May 29, 2021

GTU Department of Computer Engineering
CSE312/CSE504 - Spring 2021
Homework 3 Report

Akif Kartal
171044098

1 Problem Definition

The problem is to implement a simulated and simplified virtual memory management system and a number of page replacement algorithms in C/C++.

2 Solution

In this Homework, I only implemented part1 because of time constraint.

2.1 Part1

2.1.1 Virtual Memory Structure

My virtual memory structure is same as with book. I implement following virtual memory structure directly.

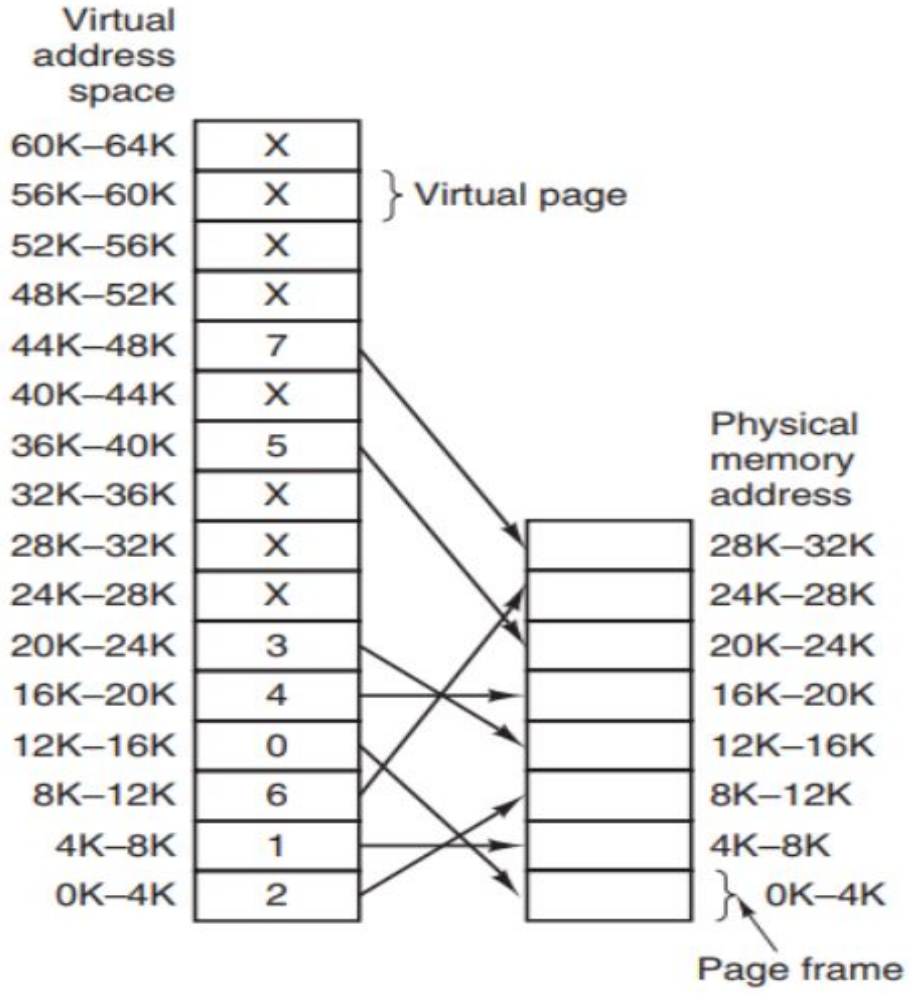


Figure 1: My virtual Memory Structure

2.1.2 Page Table Structure

My Page Table is used while adress translation as following picture.

