HW2

Tutor:İlhan Aytutuldu
Student Name:Ozan Argıt Önçeken
Student Number:1901042259

Introduction

getPage() function gets the page and works like:

The VirtualAddressEntity class represents a entity for the virtual addresses used in virtual memory management.

1. When an instance of VirtualAddressEntity is created, an array is initialized with a specific size. This size represents the number of addresses in the virtual memory and is determined by the VIRTUAL_ADDRESS constant.
2. The array is then filled with randomly generated virtual addresses. The addresses are assigned in a random order and the sequence does not matter. For example, if VIRTUAL_ADDRESS is set to 16, the array may look like this:

   [2, 8, 15, 4, 11, 1, 6, 12, 3, 10, 5, 0, 14, 7, 9, 13]
3. This array represents the virtual addresses in the virtual memory. Each element represents a virtual address.

Working of the getPage() function:

1. When the getPage() function is called, it receives a normal_address as input, which is the address that the processor wants to access.
2. The first step is to calculate the corresponding page number for the normal_address. This calculation is done by dividing the normal_address by the page size (PAGE_SIZE). For example, if normal_address = 14 and the page size is 4, the page number would be 14 / 4 = 3.
3. The page number is then used to look up the virtual address associated with it in the page table (pageTable). The page table stores a mapping of each page number to a virtual address, which indicates the physical address of the page in the physical memory.
4. The obtained virtual_address is checked in the virtual frame table (virtual_frame_table). The virtual frame table maps each virtual address to a physical address in the physical memory.
5. The physical address (physical_address) obtained from the virtual frame table represents the corresponding page (PageEntity) in the physical memory.
6. The getPage() function returns the retrieved page as the result. This allows the processor to access the data at the requested normal_address.

For Second Chance Algorithm:

When the getPage() function is called, it takes the normal_address parameter as input. The page number is calculated from the normal address. This is obtained by dividing the normal_address by the page size (PAGE_SIZE). The page number is then searched in the page table (pageTable). If the page number is found in the page table, the corresponding virtual address (virtual_address) is obtained. The obtained virtual address is searched in the virtual frame table (virtual_frame_table), and the corresponding physical address (physical_address) is obtained. Using the physical address, the relevant page (PageEntity) in the physical memory is obtained. The

referenced bit of the page is checked. If the referenced bit is 1, the referenced bit in the page table is cleared, and the page is added to the end of the queue to wait for another round. If the referenced bit is 0, the page is directly returned and the operation is completed.

For LRU Algorithm:

1. The normal_address parameter is received.
2. The page number is calculated from the normal_address by dividing it by the page size (PAGE_SIZE).
3. The page number is checked in the page table (pageTable).
4. If the page number is found in the page table, the corresponding virtual address (virtual_address) is obtained.
5. The virtual frame table (virtual_frame_table) is checked using the virtual address to retrieve the corresponding physical address (physical_address).
6. The physical address is used to access the respective page (PageEntity) in the physical memory.
7. The accessed page is marked as recently used, indicating that it has been accessed most recently.
8. The LRU algorithm keeps track of the usage history of pages. When a page needs to be replaced, the page with the least recent access time is chosen for replacement.
9. If the accessed page was not recently used, it is considered as the most recently used page.
10. If the accessed page was recently used, it is updated as the most recently used page.
11. The LRU algorithm ensures that the page replaced is the one that has been accessed the least recently, reducing the likelihood of removing a frequently used page.

For WSClock Algorithm:

The WSClock (Working Set Clock) algorithm is a page replacement algorithm used in virtual memory management. It combines the concepts of the Clock algorithm and the Working Set model.

When the getPage() function is called using the WSClock algorithm, the following steps are performed:

1. The normal_address parameter is received.
2. The page number is calculated from the normal_address by dividing it by the page size (PAGE_SIZE).
3. The page number is checked in the page table (pageTable).
4. If the page number is found in the page table, the corresponding virtual address (virtual_address) is obtained.
5. The virtual frame table (virtual_frame_table) is checked using the virtual address to retrieve the corresponding physical address (physical_address).
6. The physical address is used to access the respective page (PageEntity) in the physical memory.
7. The WSClock algorithm maintains a clock hand that points to a page in the physical memory.
8. The clock hand is initially set to the first page in the physical memory.
9. When a page needs to be replaced, the clock hand is moved forward, examining each page in a circular manner.

