CMPT 300
Introduction to Operating Systems

Project 3 Explanations

Goal

- Implement 3 page replacement algorithms: FIFO, CLOCK and LRU, as discussed in class, in a virtual memory simulation program. (We do not use nachos in this project.)

Steps

- Download and compile vmsim in a directory called ~/prj3:

  $wget http://www.cs.sfu.ca/CourseCentral/300/zonghuag/projects/vmsim.tar.gz

  $ tar -xxvf vmsim.tar.gz
  $ cd vmsim
  $ make
  $ ./vmsim -h (will tell you how to run the simulator and the arguments/options it takes)

- Inspect the code carefully:
  - vmsim.c -- main function, some initialization code, and simulate(). simulate() reads one memory reference at a time from the input trace file, searches for it in the page table, invokes a page-fault handler if it is not found, and updates some statistics.
  - fault.c -- page fault handlers. Most of your code will be in this file. A 'random' replacement algorithm is given in this file as an example.
  - stats.c -- functions to collect statistics.
  - pagetable.c -- implementation of a page table with multiple levels.
  - physmem.c -- structure to represent physical memory along with some functions, e.g., load and evict.
  - options.c -- functions to process command line arguments.
  - util.c -- some utility functions.

- Test the random policy with some sample traces (e.g., example_trace.txt)

Your Task

- You need to implement 3 page replacement algorithms: FIFO, CLOCK and LRU. In fault.c, implement 3 functions
  - fault_fifo(), fault_clock(), fault_lru();
- We have provided implementation of fault_random() for randomly replacing any page, so you can play with it to understand the code.

Help

- Run “./vmsim –h” to get help:

  - Usage: vmsim [OPTIONS] ALGORITHM [TRACEFILE|-]
  - Process TRACEFILE, simulating a VM system. Reports stats on paging behavior.
  - If TRACEFILE is not specified or is '-', input will be taken from stdin.

  - ALGORITHM specifies the fault handler, and should be one of:
  - random, fifo, clock, lru, e_clock

  - Options:
  - -h Print this message and exit.
  - -V Print the version information.
  - -v Verbose output. Includes progress output.
  - -t Run self tests.
  - -o FILE Append statistics output to the given file.
  - -l REFS Stop after first REFS refs.
  - -p PAGES Simulate PAGES physical pages.
  - Minimum value 3.
  - -s SIZE Simulate a page size of SIZE bytes.
  - Size must be a power of 2.
- Example command: ./vmsim -p 3 -s 32 random example_trace.txt
  - Number of physical pages: 3; each page is 32 bytes; page replacement algorithm is random; input file is example_trace.txt

Debug Output

- In options.h, the line “#define DEBUG” defines a debug flag. If enabled, the program will print out step-by-step page table and physical memory dumps to help you debug your code.
- Remove it when you have finished debugging, or your page table entries have grown too many to be printed and viewed.

vsim.c

- In vmsim.c, the main function simulates the working of a virtual memory unit: It reads memory references (from a trace file) and checks whether this reference is in memory. If it is in memory, some statistics are updated. If is is not in memory, a page fault occurs and the page fault handler is invoked. The page fault handler uses the replacement policy to evict a page from the memory to make room for the requested page. Several replacement policies can be used with the simulator. You specify which replacement policy to be used when you execute the simulator (check the output of ./vmsim -h).

Input File Format

- Process ID, Mode (Read or Write), Virtual Address
- e.g. 4 memory references by process 1, all for “Read”:
  - 1, R, 0x0000E00
  - 1, R, 0x0000000
  - 1, R, 0x0000200
  - 1, R, 0x0000400
- Note:
  - We assume a single-process scenario, so “Process ID” is actually not useful to you. (In general, it is needed to distinguish between page tables of different processes).
  - “Mode” is mainly used for statistics collection, also not useful to you in this project.

pagetable.h

- //Default values that can be overwritten from the command line
- const static int pagesize = 4096;
- const static int log_pagesize = 12;

- typedef struct _pte {
- uint vfn; /* Virtual frame number */
- uint pfn; /* Physical frame number iff valid=1 */
- int reference;
- bool_t valid; /* True if in physmem, false otherwise */
- bool_t modified;
- int counter; /* used for LRU, FIFO */
- } pte_t;
- "physmem" is an array of page table entries (pte_t **physmem).

