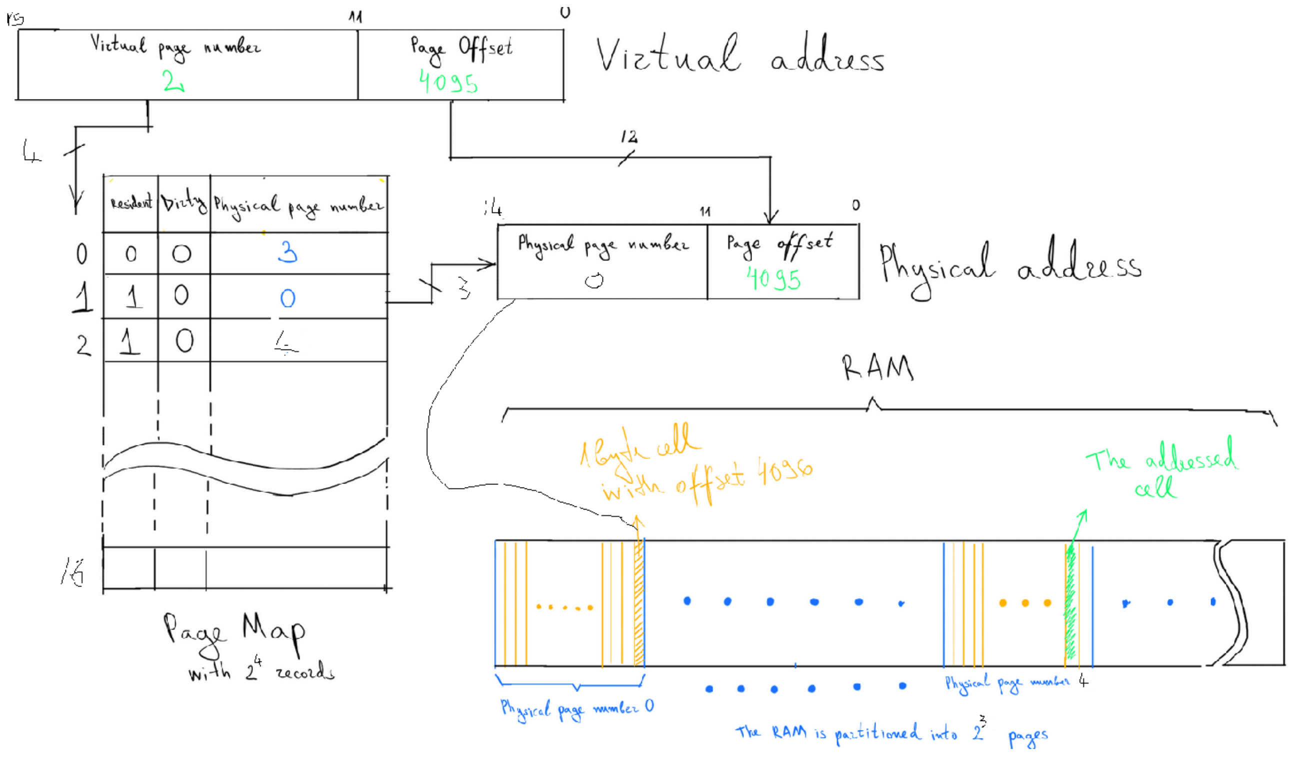
GIT Department of Computer Engineering
CSE 312 – Spring 2023

Homework 2 #Report

Okan Torun
1801042662

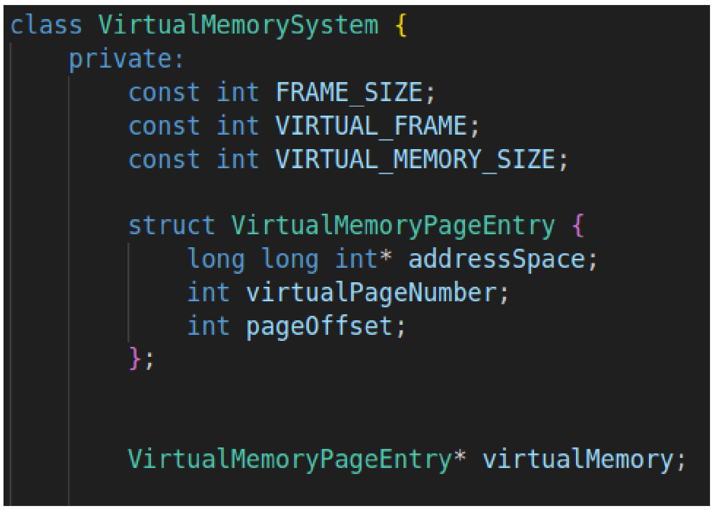
Virtual Memory Management System

For the virtual memory management system, I first started with a virtual memory design, I initialized certain features of my virtual memory with the parameters entered at the beginning in the structure I designed.



1) Virtual Memory

For the virtual memory management system, I first started with a virtual memory design, I initialized certain features of my virtual memory with the parameters entered at the beginning in the structure I designed.



```
class VirtualMemorySystem {
    private:
        const int FRAME_SIZE;
        const int VIRTUAL_FRAME;
        const int VIRTUAL_MEMORY_SIZE;

        struct VirtualMemoryPageEntry {
            long long int* addressSpace;
            int virtualPageNumber;
            int pageOffset;
        };


        VirtualMemoryPageEntry* virtualMemory;
```

For the parameters entered at the beginning;

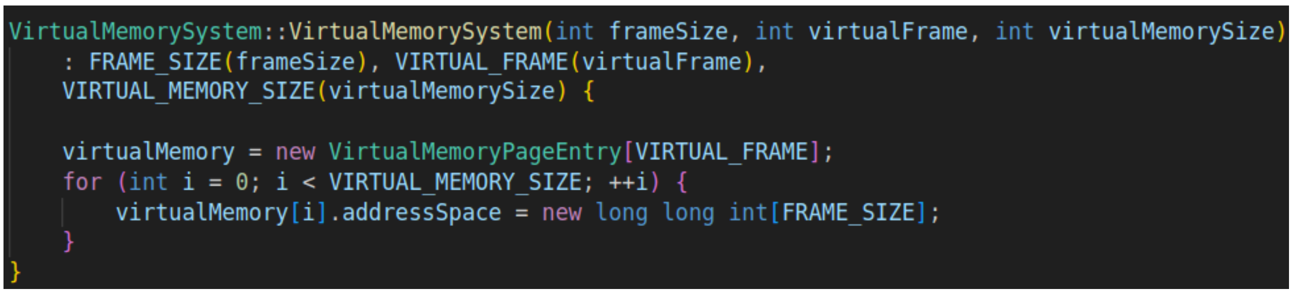
./operateArrays 12 3 4 WSClock inverted 10000 diskFileName.dat

For example, suppose we enter this.
The number 12 here determines the size of each frame in memory. With Yabi 2^12 representation, each frame consists of a size of 4kb.

Does it determine the number of 3 Physical Memory frames, I will have 8 physical frames with 2^3.

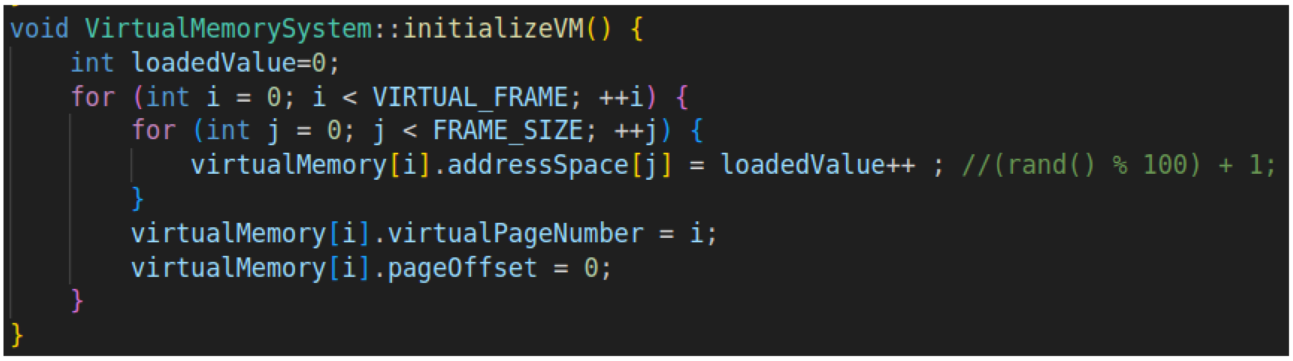
Does it determine the number of 4 Virtaul Memory frames, I will have 16 virtual frames with 2^4.

My Virtual Memory Size value is determined by the number of virtual frames * frame Size, so in the example I gave above it is 16 * 4096 = 65536



```
VirtualMemorySystem::VirtualMemorySystem(int frameSize, int virtualFrame, int virtualMemorySize)
    : FRAME_SIZE(frameSize), VIRTUAL_FRAME(virtualFrame),
    VIRTUAL_MEMORY_SIZE(virtualMemorySize) {

    virtualMemory = new VirtualMemoryPageEntry[VIRTUAL_FRAME];
    for (int i = 0; i < VIRTUAL_MEMORY_SIZE; ++i) {
        virtualMemory[i].addressSpace = new long long int[FRAME_SIZE];
    }
}
```

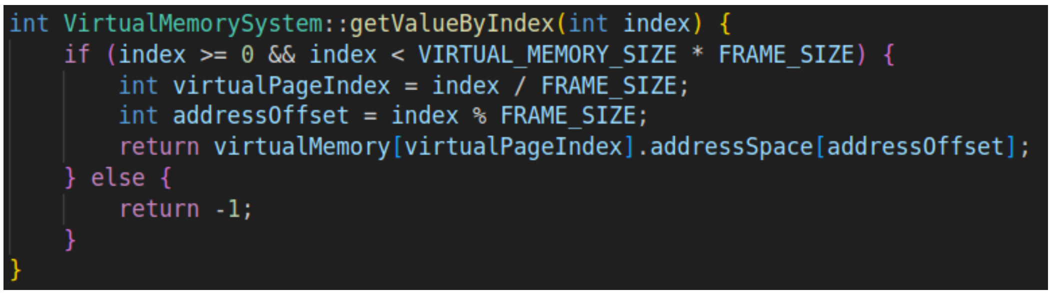
With these determined values, I create the VirtualMemory with the constructor and open the necessary fields.In the structure I use, each virtual frame contains address Space, virtual Page number and pageOffSet.



```
void VirtualMemorySystem::initializeVM() {
    int loadedValue=0;
    for (int i = 0; i < VIRTUAL_FRAME; ++i) {
        for (int j = 0; j < FRAME_SIZE; ++j) {
            virtualMemory[i].addressSpace[j] = loadedValue++ ; //(rand() % 100) + 1;
        }
        virtualMemory[i].virtualPageNumber = i;
        virtualMemory[i].pageOffset = 0;
    }
}
```

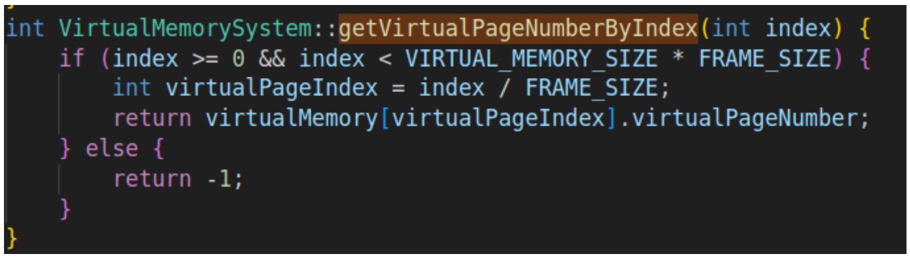
Afterwards, I fill the entire VirtualMemory with numbers as requested in the assignment.