10. Each page is checked for the referenced bit: a. If the referenced bit is 0, it means the page has not been accessed recently.
    - If the modified bit is 0, the page is chosen for replacement.
    - If the modified bit is 1, the page is marked as dirty and the modified bit is set to 0. The clock hand moves to the next page. b. If the referenced bit is 1, it means the page has been accessed recently.
    - The referenced bit is set to 0, indicating that the page has been given a second chance.
    - The clock hand moves to the next page.
11. The clock hand continues moving until a suitable page for replacement is found.
12. The chosen page is replaced with the new page to be brought into the physical memory.
13. If the chosen page is dirty (modified bit is 1), it is written back to the disk before replacement.
14. The new page is loaded into the physical memory, and its referenced bit and modified bit are set accordingly.
15. The clock hand is updated to the next page for future replacements.

FUNCTIONS

Main:

getPage: This function calls the getPage2 of VirtualMemoryManagement. (it is getpage2 because at first I wrote getPage but it did not worked and I wrote the getPage2 after that and it worked and I used it.)

Main: main takes inputs to getpage function until user sends ctrl+C signal.

VirtualMemoryManagement:

disk: An array of PageEntity objects representing the disk memory.
physicalMemory: An array of PageEntity objects representing the physical memory.
virtualAddress: An array of VirtualAddressEntity objects representing the virtual addresses.
pageTable: An unordered map that maps page numbers to virtual addresses.
virtual_frame_table: An unordered map that maps virtual addresses to physical frames.
physical_mem_queue: A queue that keeps track of the physical memory allocation order.

Private member functions:

connectVirtToPhysicalMem(int virt_addr, int physical_addr): Connects a virtual address to a physical address.
initializeDisk(): Initializes the disk memory with PageEntity objects.
initializePhysicalMemory(): Initializes the physical memory with PageEntity objects.
setupVirtualAdress(): Sets up the virtual addresses.
setPageTableEntry(int page_number, int virtual_address): Sets an entry in the page table.
getPageTableEntry(int page_number): Retrieves the virtual address from the page table.
getPageFrameToVirtualAddress(int page_number): Retrieves the physical frame corresponding to a page number.
isPhysicalMemoryFull(): Checks if the physical memory is full.
find_empty_physical_mem(): Finds an empty index in the physical memory.
getPageFromDiskToPhyMem(int disk_index, int ram_index): Moves a page from the disk to the physical memory.
updateOldVirtualAddress(int old_page_numb): Updates a virtual address when a page replacement occurs.
updateNewVirtualAddress(int old_page_numb, int physical_address): Updates a new virtual address after page replacement.
updateVirtualAddress(int disk_index, int ram_index, int virtual_address): Updates a virtual address after moving from disk to physical memory.
updateInitVirtualAddress(int disk_index, int ram_index, int virtual_address): Initializes a virtual address when it is first allocated.
fifoWay(int disk_index): Performs the First-In-First-Out (FIFO) page replacement algorithm.
secondChanceAlgorithm(int page_number): Performs the Second Chance page replacement algorithm.

lruAlgorithm(int page_number): Performs the Least Recently Used (LRU) page replacement algorithm.
WSClockAlgorithm(int page_number): Performs the WSClock page replacement algorithm.
fifoWay2(int new_page_num): Performs the FIFO algorithm when a new page is allocated.
physicalMemtoDisk(int mem_index, int disk_index): Moves a page from physical memory to disk.
fillRandomVirtualAddress(): Fills the virtual addresses with random page numbers.
pageFault2(int page_number): Handles a page fault and returns the corresponding page from disk.

Public member functions:

VirtualMemoryManagement(): Default constructor of the VirtualMemoryManagement class.
VirtualMain(): The main function for the virtual memory management simulation.
pageInfo(int pageNumber): Prints the information of a specific page.
getPage2(int normal_address): Retrieves a page from the virtual memory given a normal address.
setPage(int normal_address, int value): Sets the value of a page in the virtual memory.
showSimulatedAdresses(int restriction = VIRTUAL_ADDRESS): Prints the simulated addresses up to a specified restriction.
initializeVirtualMemory(): Initializes the disk, physical memory, and virtual addresses before starting the simulation.

