Gebze Technical University

CSE 344 – System Programming

Homework #1

29 March 2023

Abdurrahman Bulut

1901042258

Introduction

This report contains the answers and explanations of the 3 questions asked in the first assignment of the CSE344 course.

Q1 Explanation

It asked to us to write a program that allows us to append bytes to a file using a command line. It will take up to three arguments and creates the file if it does not exist. By default, it opens the file with a flag that appends the new bytes to the end of the file. If we supply a third argument, the program will omit that flag and instead use a function to seek the end of the file before appending each byte.

So, I wrote a program in C that allows us to write a specified number of bytes to a file. First of all, I used the command line arguments to get the name of the file and the number of bytes I want to write. I also added an optional third argument that allows us to choose whether to append to the file or start writing from the beginning. If the third argument is not provided, the program defaults to appending. I also added some error handling to the program to make sure that the command line arguments are valid. If the user provides too many or too few arguments, the program prints a usage message and exits. If the user provides an invalid third argument, the program prints an error message and exits.

To open the file, I used the open system call, which takes three arguments: the name of the file, a set of flags that determines how the file should be opened, and a set of permissions that determines who can access the file. I chose to give read and write permissions to the user, the group, and others.

To write to the file, I used a simple loop that writes one byte at a time. If the user specified that they want to start writing from the beginning of the file, I used the lseek system call to move the file pointer to the end of the file before writing each byte. And finally, I closed the file using the close system call and checked for errors.

Test 1:

The command is for creating 2 files in same size of bytes. One of them is created by using lseek, the other one is created by using append flag. As we can see, there are some size difference between them. 2000000 bytes && 1671742 bytes.

```
bulut@bulut:~/Desktop/System Programming/hws/CSE344/hw1/src$ make
gcc -Wall -o appendMeMore appendMeMore.c
gcc -Wall -o q2 q2.c
gcc -Wall -o q3 q3.c
bulut@bulut:~/Desktop/System Programming/hws/CSE344/hw1/src$ ./appendMeMore f1 1000000 & ./appendMeMore f1 1000000
[1] 6755
[1]+  Done                    ./appendMeMore f1 1000000
bulut@bulut:~/Desktop/System Programming/hws/CSE344/hw1/src$ ./appendMeMore f2 1000000 x & ./appendMeMore f2 1000000 x
[1] 6789
bulut@bulut:~/Desktop/System Programming/hws/CSE344/hw1/src$ ls -l f1 f2
[1]+  Done                    ./appendMeMore f2 1000000 x
-rw-rw-r-- 1 bulut bulut 2000000 Mar 29 10:57 f1
-rw-rw-r-- 1 bulut bulut 1671742 Mar 29 10:58 f2
bulut@bulut:~/Desktop/System Programming/hws/CSE344/hw1/src$
```

Main question is: Why did the file size difference occurs? f1 is 200000 bytes, f2 is 1671742 bytes?

When I search in the internet, the people says it is an “atomic append problem”.

When we add something to a file, we want it to go at the end of the file and to be added as one whole thing. We are using flag called "O_APPEND". This flag tells the computer to add the new thing to the end of the file, all at once. When this flag is used, the write() system call automatically seeks to the end of the file before writing the data. It ensures that each write operation is atomic and writes the data to the end of the file, no matter where the end of the file is.