Set/Get Functions



```
int VirtualMemorySystem::getValueByIndex(int index) {
    if (index >= 0 && index < VIRTUAL_MEMORY_SIZE * FRAME_SIZE) {
        int virtualPageIndex = index / FRAME_SIZE;
        int addressOffset = index % FRAME_SIZE;
        return virtualMemory[virtualPageIndex].addressSpace[addressOffset];
    } else {
        return -1;
    }
}
```

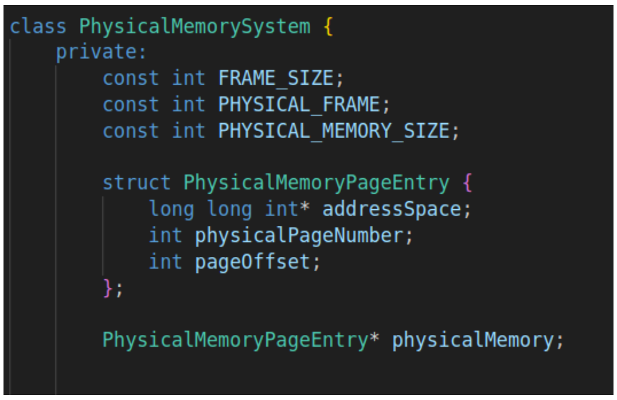
Here is the get function that I use to access the numbers that I initially filled in virtual memory and set them to physical memory.
If I want to access the 30,000 index in Virtual Memory, I find the frame and pageOffSet values there and return the relevant value.



```
int VirtualMemorySystem::getVirtualPageNumberByIndex(int index) {
    if (index >= 0 && index < VIRTUAL_MEMORY_SIZE * FRAME_SIZE) {
        int virtualPageIndex = index / FRAME_SIZE;
        return virtualMemory[virtualPageIndex].virtualPageNumber;
    } else {
        return -1;
    }
}
```

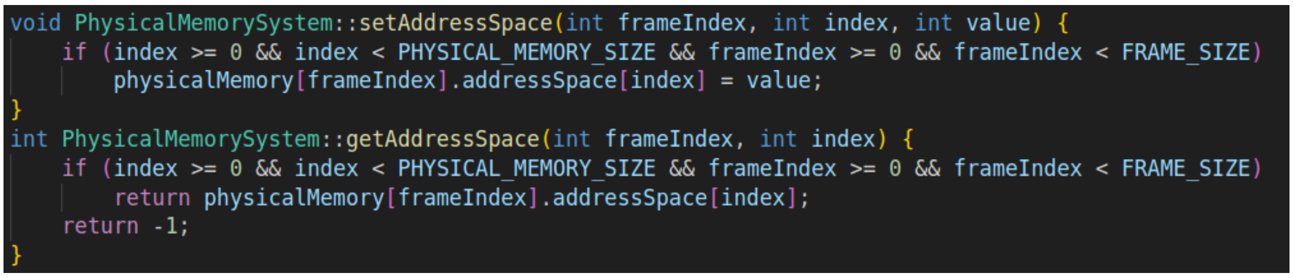
Here, it is designed to access the index in virtual memory and find the pageNumber it represents in physical memory. With the virtualAddress returning from here, I go to the relevant index in the PageTable and access the physicalNumber there.

2) Physical Memory



```
class PhysicalMemorySystem {
    private:
        const int FRAME_SIZE;
        const int PHYSICAL_FRAME;
        const int PHYSICAL_MEMORY_SIZE;

        struct PhysicalMemoryPageEntry {
            long long int* addressSpace;
            int physicalPageNumber;
            int pageOffset;
        };

        PhysicalMemoryPageEntry* physicalMemory;
```

It is designed for physical memory, very similar to the structure designed for Virtual Memory, the physical memory structure is initialized with the values entered from the terminal.

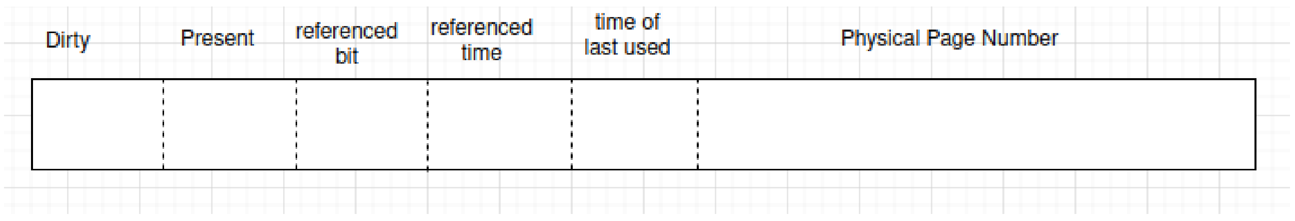


```
void PhysicalMemorySystem::setAddressSpace(int frameIndex, int index, int value) {
    if (index >= 0 && index < PHYSICAL_MEMORY_SIZE && frameIndex >= 0 && frameIndex < FRAME_SIZE)
        physicalMemory[frameIndex].addressSpace[index] = value;
}
int PhysicalMemorySystem::getAddressSpace(int frameIndex, int index) {
    if (index >= 0 && index < PHYSICAL_MEMORY_SIZE && frameIndex >= 0 && frameIndex < FRAME_SIZE)
        return physicalMemory[frameIndex].addressSpace[index];
    return -1;
}
```

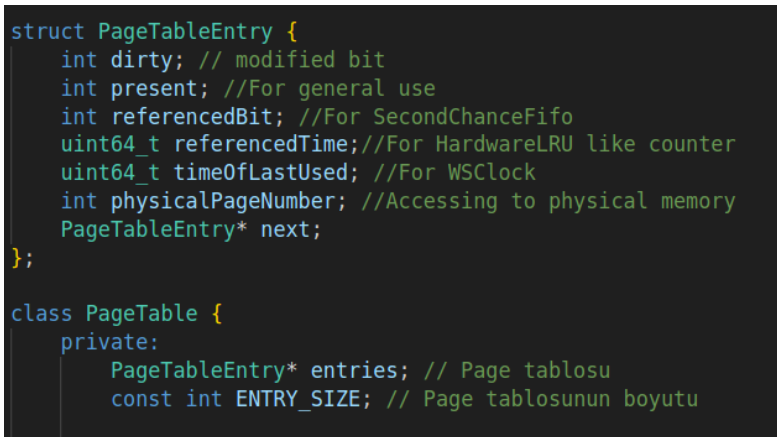
With the set/get functions here, after I access the virtual memory with the index, I provide physical memory access with the PageTable. Since the PageOfSet value is common, my access to the data is provided with these values.Index parameter is actually my pageOfSet value, I determine which cell to write in.

3) Page Table

The reason for using PageTable is because address conversion is done. So when I want to access an index in virtual memory, it finds the frame there and the physical number representing the pageOffset in the table.



Each entry of the created Page Table represents the features in the figure. The page replacement algorithms used check the values here and perform their operations.



```
struct PageTableEntry {
    int dirty; // modified bit
    int present; //For general use
    int referencedBit; //For SecondChanceFifo
    uint64_t referencedTime;//For HardwareLRU like counter
    uint64_t timeOfLastUsed; //For WSClock
    int physicalPageNumber; //Accessing to physical memory
    PageTableEntry* next;
};

class PageTable {
    private:
        PageTableEntry* entries; // Page tablosu
        const int ENTRY_SIZE; // Page tablosunun boyutu
```

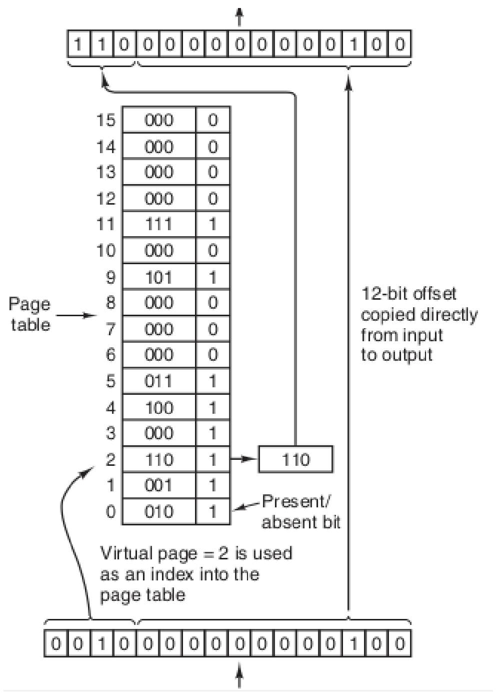
Dirty bit : It represents whether the page has been updated, there is no need to rewrite the memory when deleting the page with the dirty bit 0.

Present: Present bit indicates whether the desired page is in memory.

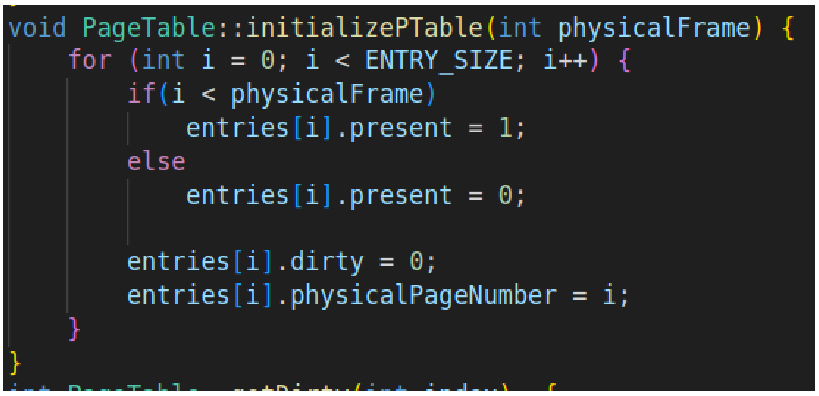
Referenced Bit: When trying to access the Page, the referenced bit is set to 1, SC and WSClock algorithms control this bit and perform the replace operation.

Referenced Time: This bit is used by the LRU algorithm. The counter incremented externally is transferred to this bit every time a hit is made.

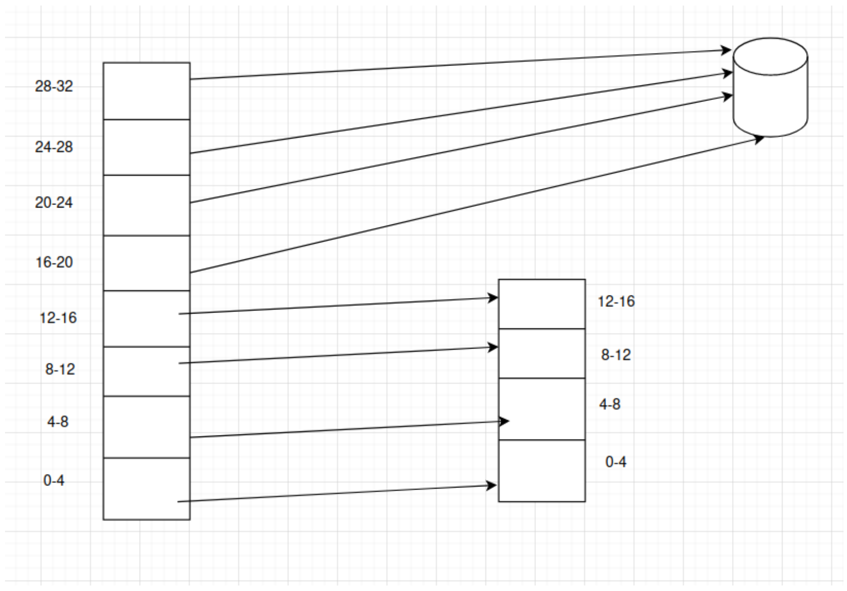
Time of Last Used : This bit is used in the WSClock algorithm. During each page fault, this bit is updated by the algorithm according to certain conditions.