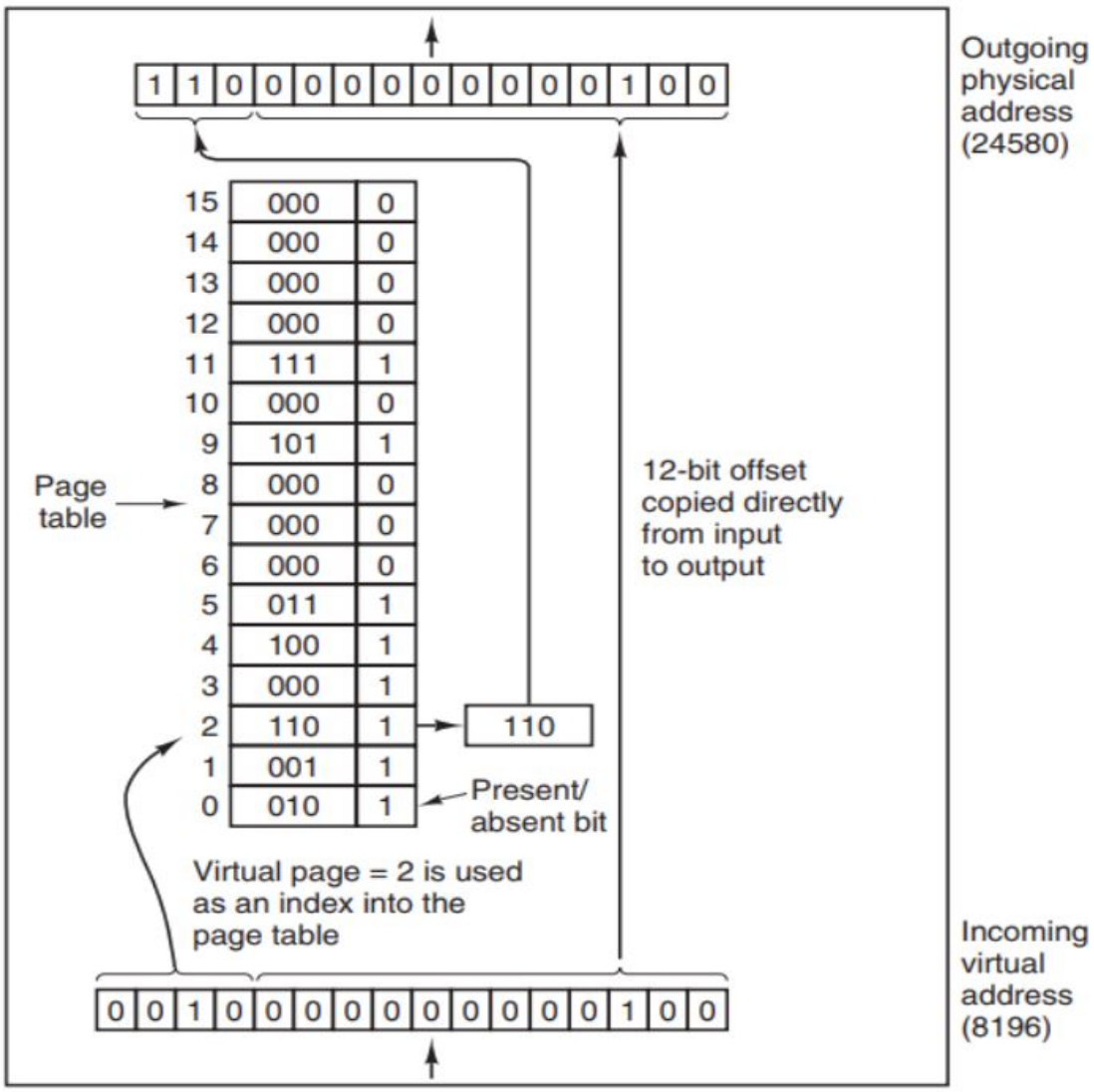


Figure 2: My Page Table usage Structure

2.1.3 Page Table Entry Structure

My Page Table entry is same as with following picture.

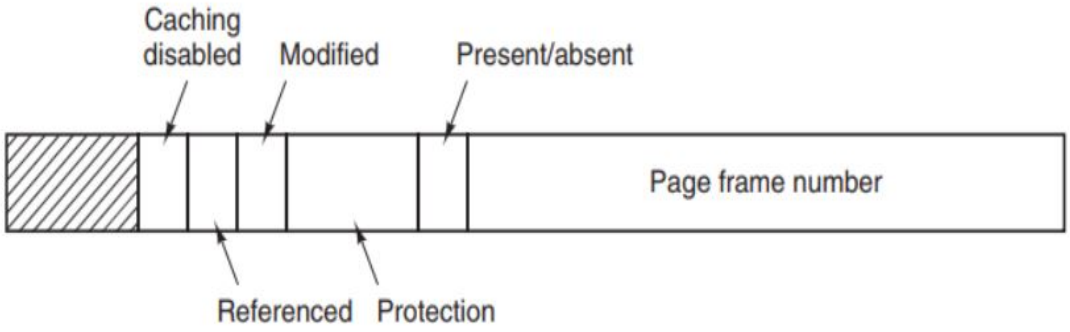


Figure 3: Page Table Entry Structure

My Page Table Entry Code

```
7   class Entry{
8       friend class PageTable;
9   public:
10      Entry();
11
12      unsigned int getFrameNumber() const;
13      bool isReferenced() const;
14      bool isModified() const;
15      bool isPresent() const;
16
17      void setReferenced(bool);
18      void setModified(bool);
19      void setPresent(bool);
20      void setPageFrameNumber(unsigned int frmNumber);
21
22  private:
23      bool referenced;
24      bool modified;
25      bool present;
26      unsigned int pageFrameNumber;
27
28  };
```