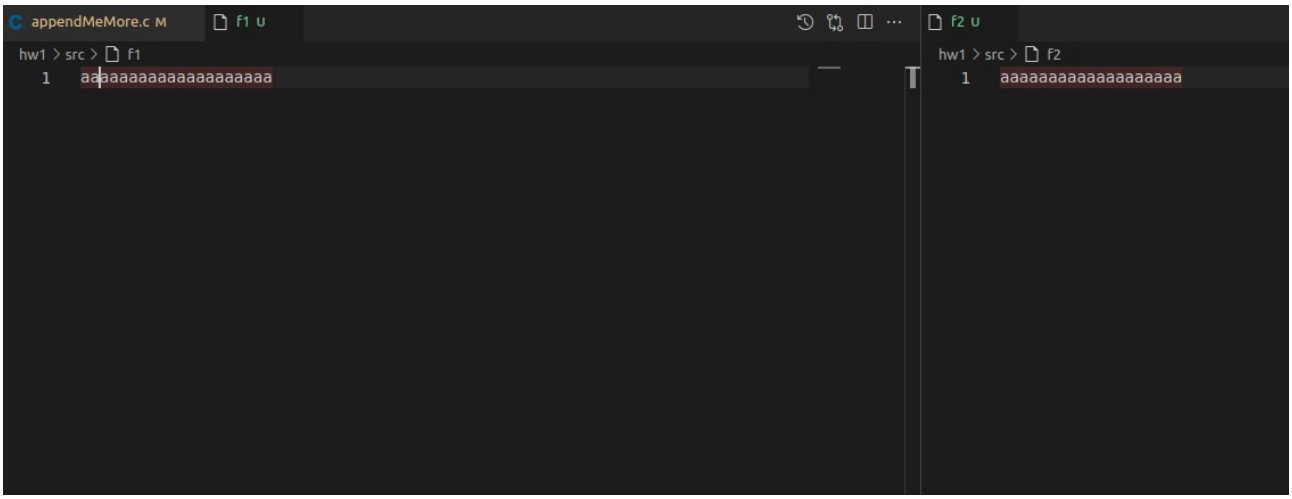
When we use a command called "lseek" to find the end of the file before we add the new data, it can cause a problem called a "race condition". For example, if one part of the computer uses "lseek" to find the end of the file but gets interrupted before it can add data, another part of the computer might add data to the file before the first part can start again. This can cause the first part to add data to the wrong place, making the file bigger or corrupted.

The lseek() and write() system calls are two separate calls, and when used together, they do not work atomically. This means that the end of the file can change between the lseek() and write() calls, especially when multiple processes are appending to the same file simultaneously.

For example, when I want to write 20 bytes with letter “a”, the results will be 20 and 19 bytes for each file.

```
bulut@bulut:~/Desktop/System Programming/hws/CSE344/hw1/src$ ./appendMeMore f1 10 & ./appendMeMore f1 10
[1] 8795
[1]+  Done                    ./appendMeMore f1 10
bulut@bulut:~/Desktop/System Programming/hws/CSE344/hw1/src$ ./appendMeMore f2 10 x & ./appendMeMore f2 10 x
[1] 8817
[1]+  Done                    ./appendMeMore f2 10 x
bulut@bulut:~/Desktop/System Programming/hws/CSE344/hw1/src$ ls -l f1 f2
-rw-rw-r-- 1 bulut bulut 20 Mar 29 11:34 f1
-rw-rw-r-- 1 bulut bulut 19 Mar 29 11:35 f2
bulut@bulut:~/Desktop/System Programming/hws/CSE344/hw1/src$
```

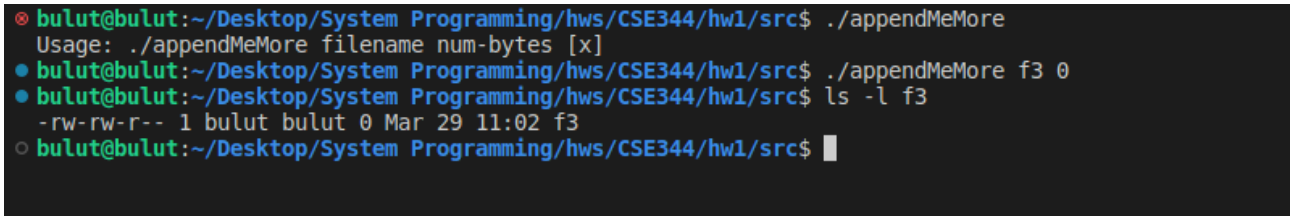
If we open those files we can see number of letter “a”.



