CSE 344

SYSTEM PROGRAMMING

SPRING 2023

AHMET CEBECİ

1901042708

QUESTION 1)

The reason for the difference in the output between the two calls to of the program is due to the use of the lseek flag in the second call, which can cause a race condition.

In the first call, the program opens the file with the O_APPEND flag, which ensures that all writes are appended to the end of the file in a mutually exclusive manner.

Therefore, both instances of the program can write to the same file without causing any conflicts, and the total number of bytes written is exactly 2,000,000.

First Call like this:

```
$ ./appendMeMore f1.txt 1000000  & ./appendMeMore f1.txt
1000000
```

In the second call, the program uses the lseek flag, which seeks to the end of the file before writing each byte.

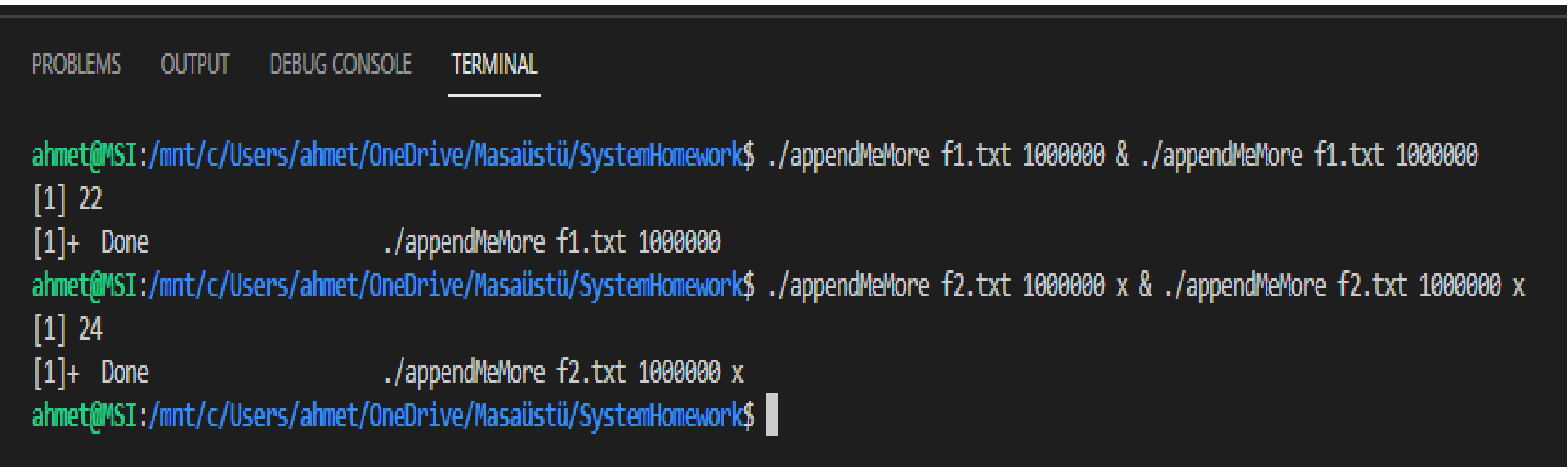
However, since both instances of the program are writing to the same file, it is possible for one instance to seek to the end of the file, then be interrupted by the other instance before it writes its byte.

This can result in one or more bytes being skipped, causing the total number of bytes written to be less than 2,000,000.

