CSE 344 – System Programming

Homework #1

Yusuf Arslan

200104004112

1- appendMeMore.c

- *How did I solve it?*

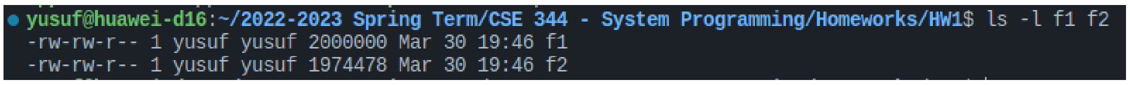
  Looking at the number of command line arguments, I decided how to append the file.

  If the 4th argument is x, I set the `use_lseek` variable to true and open the file without O_APPEND mode, otherwise I open the file in O_APPEND mode.

  Then if `use_lseek` is true, inside the for loop, by setting the offset to the end of the file, I added '#' by `write ()` function.

  Otherwise, since the file opened with O_APPEND mode, there is no need to set offset anymore, so, I just used to `write ()` function to add '#' character that represents 1 byte.

- *Difference between f1 and f2:*



```
yusuf@huawei-d16:~/2022-2023 Spring Term/CSE 344 - System Programming/Homeworks/HW1$ ls -l f1 f2
-rw-rw-r-- 1 yusuf yusuf 2000000 Mar 30 19:46 f1
-rw-rw-r-- 1 yusuf yusuf 1974478 Mar 30 19:46 f2
```

  Although the `for` loops iterate an equal number of times, there is a problem with the file expanded using `lseek` because the size is constantly changing, which can lead to a race condition bug. This occurs because we are not using synchronization techniques in conjunction with `lseek`, and we are not able to prevent data corruption methods such as 'deadlock'.

  To avoid this bug, we can use synchronization techniques like locks to ensure that only one thread or process can access the shared file at a time.
  Additionally, we can use file-locking mechanisms to prevent concurrent write access to the same file. By using these techniques, we can prevent race conditions and ensure that our program behaves predictably and correctly.

2- duplication.c

- *How did I solve it?*

  dup(int: oldfd):
  This function used for copying file descriptor (fd).
  To implement this, I used \`fcntl\` function to copy the \`oldfd\`.

  If, \`fcntl\` return –1, this mean there is an error.
  Error can be happened due to, oldfd is not a valid \`fd\`. If we encounter such a situation, the EBADF flag is set to errno so that the user can be informed.

  dup2(int: oldfd, int: newfd):
  This function used for copying \`oldfd\` to \`newfd\`. \`dup2\` requires more parameter control.

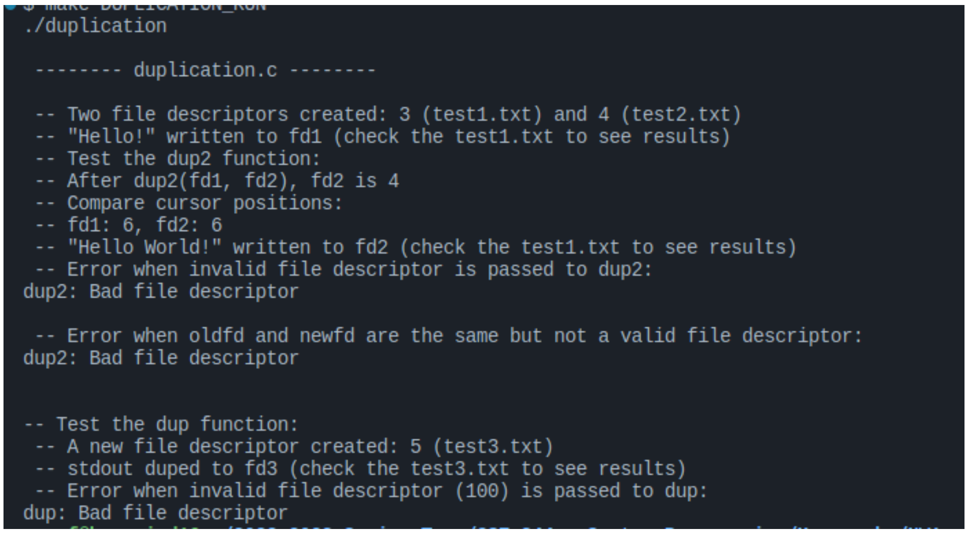
  1- \`oldfd\` can be an invalid file descriptor.
  2- \`newfd\` can already be used.

  If \`oldfd\` is an invalid *fd* so the function will return –1.
  If \`newfd\` is on use, first \`newfd\` is closed, then the \`oldfd\` copied to it.
  Finally, if the \`fcntl\` function copy the *fd* successfully, the result will be return as new file description ID.

