**Tutorial 7** **CS3103 Operating Systems**

Student Name: ____________________
Student No. : | | | | | | | | |

Day: ☐ Monday ☐ Tuesday
Time: ☐ 10:00 - 10:50 ☐ 11:00 - 11:50 ☐ 14:00 - 14:50 ☐ 16:00 - 16:50 ☐ 18:00 - 18:50

# Page Replacement Policy

## Introduction

Topics to be covered in this tutorial include:

- Evaluate replacement algorithms by running them on a particular string of memory references (reference string) and computing the number of page faults on that string.

## Acknowledgement

This tutorial was adapted from OSTEP book written by Remzi and Andrea Arpaci-Dusseau at the University of Wisconsin. This free OS book is available at http://www.ostep.org.

## Getting Started

### 1. Logging in to the Linux server

- Start the SSH client, e.g., MobaXterm or Xshell.
- Log in to the Linux server using the following details:

| | |
|---|---|
| Host Name: | gateway.cs.cityu.edu.hk |
| User Name: | your EID (e.g., cctom2) |
| Password: | your password |

**Your password will not be shown on the screen as you type it, not even as a row of stars (******).**

**NOTE:** The shell will always give you a prompt if it is ready to accept commands. The shell prompt normally ends in a `$` sign as we use in this tutorial. Some shell prompts end in `%` or `>` instead. Never copy/type the shell prompt used in this tutorial.

**NOTE:** Please don't forget to log out (use the `exit` command) after you finish your work.

### 2. Getting the Simulators

This tutorial lets you explore how simple virtual-to-physical address translation works with a linear page table or a multi-level page table. The simulators are available at */public/cs3103/tutorial7* on the gateway server:

- `paging-policy.py`: allows you to play around with different page-replacement policies.

As before, follow the same copy procedure as you did in the previous tutorials to get the simulators.

## Introduction to paging-policy.py

This simulator, paging-policy.py, allows you to play around with different page-replacement policies. For example, let's examine how LRU performs with a series of page references with a cache of size 3:

0 1 2 0 1 3 0 3 1 2 1

To do so, run the simulator as follows:

```
$ ./paging-policy.py --addresses=0,1,2,0,1,3,0,3,1,2,1 --policy=LRU --cachesize=3 -c
```

And what you would see is:

```
ARG addresses 0,1,2,0,1,3,0,3,1,2,1
ARG addressfile
ARG numaddrs 10
ARG policy LRU
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

Access: 0  MISS LRU ->          [0] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 1  MISS LRU ->       [0, 1] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 2  MISS LRU ->    [0, 1, 2] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 0  HIT  LRU ->    [1, 2, 0] <- MRU Replaced:- [Hits:1 Misses:3]
Access: 1  HIT  LRU ->    [2, 0, 1] <- MRU Replaced:- [Hits:2 Misses:3]
Access: 3  MISS LRU ->    [0, 1, 3] <- MRU Replaced:2 [Hits:2 Misses:4]
Access: 0  HIT  LRU ->    [1, 3, 0] <- MRU Replaced:- [Hits:3 Misses:4]
Access: 3  HIT  LRU ->    [1, 0, 3] <- MRU Replaced:- [Hits:4 Misses:4]
Access: 1  HIT  LRU ->    [0, 3, 1] <- MRU Replaced:- [Hits:5 Misses:4]
Access: 2  MISS LRU ->    [3, 1, 2] <- MRU Replaced:0 [Hits:5 Misses:5]
Access: 1  HIT  LRU ->    [3, 2, 1] <- MRU Replaced:- [Hits:6 Misses:5]

FINALSTATS hits 6   misses 5   hitrate 54.55
```

The complete set of possible arguments for paging-policy is listed on the following page, and includes a number of options for varying the policy, how addresses are specified/generated, and other important parameters such as the size of the cache.

To do so, run the simulator as follows:

```
$ ./paging-policy.py -help
Usage: paging-policy.py [options]

Options:
  -h, --help            show this help message and exit
  -a ADDRESSES, --addresses=ADDRESSES
                        a  set  of  comma-separated  pages  to  access;  -1  means
randomly generate
  -f ADDRESSFILE, --addressfile=ADDRESSFILE
                        a file with a bunch of addresses in it
  -n NUMADDRS, --numaddrs=NUMADDRS
                        if -a (--addresses) is -1, this is the number of addrs
to generate
  -p POLICY, --policy=POLICY
                        replacement policy: FIFO, LRU, OPT, UNOPT, RAND, CLOCK
  -b CLOCKBITS, --clockbits=CLOCKBITS
                        for CLOCK policy, how many clock bits to use
  -C CACHESIZE, --cachesize=CACHESIZE
                        size of the page cache, in pages
  -m MAXPAGE, --maxpage=MAXPAGE
                        if randomly generating page accesses, this is the max
page number
  -s SEED, --seed=SEED  random number seed
  -N, --notrace         do not print out a detailed trace
  -c, --compute         compute answers for me
```

As usual, "-c" is used to solve a particular problem, whereas without it, the accesses are just listed (and the program does not tell you whether or not a particular access is a hit or miss).

