Mehmet Hüseyin YILDIZ
CSE 344 System Programming
Hw-1 Report

Part 1

In this part the aim is writing to a file with append flag and with lseek and compare them by looking the results of ls -l.

Firstly, I did parameter checking. If the argc is 4, then the O_APPEND parameter is added to flags. If argc 4 and 3rd parameter is x then holding a custom flag for lseek. If the both cases wrong, then prints an error message. I used '|' operator to add O_APPEND parameter to the other parameters.

Then the given file is open with O_CREAT O_WRONLY flags. Then in a for loop I wrote 1 byte as much as given in the num_bytes parameter. If the x parameter is entered then I called lseek(fd, 0, SEEK_END) to put cursor to the end of file. Then lastly closing the file.

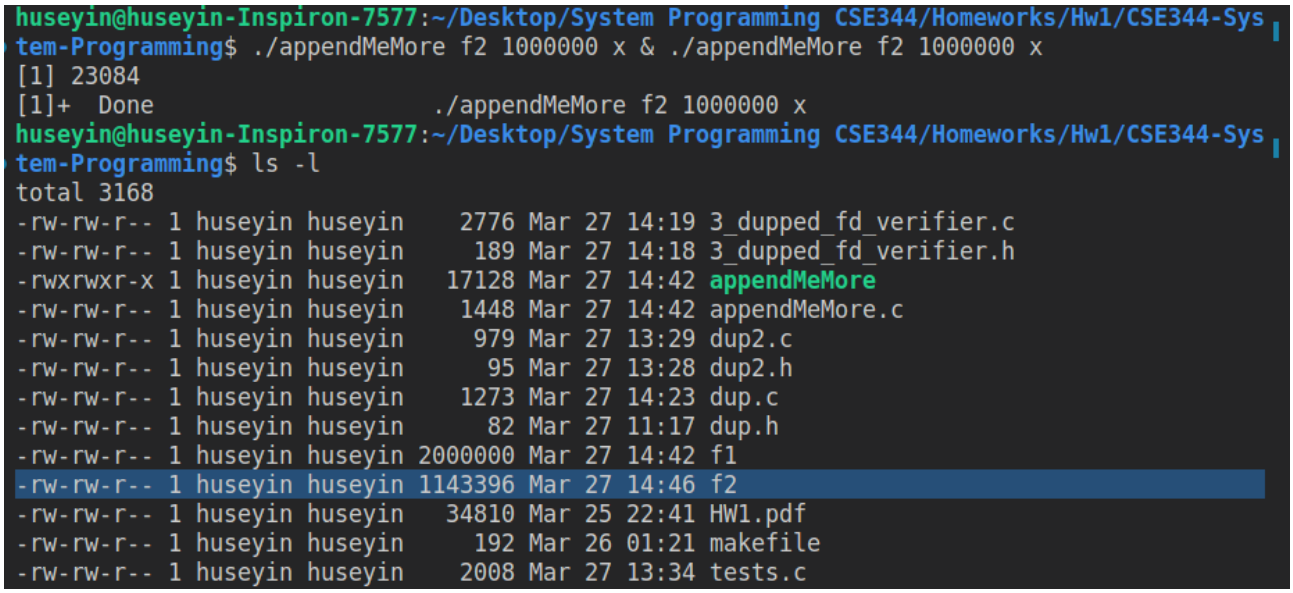
All syscalls are checked and errrors are printed to the STDERR.

