**CSE3033 PROJECT3**

**TAHA YUSUF KÖMÜR: 150114064**

**FIRAT ÇETİNER: 150115047**

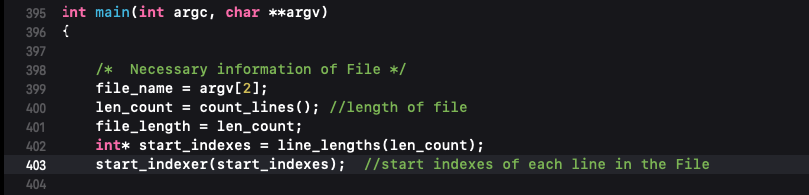
**Project Definition:** Aim of the project is, coding a multi threaded file updater program. Threads are, small work pieces which can be handled by different CPU’s concurrently. Due to concurrent work, we need to synchronize them.   
To synchronizing threads, there are several choices due to the problem.   
  
  
  


**Our Approach:** Our solution is using a linked list, and making necessary jobs via threads on it. For shared memories, mutex locks are so common and safe, so we used mutexes in our projects.

**Algorithm definition:**

1 - We get necessary information of FILE as:

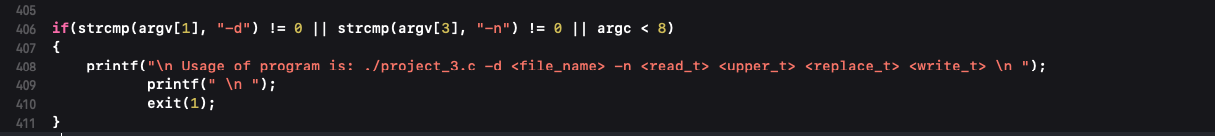
* How many line it has
* Start indexes of each line



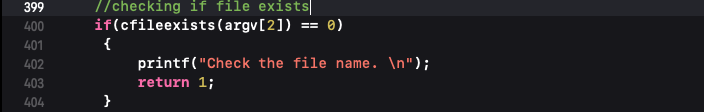
To keep report simple, we don’t ss simple functions.

2 - We handle the command line arguments

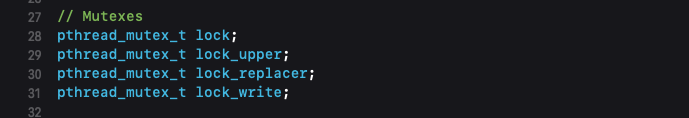
Usage of the program is: ./program -d file.txt -n <read#> <upper#> <replace#> <write#>



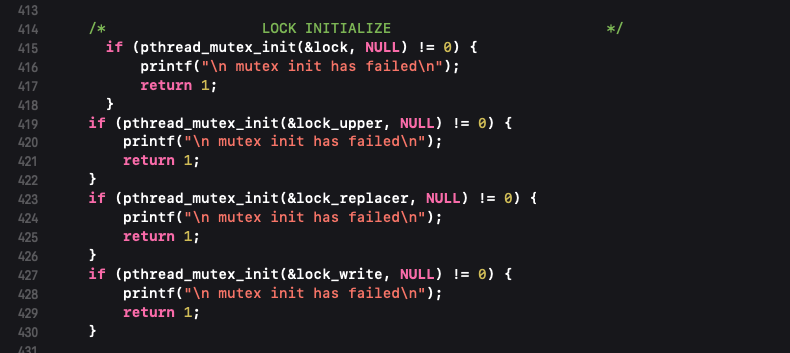
File exists check

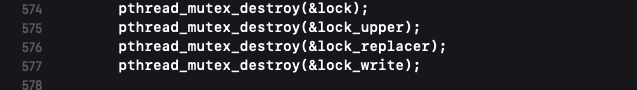


3 - We defined 4 global mutex, and initialized them inside main, and destroyed

Globals:  


Initialize:



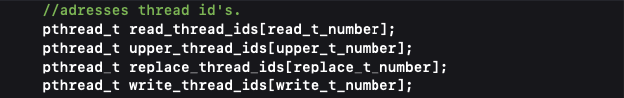
Destroy:

4 – We are taking our thread input with atoi(string to integer converter) and check them for negative values ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**5- THREAD CREATION INSIDE MAIN**

Creating thread adresses count of given amount



Arguments for threads:

