CSE 344

HW1

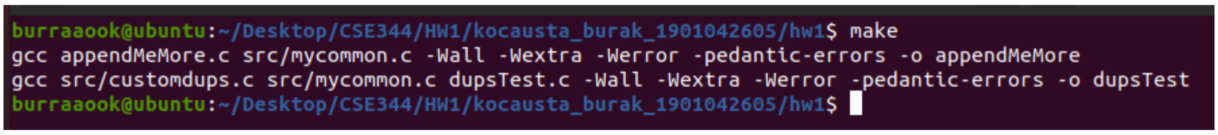
Burak Kocausta

1901042605

Content

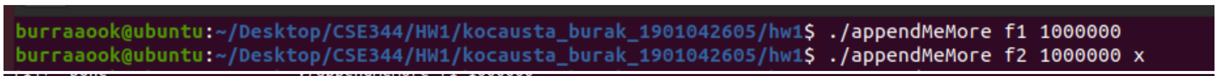
1- Compilation and Run

All parts are compiled with ‘make’ command.

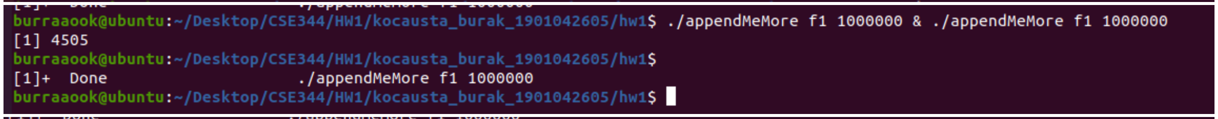


```
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ make
gcc appendMeMore.c src/mycommon.c -Wall -Wextra -Werror -pedantic-errors -o appendMeMore
gcc src/customdups.c src/mycommon.c dupsTest.c -Wall -Wextra -Werror -pedantic-errors -o dupsTest
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$
```

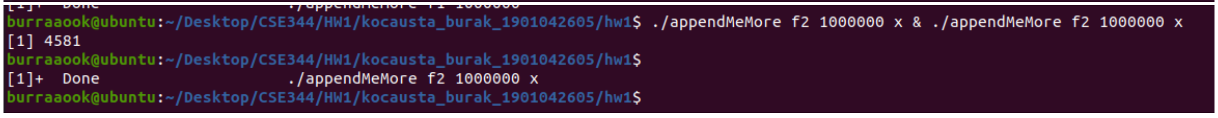
1- Run PART1



```
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ ./appendMeMore f1 1000000
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ ./appendMeMore f2 1000000 x
```



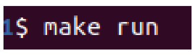
```
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ ./appendMeMore f1 1000000 & ./appendMeMore f1 1000000
[1] 4505
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$
[1]+  Done                    ./appendMeMore f1 1000000
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$
```



```
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ ./appendMeMore f2 1000000 x & ./appendMeMore f2 1000000 x
[1] 4581
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$
[1]+  Done                    ./appendMeMore f2 1000000 x
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$
```

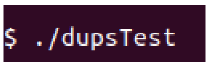
2- Run PART2,3

- File name arguments can be given, if not default files are used for testing.



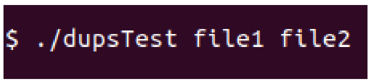
```
$ make run
```

or



```
$ ./dupsTest
```

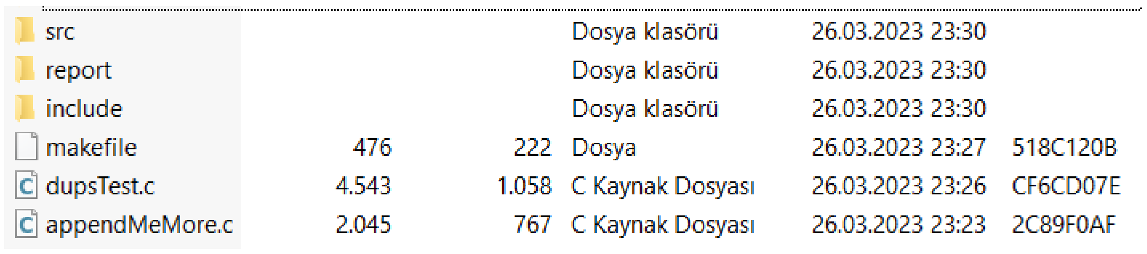
or



```
$ ./dupsTest file1 file2
```

2- Files

After opening the zip file, and entering the hw1 directory. There are include, src, report directories two .c files and makefile.



src			Dosya klasörü	26.03.2023 23:30	
report			Dosya klasörü	26.03.2023 23:30	
include			Dosya klasörü	26.03.2023 23:30	
makefile	476	222	Dosya	26.03.2023 23:27	518C120B
dupsTest.c	4.543	1.058	C Kaynak Dosyası	26.03.2023 23:26	CF6CD07E
appendMeMore.c	2.045	767	C Kaynak Dosyası	26.03.2023 23:23	2C89F0AF

main files: appendMeMore.c is the main file for the first part, dupsTest.c is the main file for dups functions.

src: Implementations of header files -> mycommon.c, and customdubs.c

include: Header files -> mycommon.h, and customdubs.h

report: Report in pdf format

makefile: make compiles the program, make clean removes target files, and make run runs the dupsTest target file. (appendMeMore doesn't run because it must be tested as indicated in the pdf.)

