COMP1521 17s2

Computer System Fundamentals

Assignment 2
Virtual Memory Simulator
Draft

Objectives

- to give you more practice writing C
- to give you experience with a simulation
- to learn more about virtual memory systems

Admin

Marks	13 (towards total course mark) (Note that the marks below add up to XX; they will be scaled to a mark out of 13)
Group?	This assignment is completed **individually**
Due	by 11:59:59pm on Sunday 15th October
Submit	`give cs1521 ass2 PageTable.c` or via Webcms
Late Penalty	0.1 marks per hour late (2.4 marks per day) off the ceiling (e.g. if you are 48 hours late, your maximum possible mark is 8.2/13)

Background

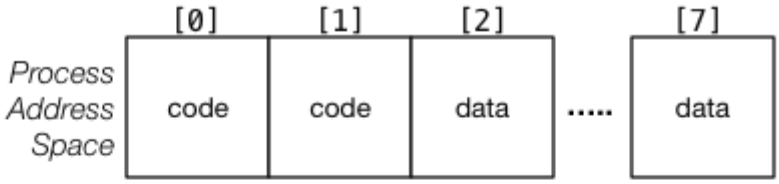
Virtual memory is a powerful technique that allows the main memory of a computer system to be shared amongst a large number of processes. It also allows indivudual processes to have a process address space that is potentially larger than main memory.

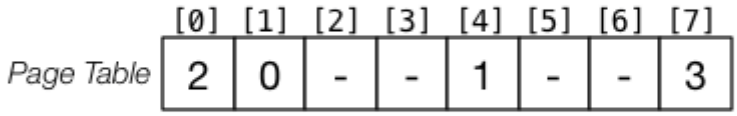
The implementation of virtual memory requires the system to maintain, for each process, a mapping table (page table) that tells where each page is loaded in memory, or that it is not loaded. Whenever the process references an address in its address space, we use the page table to map this virtual address to a real location in the machine's memory. If a referenced page is not currently loaded, then we need to place it in a memory frame. If there is a free frame, we can use that; if all frames are in use, then we need to make space, by replacing the contents of one of the existing frames.

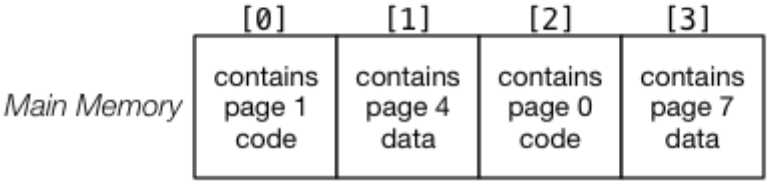
For more details on page replacement strategies, see the lecture notes. For this exercise, we focus on:

- LRU ... replace the page that hasn't been accessed for longest time
- FIFO ... replace the page that was loaded earliest

The following diagram shows a page table that maps the pages of a process with 8 pages into the 4 frames in main memory:







The aim of this assignment is to build a simple simulation of a system like the above, where we can measure how the system behaves if we change the number of pages, number of frames, and the page replacement policy.

We do not fully represent pages or frames in our system, nor do we "execute" programs. We provide data structures for the page table and a minimal representation of memory pages. This gives enough to read a sequence of page references and measure how the page table might behave in a real implementation.

Setting Up

Create a new directory for working on the assignment, change into that directory, and then run the command:

```
$ unzip /home/cs1521/web/17s2/assigns/ass2/ass2.zip
```

This will add the following files into the directory:

`Makefile`	to control compilation
`Stats.[ch]`	implementation and interface to the summary statistics collection and display
`Memory.[ch]`	implementation and interface to the representation of physical memory frames
`PageTable.[ch]`	implementation and interface of the page table (what you need to modify)
`vmsim.c`	the main program for the virtual memory simulator
`mktrace.c`	a program to generate memory reference traces

The page table is represented by an array of PTE structures which is dynamically created at the start of the simulation. Each PTE contains the following information:

```
char status;      // NOT_USED, IN_MEMORY, ON_DISK
char modified;    // boolean: has the page changed since loaded?
```

```
int  frame;       // memory frame holding this page
int  accessTime;  // clock tick for last time the page was accessed
int  loadTime;    // clock tick for last time the page was loaded
int  nPeeks;      // total number times this page referenced for reading
int  nPokes;      // total number times this page referenced for writing
```

You can find more details on the supplied code, and how to use it in the **Assignment 2 Video**.

Exercise

The aim of this exercise is to implement two page replacement strategies (LRU and FIFO) and collect statistics on the page table's behaviour. Implementing the page replacement strategies requires you to complete the `findVictim()` function. Collecting he statiscs requires you to place calls to the statistics functions in the appropriate places. You can find out more details on what statistics need to be collected in the `Stats.c` file.

A simple, and extremely inefficient, implementation of LRU replacement would work as follows:

```
oldest = now
victim = NONE
for (i = 0; i < #Pages; i++) {
   P = PageTable[i]
   if (P's accessTime < oldest)
      oldest = P's accessTime
      victim = P's page
   }
}
```

A better strategy would be maintain a list of pages, ordered by access time from oldest (first in list) to most recent (last in list). The above code would then simply remove the first page from the list and use that as the page to be replaced. Of course, the list needs to be updated each time a page is accessed.

The list should not be implemented as a separate data structure, but could be done by incorporating a "link" to the next page in the list in each PTE. You would also need to maintain a global variable to hold the page number of the first page in the list.

Similarly, a simple and inefficient implementation of FIFO replacement would work as follows:

```
oldest = now
victim = NONE
for (i = 0; i < #Pages; i++) {
   P = PageTable[i]
   if (P's loadTime < oldest)
      oldest = P's loadTime
      victim = P's page
   }
}
```

As above, a better strategy would be to maintain a list of pages, ordered by load time from oldest (first in list) to most recently loaded (last in list). The implementation would be similar to that for the LRU list.

What you are required to do (and you only need to change the `PageTable.c` file for this assignment) is specified in comments in the `PageTable.c` file.

Extensions

(Worth kudos, but no marks)

Implement the Clock sweep page replacement strategy.

See the Wikipedia entry for page replacement algorithms for more details.

Have fun, *jas*