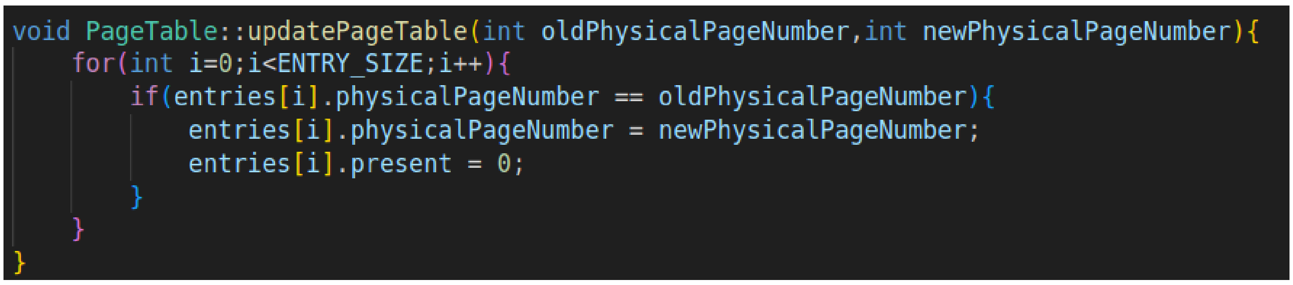
The page table structure I designed is exactly like this. The page table has as many page entries as the number of virtual frames in the virtual address. The virtual page number of the virtual address indicates which index the page table should go to. If the present bit of that entry is 1, the physical page number in the entry is sent to the frame in the physical page number and the value is reached. If the present bit is 0, the desired page is pulled from the disk and therefore page replacement is performed.
ENTRY_SIZE = Virtual Frame Number



```
void PageTable::initializePTable(int physicalFrame) {
    for (int i = 0; i < ENTRY_SIZE; i++) {
        if(i < physicalFrame)
            entries[i].present = 1;
        else
            entries[i].present = 0;

        entries[i].dirty = 0;
        entries[i].physicalPageNumber = i;
    }
}
```



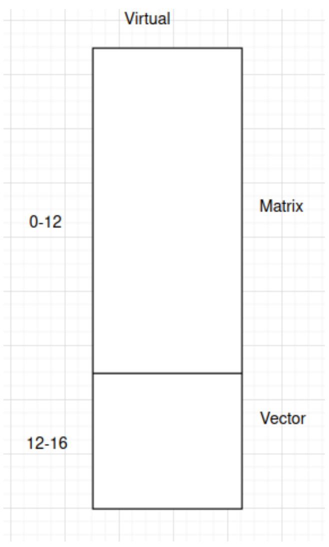
Initially, I initialized the Page Table construction to be mapped like this. It goes to virtual and physical, respectively. The excesses do not show physical.I have assigned physical page numbers to virtual addresses that do not fit into RAM, but they do not show physical memory, they will fill the disk.

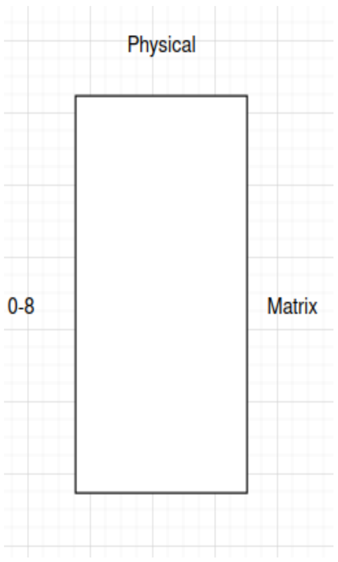


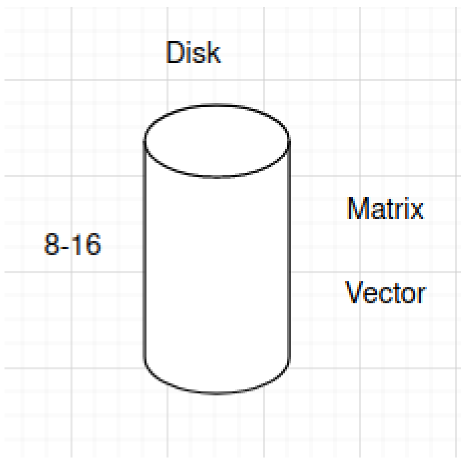
```
void PageTable::updatePageTable(int oldPhysicalPageNumber,int newPhysicalPageNumber){
    for(int i=0;i<ENTRY_SIZE;i++){
        if(entries[i].physicalPageNumber == oldPhysicalPageNumber){
            entries[i].physicalPageNumber = newPhysicalPageNumber;
            entries[i].present = 0;
        }
    }
}
```

I also perform Page Table update in page replacement cases. Instead of a virtual address pointing to physical memory, another virtual frame now points to the same place.

Page Replacement Algorithms

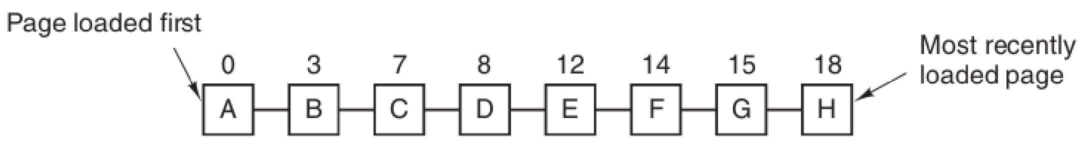




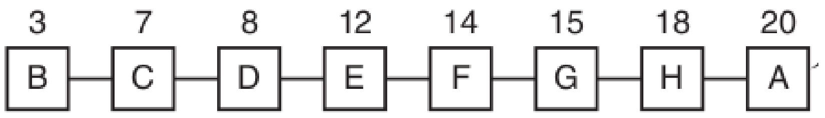


When I want to perform multiplication in the image in the figure, I need to reach the vector value after reaching the matrix value. But the virtual addresses that contain the vector values point to the disk. That's why I have to bring the vector to ram using page replacement algorithms from disk. So I have to do the necessary operations.

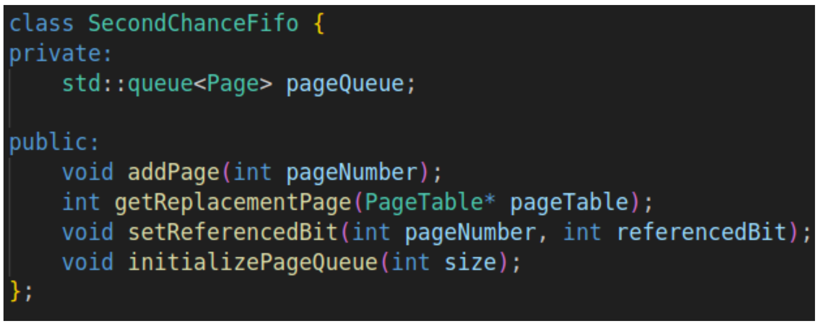
Second-Chance



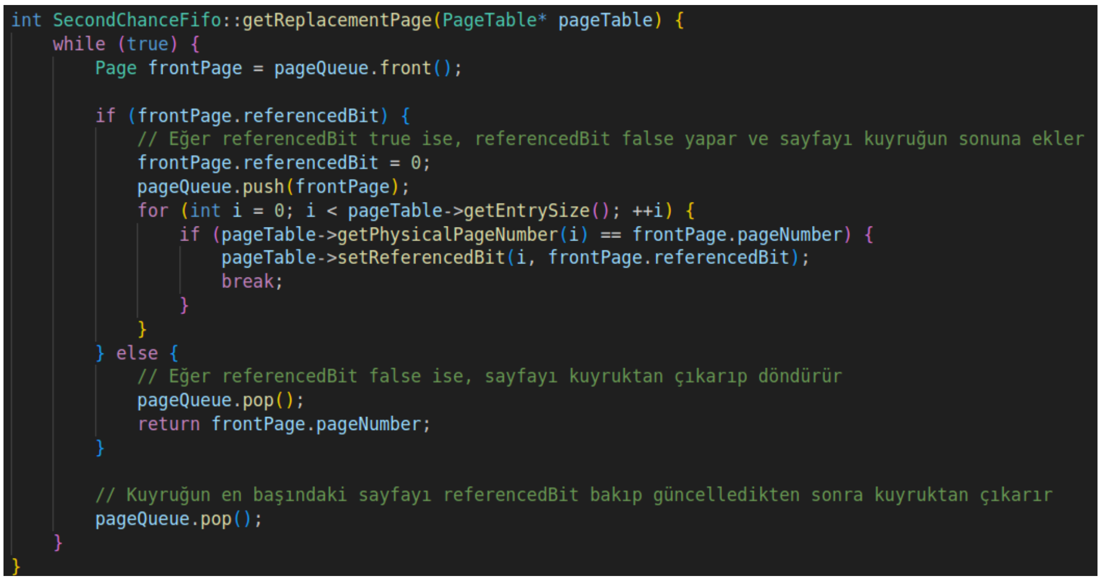
I used a queue<pageEntry> in the seconda chance algorithm, I stored all the pages that I loaded in physical memory at first in this queue. In the image above, let's assume that the first page I load on queue is A, when I need to replace my page, I start checking from A at the very beginning of the queue. I also check the Referenced bit. If the Refrenced bit of A is 0, I change A, if it is 1 I reset the referenced bit of A and send it to the very end of the queue and check the next page.



In the example above, A was normally the oldest page, but since the reference bit is 1, we gave it a second chance.

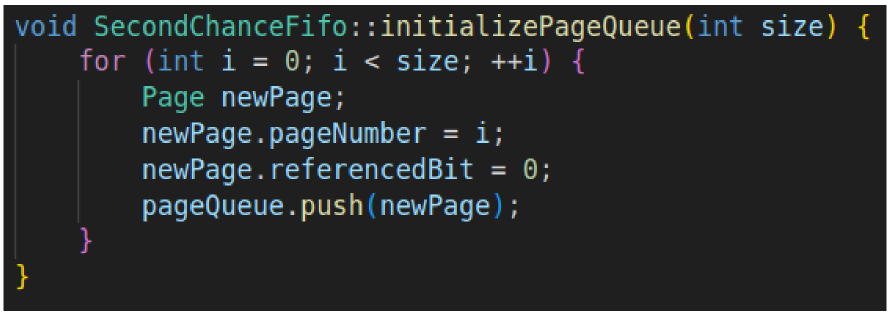


```
class SecondChanceFifo {
private:
    std::queue<Page> pageQueue;

public:
    void addPage(int pageNumber);
    int getReplacementPage(PageTable* pageTable);
    void setReferencedBit(int pageNumber, int referencedBit);
    void initializePageQueue(int size);
};
```



```cpp
int SecondChanceFifo::getReplacementPage(PageTable* pageTable) {
    while (true) {
        Page frontPage = pageQueue.front();

        if (frontPage.referencedBit) {
            // Eğer referencedBit true ise, referencedBit false yapar ve sayfayı kuyruğun sonuna ekler
            frontPage.referencedBit = 0;
            pageQueue.push(frontPage);
            for (int i = 0; i < pageTable->getEntrySize(); ++i) {
                if (pageTable->getPhysicalPageNumber(i) == frontPage.pageNumber) {
                    pageTable->setReferencedBit(i, frontPage.referencedBit);
                    break;
                }
            }
        } else {
            // Eğer referencedBit false ise, sayfayı kuyruktan çıkarıp döndürür
            pageQueue.pop();
            return frontPage.pageNumber;
        }

        // Kuyruğun en başındaki sayfayı referencedBit bakıp güncelledikten sonra kuyruktan çıkarır
        pageQueue.pop();
    }
}
```

The implementation of the structure I described above is like this. There is a page to replace based on the queue in the queue and the refernce bit.

