CS356 Operating System Projects Spring 2015

Project 2: Simulating Virtual Memory and Scheduling

Objectives:

- Get experience with page replacement algorithms
- Get experience with round robin CPU scheduling algorithm
- Get experience with writing simulation
- Get experience with software engineering techniques

Problem Statements:

Your primary goal for this project is to implement and experiment with page replacement algorithms. To do this, you will write a paging **simulator**. It will read in a set of data files specifying the scheduling and memory access behavior of programs, and will simulate the paging requirements of those programs.

As input, you will receive two types of files: scheduling traces and memory traces.

1. Scheduling Traces

Scheduling files contain a trace of process start time, CPU time, and I/O count. For the purposes of this project, you will only need to look at start time and CPU time. The format of these files is illustrated below:

```
xlogout     0     8.58     102720
login.kr    3     1.36     137024
rm          4     0.03     96
```

The format is:

Program name	Start time (in seconds)	CPU time (in seconds)	I/O Count

This trace indicates that *xlogout* started at time **0**, required **8.58** CPU seconds to execute, and issued **102,720** I/O operations. The *login.kr* program started at time **3**, required **1.36** CPU seconds to execute, and issued **137,024** I/O operations.

The trace is in order, meaning that each process has a start time later than or equal to the previous start time.

2. Memory Traces

Memory trace files contain a list of addresses accessed by a program. The addresses are truncated to virtual page numbers by removing the lower 12 bits, which makes the files smaller. Here is an example for PowerPoint program:

```
0x0012f
0x77f46
0x0012f
0x7ffde
0x0012f
```

```
0x77f3c
0x0012f
0x7ffde
0x0012f
```

As an aside, the even numbered addresses (e.g. index 0,2,4) are instruction addresses and the odd numbered addresses (e.g. at index 1,3,5) are data addresses. This accounts for the alternating low values (instructions) and high values (data).

The memory traces are a list of virtual page numbers. This means that the page size does not matter for this project. In addition, you can assume that there is no sharing: the virtual page is private to a process.

3. Simulating scheduling

You will need to simulate a simple scheduling algorithm that allows programs to alternate running. For this project, you only need implement round robin between runnable processes with fixed time slices. Your simulator should take as input the number of memory references in a time slice, which is called the **quantum**.

Your scheduler will keep track of the current time in terms of cycles, where a cycle is one memory reference in the memory trace. You will simulate a fairly slow system, where there are 100,000 cycles in a second.

Your scheduler should run (1) when the current quantum expires, to decide whether to context switch the process, and (2) when a process blocks on a page fault (this will be explained below). Context switching should take 50 cycles. After a context switch, you should start simulating memory references from where you left off.

The scheduler should pick the next process to run and then start simulating the memory accesses for that process (see next section). Like a real scheduler, your simulated scheduler must keep track of which process is running, which are ready, and which are blocked.

Your simulator should maintain the current time (e.g., the number of seconds or cycles since the simulation started). A process from the trace should be put in the ready queue (or run) when the current time equals the start time in the scheduling trace file. It executes memory instructions for the CPU time value from the trace file, and then exits when it has used up its CPU time.

The scheduler should report elapsed time (time between when it started and when it completed) for each process. In addition, at the end of the simulation (when all processes have completed), it should report the total number of cycles executed and the number of cycles where no process was executing (i.e., there was nothing runnable).

You should experiment with quanta values of 10,000 cycles, 50,000 cycles, and 200,000 cycles.

4. Simulating page replacement

Your simulated machine will have a fixed number of physical pages that you must share between the currently running processes. Hence, when the memory required by the running programs exceeds the available physical memory, you will need to store some of the memory on a disk.

On each cycle, you will simulate the next memory reference in the trace for the running program. When a program starts, you should open the corresponding trace file based on the program name. For example, if the program name is "grep", you should open the "grep" memory trace.

You will simulate this by keeping track of which pages are in physical memory and which are on disk. It takes 1000 cycles to move a page from disk to memory. Moving pages from memory to disk is free, as this can be overlapped with computation. Only one page at a time can be read off disk. For example, if two processes both fault on different pages, you need to wait 1000 cycles for the first page and then 1000 cycles for the second page. You may want to maintain a queue of pending requests for the disk. While a page is moving from disk to memory, you can execute other processes -- the operation is asynchronous.

For each cycle that process executes, you should check whether the virtual page it accesses is in physical memory. If so, advance to the next address. If it is not in physical memory, you must simulate a **page fault** by:

1. Context switching to a new process
2. Putting the current process on a blocked queue
3. Reading the page off disk
4. Making the process ready again when the page comes back off disk

You must keep track, for each process, which pages are in physical memory and which are on disk. Assume that all pages start off on disk and you are doing demand paging. Because writing pages to disk is free, you don't have to worry about whether a page is dirty or not (and in fact, the trace does not indicate whether a memory reference is a read or a write).

For each process, you should report the number of page faults it experienced. In addition, when the simulation terminates, you should report the total number of page faults experienced across all processes.

