

# **CSE-344 HOMEWORK-1 REPORT**

**AHMET OKUR**

**1801042655**

## PART-1

In this section, we will see the application and difference of append and lseek.

There are two types of commands that can be entered in the terminal:

`./appendMeMore f1 1000000 & ./appendMeMore f1 1000000`

The code block that runs when this command is entered is as follows:

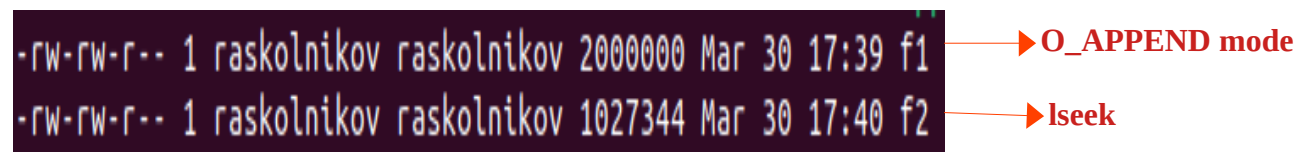
```
fd = open(fileName, (O_CREAT | O_WRONLY | O_APPEND), 0666);
for (int i = 0; i < numBytes; i++)
{
    if (write(fd, "a", 1) != 1)
    {
        perror("write");
        return -1;
    }
}
```

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

The code block that runs when this command is entered is as follows:

```
fd = open(fileName, (O_CREAT | O_WRONLY | O_APPEND), 0666);
for (int i = 0; i < numBytes; i++)
{
    if (write(fd, "a", 1) != 1)
    {
        perror("write");
        return -1;
    }
}
```

Let's examine the differences between the two code blocks. When we enter the "ls -l" command in the terminal, the output is as follows:



```
-rw-rw-r-- 1 raskolnikov raskolnikov 2000000 Mar 30 17:39 f1
-rw-rw-r-- 1 raskolnikov raskolnikov 1027344 Mar 30 17:40 f2
```

→ O\_APPEND mode

→ lseek

As you can see, the size of the two files is different. This is because for the first file the file is opened in append mode. The modification of the file offset and the write operation are performed as a single **atomic** step.

But 2nd file lseek doesn't work **atomically** so file overwrite occurs.

While 2 processes are running, append atomically writes the data in one move, and it can append to the end of the file each time. But lseek cannot do the operation in one go, so it can sometimes overwrite

## PART-2

In this part, we will write dup() and dup2() functions using fcntl.

### For dup()

The myDup() allocates a new file descriptor that refer to the same open file description as the descriptor oldfd. The new file descriptor number is guaranteed to be the lowest-numbered file descriptor that was unused in the calling process.

if the old fd is faulty, the return value is -1 and the error command is returned.

F\_DUPFD Duplicate the file descriptor fd using the lowest-numbere available file descriptor greater than or equal to arg. On success, the new file descriptor is returned

```
int myDup(int oldfd)
{
    int newfd = fcntl(oldfd, F_DUPFD, 0);
    if (newfd == -1)
    {
        errno = EBADF;
        return -1;
    }
    return newfd;
}
```

### For dup2()

It makes a duplicate of the file descriptor given in oldfd using the descriptor number supplied in newfd. If the file descriptor specified in newfd is already open, it closes it first.

The dup2() system call performs the same task as dup(), but instead of using the lowest-numbered unused file descriptor, it uses the file descriptor number specified in newfd.

```
int myDup2(int oldfd, int newfd)
{
    if (oldfd == newfd)
    {
        if (fcntl(oldfd, F_GETFL) == -1)
        {
            errno = EBADF;
            return -1;
        }
        return newfd;
    }

    if (fcntl(newfd, F_GETFD) != -1)
    {
        if (close(newfd) == -1){
            return -1;
        }
    }

    int result = fcntl(oldfd, F_DUPFD, newfd);
    if (result == -1)
    {
        errno = EBADF;
        return -1;
    }
    return result;
}
```

→ If newfd is bad file descriptor ( for ex -5) gives an error

→ If oldfd is a valid file descriptor, and newfd has the same value as oldfd, then dup2() does nothing, and returns newfd. (man dup2 )

→ newfd is already open, it closes it first.