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Creates&Joins

ekran görüntüsü, metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**END OF MAIN**

**READ THREAD**

* The logic here is:
* Due to threads are already assigned to their job since they created, we only need to be carefull about the inserting job. Than we use mutex to keep insert mechanism safe.
* Interesting point, to handle any number of threads can work concurrently, critical section is too short, and when thread got the work, another threads can take its own job.
* Also, to handle odd, prime, even numbers of threads, we have some
* Remainder system here, if there are problem of dividing the work between threads, the last threads owns the problem and fix it.
* We use fseek() to manipulate file pointer to the point we want.
* We use strcpy than direct copy, because it is safer.

All threads exits after their work is done.

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**UPPER THREAD**

* No work assigned for upper thread.
* At start point, if there is no work, they wait inside the while loop while(readed\_count<1)
* Since there is work, they all out of it, and the first who takes the key, gets inside the critical section.
* Inside critical section, we use find\_not\_uppered() function to find node, readed but not touched by upper threads.
* Since we find it, we check it for null to see if there is any problem.
* If there is no problem, upper thread ups the content of it.
* turn uppered -> true
* increment uppered\_count;
* If the node already replaced by replace thread we increment writable to signal write threads.
* Interesting point here, we have another lock which called memory\_lock to keep node safe for the other type of threads to come.
* First lock is for same type threads
* Second lock is for other type of threads

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**REPLACE THREAD**

* No work assigned for upper thread.
* At start point, if there is no work, they wait inside the while loop while(readed\_count<1)
* Since there is work, they all out of it, and the first who takes the key, gets inside the critical section.
* Inside critical section, we use find\_not\_replaced() function to find node, readed but not touched by replace threads.
* Since we find it, we check it for null to see if there is any problem.
* If there is no problem, replace thread converts the content of it.
* turn replaced -> true
* increment replaced\_count;
* If the node already uppered by replace thread we increment writable to signal write threads.
* Interesting point here, we have another lock which called memory\_lock to keep node safe for the other type of threads to come.
* First lock is for same type threads
* Second lock is for other type of threads

ekran görüntüsü, metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**WRITE THREAD**

* No work assigned for write thread.
* At start point, if there is no work, they wait inside the while loop while(writable\_count<1)
* Since there is work, they all out of it, and the first who takes the key, gets inside the critical section.
* Inside critical section, we use indexOf() function to find node randomly with iterator and check if its readed, uppered and replaced.
* Since it is, we check it for null to see if there is any problem.
* If there is no problem, write threads writes the content of it to the file.
* turn writed -> true
* increment wrote\_count;
* Interesting point here, we have another lock which called memory\_lock to keep node safe for the other type of threads to come.
* First lock is for same type threads
* Second lock is for other type of threads

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Interesting Functions**

1. **file\_updater()**

**This function is not creates a new txt file, by power of using fseek(), its definetely overwrites to file.**

ekran görüntüsü, metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

1. the finders:

