CS 481/ECE 437 Lab 2: MMU Simulator

Due Tuesday, March 22 at 08:00

- **This assignment is to be done individually.**
- You may discuss this lab with others but you may not share code; each group must turn in their own original work.
- You are allowed to use pre-existing code (of your creation or otherwise) for container data structures only, i.e. linked lists, hash tables, search trees, etc. However, these codes must be kept in separate files, and your comments and write-up.txt file must clearly credit the source of these files. When in doubt, ask!
- For this lab, your may use any programming language you choose. **If you plan to use a language other than C, C++ or Java, consult with the professor.**

Concepts Page Tables, TLBs and Memory Traces

Overview

In this lab, you will implement a memory management unit (MMU) that simulates virtual to physical address translations. Your MMU will simulate access to a TLB, a 1-level page table, main memory and a disk that acts as a backing store for main memory. You will be implementing and evaluating various page replacement strategies.

The TLB The fully-associative TLB caches both data and instruction address translations. The TLB does not support a PID tag, which would allow you to distinguish per process entries; therefore, it must be flushed on each context switch (i.e. when you start processing trace addresses from a different process). The TLB should use a LRU replacement policy.

Paging You will simulate a 1-level, page table per process scheme for a 32-bit address space. You will simulate pure demand paging – no page will be brought into main memory until it is requested. For page replacement, your simulation will support the following schemes:

- first in, first out: pages will be evicted in the order they were brought into main memory
- least recently used: least recently used pages will be evicted first
- least frequently used: least accessed pages will be evicted first
- most frequently used: most accessed pages will be evicted first
- random: randomly select a page to evict

You should use a global page replacement policy: a process can evict another process' page and claim its memory frame.

The MMU Your mmu will be invoked as follows:

```
prompt> mmu <config_file> <trace_file>
```

- `config_file`: name of file containing simulation parameters
- `trace_file`: name of file containing trace of virtual addresses

After parsing the simulation parameters from the configuration file, the mmu reads and simulates the translations of the virtual addresses from the trace file.

The Configuration File

The configuration file has 8 lines as follows:

1. `physical-memory-size`: main memory size in bytes (always a power of 2).
2. `frame-size`: physical frame size in bytes. (always a power of 2.)
3. `memory-latency`: time in nanoseconds to read/write main memory page.
4. `page-replacement`: page replacement policy. Valid values are "RANDOM", "FIFO", "LRU", "LFU", and "MFU".
5. `tlb-size`: number of entries in the TLB (0 means no TLB).
6. `tlb-latency`: time in nanoseconds to read/write a TLB entry.
7. `disk-latency`: time in milliseconds to read/write a page to/from disk.
8. `logging-output`:"on" or "off"– whether or not to generate logging output.

An example configuration file is:

```
physical-memory-size: 1073741824
frame-size: 1024
memory-latency: 100
page-replacement: LRU
tlb-size: 128
tlb-latency: 20
disk-latency: 10
logging-output: off
The Virtual Address Trace File
```

The Memory Reference Trace File

The trace file has one line per virtual address in the following format: `<pid> <op> <address>` where:

- pid: is the process id of the relevant process

- op: is the operation the process is executing. Valid values are:
  - 'R': for a read from main memory (load)
  - 'W': for a write to main memory (store)
  - 'I': for an instruction fetch from main memory (load)
- address: the 32-bit hexadecimal address being referenced by the process.

Example trace file:

```
24740 I 0x5cb810
24740 I 0x5cb812
24740 W 0xbf97220c
24740 I 0x5cbf60
24740 W 0xbf972208
24740 I 0x5cbf61
24740 I 0x5cbf63
24740 W 0xbf972204
24740 I 0x5cbf64
24740 W 0xbf972200
24740 I 0x5cbf65
24740 W 0xbf9721fc
24740 I 0x5cbf66
24740 I 0x5cbf69
24740 W 0xbf9721bc
24740 I 0x5e0e9b
24740 R 0xbf9721bc
24747 I 0x5cb810
24747 I 0x5cb812
24747 W 0xbf97220c
24747 I 0x5cbf60
24747 W 0xbf972208
24747 I 0x5cbf61
24747 I 0x5cbf63
24747 W 0xbf972204
24747 I 0x5cbf64
24747 W 0xbf972200
24747 I 0x5cbf65
24747 W 0xbf9721fc
24747 I 0x5cbf66
24747 I 0x5cbf69
24747 W 0xbf9721bc
24747 I 0x5e0e9b
24747 R 0xbf9721bc
```

The Simulation Output

This program will be graded by computer scripts. It is important that the format of your simulation output matches the specification below.

