINGW64 /c/HSE/HSE-FCS-SE-2-course/ACS/linux/Home
works/HW7 (main)
$ ./a.out code.c prog1

МАРГАРИТА@HP15-EH2064CI MINGW64 /c/HSE/HSE-FCS-SE-2-course/ACS/linux/Home
works/HW7 (main)
$ ./a.out a.out prog2

МАРГАРИТА@HP15-EH2064CI MINGW64 /c/HSE/HSE-FCS-SE-2-course/ACS/linux/Home
works/HW7 (main)
$ ./prog2 prog1 prog3
```

Объясним:

1. Скомпилируем программу стандартно.
2. В качестве аргумента передаем текст данной программы, запишем данные в файл prog1 . Файл правильно записался(скриншот с моего репо на GitHub):



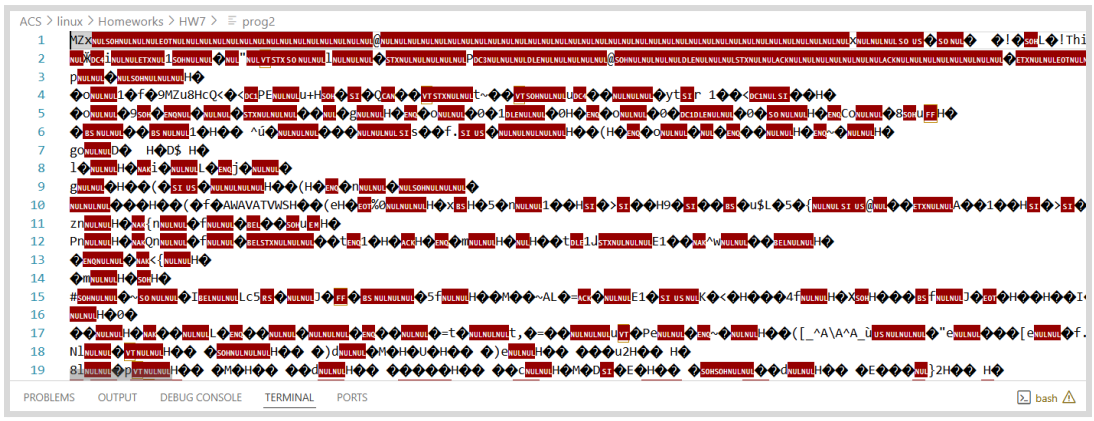
HSE-FCS-SE-2-course / ACS / linux / Homeworks / HW7 / **prog1**

Misss-Lacoste program files added

Code | Blame | 64 lines (53 loc) · 2.54 KB

```
#include <stdio.h> //заголовочный файл для ввода-вывода(как iostream в C++)
#include <stdlib.h> //заголовочный файл для работы с памятью
#include <fcntl.h> //для функций open, readonly, writeonly
#include <sys/stat.h> //для системных вызовов read, write
#include <unistd.h> //для функций read(), write()

/*#define BUFFER 32*/
const int bufferSize = 32; //задаём константой буффера размер 32байта

/*#define BUFFER 32*/

int main(int argc, char* argv[]) //1) число command-line arguments; 2)сам vector of command-line arguments
{
    if(argc != 3) { //кол-во аргументов: программа + 2 файла = 3 штуки
        fprintf(stderr, "Wrong arguments entered.\n" /*, argv[0]*/);
        exit(1);
    }

    const char *FirstFile = argv[1]; //имя файла для чтения сохр. в 1й арг
    const char *SecondFile = argv[2]; //для записи файл, это уже 2й арг

    char buffer[bufferSize]; //временное хранение данных. создали буфер
    ssize_t read_bytes; //signed daya type для представления байт объекта

    /*int src_fd = open(source_file, O_RDONLY);*/

    int file;
```

3. В качестве аргумента передадим исполняемый файл данного скрипта и выведем запишем данные в файл prog2. Согласно дз, такой файл как раз и должен отображаться нестандартно.



4. Теперь, чтобы проверить, что в прошлом тесте-запуске программа отработала корректно, запустим файл prog2 с аргументами prog1 и prog3 . Всё хорошо 🙂