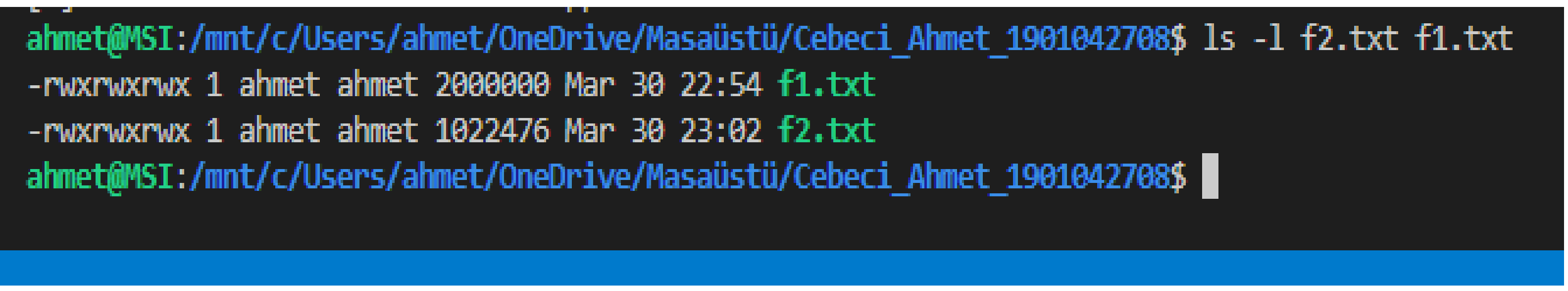
Second Call like this:

```
$ ./appendMeMore f2.txt 1000000 x  & ./appendMeMore f2.txt
1000000 x
```

To avoid this race condition when using lseek, the program could use file locking or other synchronization mechanisms to ensure that only one instance of the program is seeking and writing to the file at a time.


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/SystemHomework$ ./appendMeMore f1.txt 1000000 & ./appendMeMore f1.txt 1000000
[1] 22
[1]+  Done                    ./appendMeMore f1.txt 1000000
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/SystemHomework$ ./appendMeMore f2.txt 1000000 x & ./appendMeMore f2.txt 1000000 x
[1] 24
[1]+  Done                    ./appendMeMore f2.txt 1000000 x
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/SystemHomework$
```



