Measuring the Performance of Page Replacement Algorithms on Real Traces

Learning Objective:

- Understand the costs of page replacement algorithms
- Observe the effect of memory traces and the effect of page replacement algorithms on system performance.

In this project, you are to evaluate how real applications respond to a variety of page replacement algorithms. For this, you are to write a memory simulator and evaluate memory performance using provided traces from real applications. Like all other projects, project 2 is an individual project and no collaboration is allowed.

Memory Traces

You are provided with memory traces to use with your simulator. Each trace is a real recording of a running program, taken from the SPEC benchmarks. Real traces are enormously big: billions and billions of memory accesses. However, a relatively small trace will be more than enough to keep you busy. Each trace only consists of one million memory accesses taken from the beginning of each program. The traces are the following (linked from the Canvas project): 1. bzip.trace 2. sixpack.trace
Each trace is a series of lines, each listing a hexadecimal memory address followed by R or W to indicate a read or a write. For example:

```
0041f7a0 R
13f5e2c0 R
05e78900 R
004758a0 R
31348900 W
```

Note, to scan in one memory access in this format, you can use fscanf() as in the following:

```
 unsigned addr; char
      rw;
      ...
      fscanf(file,"%x %c",&addr,&rw);
```

Simulator Requirements

Your job is to build a simulator that reads a memory trace and simulates the action of a virtual memory system with a single level page table. Your simulator should keep track of what pages are loaded into memory. As it processes each memory event from the trace, it should check to see if the corresponding page is loaded. If not, it should choose a page to remove from memory. If the page to be replaced is "dirty" (that is, previous accesses to it included a Write access), it must be saved to disk. Finally, the new page is to be loaded into memory from disk, and the page table is updated. Assume that all pages and page frames are 4 KB (4096 bytes).

Of course, this is just a simulation of the page table, so you do not actually need to read and write data from disk. Just keep track of what pages are loaded. When a simulated disk read or write must occur, simply increment a counter to keep track of disk reads and writes, respectively.

Implement the following page replacement algorithms to replicate the experimental evaluation in this 1981 paper1[1]:

1. FIFO
2. LRU
3. Segmented FIFO

Structure and write your simulator in any reasonable manner. You may need additional data structures to keep track of which pages need to be replaced depending on the algorithm implementation. Think carefully about which data structures you are going to use and make a reasonable decision.

[1] *Rollins Turner and Henry Levy. 1981. Segmented FIFO page replacement. In Proceedings of the 1981 ACM SIGMETRICS conference on Measurement and modeling of computer systems (SIGMETRICS '81). Association for Computing Machinery, New York, NY, USA, 48-51. DOI:https://doi.org/10.1145/800189.805473*

You need to follow strict requirements on the interface to your simulator so that we will be able to test and grade your work in an efficient manner. The simulator (called `memsim`) should take the following arguments:

```
memsim <tracefile> <nframes> <lru|fifo|vms> <debug|quiet>
```

The first argument gives the name of the memory trace file to use (thus, <tracefile> can be any file in the format described above and in the trace files provided. The second argument gives the number of page frames in the simulated memory. The third argument gives the page replacement algorithm to use. The fourth argument may be "debug" or "quiet" (explained below.)

If the fourth argument is "quiet", then the simulator should run silently with no output until the very end, at which point it should print out a few simple statistics like this (follow the format as closely as possible):

```
total memory frames: 12 events
in trace: 1002050 total disk
reads: 1751 total disk writes:
932
```

If the fourth argument is "debug", then the simulator should print out messages displaying the details of each event in the trace. You may use any format for this output, it is simply there to help you debug and test your code.

If the third argument is "vms", you have to take an extra parameter that indicates the percentage of the total program memory to be used in the secondary buffer. In that case, the simulator should take the following arguments:

```
memsim <tracefile> <nframes> <lru|fifo|vms> <p>
<debug|quiet>
```
"p" will be a number between 1 to 100.

Use separate functions for each page replacement algorithm, i.e., your program must declare the following high level functions: `fifo()`, `lru()`, `segmented-fifo()`.