Project 3 – Demand Paging

Introduction to Operating Systems **CSE4/521**

Due : Dec. 8, 11:59:59 PM

The third Nachos project is to build a demand paging system. You will create the mechanisms and policies for a demand paging system with a simple page table system. You will also implement a simple page replacement algorithm.

Phase 1: Understand the Code

You will modify the set of files that you have after completion of **projects 1**. It is important that you understand how the demand paging and address translation system in Nachos works. To this end, we suggest that you examine the following files:

machine.*	**We** will not use TLB. We will compile in the vm directory to make use of the dependency setting stated in the Makefile there. Remove USE_TLB flag from the lines stating the various components used.
translate.*	Examine the ReadMem, WriteMem, and Translate methods of machine. It is important to understand how a PageFaultException is generated for a page miss. Pay particular attention to where the invalid virtual address is stored when the exception occurs, since you will need to access it later.
addrspace.*	The role of address space will change, but it is now important to understand what ALL of the methods do, not just the constructor or deconstructor.
exception.cc	The majority of your demand paging code will be called from the PageFaultException handler.
bitmap.*	routines for manipulating bitmaps
filesys.h	
openfile.h	Pay careful attention to the readAt and writeAt methods!
../test/*	C programs that will be cross-compiled to MIPS and run in Nachos

Phase 2 : Design Considerations

When the machine object is instantiated, the pagetable will be initialized with all entries invalid. This will generate a page fault when the first memory address is accessed. The majority of your design will deal with what happens when a page fault occurs.

In order to implement full demand paging, you will also be required to create and modify several other data structures. Such data structures include:

- Modification of the TranslationEntry data structure to support a reference word.
- A method for determining the source and destination expression for the page to be loaded, from a file to the main memory.

You will also have to account for all phases of the demand paging algorithm. These phases include:

- Page Table Hit
- Page Table Miss, free entry in the main memory
- Page Table Miss, page replacement in the main memory, replacement page clean, and

- Page Table Miss, page replacement in the main memory, replacement entry dirty.

You should also make sure that you separate the mechanisms and policies. Initially you will use FIFO policy for replacement. After testing pagefault handler with FIFO, replace it with LRU page replacement algorithm.

Phase 3: [90%] Lab 3 Assignment
NOTE: It is very important to compile this project in the VM directory! Your files will be in userprog.

a) **Understand Page Tables –** This is very important, but will not count towards your grade.

b) **Change Constants in machine.h –** Please change the constants in machine.h to the following:
NumPhysPages 32

c) **Change class TranslationEntry in translate.h** – Add 'int refword' as an instance variable to this class (under the dirty bit)

d) **Moving pages in and out of main memory:** A critical step in the demand paging in rolling pages needed from an executable file into the main memory and rolling out dirty pages back to the file. It is suggested that you write a helper class that will exclusively focus on these operations. For more information on the code look at address space constructor.

e) **Data structure for the Page Table –** Base the design of the page table on the TranslationEntry class. You may need to add an extra entry for page replacement policy.

f) **Implement FIFO Page Table Demand Paging –** You must now process all PageFaultExpections, and implement the core demand paging algorithm. After complete testing use LRU method.

g) Design three user programs that can be run inside the shell to demonstrate the robustness of your overall project solution.

Phase 4: [10%] Documentation

This includes internal documentation (comments) and a ***BRIEF, BUT COMPLETE*** external document (read as: paper) describing what you did to the code and why you made your choices.

Deliverables and grading

When you complete your project, remove all executables and object files. If you want me to read a message with your code, create a README.NOW file and place it in the nachos *code* directory. Tar and compress the code, and submit the file using the online submission system. It is important that you follow the design guidelines presented for the system calls. I will be running my own shells and test programs against your code to determine how accurately you designed your lab, and how robust your answers are. Again, grading will be based on how your solution fares against my test cases. Remember, the user should not be able to do anything to corrupt the system, and system calls should trap as many error conditions as possible.