PageEntity Class

The PageEntity class represents a page entity within the virtual memory management system. This class serves as a representation of a page, storing and providing access to the information of a page in either the disk memory or the physical memory. Below is a detailed report describing the properties and functions of the PageEntity class:

Properties:

pageNumber (int): Represents the page number associated with the PageEntity. It uniquely identifies the page within the virtual memory system.

Functions:

getPageNumber(): Retrieves the page number of the PageEntity.
setPageNumber(int): Sets the page number of the PageEntity to the specified value.
getPageData(): Retrieves the data stored in the page. This could be the content of the page in the disk memory or the physical memory.
setPageData(): Sets the data of the page to the specified value.
isValid(): Checks if the page is valid or not. It determines whether the page is currently being used and contains valid data.
setValid(bool): Sets the validity status of the page. It marks the page as valid or invalid based on the specified boolean value.
isDirty(): Checks if the page has been modified or not. It indicates whether the page content in the physical memory is different from the content in the disk memory.

setDirty(bool): Sets the dirty status of the page. It marks the page as dirty or clean based on the specified boolean value.
getLastAccessTime(): Retrieves the timestamp of the last access to the page. It provides information about when the page was last accessed.
setLastAccessTime(int): Sets the last access timestamp of the page to the specified value. It updates the timestamp when the page is accessed.

VirtualAddressEntity Class Report

The VirtualAddressEntity class represents a virtual address entity within the virtual memory management system. This class is responsible for storing and managing the information related to a virtual address. Below is a detailed report describing the properties and functions of the VirtualAddressEntity class:

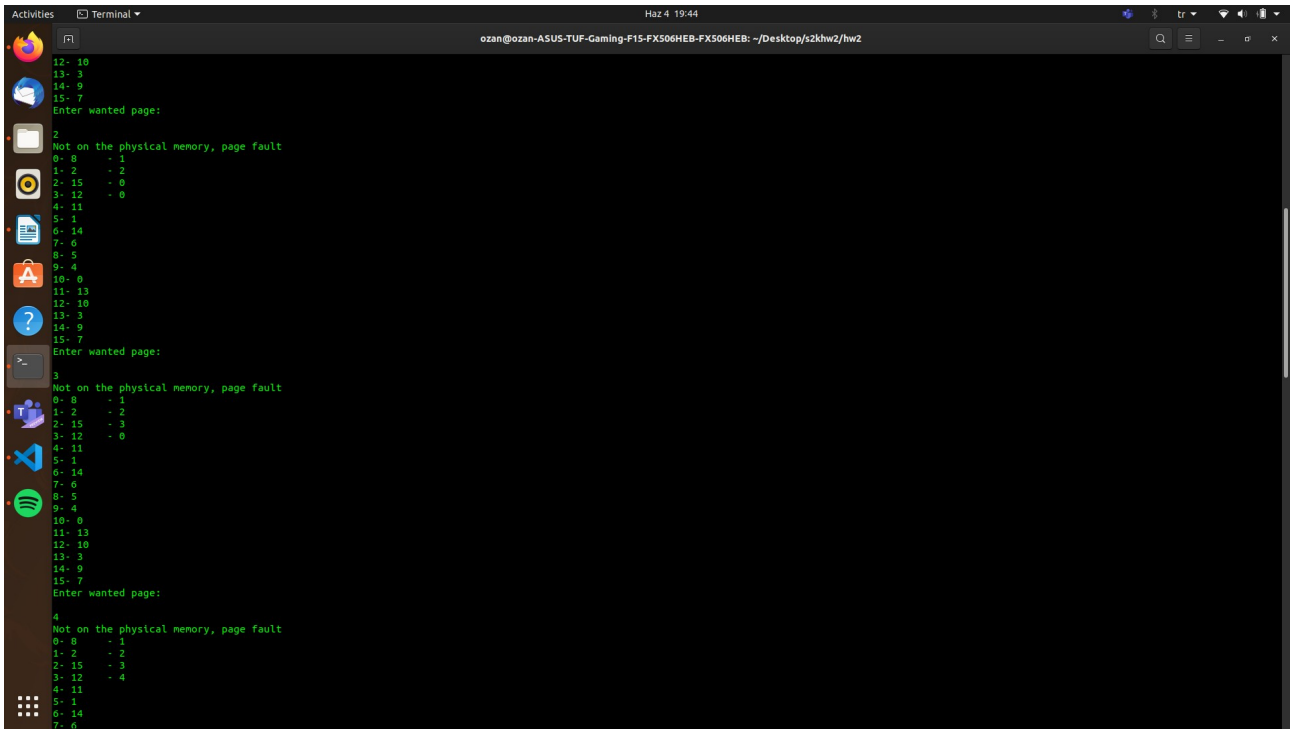
Properties:

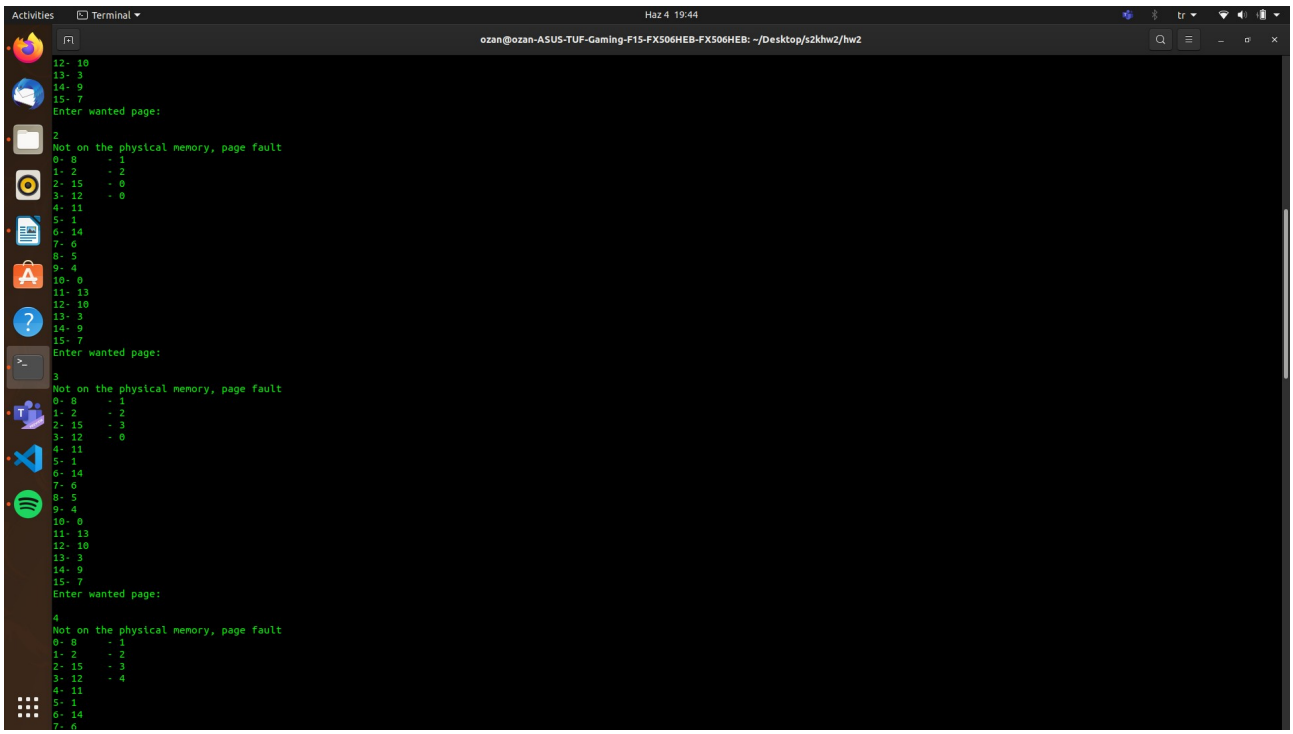
virtualAddress (int): Represents the virtual address value. It holds the address value that maps to a specific page within the virtual memory system.
pageEntity (PageEntity): Represents the associated PageEntity object for the virtual address. It provides a reference to the page entity that corresponds to the virtual address.

