CS 45: Operating Systems

CLab 7: swapping

DUE by next class

Write your answers in `README.md`. Create a `COLLAB.md` file to keep track of any outside resources you might use. Be sure to push to the repo after class (even if you are not done).

1 Part 1: Translation Practice

NOTE: Feel free to use `python3` to do some base conversions and bitwise operations, the `hex`, `bin`, and `format` functions are very useful, as are, the syntax for binary and hexadecimal literals, for example, `0b11010` and `0x1C3`.

```
$ python3
>>> bin(125)
'0b1111101'
>>> hex(125)
'0x7d'
>>> bin(0x7d)
'0b1111101'
>>> format(125, "#x")
'0x7d'
>>> format(125, '#b')
'0b1111101'
>>> format(125, '#010b')                  # show all 8-bits
'0b01111101'
>>> format((125 >> 4), '#010b')           # shift to the right by 4 to get high nibble
'0b00000111'
>>> format((125 & 0b11110000), '#010b')   # mask out the lower nibble
'0b01110000'
```

1. Imagine a virtual memory system that relies on **segmentation** and a `ten-bit virtual address`. Assume the `2 most-significant bits` are used to determine which segment (`0b00`: code, `0b01`: heap, `0b11`: stack), and the remaining `8 bits` represent the offset.
   a. How many addresses are in this address space?
   b. For all three virtual addresses, identify which **segment** the virtual address lives in, and translate that address to its physical location, given the base addresses of the segments.

```
CODE BASE:  0x9000          MAX SIZE: 0x100
HEAP BASE:  0xA000          MAX SIZE: 0x100
STACK BASE: 0x7000          MAX SIZE: 0x100

VA = 0x42    SEGMENT =      PA =
VA = 0x303   SEGMENT =      PA =
VA = 0x1C3   SEGMENT =      PA =
```

2. Imagine a virtual memory system that relies on **paging** using 128-byte pages. Assume the `3 most-significant` bits are used to determine the virtual page number, and the remaining `7 bits` represent the offset.
   a. How many addresses are in this address space?
   b. Given the following page table below, translate the addresses to their physical location.
   c. The physical RAM is at least how big?

```
-----------------
|  VPN  |  PFN  |
----------------|
|  0x0  |  0x0F |
|  0x1  |  0x01 |
|  0x2  |  0x03 |
|  0x3  |  0x04 |
|  0x4  |  0x02 |
|  0x5  |  0x09 |
|  0x6  |  0x0A |
|  0x7  |  0x0C |
|  0x8  |  0x05 |
-----------------

VA = 0x42     PA =
VA = 0x303    PA =
VA = 0x1C3    PA =
```

2 Part 2: Swapping

Use the program `paging-policy.py`, to see if you how different replacement policies work.

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 address traces are hits or misses.
2. For a cache of size 5, generate worst-case address reference streams for each of the following policies: FIFO, LRU, and MRU (worst-case reference streams cause the most misses possible. For the worst case reference streams, how much bigger of a cache is needed to improve performance dramatically and approach OPT?
3. Generate a random trace (use python or perl). How would you expect the different policies to perform on such a trace?

```
$ shuf -i 1-100 -n 20 > trace.txt
```

4. Now generate a trace with some locality. How can you generate such a trace? How does LRU perform on it? How much better than RAND is LRU? How does CLOCK do? How about CLOCK with different numbers of clock bits?
5. Use a program like `valgrind` to instrument a real application and generate a virtual page reference stream. For example, running `valgrind --tool=lackey --trace-mem=yes ls` will output a nearly-complete reference trace of every instruction and data reference made by the program `ls`. What do you think the `S`, `L`, and `I` markers are?

```
$ valgrind --tool=lackey --trace-mem=yes --log-file=out.log ls
$ less out.log
```

6. To make this useful for the simulator above, we'll have to first transform each virtual memory reference into a virtual page-number reference (done by shifting the resulting bits downward). How big of a cache is needed for your application trace in order to satisfy a large fraction of requests? Plot a graph of its working set as the size of the cache increases.

```
$ awk 'match($0, /([0-9a-fA-F]+),/, a) { print rshift(strtonum("0x" a[1]), 12) }' out.log > trace.txt
```

NOTE: The `W` in `AWK` is Swat alum Peter Weinberger.