CS330 : Assignment 3

Siddharth Agrawal (150716)
K Karthikeyan (150311)
Shubham Kumar Bharti (150702)

November 15, 2017

Statistics

Table 1: Statistics for `vmtest1.c`

Algorithm		16	32	64	128	256	512
Random	# of Page Faults	2355	1914	1510	983	522	376
	Total Ticks	3925393	3449419	3011185	2439218	1941773	1783055
FIFO	# of Page Faults	2381	2055	1732	1247	381	376
	Total Ticks	3960705	3602670	3253695	2726675	1788607	1783055
LRU	# of Page Faults	2021	1896	1663	1225	379	376
	Total Ticks	3566215	3429717	3174593	2704757	1786250	1783055
LRU-Clock	# of Page Faults	2021	1897	1666	1225	378	376
	Total Ticks	3566215	3431293	3183153	2704757	1785174	1783055

Table 2: Statistics for `vmtest2.c`

Algorithm		16	32	64	128	256	512
Random	# of Page Faults	762	645	565	492	415	377
	Total Ticks	1256516	1129308	1044029	964395	879881	837200
FIFO	# of Page Faults	732	657	596	515	381	377
	Total Ticks	1226770	1141173	1075940	987655	843385	837200
LRU	# of Page Faults	632	613	578	509	380	377
	Total Ticks	1117514	1093304	1056365	981129	840672	837200
LRU-Clock	# of Page Faults	632	613	577	509	380	377
	Total Ticks	1117514	1093304	1055439	981129	840672	837200

Explanation for the above results

vmtest1.c

- This program consists of two nested loops for each of the four arrays.
- For convenience, we can divide each array into 3 parts (say - (I),(II) and (III)).
- Let us assume that each of these parts require one page (note:- since one page's size is taken as 128 and the array size is a multiple of 256, hence each part will have integral multiple pages). Assume that we have 2 page frames.
- Consider array1:
  - Outer loop is iterating 4 times.
  - Here the sequence of execution of the parts is - (I) (II) (I) (III) (I) (II) (I) (III) (I) (II) (I) (III) (I) (II) (I) (III)
  - Note that part (I) is executed between parts (II) and (III) every time. Since FIFO replaces the *firstly added* page, hence the replacements are maximized in this program.

  - For LRU, it replaces the *least recently* used page. Hence the page replacements are minimized.
  - LRU-Clock works comparable to LRU.
  - Random behaves randomly.

- array 2, 3 & 4 behave similar to array 1

vmtest2.c

- We can analyze this program similar to `vmtest1.c`,.i.e., divide each array into 3 parts (say - (I),(II) and (III)) and carry out the page assumption as given above.
- Consider array1: (Note array 2, 3 & 4 work similarly)
  - No nested loops.
  - FIFO may replace the pages which have been most recently used. Hence it experiences more page faults compared to LRU and LRU-Clock.
  - Here the sequence of execution of the parts is - (I) (II) (I) (III) (II)
  - Notice that this sequence gives us an intuition that FIFO should work best, but still it performs worse than LRU (and LRU-Clock) since FIFO is repeatedly replacing the code-block page which causes the page faults to swell.
  - The above paradox is more evident when the available page frames are less.

Changes in Command Line Input

Since we were required to change the number of physical pages available (`NumPhysPages`) repeatedly, hence we removed `NumPhysPages` from `#define` in `machine.h` and made `NumPhysPages` as a global variable initialized to 1024 (we added global `NumPhysPages` in `system.cc` and `externed` it in `system.h`).
To allow the user to input `NumPhysPages` from command line, we added an extra field with `-R` flag which will specify the number of physical pages available in main memory.

Hence, the user has to run **NachOS** from `nachos/code/userprog/` directory as given below:
`> ./nachos -rs s -A n -P p -R r N -x ../test/executablename`
All the flags (except `-R`) behave normally.
In flag `-R`, r denotes the algorithm number and N denotes the number of physical pages available (`NumPhysPages`).