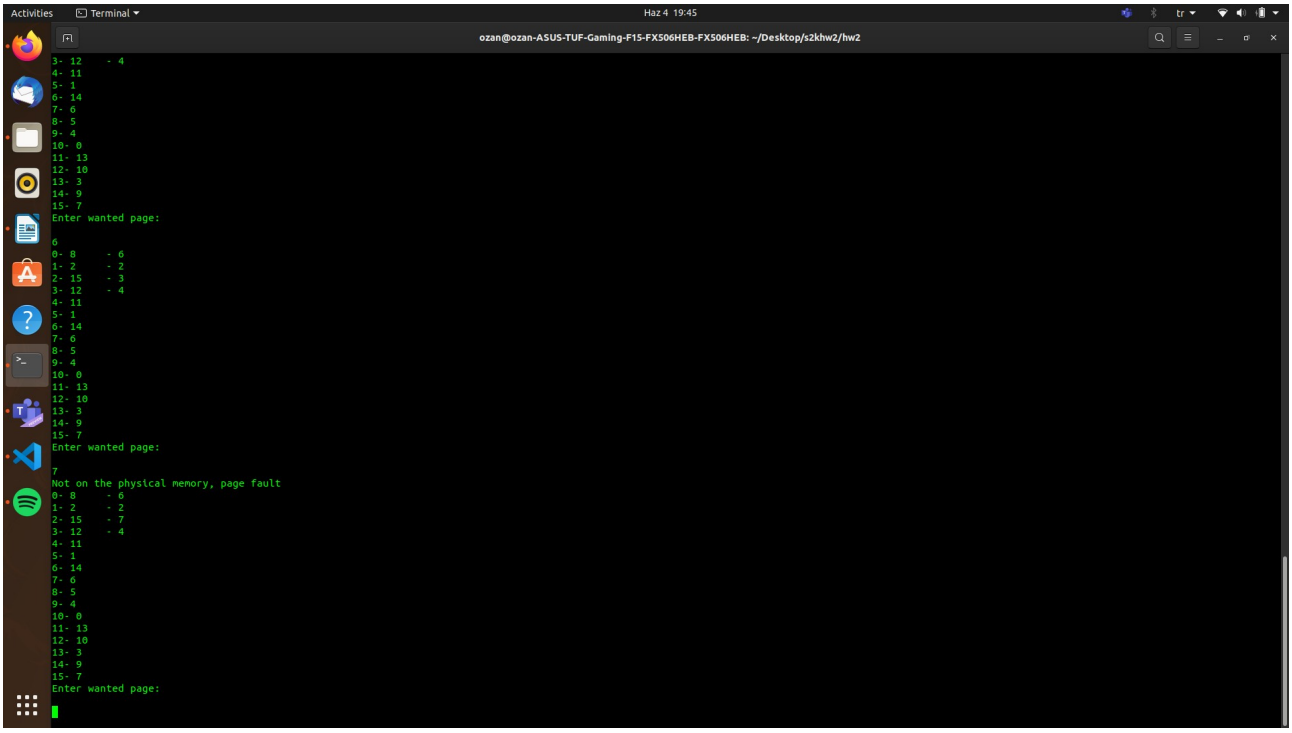
Functions:

getVirtualAddress(): Retrieves the virtual address value from the VirtualAddressEntity.
setVirtualAddress(int): Sets the virtual address value of the VirtualAddressEntity to the specified value.
getPageEntity(): Retrieves the associated PageEntity object for the virtual address.
setPageEntity(PageEntity): Sets the associated PageEntity object of the VirtualAddressEntity to the specified value.

