**Memory Management**

Mathew Paschall

CST-221

John Zupan

February 11, 2018

https://github.com/Mpaschall/CST-221/tree/master/CST-221/assn6

**Memory Management Unit (MMU)**

The MMU is responsible for virtual and physical address translation, and also provides some other features. One way the MMU translates virtual and physical addresses is by using the Translation Look-Aside Buffer (TLB), which is inside the MMU. The TLB contains a list of pages (virtual address) and frames (physical address) visited recently. With the page and frame the MMU can bypass the page table and go directly to the memory location without having to check memory twice for these two variables. This essentially cuts operating time in half for any page that has been accessed recently. Once the TLB is full, it is cleared and begins the cycle of storing addresses again in order to speed up accessing these areas of memory. The pseudo code for accessing memory is shown below.

bool inCache = True;

int page;

int frame;

TLB tLB;

//Some functions to get info from memory

int[] findInMemory(int page, int frame){

int[2] address;

address[0] = page;

address[1] = frame;

return address[];

}

int findPage(page){

this.page = page;

return page;

}

int findFrame(frame){

this.frame = frame;

return frame;

}

//Check TLB

void CheckTLB(int page, int frame){

if (tLB.contains(page)){

inCache = True;

findInMemory(page, frame);

}

else{

inCache = False;

}

}

//Find address using page table

void pageTable(int page, int frame){

findPage(page);

findFrame(frame);

findInMemory(page, frame);

}

//main method

void main(){

//check if info is in TLB.

checkTLB();

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

if (inCache == False){

pageTable();

}

}

**Page Fault Handling**

Page fault handling is used when the page table does not contain all of the required pages needed for the process to proceed. This can be due to the fact that it only loads the pages that are deemed absolutely necessary to increase the speed at which the process is started. This results in a need for missing pages to be gotten later and added to the page table. This is rare due to locality of reference (Merwyn, 2015). Page fault handling is important because it allows the page table to obtain missing pages instead of having to go through standard memory access for the page every time it needs the information. It also checks to ensure that the request for the page is valid when it finds a page fault. For instance, a process may be requesting to write to a read only portion of memory. If this is the case, the process is terminated.

**Separation of Policy and Mechanism**

The separation of policy and mechanism is achieved in virtual memory by using the external pager, the MMU handler, and the page fault handler. When a user process requests a page that is not present in the page table, a page fault occurs. Then the fault handler reaches out to the external pager, which runs in the user space. This gets the requested page from the disk and copies it to its memory. This is passes back to the page fault handler and given to the MMU handler. The MMU handler then places this in the correct user address space. This is important because it allows the user and kernel space to work together to allow faults to be corrected properly. If this system was not in place, it would be hard for the user space to get the needed pages into the kernel space. If faults could not be corrected, the process would likely be terminated.

**Resources**

Buettcher, Stephan. Fall, 2006. Memory Management. Retrieved February 11, 2018 from website https://www.student.cs.uwaterloo.ca/~cs350/F06/slides/cs350\_D.pdf

Merwyn. October 16, 2015. Steps for handling page fault. Retrieved February 11, 2018 from website https://professormerwyn.wordpress.com/2015/10/16/steps-for-handling-page-fault/

Admin. May 8, 2014. Separation of Policy and Mechanism. Retrieved February 11, 2018 from website http://www.osinfoblog.com/post/133/separation-of-policy-and-mechanism/