Задание №2.
Перенаправление ввода-вывода

2 апреля 2016 г.

По умолчанию при запуске любого процесса для него открывается три файловых дескриптора с номерами 0, 1 и 2. Их называют стандратными потоками ввода-вывода.

0 - поток ввода;

1 - поток вывода;

2 - поток вывода ошибок.

Дескриптор стандартного потока ввода обычно связан с клавиатурой, а два других — с экраном терминала. В POSIX существуют удобные макроопределения

```
#define STDIN_FILENO   0
#define STDOUT_FILENO  1
#define STDERR_FILENO  2
```

для именования этих дескрипторов.

Одной из мощнейших возможностей оболочки является такой функционал, как перенаправление ввода-вывода. Т.е. возможность изменить связь стандартных файловых дескрипторов при запуске программы.

Пример

```
1 $echo "Hello world"
2 Hello world
3 $echo "Hello world" > hello.txt
4 $cat hello.txt
5 Hello world
6 $cat < hello.txt > hello2.txt
```

На строке 3 при выполнении программы `echo` ее вывод будет направлен не на экран, а в файл hello.txt. На строке 6 программа `cat` будет читать данные не с клавиатуры, а их файла `hello.txt` и писать данные в файл `hello2.txt`.

Задание: доработать программу из первой работы, добавив к оболочке функционал по перенаправления стандартного потока ввода и стандартного потока вывода с использованием синтаксиса приведенного выше. Стоит отметить, что в стандартных оболочках нет необходимости ставить пробелы вокруг символов '<' и '>', которые считаются разделителями слов.

Уаазания: для выполнения задания вам потребуется использовать системные вызовы `open` и `dup2`. Рассмотрим простой пример перенаправления вывода с их помощью:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

int main(int argc, char* argv[]) {
    if (argc < 2) {
        printf("Usage: %s filename [ARGS...]\n", argv[0]);
        return EXIT_FAILURE;
    }

    int fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0666);
    if (fd == -1) {
        perror("open");
        return EXIT_FAILURE;
    }

    /* Forward STDOUT_FILENO to fd */
    if (-1 == dup2(fd, STDOUT_FILENO)) {
        perror("dup2");
        return EXIT_FAILURE;
    }

    int i;
    for (i = 2; i < argc; ++i) {
        printf("%s\n", argv[i]);
    }

    return EXIT_SUCCESS;
}
```

Данная программа создаст файл, имя котором переданно как первый аргумент и запишет в него все свои аргументы начиная со второго. Обратите внивание, что печать производится с помощью функции `printf`, которая использует стандартный поток вывода, но поскольку за счет вызова `dup2` он перенаправлен в файл, то и печать будет происходить в файл.

Стоит отметить, что про создании процесса происходит копирование всех открытых файловых дескрипторов. Т.е. если дочерний процесс изме-

нит свои файловые дескрипторы, то это никак не отразится на родительском. Это подсказывает нам, что системные вызов `dup2` следует использовать в дочернем, но не в родительском процессе.