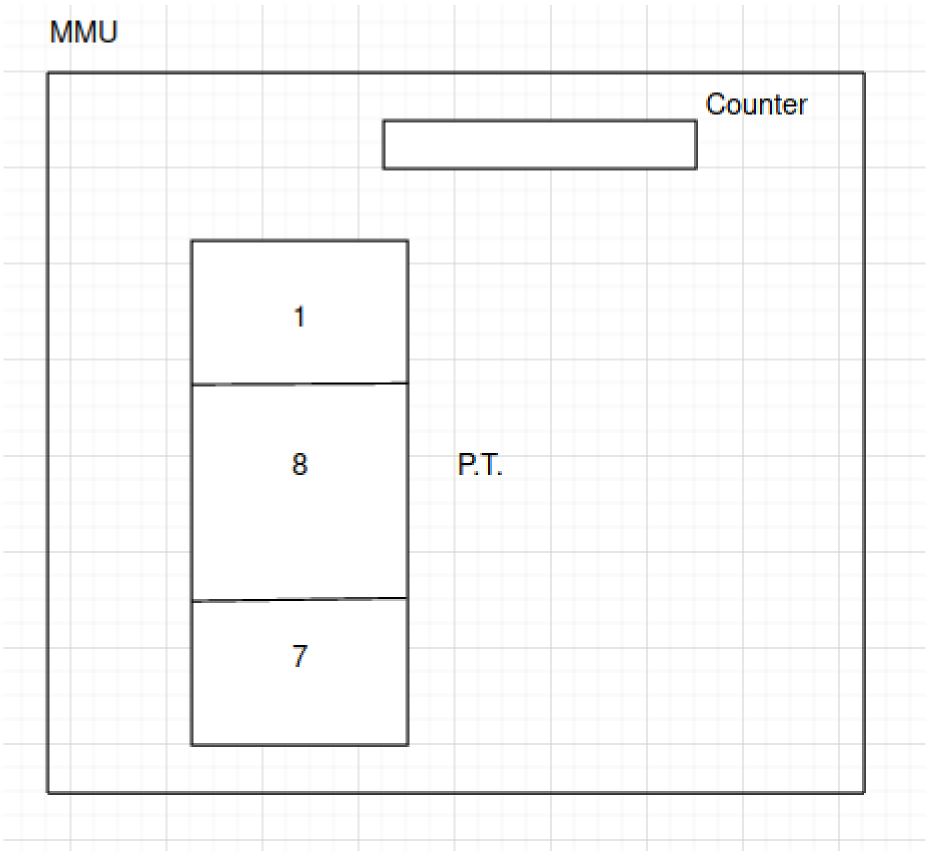


```cpp
void SecondChanceFifo::initializePageQueue(int size) {
    for (int i = 0; i < size; ++i) {
        Page newPage;
        newPage.pageNumber = i;
        newPage.referencedBit = 0;
        pageQueue.push(newPage);
    }
}
```

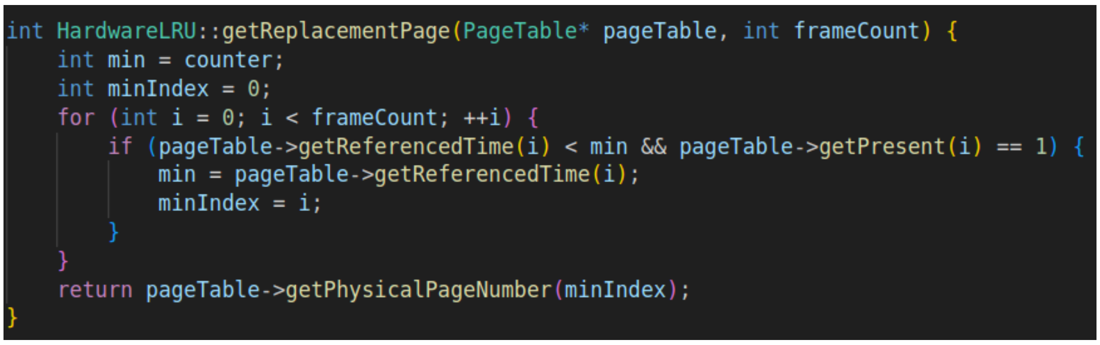
Initially, the queue structure is set up.

LRU Algorithm

In the given assignment, we were asked to simulate hardware in algorithms. Therefore, I designed the Hardware-LRU structure for the LRU algorithm. A counter is kept globally in the Hardware-LRU. For each memory access, we set the counter value here to the referencedTime variable in the page table for the accessed Page.

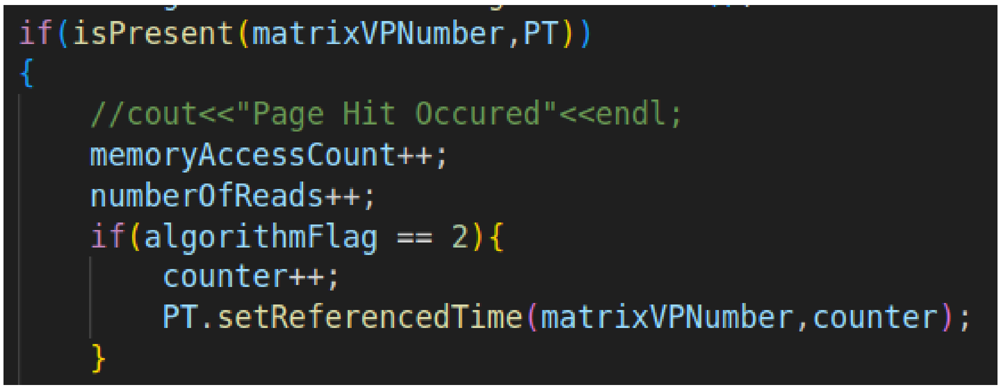


In the image I drew, the numbers written in the page table are the counter values that represent the physical page in each entry. When we look at the image above, the page with the lowest counter value is the last referenced page. Therefore, the counter value in the LRU algorithm when the page fault occurs. The lowest page is removed from memory.



```
int HardwareLRU::getReplacementPage(PageTable* pageTable, int frameCount) {
    int min = counter;
    int minIndex = 0;
    for (int i = 0; i < frameCount; ++i) {
        if (pageTable->getReferencedTime(i) < min && pageTable->getPresent(i) == 1) {
            min = pageTable->getReferencedTime(i);
            minIndex = i;
        }
    }
    return pageTable->getPhysicalPageNumber(minIndex);
}
```

As seen in the implementation, the number of the page with the lowest counter value is returned.

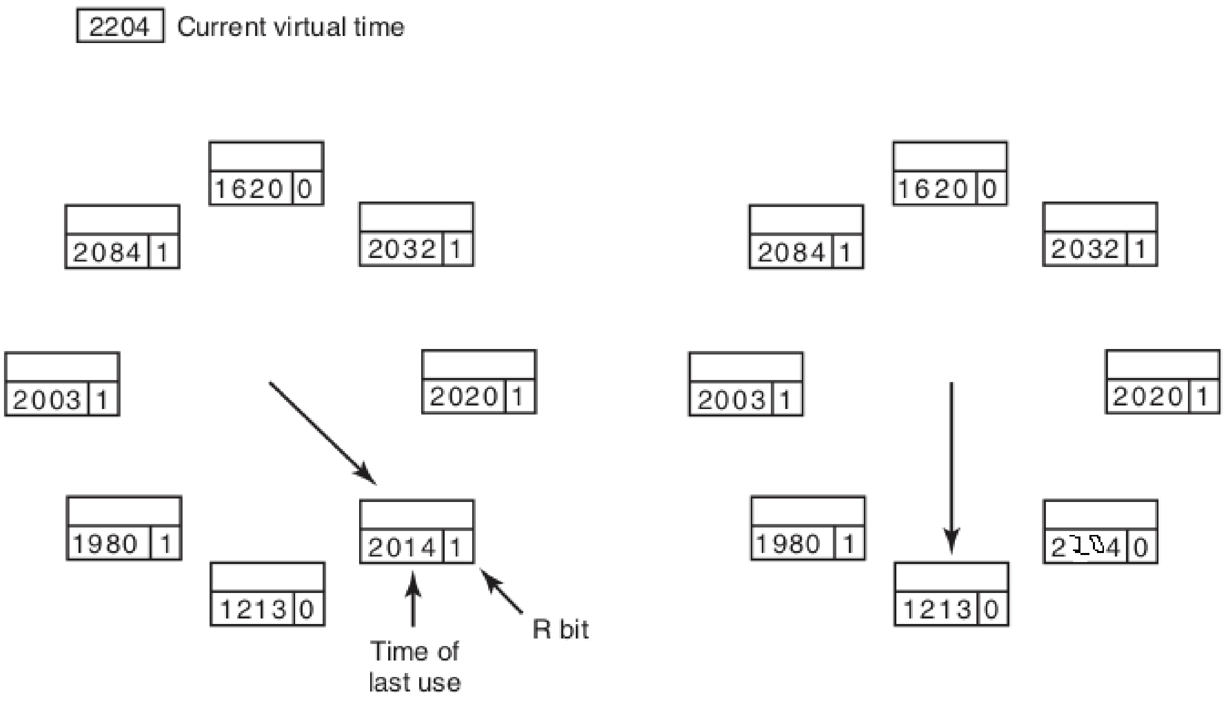


```
if(isPresent(matrixVPNumber,PT))
{
    //cout<<"Page Hit Occured"<<endl;
    memoryAccessCount++;
    numberOfReads++;
    if(algorithmFlag == 2){
        counter++;
        PT.setReferencedTime(matrixVPNumber,counter);
    }
```

As seen in the test code, if the searched value is in memory, the counter is increased and the final state of the counter is set to the rerefrenced page.

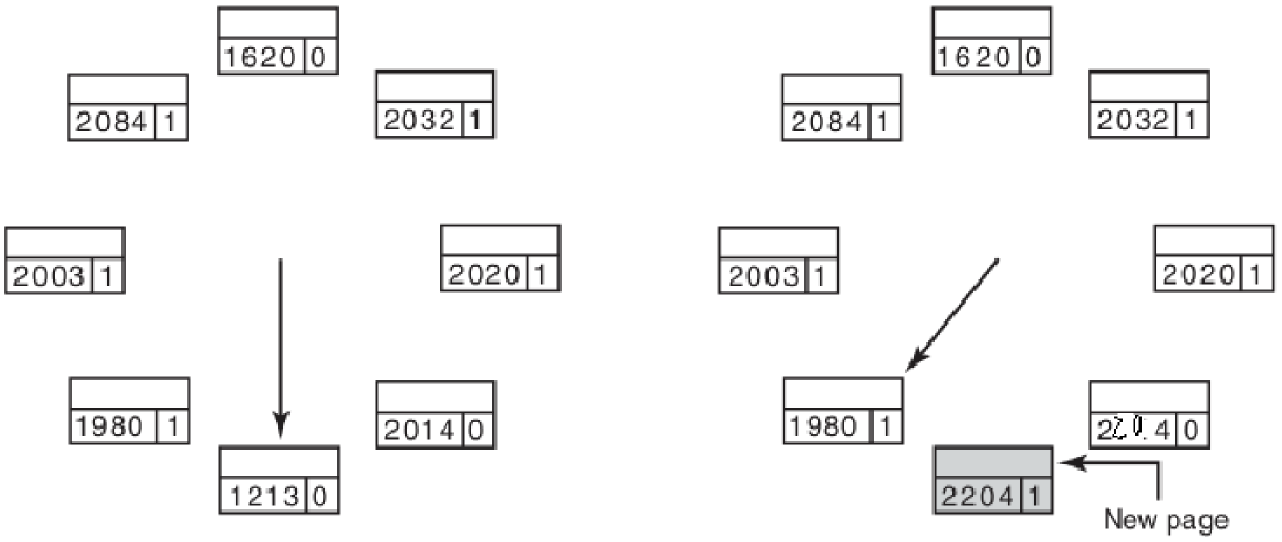
Note:Normally clock tick is used, since this is not possible, I increased the counter during memory hit times.

WSClock Algorithm

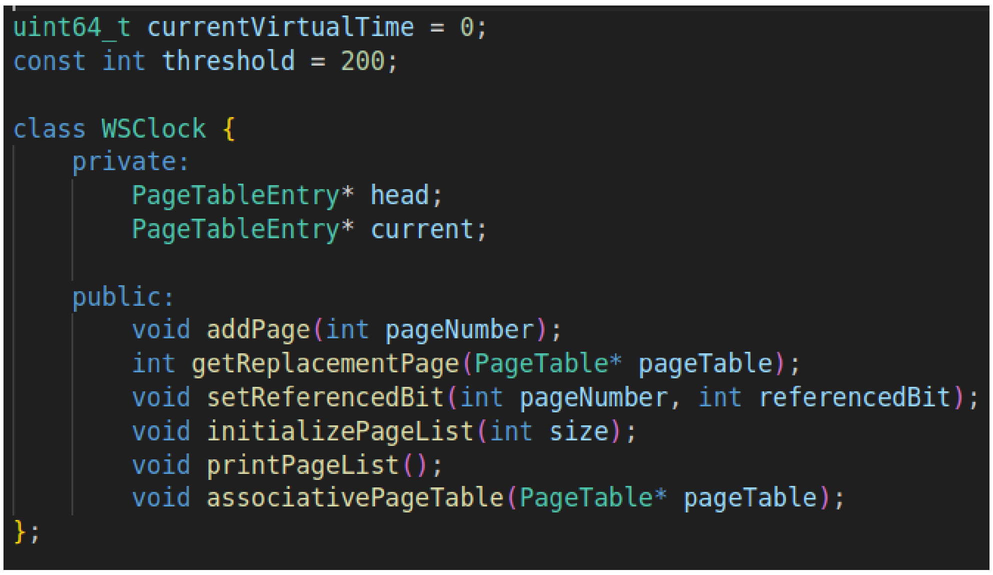


In the WSClock algorithm, there are bits checked in the page table, these are time of last use and Referenced ones.

