COMP SCI 3004/7064 - Operating Systems Assignment 2

DUE: 23:30pm, 28th Oct, 2019

Important Notes

- Handins:
  - The deadline for submission of your assignment is **23:30pm, 28th Oct, 2019**.
  - **For undergraduate students**, you may do this assignment as a team of two students and hand in one submission per team.
  - **For postgraduate students**, you have to do this assignment individually and make individual submissions.
  - All implementations have to be done in **C++**.
  - You need to submit your source code using the web submission system. You should attach you and your partner's name and student number in your submission.
  - Late submissions will attract a penalty: the maximum mark you can obtain will be reduced by 25% per day (or part thereof) past the due date or any extension you are granted.
- Marking scheme: 20 marks for online testing in total:
  - 4 marks for FIFO (First In First Out) implementation (2 random cases, automarked)
  - 4 marks for LRU (Least Recently Used) implementation (2 random cases, automarked)
  - 6 marks for ARB (Additional Reference Bits) implementation (3 random cases, automarked)
  - 6 marks for WSARB (Working-Set Additional Reference Bits) implementation (3 random cases, automarked)

If you have any questions, please send them to the student discussion forum. This way you can all help each other and everyone gets to see the answers.

The assignment

Introduction

The aim of this assignment is to improve your learning experience in the page replacement algorithms. Following our discussion of paging in lectures, this practical will allow you to explore how real applications respond to a variety of page replacement schemes. Since modifying a real operating system to use different page replacement algorithms can be quite a technical exercise, your task will be to implement a program that simulates the behaviour of a memory system using a variety of paging schemes.

Memory Traces

We provide you with some memory traces to assist you developing your simulator.

Each trace is a series of lines, beginning with #, each listing a hexadecimal memory address preceded by R or W to indicate a read or a write. There are also lines throughout the trace starting with a # followed by a process ID and command. For example, a trace file for gcc might start like this:

```
# gcc
R 0041f7a0
R 13f5e2c0
R 05e78900
R 004758a0
W 31348900
```

Simulator Requirements

Your task is to write a simulator that reads a memory trace and simulates the action of a virtual memory system with a single level page table.
Your memory system should keep track of which pages are loaded into memory.

- As it processes each memory event from the trace, it should check to see if the corresponding page is loaded.
- If not, it should choose a victim page in memory to replace.
- If the victim page to be replaced is dirty, it must be saved to disk before replacement.
- Finally, the new page is loaded into memory from disk, and the page table is updated.

This is just a simulation of the page table, so you do not actually need to read and write data from disk. When a simulated disk read or write occurs, simply increment a counter to keep track of disk reads and writes, respectively.

You must implement the following page replacement algorithms:

- FIFO: Replace the page that has been resident in memory longest (oldest).
- LRU: Replace the page hat has not been used for the longest period of time.
- ARB: Use multiple reference bits to approximate LRU. Your implementation should use an a-bit shift register and the given regular interval b. See textbook section 9.4.5.1.
  - If two page's ARB are equal, you should use FIFO (longest) to replace the frame.
- WSARB: Combine the ARB page replacement algorithm with the working set model that keeps track of the page reference frequencies over a given window size (page references) for each process (see textbook section 9.6.2). It works as follows:
  - Associate each page with a shift register (R) of a ($1 \leq a \leq 8$) bits to track the reference pattern over the recent time intervals (for a given time interval length b), and a reference frequency (integer) counter (C) of 5 bits to record the reference frequency of the pages in the current working set window. R and C are initialized to 0.
  - Page replacement is done by first selecting the victim page of the smallest reference frequency (the smallest value of C), and then selecting the page that has the smallest reference pattern value in the ARB shift register (the smallest value of R) for those with the same frequency.

Test

Arguments

Your code will be compiled using following order in your SVN folder.
g++ -std=c++11 PageReplacement.cpp -o PageReplacement
The simulator should accept arguments as follows:

1. The filename of the trace file
2. The page/frame size in bytes (we recommend you use 4096 bytes when testing).
3. The number of frames allocated to each input trace (page reference string) in the simulated memory.

The trace provided should be opened and read as a file, **not** parsed as text input from stdin.
For example, your code might be run like this:
“‘ ./PageReplacement input.txt 4096 32 FIFO “‘ Where:

- The program being run is ‘./PageReplacement‘,
- The name of the input file is ‘input.txt‘,
- A page is ‘4096‘ bytes,
- There are ‘32‘ frames of physical memory allocated,
- The page replacement algorithm to use is FIFO.

If the page replacement algorithm is ARB , it should accept the following additional argument:

1. a**, the number of bits of the shift register (R)** ($1 \leq a \leq 8$)**.**
2. b**, the length (= the number of page references) of the regular (time) interval** ($1 \leq b \leq 10$)**.**

If the page replacement algorithm is WSARB, it should accept the following additional argument:

1. a**, the number of bits of the shift register (R)** ($1 \leq a \leq 8$)**.**
2. b**, the length (= the number of page references) of the regular (time) interval** ($1 \leq b \leq 10$)**.**
3. δ**, the size (= the number of page references) of the working set window** ($1 \leq \delta \leq 20$)**.**

For example, your code might be run like this:

```
./PageReplacement input.txt 1024 16 ARB 3 3
```

```
./PageReplacement input.txt 4096 32 WSARB 8 3 11
```

Input

We will provide you with a selection of memory traces to assist you developing your simulator. These will be a mix of specific test cases and real traces from running systems. Each trace is a series of lines, containing two(2) values that represent memory accesses:

1. A character ‘R‘ or ‘W‘ that represents whether the memory access is a Read or Write respectively.
2. A hexadecimal memory address.