Your simulation will produce two types of output: *summary output* and *logging output*. Summary output is printed to `stdout`, and `logging output` is printed to stderr when logging is turned on. (No logging output should be produced when logging is turned off.)

Summary Output Summary output is printed to stdout at the beginning and end of the simulation as specified below.

Pre-Simulation Output:

1. Page bits: number of high-order bits of the 32-bit address used for the page number
2. Offset bits: number of low-order bits of the 32-bit address used for the page offset
3. TLB size: the number of entries in the TLB.
4. TLB latency (milliseconds): time to read/write the TLB (to 6 decimal places)
5. Physical memory (bytes): the size of main memory.
6. Physical frame size (bytes): the size of a frame of main memory.
7. Number of physical frames: the number of frames in main memory
8. Memory latency (milliseconds): time to read/write to main memory (to 6 decimal places)
9. Number of page table entries: the number of entries in each process' page table
10. Page replacement strategy: the page replacement strategy
11. Disk latency (milliseconds): time to read/write to disk (to 2 decimal places).
12. Logging: whether logging is turned on or off.

Sample pre-simulation output:

```
Page bits: 22
Offset bits: 10
TLB size: 128
TLB latency (milliseconds): 0.000020
Physical memory (bytes): 1073741824
Physical frame size (bytes): 1024
Number of physical frames: 1048576
Memory latency (milliseconds): 0.000100
Number of page table entries: 4194304
Page replacement strategy: RANDOM
Disk latency (milliseconds): 10.00
Logging: on
```

Post-Simulation Output:

1. Overall latency (milliseconds): overall latency to execute the trace (to 6 decimal places)
2. Average memory access latency (milliseconds/reference): average memory access latency (to 6 decimal places)
3. Slowdown: ratio of average memory access latency to ideal memory access latency, i.e. without translation or page fault overheads.

Followed by the following statistics first for the entire simulation, and then per each process in the simulation:

4. Overall/Process : "Overall" for entire simulation statistics, or "Process " for per process statistics
5. Memory references: number of memory references
6. TLB misses: number of TLB misses
7. Page faults: number of page faults
8. Clean evictions: number of clean pages evicted from main memory
9. Dirty evictions: number of dirty pages evicted from main memory
10. Percentage dirty evictions: percentage of dirty pages evicted from main memory

Sample post-simulation output:

```
Overall latency (milliseconds): 20.012240.
Average memory access latency (milliseconds/reference): 0.200122.
Slowdown: 2001.22.

Overall
        Memory references: 10000
        TLB misses: 24
        Page faults: 24
        Clean evictions: 0
        Dirty evictions: 0
        Percentage dirty evictions: 0.00%

Process 24740
        Memory references: 5000
        TLB misses: 18
        Page faults: 18
        Clean evictions: 0
        Dirty evictions: 0
        Percentage dirty evictions: 0.00%

Process 24747
        Memory references: 5000
        TLB misses: 6
        Page faults: 6
        Clean evictions: 0
        Dirty evictions: 0
        Percentage dirty evictions: 0.00%
```

Per process reports should be in order of increasing pids.

Logging Output

When logging is turned on, a trace of simulated events is printed to `STDERR` during the simulation.

During the simulation, you will log each address translation that you simulate. For example, the page number and offset of the address, whether it was found in the TLB, whether it generated a page fault, whether it generated the eviction of another page, and the frame to which the reference page was assigned. **The following lines should be printed in the given order as appropriate**:

1. "Process[%u]: %s 0x%x (page: %u, offset: %u)\n": **(Always print this line.)**
    - %u: process id of relevant process
    - %s: "Store to", "Load from" or "Instruction fetch from" as appropriate
    - %x: virtual address
    - %u: virtual address page number
    - %u: virtual address page offset
2. "\tTLB hit? %s\n": **(Always print this line.)**
    - %s: "yes" or "no" as appropriate
3. "\Page fault? %s\n": (Only print on TLB misses)
    - %s: "yes" or "no" as appropriate
4. "\Main memory eviction? %s\n": (Only print on page faults)
    - %s: "yes" or "no" as appropriate
5. "\Process %u page %u (%s) evicted from memory\n": (Only print on main memory evictions)
    - %u: process id of relevant process
    - %u: virtual address page number
    - %s: "clean" or "dirty" as appropriate
6. "\TLB eviction? %s\n": (Only print on TLB miss)
    - %s: "yes" or "no" as appropriate
7. "\page %u evicted from TLB\n": (Only print on TLB eviction)
    - %u: virtual address page number
8. "\page %u in frame %u\n": (Always print this line)
    - %u: virtual address page number
    - %u: memory frame into which page is loaded

