**Fredrick Onduso Ondieki**

**CST-221 Operating Systems Concepts**

**CST-221-Memory Management**

**John Zupan**

**March 31, 2019**

Git URL https://github.com/FREDDYSMALLZ/Operating-Systems-Concepts-CST-221.git

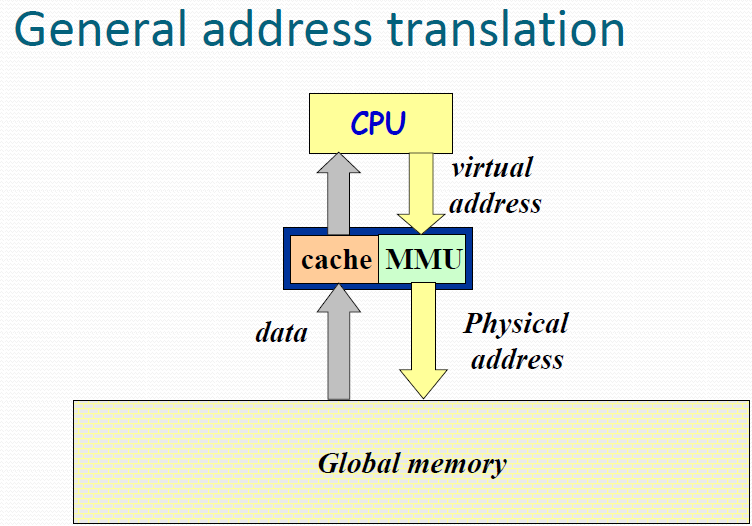
Introduction

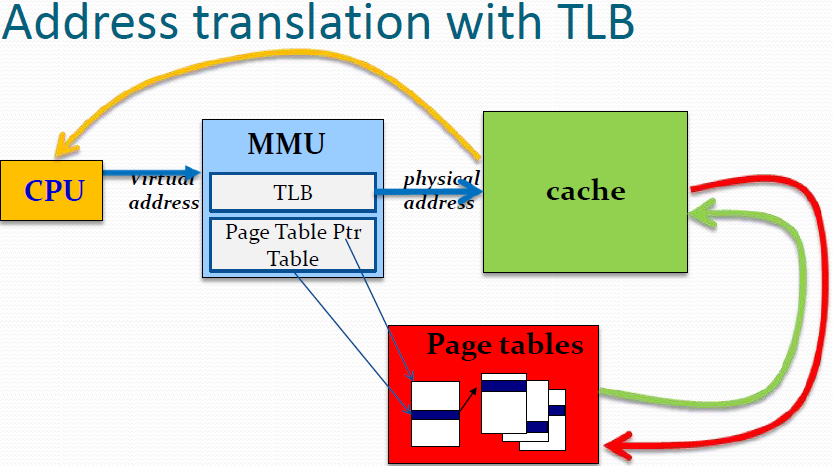
I n this project assignment, we are going to cover as well as learn a memory management unit (MMU), page fault handling algorithm and lastly a policy and mechanism for a virtual memory system. In addition, we will also write a C program that will work with various number of systems, such as decimal, binary, hexadecimal, as well as working with a binary shift operator. The objective is learning multiple numbering systems and the bit shift operators how it is important when working with computer addresses, the MMU and the memory management system.

**Memory Management Unit (MMU)-Question 1**

The memory management unit is a computer component that manages all computer memory and caching activities associated with the processor. It is the part that focusses on the memory management of the computer. In addition, MMU serves as a buffer between the CPU and system memory. The functions performed by the memory management unit can typically be divided into three areas: Hardware memory management, Operating system memory management and Application memory management.

Although the memory management unit can be a separate chip component, it is usually integrated into the central processing unit (CPU). On the other hand, MMU uses the Translation Look-Aside Buffer (TLB) to work on the virtual and physical addresses alongside with the page tables. The TLB hosts virtual address in the form of page lists and frames which are physical address which have been visited recently or about to be visited. For instance, if a process requests for a page reference, the TLB is checked for a page reference, then main memory is checked for the page reference, and finally the page is retrieved from the disk if all else fails. The following diagrams will expound in this text.





