GTU CSE-344

HW1

Md Sarwar Hossain

Id: 161044121

**Question 1:**

The program is a simple C program that takes in command line arguments and appends 'a' characters to a file. The program is named appendMeMore and it requires a filename and the number of bytes to be appended. It also accepts an optional argument 'x' which tells the program to call lseek() on every iteration of the loop. The program is designed to handle errors such as invalid arguments or failed file opening.

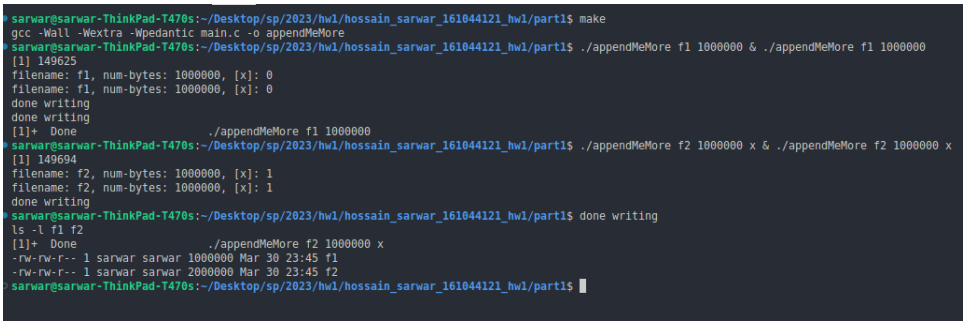
The program first checks if the number of command line arguments is valid. If not, it calls the usage() function which prints the correct format of the command line arguments. If the number of arguments is correct, the program proceeds to assign values to the filename, num_bytes, and isX variables based on the command line arguments.

The program then proceeds to open the file. If the file exists and the 'x' argument is provided, the program uses the O_APPEND flag to ensure that the characters are appended to the end of the file. If the file already exists but 'x' is not provided, the program opens the file with the O_WRONLY flag. If the file does not exist, the program uses the O_CREAT flag to create a new file with read and write permission for all users. Before creating, O_CREAT and O_EXCL is used together to make sure there is file or not. If the file exists, then it proceeds to writing.

After opening the file, the program enters a loop that writes 'a' characters to the file. If the 'x' argument is provided, the program calls lseek() on every iteration of the loop to ensure that the characters are appended to the end of the file. The

program uses the write() function to write the characters to the file. The program also handles errors such as write failure and interruption by signals during the write operation. This allowes to write given amount of byte unless there is a major issue.

Once the loop has completed, the program checks if the number of bytes written is equal to the number of bytes specified in the command line arguments. If it is, the program prints "done writing" to standard output.




Since The First two process was just appending to the same file from the beginning. It was rewriting in the same place. So f1 size is same as the num_bytes provided.

But for the second file, there is a lseek from the end. So both the process writes to the same file from the end postion. So the file size becomes, 2 time of num_bytes.

**Question 2:**

There are two function, dup and dup2. These functions are commonly used in operating systems to create new file descriptors that refer to the same underlying file or device as an existing file descriptor.

The dup function takes a file descriptor oldfd as an argument and returns a new file descriptor that refers to the same file or device as oldfd. To achieve this, dup internally uses the fcntl function with the F_DUPFD flag set to zero, which creates a new file descriptor that is the lowest-numbered available file descriptor greater than or equal to zero.

The dup2 function is similar to dup, but it allows the caller to specify the desired file descriptor number (newfd) for the duplicated file descriptor. If newfd is already open, dup2 closes it first. If oldfd and newfd are equal, dup2 checks if oldfd is a valid file descriptor by calling fcntl with the F_GETFL flag. If oldfd is not valid, dup2 returns an error with the EBADF error code.

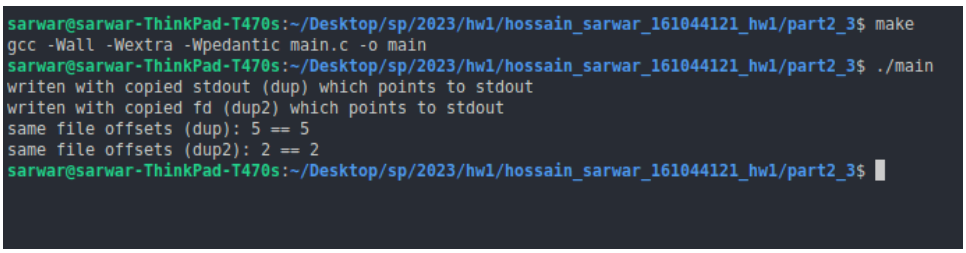
If oldfd and newfd are different, dup2 internally uses fcntl with the F_DUPFD flag to create a new file descriptor that has the same underlying file or device as oldfd. dup2 then uses close to close newfd if it was already open, and assigns the new file descriptor to newfd.

**Question 3:**

verify_dup() and verify_dup2(), is used to verify if the file offset of two file descriptors are the same after duplicating them using the dup() and dup2() system calls.

The verify_dup() function opens a file named "file.txt" for reading and writing, writes the string "hello" to the file, duplicates the file descriptor using dup(), and then checks if the file offsets of the two descriptors are the same using the lseek() function. If the offsets are different, the function prints an error message and returns. Otherwise, it prints a message indicating that the offsets are the same.

The verify_dup2() function is similar, but instead of using dup(), it uses dup2() to duplicate the file descriptor with a specific file descriptor number (I used 5). It also sets the offset of the original file descriptor to 2 before duplicating it, to ensure that the function is testing the offset of the duplicated file descriptor rather than the original.




for question 2 & 3