→ If oldfd is bad file descriptor ( for ex -5) gives an error

!!! Test cases for mydup and mydup2 functions that I wrote in part 2, as well as the standard dup and dup2 test cases, are located in part 3.

## PART-3

This section contains the tests for the dup functions I wrote and the standard dup functions.

- 1) For each test, I first opened a txt file and wrote some text into it.
- 2) Then, I applied my dup method and created fd2.
- 3) I printed the offset and inode values of fd1 and fd2 on the screen, thus proving that my function works.
- 4) Finally, I used lseek to reset the offset. I read the file that I wrote with fd1 using fd2 and printed it on the screen.

### TEST SUCCES MY DUP2

```
fd1 = open("test1.txt", O_CREAT | O_RDWR, 0644);

printf("fd1 writing test1.txt -> Gebze Technical University\n");
write(fd1, "Gebze Technical University\0", 27);

if ((fd2 = myDup2(fd1, 21)) == -1)
    perror("myDup2");
```

```
fstat(fd1, &fileStat1);
fstat(fd2, &fileStat2);
printf("fd1 inode value = %ld fd2 inode value = %ld\n",
      fileStat1.st_ino, fileStat2.st_ino);
```

```
printf("offset fd1 = %ld, offset fd2 = %ld\n",
      lseek(fd1, 0, SEEK_CUR), lseek(fd2, 0, SEEK_CUR));
```

```
read(fd2, arr, 27);
printf("fd2 reading test1.txt -> %s\n", arr);
```

\*\*\*\*TEST SUCCESS MY DUP2 \*\*\*\*

fd1 writing test1.txt -> Gebze Technical University  
file descriptor fd1 = 3, file descriptor fd2 = 21

fd1 inode value = 7366903 fd2 inode value = 7366903

offset fd1 = 27, offset fd2 = 27

reset offset fd1 = 0, offset fd2 = 0

fd2 reading test1.txt -> Gebze Technical University

## TEST EQUALS FDS MY DUP2

!! If oldfd is a valid file descriptor, and newfd has the same value as oldfd, then dup2() does nothing, and returns newfd. (man dup2 )

```
printf("fd1 writing test1.txt -> Gebze Technical University\n");
write(fd1, "Gebze Technical University\0", 27);
if ((fd2 = myDup2(fd1, fd1)) == -1)
    perror("MyDup2");
```

```
fstat(fd1, &fileStat1);
fstat(fd2, &fileStat2);
printf("fd1 inode value = %ld fd2 inode value = %ld\n",
       fileStat1.st_ino, fileStat2.st_ino);
```

```
printf("offset fd1 = %ld, offset fd2 = %ld\n",
       lseek(fd1, 0, SEEK_CUR), lseek(fd2, 0, SEEK_CUR));
```

```
read(fd2, arr, 27);
printf("fd2 reading test1.txt -> %s\n", arr);
```

\*\*\*\*TEST EQUALS MY DUP2 \*\*\*\*

fd1 writing test1.txt -> Gebze Technical University

file descriptor fd1 = 3, file descriptor fd2 = 3

fd1 inode value = 7366903 fd2 inode value = 7366903

offset fd1 = 27, offset fd2 = 27

reset offset fd1 = 0, offset fd2 = 0

fd2 reading test1.txt -> Gebze Technical University

## TEST NOT VALID NEW FD MY DUP2

There can't be a fd called -5, so there is an error

```
if ((fd2 = myDup2(fd1, -5)) == -1){
    printf("new value = -5\n");
    perror("MyDup2");
}
```

\*\*\*\*TEST NOT VALID NEW FD MY DUP2 \*\*\*\*

new value = -5

MyDup2: Bad file descriptor

file descriptor fd1 = 3, return value fd2 = -1

## TEST NOT VALID TWO FD MY DUP2

new and old fd invalid

```
int fd1 = -5;
int fd2;
if ((fd2 = myDup2(fd1, fd1)) == -1)
    perror("myDup2");
```