Example PT Config

- In vmsim.h, we hardcode 16-bit address space, i.e, vaddr (Virtual Address) has 16 bits,
  - const static uint addr_space_bits = 16;
- With this command "./vmsim -p 3 -s 32 random example_trace.txt", vaddr consists of higher 11 bits for vfn (Virtual Frame Number), and lower 5 bits for offset within each page (log_2(pagesize=32)). (Page table size is 2^{11})

vaddr→vfn Conversion

- The function in util.h grabs the higher-order vfn_bits from vaddress to form vfn
  - static inline uint vaddr_to_vfn(vaddr_t vaddress) {
  - return getbits(vaddress, addr_space_bits-1, vfn_bits);}
- Consider vaddr=0x0000E00 in hex (3584 in decimal). Convert this hex number to binary to get 111000000000. Padding it to 16 bits to get 0000111000000000. Take the higher 11 bits to get 00001110000, which is vfn=0x70 in hex (112 in decimal)
  - Online hex converter: http://easycalculation.com/hex-converter.php
- (This conversion is done for you. You do not need to care about the details.)

pagetable.c

- This code allows multi-level page tables, but with our constraints of 16-bit address space and min page size of 16 bytes (4 bits), we will never need more than 12 bits for vfn, hence we always have a single level page table.

```
typedef struct _pagetable_level {
  uint size;
  uint log_size;
  bool_t is_leaf;
} pagetable_level_t;

/* Define a multi-level page table.
 * This defines the largest size each level can be.*/
pagetable_level_t levels[3] = {
  { 4096, 12, FALSE }, /*levels[0] has 12 bits. If vfn_bits exceeds 12, then need additional levels*/
  { 4096, 12, FALSE }, /*levels[1] has 12 bits. If vfn_bits exceeds 24, then need additional levels*/
  { 256, 8, TRUE } /*levels[2] has 8 bits. So the maximum vfn_bits that can be handled is 12+12+8=32*/
};
typedef struct _pagetable {
  void **table; /* If lowest-level, array of pte_t pointers;
                 * otherwise array of pagetable_t pointers. */
  int level;
} pagetable_t;
static pagetable_t *root_table; /*root_table->table is the current page table. Use pte_t *pte=(pte_t *) root_table->table[i] to access each pte entry*/
```

pagetable.c

- A page fault happens when a virtual address is requested, but “pte->valid==FALSE”, either because the entry is empty, or it is filled with the wrong virtual address
  - pte = pagetable_lookup_vaddr(vaddr_to_vfn(vaddr), type);
    if (!pte->valid) { /* Fault */
    stats_miss(type); /* update some statistics */
    handler(pte, type); /* call the page fault handler*/
    }
- Calling the handler() transfers control to the currently used page fault handler. For example, if you issued "./vmsim random", the control will transfer to the function fault_random() in fault.c. For example, to implement LRU, you put your code inside the fault_lru().

physmem.c

- physmem is an array of page table entries (pte_t *). physmem_evict() and physmem_load() are the main functions you will use to implement each page replacement algorithm.
  - For example, physmem[2]->reference = 0; will clear the reference bit of page 2 in memory.
- physmem and root_table->table both contain an array of pte pointers, but physmemis indexed by Physical Frame Number (phsmem[pfn]), while root_table->table is indexed by Virtual Frame Number (root_table->table[vfn]). You can see this clearly by enabling DEBUG and invoking physmem_dump() andpagetable_dump().

What to Submit

- Source code file fault.c
- Output files from running the following 3 commands with the DEBUG flag turned off:
  - ./vmsim -p 3 -s 32 fifo example_trace.txt > fifo.output
  - ./vmsim -p 3 -s 32 lru example_trace.txt > lru.output
  - ./vmsim -p 3 -s 32 clock example_trace.txt > clock.output