3- Detailed Explanation of The Solution Approach

1) Appending Bytes

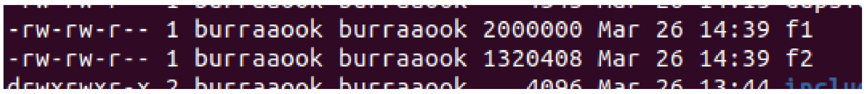
Firstly, I determined a default byte as '0' character. And I make maximum file size can be given by the user as 10 GB. Then I made the error handling for the arguments, program won't accept more than necessary arguments, or less than minimum. I wrote a function for converting string into a size_t variable for representing bytes. It also checks the wrong input situation. Also, program handles all errors for system calls. Every system call is checked, and error is printed using perror for those situations.

According to the x arguments existence, program flows differently. If x does not exist, file is opened on the append mode. If it exists, append mode is omitted. I opened the file on 0666 mode, so read and write permissions are given to all users. Then file is opened, and writing part starts according to the x argument.

If x argument exists, lseek system call moves the position to the end of the file in each write. With this way, it is expected to append without append flag, but it would not work as expected when 2 process works in parallel on the same file, it is explained later.

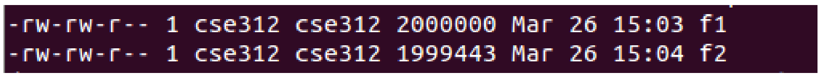
If x argument does not exist, it simply appends all the bytes. Each byte is appended one at a time.

Size of the files difference explanation:



```
-rw-rw-r-- 1 burraaook burraaook 2000000 Mar 26 14:39 f1
-rw-rw-r-- 1 burraaook burraaook 1320408 Mar 26 14:39 f2
```

(ubuntu 20.04)



```
-rw-rw-r-- 1 cse312 cse312 2000000 Mar 26 15:03 f1
-rw-rw-r-- 1 cse312 cse312 1999443 Mar 26 15:04 f2
```

(ubuntu 14.04)

- If x argument does not exist, there is an append flag, therefore it directly appends without overwriting each other’s data. It is done atomically. Because it is guaranteed on each write call that position is the end of the file. There is no other system call between them. For the other condition they might be overwriting each other’s data. Because there are 2 system call, and 2 process are running. One of them uses lseek to set the position to end of the file, but other process might write before this process writes to the file. In this case it is possible to write over other process’s byte. Because of this overlapping, f2 file’s size is less than f1. It might be change on every run because, kernel decides which system call is executed first. Kernel could decide switch between lseek and write system calls, it causes race condition. That part of the code is a critical region, if homework weren’t aim that we should see that, I would solve the problem using semaphores, or locks.

2) Implementing dup, dup2 Functions

Since there are dup, and dup2 functions I used different names for these functions such as custom_dup, and custom_dup2.

Firstly I implemented custom_dup function, it is easy because it simply redirects the file descriptor with lowest possible file descriptor. Since it is expected to do this using fcntl, with giving F_DUPFD operation, and giving 0 as an argument to fcntl system call, it behaves like dup function. 0 means starting from 0, it searches minimum possible file descriptor.

For the custom_dup2 function, it required a little bit more operation and check. dup2 function, redirects the file descriptor to the given new file descriptor. There are some conditions for doing that, like if new file descriptor is already opened it must be closed silently. It must be checked if old file descriptor is valid or not. It must also handle the case that old file descriptor and new file descriptor are equal. I initially checked the case that old file descriptor is valid or not, I do this with fcntl system call using F_GETFD operation. According to its return value, it can be determined that system call is invalid or not, and set errno to the EBADF if it is invalid. I checked for equality of old and new file descriptors, if they are equal one of them simply returned, and validity of them is already checked, so it can be used for checking file descriptors validity. After that I checked if new file descriptor is already used or not, if it is used then I close that file descriptor. After that fcntl is again used for duplicating the file descriptor, operation is F_DUPFD again but this time argument is new file descriptor.

3) Testing dup Implementations

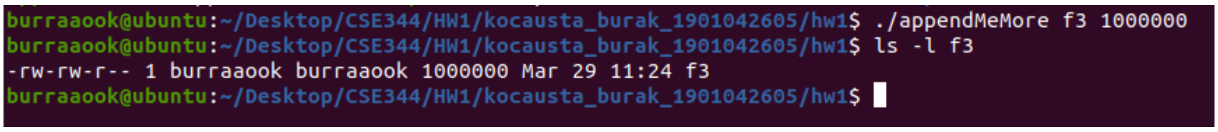
I wrote a program for testing part 2, and verifying that duplicated file descriptors share same offset value, and open files. If provided it can take 2 filename arguments, otherwise default files will be used for testing.