Test 2-3:

If we do not enter proper number of arguments, It will give proper command argument template.

And also if we give 0 byte as size, It will create a file which has 0 bytes data in the file.



Other tests:

Test 4:

- Command: ./appendMeMore f3 10000
- Output: A file named "f3" with 10,000 bytes appended.

Test 5:

- Command: ./appendMeMore f4 5000 x
- Output: A file named "f4" with 5,000 bytes appended.

We can use “make clean” to clean object files.

Q2 Explanation

It is asked us to implement dup() and dup2() functions in C language.

The dup() is a system call that creates a new file descriptor that is a copy pf the original file descriptor. dup2() is also create a new file descriptor but it uses a specified value specified by the second argument..

The fcntl() is a system call that can perform various operations on file descriptors. One of the operations it can perform is duplicating a file descriptor. We will use this for this problem.

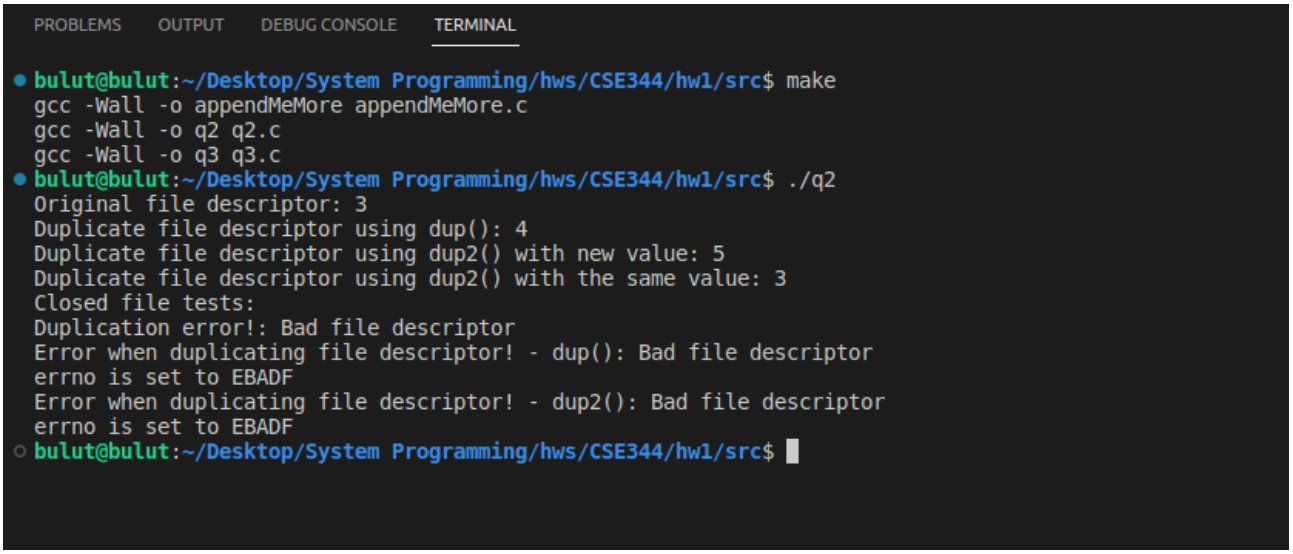
My program first opens a file named "test.txt" in read-write mode with open(). I used the O_CREAT, O_RDWR, and O_APPEND flags. The mode of the file is set to allow read and write access for the user, group, and others using the mode_t data type. The file descriptor returned by open() is stored in the fd variable. After that the program demonstrates the use of dup() to create a duplicate file descriptor fd1. The fd file descriptor is duplicated by calling dup() with fd as an argument, and the new file descriptor value is stored in fd1.