A circular linked list structure is used in WSClock. Current Virtual Time and the counter in the LRU algorithm are similar. CVT increases with each clock tick. When Page fault occurs, the pages in the Working Set are cycled in order. The R bit of the page whose R bit is 1 is reset and the current virtual time value is set to the last time used value. And the next page is checked. If the R bit is zero, the following is checked;
current virtual time - last time used > treshold
this represents the old age of the page, if this is provided, the current virtual time is assigned to the last time used value of the page and the page is replaced.

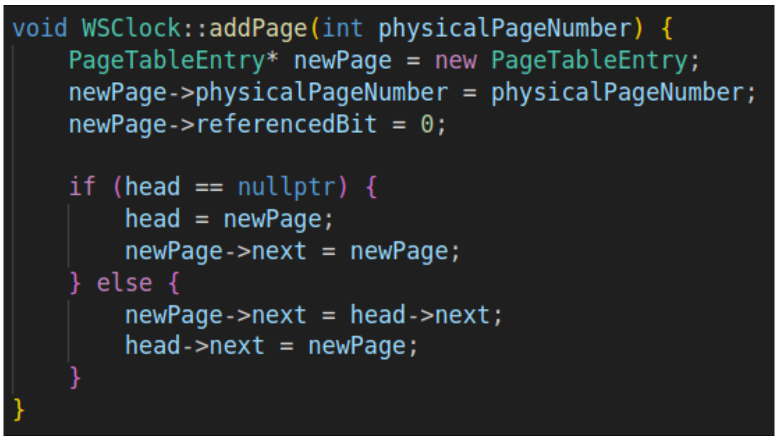


A simulated version of the situation I described above.



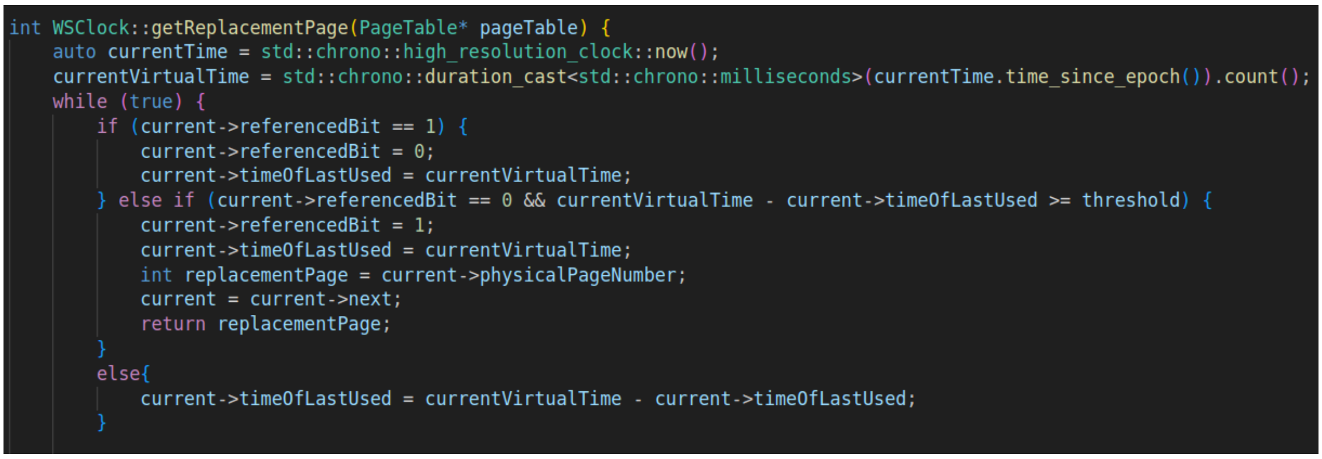
```
uint64_t currentVirtualTime = 0;
const int threshold = 200;

class WSClock {
    private:
        PageTableEntry* head;
        PageTableEntry* current;

    public:
        void addPage(int pageNumber);
        int getReplacementPage(PageTable* pageTable);
        void setReferencedBit(int pageNumber, int referencedBit);
        void initializePageList(int size);
        void printPageList();
        void associativePageTable(PageTable* pageTable);
};
```

I have the head and current pointers that I created for my circular linked list production. At the same time, I set a treshold value and kept a timestamp for the current virtual time.



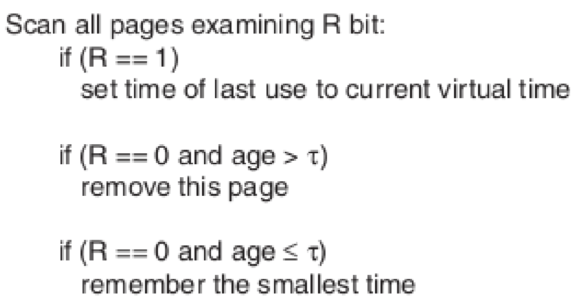
```
void WSClock::addPage(int physicalPageNumber) {
    PageTableEntry* newPage = new PageTableEntry;
    newPage->physicalPageNumber = physicalPageNumber;
    newPage->referencedBit = 0;

    if (head == nullptr) {
        head = newPage;
        newPage->next = newPage;
    } else {
        newPage->next = head->next;
        head->next = newPage;
    }
}
```

When adding a new page, I add it to the end of the circular linked list.



```
int WSClock::getReplacementPage(PageTable* pageTable) {
    auto currentTime = std::chrono::high_resolution_clock::now();
    currentVirtualTime = std::chrono::duration_cast<std::chrono::milliseconds>(currentTime.time_since_epoch()).count();
    while (true) {
        if (current->referencedBit == 1) {
            current->referencedBit = 0;
            current->timeOfLastUsed = currentVirtualTime;
        } else if (current->referencedBit == 0 && currentVirtualTime - current->timeOfLastUsed >= threshold) {
            current->referencedBit = 1;
            current->timeOfLastUsed = currentVirtualTime;
            int replacementPage = current->physicalPageNumber;
            current = current->next;
            return replacementPage;
        }
        else{
            current->timeOfLastUsed = currentVirtualTime - current->timeOfLastUsed;
        }
```

While doing replacement, I check the conditions that the WorkingSet algorithm theoretically wants.

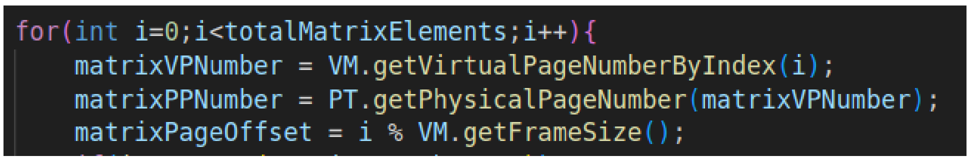


```
Scan all pages examining R bit:
    if (R == 1)
      set time of last use to current virtual time

    if (R == 0 and age > τ)
      remove this page

    if (R == 0 and age ≤ τ)
      remember the smallest time
```

I check the R bit in the page indicated by the current pointer in the circular linked list, if it is 1, I set it to 0 and update the lastOfUsed value.
If the R bit is 0 and your page is theoretically old, I set the R bit of that page to 1, update lastOfUsed, and return the page number and start the replace operation.
If the R bit is 0 hr but the page is not old, I set the minimum value to the lastOfUSed value.

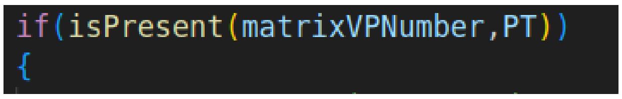
Physical memory access with GET/SET methods

I will explain the parts of reaching the values starting from the Matrix-Vector product.



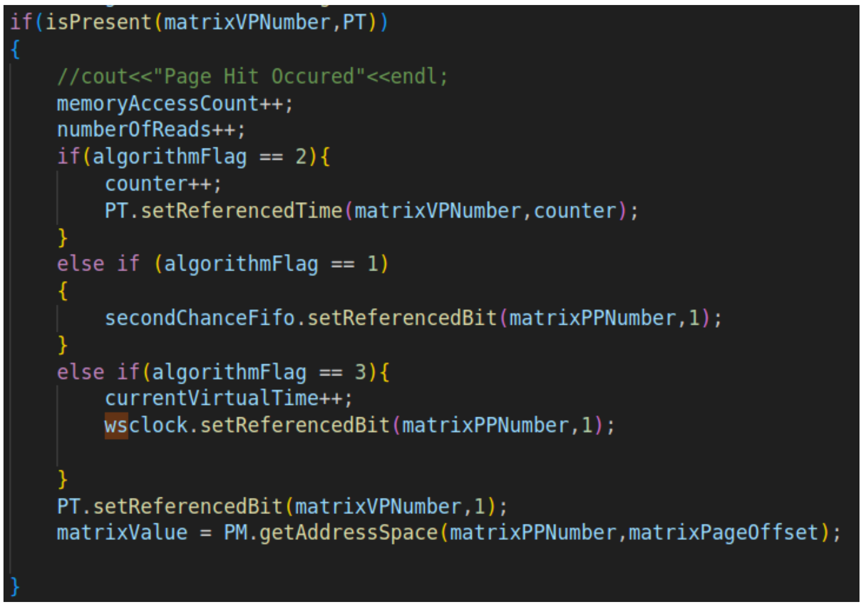
```
for(int i=0;i<totalMatrixElements;i++){
    matrixVPNumber = VM.getVirtualPageNumberByIndex(i);
    matrixPPNumber = PT.getPhysicalPageNumber(matrixVPNumber);
    matrixPageOffset = i % VM.getFrameSize();
```

Priorities, I perform for operations as much as the size of the total matrix elements in memory.
Matrix i. index of virtual memory i. is in the index.
Using VM.getVirtualPageNumberByIndex(i) returns me which virtual frame the index is in.
PT.getPhysicalPageNumber(matrixVPNumber); then I go to the index in the page table with the virtual frame number I obtained and return the corresponding Physical page number there.
matrixPageOffset = i % VM.getFrameSize(); The reason for this use is that there is a 4kb array on each page. With this process, we find which index of that array I want to access.



```
if(isPresent(matrixVPNumber,PT))
{
```

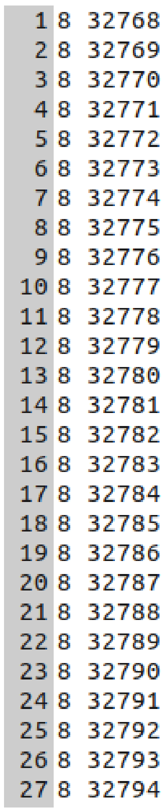
After obtaining this data, I check the page table again with the VP number I have, to see if the place indicated by that virtual address is in memory.



```
if(isPresent(matrixVPNumber,PT))
{
    //cout<<"Page Hit Occured"<<endl;
    memoryAccessCount++;
    numberOfReads++;
    if(algorithmFlag == 2){
        counter++;
        PT.setReferencedTime(matrixVPNumber,counter);
    }
    else if (algorithmFlag == 1)
    {
        secondChanceFifo.setReferencedBit(matrixPPNumber,1);
    }
    else if(algorithmFlag == 3){
        currentVirtualTime++;
        wsclock.setReferencedBit(matrixPPNumber,1);

    }
    PT.setReferencedBit(matrixVPNumber,1);
    matrixValue = PM.getAddressSpace(matrixPPNumber,matrixPageOffset);

}
```

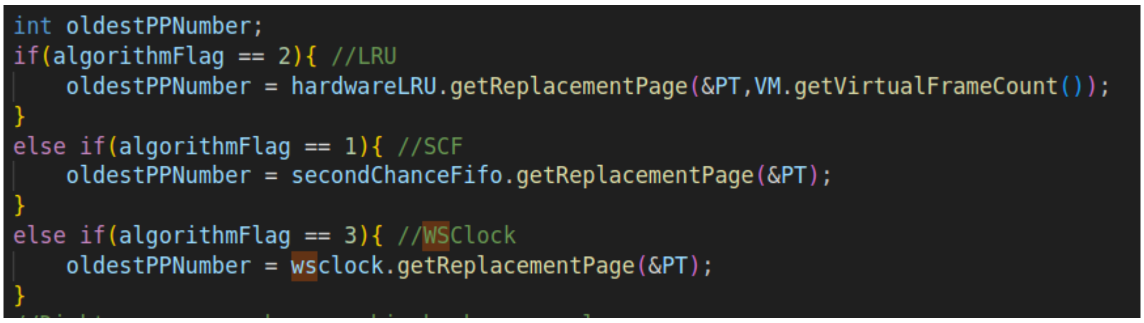
If the present bit is 1, it is in memory. So I do the above operations. If I am using the Hnagi algorithm, I update the necessary bit about that algorithm.
Finally PM.getAddressSpace(matrixPPNumber,matrixPageOffset);
With this operation, I reach the value in Physical Memory. I find which physical frame to go with matrixPPNumber. I find which index it corresponds to in the frame, which is also found with matrixPageOffset, and I pull the data.
If the present bit is 0, I understand that data is on disk.