I used 4 file descriptors, fd1 is initialized during opening, fd2 is initialized with custom_dup function, then fd3 is initialized using custom_dup2 function. Each step, it is written to the file that which file descriptor is used to verify it works correct. Then I set the offset using lseek of fd1, after that printed all three descriptors offset values to verify they share same offset value.

After that other cases are tested such as, bad file descriptor as old fd, already used file descriptor as new fd, and same file descriptor as old and new fd (both invalid and valid file descriptors).

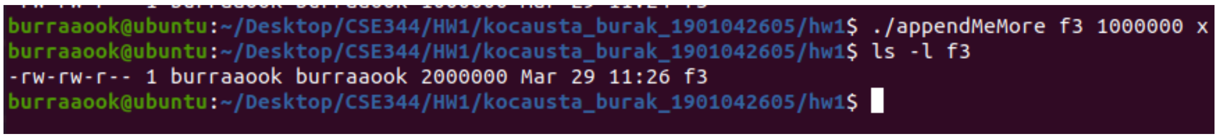
4- Test Cases and Results

PART1:



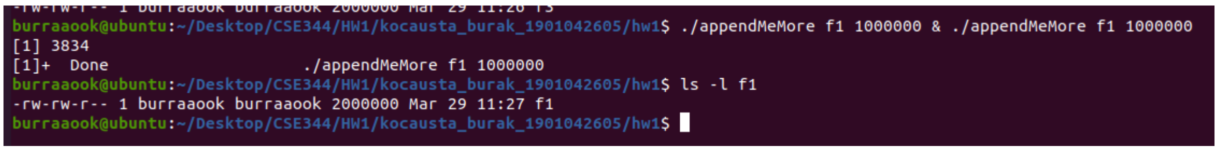
```
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ ./appendMeMore f3 1000000
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ ls -l f3
-rw-rw-r-- 1 burraaook burraaook 1000000 Mar 29 11:24 f3
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ 
```

1 million byte is appended to f3, and file size is 1 million.



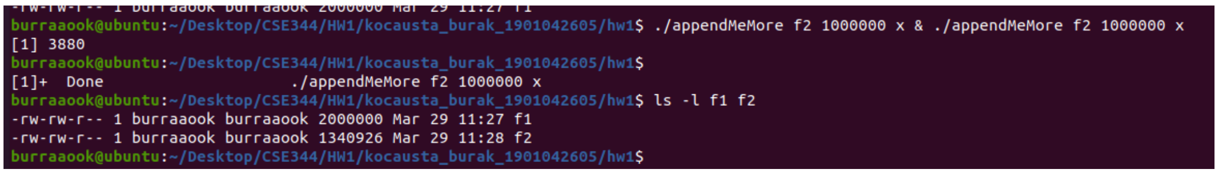
```
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ ./appendMeMore f3 1000000 x
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ ls -l f3
-rw-rw-r-- 1 burraaook burraaook 2000000 Mar 29 11:26 f3
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ 
```

1 million more is appended using x argument to f3, and size is 2 million.



```
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ ./appendMeMore f1 1000000 & ./appendMeMore f1 1000000
[1] 3834
[1]+  Done                    ./appendMeMore f1 1000000
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ ls -l f1
-rw-rw-r-- 1 burraaook burraaook 2000000 Mar 29 11:27 f1
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ 
```

2 instance of the program runs without x argument, file size is 2 million.

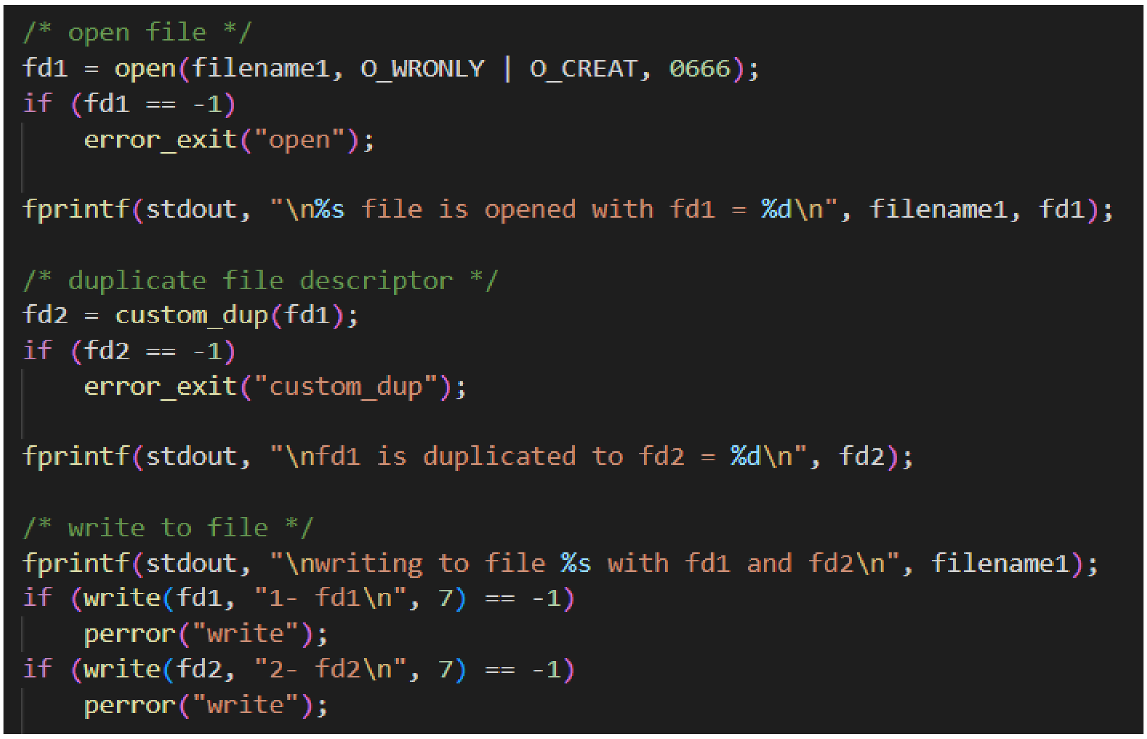


```
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ ./appendMeMore f2 1000000 x & ./appendMeMore f2 1000000 x
[1] 3880
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$
[1]+  Done                    ./appendMeMore f2 1000000 x
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ ls -l f1 f2
-rw-rw-r-- 1 burraaook burraaook 2000000 Mar 29 11:27 f1
-rw-rw-r-- 1 burraaook burraaook 1340926 Mar 29 11:28 f2
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$
```