\*\*\*\*TEST NOT VALID SAME TWO FD MY DUP2 \*\*\*\*

myDup2: Bad file descriptor

file descriptor fd1 = -5, return value fd2 = -1

## TEST SUCCES MY DUP

allocates a new file descriptor that refer to the same open file description as the descriptor oldfd. The new file descriptor number is guaranteed to be the lowest-numbered file descriptor that was unused in the calling process.

```
fd1 = open("test1.txt", O_CREAT | O_RDWR, 0644);

printf("fd1 writing test1.txt -> Gebze Technical University\n");
write(fd1, "Gebze Technical University\0", 27);

if ((fd2 = myDup(fd1)) == -1)
    perror("myDup");
```

```
fstat(fd1, &fileStat1);
fstat(fd2, &fileStat2);
printf("fd1 inode value = %ld fd2 inode value = %ld\n",
      fileStat1.st_ino, fileStat2.st_ino);
```

```
printf("offset fd1 = %ld, offset fd2 = %ld\n",
      lseek(fd1, 0, SEEK_CUR), lseek(fd2, 0, SEEK_CUR));
```

```
read(fd2, arr, 27);
printf("fd2 reading test1.txt -> %s\n", arr);
```

\*\*\*\*TEST SUCCESS MY DUP \*\*\*\*

fd1 writing test1.txt -> Gebze Technical University  
file descriptor fd1 = 3, file descriptor fd2 = 4

fd1 inode value = 7366903 fd2 inode value = 7366903  
offset fd1 = 27, offset fd2 = 27

reset offset fd1 = 0, offset fd2 = 0

fd2 reading test1.txt -> Gebze Technical University

## TEST UNSUCCESS MY DUP

```
int fd1, fd2;
fd1 = -5;
if ((fd2 = myDup(fd1)) == -1)
    perror("myDup");
```

\*\*\*\*\*TEST UNSUCCESS MY DUP \*\*\*\*

myDup: Bad file descriptor

file descriptor fd1 = -5, file descriptor fd2 = -1



## TEST SUCCES STANDART DUP2

```
fd1 = open("test1.txt", O_CREAT | O_RDWR, 0644);

printf("fd1 writing test1.txt -> Gebze Technical University\n");
write(fd1, "Gebze Technical University\0", 27);

if ((fd2 = dup2(fd1, 21)) == -1)
    perror("myDup2");
```

```
fstat(fd1, &fileStat1);
fstat(fd2, &fileStat2);
printf("fd1 inode value = %ld fd2 inode value = %ld\n",
       fileStat1.st_ino, fileStat2.st_ino);
```

```
printf("offset fd1 = %ld,offset fd2 = %ld\n",
       lseek(fd1, 0, SEEK_CUR), lseek(fd2, 0, SEEK_CUR));
```

```
read(fd2, arr, 27);
printf("fd2 reading test1.txt -> %s\n", arr);
```

```
****TEST SUCCESS STANDART DUP2 ****
fd1 writing test1.txt -> Gebze Technical University
file descriptor fd1 = 3, file descriptor fd2 = 21
fd1 inode value = 7366903 fd2 inode value = 7366903
offset fd1 = 27,offset fd2 = 27
reset offset fd1 = 0,offset fd2 = 0
fd2 reading test1.txt -> Gebze Technical University
```