Backing Store



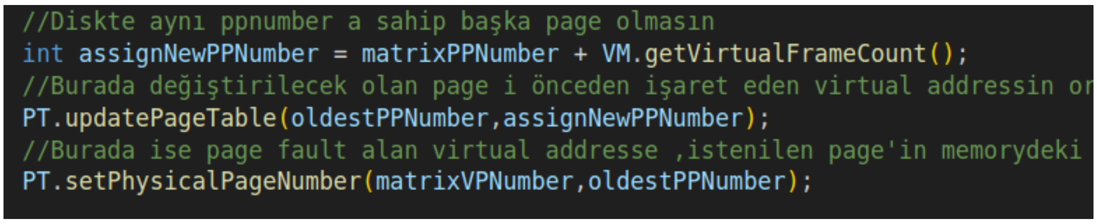
```
8 32768
8 32769
8 32770
8 32771
8 32772
8 32773
8 32774
8 32775
8 32776
8 32777
8 32778
8 32779
8 32780
8 32781
8 32782
8 32783
8 32784
8 32785
8 32786
8 32787
8 32788
8 32789
8 32790
8 32791
8 32792
8 32793
8 32794
```

This is how I record the values to fit in the physical memory to the disk. The number 8 on the left is the numbers given for the special representation of each page on the disk. Virtual memory pages that do not show physical memory show these areas on the disk. These values are recorded in the page table.



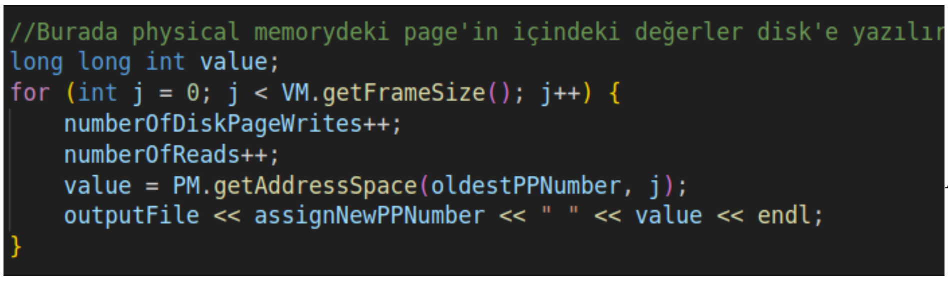
```
int oldestPPNumber;
if(algorithmFlag == 2){ //LRU
    oldestPPNumber = hardwareLRU.getReplacementPage(&PT,VM.getVirtualFrameCount());
}
else if(algorithmFlag == 1){ //SCF
    oldestPPNumber = secondChanceFifo.getReplacementPage(&PT);
}
else if(algorithmFlag == 3){ //WSClock
    oldestPPNumber = wsclock.getReplacementPage(&PT);
}
```

Here I find which page should be replaced from physical memory with the desired algorithm.



```
//Diskte aynı ppnumber a sahip başka page olmasın
int assignNewPPNumber = matrixPPNumber + VM.getVirtualFrameCount();
//Burada değiştirilecek olan page i önceden işaret eden virtual addressin or
PT.updatePageTable(oldestPPNumber,assignNewPPNumber);
//Burada ise page fault alan virtual addresse ,istenilen page'in memorydeki
PT.setPhysicalPageNumber(matrixVPNumber,oldestPPNumber);
```

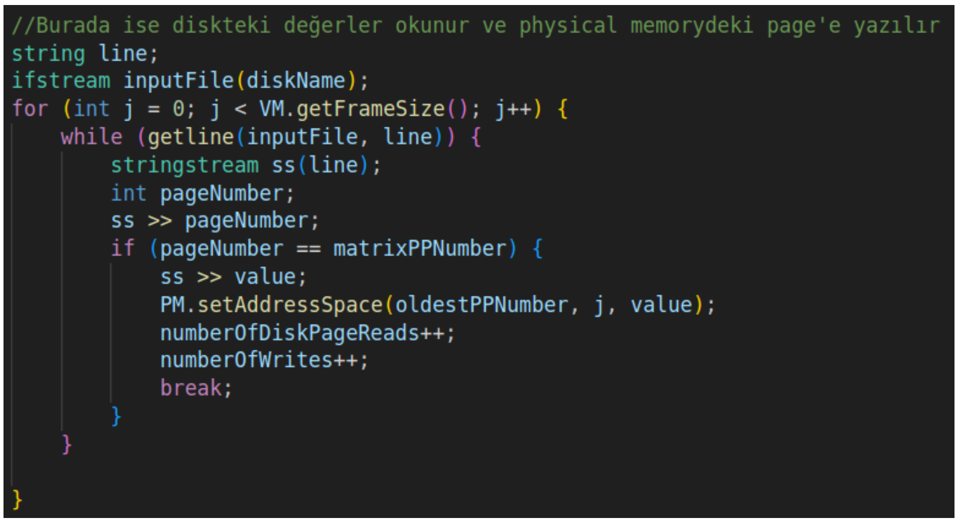
Then, by summing the physical address number that the virtual address points to on the disk and the frame count, I generate a temporary physical page number to write to the disk.
That is, the virtual page that previously pointed to the disk now points to the physical memory, and the virtual page that points to the physical memory is pointing to the disk.



```
//Burada physical memorydeki page'in içindeki değerler disk'e yazılır
long long int value;
for (int j = 0; j < VM.getFrameSize(); j++) {
    numberOfDiskPageWrites++;
    numberOfReads++;
    value = PM.getAddressSpace(oldestPPNumber, j);
    outputFile << assignNewPPNumber << " " << value << endl;
}
```

A

After the above page table updates are completed, I write the old page in the physical memory to the disk.

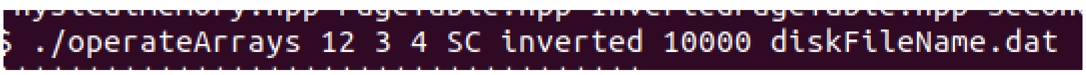


```
//Burada ise diskteki değerler okunur ve physical memorydeki page'e yazılır
string line;
ifstream inputFile(diskName);
for (int j = 0; j < VM.getFrameSize(); j++) {
    while (getline(inputFile, line)) {
        stringstream ss(line);
        int pageNumber;
        ss >> pageNumber;
        if (pageNumber == matrixPPNumber) {
            ss >> value;
            PM.setAddressSpace(oldestPPNumber, j, value);
            numberOfDiskPageReads++;
            numberOfWrites++;
            break;
        }
    }

}
```

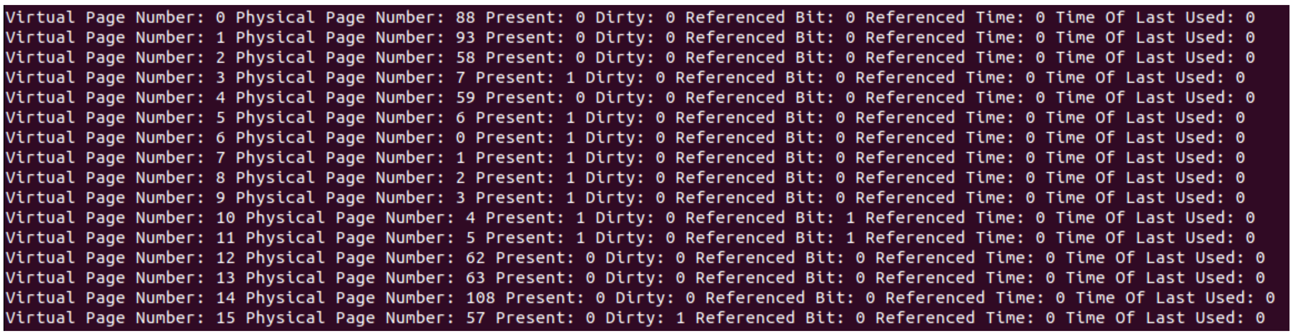
For the page I want from the disk, I read the disk one by one and save each value I read to the memory.
then matrixValue = PM.getAddressSpace(oldestPPNumber,matrixPageOffset);
With the process, I can meet the value that the virtual memory just wanted from the memory.

TEST CASES

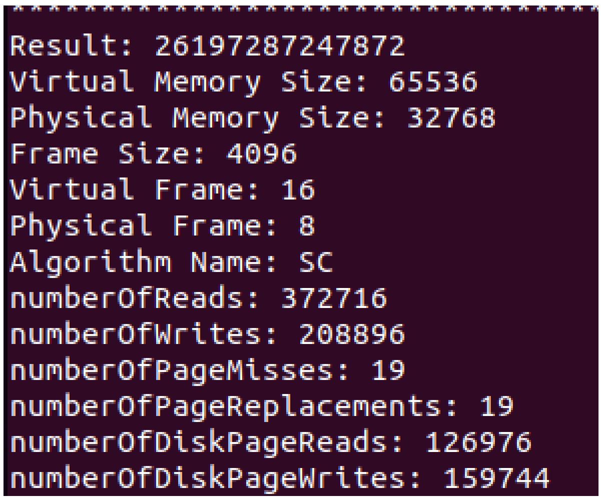
Matrix-Vector multiplication and then summing the multiplication result:



```
$ ./operateArrays 12 3 4 SC inverted 10000 diskFileName.dat
```

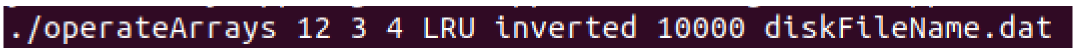


```
Virtual Page Number: 0 Physical Page Number: 88 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 1 Physical Page Number: 93 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 2 Physical Page Number: 58 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 3 Physical Page Number: 7 Present: 1 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 4 Physical Page Number: 59 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 5 Physical Page Number: 6 Present: 1 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 6 Physical Page Number: 0 Present: 1 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 7 Physical Page Number: 1 Present: 1 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 8 Physical Page Number: 2 Present: 1 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 9 Physical Page Number: 3 Present: 1 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 10 Physical Page Number: 4 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 11 Physical Page Number: 5 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 12 Physical Page Number: 62 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 13 Physical Page Number: 63 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 14 Physical Page Number: 108 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 15 Physical Page Number: 57 Present: 0 Dirty: 1 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
```

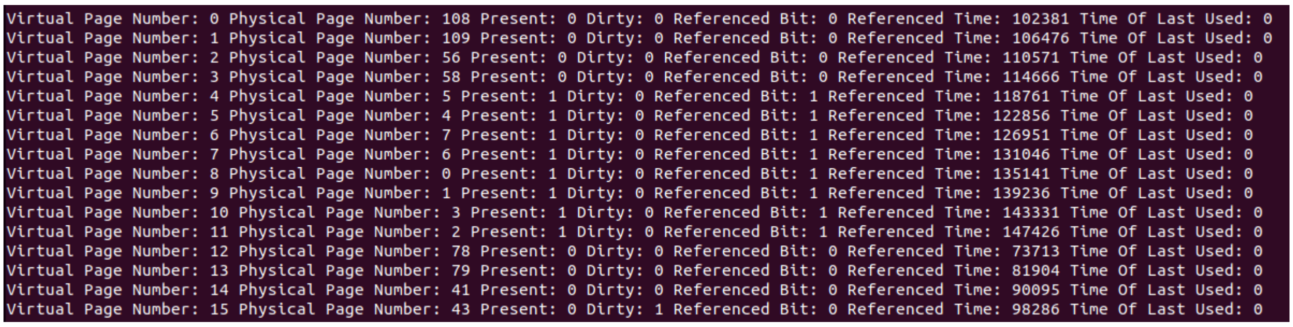


```
***********************************
Result: 26197287247872
Virtual Memory Size: 65536
Physical Memory Size: 32768
Frame Size: 4096
Virtual Frame: 16
Physical Frame: 8
Algorithm Name: SC
numberOfReads: 372716
numberOfWrites: 208896
numberOfPageMisses: 19
numberOfPageReplacements: 19
numberOfDiskPageReads: 126976
numberOfDiskPageWrites: 159744
```