Figure 4: My Page Table Entry Class in C++

2.1.4 My Page Table using my Entry Structure

My Page Table Code

```
10  class PageTable{
11  public:
12      PageTable(unsigned int frmSize, unsigned int numVr, unsigned int numPh);
13      void set(unsigned int address, int value);
14      int get(unsigned int address);
15      void printInfo() const;
16      bool isPresent(unsigned int address) const;
17      void setModified(unsigned int address);
18      ~PageTable();
19  private:
20      unsigned int frameSize;
21      unsigned int pageSize;
22      unsigned int numVirFrame;
23      unsigned int numPhysFrame;
24      unsigned int numPhys;
25      unsigned int numVir;
26      unsigned int virtualSize;
27      unsigned int physSize;
28
29      Entry *table;
30      int getIndex(unsigned int address) const;
31
32  };
```

Figure 5: My Page Table Class in C++

Calculating the Page Table Size

In order to calculate page table size I did following operations;

```
PageTable::PageTable(unsigned int frmSize, unsigned int numVr, unsigned int numPh)
:frameSize(frameSize),numVir(numVr),numPhys(numPh),pageSize(pow(2,frmSize)),numVirFrame(pow(2,numVr)),numPhysFrame(pow(2,numPh)){

    virtualSize = numVirFrame*pageSize;
    physSize = numPhysFrame*pageSize;
    table = new Entry[numVirFrame];

}
```

Figure 6: Initializing Page Table

Note that page table size is equal to virtual memory size because there must be an entry for each virtual address in page table.

Simply my page table size formula is following;

$TableSize = 2^{\text{numVirtual}}$

Now, since we have implemented page table structure, we can implement virtual memory structure.

2.1.5 My Virtual Memory

```
 9  class VirtualMemory
10  {
11  public:
12      VirtualMemory(unsigned int frmSize, unsigned int numVr, unsigned int numPy, unsigned int interval);
13      void set(unsigned int index, int value, char *tName);
14      int get(unsigned int index, char *tName);
15      ~VirtualMemory();
16
17  private:
18      unsigned int frameSize;
19      unsigned int numPhys;
20      unsigned int numVir;
21      unsigned int virtualSize;
22      unsigned int physicalSize;
23      unsigned int printInterval;
24
25      int *memory; //physical memory
26      PageTable *pageTable; // virtual memory
27  };
```

Figure 7: My Virtual Memory Class in C++

As you can see from code, I have get and set method as expected in pdf file.

Initializing Virtual Memory

```
28  VirtualMemory::VirtualMemory(unsigned int frmSize, unsigned int numVr, unsigned int numPy, unsigned int interval)
29      : frameSize(frmSize), numPhys(numPy), numVir(numVr), printInterval(interval)
30  {
31      virtualSize = pow(2,frameSize) * pow(2,numVir);
32      physicalSize = pow(2,frameSize) * pow(2,numPhys);
33      memory = new int[physicalSize];
34
35      for (int i = 0; i < physicalSize; i++)
36          memory[i] = 0;
37      pageTable = new PageTable(frameSize,numVir,numPhys);
38
39  }
```

Figure 8: Initialize Virtual Memory

2.2 Testing My Virtual Memory Data Structure

Before start to test first we need to understand my structure and how it works.