To generate a random problem, instead of using "-a/--addresses" to pass in some page references, you can instead pass in "-n/--numaddrs" as the number of addresses the program should randomly generate, with "-s/--seed" used to specify a different random seed. For example:

```
$ ./paging-policy.py -s 10 -n 3
ARG addresses -1
ARG addressfile
ARG numaddrs 3
ARG policy FIFO
ARG clockbits 2
ARG cachesize 3
ARG maxpage 10
ARG seed 10
ARG notrace False

Assuming a replacement policy of FIFO, and a cache of size 3 pages,
figure out whether each of the following page references hit or miss
in the page cache.

Access: 5  Hit/Miss?  State of Memory?
Access: 4  Hit/Miss?  State of Memory?
Access: 5  Hit/Miss?  State of Memory?
```

As you can see, in this example, we specify "-n 3" which means the program should generate 3 random page references, which it does: 5, 7, and 5. The random seed is also specified (10), which is what gets us those particular numbers. After working this out yourself, have the program solve the problem for you by passing in the same arguments but with "-c" (showing just the relevant part here):

```
$ ./paging-policy.py -s 10 -n 3 -c
......
Solving...

Access: 5  MISS FirstIn ->          [5] <- Lastin  Replaced:- [Hits:0 Misses:1]
Access: 4  MISS FirstIn ->       [5, 4] <- Lastin  Replaced:- [Hits:0 Misses:2]
Access: 5  HIT  FirstIn ->       [5, 4] <- Lastin  Replaced:- [Hits:1 Misses:2]

FINALSTATS hits 1   misses 2   hitrate 33.33
```

The default policy is FIFO, though others are available, including LRU, MRU, OPT (the optimal replacement policy, which peeks into the future to see what is best to replace), UNOPT (which is the pessimal replacement), RAND (which does random replacement), and CLOCK (which does the clock algorithm). The CLOCK algorithm also takes another argument (-b), which states how many bits should be kept per page; the more clock bits there are, the better the algorithm should be at determining which pages to keep in memory.

Other options include: "-C/--cachesize" which changes the size of the page cache; "-m/--maxpage" which is the largest page number that will be used if the simulator is generating references for you; and "-f/--addressfile" which lets you specify a file with addresses in them, in case you wish to get traces from a real application or otherwise use a long trace as input.

## Questions

**All questions should be answered on the separate answer sheet provided.**

1. Generate random addresses with the following arguments: -s 0 -n 10, -s 1 -n 10, and -s 2 -n 10. Change the policy from FIFO, to LRU, to OPT. Compute whether each access in said addresses is hit or miss.

2. For a cache of size 3, generate worst-case address reference streams (the number of addresses is 10 and the maximum of addresses is 10) for each of the following policies: FIFO, LRU, and MRU. A worst-case reference stream for a specific policy, for example FIFO, causes the largest (among FIFO, LRU, and MRU) number of misses when using this policy. For the worst-case reference streams, how much bigger of a cache is needed to improve performance dramatically and approach OPT?

3. Use valgrind to instrument a real application and generate a virtual page reference stream. Running `./valgrind --tool=lackey --trace-mem=yes --log-file="trace.txt" ls` will output a nearly-complete reference trace of every instruction and data reference made by the program ls (try different commands, such as `cp, mkdir, cd` and etc.) to trace.txt file. This step may take several minutes. Each item in trace.txt is like `I 05695e5c,2`. To make this useful for the simulator above, you'll have to first transform each virtual memory reference into a virtual page-number reference. Recall that a virtual address consists of the virtual page number and the offset in the page. To get the virtual page number, we need to mask off the offset and shift the resulting bits downward. The page size is usually 4KB (You can check it by running `getconf PAGESIZE`), thus the offset needs 12 bits. Therefore, the rightmost three digits are the offset and the first five (or more) digits represent the virtual page number (for 05695e5c in the above example, the virtual page number is 05695 in hexadecimal). We provide you a small program to convert the trace into an address file (`./va2vpn trace.txt`). Address files can be directly run by the simulator (use `-f` to specify a file with addresses in it when running the simulator). How big of a cache is needed for your application trace in order to satisfy a large fraction of requests (for example, guarantee the hit ratio higher than 95%, 98% and 99%, respectively)? Use -C to change the size of page cache and -N to only print the final hit and miss statistics without the detailed hit/miss information for each address. To make it clear, run the following two sets of commands. You are encouraged to collect the traces of more commands and compare their differences. Also, compare the performance among different policies: FIFO, LRU, OPT and MRU. How does the difference change when you increase the size of page cache?

```
$ ./valgrind --tool=lackey --trace-mem=yes --log-file="trace-ls.txt" ls
$ ./va2vpn trace-ls.txt
$ ./paging-policy.py -f trace-ls.txt –c –N –C 3 -p FIFO
```

```
$  ./valgrind  --tool=lackey  --trace-mem=yes  --log-file="trace-cp.txt"  cp  paging-
policy.py paging-policy-copy.py
$ ./va2vpn trace-cp.txt
$ ./paging-policy.py -f trace-cp.txt –c –N –C 3 -p FIFO
```