Lianming Wu

CSC33200 – Donald Douglas Gordon

Assignment2 Multithreading

In this assignment, I created 3 threads each with access to the main data file at the same time to finish the task of merge new data 1,2 and 3 with the data in the main file and write the sorted data back. Because those three threads are having access to the main file at the same time, a critical region is created. I used mutex to solve the problem by locking the file whenever a new thread accessing the main file, so no other threads could enter the critical region before the prior thread finished merge data.

I used Python's built-in library 'threading' to do this assignment. The reason I chose to use Python was that first, Python have the 'with' keyword, which simplified the process of file manipulating, I no longer need to worry about memory leaking because forgetting to close the files. Second, the built-in library 'threading' have a lock(mutex) built in it, making the process of creating lock very easy. Lastly the list data structure in python is very convenient to use as it dynamic. I could just insert the data without worrying about the size of the array.

Compile instruction:

Install Python3 and use the command 'python3 OS_assignment2.py' in the command prompt.

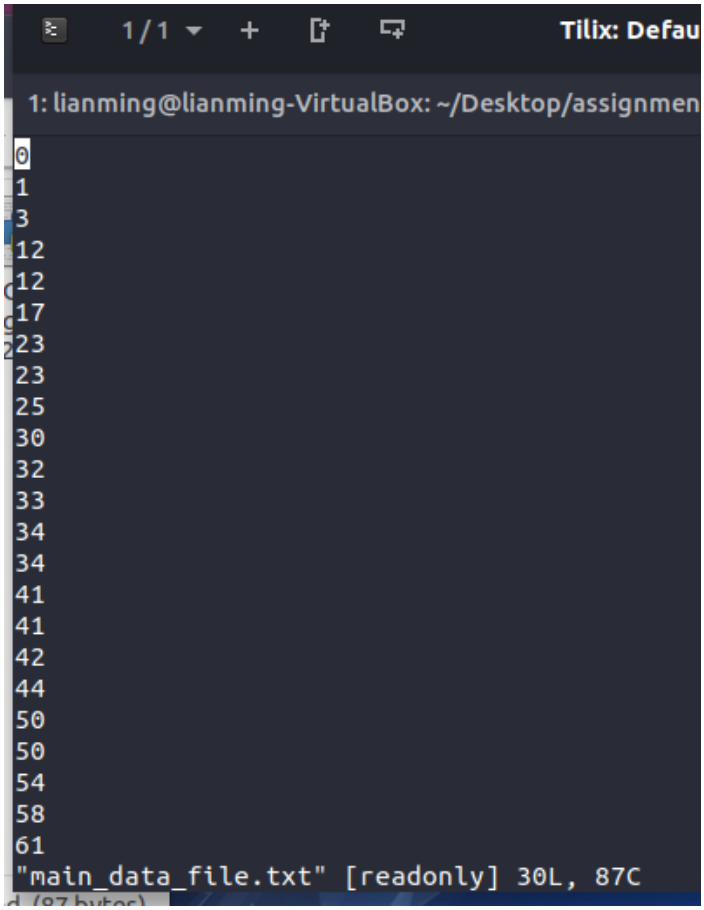
Sample Data without using the mutex:

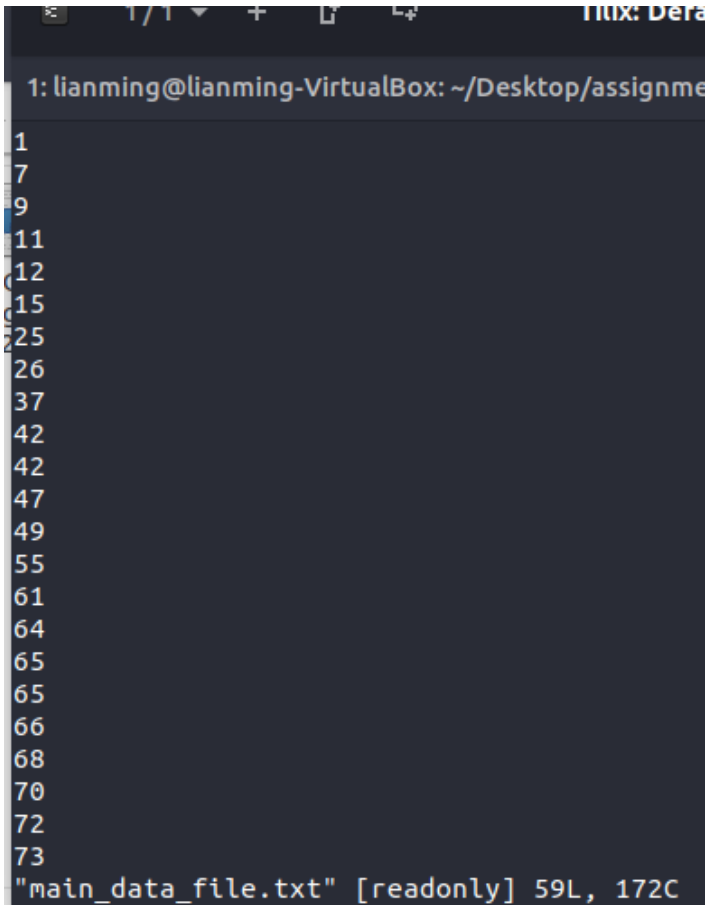
```
# Merge Functions
def merge_files(filename,mutex):
# Lock when entering the critical region
    #mutex.acquire()
    print(threading.current_thread(), "now merging files")
    merge = []
    try:
        # Open the main_data file and the new data file
        with open('main_data_file.txt','r') as mainfile, open(filename,'r') as infile:
            #Because we have \n in our data files, we get rid of that by split the inp
            #This is possiblle as we only have int\n in each line, so the first elemen
            for line in infile:
                merge.append(line.split()[0])

            for line in mainfile:
                merge.append(line.split()[0])
            #Sort the result in ascending order
            merge.sort(key=int)
            #Write the result back to our main_file
            with open('main_data_file.txt','w') as outfile:
                for elt in merge:
                    outfile.write(elt)
                    outfile.write("\n")
    except:
        print("Error occurs when merge data!")

    # The finally keyword release the lock even if exceptions are thrown, make sure we
    finally:
        print(filename,"successfully merged!")
        #mutex.release()
```