My virtual Memory Data Structure Explanation

FrameSize = 12
NumPhysical = 5
NumVirtual = 10

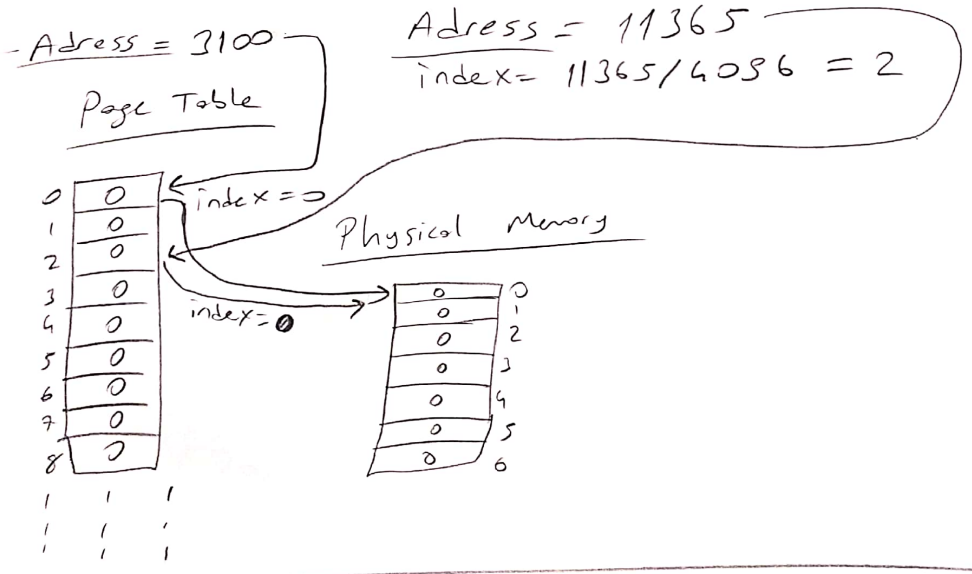
Virtual Memory Total Size = $2^{12} * 2^{10}$
Physical Memory Total Size = $2^{12} * 2^{5}$
Page Size = 2^{12} = 4096 = 4K

Page Table and Address Translation

given Address = 3100

Index = 3100/4096 = 0

Note = Initially all page frame numbers in page table are 0.



As you can see, first we translate address into page table index then we go there we take page frame number from there and then we go that page in physcial memory.

Figure 9: My Structure of Virtual Memory

Let see how this work in test; **Test Code;**

```
4   int main(int argc, char const *argv[])
5   {
6       char arr[20] ="test";
7       VirtualMemory vtr(12,10,5,1000); //create virtual memory
8       vtr.set(3100,2,arr);
9       vtr.set(4289,5,arr);
10      vtr.set(65897,3,arr);
11
12
13      cout << "Data in 3100: " << vtr.get(3100,arr) << endl;
14      cout << "Data in 11365: " << vtr.get(11365,arr) << endl;
15      cout << "Data in 29896: " << vtr.get(29896,arr) << endl;
16      cout << "Data in 65897: " << vtr.get(65897,arr) << endl;
17
18      return 0;
19  }
```

Figure 10: Test C++ Code

Result;

```
cse312@ubuntu:~/Desktop/operating-system/HW3$ g++ test.cpp
cse312@ubuntu:~/Desktop/operating-system/HW3$ ./a.out
Data in 3100: 3
Data in 11365: 3
Data in 29896: 3
Data in 65897: 3
cse312@ubuntu:~/Desktop/operating-system/HW3$
```

Figure 11: Result of Test C++ Code

Now, since at the besining all page frame numbers in page table is zero(0) even we try to access address 60341 or 4586 we will get same data, because they are showing same page in physical memory. Note that this is only for initial case.

Figure 12: Comment on Result

Let change page frame numbers at the beginning and run again;

```
33  PageTable::PageTable(unsigned int frmSize, unsigned int numVr, unsigned int numPh)
34  :frameSize(frameSize),numVir(numVr),numPhys(numPh),pageSize(pow(2,frmSize)),numVirFrame(pow(2,numVr)),numPhysFrame(pow(2,numPh)){
35
36      virtualSize = numVirFrame*pageSize;
37      physSize = numPhysFrame*pageSize;
38      table = new Entry[numVirFrame];
39      for (int i = 0; i < numVirFrame; i++)
40      {
41          table[i].setPageFrameNumber(i);
42      }
43  }
```

Figure 13: Initialize Page Frame numbers in page Table

New Result;

```
cse312@ubuntu:~/Desktop/operating-system/HW3$ g++ test.cpp
cse312@ubuntu:~/Desktop/operating-system/HW3$ ./a.out
Data in 3100: 2
Data in 11365: 0
Data in 29896: 0
Data in 65897: 3
cse312@ubuntu:~/Desktop/operating-system/HW3$
```

Figure 14: New Result

2.3 Other Parts of Homework

Now, we have created Virtual Memory Data structure successfully, we can start to implement page replacement algorithms but because of time problem I leave homework as it is because I couldn't implement them.