Then it demonstrates the use of dup2() to create another duplicate file descriptor fd2 with a new value of 5. dup2() is called with fd as the first argument, and 5 as the second argument. The new file descriptor value is stored in fd2. It also demonstrates the use of dup2() to create a duplicate file descriptor fd3 with the same value as the original fd. dup2() is called with fd as both the first and second arguments. The new file descriptor value is stored in fd3. Then it tests the behavior of dup() and dup2() when attempting to duplicate a closed file descriptor. Two new files are opened with open() and then immediately closed with close(). The first file is named "test.txt", which was already opened earlier in the program. The second file is named "abs.txt". If the file cannot be opened or closed, an error message is printed to the console. I wanted to test closed files. The code attempts to duplicate the closed file descriptors using dup() and dup2(). When dup() is called with a closed file descriptor, it returns -1 and sets the errno variable to EBADF, indicating a bad file descriptor. When dup2() is called with a closed file descriptor, it returns -1 and sets errno to EBADF if either the old file descriptor or the new file descriptor is invalid. Finally, the program closes the original file descriptor with close().

Tests

```c
mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH;
int fd = open("test.txt", O_CREAT | O_RDWR | O_APPEND, mode); // 0-1-2, fd = 3

if (fd == -1)
{
    perror("Error when opening the file!");
    return 1;
}

printf("Original file descriptor: %d\n", fd);

// Duplicate the file descriptor using dup()
int fd1 = dup(fd); // fd1 = 4

if (fd1 == -1)
{
    perror("Error when duplicating file descriptor! - dup()");
    return 1;
}

printf("Duplicate file descriptor using dup(): %d\n", fd1);
```

```c
// Dup2() with a new value
int fd2 = dup2(fd, 5);
if (fd2 == -1)
{
    perror("Error when duplicating file descriptor! - dup2()");
    return 1;
}
printf("Duplicate file descriptor using dup2() with new value: %d\n", fd2);

// Dup2() with the same value
int fd3 = dup2(fd, fd);
if (fd3 == -1)
{
    perror("Error when duplicating file descriptor! - dup2()");
    return 1;
}
printf("Duplicate file descriptor using dup2() with the same value: %d\n", fd3);
printf("Closed file tests:\n");

// Closed files
int closed_fd = open("test.txt", O_CREAT | O_RDWR | O_APPEND, mode);

if (closed_fd == -1)
{
    perror("Error when opening the file!");
}
else
{
    if (close(closed_fd) == -1)
    {
        perror("Error when closing the file!");
    }
    int fd4 = dup(closed_fd);
    if (fd4 == -1)
    {
        perror("Error when duplicating file descriptor! - dup()");
        if (errno == EBADF)
        {
            printf("errno is set to EBADF\n");
        }
    }
```


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
bulut@bulut:~/Desktop/System Programming/hws/CSE344/hw1/src$ make
gcc -Wall -o appendMeMore appendMeMore.c
gcc -Wall -o q2 q2.c
gcc -Wall -o q3 q3.c
bulut@bulut:~/Desktop/System Programming/hws/CSE344/hw1/src$ ./q2
Original file descriptor: 3
Duplicate file descriptor using dup(): 4
Duplicate file descriptor using dup2() with new value: 5
Duplicate file descriptor using dup2() with the same value: 3
Closed file tests:
Duplication error!: Bad file descriptor
Error when duplicating file descriptor! - dup(): Bad file descriptor
errno is set to EBADF
Error when duplicating file descriptor! - dup2(): Bad file descriptor
errno is set to EBADF
bulut@bulut:~/Desktop/System Programming/hws/CSE344/hw1/src$

Q3 Explanation

It is asked to code a program that verifies whether the duplicated file descriptors share a file offset value and open file.