Results

$ appendMeMore f1 1000000 & appendMeMore f1 1000000

```
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/Hw1/CSE344-Sys
tem-Programming$ ./appendMeMore f1 1000000 & ./appendMeMore f1 1000000
[1] 22751
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/Hw1/CSE344-Sys
tem-Programming$ ls -l
[1]+  Done                    ./appendMeMore f1 1000000
total 2048
-rw-rw-r-- 1 huseyin huseyin    2776 Mar 27 14:19 3_dupped_fd_verifier.c
-rw-rw-r-- 1 huseyin huseyin     189 Mar 27 14:18 3_dupped_fd_verifier.h
-rwxrwxr-x 1 huseyin huseyin   17128 Mar 27 14:42 appendMeMore
-rw-rw-r-- 1 huseyin huseyin    1448 Mar 27 14:42 appendMeMore.c
-rw-rw-r-- 1 huseyin huseyin     979 Mar 27 13:29 dup2.c
-rw-rw-r-- 1 huseyin huseyin      95 Mar 27 13:28 dup2.h
-rw-rw-r-- 1 huseyin huseyin    1273 Mar 27 14:23 dup.c
-rw-rw-r-- 1 huseyin huseyin      82 Mar 27 11:17 dup.h
-rw-rw-r-- 1 huseyin huseyin 2000000 Mar 27 14:42 f1
-rw-rw-r-- 1 huseyin huseyin   34810 Mar 25 22:41 HW1.pdf
-rw-rw-r-- 1 huseyin huseyin     192 Mar 26 01:21 makefile
-rw-rw-r-- 1 huseyin huseyin    2008 Mar 27 13:34 tests.c
```

$ appendMeMore f2 1000000 x & appendMeMore f2 1000000 x



As can be seen here, while f1 has the size 1000000, the f2 has less than it but more than 1000000. This is happening because when we have x parameter in the writing loop we use use lseek and write functions. When we run 2 program at the same time they both will try to lseek and write. Since we have 2 operations (lseek and write). The execution will be cutted inbetween lseek and write. The other process will execute but when we return to the first process, it will write on the what process 2 writes. So we need an atomic operation for this purpose. When we write with O_APPEND flag, the OS will write to the end of file atomically. This is why we get this result when we use lseek and write.

Part 2.1 (dup)

For this part we would implement the dup and dup2 functions.
For dup function I used fctnl commands. We can duplicate a file descriptor using F_DUPFD command.

```
// Finds the lowest numbered fd greater than 0.
newfd = fcntl(oldfd,F_DUPFD, 0);
```

The last parameter we give will be initial value of file descriptor. It will find a unallocated fd by starting 0. This is 0 because when I read manual page of dup. It says dup starts from 0 when finding a unallocated fd..

But there is another issue, we need to validate oldfd. For this purpose I used F_GETFL command as told in the problem definiton. F_GETFL command will return the flags and access modes of the given file descriptor. If there is no fd with given fd then it will return a error value that smaller than 0.

```
// Validation of old file descriptor
if(fcntl(oldfd, F_GETFL) < 0)
{
    errno = EBADF;
    return -1;
}
```

Tests
Test Case 1,5 : If olfd is invalid then we must get an error. I used a negative and a positive number.
Test Case 2,3,4 : If the oldfd is valid we must get a new fd without error. I tested STDOUT, STDIN, STDERR.

```
// Tests
int main(int argc, char const *argv[])
{
    int result;

    // Test Case 1 :Negative Invalid oldfd
    result = dup(-5);
    if(result == -1)
        perror("FD_ERROR");
    printf("Test with  -5 : %d\n", result);

    // Test Case 2 :Valid oldfd (0)
    result = dup(0);
    if(result == -1)
        perror("FD_ERROR");
    printf("Test with   0 : %d\n", result);
```

```c
    // Test Case 3 :Valid oldfd (1)
    result = dup(1);
    if(result == -1)
        perror("FD_ERROR");
    printf("Test with   1 : %d\n", result);

    // Test Case 4 :Valid oldfd (2)
    result = dup(2);
    if(result == -1)
        perror("FD_ERROR");
    printf("Test with   2 : %d\n", result);

    // Test Case 5 :Positive invalid oldfd
    result = dup(100);
    if(result == -1)
        perror("FD_ERROR");
    printf("Test with 100 : %d\n", result);


    return 0;
}
```