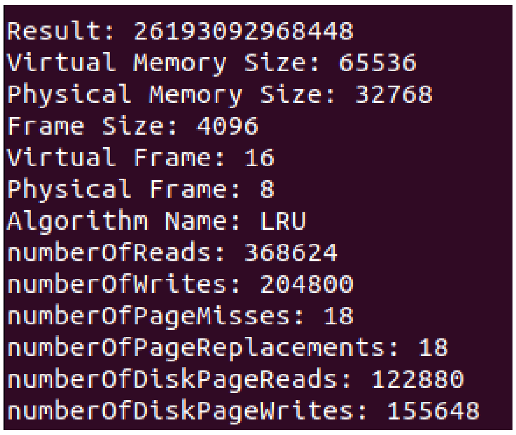
****



```
./operateArrays 12 3 4 LRU inverted 10000 diskFileName.dat
```

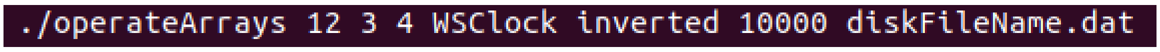


```
Virtual Page Number: 0 Physical Page Number: 108 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 102381 Time Of Last Used: 0
Virtual Page Number: 1 Physical Page Number: 109 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 106476 Time Of Last Used: 0
Virtual Page Number: 2 Physical Page Number: 56 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 110571 Time Of Last Used: 0
Virtual Page Number: 3 Physical Page Number: 58 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 114666 Time Of Last Used: 0
Virtual Page Number: 4 Physical Page Number: 5 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 118761 Time Of Last Used: 0
Virtual Page Number: 5 Physical Page Number: 4 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 122856 Time Of Last Used: 0
Virtual Page Number: 6 Physical Page Number: 7 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 126951 Time Of Last Used: 0
Virtual Page Number: 7 Physical Page Number: 6 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 131046 Time Of Last Used: 0
Virtual Page Number: 8 Physical Page Number: 0 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 135141 Time Of Last Used: 0
Virtual Page Number: 9 Physical Page Number: 1 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 139236 Time Of Last Used: 0
Virtual Page Number: 10 Physical Page Number: 3 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 143331 Time Of Last Used: 0
Virtual Page Number: 11 Physical Page Number: 2 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 147426 Time Of Last Used: 0
Virtual Page Number: 12 Physical Page Number: 78 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 73713 Time Of Last Used: 0
Virtual Page Number: 13 Physical Page Number: 79 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 81904 Time Of Last Used: 0
Virtual Page Number: 14 Physical Page Number: 41 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 90095 Time Of Last Used: 0
Virtual Page Number: 15 Physical Page Number: 43 Present: 0 Dirty: 1 Referenced Bit: 0 Referenced Time: 98286 Time Of Last Used: 0
```

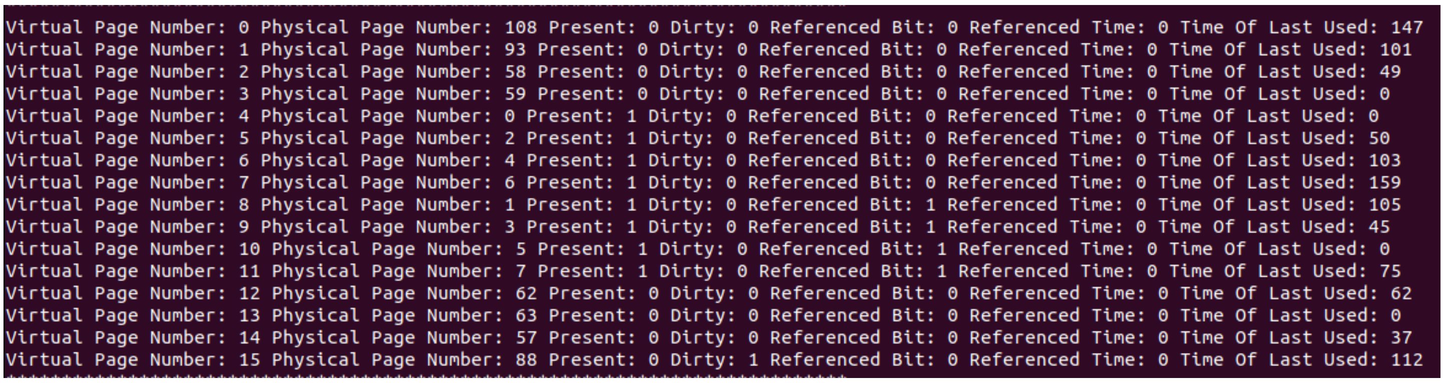


```
Result: 26193092968448
Virtual Memory Size: 65536
Physical Memory Size: 32768
Frame Size: 4096
Virtual Frame: 16
Physical Frame: 8
Algorithm Name: LRU
numberOfReads: 368624
numberOfWrites: 204800
numberOfPageMisses: 18
numberOfPageReplacements: 18
numberOfDiskPageReads: 122880
numberOfDiskPageWrites: 155648
```

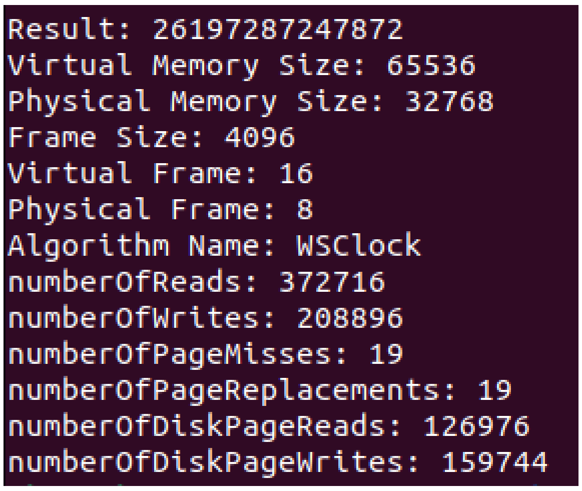
****



```
./operateArrays 12 3 4 WSClock inverted 10000 diskFileName.dat
```



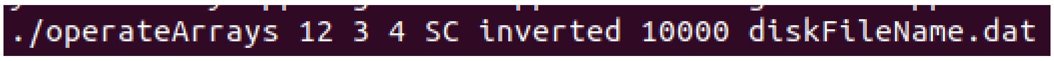
```
Virtual Page Number: 0 Physical Page Number: 108 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 147
Virtual Page Number: 1 Physical Page Number: 93 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 101
Virtual Page Number: 2 Physical Page Number: 58 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 49
Virtual Page Number: 3 Physical Page Number: 59 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 4 Physical Page Number: 0 Present: 1 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 5 Physical Page Number: 2 Present: 1 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 50
Virtual Page Number: 6 Physical Page Number: 4 Present: 1 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 103
Virtual Page Number: 7 Physical Page Number: 6 Present: 1 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 159
Virtual Page Number: 8 Physical Page Number: 1 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 105
Virtual Page Number: 9 Physical Page Number: 3 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 45
Virtual Page Number: 10 Physical Page Number: 5 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 11 Physical Page Number: 7 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 75
Virtual Page Number: 12 Physical Page Number: 62 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 62
Virtual Page Number: 13 Physical Page Number: 63 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 14 Physical Page Number: 57 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 37
Virtual Page Number: 15 Physical Page Number: 88 Present: 0 Dirty: 1 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 112
```



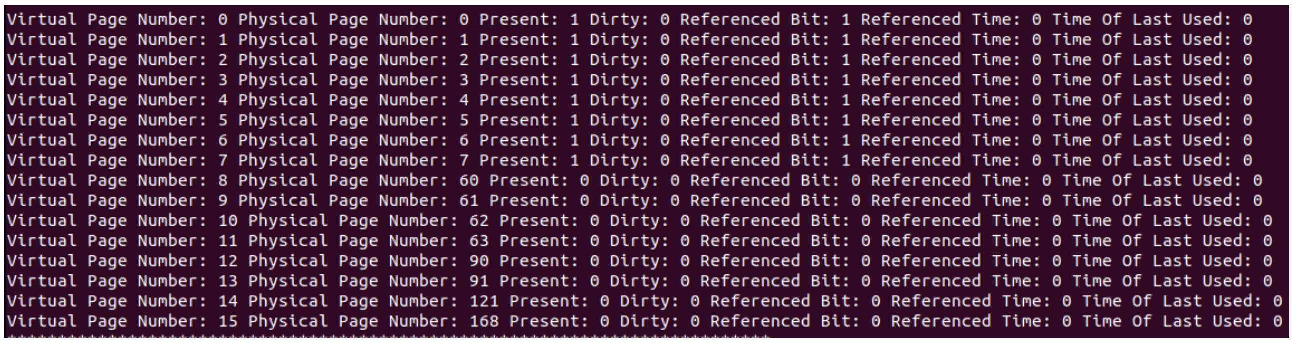
```
Result: 26197287247872
Virtual Memory Size: 65536
Physical Memory Size: 32768
Frame Size: 4096
Virtual Frame: 16
Physical Frame: 8
Algorithm Name: WSClock
numberOfReads: 372716
numberOfWrites: 208896
numberOfPageMisses: 19
numberOfPageReplacements: 19
numberOfDiskPageReads: 126976
numberOfDiskPageWrites: 159744
```

****

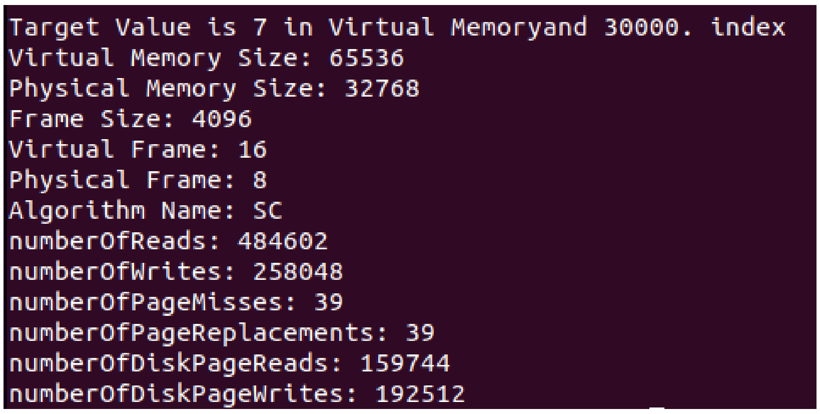
Vector * Vector^T and Linear Search



```
./operateArrays 12 3 4 SC inverted 10000 diskFileName.dat
```

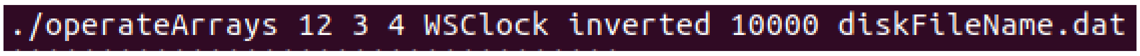


```
Virtual Page Number: 0 Physical Page Number: 0 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 1 Physical Page Number: 1 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 2 Physical Page Number: 2 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 3 Physical Page Number: 3 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 4 Physical Page Number: 4 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 5 Physical Page Number: 5 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 6 Physical Page Number: 6 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 7 Physical Page Number: 7 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 8 Physical Page Number: 60 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 9 Physical Page Number: 61 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 10 Physical Page Number: 62 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 11 Physical Page Number: 63 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 12 Physical Page Number: 90 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 13 Physical Page Number: 91 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 14 Physical Page Number: 121 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 15 Physical Page Number: 168 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
```

