Barış Ayyıldız
1901042252

System Programming - Homework 1

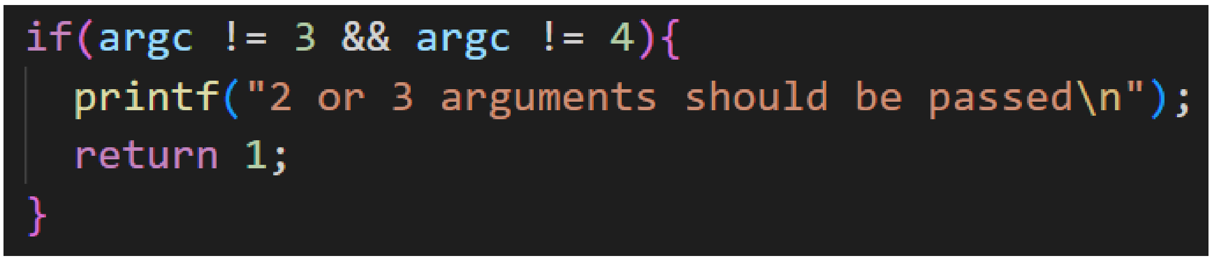
Spring 2023

Table Of Contents

Question 1

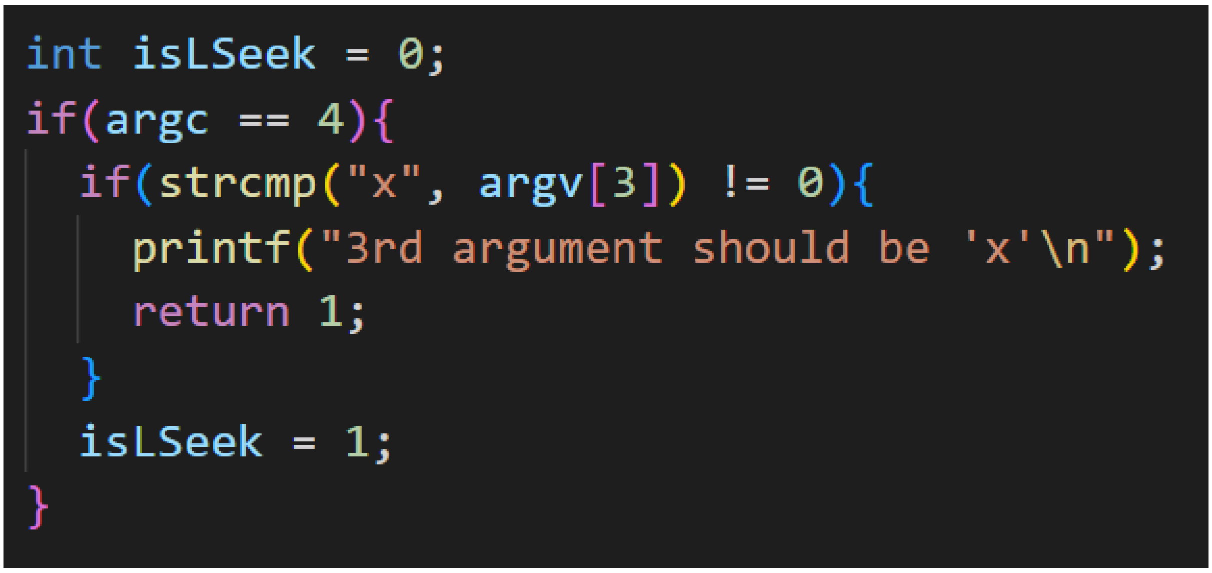
Explanation of the code

In this function first I checked the number of arguments that are passed from the user. If it is not equal to 3 or 4, the function stops there.