Outputs:

```
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/H
w1/CSE344-System-Programming$ make dup
rm -f appendMeMore dup dup2
cc dup.c -o dup
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/H
w1/CSE344-System-Programming$ ./dup
FD_ERROR: Bad file descriptor
Test with  -5 newfd: -1

Test with   0 newfd: 3

Test with   1 newfd: 4

Test with   2 newfd: 5

FD_ERROR: Bad file descriptor
Test with 100 newfd: -1
```

As can be seen here when oldfd is not valid it sets the error and returns 1 and the new file descriptors grows.

Note: In dup.c file there are tests in main function. They may used by uncommenting the main function.

Part 2.2 (dup2)

In this part I implemented dup2 function which is used for duplicating the fd to a specific fd.

For duplication of a file descriptor we can use F_GETFL command with fcntl function and we give the newfd as parameter. But here there are some cases that we should handle them.

```
// Validation of old file descriptor
if(fcntl(oldfd, F_GETFL) < 0)
{
    errno = EBADF;
    return -1;
}
```

Firstly the validation of oldfd is realized. For this purpose I used F_GETFL as I told in the dup part. The other case we should check if the oldfd and newfd is equal. If so, i just return the newfd.

```
// if the oldfd and newfd equal then just return the newfd
if(oldfd == newfd)
    return newfd;
```

Another case is if the newfd is already open, then we must close it before duplicating. I used F_GETFL command for this purpose.

```
// If the newfd is open then close it
if( fcntl(newfd,F_GETFL) >= 0)
    close(newfd);
```

Then I called the fctnl function with F_DUPFD command (parameter) and newfd.

```
result = fcntl(oldfd, F_DUPFD, newfd);
```

Another case I thought is, If somehow the result of fcntl command is not newfd and not an error then i closed it and return -1. Otherwise I returned the result of fcntl.

```
// if the result is error, then return same result
if (result < 0)
  return result;

// If the result fd is not that we want, then we close it
else if (result != newfd) {
  close(result);
  return -1;
}
else
  return result;
```

Tests

There are so much things to check so, I used assert function which placed in standart lib and returns error and exits if the argument is false. I wrote all detailed explanations with comments in the screenshots. So, if we see the success message then this means there are no any error.

```
// Test Case 1: Valid oldfd and newfd descriptor

int fd = open("file.txt", O_CREAT | O_WRONLY, 0666);
int newfd = mydup2(fd, 5);
// Verify that the return value of dup2 is equal to newfd
assert(newfd == 5);
// Verify that the file descriptors refer to the same file
assert(fcntl(fd, F_GETFL) == fcntl(newfd, F_GETFL));
close(fd);



// Test Case 2: Invalid oldfd

newfd = mydup2(100, 5);
// Verify that the return value of dup2 is -1
assert(newfd == -1);
// Verify that errno is set to EBADF
assert(errno == EBADF);



// Test Case 3: Special case where oldfd equals newfd and oldfd is valid

fd = open("file.txt", O_CREAT | O_WRONLY, 0666);
newfd = mydup2(fd, fd);
// Verify that the return value of dup2 is equal to newfd
assert(newfd == fd);
// Verify that the file descriptors refer to the same file
assert(fcntl(fd, F_GETFL) == fcntl(newfd, F_GETFL));
close(fd);
```

