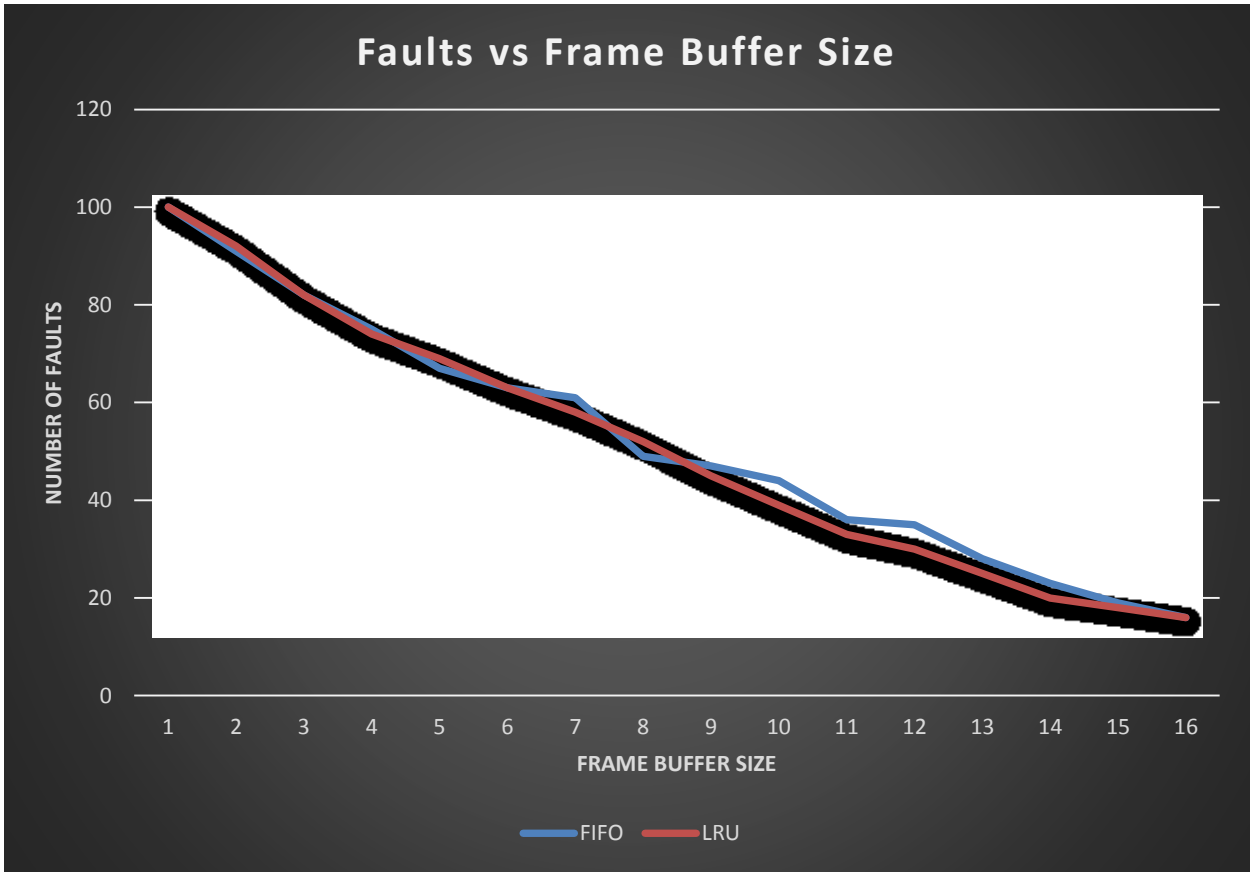
CSC415 – Homework 6 – Page Translation and Page Replacement

By: Aleksandr Kibis

11/11/2014

This assignment was a very nice break from the usual system call assignments. Especially since the Windows API did not have to be used. For the first part, I was tasked with building a program that translates virtual page addresses. A hex value would be input, returning page offset and page number. I additionally supplied the page size and the rebuilt virtual address as proof that the translation is correct. This portion of the assignment was very straightforward and look very little time to do as it was mostly math based.

For the second part, the goal was to design FIFO and LRU page-replacement algorithms. Each one was run 16 times, for frame buffer sizes of 1 to 16. At the beginning of the assignment I assumed that LRU was going to be much faster than FIFO, but was pleasantly surprised to find that not be exactly the case. What I learned is that LRU is very similar to FIFO in terms of structure. While FIFO doesn't do anything when a page number is matched to an index in the frame buffer, LRU moves that index to the top. Upon gathering and analyzing my data, it looks like LRU is still the better algorithm but for a different reason. Even for random page accesses, LRU tends to be much more predictable in terms of buffer size vs page faults. Whereas FIFO has a more uneven spread.



