Gebze Technical University

Department Of Computer Engineering

CSE 312 /CSE 504 Spring 2022

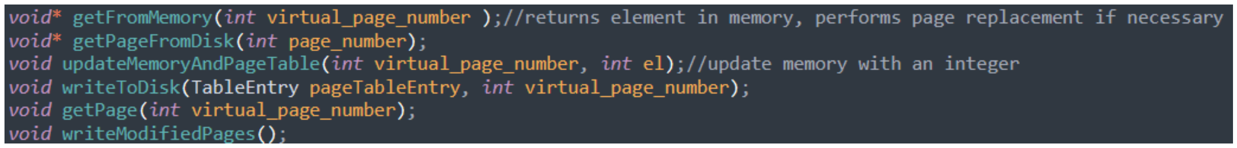
Operating Systems

Homework #02

Đuro Radusinović, Student no. 171044095

First thing implemented were sorting algorithms:
- bubble sort
- insertion sort
- quick sort

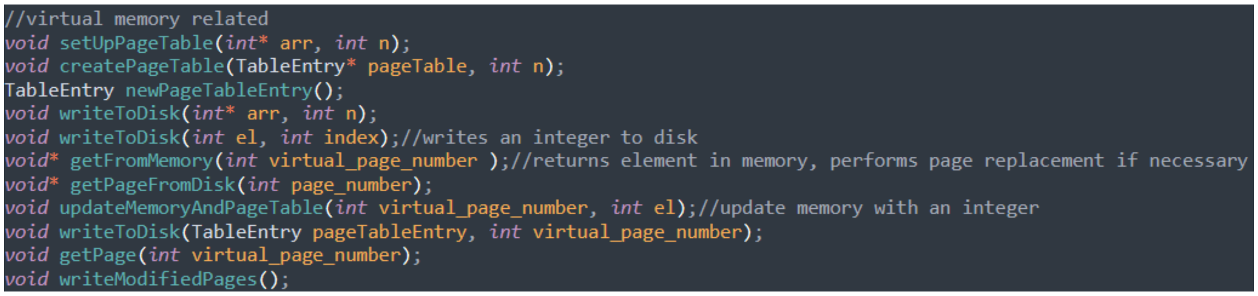
I won't talk much about how they work as those are topics from some other computer engineering classes, like data structures, though it is important to say that they work with integer numbers in this case and are written using C/C++ programming language. Also, when any of these sorting algorithms are accessing an array, the following functions are used.



```
void* getFromMemory(int virtual_page_number );//returns element in memory, performs page replacement if necessary
void* getPageFromDisk(int page_number);
void updateMemoryAndPageTable(int virtual_page_number, int el);//update memory with an integer
void writeToDisk(TableEntry pageTableEntry, int virtual_page_number);
void getPage(int virtual_page_number);
void writeModifiedPages();
```

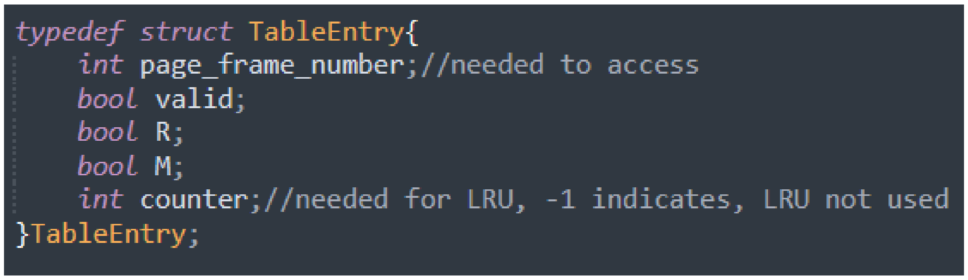
These represent functions used in order to access pages inside memory. Again, when using sorting algorithms, some of these methods are used. Mainly, getFromMemory(virutal_page_number) to get array's entry and updateMemoryAndPageTable(virutal_page_number, new_element) to update page table during things like swapping.