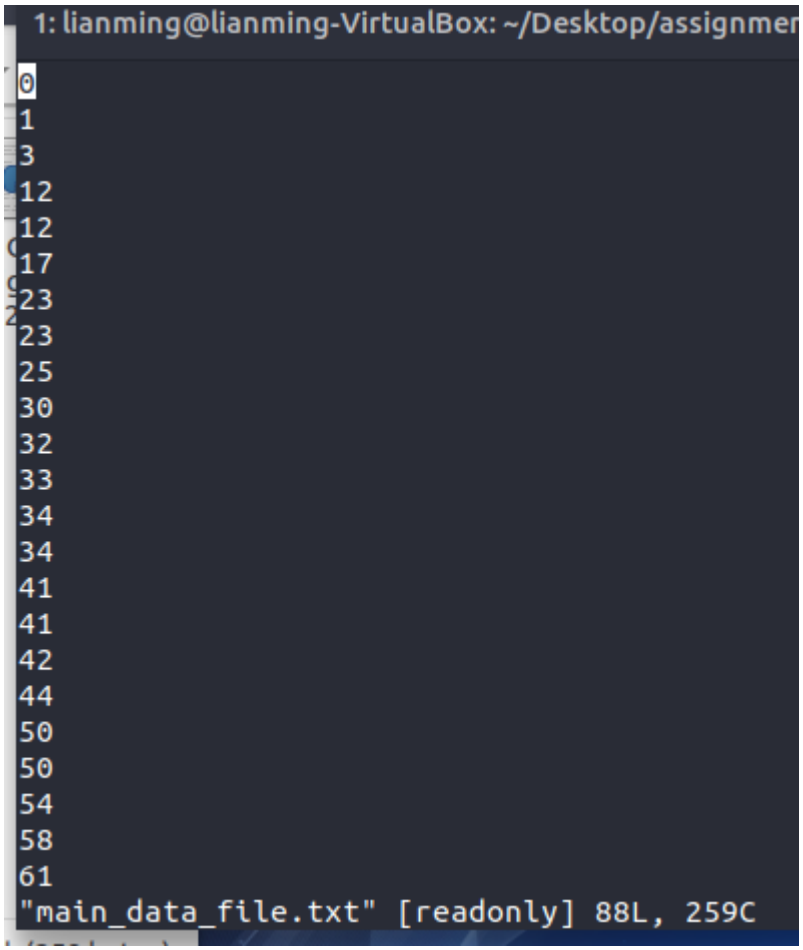
```
lianming@lianming-VirtualBox:~/Desktop/assignment2$ python3 OS_assignment2.py
<Thread(Thread-1, started 139933548283648)> now merging files
<Thread(Thread-2, started 139933539890944)> now merging files
new_data_2.txt successfully merged!
<Thread(Thread-3, started 139933457053440)> now merging files
new_data_1.txt successfully merged!
new_data_3.txt successfully merged!
```

Other threads entering the critical region before prior thread finish
Reproduce race conditions around 90% of the time

```
1
1
2
3
3
4
5
7
7
9
9
11
12
15
17
18
22
23
25
26
26
26
27
"main_data_file.txt" [readonly] 90L, 259C
```


1 / 1
Tilix: Defau
1: lianming@lianming-VirtualBox: ~/Desktop/assignmen
0
1
3
12
12
17
23
23
25
30
32
33
34
34
41
41
42
44
50
50
54
58
61
"main_data_file.txt" [readonly] 30L, 87C


1: lianming@lianming-VirtualBox: ~/Desktop/assignme
1
7
9
11
12
15
25
26
37
42
42
47
49
55
61
64
65
65
66
68
70
72
73
"main_data_file.txt" [readonly] 59L, 172C


1: lianming@lianming-VirtualBox: ~/Desktop/assignmer
0
1
3
12
12
17
23
23
25
30
32
33
34
34
41
41
42
44
50
50
54
58
61
"main_data_file.txt" [readonly] 88L, 259C