Sample logging output:

```
Process[0]: Store to 0x0 (page: 0, offset: 0)
        TLB hit? no
        Page fault? yes
        Main memory eviction? no
        TLB eviction? no
        page 0 in frame 0
Process[0]: Load from 0x200 (page: 0, offset: 512)
        TLB hit? yes
        page 0 in frame 0
Process[0]: Load from 0x800 (page: 2, offset: 0)
        TLB hit? no
        Page fault? no
        TLB eviction? yes
        page 0 evicted from TLB
        page 2 in frame 2
```

```
Process[0]: Load from 0xc00 (page: 3, offset: 0)
        TLB hit? no
        Page fault? yes
        Main memory eviction? no
        TLB eviction? yes
        page 1 evicted from TLB
        page 3 in frame 3
Process[0]: Load from 0x1000 (page: 4, offset: 0)
        TLB hit? no
        Page fault? yes
        Main memory eviction? yes
        Process 0 page 3 (clean) evicted from memory
        TLB eviction? no
        page 4 in frame 3
```

Guides & Tips

- The performance of your code will be important. In choosing container types, for your TLB and page tables, carefully consider their potential sizes, as well as the frequency with which they will be searched or have elements inserted or deleted. Your traces may have millions of address look-ups to simulate, and the performance of your data structures will determine if you program takes seconds, minutes or hours to run.
- Initially, You should validate your simulation using parameters that allow you to manually evaluate what is happening. For example, you should be able to easily track a TLB and page tables with on the order of 10 entries. During the debugging phase, consider printing out the contents of a page table or the TLB each time it is accessed.
- The ”
- You will find C's bit right shift operator, ¿¿, and logical and operator, &, handy in partitioning the 32-bit addresses into a page number and page offset.
- When computing total overall latency, the following should be included in your calculation:
  - TLB look-up
  - Page table look-ups
  - Writes of dirty pages to disk
  - Loads of faulting pages from disk
  - Page table updates
  - TLB updates
  - Actual memory reference
- You may find the following references useful:
  - trace_gen.c: A simple C program to generate more controlled virtual address traces. Usage:
    - –num-references, -r:
    - –num-processes, -p: number of processes in trace
    - –max-process-size, -m: the maximum size of each process' address space
    - –reference-pattern, -f: virtual address memory reference pattern (”sequential” or ”random”)
    - –mem-stride, -s: stride (number of skipped bytes) for sequential access
    - –rw-ratio, -a: ratio of memory reads to memory writes in trace

    - –time-slice, -t: number of references per process time slice
  - vaddr_trace.cpp: A PIN tool for collecting virtual address traces
  - SGI's STL Programmer's Guide
  - Boost C++ Libraries
  - dllist.h and dllist.c: Plank's doubly-linked list implementation. Documentation can be found here
  - jval.h and jval.c: Plank's generic data type implementation. Documentation can be found here
  - jrb.h and jrb.c: Plank's red-black tree implementation. Documentation can be found here

You can download all the referenced files from: `lab3_files.tgz`.

What to turn in

YOU MUST FOLLOW THESE INSTRUCTIONS PRECISELY.

You should turn in the following:

- **all** .c and .h files needed to build your executable files, including any you've downloaded. **Do not turn in any object files (.o) or binary executable files.**
- the single `makefile` that builds the mmu program.

When you are ready to turn in your assignment:

- Place the requisite files in a directory named lastname_lab3 where lastname is your last name;
- Move to the parent directory that contains this lab3 directory;
- execute the command: `tar -czf <lab3_dir>.tgz <lab3_dir>`
  where <lab3_dir> is the name of the directory containing your lab 3 files.
- This will create a new file <lab3_dir>.tgz containing the contents of your <lab3_dir>. You can verify the contents of this "compressed tar file" with the following command: `tar -tzf <lab3_dir>.tgz`
- Submit the file `<lab3_dir>.tgz` via UNM Blackboard.