You should experiment with total physical memory sizes of 50 pages, 250 pages, and 1000 pages.

5. Page replacement policies

You should simulate three different page replacement policies, specified as a command line parameter:

1. **FIFO:** The first page brought in is the first to be removed.
2. **LRU:** The least recently accessed page is the one to be removed
3. **Second-chance algorithm:** You should maintain a clock hand that rotates around memory. Each physical page is marked "unused". Each time a page is needed, the hand advances until it finds a page marked "unused", at which time that page is chosen for eviction. If it finds a page marked "used" in this process, the page is changed to be "unused". In addition, on every memory reference, the page accessed should be marked "used".

For the provided scheduling traces, you should try all three page replacement policies to see how each one performs.

6. Extra Credit

If you finish this project early, you can earn up to **15%** extra credit by trying to invent your own page replacement policy. Wikipedia lists several policies you could try.

The extra credit will be a competition: we will run your extra replacement policy on a set of programs with a medium (not small) amount of memory. The person with the best policy will get up to 15% extra credit, and other guys will receive proportionally less based on his/her rank.

7. Write-up

You should write a 3-5 page document describing your system and explaining your results.

You should explain the architecture of your simulator: what the components are, what the interfaces between components are, and why you chose this design. **In addition, you should document policy decisions you made, such as:**

- Do new processes go at the head of the ready list or the end?
- When a process starts up after a page fault, is it guaranteed to have the faulting page in memory, or could it fault again?
- Do new processes or processes returning from a page fault get priority?

This is not a complete list -- you should include any policy decisions you made.

You should present three sets of experiments. For each one, you should present your results graphically (e.g., with a chart):

1. **Page replacement algorithm:** Pick a reasonable amount of memory and time quantum, and compare the performance of the page replacement algorithms in terms of total page faults experienced and the total run time of programs (i.e., the average elapsed time of all programs).
2. **Time quantum:** With second-chance algorithm, show the difference that the time quantum has on page faults and run time. You should present the results of your simulations with the three quantum values (10,000, 50,000 and 200,000). You should use a reasonable amount of memory (i.e., not 50 pages but not so much that there are no page faults).
3. **Memory size:** With second-chance algorithm, demonstrate the difference that memory size has on page replacement. You should experiment with three memory sizes: 50, 500, and 5000 pages. Use a reasonable value for time quantum (e.g., 50,000 or 100,000 cycles).

For each experiment, explain the differences in performance between the different configurations.

8. Project Specification

The command line for your simulator should be:

mem-sim pages quantum pr-policy trace-file

where

- **pages** is the number of physical pages

- **quantum** is the number of cycles in a scheduling quantum
- **pr-policy** is one of "fifo", "lru" or "2ch-alg" and indicates which page replacement policy to use.
- **trace-file** is the name of the scheduling trace file

The output of the simulator should be:

- For each process, the elapsed time it took to run and the number of pages faults it experienced.
- For the entire simulation, the total elapsed time, the total number of page faults, and the total idle time.

The simulator should handle these events:

- Memory reference
- Process creation
- Timer interrupt for time slices
- Disk I/O with disk interrupts
- Page faults

The costs of events for the simulation are:

- Memory reference: 1 cycles
- Context switch: 50 cycles
- Read page off disk: 1000 cycles

Other overheads, such as running the scheduler or handling a page fault, need not be considered. The simulator should execute one memory instruction per cycle. Context switches take 50 cycles during which no memory instructions may be executed. Replacing a page (evicting it from memory) is free, but bringing it back from disk takes 1000 cycles. During this time, other programs may execute.

You should experiment with total physical memory sizes of 50 pages, 500 pages, and 5000 pages. You should experiment with quanta values of 10,000 cycles, 50,000 cycles, and 200,000 cycles. You should run experiments with FIFO, LRU, and 2nd-chance algorithm page replacement algorithms.

Material to be submitted:

1. Compress the source code of the programs into **Prj2+StudentID.tar** file. Use meaningful names for the file so that the contents of the file are obvious. A single **makefile** that makes the executable out of any of the source code should be provided in the compressed file. Enclose a README file that lists the files you have submitted along with a one sentence explanation. Call it **Prj2README**. (-10 points if documentation is missing)
2. Please state clearly the purpose of each program at the start of the program. Add comments to explain your program. (-5 points, if insufficient.)
3. Test runs: It is very important that you show that your program works for all possible inputs. Submit online a single typescript file clearly showing the working of all the programs for correct input as well as graceful exit on error input.

4. Send your **Prj2+StudentID.tar** file and the 3-5 pages report to cs356.sjtu@gmail.com with an email titled **Prj2+StudentID submission**.
5. Due date: June 12, 2015, submit on-line before midnight.
6. Demo slots: June 13-14, 2015. Demo slots will be posted on the door of East 3-309 SEIEE Building. Please sign your name in one of the available slots.
7. You are encouraged to present your design of the project optionally. The presentation date will be announced soon. Please pay attention to the course website and mailing list.