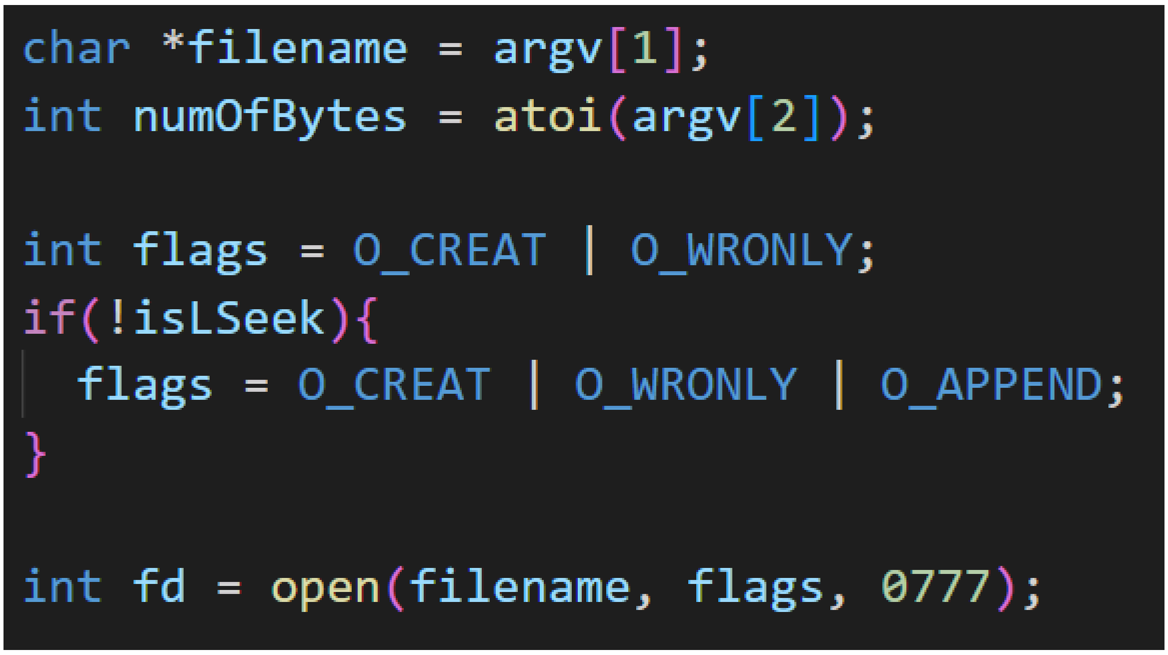
```c
if(argc != 3 && argc != 4){
  printf("2 or 3 arguments should be passed\n");
  return 1;
}
```

Then I check if the fourth argument is given or not. If it is given as ‘x’ I set the sLeek flag as 1. If it is given but it is not equal to ‘x’ then again the function stops executing.



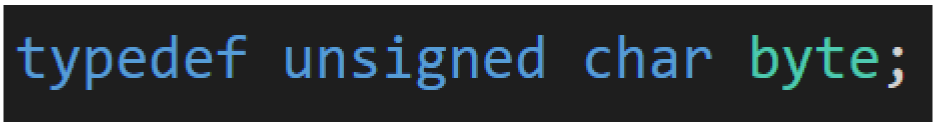
```c
int isLSeek = 0;
if(argc == 4){
  if(strcmp("x", argv[3]) != 0){
    printf("3rd argument should be 'x'\n");
    return 1;
  }
  isLSeek = 1;
}
```

Then I extract the filename and number of bytes that is given by the user and set the flags accordingly. If the fourth argument is provided by the user I omit the O_APPEND flag.

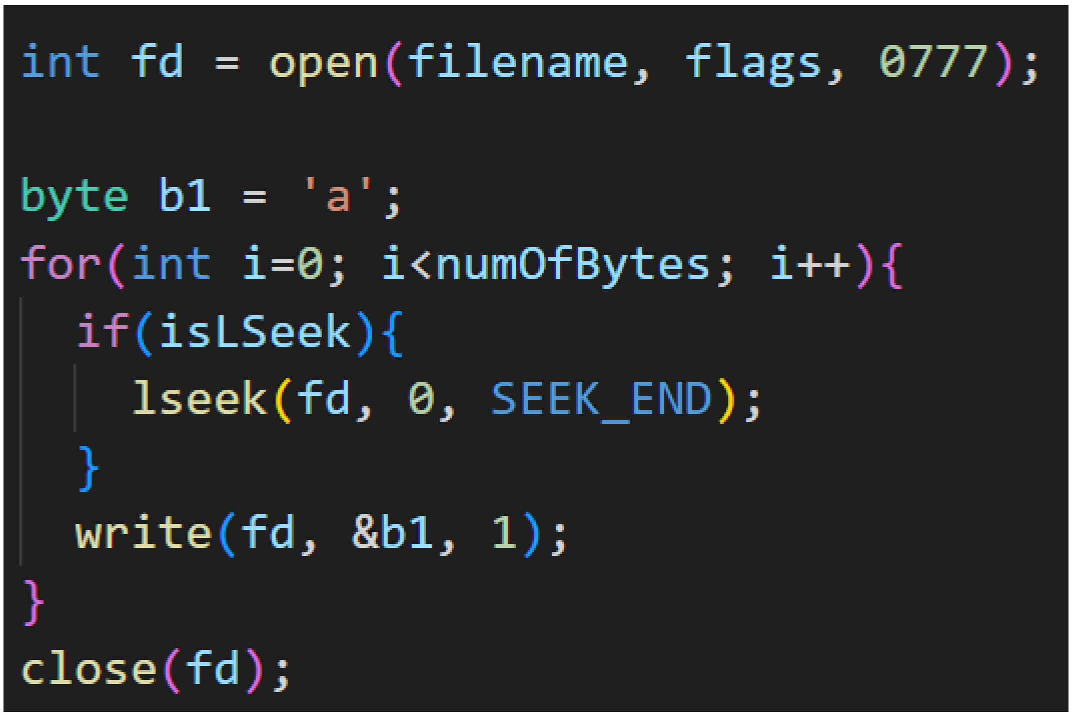


```c
char *filename = argv[1];
int numOfBytes = atoi(argv[2]);

int flags = O_CREAT | O_WRONLY;
if(!isLSeek){
  flags = O_CREAT | O_WRONLY | O_APPEND;
}

int fd = open(filename, flags, 0777);
```