2 instances of the process runs with x argument for file f2, and file size is less than f1 as expected, and explained in the explanation section.

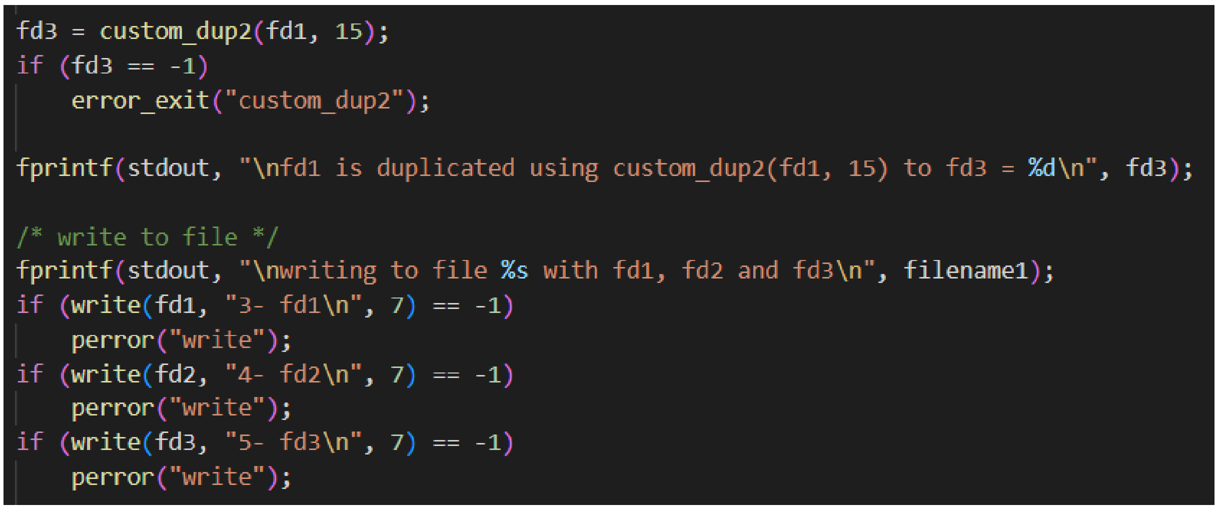
PART2 & PART3:

Cases



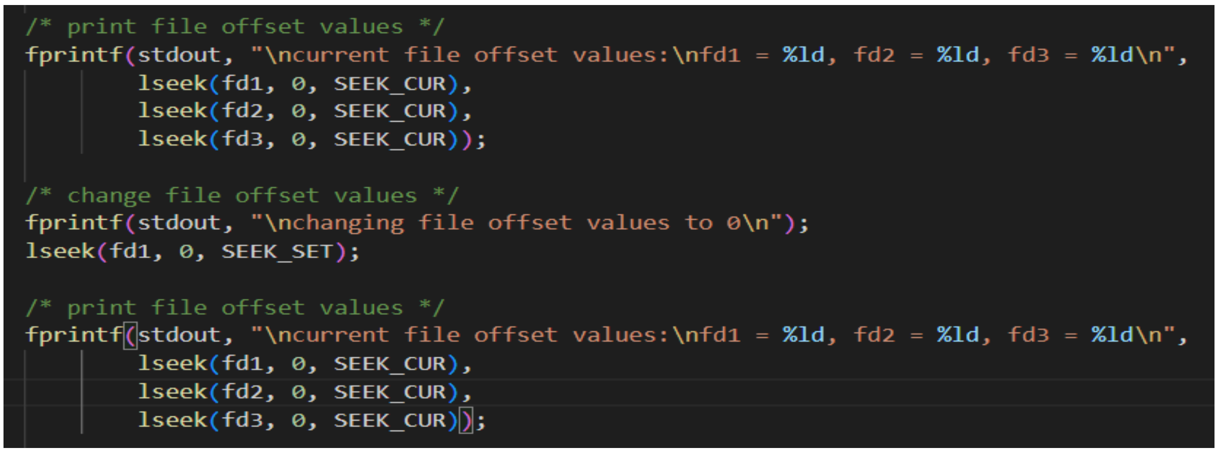
```c
/* open file */
fd1 = open(filename1, O_WRONLY | O_CREAT, 0666);
if (fd1 == -1)
    error_exit("open");

fprintf(stdout, "\n%s file is opened with fd1 = %d\n", filename1, fd1);

/* duplicate file descriptor */
fd2 = custom_dup(fd1);
if (fd2 == -1)
    error_exit("custom_dup");

fprintf(stdout, "\nfd1 is duplicated to fd2 = %d\n", fd2);

/* write to file */
fprintf(stdout, "\nwriting to file %s with fd1 and fd2\n", filename1);
if (write(fd1, "1- fd1\n", 7) == -1)
    perror("write");
if (write(fd2, "2- fd2\n", 7) == -1)
    perror("write");
```

