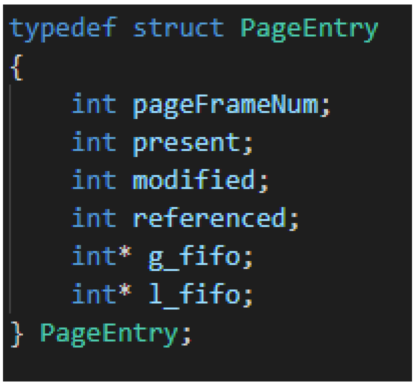
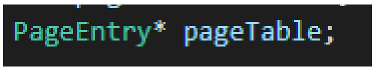
151044010 – TARIK KILIÇ

Page Table Tasarımı

Page Table is an array of type Page entry. It has Frame Number, reference, modified and fifo(2 variables). For global and local. You can see the following structer in line 21.

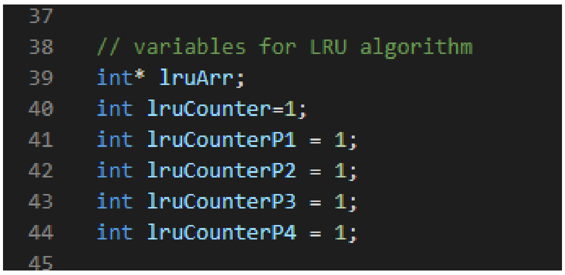


```
typedef struct PageEntry
{
    int pageFrameNum;
    int present;
    int modified;
    int referenced;
    int* g_fifo;
    int* l_fifo;
} PageEntry;
```



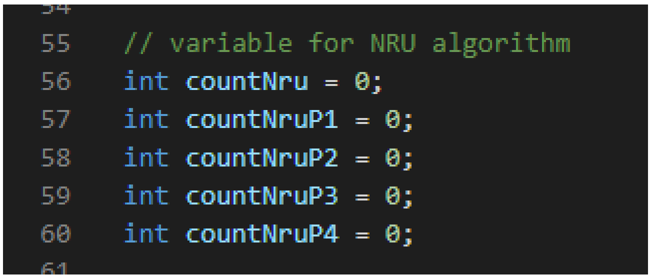
```
PageEntry* pageTable;
```

I kept normal c variables for NRU and LRU. I kept separate variables for global and local. I had to keep it different for each program. The following are the variables (with line numbers):



```
37
38   // variables for LRU algorithm
39   int* lruArr;
40   int lruCounter=1;
41   int lruCounterP1 = 1;
42   int lruCounterP2 = 1;
43   int lruCounterP3 = 1;
44   int lruCounterP4 = 1;
45
```

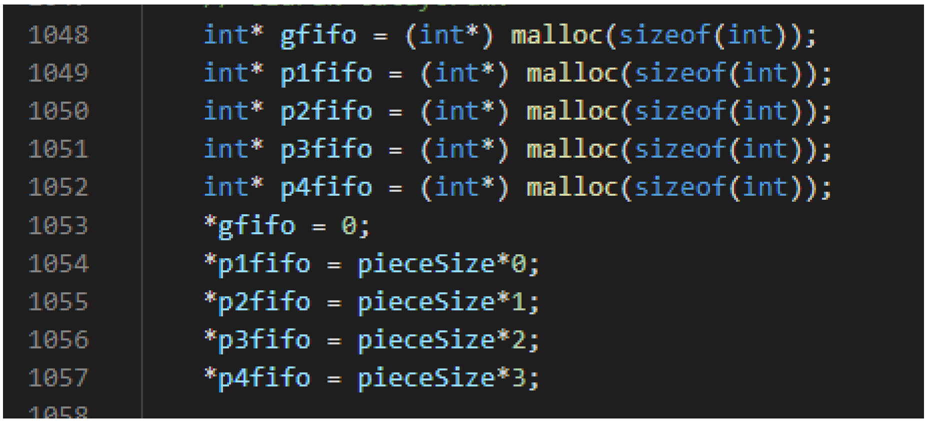
I have defined an integer array for LRU. I kept 4 counters for each program. Each shows the last time. When I replace it, I increase it and put it in the integer array. This is how I choose the oldest frame.



```
54
55   // variable for NRU algorithm
56   int countNru = 0;
57   int countNruP1 = 0;
58   int countNruP2 = 0;
59   int countNruP3 = 0;
60   int countNruP4 = 0;
61
```

I use these variables to reset the referenced bits. When it is 20 times, I reset it all if it is global and 4 if it is local.

The variables of l_fifo and g_fifo hold the memory frame first entered. I kept it in a pointer type because it should all be the same. On the other hand, I defined 4 different pointers and made reference assignments. Sample code below:



```
    int* gfifo = (int*) malloc(sizeof(int));
    int* p1fifo = (int*) malloc(sizeof(int));
    int* p2fifo = (int*) malloc(sizeof(int));
    int* p3fifo = (int*) malloc(sizeof(int));
    int* p4fifo = (int*) malloc(sizeof(int));
    *gfifo = 0;
    *p1fifo = pieceSize*0;
    *p2fifo = pieceSize*1;
    *p3fifo = pieceSize*2;
    *p4fifo = pieceSize*3;
```

The pieceSize here is the starting frame of that program in the page table.

This is my Page Table design in general.

Set Function (639. Line)

In the following order:

Print Page

I set a condition for printing the page table in the 652-655 line spacing. I keep pressing the counter at the intervals according to the inputa that comes.

Clear Bit

When using NRU algorithm in 659-661 line spacing, I reset the reference bits if get set request is made 20 times.

