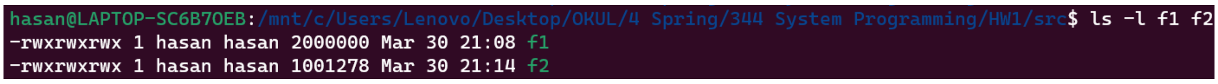
GTU Department of Computer Engineering
CSE 344 – Spring 2023
Homework #1 Report

Hasan Mutlu
1801042673

Part 1

Implementation: Firstly, the validity of the command line argument is checked. Then, the presence of optional [x] argument is checked. According to the presence of [x] argument, the file is opened with or without O_APPEND flag. After that, the character ‘x’ is written to the file one at a time. If the program is called with [x] argument, lseek(fd, 0, SEEK_END) is performed before each write. After writing is finished, file is closed.

Test Results:



```
hasan@LAPTOP-SC6B7OEB:/mnt/c/Users/Lenovo/Desktop/OKUL/4 Spring/344 System Programming/HW1/src$ ls -l f1 f2
-rwxrwxrwx 1 hasan hasan 2000000 Mar 30 21:08 f1
-rwxrwxrwx 1 hasan hasan 1001278 Mar 30 21:14 f2
```

The program is run with the example prompts in the PDF and the resulting file sizes are as above. For the f1, the size of the file is exactyle 2 million bytes whereas the size of the f2 is a little over a million.

This is because when the file is opened with O_APPEND flag, setting the offset position to the end of the file and writing to file is performed as one atomic operation. Therefore, the file offset position is guaranteed to be at the end of the file in every write operation.

However, in the second case, when the file is not opened with O_APPEND flag but instead lseek is used to set offset position to the end of the file, race condition happens between multiple processes. In this case, the lseek operations of two processes might be executed in the CPU at the same time or context switches happen between write and lseek calls. Therefore, the position offset after the lseek operation isn’t always up to date and some of the data is overwritten.

Part 2:

Implementation: dup() and dup2() functions are implemented with the names mydup() and mydup2() in the mydup.c file.

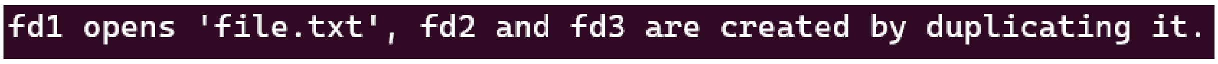
In mydup(), the new duplicate file descriptor is created using fcntl(oldfd, F_DUPFD, 0) function call and the newfd is returned directly. If fcntl returns -1, it can be checked and handled in the calling program.

In mydup2(), firstly, equality of newfd and oldfd is checked. If they are equal, validity of the oldfd is checked using fcntl(oldfd, F_GETFL) and in case of an error, errno is set to EBADF and -1 is returned. Otherwise, oldfd is returned directly. If they are not equal, firstly, newfd is closed if it is open. Then, new duplicate file descriptor is created using fcntl(oldfd, F_DUPFD, newfd) call and returned.

Part 3

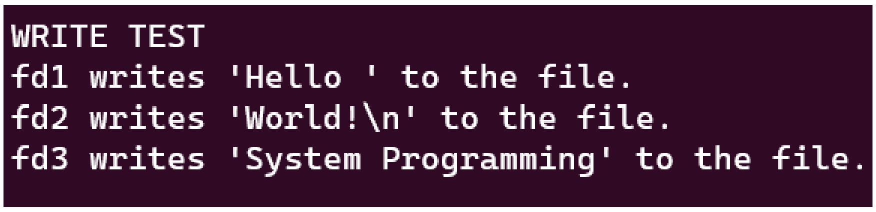
A driver program 'duptest' is implemented to check the functionality of mydup() and mydup2().

In the beginning, program creates a file descriptor named fd1, and opens a file named 'file.txt' with it. After that, two file descriptors, fd2 and fd3 are created using mydup() and mydup2().



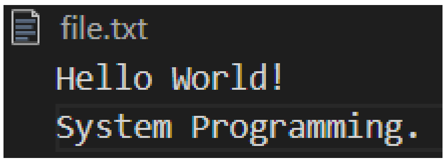
```
fd1 opens 'file.txt', fd2 and fd3 are created by duplicating it.
```

Firstly, write test starts. In write test, all three file descriptors write some text into the file.



