Name: Min Sithu Maung
UIN: 533002483

Q1

Files: q1.cpp, openr.s , testfile.txt

How to Run

g++ -o q1 q1.cpp openr.s
./q1 /path/to/the/testfile.txt

Output

maung@instance-1:~/csce_313_homework$ ./q1
/home/maung/csce_313_homework/testfle.txt
Sample text from the some textfile

Q2

Files: q2.cpp

Output

maung@instance-1:~/csce_313_homework/q2$ ./q2
Average time per function call: 3.62674 nanoseconds
Average time per system call: 839.306 nanoseconds
Ratio (system call / function call): 231.421

Response

The Ratio is approximately 231 times.
I used high_resolution_clock::now() to get the time before and after the system calls and function calls. And I get the duration by getting the difference between them. I found their averages and also displayed their ratio.

Q3

Files: q3.cpp and testfile.txt

Output

maung@instance-1:~/csce_313_homework$ ./q3
Verifying if file is available with ls:
ls: cannot access 'testfile.txt': No such file or directory
Reading the file by the child process: Sample text from the some textfile

Response

When the child continues, it should still be able to read the file even though the file is (seemingly) deleted and is no longer visible via ls. Briefly explain what might be happening:

The parent process deletes the file using unlink(), which removes the directory entry (name) of the file and decrements the inode's reference count. However, because the child process still has an open file descriptor pointing to that inode, the inode's reference count hasn't dropped to zero. Thus, the actual data of the file is still on the disk and accessible through the file descriptor. So, even though the file name is no longer visible with ls , the child process can still read the file's contents using the inherited file descriptor.

Q4

Files: q4.cpp

Output

```
maung@instance-1:~/csce_313_homework$ ./q4
Enter the file path: /home/maung/csce_313_homework/testfle.txt
File Type: Regular File
Owner Permissions: rw-
```

Response

The importance of inodes in the Unix File System is that it contains information about the metadata of the files such as file ownership, access mode (read, write, execute permissions) and file type.

Q5

Files: q5.cpp

Output

```
maung@instance-1:~/csce_313_homework$ g++ -o q5 q5.cpp
maung@instance-1:~/csce_313_homework$ ./q5
Maximum number of open files reached: 1048569
```

Response

If the file descriptor returns an error (-1), I check the errno with another if statement and if errno equals to EMFILE, that means the maximum number of files allowed to open has reached and I break the loop.

The maximum number of files allowed to open on my GCP Ubuntu VM is around 1048569.