open fd1, redirect fd1 to fd2, then write to file using both.



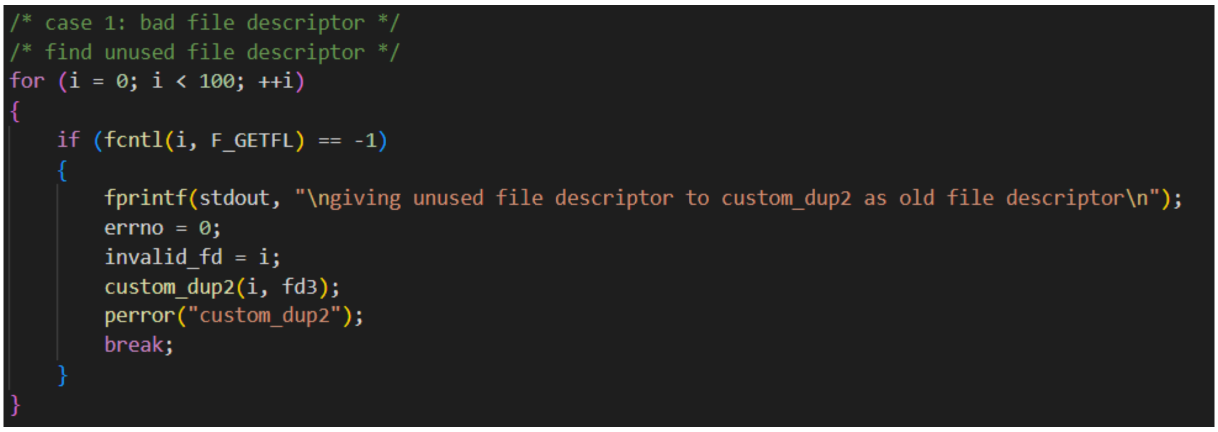
```c
fd3 = custom_dup2(fd1, 15);
if (fd3 == -1)
    error_exit("custom_dup2");

fprintf(stdout, "\nfd1 is duplicated using custom_dup2(fd1, 15) to fd3 = %d\n", fd3);

/* write to file */
fprintf(stdout, "\nwriting to file %s with fd1, fd2 and fd3\n", filename1);
if (write(fd1, "3- fd1\n", 7) == -1)
    perror("write");
if (write(fd2, "4- fd2\n", 7) == -1)
    perror("write");
if (write(fd3, "5- fd3\n", 7) == -1)
    perror("write");
```

fd1 is redirected to fd3 using custom_dup2, and they are called with write function.