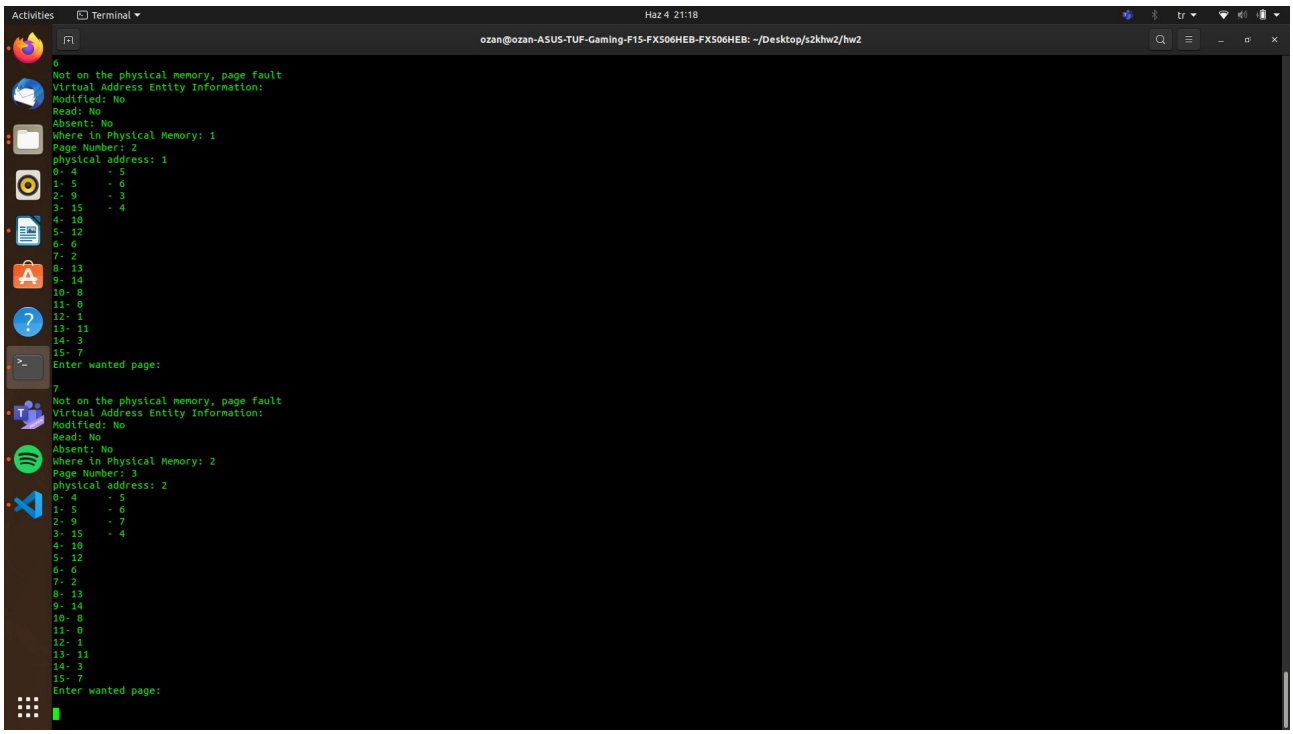
TESTS


Activities
Terminal
Haz 4 19:44
ozan@ozan-ASUS-TUF-Gaming-F15-FX506HEB-FX506HEB: ~/Desktop/s2khw2/hw2
12- 10
13- 3
14- 9
15- 7
Enter wanted page:
2
Not on the physical memory, page fault
0- 8 - 1
1- 2 - 2
2- 15 - 0
3- 12 - 0
4- 11
5- 1
6- 14
7- 6
8- 5
9- 4
10- 0
11- 13
12- 10
13- 3
14- 9
15- 7
Enter wanted page:
3
Not on the physical memory, page fault
0- 8 - 1
1- 2 - 2
2- 15 - 3
3- 12 - 0
4- 11
5- 1
6- 14
7- 6
8- 5
9- 4
10- 0
11- 13
12- 10
13- 3
14- 9
15- 7
Enter wanted page:
4
Not on the physical memory, page fault
0- 8 - 1
1- 2 - 2
2- 15 - 3
3- 12 - 4
4- 11
5- 1
6- 14
7- 6