**Pseudocode**

*Boolean isInCache = True;*

*int page;*

*int frame;*

*TLB tLB;*

//Get information from the memory

*Int() searchInMemory(int page, int frame){*

*Int(2) address;*

*Address(0) = page;*

*Address(1) = frame;*

*return address();*

*}*

*int findThePage(page){*

*this.page = page;*

*return page;*

*}*

*int findTheFrame(frame){*

*this.frame = frame;*

*return frame;*

*}*

*//This code here checks for the TLB*

*void CheckForTLB(int page, int frame){*

*if (tLB.contains(page)){*

*isInCache = True;*

*searchInMemory(page, frame);*

*}*

*else{*

*isInCache = False*

*//Locate the addresses using the Page Table*

*void pageTable(int page, int frame){*

*findPage(page);*

*findFrame(frame);*

*searchInMemory(page, frame);*

*}*

*//main method*

*void main(){*

*//check if the information is in TLB.*

*checkTLB();*

*//Else, if not in TLB, find address using page table.*

*if (isInCache == False){*

*pageTable();*

*}*

*}*

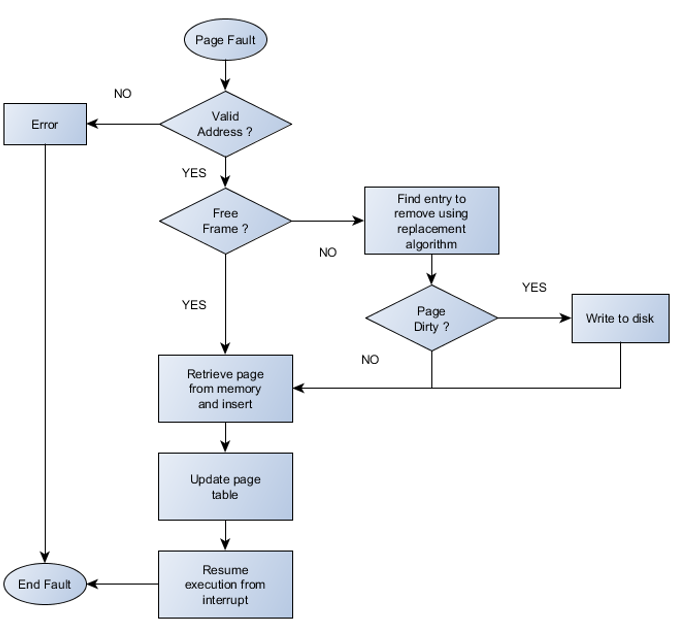
**Page Fault Handling-Question 2**

The basic idea behind paging is that when a process is swapped in, the pager only loads into memory those pages that it expects the process to need right away. Those pages that are not loaded into memory are marked as invalid in the page table, using the invalid bit. (The rest of the page table entry may either be blank or contain information about where to find the swapped-out page on the hard drive.) Otherwise, if the process only ever accesses pages that are loaded in memory then the process runs exactly as if all the pages were loaded in to memory. On the other hand, if a page is needed that was not originally loaded up, then a **page fault trap** is generated, which must be handled in a series of steps:

* The memory address requested is first checked, to make sure it was a valid memory request.
* If the reference was invalid, the process is terminated. Otherwise, the page must be paged in.
* A free frame is located, possibly from a free-frame list.
* A disk operation is scheduled to bring in the necessary page from disk
* When the I/O operation is complete, the process’s page table is updated with the new frame number, and the invalid bit is changed to indicate that this is now a valid page reference.
* The instruction that caused the page fault must now be restarted from the beginning, (as soon as this process gets another turn on the CPU).