```
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/Cebeci_Ahmet_1901042708$ ls -l f2.txt f1.txt
-rwxrwxrwx 1 ahmet ahmet 2000000 Mar 30 22:54 f1.txt
-rwxrwxrwx 1 ahmet ahmet 1022476 Mar 30 23:02 f2.txt
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/Cebeci_Ahmet_1901042708$
```

QUESTION 2 & 3)

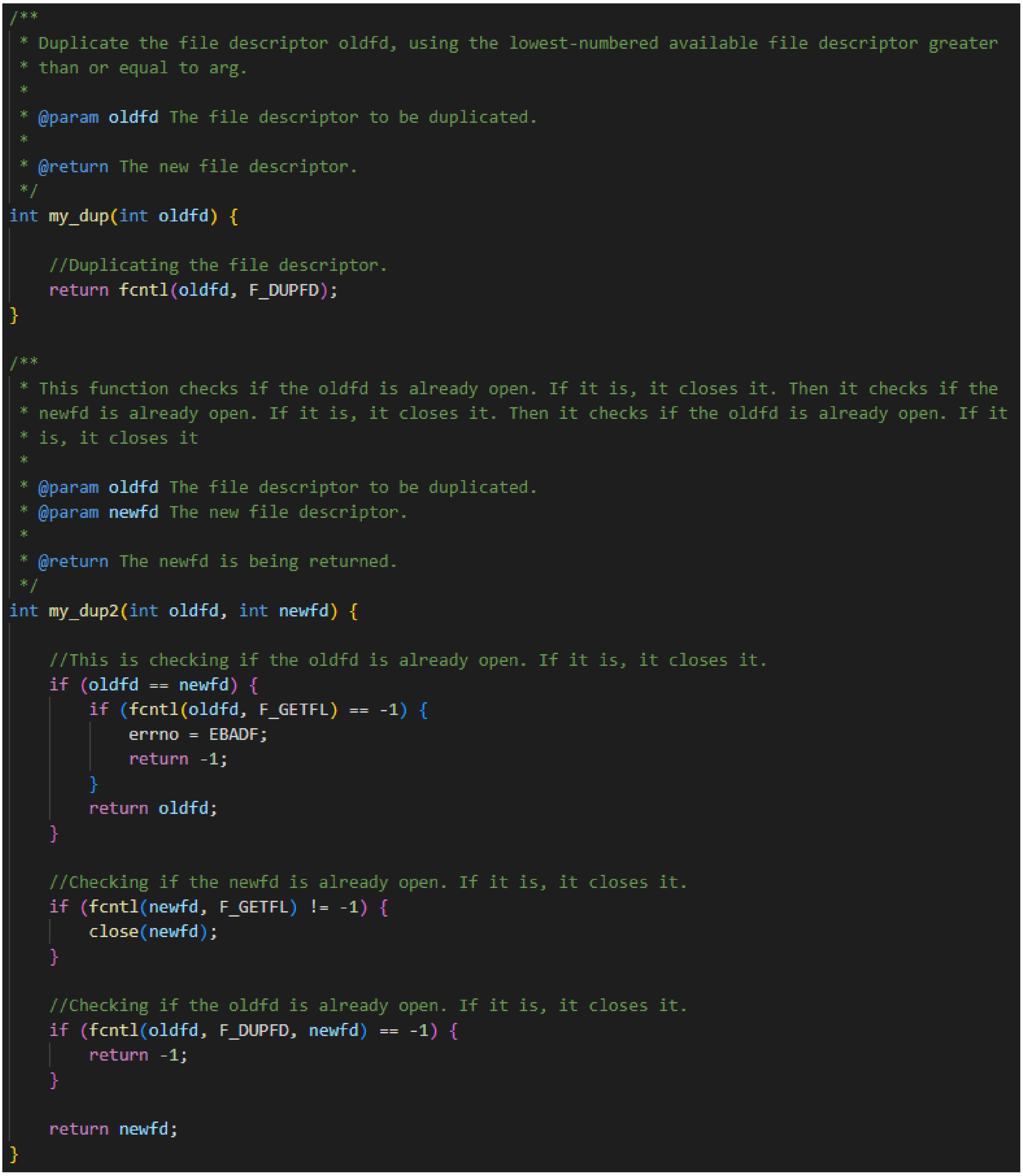
This program demonstrates the usage of the my_dup and my_dup2 functions in C, which are used to duplicate file descriptors. The program opens a file named "test.txt" in write-only mode and creates two additional file descriptors using my_dup and my_dup2.