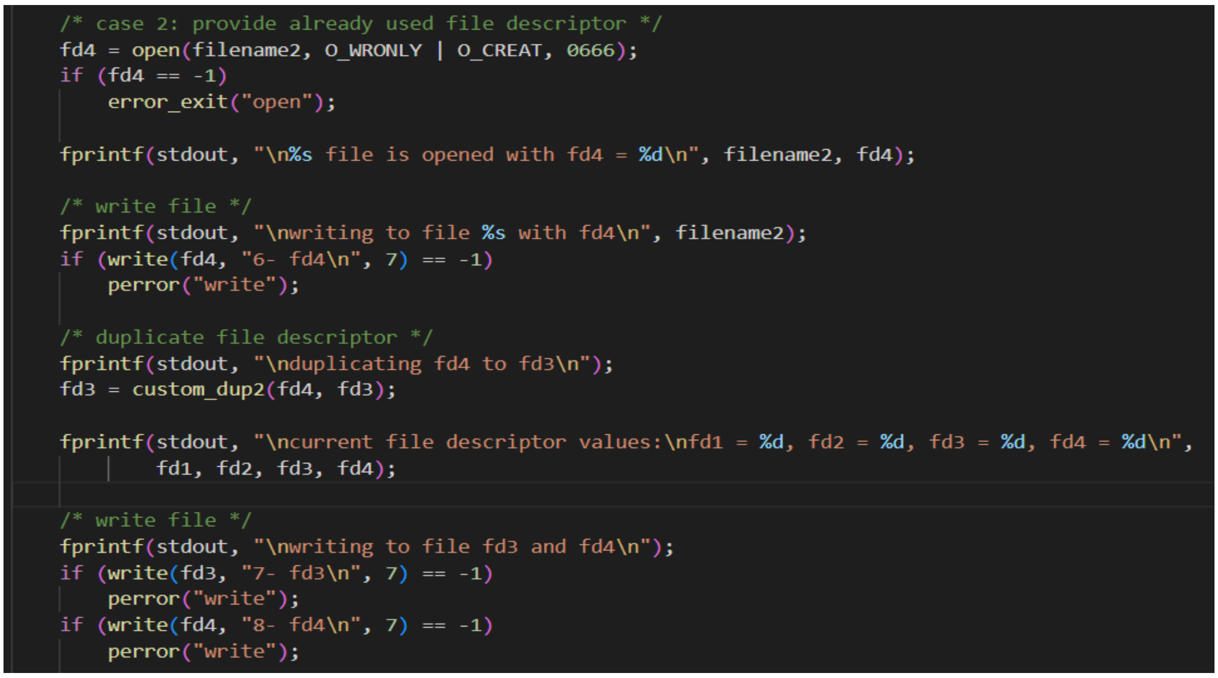
```c
/* print file offset values */
fprintf(stdout, "\ncurrent file offset values:\nfd1 = %ld, fd2 = %ld, fd3 = %ld\n",
        lseek(fd1, 0, SEEK_CUR),
        lseek(fd2, 0, SEEK_CUR),
        lseek(fd3, 0, SEEK_CUR));

/* change file offset values */
fprintf(stdout, "\nchanging file offset values to 0\n");
lseek(fd1, 0, SEEK_SET);

/* print file offset values */
fprintf(stdout, "\ncurrent file offset values:\nfd1 = %ld, fd2 = %ld, fd3 = %ld\n",
        lseek(fd1, 0, SEEK_CUR),
        lseek(fd2, 0, SEEK_CUR),
        lseek(fd3, 0, SEEK_CUR));
```

change fd1’s offset for checking if they share same file descriptor.



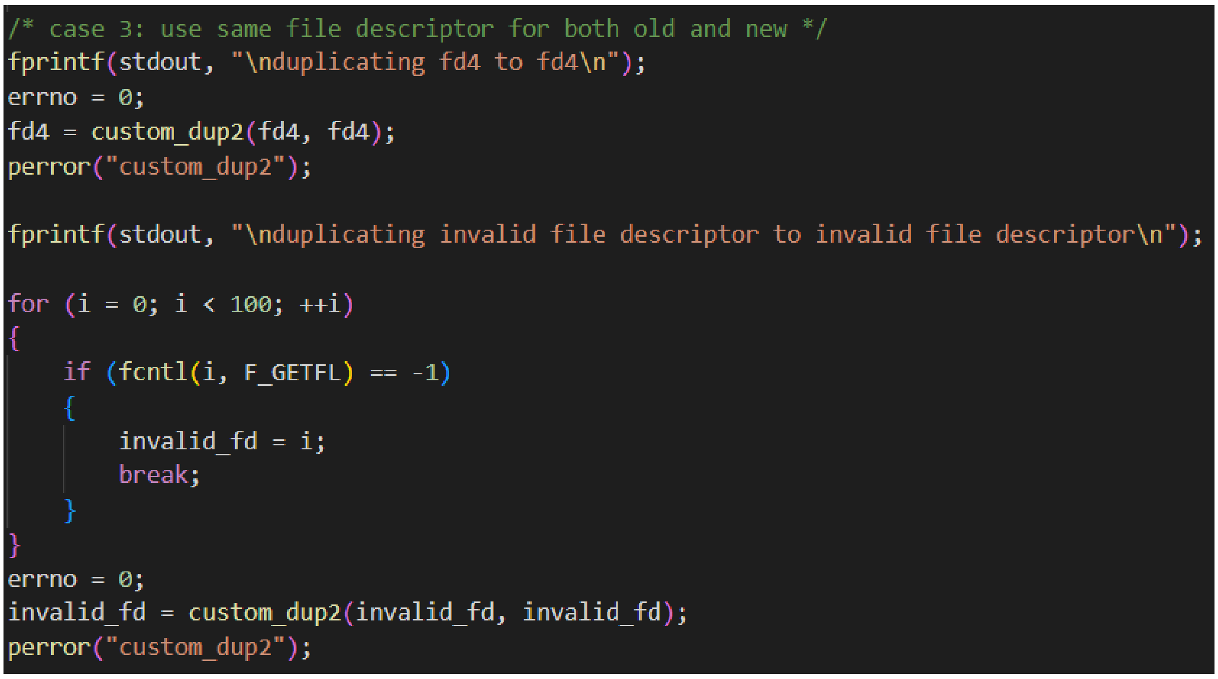
```c
/* case 1: bad file descriptor */
/* find unused file descriptor */
for (i = 0; i < 100; ++i)
{
    if (fcntl(i, F_GETFL) == -1)
    {
        fprintf(stdout, "\ngiving unused file descriptor to custom_dup2 as old file descriptor\n");
        errno = 0;
        invalid_fd = i;
        custom_dup2(i, fd3);
        perror("custom_dup2");
        break;
    }
}
```