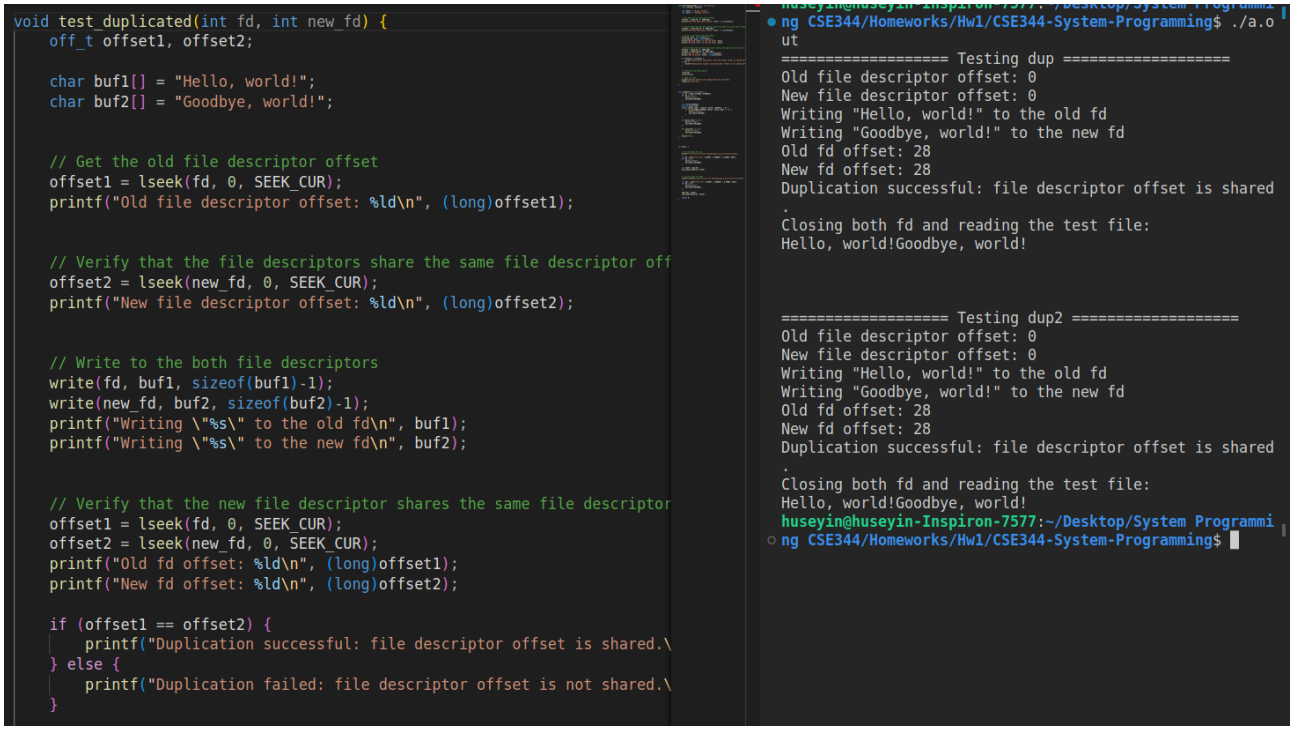
```c
// Test Case 4: Special case where oldfd equals newfd and oldfd is invalid

newfd = mydup2(100, 100);
// Verify that the return value of dup2 is -1
assert(newfd == -1);
// Verify that errno is set to EBADF
assert(errno == EBADF);



// Test Case 5: Multiple calls to dup2 with same oldfd and different newfd

fd = open("file.txt", O_CREAT | O_WRONLY, 0666);
int newfd1 = mydup2(fd, 5);
int newfd2 = mydup2(fd, 6);
// Verify that the return value of dup2 is equal to newfd
assert(newfd1 == 5 && newfd2 == 6);
// Verify that the file descriptors refer to the same file
assert(fcntl(fd, F_GETFL) == fcntl(newfd1, F_GETFL));
assert(fcntl(fd, F_GETFL) == fcntl(newfd2, F_GETFL));
// Verify that newfd1 and newfd2 are different
assert(newfd1 != newfd2);


// If there is no any error until here then the program can print this
printf("All cases tested with asserts. No any error in all cases \n");

return 0;
```