To be able to do this homework properly, I implemented some memory related functions, alongside the virtual memory. Set of following functions are used to manipulate memory



```c
//virtual memory related
void setUpPageTable(int* arr, int n);
void createPageTable(TableEntry* pageTable, int n);
TableEntry newPageTableEntry();
void writeToDisk(int* arr, int n);
void writeToDisk(int el, int index);//writes an integer to disk
void* getFromMemory(int virtual_page_number );//returns element in memory, performs page replacement if necessary
void* getPageFromDisk(int page_number);
void updateMemoryAndPageTable(int virtual_page_number, int el);//update memory with an integer
void writeToDisk(TableEntry pageTableEntry, int virtual_page_number);
void getPage(int virtual_page_number);
void writeModifiedPages();
```

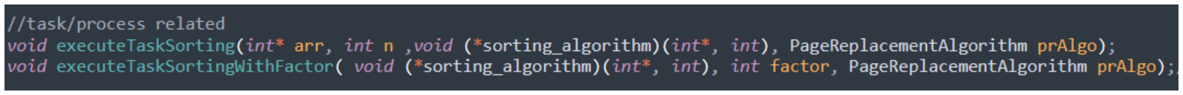
To keep a page table that will track these virtual memory related changes a structure was defined.



```c
typedef struct TableEntry{
    int page_frame_number;//needed to access
    bool valid;
    bool R;
    bool M;
    int counter;//needed for LRU, -1 indicates, LRU not used
}TableEntry;
```

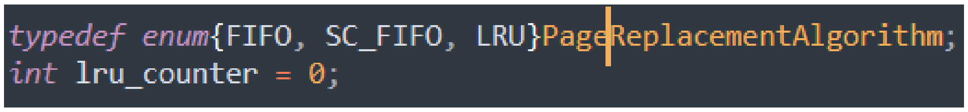
This is good as it allows work for both LRU and FIFO / Second chance FIFO. It was more flexible to use it like this even though LRU does not use R and M bits, and FIFO related algorithms do not use counter. For full optimization, of course a union can be used, though it was found to be unnecessary in this case.

For executing these sorting algorithms, 2 methods were defined. One uses a 'factor' mentioned in the homework and generates somewhat of a random number (big hexadecimal number on which various bitwise operations are performed), the other can use a custom array that will be sorted.



```c
//task/process related
void executeTaskSorting(int* arr, int n ,void (*sorting_algorithm)(int*, int), PageReplacementAlgorithm prAlgo);
void executeTaskSortingWithFactor( void (*sorting_algorithm)(int*, int), int factor, PageReplacementAlgorithm prAlgo);
```

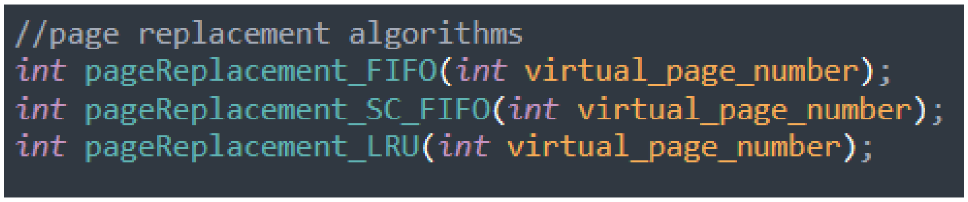
Also, as it can be perceived from the photo, a special type defining page replacement algorithm is included alongside with global counter necessary for tracking pages in LRU algorithm.



```c
typedef enum{FIFO, SC_FIFO, LRU}PageReplacementAlgorithm;
int lru_counter = 0;
```

So accordingly, as can be deducted from the PargeReplacementAlgorithm enumeration, the following page replacement algorithms are implemented:

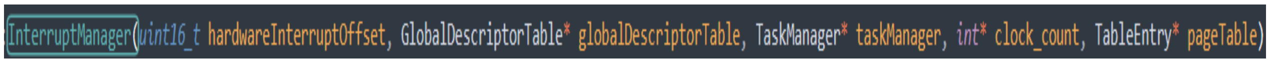
- First in first out
- Second chance first in first out
- Least recently used