checking if errno is set properly, when invalid oldfd provided.



```c
    /* case 2: provide already used file descriptor */
    fd4 = open(filename2, O_WRONLY | O_CREAT, 0666);
    if (fd4 == -1)
        error_exit("open");

    fprintf(stdout, "\n%s file is opened with fd4 = %d\n", filename2, fd4);

    /* write file */
    fprintf(stdout, "\nwriting to file %s with fd4\n", filename2);
    if (write(fd4, "6- fd4\n", 7) == -1)
        perror("write");

    /* duplicate file descriptor */
    fprintf(stdout, "\nduplicating fd4 to fd3\n");
    fd3 = custom_dup2(fd4, fd3);

    fprintf(stdout, "\ncurrent file descriptor values:\nfd1 = %d, fd2 = %d, fd3 = %d, fd4 = %d\n",
            fd1, fd2, fd3, fd4);

    /* write file */
    fprintf(stdout, "\nwriting to file fd3 and fd4\n");
    if (write(fd3, "7- fd3\n", 7) == -1)
        perror("write");
    if (write(fd4, "8- fd4\n", 7) == -1)
        perror("write");
```

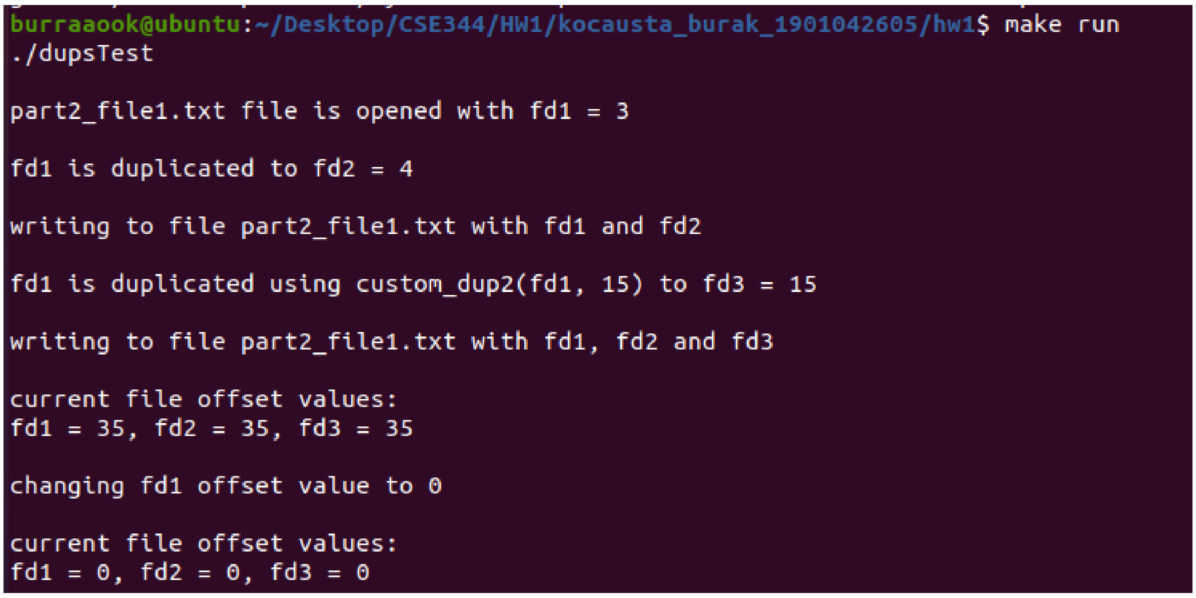
checks if new file descriptors already used case for dup2. Writing to file again to check if it is redirected properly.



```c
/* case 3: use same file descriptor for both old and new */
fprintf(stdout, "\nduplicating fd4 to fd4\n");
errno = 0;
fd4 = custom_dup2(fd4, fd4);
perror("custom_dup2");

fprintf(stdout, "\nduplicating invalid file descriptor to invalid file descriptor\n");

for (i = 0; i < 100; ++i)
{
    if (fcntl(i, F_GETFL) == -1)
    {
        invalid_fd = i;
        break;
    }
}
errno = 0;
invalid_fd = custom_dup2(invalid_fd, invalid_fd);
perror("custom_dup2");
```

