Programming Assignment 5 Report

Names: Andrew Ortiz and Mitchell Curtis

Problem Description

For this programming assignment we are implementing a page table and different page replacement algorithms. Were instructed to assume a single running process and that the physical memory is fixed size, using the replacement algorithms FIFO, LIFO, and LRU.

Program Design

The basic program design is separate class files for each replacement algorithm (FIFO, LIFO, LRU) that get called and executed in the main class. The main class takes user input for page size in bytes (256-8192), and the physical memory size in megabytes (4-64) that's used for the simulated memory management. The program then runs all three algorithms with two tests, a small-sample test, and a large-same test. This allows us to see how the algorithms scale. The program outputs the number of references, page faults, and total page replacements.

System Implementation

The FIFO and LIFO algorithms are built pretty similarly with just the order of withdrawal being reversed. Both have the same functions: load_page() and replace_page() which do as they say. The LRU algorithm still includes load_page() and replace_page() but adds touch_page() to update its position on how recently it was used.

lifo_replacement.cpp | lifo_replacement.h | lru_replacement.cpp | lru_replacement.h

Users > mitchellcurtis > Downloads > 253325317 > lru_replacement.cpp

```cpp
void LRUReplacement::touch_page(int page_num)
{
    // page is in the page table need to update its position in the list
    // search the list for the page number
    // erase that from the list and push it to the the front
    auto it = find(buffer.begin(), buffer.end(), page_num);
    buffer.erase(it);
    buffer.push_front(page_num);
}

// Access an invalid page, but free frames are available
void LRUReplacement::load_page(int page_num) {
    // page table has free frames so we just load a page
    // create new PageEntry and insert into table
    PageEntry temp;
    temp.frame_num = num_frames;
    temp.valid = true;
    temp.dirty = false;
    page_table[page_num] = temp; // insert into page table

    //edit the map and list
    indexes[page_num] = indx; // insert pagenum and used index of the page
    indx++;
    buffer.push_front(page_num); // push the new page to the front and push everything behind it


}

// Access an invalid page and no free frames are available
int LRUReplacement::replace_page(int page_num) {
    // page table has no free frames now we replace
    int replaced_page_num = buffer.back();

    // remove top of buffer
    buffer.pop_back();

    // update the page table entry for the replaced page
    page_table[replaced_page_num].valid = false; // make as invalid

    // create new page entry for the page table
    PageEntry new_page;
    new_page.frame_num = page_table[replaced_page_num].frame_num; // reuse the frame
    new_page.valid = true; // mark new page as valid
    new_page.dirty = false;

    // update the page table with the new entry
    page_table[page_num] = new_page;

    // Add the index of the new page to the top of the buffer
    buffer.push_front(page_num);

    // Update the current page index
    indexes[page_num] = indx;
    //[replaced_page_num] = NULL;
```

The pagetable file contains two classes: PageEntry and PageTable that store the actual pages and entries used during the simulation.

```
24  class PageEntry
25  {
26  public:
27      // Physical frame number for a given page
28      int frame_num;
29      // valid bit represents whether a page is in the physical memory
30      bool valid = false;
31      // dirty bit represents whether a page is changed
32      bool dirty = false;
33  };
34
35
36  /**
37   * @brief A page table is like an array of page entries.
38   * The size of the page table should equal to the number of pages in logical mem    ory
39   */
40  class PageTable
41  {
42  private:
43      // A page table is like an array of page entries.
44      vector<PageEntry> pages;
45
46  public:
47      // Constructor
48      PageTable(int num_pages);
49      // Destructor
50      ~PageTable();
51
52      // TODO: Add your implementation of the page table here
53
54      /**
55       * @brief Access a page in the page table.
56       * @param i
57       * @return
58       */
59      PageEntry& operator [] (int i) {
60          return pages[i];
61      }
62  };
```

Results

```
mitchellcurtis@MacBook-Pro-9 253325317 % ./prog5 256 4
================================================================
CS 433 Programming assignment 5
Author: xxxxxx and xxxxxxx
Date: xx/xx/20xx
Course: CS433 (Operating Systems)
Description : Program to simulate different page replacement algorithms
================================================================

Page size = 256 bytes
Physical Memory size = 4194304 bytes
Number of pages = 524288
Number of physical frames = 16384

================================Test 1================================================
Logical address: 101853141,     page number: 397863,    frame number = 0,       is page fault? 1
Logical address: 101853027,     page number: 397863,    frame number = 0,       is page fault? 0
Logical address: 72042423,      page number: 281415,    frame number = 1,       is page fault? 1
Logical address: 71971407,      page number: 281138,    frame number = 2,       is page fault? 1
Logical address: 72084158,      page number: 281578,    frame number = 3,       is page fault? 1
Logical address: 101853169,     page number: 397863,    frame number = 0,       is page fault? 0
Logical address: 101854129,     page number: 397867,    frame number = 4,       is page fault? 1
```

```
Logical address: 101853782,     page number: 397866,    frame number = 14,      is page fault? 0
Logical address: 101853138,     page number: 397863,    frame number = 0,       is page fault? 0
Logical address: 71971390,      page number: 281138,    frame number = 2,       is page fault? 0
Logical address: 71971405,      page number: 281138,    frame number = 2,       is page fault? 0
Number of references:           100
Number of page faults:          35
Number of page replacements:    0

================================Test 2================================================
Total number of references: 2000000
**********************Simulate FIFO replacement*****************************
Number of references:           2000000
Number of page faults:          340108
Number of page replacements:    323724
Elapsed time = 0 seconds
**********************Simulate LIFO replacement*****************************
Number of references:           2000000
Number of page faults:          829325
Number of page replacements:    812941
Elapsed time = 0 seconds
**********************Simulate LRU replacement*****************************
Number of references:           2000000
Number of page faults:          293297
Number of page replacements:    276913
Elapsed time = 21 seconds
mitchellcurtis@MacBook-Pro-9 253325317 %
```

Conclusion

Our program output shows that while LRU is taking more time, its producing considerably less page faults and replacements which implies that there may be a faster way of implementing this so that LRU is the fastest algorithm.