

Операционные системы

ДЗ 4. Отчёт

Работа на 10 баллов

Фролов-Буканов Виктор Дмитриевич БПИ-228

13 февраля 2024

1 Код программы

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>

const int buffSizeLarge = 4096;
const int buffSizeSmall = 16;

int main(int argc, char* argv[]) {
    if (std::atoi(argv[1]) != 2) {
        printf("Wrong_input!\n");
        return 0;
    }

    struct stat fileStat;
    if (stat(argv[2], &fileStat) < 0) { // collecting information about
        the file in the variable
        printf("An_error_has_occurred_while_processing_the_file");
        exit(-1);
    }

    //int size = buffSizeLarge;
    int size = buffSizeSmall;

    int fdRead = open(argv[2], O_RDONLY);
    int fdWrite = open(argv[3], O_WRONLY | O_CREAT, fileStat.st_mode); //
        creating a new file with access rights of the first file

    char buffer[size];
    ssize_t readBytes;

    if (fdRead < 0) {
        printf("Cannot_open_the_read-file\n");
        exit(-1);
    }

    if (fdWrite < 0) {
        printf("Cannot_open_the_write-file\n");
        exit(-1);
    }

    if (size == buffSizeLarge) {
        readBytes = read(fdRead, buffer, size);
        write(fdWrite, buffer, readBytes);
    } else {
        do {
            readBytes = read(fdRead, buffer, size);
            if (readBytes < 0) {
                printf("Cannot_read_the_file!\n");
                exit(-1);
            }
        } while (readBytes > 0);
    }
}
```

```

    }
    write(fdWrite, buffer, readBytes);
} while (readBytes == size);
}

if (close(fdRead) < 0) {
    printf("Cannot_close_the_read-file");
    exit(-1);
}

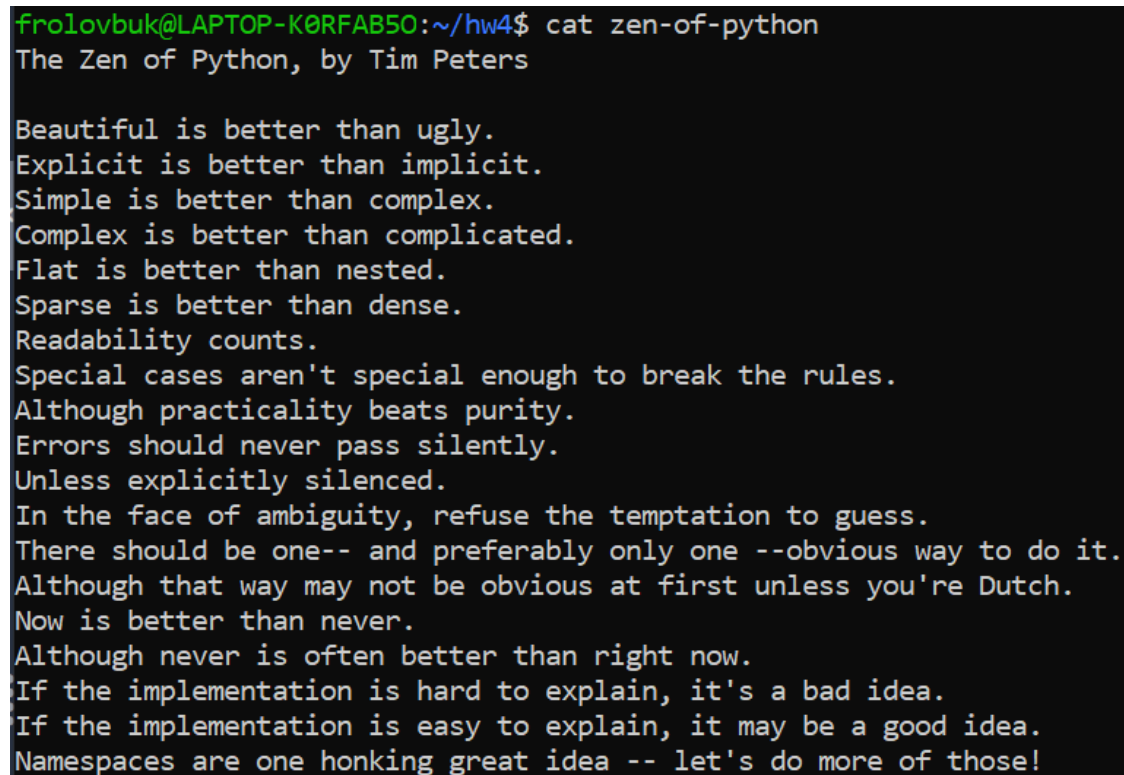
if (close(fdWrite) < 0) {
    printf("Cannot_close_the_write-file");
    exit(-1);
}

return 0;
}

```

2 Результат работы программы

В качестве примера текста, который мы будем копировать, я взял "Zen of Python": (:D)



