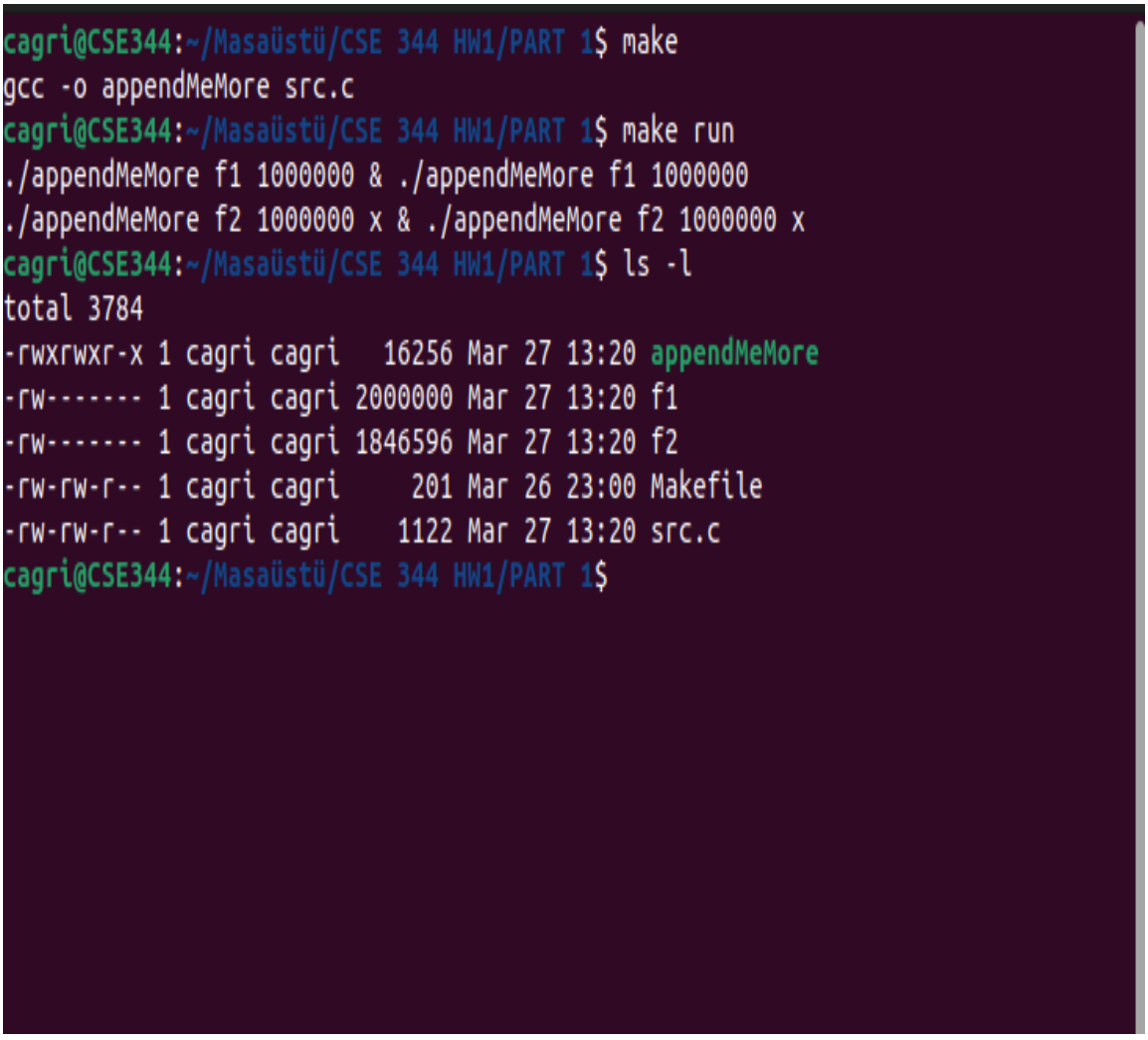