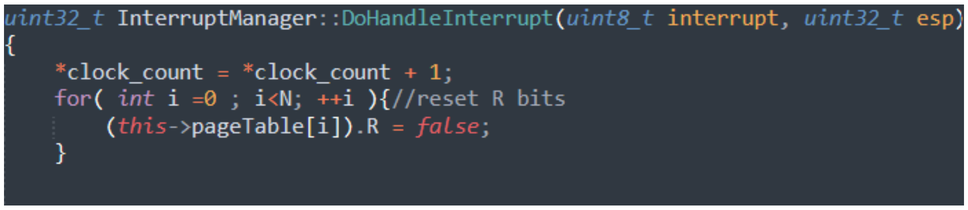
```
//page replacement algorithms
int pageReplacement_FIFO(int virtual_page_number);
int pageReplacement_SC_FIFO(int virtual_page_number);
int pageReplacement_LRU(int virtual_page_number);
```

These functions either analyze the page table (LRU and Second chance FIFO) or go through and choose one of the pages in a queue used by FIFO related page replacement algorithms (FIFO, Second chance FIFO). All in all, their job is to find and return the page number of the page that is being replaced when necessary.

Clock cycles are followed within InterruptManager class. It is updated so that for constructor it additionally takes clock count and page table. This way, in interrupts.cpp where this class is implemented changes of clock count and updating page table (resetting R bit needed for FIFO related functions) can be achieved.



```
InterruptManager(uint16_t hardwareInterruptOffset, GlobalDescriptorTable* globalDescriptorTable, TaskManager* taskManager, int* clock_count, TableEntry* pageTable)
```



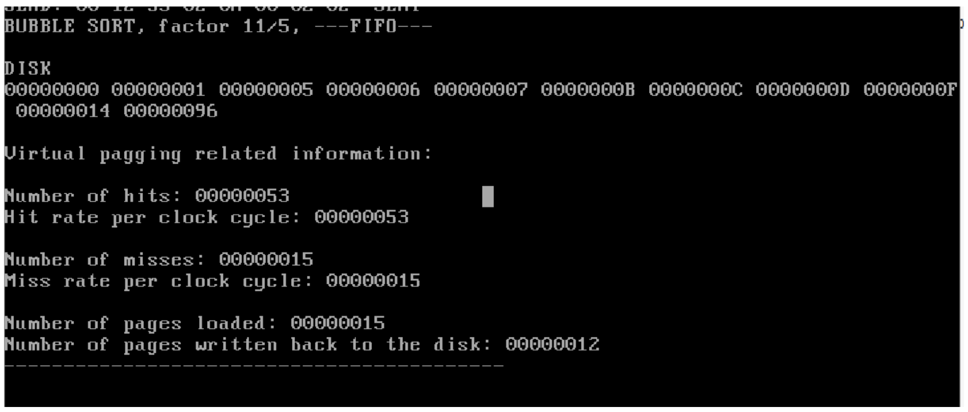
```
uint32_t InterruptManager::DoHandleInterrupt(uint8_t interrupt, uint32_t esp)
{
    *clock_count = *clock_count + 1;
    for( int i =0 ; i<N; ++i ){//reset R bits
        (this->pageTable[i]).R = false;
    }
```

Tests and results of the tests performed are as follows:

Bubble sort with factor 11/5

Page replacement algorithm: FIFO

Unsorted array is { 12, 11, 13, 5, 6, 1, 20, 0, 7, 15, 150 };

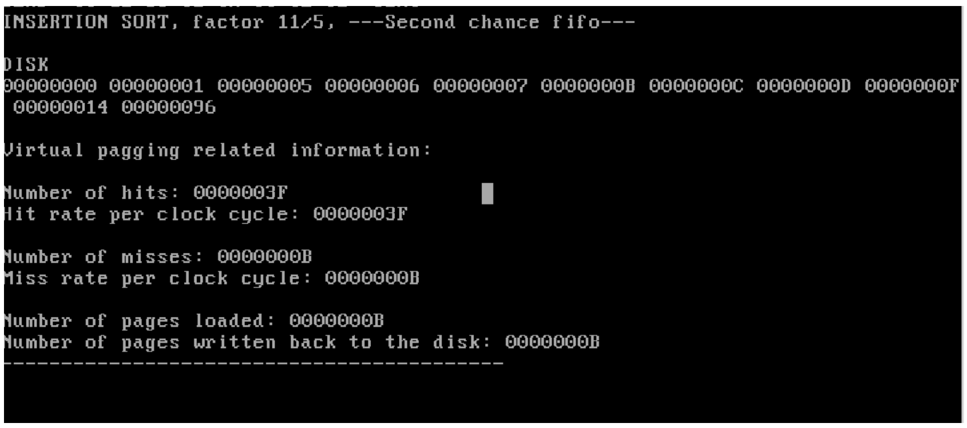


```
BUBBLE SORT, factor 11/5, ---FIFO---

DISK
00000000 00000001 00000005 00000006 00000007 0000000B 0000000C 0000000D 0000000F
 00000014 00000096

Virtual pagging related information:

Number of hits: 00000053
Hit rate per clock cycle: 00000053

Number of misses: 00000015
Miss rate per clock cycle: 00000015

Number of pages loaded: 00000015
Number of pages written back to the disk: 00000012
-----------------------------------------
```