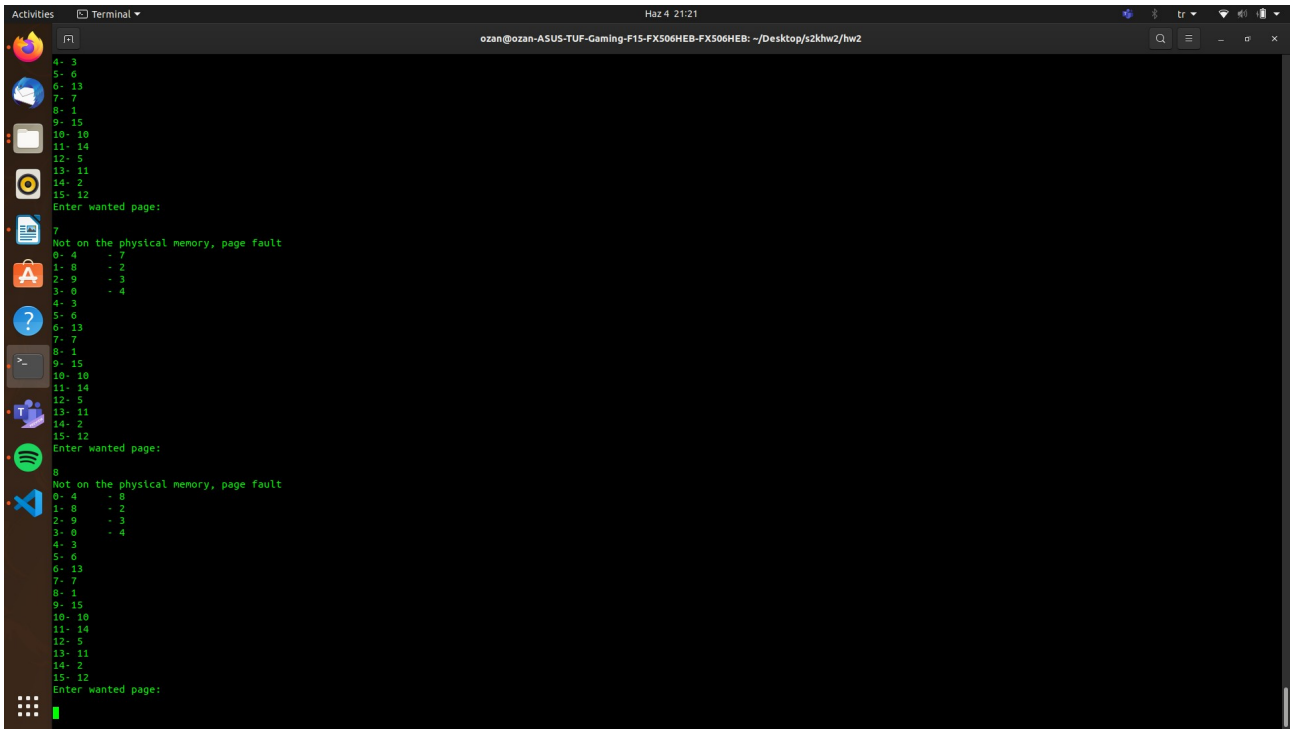

Activities
Terminal
Haz 4 19:44
ozan@ozan-ASUS-TUF-Gaming-F15-FX506HEB-FX506HEB: ~/Desktop/s2khw2/hw2
12- 10
13- 3
14- 9
15- 7
Enter wanted page:
2
Not on the physical memory, page fault
0- 8 - 1
1- 2 - 2
2- 15 - 0
3- 12 - 0
4- 11
5- 1
6- 14
7- 6
8- 5
9- 4
10- 0
11- 13
12- 10
13- 3
14- 9
15- 7
Enter wanted page:
3
Not on the physical memory, page fault
0- 8 - 1
1- 2 - 2
2- 15 - 3
3- 12 - 0
4- 11
5- 1
6- 14
7- 6
8- 5
9- 4
10- 0
11- 13
12- 10
13- 3
14- 9
15- 7
Enter wanted page:
4
Not on the physical memory, page fault
0- 8 - 1
1- 2 - 2
2- 15 - 3
3- 12 - 4
4- 11
5- 1
6- 14
7- 6



In this scenario user selects Second Chance Algorithm and wants 1 ,2 ,3 ,4 and at first they are not in page so they are all page fault, after that user wants 5 and 5 goes to 1's place, after that user wants 6 and 6 goes to 5's place. After that user wants 2 and 6 so 2 and 6 are wanted from user and that means 2 and 6 are used. When user wants 7, it means a page fault but instead of putting it to 6's place it puts it to 3's place.



In LRU algorithm when user enters 1,2,3,4,5,6,7 it works like above



In WSCLock when user enters 1,2,3,4,5,6,7,8 code it works like above.

Because the none of them is modified.