Then I open the file with 777 permissions and start writing a single byte to that file for a number of bytes times. If the isLSeek flag is true we execute this lseek function which makes the cursor jump at the end of the file.



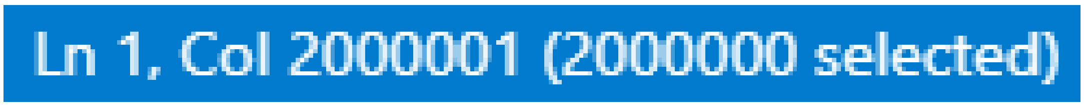
```c
typedef unsigned char byte;
```



```c
int fd = open(filename, flags, 0777);

byte b1 = 'a';
for(int i=0; i<numOfBytes; i++){
  if(isLSeek){
    lseek(fd, 0, SEEK_END);
  }
  write(fd, &b1, 1);
}
close(fd);
```

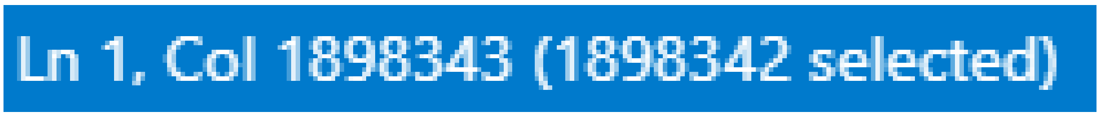
Testing

First I ran the code **./q1 f1 1000000 & ./q1 f1 1000000** and as you can see it wrote 2 million bytes to the file f1.

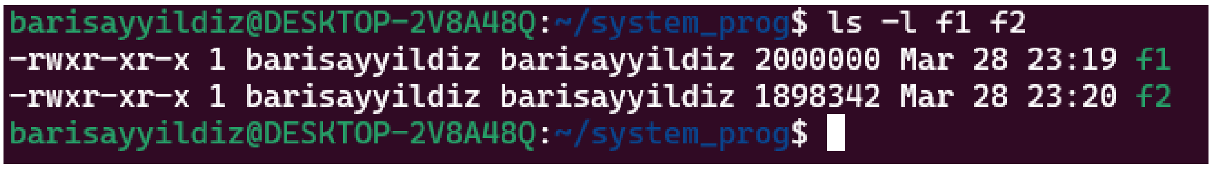


Ln 1, Col 2000001 (2000000 selected)

The I ran **./q1 f2 1000000 x & ./q1 f2 1000000 x** and it only wrote 1.898.342 bytes.



Ln 1, Col 1898343 (1898342 selected)



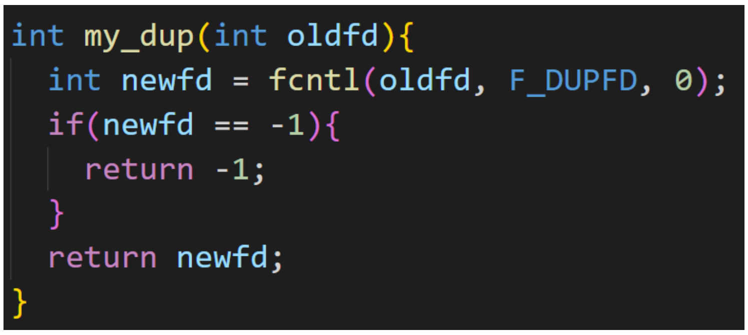
```
barisayyildiz@DESKTOP-2V8A48Q:~/system_prog$ ls -l f1 f2
-rwxr-xr-x 1 barisayyildiz barisayyildiz 2000000 Mar 28 23:19 f1
-rwxr-xr-x 1 barisayyildiz barisayyildiz 1898342 Mar 28 23:20 f2
barisayyildiz@DESKTOP-2V8A48Q:~/system_prog$
```

In this test case we ran two different processes at the same time and we lost some data in the second one. The reason is that only the O_APPEND flag guarantees atomicity. In the second one we manually adjusted the cursor in the file and in some cases two processes intervened with each other and override them.