Insertion sort with factor 11/5

Page replacement algorithm: Second chance FIFO

Unsorted array is { 12, 11, 13, 5, 6, 1, 20, 0, 7, 15, 150 };

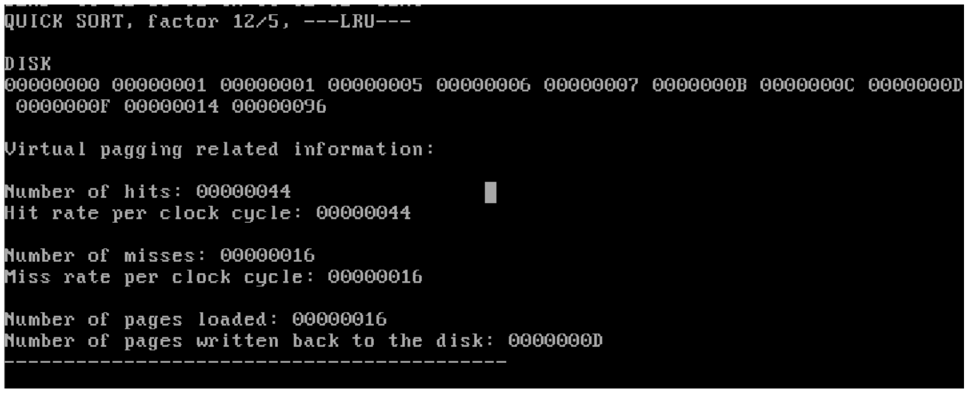


```
INSERTION SORT, factor 11/5, ---Second chance fifo---

DISK
00000000 00000001 00000005 00000006 00000007 0000000B 0000000C 0000000D 0000000F
 00000014 00000096

Virtual pagging related information:

Number of hits: 0000003F
Hit rate per clock cycle: 0000003F

Number of misses: 0000000B
Miss rate per clock cycle: 0000000B

Number of pages loaded: 0000000B
Number of pages written back to the disk: 0000000B
-----------------------------------------
```

Quick sort with factor 12/5

Page replacement algorithm: LRU

Unsorted array is { 12, 11, 13, 5, 6, 1, 20, 0, 7, 15, 150, 1 };

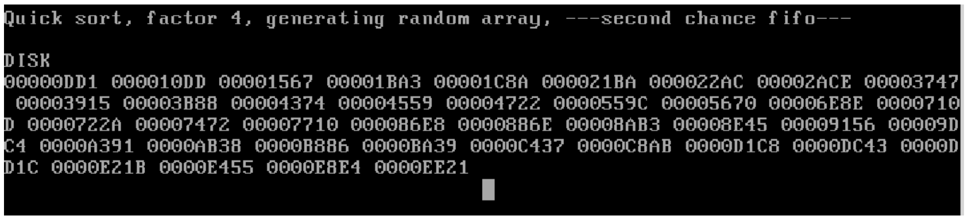


```
QUICK SORT, factor 12/5, ---LRU---

DISK
00000000 00000001 00000001 00000005 00000006 00000007 0000000B 0000000C 0000000D
 0000000F 00000014 00000096

Virtual pagging related information:

Number of hits: 00000044
Hit rate per clock cycle: 00000044

Number of misses: 00000016
Miss rate per clock cycle: 00000016

Number of pages loaded: 00000016
Number of pages written back to the disk: 0000000D
-------------------------------------------
```

