Submit response?

Your username (**hermann@pdx.edu**) and responses will be recorded when you submit this form.

SWITCH ACCOUNT SUBMIT

Assignment: page replacement simulation

Work alone or with a partner.

Do the Chapter 22 homework set. See below for more details and adjustments. Specifically, problems 4 and 5 have been altered significantly.

The name, username and photo associated with your Google account will be recorded when you upload files and submit this form. Not **hermann@pdx.edu**? Switch account

* Required

Your Name *

Hermann Yepdjio

Partner's Name (if you worked with a partner)

N.A

Find the paging-policy.py simulator within the book's code directories. Read its README.md file and run it with the --help switch so that you understand the tool's features. *

[✓ check here after you have read the README.md file and studied the --help output]

Complete Problem #1. How did the results change as you varied the page replacement policy from FIFO, to LRU, to OPT? *

- OPT seems to always produce the best performance (lowest number of misses)
- LRU always produces the same or slightly better results than FIFO
- When the seed = 2, all 3 policies resulted in 4 hits and 6 misses.

Complete Problem #2. List your worst-case page reference streams here (for each of FIFO, LRU and MRU). Then iteratively increase the size of the cache to find the cache size(s) where each algorithm's performance improves and approaches the performance of OPT. How much bigger of a page cache is needed to improve performance dramatically and approach OPT? *

FIFO: addresses=0,1,2,3,4,5,6,0,1,2. Cache = 7 -To reach performance of OPT (7 misses)
LRU: addresses=0,1,2,3,4,5,6,0,1,2. Cache = 7 -To reach performance of OPT (7 misses)
MRU: addresses=0,1,2,3,4,5,6,5,6,5 . Cache = 7 -To reach performance of OPT (7 misses)

Complete Problem #3. No need to write your own code for this. To generate a random memory reference trace just use the "-a -1" option with varying values for the -n, -m, -s and -C options. Use -n values of 1000 or more. How do the various paging algorithms (the -p switch) compare with each other? Summarize your results here. to perform on such a trace? *

LRU and FIFO seem to always produce similar results
MRU seem to be slightly better than both LRU and FIFO when the cache size is small and slightly worse when cache size increases
OPT seems to alway produce the best results by far.

Complete Problem #4. If you prefer, instead of generating a trace "with some locality", repeat your runs from Problem #3 with 4 or 5 seed values (i.e., vary the input to the -s option). Does the relative performance of the paging algorithms change significantly from case to case? *

Changing the seed values affects the performance of the paging algorithms but not by much. In difference in the number of misses

Analyze Real Memory Reference Trace

On a linux system, run "valgrind --tool=lackey --trace-mem=yes <command>" (replace <command> with a linux command of your choosing. if you don't have one then use something like "find /tmp -type f" which finds all of the files within the /tmp directory on your linux system). This will generate a real address reference trace for your <command>. Beware that real memory traces can be huge, so use a <command> that is relatively quick; you will save yourself space and time. valgrind --tool=lackey writes its output to stderr, so you need to use 2> valgrind.trc to capture the output.

then convert your valgrind trace to a list of page numbers suitable for use with paging-policy.py. Use the provided python tool "vgxlate.py" to translate valgrind's address trace to a page trace. Run it like this: "./vgxlate.py < valgrind.trc > pages.trc"

Then run "./paging-policy.py -f pages.trc ..." to use your trace with paging-policy.py

Here is a full example:

```
valgrind --tool=lackey --trace-mem=yes find /tmp -type f 2> valgrind.trc
./vgxlate.py < valgrind.trc > pages.trc
./paging-policy.py -f pages.trc -p CLOCK -b 2 -N -c -C 3
```

Problem #5: how large is your real memory trace? (hint: use the wc command on your valgrind.trc file) *

409072

Problem #5. using your real trace, how big of a page cache is needed for your application trace in order to satisfy 98% of requests? Plot a graph with X axis as the cache size (from size 1 to whatever it takes to reach 98% hit ratio) and y axis as the hit ratio. The cache size at which the hit ratio reacbes 98% is a good estimate for the "working set size" of the process. Do this for each of the algorithms LRU, CLOCK and OPT. For CLOCK use '-b 2'. Upload your graph here. *

plot_for_different...

Problem #5 Continued: what observations do you have about the relative performance of the page replacement policies when run on your real memory reference trace? *

- When cache size = 1, all the policies perform at hit-rate = 51.
- This rate increases as the cache size increase
- However the rate of OPT increases faster and reaches 98% when cache size = 7
- Next is LRu which reaches 98% hit-rate when cache size = 9
- Finally Clock reaches 98% hit-rate when cache size = 10

Submit