Code

Page Translation:

```c
/*
 * File:   translatePage.c
 * Author: Aleksandr Kibis
 *
 * Compile: gcc -o run translatePage.c
 * Run: ./run <0x00000000>
 *
 * This program takes in a 32-bit signed hex number. Outputs page number
 * and offset for 8KB page size.
 *
 * Created on November 8, 2014, 2:10 PM
 */

#include <stdio.h>
#include <stdlib.h>

#define EXIT_SUCCESS 0
#define EXIT_FAILURE 1

#define PAGE_SIZE 8192

void getInfo(const char* virtualAddress);

/*
 *
 */
int main(int argc, char** argv) {

    if (argc != 2){
        printf("Usage: ./run <0x00000000>\n");
        exit(EXIT_FAILURE);
    }

    // get paging and offset info
    const char* hexstring = argv[1];
    getInfo(hexstring);

    return (EXIT_SUCCESS);
}

void getInfo(const char* virtualAddress){
    int address = (int)strtol(virtualAddress, NULL, 0);

    // page number
    int pageNumber = address / PAGE_SIZE;

    // offset
    int offset = address % PAGE_SIZE;

    // virtual address: used to check correct calculation
    int virtual = PAGE_SIZE * pageNumber + offset;

    // write back as hex
    printf("Virtual Address: 0x%x\nPage Size: %dKB\nPage Number: 0x%x\nOffset: 0x%x\n",
            virtual, PAGE_SIZE / 1024, pageNumber, offset);
}
```