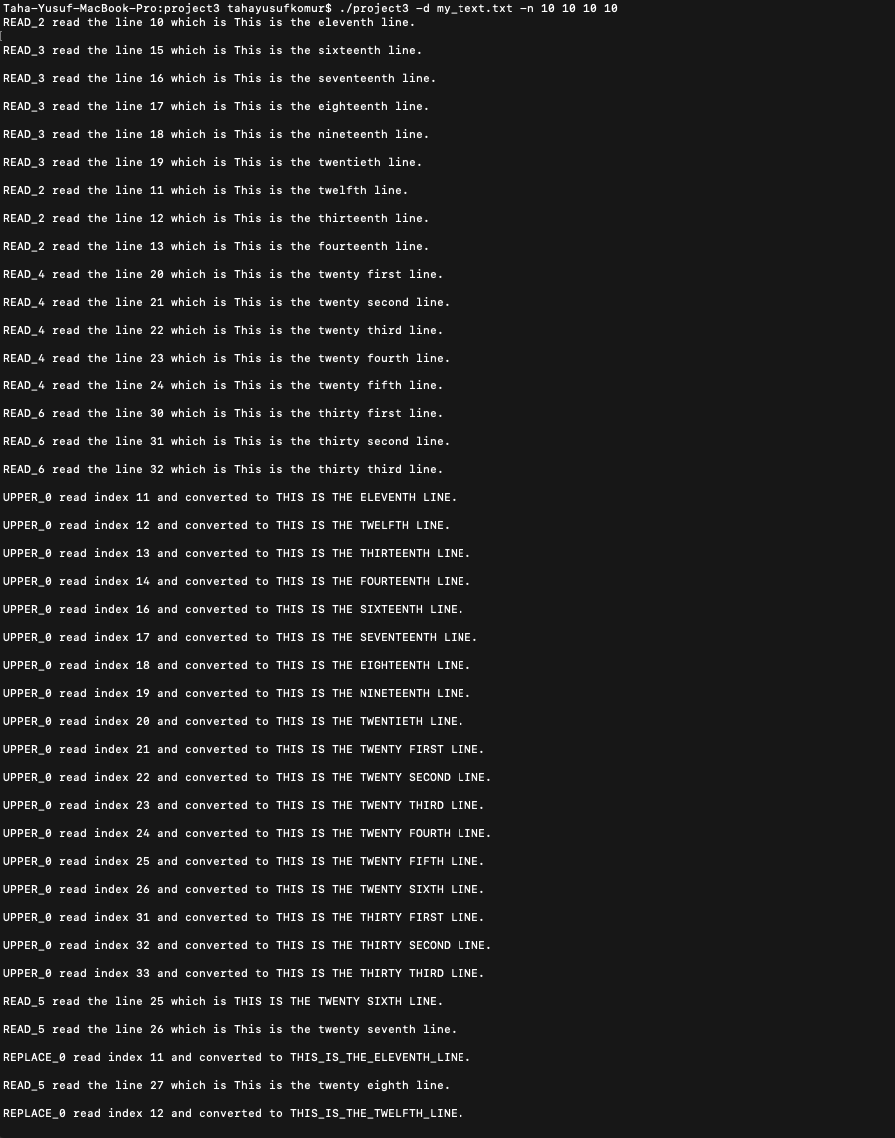
**Those functions all use the same way, use indexOf function, and iterate all the lists and depends on their types, they return the nodes.**

ekran görüntüsü içeren bir resim

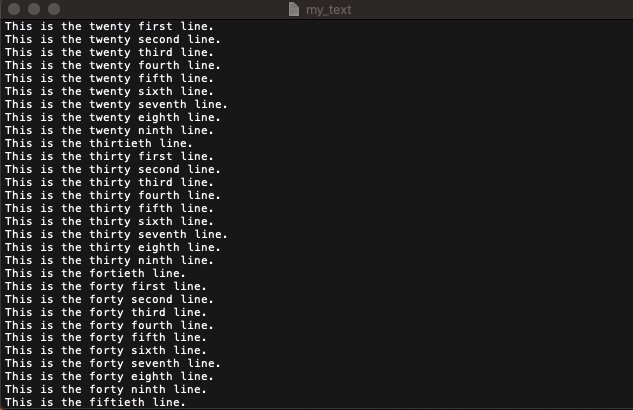
Açıklama otomatik olarak oluşturuldu

**Runtime ScreenShoots**

1. Taha-Yusuf-MacBook-Pro:project3 tahayusufkomur$ ./project3 -d my\_text.txt -n 10 10 10 10



**Before:**



**After:**

ekran görüntüsü içeren bir resim

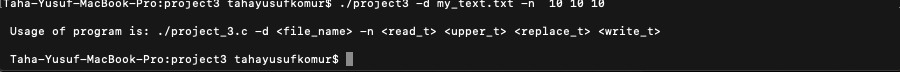
Açıklama otomatik olarak oluşturuldu

1. Taha-Yusuf-MacBook-Pro:project3 tahayusufkomur$ ./project3 -d my\_tex.txt -n 10 10 10 10 (file existing check)

cihaz içeren bir resim

Açıklama otomatik olarak oluşturuldu

1. Taha-Yusuf-MacBook-Pro:project3 tahayusufkomur$ ./project3 -d my\_text.txt -n 10 10 10 (argument check)



**Note:** Not all cases the program works %100, for some number of threads, its having issues. But for most of the cases its work fine and it handles all necessary errors. All threads are working concurrently, for all threads, race conditions between their type is fair, also race condition for other types is fair.