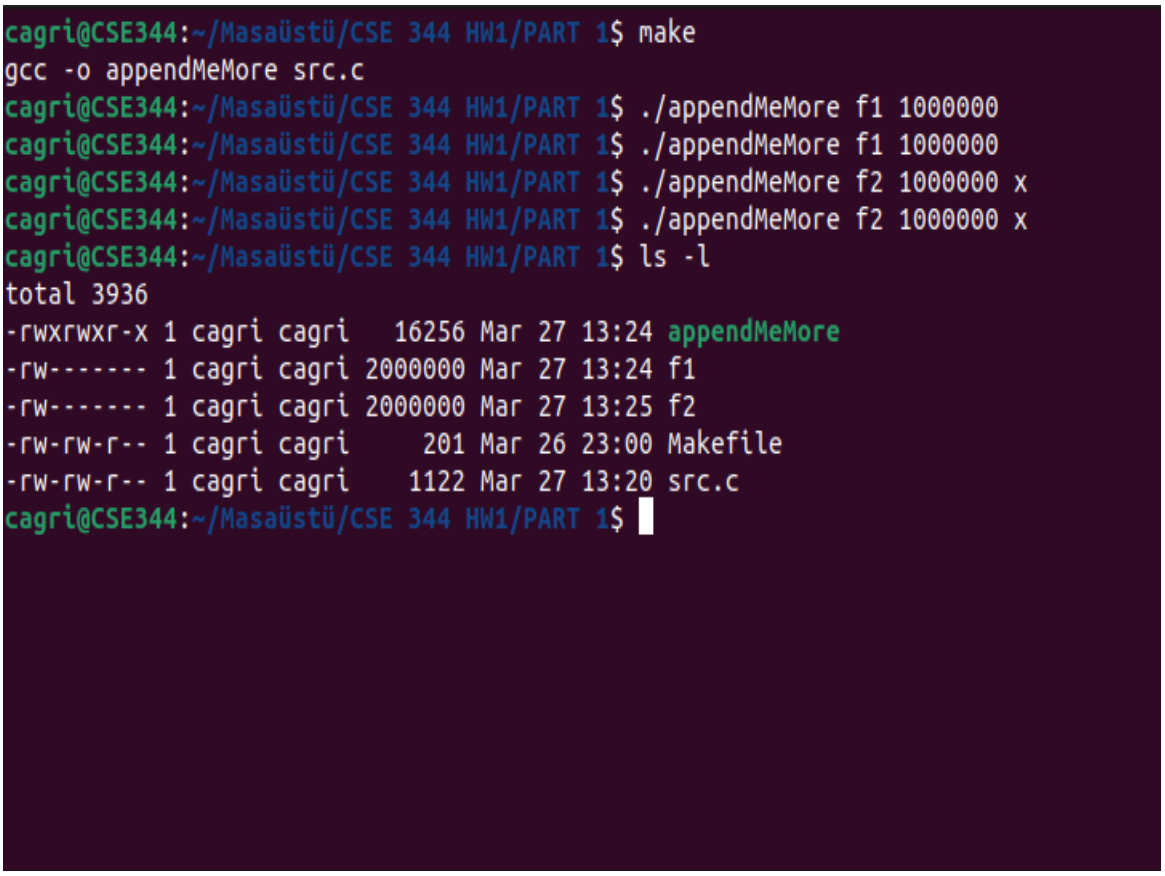
CSE 344 HOMEWORK 1 REPORT