- *Test Cases*

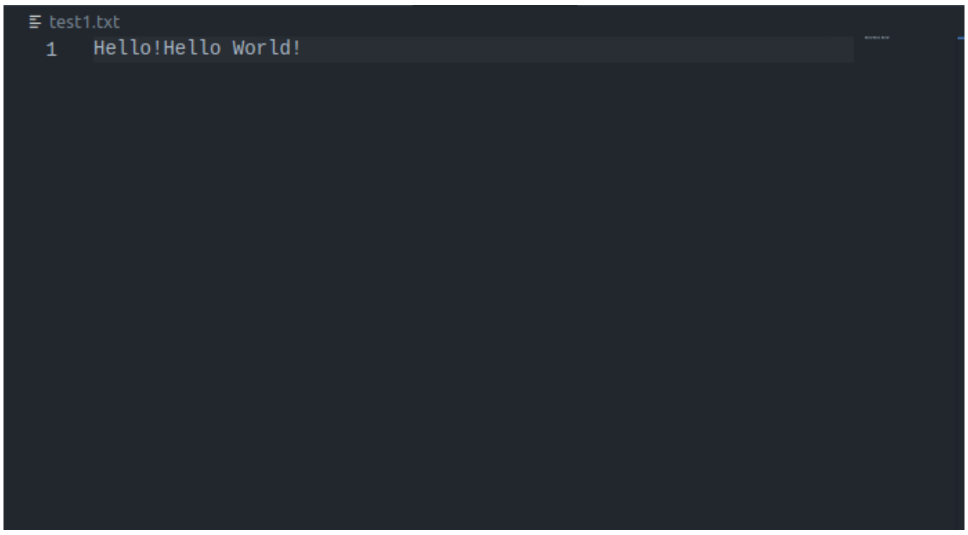
  - Create files in read and write mode.
  - Add text with \`write\` function.
  - Copying file descriptor with \`dup2\`.
  - After copying the *fd*s, check for the cursor positions. (Cursor positions are same)
  - Give invalid input as \`oldfd\`.
  - Give equal inputs for \`oldfd\` and \`newfd\`.
  - Copy the stdout.
  - Copy *fd* with \`dup\`.
  - Invalid *fd* for \`dup\` function.

Screenshot from Console:



```
./duplication

 -------- duplication.c --------

 -- Two file descriptors created: 3 (test1.txt) and 4 (test2.txt)
 -- "Hello!" written to fd1 (check the test1.txt to see results)
 -- Test the dup2 function:
 -- After dup2(fd1, fd2), fd2 is 4
 -- Compare cursor positions:
 -- fd1: 6, fd2: 6
 -- "Hello World!" written to fd2 (check the test1.txt to see results)
 -- Error when invalid file descriptor is passed to dup2:
dup2: Bad file descriptor

 -- Error when oldfd and newfd are the same but not a valid file descriptor:
dup2: Bad file descriptor


-- Test the dup function:
 -- A new file descriptor created: 5 (test3.txt)
 -- stdout duped to fd3 (check the test3.txt to see results)
 -- Error when invalid file descriptor (100) is passed to dup:
dup: Bad file descriptor
```

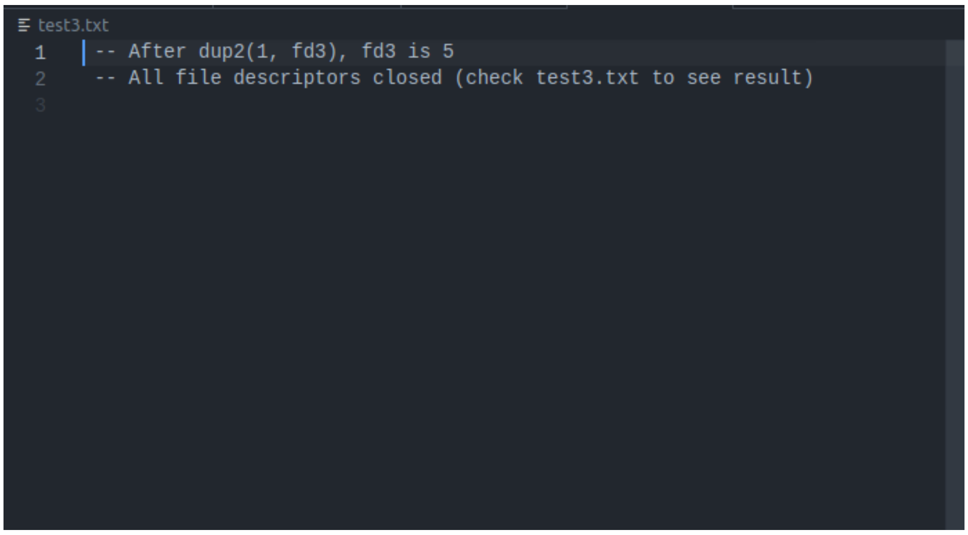
Screenshot from test1.txt:



test1.txt

```
Hello!Hello World!
```

Screenshot from test3.txt:



test3.txt

```
-- After dup2(1, fd3), fd3 is 5
-- All file descriptors closed (check test3.txt to see result)
```