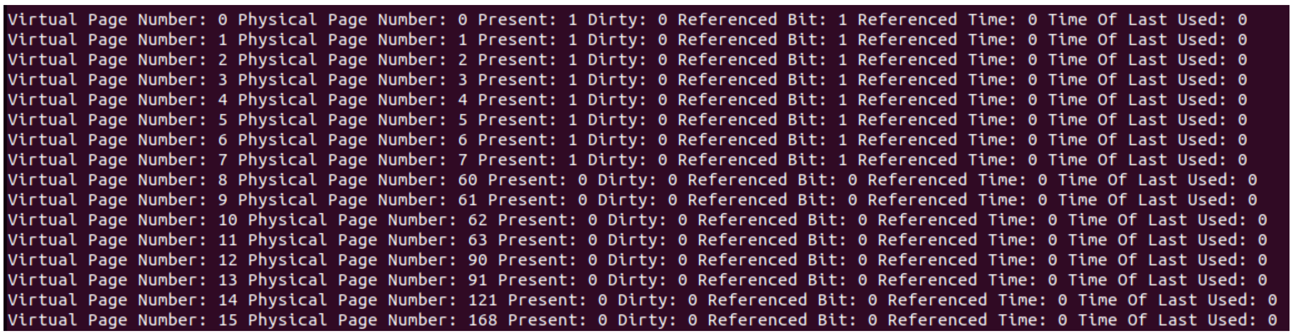


```
Target Value is 7 in Virtual Memoryand 30000. index
Virtual Memory Size: 65536
Physical Memory Size: 32768
Frame Size: 4096
Virtual Frame: 16
Physical Frame: 8
Algorithm Name: SC
numberOfReads: 484602
numberOfWrites: 258048
numberOfPageMisses: 39
numberOfPageReplacements: 39
numberOfDiskPageReads: 159744
numberOfDiskPageWrites: 192512
```

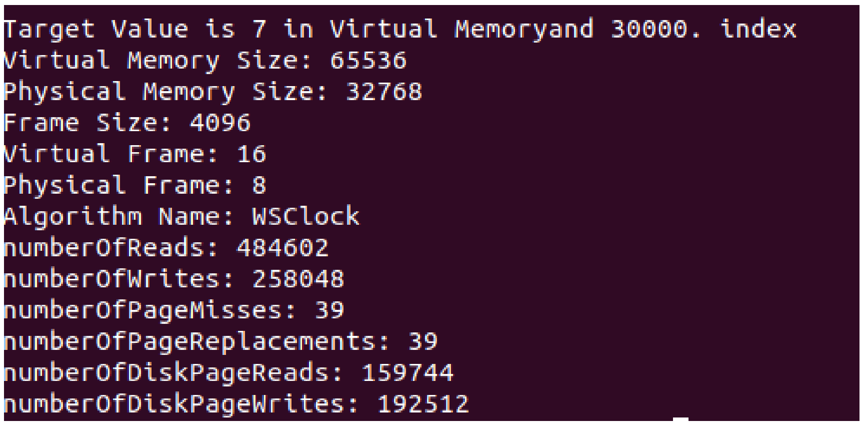
****



```
./operateArrays 12 3 4 WSClock inverted 10000 diskFileName.dat
```

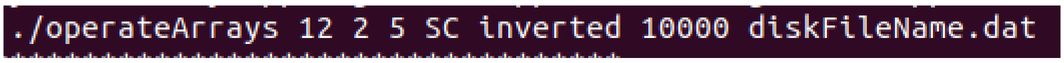


```
Virtual Page Number: 0 Physical Page Number: 0 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 1 Physical Page Number: 1 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 2 Physical Page Number: 2 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 3 Physical Page Number: 3 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 4 Physical Page Number: 4 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 5 Physical Page Number: 5 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 6 Physical Page Number: 6 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 7 Physical Page Number: 7 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 8 Physical Page Number: 60 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 9 Physical Page Number: 61 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 10 Physical Page Number: 62 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 11 Physical Page Number: 63 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 12 Physical Page Number: 90 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 13 Physical Page Number: 91 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 14 Physical Page Number: 121 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 15 Physical Page Number: 168 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
```

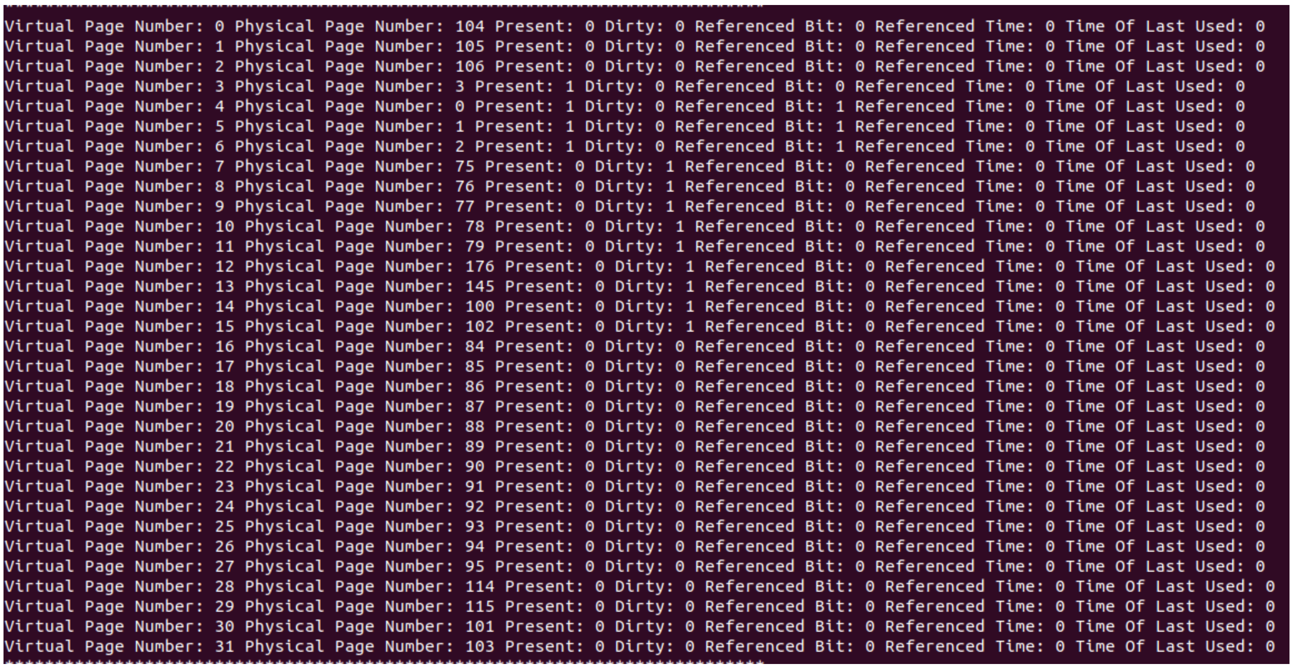


```
Target Value is 7 in Virtual Memoryand 30000. index
Virtual Memory Size: 65536
Physical Memory Size: 32768
Frame Size: 4096
Virtual Frame: 16
Physical Frame: 8
Algorithm Name: WSClock
numberOfReads: 484602
numberOfWrites: 258048
numberOfPageMisses: 39
numberOfPageReplacements: 39
numberOfDiskPageReads: 159744
numberOfDiskPageWrites: 192512
```

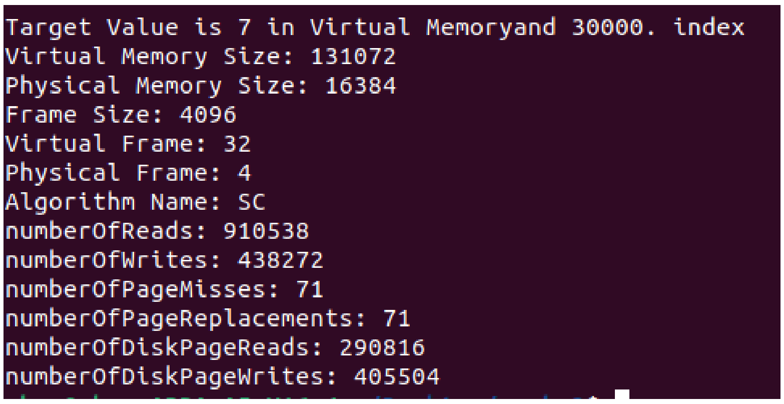
PART 3



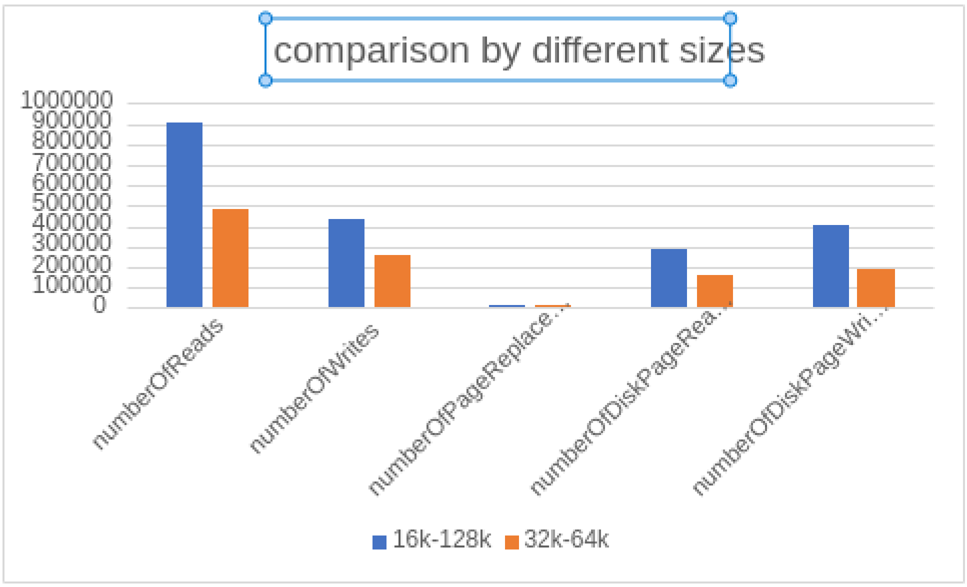
```
./operateArrays 12 2 5 SC inverted 10000 diskFileName.dat
```



```
Virtual Page Number: 0 Physical Page Number: 104 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 1 Physical Page Number: 105 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 2 Physical Page Number: 106 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 3 Physical Page Number: 3 Present: 1 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 4 Physical Page Number: 0 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 5 Physical Page Number: 1 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 6 Physical Page Number: 2 Present: 1 Dirty: 0 Referenced Bit: 1 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 7 Physical Page Number: 75 Present: 0 Dirty: 1 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 8 Physical Page Number: 76 Present: 0 Dirty: 1 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 9 Physical Page Number: 77 Present: 0 Dirty: 1 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 10 Physical Page Number: 78 Present: 0 Dirty: 1 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 11 Physical Page Number: 79 Present: 0 Dirty: 1 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 12 Physical Page Number: 176 Present: 0 Dirty: 1 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 13 Physical Page Number: 145 Present: 0 Dirty: 1 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 14 Physical Page Number: 100 Present: 0 Dirty: 1 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 15 Physical Page Number: 102 Present: 0 Dirty: 1 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 16 Physical Page Number: 84 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 17 Physical Page Number: 85 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 18 Physical Page Number: 86 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 19 Physical Page Number: 87 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 20 Physical Page Number: 88 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 21 Physical Page Number: 89 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 22 Physical Page Number: 90 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 23 Physical Page Number: 91 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 24 Physical Page Number: 92 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 25 Physical Page Number: 93 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 26 Physical Page Number: 94 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 27 Physical Page Number: 95 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 28 Physical Page Number: 114 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 29 Physical Page Number: 115 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 30 Physical Page Number: 101 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
Virtual Page Number: 31 Physical Page Number: 103 Present: 0 Dirty: 0 Referenced Bit: 0 Referenced Time: 0 Time Of Last Used: 0
```



```
Target Value is 7 in Virtual Memoryand 30000. index
Virtual Memory Size: 131072
Physical Memory Size: 16384
Frame Size: 4096
Virtual Frame: 32
Physical Frame: 4
Algorithm Name: SC
numberOfReads: 910538
numberOfWrites: 438272
numberOfPageMisses: 71
numberOfPageReplacements: 71
numberOfDiskPageReads: 290816
numberOfDiskPageWrites: 405504
```


comparison by different sizes
1000000
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
numberOfReads
numberOfWrites
numberOfPageReplace...
numberOfDiskPageRea...
numberOfDiskPageWri...
16k-128k
32k-64k