PART 1:

The code of the program is written according to the given instructions. If the number of the arguments is less than 3, prints an error message and exit. The file is opened in O_WRONLY and O_CREAT, O_APPEND modes. If the number of the command line arguments is equals to 4, O_APPEND mode is omitted and file offset is set with lseek function.



As it is seen from the above, size of the f2 is smaller than the size of the f1. The reason why f2 has smaller size is, when two or more threads are issuing lseek function for the same file descriptor, some conflicts happen. It is caused by lseek is not thread-safe function. On the other hand, O_APPEND flag guarantee that writes safely append.



When we run processes separately, there is no size difference between two files f1 and f2. So, the size difference in the first picture is caused by using lseek with multithread.

PART 2:

The code of the program is written according to the given instructions and some documentation about dup and dup2 functions.

dup (int oldfd):

In this function, oldfd must be nonnegative integer. So, oldfd is checked firstly. If the oldfd is negative, returns -1. Otherwise, returns fcntl (oldfd, F_DUPFD, 0) which is equivalent to dup(oldfd).

dup2 (int oldfd, int newfd):

In this function, oldfd and newfd must be nonnegative integers. Also, oldfd must be valid file

descriptors. If these conditions aren't satisfied, returns -1. Otherwise, oldfd and newfd is checked. If they are equal, return newfd. On the other hand, if they are not equal, closes the newfd, and returns fcntl (oldfd, F_DUPFD, newfd) which is equivalent to dup (oldfd, newfd).

TESTS

To test dup and dup2 functions, the results of the my dup and dup2 functions are compared with the results of the original dup and dup2 functions in the unistd.h library.

TEST 1)

```c
int oldfd = open("test1.txt", O_WRONLY | O_CREAT | O_APPEND, S_IWUSR | S_IRUSR);

printf("DUPLICATE VALID FILE DESCRIPTOR WITH ORIGINAL DUP FUNCTION\n");

int newfd = dup(oldfd);

printf("NEW FILE DESCRIPTOR: %d\n", newfd);

close(newfd);

printf("DUPLICATE VALID FILE DESCRIPTOR WITH MY DUP FUNCTION\n");

newfd = duplicate(oldfd);

printf("NEW FILE DESCRIPTOR: %d\n", newfd);

close(newfd);
```

(Duplicate file descriptor with valid file descriptor.)

```
DUPLICATE VALID FILE DESCRIPTOR WITH ORIGINAL DUP FUNCTION
NEW FILE DESCRIPTOR: 4
DUPLICATE VALID FILE DESCRIPTOR WITH MY DUP FUNCTION
NEW FILE DESCRIPTOR: 4
```

TEST 2)

```
printf("DUPLICATE UNVALID FILE DESCRIPTOR WITH ORIGINAL DUP FUNCTION\n");

newfd = dup(-5);

printf("NEW FILE DESCRIPTOR: %d\n", newfd);

close(newfd);

printf("DUPLICATE UNVALID FILE DESCRIPTOR WITH MY DUP FUNCTION\n");

newfd = duplicate(-5);

printf("NEW FILE DESCRIPTOR: %d\n", newfd);

close(oldfd);
```

(Duplicate file descriptor with invalid file descriptor.)

```
DUPLICATE UNVALID FILE DESCRIPTOR WITH ORIGINAL DUP FUNCTION
NEW FILE DESCRIPTOR: -1
DUPLICATE UNVALID FILE DESCRIPTOR WITH MY DUP FUNCTION
NEW FILE DESCRIPTOR: -1
```

TEST 3)

```
int oldfd2 = open("test2.txt", O_WRONLY | O_CREAT | O_APPEND, S_IWUSR | S_IRUSR);

printf("DUPLICATE VALID FILE DESCRIPTOR WITH ORIGINAL DUP FUNCTION\n");

newfd = dup(oldfd2);

printf("NEW FILE DESCRIPTOR: %d\n", newfd);

close(newfd);

printf("DUPLICATE VALID FILE DESCRIPTOR WITH MY DUP FUNCTION\n");

newfd = duplicate(oldfd2);

printf("NEW FILE DESCRIPTOR: %d\n", newfd);

close(newfd);
```

(Duplicate file descriptor with valid file descriptor. Check the new file descriptor lowest-numbered file descriptor which was unused.)

```
DUPLICATE VALID FILE DESCRIPTOR WITH ORIGINAL DUP FUNCTION
NEW FILE DESCRIPTOR: 3
DUPLICATE VALID FILE DESCRIPTOR WITH MY DUP FUNCTION
NEW FILE DESCRIPTOR: 3
```

