EXPERIMENT 3

VIRTUAL MEMORY

1. OBJECTIVES

After completing this lab, you will be able to:

- Understand why TLB (Translation Look-aside Buffer) can provide fast address translation.
- Know how address translation is done by using IPT (Inverted Page Table).
- Understand how page replacement is essential to virtual memory and how to implement a clock replacement algorithm.

2. LABORATORY

Software Lab 1B (N4-01a-02)

3. EQUIPMENT

Pentium IV PC with Nachos 3.4.

4. MODE OF WORKING

You should be working alone. No group effort.

5. ASSESSMENT

The assessment of this lab will be based on your written report.

Your assignment will be assessed according to the following criteria:

- Originality of the program.
- Documentation - this includes internal and external documentation.
- Readability of the program.
- Workability of the program.

6. SUBMISSION PROCEDURE

You must submit your lab assignment to Computing Lab II before the due date. Late submission will be penalized and submission more than one week later will not be accepted. The deadline for submission is **ONE week after you attend lab session 3.** Hence each lab group has different deadline. You can check out your deadline from the lab technicians.

Your hard-copy submission must include a cover page, clearly showing the code and name of the course (i.e.,CPE205/CSC205/CE2005/CZ2005 Operating Systems), your name, and lab group number. Please slot in your assignment report according to your group number in the wooden shelf next to the lab main door.

7. PROBLEM STATEMENT

In this lab, you are required to complete a virtual memory implementation, including how to get a physical frame for a logical page from the IPT if it exists there, how to put a physical frame/logical page entry into TLB, and how to implement a clock page replacement algorithm.

To implement virtual memory, we use the software-managed TLB to cope with the address translation. We also have an IPT which maps physical frames to virtual pages. Basically, our translation process first examines the TLB to see if there is a match, if so, we just load from the TLB. If there is a miss, we then look that up in the IPT. If found, we then load it out to update the TLB. A miss in the IPT would mean that we have to load it from disk, and we would have to perform a page in and page out.

To decide which pages to page out, we have a customable replacement policy, for example, a modified clock algorithm which we will explain late on. Of course, during each lookup process, we have to perform a lot of checking in order to make sure that we are looking up the correct entry and that the entry is valid.

The entries for our TLB are basically the same as those given to us, in order to check whether we are referencing the correct entries from the TLB, we have to check the valid bit. The TLB would get updated when an exception is raised and a page isn't in it, and we insert the new page to the TLB after a bunch of error checks. Since our TLB is small, if it is full, our replacement policy for the TLB is simply FIFO.

Our IPT is simply a hash table. The hash function is some computation based on the virtual page number and the process id of that process, therefore same virtual address in different process would get hashed to different hash entries. Each entry is basically a doubly linked list (for the sake of quick reference and quick deletion). Our inverted page table therefore allows a process to make a quick connection between a virtual and physical address.

Our modified clock algorithm works by first setting up a static variable so that we can always know where our clock is pointing. Then the clock scrolls through our memoryTable (a mapping of what pages are in memory and their properties). If the entry the clock is on is not valid (i.e., its process is dead), the clock will return the number indicating the position of this invalid entry and increment itself to the next entry for next time. Otherwise the clock will check the entry it is on to see how long it has been since that entry was referenced. If the entry has been in memory without being referenced in a set time, it is paged out. If the position has been dirtied (changed), the clock allows it to stay longer since paging out dirty pages actually requires disk output which takes more time than just invalidating a page. If the entry has indeed been in memory too long without being referenced, the clock will return the index of that memory position for paging (and increment itself). Finally the clock is set to 0 when a page is brought in since it must be referenced to be brought in, and every time a page is introduced into the TLB, the clock is reset to 0 since it is being used at that moment.

8. <u>EXERCISES</u>

1. Change working directory to vm by typing ***`cd ~/nachos-3.4/vm`***.

2. Modify the following methods in file `tlb.cc`. You may need to reference the following files: `tlb.h`, `ipt.h`, `ipt.cc`, `machine/translate.h`, `machine/translate.cc`, `machine/machine.h`, and `machine/disk.h`.

    a. `int VpnToPhyPage(int vpn)` - Gets a physical frame `phyPage` for a virtual page `vpn`, if exists in the IPT.

    b. `void InsertToTLB(int vpn, int phyPage)` - Insert a vpn/phyPage entry into the TLB.

    c. `int clockAlgorithm(void)` – Return the freed physical frame according to the clock algorithm.

3. Compile Nachos by typing ***make***. If you see **"*ln -sf arch/intel-i386-linux/bin/nachos nachos*"** at the end of the compiling output, your compilation is successful. If you encounter any anomalies, type ***make clean*** to remove all object and executable files and then type ***make*** again for a clean compilation.

4. Execute the test program to test whether the virtual memory is working properly by typing ***./nachos –x ../test/vmtest.noff -d***. Fill the form (one example is shown below) whenever there is a TLB miss (i.e., the mapping entry is not found in the TLB, but it could be found in the IPT) or an IPT miss (i.e., the requested page is not in the memory at all; it must be a page fault and must trigger page replacement). Fill as many rows as necessary until Nachos exits.

vpn	**TLB Miss/ Page Fault**	**Page In**	**Page Out**	**phyPage**	**TLBEntry Inserted**
0	Page Fault	0	-	0	0
9	Page Fault	9	-	1	1
…	…	…	…	…	
0	TLB Miss	-	-	0	1
…	…	…	…	…	
9	Page Fault	9	27	2	2

5. Write down the page size, the number of physical frames, and the TLB size defined in Nachos as well as the number of pages used by the test program and the number of page faults occurred during the execution of the test program.

9. <u>Report</u>

It is recommended that your report should include analysis of the test program output. The analysis should clearly explain which TLB entry or physical frame should be used whenever there is a TLB miss or page fault. DO NOT attach the debugging printout from the program.