The hardware necessary to support virtual memory is the same as for paging and swapping: A page table and secondary memory. A crucial part of the process is that the instruction must be restarted from scratch once the desired page has been made available in memory. For most simple instructions this is not a major difficulty. However, there are some architectures that allow a single instruction to modify a large block of data, and if some of the data gets modified before the page fault occurs, this could cause problems. One solution is to access both ends of the block before executing the instruction, guaranteeing that the necessary pages get paged in before the instruction begins (Merwyn, 2015).

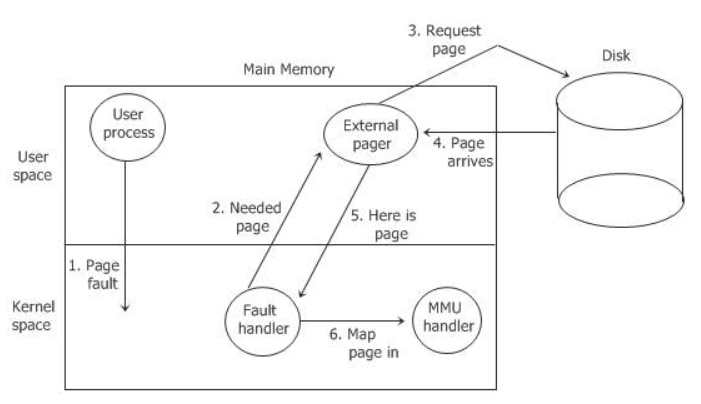
**Flow chart**



**Separation and Policy Mechanism**

This is a principle in computer science that mechanisms (those parts of a system implementation that control the authorization of operations and the allocation of resources) should not dictate (or overly restrict) the policies according to which decisions are made about which operations to authorize, and which resources to allocate. The separation of mechanism and policy is important to provide flexibility to a system. If the interface between mechanism and policy is well defined, the change of policy may affect only a few parameters. On the other hand, if interface between these two is vague or not well defined, it might involve much deeper change to the system.

In addition, it is important that the mechanism and policy to be separate to ensure that systems are easy to modify. For instance, no two system installations are the same, so each installation may want to tune the operating system to suit its needs. With mechanism and policy separate, the policy may be changed at will while the mechanism stays unchanged. This arrangement provides a more flexible system and memory management. The advantage of this principle is to separate the policy from mechanism to basically manage the complexity of any system. Also, this principal can be applied to memory management by having most of the memory manages run as a user-level process. The figure given below shows a simple example of how policy and mechanism can be separated:



**Question 4-C Program, Code and Screenshots**

The program described on this assignment takes a decimal number between 0 and 4095 as an input from the user and performs various operations described on the project assignment guidelines.

**Screenshots for program executionA screenshot of a cell phone

Description automatically generated**

A screenshot of text

Description automatically generatedA screenshot of a cell phone

Description automatically generated

Screen of a cell phone

Description automatically generatedA screenshot of text

Description automatically generated

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

References

Admin. May 8, 2014. Separation of Policy and Mechanism. Retrieved March 29, 2019 from website http://www.osinfoblog.com/post/133/separation-of-policy-and-mechanism/

Buettcher, Stephan. Fall, 2006. Memory Management. Retrieved March 29, 2019 from website https://www.student.cs.uwaterloo.ca/~cs350/F06/slides/cs350\_D.pdf

Includehelp.com. (2018). Working with Hexadecimal values in C programming language. Retrieved from http://www.includehelp.com/c/working-with-hexadecimal-values-in-c-programming-language.aspx

Merwyn, Prof. (2015, October 16). Steps for handling page fault. Retrieved March 29, 2019, from https://professormerwyn.wordpress.com/2015/10/16/steps-for-handling-page-fault/

Operating System | Memory management - mapping virtual address to physical addresses. (2018, August 09). Retrieved March 29, 2019, from https://www.geeksforgeeks.org/memory- management-mapping-virtual-address-physical-addresses/

Printing value in Decimal, Octal, Hexadecimal using printf in C. (n.d.). Retrieved March 29, 2019, from https://www.includehelp.com/code-snippets/printing-value-in-decimal-octal- hexadecimal-using-printf-in-c.aspx