Question 2

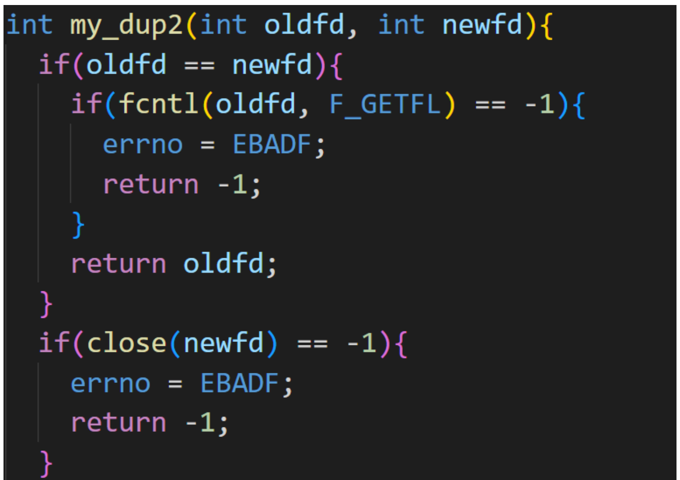
Explanation of the code

I have implemented **my_dup** with this piece of code. It copies old file descriptor using fcntl function with F_DUPFD flag and it returns -1 if fnctl doesn't work for that file descriptor



```c
int my_dup(int oldfd){
  int newfd = fcntl(oldfd, F_DUPFD, 0);
  if(newfd == -1){
    return -1;
  }
  return newfd;
}
```

And this is **my_dup2** function



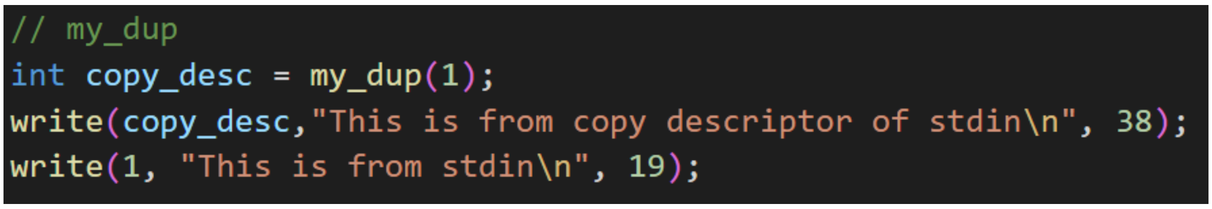
```c
int my_dup2(int oldfd, int newfd){
  if(oldfd == newfd){
    if(fcntl(oldfd, F_GETFL) == -1){
      errno = EBADF;
      return -1;
    }
    return oldfd;
  }
  if(close(newfd) == -1){
    errno = EBADF;
    return -1;
  }
```

First it checks if the new file descriptor and the old file descriptors are the same. If they are it also checks if the old file descriptor is valid. If it is not valid, an error is returned, otherwise the old file descriptor is returned, since old and new are the same file descriptors.

Then I close the new file descriptor and return -1 if closing file fails. And in the end I use the fcntl function, which basically makes the old file descriptor point to the file that the new file descriptor is pointing to.

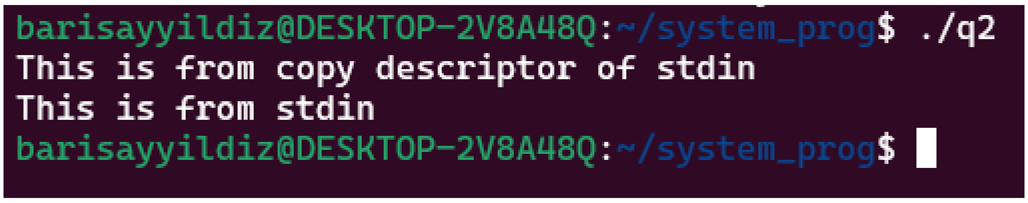
Testing

Here is an example of my_dup function. I create a copy descriptor and duplicate stdout to that.