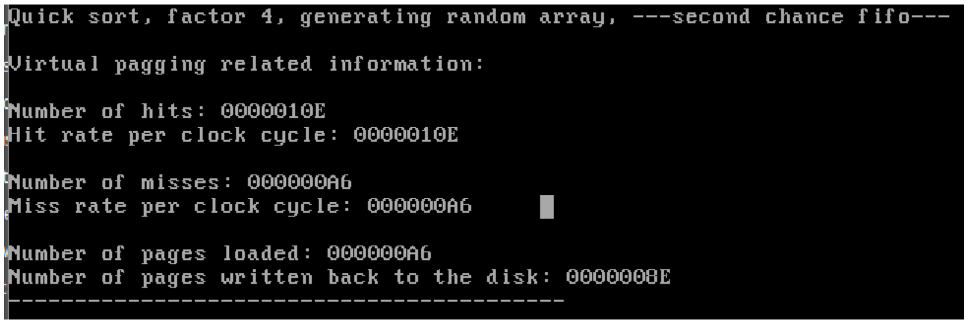
Quick sort with factor 4

Page replacement algorithm: Second chance fifo

Unsorted array is random.



```
Quick sort, factor 4, generating random array, ---second chance fifo---

DISK
00000DD1 000010DD 00001567 00001BA3 00001C8A 000021BA 000022AC 00002ACE 00003747
 00003915 00003B88 00004374 00004559 00004722 0000559C 00005670 00006E8E 0000710
D 0000722A 00007472 00007710 000086E8 0000886E 00008AB3 00008E45 00009156 00009D
C4 0000A391 0000AB38 0000B886 0000BA39 0000C437 0000C8AB 0000D1C8 0000DC43 0000D
D1C 0000E21B 0000E455 0000E8E4 0000EE21
```



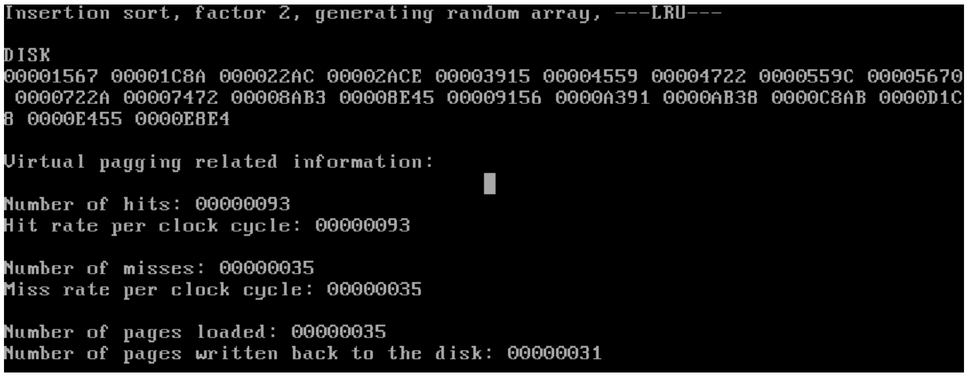
```
Quick sort, factor 4, generating random array, ---second chance fifo---

Virtual pagging related information:

Number of hits: 0000010E
Hit rate per clock cycle: 0000010E

Number of misses: 000000A6
Miss rate per clock cycle: 000000A6

Number of pages loaded: 000000A6
Number of pages written back to the disk: 0000008E
-------------------------------------------
```

Note: photo taken twice since output gets deleted since printf() function provided clears screen!

Insertion sort with factor 2

Page replacement algorithm LRU

Unsorted array is random.