In case of Page Fault

I write the first value directly to disk.

I choose a memory frame according to the selected algorithm and alloc policy between 678-715 lines.

Then, in line 718, I find the page entry index with that memory frame.

In line 727, I am writing memory frame to virtual memory. Then I mark the page frame number in that page entry as -1. Shows -1 disk in the program. Then I take the frame from the disk with the given index and put it in the memory frame. By the way, I increase the statistics variables according to the sort type. Then I update the page table (761).

In case of Page Hit

I find the direct memory address from the page table and write the value to memory. I make reference bit 1. I am increasing LRU variables between 775-795. I am increasing the statistics variables between 797-811.

The Get function works in a very similar way.

I write and read the Virtual Memory file as binary. Because it is easier to reach with index. I always calculate over 4 bytes.

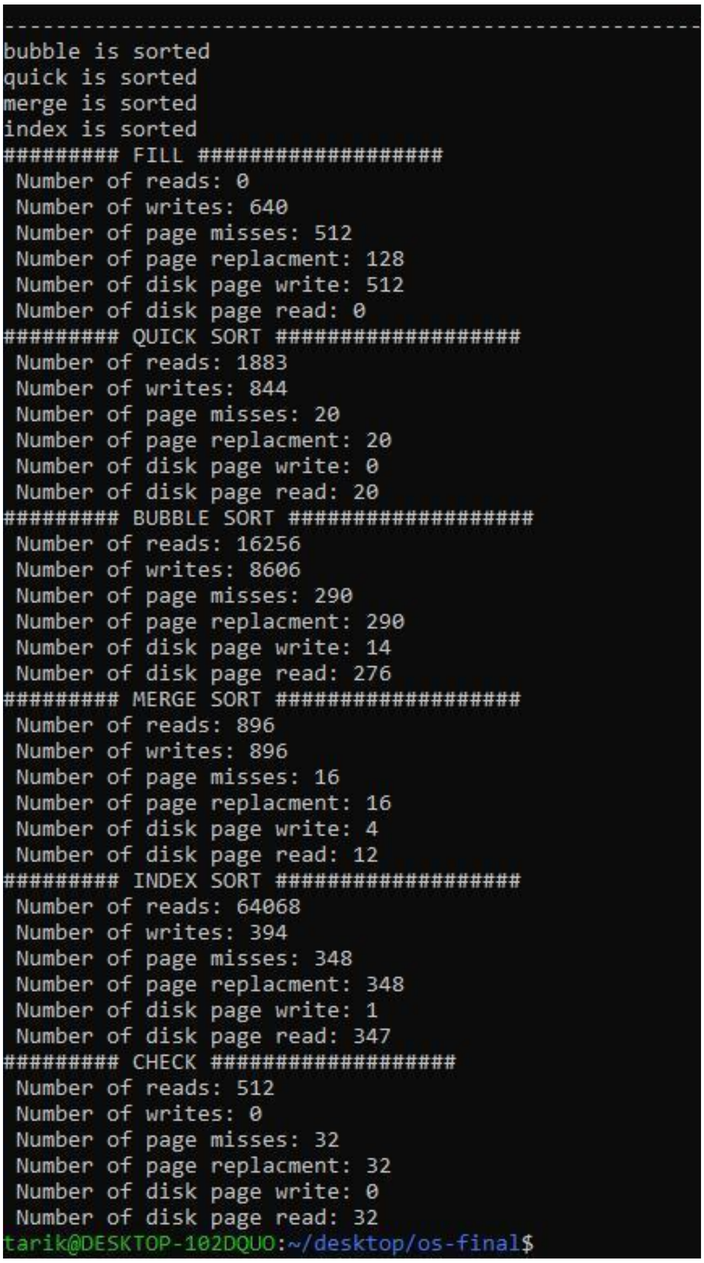
The thread function starts at line 1084. I called the sort functions according to the thread number.

While writing index sort, I wrote the pseudocode in the article I gave below.(May be same variable name is used by other student)

https://research.ijcaonline.org/volume78/number14/pxc3891325.pdf

I used the POSIX library for thread. While writing to disk and memory, I used mutex to avoid race condition. I locked where I used get and set in sort codes.

Example screenshot:



```
------------------------------------------------------
bubble is sorted
quick is sorted
merge is sorted
index is sorted
######### FILL ###################
 Number of reads: 0
 Number of writes: 640
 Number of page misses: 512
 Number of page replacment: 128
 Number of disk page write: 512
 Number of disk page read: 0
######### QUICK SORT ###################
 Number of reads: 1883
 Number of writes: 844
 Number of page misses: 20
 Number of page replacment: 20
 Number of disk page write: 0
 Number of disk page read: 20
######### BUBBLE SORT ###################
 Number of reads: 16256
 Number of writes: 8606
 Number of page misses: 290
 Number of page replacment: 290
 Number of disk page write: 14
 Number of disk page read: 276
######### MERGE SORT ###################
 Number of reads: 896
 Number of writes: 896
 Number of page misses: 16
 Number of page replacment: 16
 Number of disk page write: 4
 Number of disk page read: 12
######### INDEX SORT ###################
 Number of reads: 64068
 Number of writes: 394
 Number of page misses: 348
 Number of page replacment: 348
 Number of disk page write: 1
 Number of disk page read: 347
######### CHECK ###################
 Number of reads: 512
 Number of writes: 0
 Number of page misses: 32
 Number of page replacment: 32
 Number of disk page write: 0
 Number of disk page read: 32
tarik@DESKTOP-102DQUO:~/desktop/os-final$
```

151044010 – TARIK KILIÇ