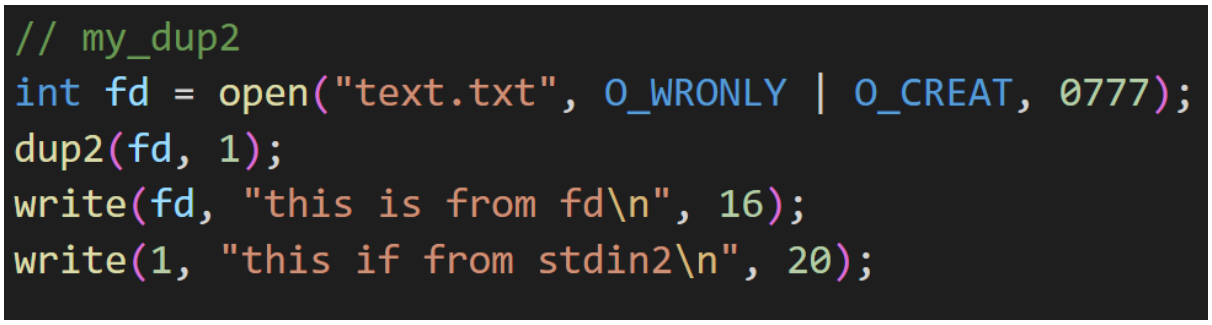
```c
// my_dup
int copy_desc = my_dup(1);
write(copy_desc,"This is from copy descriptor of stdin\n", 38);
write(1, "This is from stdin\n", 19);
```

As a result both of the strings were written to terminal(standard input).



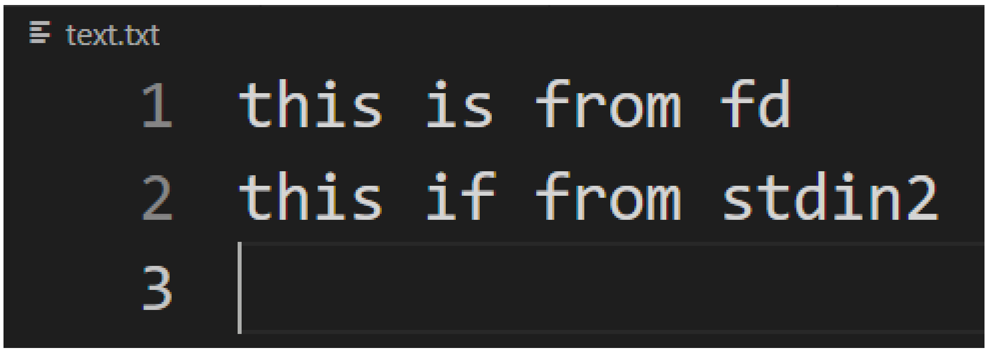
```
barisayyildiz@DESKTOP-2V8A48Q:~/system_prog$ ./q2
This is from copy descriptor of stdin
This is from stdin
barisayyildiz@DESKTOP-2V8A48Q:~/system_prog$
```

And in the test case for my_dup2, I open a file named ‘text.txt’ and set it to file descriptor 1.



```c
// my_dup2
int fd = open("text.txt", O_WRONLY | O_CREAT, 0777);
dup2(fd, 1);
write(fd, "this is from fd\n", 16);
write(1, "this if from stdin2\n", 20);
```