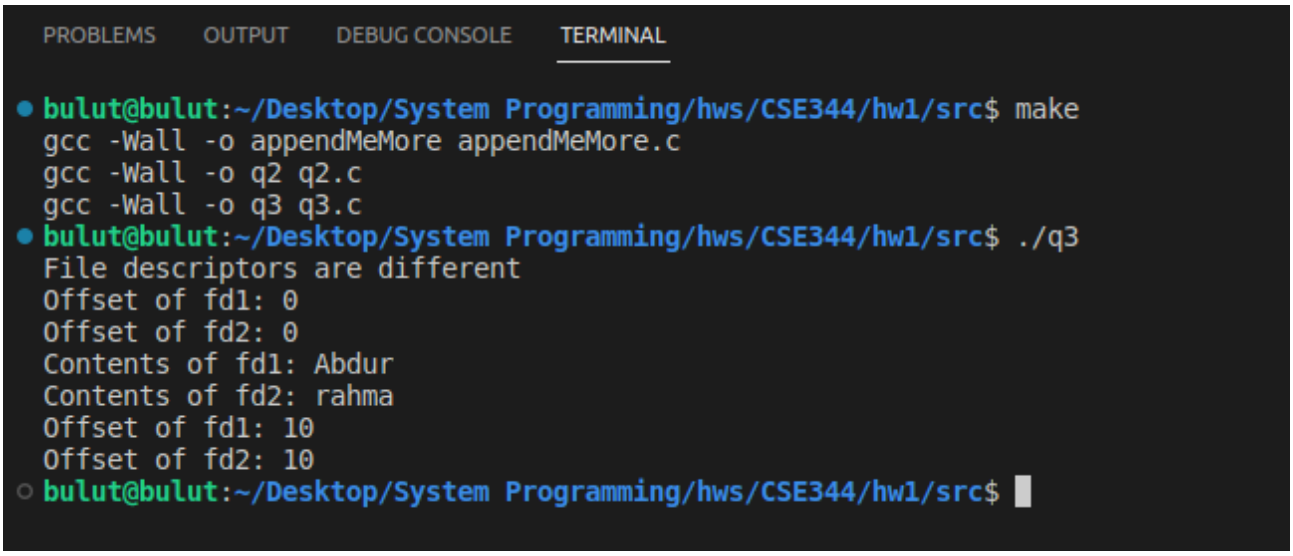
My program starts by trying to open a file called "test.txt" in read-only mode. If it can't open the file, it prints an error message and stops. Then it creates a copy of the file descriptor for the same file using the dup() function that I implemented for second question. If there's a problem creating the copy, the program prints an error message and stops. Then it checks if the two file descriptors are the same or different and prints the result. If they are the same, it means that both file descriptors refer to the same open file.

After that, It finds the current offset of both file descriptors using the lseek() function and prints them. If the two file descriptors share the same open file, they should have the same offset. In my example, offsets are zero by each initially.

It then reads the first 5 bytes of data from the file using both file descriptors and prints them. If both file descriptors share the same open file, they should read the same data. My example file content is my name and my surname. So, it gives “abdur” and “rahma”.

After reading from the file, it finds the new offset of both file descriptors using lseek() again. If the two file descriptors share the same open file, they should have the same new offset. In my example, offsets are 10 by each. Finally, the program closes both file descriptors. If both file descriptors share the same open file, closing one of them should also close the other. The dup() function in the code that I used is from 2nd question. I implemented it before. So it proves that duplicated file descriptors share a file offset value and open file.

Test


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
bulut@bulut:~/Desktop/System Programming/hws/CSE344/hw1/src$ make
gcc -Wall -o appendMeMore appendMeMore.c
gcc -Wall -o q2 q2.c
gcc -Wall -o q3 q3.c
bulut@bulut:~/Desktop/System Programming/hws/CSE344/hw1/src$ ./q3
File descriptors are different
Offset of fd1: 0
Offset of fd2: 0
Contents of fd1: Abdur
Contents of fd2: rahma
Offset of fd1: 10
Offset of fd2: 10
bulut@bulut:~/Desktop/System Programming/hws/CSE344/hw1/src$