The my_dup function duplicates the file descriptor passed as an argument, and returns the new file descriptor. The my_dup2 function is similar, but also allows the caller to specify the file descriptor number for the new file descriptor.

The program then writes some data to all three file descriptors and sets the offset of one of the duplicated file descriptors using lseek. The program then uses lseek again to check the offset of the other duplicated file descriptor and compare it to the first one.

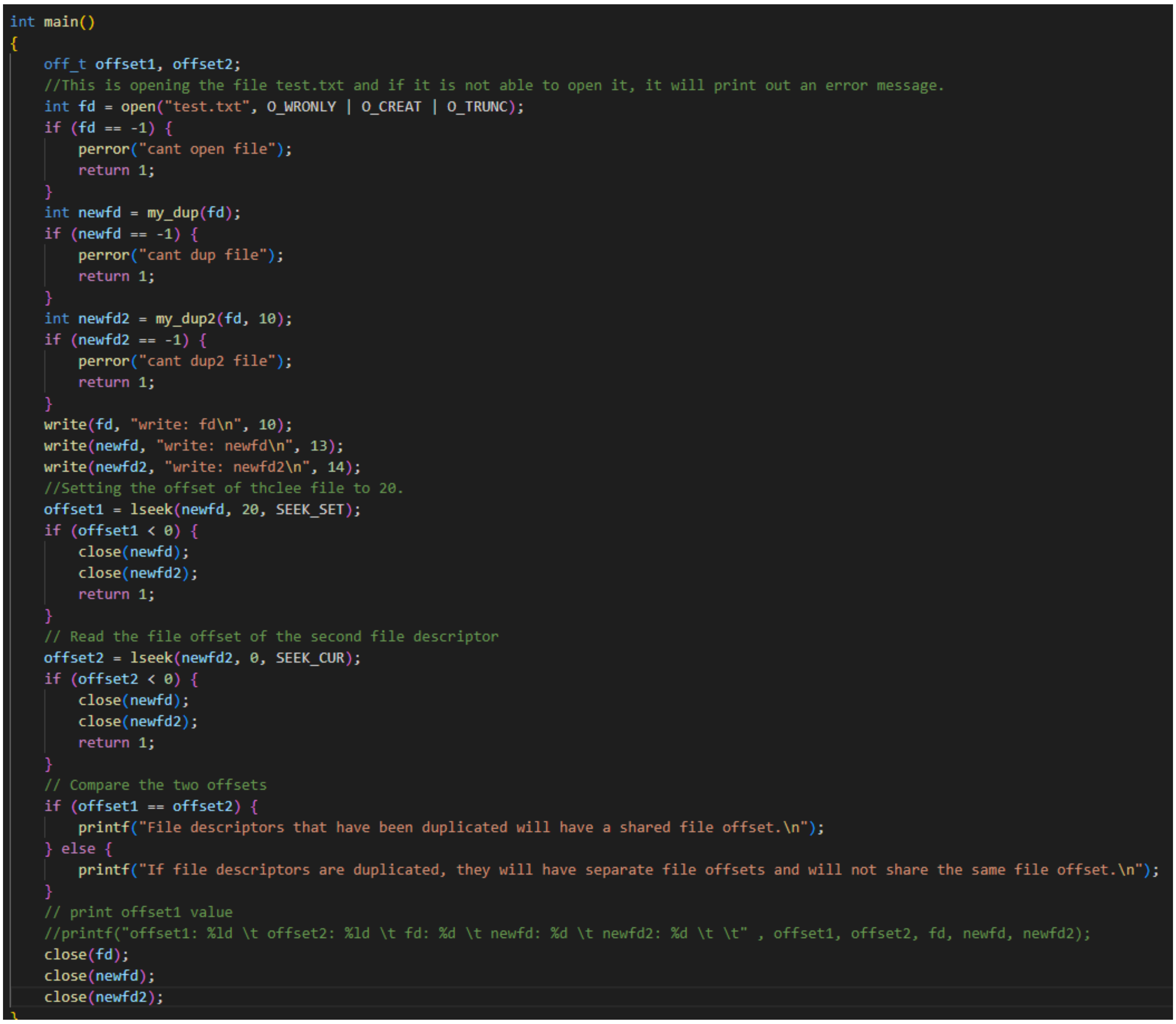
If the two offsets are the same, the program prints a message indicating that duplicated file descriptors share a file offset. If the two