Output: As expected there are no any error. Just a success message. All cases ok.

```
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/Hw1/CSE344-System-Pro
gramming$ make dup2
rm -f appendMeMore dup dup2
cc dup2.c -o dup2
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programming CSE344/Homeworks/Hw1/CSE344-System-Pro
gramming$ ./dup2
All cases tested with asserts. No any error in all cases
```

Part 3

For this part the thing I done is that, I implemented a function that takes 2 file descriptor that firstly gets current offset of them checks if they are equal then it writes different messages to them then checks the offsets of them again if they are equal again and the offset is sum of length of 2 messages then we say that these file descriptors are duplicated. Lastly, it closes the both file descriptors. Reads all file with a new file descriptor and prints. If the message is same then we got that, these file descriptors are duplicated. Also we can check if the file status flags and flags are equal or not. But I preferred first method to be sure.



```
void test_duplicated(int fd, int new_fd) {
    off_t offset1, offset2;

    char buf1[] = "Hello, world!";
    char buf2[] = "Goodbye, world!";


    // Get the old file descriptor offset
    offset1 = lseek(fd, 0, SEEK_CUR);
    printf("Old file descriptor offset: %ld\n", (long)offset1);


    // Verify that the file descriptors share the same file descriptor off
    offset2 = lseek(new_fd, 0, SEEK_CUR);
    printf("New file descriptor offset: %ld\n", (long)offset2);


    // Write to the both file descriptors
    write(fd, buf1, sizeof(buf1)-1);
    write(new_fd, buf2, sizeof(buf2)-1);
    printf("Writing \"%s\" to the old fd\n", buf1);
    printf("Writing \"%s\" to the new fd\n", buf2);


    // Verify that the new file descriptor shares the same file descriptor
    offset1 = lseek(fd, 0, SEEK_CUR);
    offset2 = lseek(new_fd, 0, SEEK_CUR);
    printf("Old fd offset: %ld\n", (long)offset1);
    printf("New fd offset: %ld\n", (long)offset2);

    if (offset1 == offset2) {
        printf("Duplication successful: file descriptor offset is shared.\
    } else {
        printf("Duplication failed: file descriptor offset is not shared.\
    }
```

```
ng CSE344/Homeworks/Hw1/CSE344-System-Programming$ ./a.o
ut
================== Testing dup ==================
Old file descriptor offset: 0
New file descriptor offset: 0
Writing "Hello, world!" to the old fd
Writing "Goodbye, world!" to the new fd
Old fd offset: 28
New fd offset: 28
Duplication successful: file descriptor offset is shared
.
Closing both fd and reading the test file:
Hello, world!Goodbye, world!



================== Testing dup2 ==================
Old file descriptor offset: 0
New file descriptor offset: 0
Writing "Hello, world!" to the old fd
Writing "Goodbye, world!" to the new fd
Old fd offset: 28
New fd offset: 28
Duplication successful: file descriptor offset is shared
.
Closing both fd and reading the test file:
Hello, world!Goodbye, world!
huseyin@huseyin-Inspiron-7577:~/Desktop/System Programmi
ng CSE344/Homeworks/Hw1/CSE344-System-Programming$
```

For testing I used dup and dup2 functions to duplicate a file descriptor then call the test_duplicated function in the main function.

```c
int main() {


    // fd verifying for dup
    printf("=================== Testing dup ===================\n");

    int fd = open("file.txt", O_CREAT | O_WRONLY | O_TRUNC, 0666);
    if (fd < 0) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    int newfd = dup(fd);
    test_duplicated(fd, newfd);



    // fd verifying for dup2
    printf("\n\n\n=================== Testing dup2 ===================\n")

    int fd1 = open("file.txt", O_CREAT | O_WRONLY | O_TRUNC, 0666);
    if (fd < 0) {
        perror("open");
        exit(EXIT_FAILURE);
    }

    dup2(fd1, newfd);
    test_duplicated(fd, newfd);
```

That’s all I done for this homework. I tried to do best. I hope there is no any missing or wrong part.