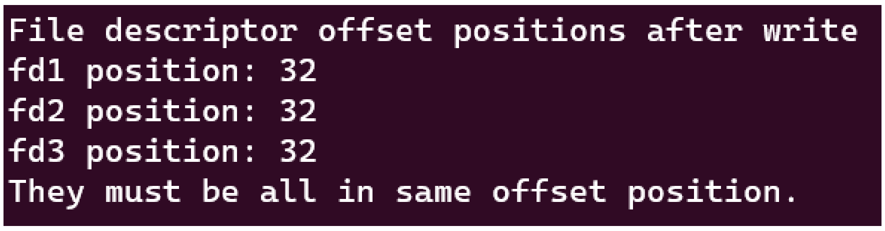
```
WRITE TEST
fd1 writes 'Hello ' to the file.
fd2 writes 'World!\n' to the file.
fd3 writes 'System Programming' to the file.
```

As expected, the file descriptors write to file as one. After the write operations, the file has the text written by all of them.



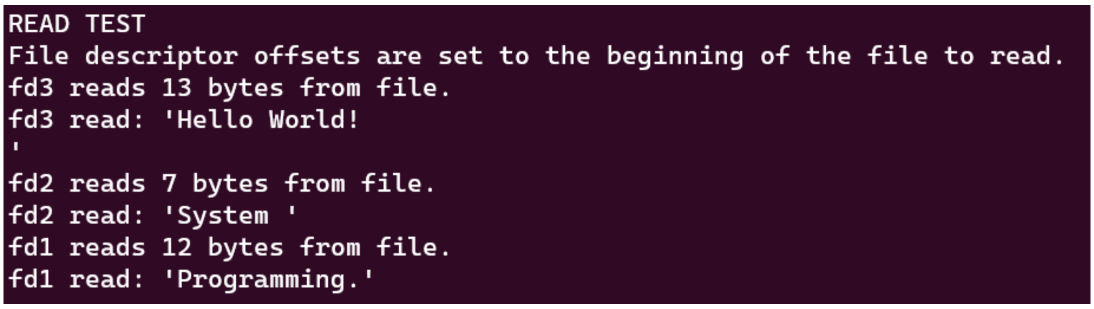
```
file.txt
Hello World!
System Programming.
```

After that, their offset positions are printed. They are all in the same offset.



```
File descriptor offset positions after write
fd1 position: 32
fd2 position: 32
fd3 position: 32
They must be all in same offset position.
```

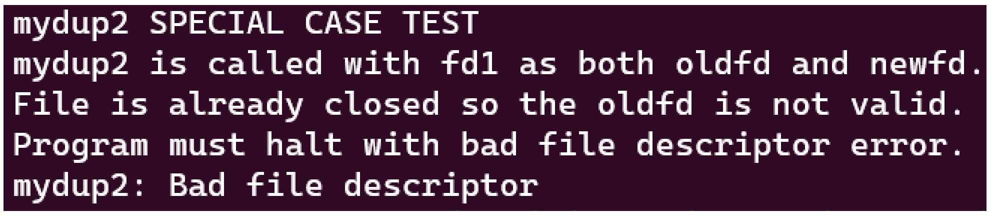
Secondly, read test is performed. All file descriptors are set to the beginning of the file to read the text in it. Then, they read from file one by one. I arranged the number of bytes they read to make them read one or two words each.



```
READ TEST
File descriptor offsets are set to the beginning of the file to read.
fd3 reads 13 bytes from file.
fd3 read: 'Hello World!
'
fd2 reads 7 bytes from file.
fd2 read: 'System '
fd1 reads 12 bytes from file.
fd1 read: 'Programming.'
```

As expected, the file descriptors read from file as one. Their offsets are equal again.

Lastly, the special case is tested. Firsly, all files are closed. Then, mydup2() is called with the already closed fd1 as both oldfd and newfd argument. In this case, the oldfd is not valid and the program is halted with bad file descriptor (EBADF) error.



```
mydup2 SPECIAL CASE TEST
mydup2 is called with fd1 as both oldfd and newfd.
File is already closed so the oldfd is not valid.
Program must halt with bad file descriptor error.
mydup2: Bad file descriptor
```

The test program ‘duptest’ is run with the regular dup() and dup2() functions as well and given the same exact outputs.