offsets are different, the program prints a message indicating that duplicated file descriptors have separate file offsets.

The fcntl function is also used in the my_dup2 function to check if a file descriptor is already open, and close it if necessary. The errno variable is used to indicate errors that occur during the duplication process.



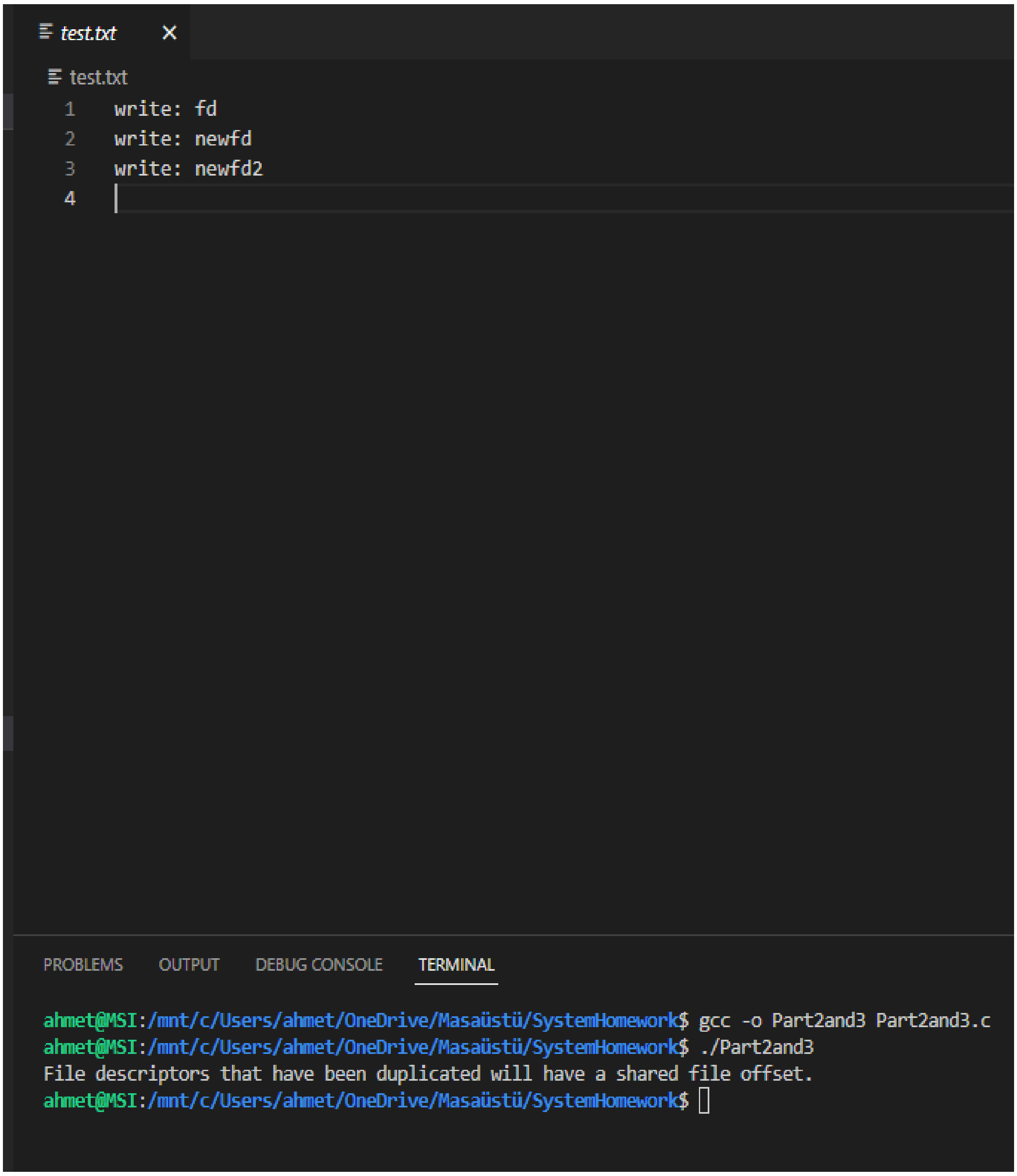
```c
/**
 * Duplicate the file descriptor oldfd, using the lowest-numbered available file descriptor greater
 * than or equal to arg.
 *
 * @param oldfd The file descriptor to be duplicated.
 *
 * @return The new file descriptor.
 */
int my_dup(int oldfd) {