TEST 4)

```
int fd = open("test1.txt", O_WRONLY | O_CREAT | O_APPEND, S_IWUSR | S_IRUSR);

printf("DUPLICATE INVALID OLD FILE DESCRIPTOR WITH ORIGINAL DUP2 FUNCTION\n");

int newfd = dup2(-5, fd);

printf("NEW FILE DESCRIPTOR: %d\n", newfd);

close(newfd);

printf("DUPLICATE INVALID OLD FILE DESCRIPTOR WITH MY DUP2 FUNCTION\n");

newfd = duplicate2(-5, fd);

printf("NEW FILE DESCRIPTOR: %d\n", newfd);

close(newfd);
```

(Duplicate file descriptor with invalid old file descriptor.)

```
DUPLICATE INVALID OLD FILE DESCRIPTOR WITH ORIGINAL DUP2 FUNCTION
NEW FILE DESCRIPTOR: -1
DUPLICATE INVALID OLD FILE DESCRIPTOR WITH MY DUP2 FUNCTION
NEW FILE DESCRIPTOR: -1
```

TEST 5)

```
printf("DUPLICATE INVALID NEW FILE DESCRIPTOR WITH ORIGINAL DUP2 FUNCTION\n");

newfd = dup2(fd, -5);

printf("NEW FILE DESCRIPTOR: %d\n", newfd);

close(newfd);

printf("DUPLICATE INVALID NEW FILE DESCRIPTOR WITH MY DUP2 FUNCTION\n");

newfd = duplicate2(fd, -5);

printf("NEW FILE DESCRIPTOR: %d\n", newfd);

close(newfd);
```

(Duplicate file descriptor with invalid new file descriptor.)

```
DUPLICATE INVALID NEW FILE DESCRIPTOR WITH ORIGINAL DUP2 FUNCTION
NEW FILE DESCRIPTOR: -1
DUPLICATE INVALID NEW FILE DESCRIPTOR WITH MY DUP2 FUNCTION
NEW FILE DESCRIPTOR: -1
```

TEST 6)

```
printf("DUPLICATE OLDFD EQUALS NEWFD WITH ORIGINAL DUP2 FUNCTION\n");

newfd = dup2(fd, fd);

printf("NEW FILE DESCRIPTOR: %d\n", newfd);

close(newfd);

close(fd);

fd = open("test1.txt", O_WRONLY | O_CREAT | O_APPEND, S_IWUSR | S_IRUSR);

printf("DUPLICATE OLDFD EQUALS NEWFD WITH MY DUP2 FUNCTION\n");

newfd = duplicate2(fd, fd);

printf("NEW FILE DESCRIPTOR: %d\n", newfd);

close(newfd);

close(fd);
```

(Duplicate file descriptor where old file descriptor is equal to new file descriptor.)

```
DUPLICATE OLDFD EQUALS NEWFD WITH ORIGINAL DUP2 FUNCTION
NEW FILE DESCRIPTOR: 3
DUPLICATE OLDFD EQUALS NEWFD WITH MY DUP2 FUNCTION
NEW FILE DESCRIPTOR: 3
```

TEST 7)

```
fd = open("test1.txt", O_WRONLY | O_CREAT | O_APPEND, S_IWUSR | S_IRUSR);

int oldfd2 = open("test2.txt", O_WRONLY | O_CREAT | O_APPEND, S_IWUSR | S_IRUSR);

printf("DUPLICATE OLDFD AND ALREADY OPENED NEWFD (4) WITH ORIGINAL DUP2 FUNCTION\n");

newfd = dup2(fd, oldfd2);

printf("NEW FILE DESCRIPTOR: %d\n", newfd);

close(newfd);

close(oldfd2);

oldfd2 = open("test2.txt", O_WRONLY | O_CREAT | O_APPEND, S_IWUSR | S_IRUSR);

printf("DUPLICATE OLDFD AND ALREADY OPENED NEWFD (4) WITH MY DUP2 FUNCTION\n");

newfd = duplicate2(fd, oldfd2);

printf("NEW FILE DESCRIPTOR: %d\n", newfd);

close(newfd);

close(oldfd2);
```

(Duplicate file descriptor with already opened new file descriptor.)