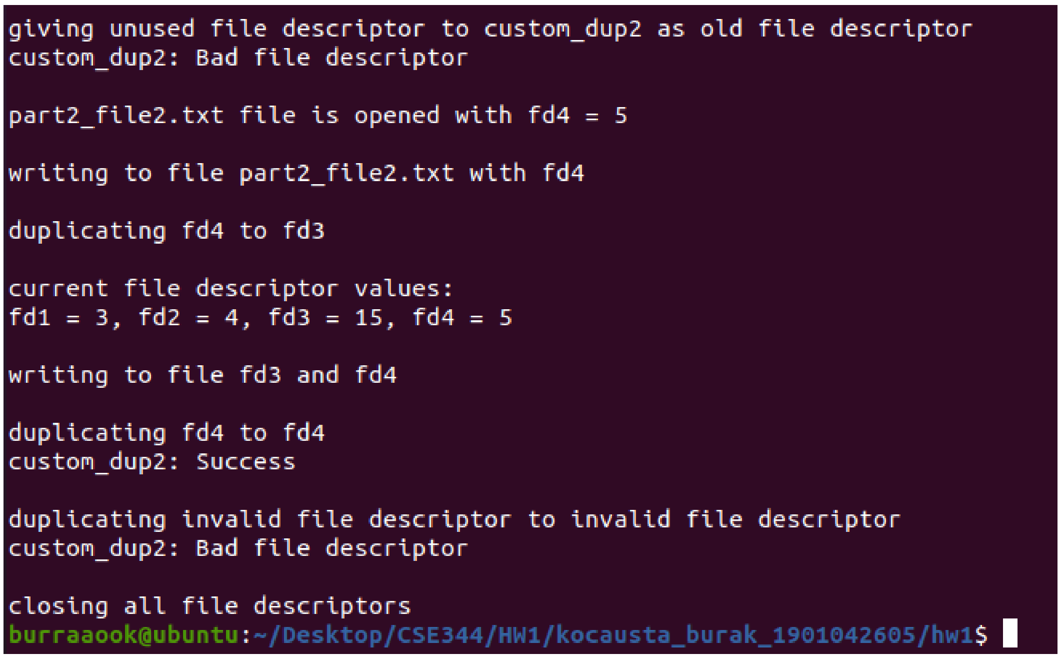
checks for passing same file descriptor case. There are 2 possibility, they could be both valid or invalid.

Results



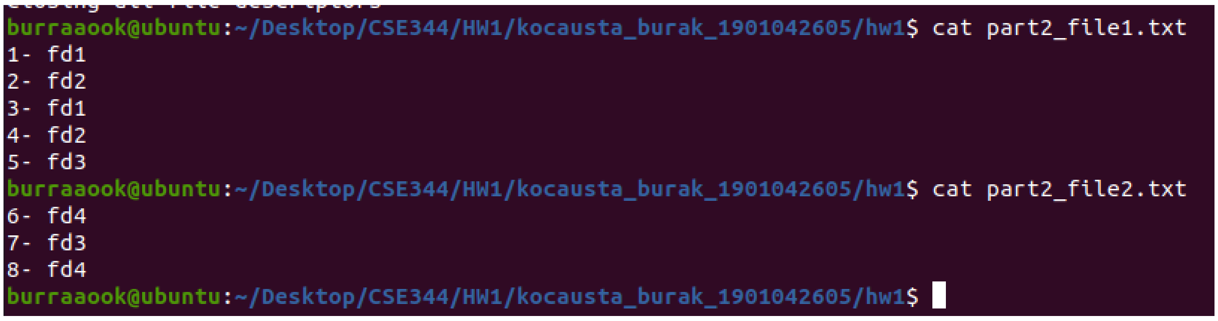
```
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ make run
./dupsTest

part2_file1.txt file is opened with fd1 = 3

fd1 is duplicated to fd2 = 4

writing to file part2_file1.txt with fd1 and fd2

fd1 is duplicated using custom_dup2(fd1, 15) to fd3 = 15

writing to file part2_file1.txt with fd1, fd2 and fd3

current file offset values:
fd1 = 35, fd2 = 35, fd3 = 35

changing fd1 offset value to 0

current file offset values:
fd1 = 0, fd2 = 0, fd3 = 0
```

first, fd1 is opened, then fd1 is redirected to fd2. After that write function called with fd1, and fd2. It can be seen that they write to same file. Then, fd1 again redirected to fd3 with dup2 this time. 3 of them called with write system call, and they write to the same file. fd1's offset is changed, and all offsets changed. So, **It is proven that they share same file offset, and open file.**



```
giving unused file descriptor to custom_dup2 as old file descriptor
custom_dup2: Bad file descriptor

part2_file2.txt file is opened with fd4 = 5

writing to file part2_file2.txt with fd4

duplicating fd4 to fd3

current file descriptor values:
fd1 = 3, fd2 = 4, fd3 = 15, fd4 = 5

writing to file fd3 and fd4

duplicating fd4 to fd4
custom_dup2: Success

duplicating invalid file descriptor to invalid file descriptor
custom_dup2: Bad file descriptor

closing all file descriptors
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$
```

unused file descriptor is given, to custom_dup2, errno is set properly. fd4 is initialized with opening new file. Then fd3 is redirected to fd4, and fd3 writes to new file after that properly. Lastly, duplication of same file descriptors are tested, and perror gives success for valid descriptors, bad file descriptor for invalid descriptors.



```
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ cat part2_file1.txt
1- fd1
2- fd2
3- fd1
4- fd2
5- fd3
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ cat part2_file2.txt
6- fd4
7- fd3
8- fd4
burraaook@ubuntu:~/Desktop/CSE344/HW1/kocausta_burak_1901042605/hw1$ 
```

fd1, fd2, fd3 writes to the file1 initially, after fd3 is changed it writes to file2 with fd4.