A trace may also contain comment lines, # followed by a process Name.

An example of a trace:

```
# chrome
R 0041f7a0
R 13f5e2c0
R 05e78900
R 004758a0
W 31348900
```

Output

The simulator should run silently with no output until the very end, at which point it prints out a summary like this:

events in trace: 1025
total disk reads: 151
total disk writes: 92
page faults: 151

Where:

- * **events in trace** is the number of memory access' in the trace. Should be equal to number of lines in the trace file that start with R or W. Lines starting with # do not count.
- * **total disk reads** is the number of times pages have to be read from disk.
- * **total disk writes** is the number of times pages have to be written back to disk.
- * **page faults** is the number of disk reads in a demand paging system. It may be same with total disk reads in this problem.

We will provide a set of expected outputs(on web-submission) to match the given memory traces.

Web-submission instructions

- First, type the following command, all on one line (replacing xxxxxxx with your student ID):
  svn mkdir –parents -m "OS"
  https://version-control.adelaide.edu.au/svn/axxxxxxx/2019/s2/os/assignment2

- Then, check out this directory and add your files:
  svn co https://version-control.adelaide.edu.au/svn/axxxxxxx/2019/s2/os/assignment2
  cd assignment2
  svn add TicketBooker.cpp
  ...
  svn commit -m "assignment2 solution"

- Next, go to the web submission system at:
  `https://cs.adelaide.edu.au/services/websubmission/`
  Navigate to 2019, Semester 2, Operating Systems, Assignment 1. Then, click Tab "Make Submission" for this assignment and indicate that you agree to the declaration. The automark script will then check whether your code compiles. You can make as many resubmissions as you like. If your final solution does not compile you won't get any marks for this solution.

- We will test your codes by the following Linux commands:
  g++ -std=c++11 PageReplacement.cpp -o PageReplacement
  ./PageReplacement input.txt 4096 32 FIFO >output.txt

FAQ

1. How are the page numbers and memory addresses connected? Is there a relation to page size?

   This comes down to the fundamentals of paging. Be sure to read and understand section 8.5.1 of the text book before going any further. It will save you much confusion later on.

2. I'm confused by the ARB algorithm. How is it actually supposed to work?

   For those of you Googling along at home you may have come across a variety of somewhat conflicting descriptions of this algorithm. For reference, we're strictly following this one outlined in the book:

   > **9.4.5.1 Additional-Reference-Bits Algorithm**
   >
   > We can gain additional ordering information by recording the reference bits at regular intervals. We can keep an 8-bit byte for each page in a table in memory. At regular intervals (say, every 100 milliseconds), a timer interrupt transfers control to the operating system. The operating system shifts the reference bit for each page into the high-order bit of its 8-bit byte, shifting the other bits right by 1 bit and discarding the low-order bit. These 8-bit shift registers contain the history of page use for the last eight time periods. If the shift register contains 00000000, for example, then the page has not been used for eight time periods. A page that is used at least once in each period has a shift register value of 11111111. A page with a history register value of 11000100 has been used more recently than one with a value of 01110111. If we interpret these 8-bit bytes as unsigned integers, the page with the lowest number is the LRU page, and it can be replaced. Notice that the numbers are not guaranteed to be unique, however. We can either replace (swap out) all pages with the smallest value or use the FIFO method to choose among them.
   >
   > The number of bits of history included in the shift register can be varied, of course, and is selected (depending on the hardware available) to make the updating as fast as possible. In the extreme case, the number can be reduced to zero, leaving only the reference bit itself. This algorithm is called the **second-chance page-replacement algorithm**.

   Figure 1: ARB

   Some additional notes on ARB:

   - We use a-bit shift register (R), instead of 8-bit shift register in the textbook.
   - Each input trace is to be run on a 64-bit system which uses a larger address space. If you're converting these values to numbers, you may need to use the long long data type.

3. I'm confused by the working set model. How is it actually supposed to work?

   Please refer to 9.6.2 of pages 427-429 of the text.

   For Working-Set ARB. Some extra notes:

   - We are concerned only with single-process memory allocation, so each input reference string is only for the same process.
   - Ensure that your code correctly keeps track of the reference frequencies of all the pages in the current working set window that covers the most recent δ page references on the input reference string.
   - All page faults should be recorded and provided in the output.
   - The working set window δ is unrelated to ARB shift interval b.
   - For WSARB, we assume that the number of frames allocated to the process is at least equal to the size of its working set window.
     For example:
     Time: 1 2 3 4 5 6 7 8 9 10...
     request pages: 2 1 3 4 5 2 4 6 1 2...
     frames size=4;
     window size=4;
     At end of time 4, four frames are in a table in memory: 2 1 3 4.
     Time: 1 ***2 3 4 5*** 6 7 8 9 10...
     request pages: 2 ***1 3 4 5*** 2 4 6 1 2...
     At time 5, a new page 5 need into the frames. And the current working set window is 1 3 4 5. The 5-bit reference frequency counter (C) will find that the frequency of page 2 is zero (the smallest value of C) in the current working set, and then the new page 5 will replace page 2 in the memory table.