```

frolovbuk@LAPTOP-KØRFAB50:~/hw4$ cat zen-of-python
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than right now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

```

Figure 1: 1 скрипт

В коде программы в глобальной области я объявил два размера буфера. Один заметно превышает размер текста, другой же слишком мал, так что чтение придется делать в цикле, пока не считаем весь файл. Оба метода работают верно на предоставленном файле, но в коде будем использовать копирование маленьким буфером в цикле, так как в будущем нам предстоит скопировать исполняемый файл текущей программы с сохранением прав доступа,

а размер этого исполняемого файла мы не знаем, так что использование разового буфера на 4096 байта может быть опасным для такой задачи

```
frolovbuk@LAPTOP-K0RFAB50:~/hw4$ ls
CMakeLists.txt  cmake-build-debug  main.cpp  zen-of-python
frolovbuk@LAPTOP-K0RFAB50:~/hw4$ g++ main.cpp
frolovbuk@LAPTOP-K0RFAB50:~/hw4$ ./a.out 2 zen-of-python ans
frolovbuk@LAPTOP-K0RFAB50:~/hw4$ ls
CMakeLists.txt  a.out  ans  cmake-build-debug  main.cpp  zen-of-python
frolovbuk@LAPTOP-K0RFAB50:~/hw4$ cat ans
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than right now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Figure 2: 2 скрипт

Для начала я показываю, что файла `ans` в текущей директории нет (командой `ls`). Далее компилирую файл и запускаю свою программу для исходного файла (первый параметр), а вторым параметром передаю `ans` - результирующий файл, куда я скопирую содержимое исходного. Командой `cat` показываю корректность выполнения программы

Теперь попробуем скопировать текущий исполняемый файл (`a.out`) - этот файл получился в результате компиляции

```

frolovbuk@LAPTOP-K0RFAB50:~/hw4$ ls
CMakeLists.txt  a.out  ans  cmake-build-debug  main.cpp  zen-of-python
frolovbuk@LAPTOP-K0RFAB50:~/hw4$ ./a.out 2 a.out b.out
frolovbuk@LAPTOP-K0RFAB50:~/hw4$ ls -l
total 60
-rw-r--r-- 1 frolovbuk frolovbuk 116 Feb 13 00:23 CMakeLists.txt
-rwxr-xr-x 1 frolovbuk frolovbuk 16432 Feb 13 03:28 a.out
-rw-r--r-- 1 frolovbuk frolovbuk 876 Feb 13 03:28 ans
-rwxr-xr-x 1 frolovbuk frolovbuk 16432 Feb 13 03:33 b.out
drwxr-xr-x 4 frolovbuk frolovbuk 4096 Feb 13 00:24 cmake-build-debug
-rw-r--r-- 1 frolovbuk frolovbuk 1559 Feb 13 03:14 main.cpp
-rw-r--r-- 1 frolovbuk frolovbuk 876 Feb 13 02:02 zen-of-python

```

Figure 3: 3 скрипт

Показав, что файла `b.out` до выполнения программы не существовало, я выполняю программу, после чего вызываю `ls -l`, чтобы содержимое директории **с правами доступа**. Из этой записи видно, что права доступа файла `a.out` и `b.out` **идентичны**. Попробуем теперь скопировать "Zen of Python" в новый файл, используя новосозданный исполняемый файл:

```

frolovbuk@LAPTOP-K0RFAB50:~/hw4$ ls
CMakeLists.txt  a.out  ans  b.out  cmake-build-debug  main.cpp  zen-of-python
frolovbuk@LAPTOP-K0RFAB50:~/hw4$ ./b.out 2 zen-of-python bns
frolovbuk@LAPTOP-K0RFAB50:~/hw4$ ls
CMakeLists.txt  a.out  ans  b.out  bns  cmake-build-debug  main.cpp  zen-of-python
frolovbuk@LAPTOP-K0RFAB50:~/hw4$ cat bns
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than right now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

```

Figure 4: 4 скрипт

3 Пару замечаний

1. Ввод в программе предусмотрен через аргументы командной строки с проверкой на невалидный ввод
2. Для получения информации о правах доступа файла в C++ я использую заголовочный файл `sys/stat.h`, который предоставляет специальную структуру данных и методы для

работы с ней, выполняющие именно эту функцию (sys в названии обозначает то, что файл содержит код, связанный с операционной системой и системными вызовами)

3. Чтобы изменить способ чтения файла (с чтения циклом на чтение единым буфером) достаточно раскомментировать соответствующие строки. Но в целях безопасности работы с исполняемым файлом, я оставил чтение и запись маленьким буфером в цикле