Output

Virtual Address: 0x18ac1329

Page Size: 8KB

Page Number: 0xc560

Offset: 0x1329

Code

Page Replacement:

```
/*
 * File:   pageReplacement.c
 * Author: Aleksandr Kibis
 *
 * Compile: gcc -o run pageReplacement.c
 * Run: ./run
 *
 * Created on November 9, 2014, 2:54 PM
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define EXIT_SUCCESS 0
#define EXIT_FAILURE 1
#define NUMBER_OF_PAGES 16
#define NUMBER_OF_REFERENCES 100

int runFIFO(int numFrames);
int runLRU(int numFrames);

int inQueue(int numFrames, int searchIndex);
void rearrange(int numFrames, int newRef);
void rearrangeNewest(int numFrames, int existingRef);
void printFrames(int numFrames);
void printRefs();

void pageGen(); // last index = newest
int randomNumber(); // kernel-based randomization

int* pageQueue;
int* frameQueue;

/*
 *  This program generates 1 random string of page references. Then runs
 *  FIFO and LRU page replacement algorithms on the string with frame sizes
 *  1 to numFrames (Default is 16).
 *
 *  Prints out number of faults after each calculation.
 */
int main(int argc, char** argv) {

    int numFrames = 16;
    int FIFO, LRU, i;

    // Get random string of references
    pageGen();
    printRefs();
```

```
    // Run FIFO tests
    printf("FIFO page-replacement results:\n");
    printf("Frame Buf Size\tFIFO\tLRU\n");
    for (i=1; i <= numFrames; i++){
        FIFO = runFIFO(i);
        LRU = runLRU(i);
        printf("%d\t\t%d\t%d\n", i, FIFO, LRU);
    }

    // Run LRU tests
//    printf("\n\nLRU page-replacement results:\n");
//    printf("Frame Buf Size\tFaults\n");
//    for (i=1; i <= numFrames; i++){
//        numberOfFaults = runLRU(i);
//        printf("%d\t\t%d\n", i, numberOfFaults);
//    }



    free(pageQueue);
    free(frameQueue);

    return (EXIT_SUCCESS);
}

// Creates random numbers between 1 and NUMBER_OF_PAGES.
// Returns single random number.

int randomNumber() {

    int number;
    srand((long)fopen("/dev/urandom", "r"));
    number = 1 + rand() % NUMBER_OF_PAGES;
    //printf("Random number: %d\n", number);

    return number;
}

// Creates array of random page references. Consecutive references are unique.

void pageGen() {

    pageQueue = malloc(NUMBER_OF_REFERENCES * sizeof (int));

    int i, number;
    for (i = 0; i < NUMBER_OF_REFERENCES; i++) {
        if (i == 0)
            pageQueue[i] = randomNumber();
        else {
            // keep getting new number while consecutive numbers are equal
            while (pageQueue[i - 1] == (number = randomNumber()))
                number = randomNumber();
            pageQueue[i] = number;
        }
        //printf("pageQueue[%d] = %d\n", i, pageQueue[i]);
    }
}

// First-In First-Out page replacement algorithm

int runFIFO(int numFrames) {
    int numFaults = 0;

    frameQueue = malloc(numFrames * sizeof (int));

    //printRefs();

    // run the rest of the replacement algo
    int i;
```

```
    for (i = 0; i < NUMBER_OF_REFERENCES; i++) {
        if ((inQueue(numFrames, pageQueue[i])) >= 0) {
            //do nothing
        } else {
            numFaults++;
            rearrange(numFrames, pageQueue[i]);
        }
        //printFrames(numFrames);
    }
    //printf("i: %d\n", i);

    return numFaults;
}

// Least-Recently-Used page replacement algorithm

int runLRU(int numFrames) {
    int numFaults = 0;
    int recentIndex;

    frameQueue = malloc(numFrames * sizeof (int));

    //printRefs();

    int i;
    for (i = 0; i < NUMBER_OF_REFERENCES; i++) {
        // move most recently used item to the top
        recentIndex = inQueue(numFrames, pageQueue[i]);
        if (recentIndex >= 0) {
            //printf("recent index: %d\n", recentIndex);
           rearrangeNewest(numFrames, recentIndex);
        } else {
            numFaults++;
            rearrange(numFrames, pageQueue[i]);
        }
        //printFrames(numFrames);
    }


    return numFaults;
}



// Checks whether reference exists in frameQueue.
// Returns index if yes, -1 if no.

int inQueue(int numFrames, int searchIndex) {
    int i;
    for (i = 0; i < numFrames; i++) {
        if (frameQueue[i] == searchIndex){
            //printf("return index: %d\n", i);
            return i;
        }
    }
    return -1;
}

// Erase oldest page reference, move up all other refs, add newest on top

void rearrange(int numFrames, int newRef) {
    int i;
    for (i = 0; i < numFrames; i++) {
        if (i + 1 == numFrames)
            frameQueue[i] = newRef;
        else
            frameQueue[i] = frameQueue[i + 1];

    }
```

```
}

// If existing index is used in LRU, move to top

void rearrangeNewest(int numFrames, int existingRef) {
    int i;
    // hold newest used frame
    int newestFrame = frameQueue[existingRef];

    //printf("Newest Frame: %d\t Index: %d\n", newestFrame, existingRef);

    // move up all items to existing reference
    for (i = existingRef; i < numFrames; i++) {
        if (i + 1 == numFrames) {
        }// do nothing
        else
            frameQueue[i] = frameQueue[i + 1];
    }
    // add most recently used to top of queue
    frameQueue[numFrames - 1] = newestFrame;
    //printFrames(numFrames);
}

void printFrames(int numFrames){
    int i;
    printf("{");
    for (i=0; i < numFrames; i++){
        if (i == numFrames - 1)
            printf("%d", frameQueue[i]);
        else
            printf("%d, ", frameQueue[i]);
    }
    printf("}\n");
}

void printRefs(){
    int i;
    printf("\nRandomly Generated String of References:\n{");
    for (i=0; i < NUMBER_OF_REFERENCES; i++){
        if (i == NUMBER_OF_REFERENCES - 1)
            printf("%d", pageQueue[i]);
        else
            printf("%d, ", pageQueue[i]);
    }
    printf("}\n\n");
}
```

Output

Randomly Generated String of References:

{16, 11, 4, 2, 5, 11, 15, 2, 11, 1, 6, 1, 14, 2, 15, 7, 6, 10, 14, 4, 6, 2, 9, 15, 9, 4, 9, 1, 15, 8, 12, 5, 4, 15, 8, 13, 14, 9, 1, 14, 16, 15, 4, 3, 16, 4, 1, 4, 16, 1, 12, 5, 12, 13, 12, 10, 2, 8, 16, 14, 12, 1, 3, 15, 1, 14, 10, 13, 9, 6, 9, 11, 15, 1, 12, 11, 1, 16, 10, 14, 5, 11, 15, 5, 1, 2, 13, 12, 3, 8, 13, 4, 7, 8, 7, 11, 8, 1, 11, 3}

FIFO page-replacement results:

Frame Buf Size	FIFO	LRU
1	100	100

2	91	92
3	82	82
4	75	74
5	67	69
6	63	63
7	61	58
8	49	52
9	47	45
10	44	39
11	36	33
12	35	30
13	28	25
14	23	20
15	19	18
16	16	16