    //Duplicating the file descriptor.
    return fcntl(oldfd, F_DUPFD);
}

/**
 * This function checks if the oldfd is already open. If it is, it closes it. Then it checks if the
 * newfd is already open. If it is, it closes it. Then it checks if the oldfd is already open. If it
 * is, it closes it
 *
 * @param oldfd The file descriptor to be duplicated.
 * @param newfd The new file descriptor.
 *
 * @return The newfd is being returned.
 */
int my_dup2(int oldfd, int newfd) {

    //This is checking if the oldfd is already open. If it is, it closes it.
    if (oldfd == newfd) {
        if (fcntl(oldfd, F_GETFL) == -1) {
            errno = EBADF;
            return -1;
        }
        return oldfd;
    }

    //Checking if the newfd is already open. If it is, it closes it.
    if (fcntl(newfd, F_GETFL) != -1) {
        close(newfd);
    }

    //Checking if the oldfd is already open. If it is, it closes it.
    if (fcntl(oldfd, F_DUPFD, newfd) == -1) {
        return -1;
    }

    return newfd;
}
```

The function of the main code block is to copy a file identifier (create another identifier), to the same file(text.txt) is to test different typed strings and compare the file distance values of two identifiers. We are learning whether the file identifiers will share distance values or have different distance values.



```c
int main()
{
    off_t offset1, offset2;
    //This is opening the file test.txt and if it is not able to open it, it will print out an error message.
    int fd = open("test.txt", O_WRONLY | O_CREAT | O_TRUNC);
    if (fd == -1) {
        perror("cant open file");
        return 1;
    }
    int newfd = my_dup(fd);
    if (newfd == -1) {
        perror("cant dup file");
        return 1;
    }
    int newfd2 = my_dup2(fd, 10);
    if (newfd2 == -1) {
        perror("cant dup2 file");
        return 1;
    }
    write(fd, "write: fd\n", 10);
    write(newfd, "write: newfd\n", 13);
    write(newfd2, "write: newfd2\n", 14);
    //Setting the offset of thclee file to 20.
    offset1 = lseek(newfd, 20, SEEK_SET);
    if (offset1 < 0) {
        close(newfd);
        close(newfd2);
        return 1;
    }
    // Read the file offset of the second file descriptor
    offset2 = lseek(newfd2, 0, SEEK_CUR);
    if (offset2 < 0) {
        close(newfd);
        close(newfd2);
        return 1;
    }
    // Compare the two offsets
    if (offset1 == offset2) {
        printf("File descriptors that have been duplicated will have a shared file offset.\n");
    } else {
        printf("If file descriptors are duplicated, they will have separate file offsets and will not share the same file offset.\n");
    }
    // print offset1 value
    //printf("offset1: %ld \t offset2: %ld \t fd: %d \t newfd: %d \t newfd2: %d \t \t" , offset1, offset2, fd, newfd, newfd2);
    close(fd);
    close(newfd);
    close(newfd2);
```

Output :

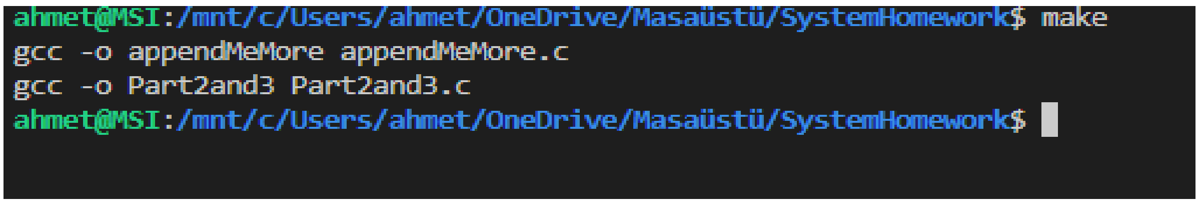

test.txt

```
write: fd
write: newfd
write: newfd2

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/SystemHomework$ gcc -o Part2and3 Part2and3.c
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/SystemHomework$ ./Part2and3
File descriptors that have been duplicated will have a shared file offset.
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/SystemHomework$
```

To compile with makefile:



```
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/SystemHomework$ make
gcc -o appendMeMore appendMeMore.c
gcc -o Part2and3 Part2and3.c
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/SystemHomework$
```