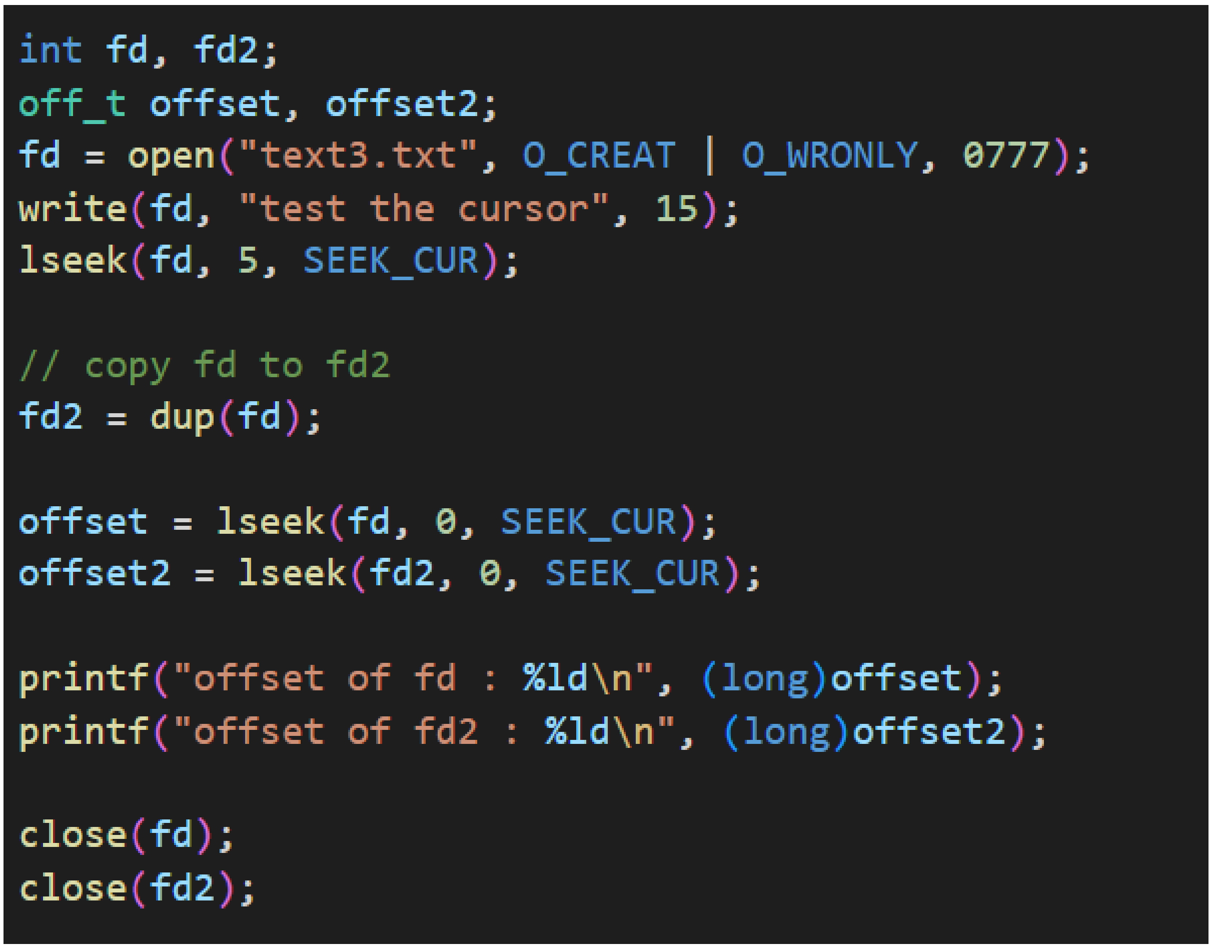
And both lines are written in text.txt


```
text.txt
this is from fd
this if from stdin2
```

Question 3

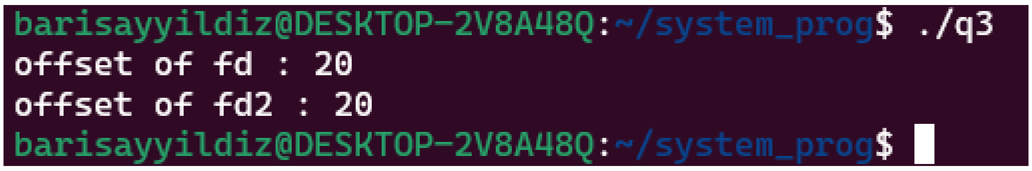
Explanation of code

In this question I created two file descriptors, opened a file with one of them. Wrote a string with 15 characters and moved the cursor to right by 5. Then I duplicated fd to fd2 and checked their offset.


```c
int fd, fd2;
off_t offset, offset2;
fd = open("text3.txt", O_CREAT | O_WRONLY, 0777);
write(fd, "test the cursor", 15);
lseek(fd, 5, SEEK_CUR);

// copy fd to fd2
fd2 = dup(fd);

offset = lseek(fd, 0, SEEK_CUR);
offset2 = lseek(fd2, 0, SEEK_CUR);

printf("offset of fd : %ld\n", (long)offset);
printf("offset of fd2 : %ld\n", (long)offset2);

close(fd);
close(fd2);
```

Testing

As you can see their offsets are the same



```
barisayyildiz@DESKTOP-2V8A48Q:~/system_prog$ ./q3
offset of fd : 20
offset of fd2 : 20
barisayyildiz@DESKTOP-2V8A48Q:~/system_prog$
```