CISC 3595—Operating Systems
Monday 18 November 2013

PROGRAMMING PROJECT # 3
Analyzing the LRU Algorithm

Date Due: Monday 9 December 2013

Suppose that we have m page frames of physical memory and n pages of virtual memory. Then we must have $m < n$; most likely, $m \ll n$. We can then represent virtual memory by a vector (say, `pgFrame`) of fixed size n, with the first m elements representing physical memory pages and the remaining $n - m$ elements representing virtual memory pages (on disk).

Recall (pg. 377 of the text) that a *stack algorithm* is a page replacement algorithm for which Bélády's anomaly cannot hold, i.e., for which $M_t(n) \subseteq M_t(n+1)$. Here, $M_t(n)$ denotes the set of pages in memory at time t, assuming that we have n frames of physical memory. The following pseudocode illustrates how a stack algorithm would process a given reference string of virtual page requests; here, `pgToEvict()` selects the victim to be yanked from physical memory into virtual memory.

```
while there are more pages in reference string do
  $r \leftarrow$ next page request
  if $r$ is already in memory then
    either do nothing, or handle this somehow
  else if $\exists$ an empty page frame $f$ for this page request then
    pgFrame[$f$] $\leftarrow r$
  else          // page fault
    $p \leftarrow$ pageToEvict()
    move pageRef[$p$] to bottom part of pageRef
    $r \leftarrow p$
  end if
end while
```

The *least-recently used* (LRU) algorithm is a variation on this theme. Let r be the current page request.

- If r is already in (physical or virtual) memory (say, $r =$ `pgFrame`$[q]$), then we move pages $0, 1, \ldots, q-1$ down one slot and then push r to the top of the stack, so that `pgFrame`$[0] \leftarrow r$.
- If r is not in memory, we push it onto the top of the stack.

For example, suppose that we have 4 pages of physical memory and 8 pages of virtual memory, with a reference string

```
0 2 1 3 5 4 6 3 7 4 7 3 3 5 5 3 1 1 1 7 1 3 4 1
```

Here's what the stack looks like as the LRU algorithm progresses through this set of page requests. The top four rows represent physical memory, the bottom four rows represent virtual memory pages that do not reside in physical memory.

	0	2	1	3	5	4	6	3	7	4	7	3	3	5	5	3	1	1	1	7	1	3	4	1
		0	2	1	3	5	4	6	3	7	4	7	7	3	3	5	3	3	3	1	7	1	3	4
			0	2	1	3	5	4	6	3	3	4	4	7	7	7	5	5	5	3	3	7	1	3
				0	2	1	3	5	4	6	6	6	6	4	4	4	7	7	7	5	5	5	7	7
					0	2	1	1	5	5	5	5	5	6	6	6	4	4	4	4	4	4	5	5
						0	2	2	1	1	1	1	1	1	1	1	6	6	6	6	6	6	6	6
							0	0	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	P	P	P	P	P	P	P		P					P			P						P	

Each P denotes a page fault, and so we see that there are 11 page faults.

Your program should accept as input the following information:

- *m*, the number of physical page frames,
- *n*, the number of virtual pages, and
- a reference string of page requests.

Given this input, your program should do the following:

1. Print the stack as the program processes each page request. Each "snapshot" of the stack should be printed as a horizontal row, rather than vertically (it's somewhat of a pain to draw a diagram such as the one above). Draw a vertical bar ("`|`") to separate the physical memory slots from the slots for virtual memory pages that are swapped out. If a page fault occurs, print a `P` at the end of the line.
2. Calculate and print the number of page faults.

Please make sure that your output is neatly formatted.

The project share directory (`~agw/class/os/share/proj3`) contains the following goodies:

- Two data files, `data1` and `data2`, which are text files having the following structure:
  1. The first line contains two integers, the first being *m* and the second being *n*.
  2. The rest of the file contains the reference string, which will consist of as many lines as are necessary to hold the reference string, with as many entries per line as will fit comfortably.
- A C++ source file `read-proj3-data.cc`, which demonstrates how to read the data file mentioned above.
- A working executable, named `proj3-agw`, which you can use for comparison's sake.
- A stub version `proj3-stub.cc`, to get you started. Note that this presupposes a certain organization of the code. Feel free to do something different.

These should be copied to your working directory (which should be `~/private/os/proj3`).

Your solution should be a clean typescript consisting of the following commands, along with their output:

```
cat proj3.cc
g++ -o proj3 proj3.cc -Wall
proj3 data1
proj3 data2
```

Good luck!