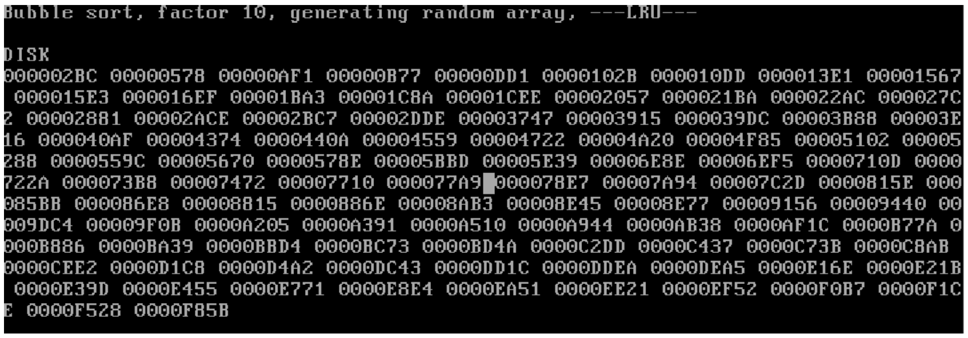


```
Insertion sort, factor 2, generating random array, ---LRU---

DISK
00001567 00001C8A 000022AC 00002ACE 00003915 00004559 00004722 0000559C 00005670
 0000722A 00007472 00008AB3 00008E45 00009156 0000A391 0000AB38 0000C8AB 0000D1C
8 0000E455 0000E8E4

Virtual pagging related information:

Number of hits: 00000093
Hit rate per clock cycle: 00000093

Number of misses: 00000035
Miss rate per clock cycle: 00000035

Number of pages loaded: 00000035
Number of pages written back to the disk: 00000031
```

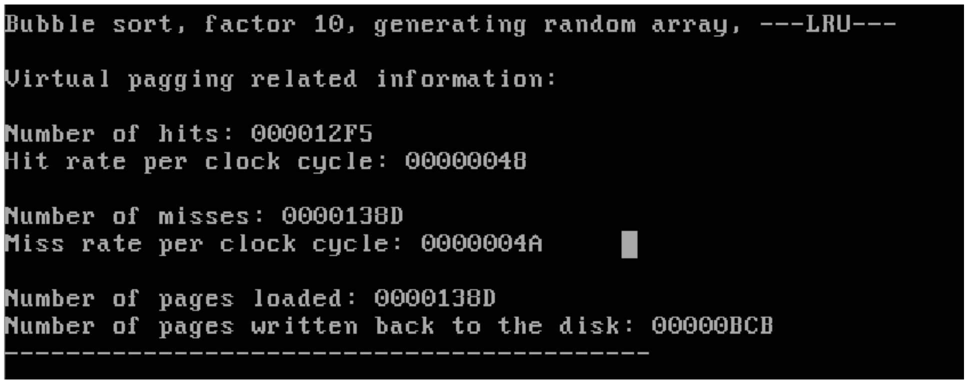
Quick sort with factor 10

Page replacement algorithm: LRU

Unsorted array is random.



```
Bubble sort, factor 10, generating random array, ---LRU---

DISK
000002BC 00000578 00000AF1 00000B77 00000DD1 0000102B 000010DD 000013E1 00001567
 000015E3 000016EF 00001BA3 00001C8A 00001CEE 00002057 000021BA 000022AC 000027C
2 00002881 00002ACE 00002BC7 00002DDE 00003747 00003915 000039DC 00003B88 00003E
16 000040AF 00004374 0000440A 00004559 00004722 00004A20 00004F85 00005102 00005
288 0000559C 00005670 0000578E 00005BBD 00005E39 00006E8E 00006EF5 0000710D 0000
722A 000073B8 00007472 00007710 000077A9 000078E7 00007A94 00007C2D 0000815E 000
085BB 000086E8 00008815 0000886E 00008AB3 00008E45 00008E77 00009156 00009440 00
009DC4 00009F0B 0000A205 0000A391 0000A510 0000A944 0000AB38 0000AF1C 0000B77A 0
000B886 0000BA39 0000BBD4 0000BC73 0000BD4A 0000C2DD 0000C437 0000C73B 0000C8AB
0000CEE2 0000D1C8 0000D4A2 0000DC43 0000DD1C 0000DDEA 0000DEA5 0000E16E 0000E21B
 0000E39D 0000E455 0000E771 0000E8E4 0000EA51 0000EE21 0000EF52 0000F0B7 0000F1C
E 0000F528 0000F85B
```



```
Bubble sort, factor 10, generating random array, ---LRU---

Virtual pagging related information:

Number of hits: 000012F5
Hit rate per clock cycle: 00000048

Number of misses: 0000138D
Miss rate per clock cycle: 0000004A

Number of pages loaded: 0000138D
Number of pages written back to the disk: 00000BCB
----------------------------------------
```

Note: photo taken twice since output gets deleted since printf() function provided clears screen!