## TEST EQUALS FDS STANDART DUP2

!! If oldfd is a valid file descriptor, and newfd has the same value as oldfd, then dup2() does nothing, and returns newfd. (man dup2 )

```
fd1 = open("test1.txt", O_CREAT | O_RDWR, 0644);
if ((fd2 = dup2(fd1, fd1)) == -1)
    perror("Dup2");

printf("fd1 writing test1.txt -> Gebze Technical University\n");
write(fd1, "Gebze Technical University\0", 27);
```

```
fstat(fd1, &fileStat1);
fstat(fd2, &fileStat2);
printf("fd1 inode value = %ld fd2 inode value = %ld\n",
       fileStat1.st_ino, fileStat2.st_ino);
```

```
printf("offset fd1 = %ld,offset fd2 = %ld\n",
       lseek(fd1, 0, SEEK_CUR), lseek(fd2, 0, SEEK_CUR));
```

```
read(fd2, arr, 27);
printf("fd2 reading test1.txt -> %s\n", arr);
```

```
****TEST EQUALS STANDART DUP2 ****
fd1 writing test1.txt -> Gebze Technical University
file descriptor fd1 = 3, file descriptor fd2 = 3
fd1 inode value = 7366903 fd2 inode value = 7366903
offset fd1 = 27,offset fd2 = 27
reset offset fd1 = 0,offset fd2 = 0
fd2 reading test1.txt -> Gebze Technical University
```

## TEST NOT VALID NEW FD STANDART DUP2

There can't be a fd called -5, so there is an error

```
fd1 = open("test1.txt", O_CREAT | O_RDWR, 0644);
fd2 = dup2(fd1, -5);

if ((fd2 = dup2(fd1, -5)) == -1)
    perror("dup2");
```

```
****TEST NOT VALID NEW FD STANDART DUP2 ****
dup2: Bad file descriptor
file descriptor fd1 = 3, file descriptor fd2 = -1
```

## TEST NOT VALID TWO FD STANDART DUP2

new and old fd invalid

```
int fd1 = -5;
int fd2;

if ((fd2 = dup2(fd1, fd1)) == -1)
    perror("dup2");
```

```
****TEST NOT VALID SAME TWO FD STANDART DUP2 ****
dup2: Bad file descriptor
file descriptor fd1 = -5, file descriptor fd2 = -1
```

## TEST SUCCES STANDART DUP

allocates a new file descriptor that refer to the same open file description as the descriptor oldfd. The new file descriptor number is guaranteed to be the lowest-numbered file descriptor that was unused in the calling process.

```
fd1 = open("test1.txt", O_CREAT | O_RDWR, 0644);

printf("fd1 writing test1.txt -> Gebze Technical University\n");
write(fd1, "Gebze Technical University\0", 27);

if((fd2 = dup(fd1))==-1)
    perror("dup");
```

```
fstat(fd1, &fileStat1);
fstat(fd2, &fileStat2);
printf("fd1 inode value = %ld fd2 inode value = %ld\n",
      fileStat1.st_ino, fileStat2.st_ino);
```

```
printf("offset fd1 = %ld,offset fd2 = %ld\n",
      lseek(fd1, 0, SEEK_CUR), lseek(fd2, 0, SEEK_CUR));
```

```
read(fd2, arr, 27);
printf("fd2 reading test1.txt -> %s\n", arr);
```

```
****TEST SUCCESS STANDART DUP ****
fd1 writing test1.txt -> Gebze Technical University
file descriptor fd1 = 3, file descriptor fd2 = 4
fd1 inode value = 7366903 fd2 inode value = 7366903
offset fd1 = 27,offset fd2 = 27
reset offset fd1 = 0,offset fd2 = 0
fd2 reading test1.txt -> Gebze Technical University
```

## TEST UNSUCCES STANDART DUP

```
fd1 = -5;

if ((fd2 = dup(fd1)) == -1)
    perror("dup");
```

```
****TEST UNSUCCESS STANDART DUP ****
dup: Bad file descriptor
file descriptor fd1 = -5, file descriptor fd2 = -1
```



## EXECUTE FILE TUTORIAL

**Terminal → “ make ” this command gives 2 file appendMeMore.o and part2andpart3.o**

**for part 1 →** ./appendMeMore f1 1000000 & ./appendMeMore f1 1000000

or

./appendMeMore f2 1000000 x & ./appendMeMore f2 1000000 x

**for part 2 and part 3 →** ./part2andpart3