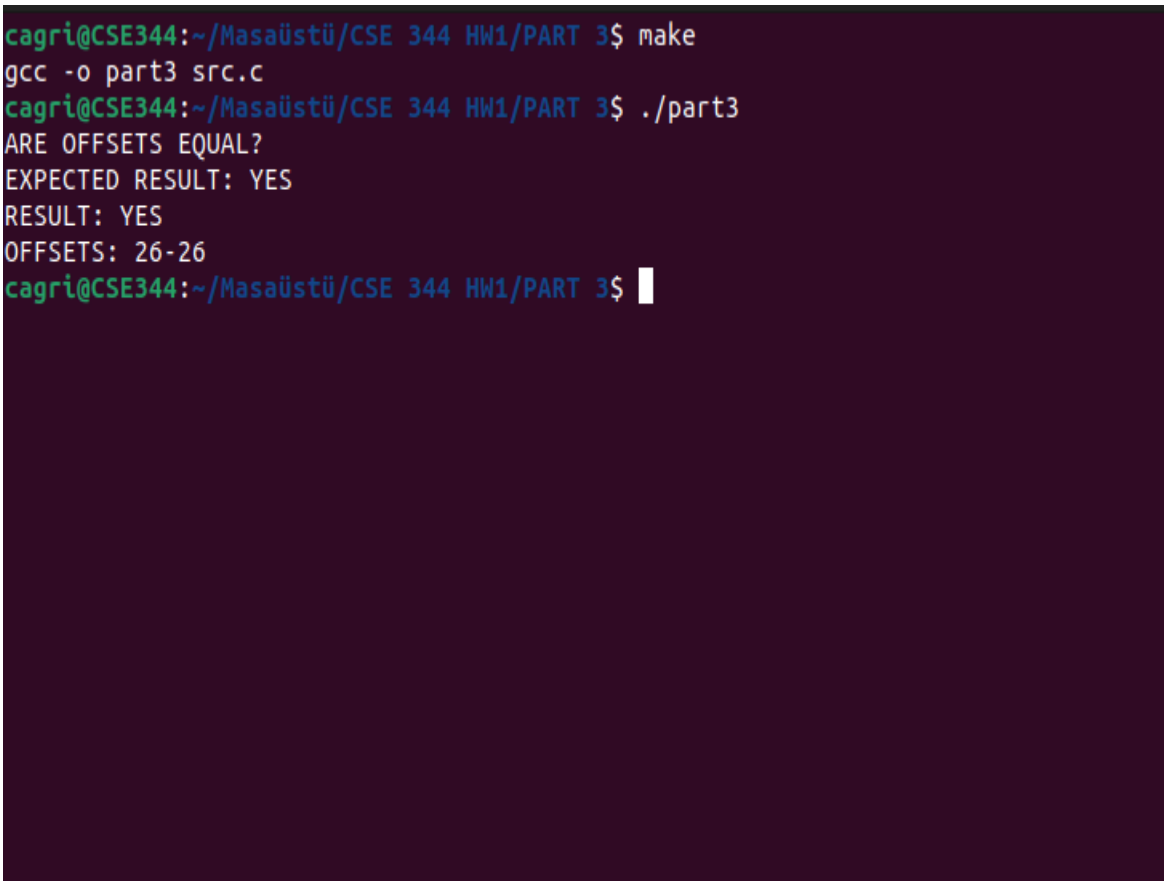
```
DUPLICATE OLDFD AND ALREADY OPENED NEWFD (4) WITH ORIGINAL DUP2 FUNCTION
NEW FILE DESCRIPTOR: 4
DUPLICATE OLDFD AND ALREADY OPENED NEWFD (4) WITH MY DUP2 FUNCTION
NEW FILE DESCRIPTOR: 4
```

TEST 8)

My dup function is also tested in the third part of the homework in terms of having same offset with duplicated file descriptors.

PART 3:

A file descriptor is created and duplicated. Some strings are written to the file by using these file descriptors. After that offsets of both file descriptors are compared with each other.



As it can be seen in the picture above, offsets of the duplicated file descriptors are the same even though different size write operation is done to the file. As a property of duplicated file descriptors, they share file offset and file status flags. If the file offset of a file descriptor is modified, the offset of the other file descriptor is also changed.