PROGRAMMING ASSIGNMENT#4 (DUE BY 11:59PM AUG 9TH)

PURPOSE

This assignment is to simulate two different page replacement algorithms: First-in First-out(FIFO) and Least Recently Used(LRU). This comparison will give you a better understanding of these two page replacement algorithms.

DESCRIPTION

A program needs to have its code and data reside in memory before it is executed. But sometimes the memory size cannot fit all the code and data of running programs. The solution is to slice both memory and programs into equal-sized pages. An OS can easily swap in the needed page of a program from disk to memory. Paging happens when a page fault occurs. A page fault happens when a needed page is not resident in memory and needs to be swapped in, possibly overwriting another page in memory. A page replacement algorithm decides which memory page must be paged out (swap out, write to disk) to make room for the requested page.

To design a good page replacement algorithm, we don't want to frequently swap the same memory page in and out. So to evaluate a page replacement algorithm, we can run it on a particular pattern of memory references and determine the number of page faults that occur, the fewer the better.

In this assignment, you need to implement the FIFO and LRU page-replacement algorithms presented in this chapter.

ASSIGNMENT

Please download four.zip, which includes the following files:

```
vmalgorithm.h     // header file defines Marcos and global variables
vmalgorithm.c     // implementation of LRU and FIFO algorithms
testvm.c          // test program
Makefile
Readme            // Design Documentation: you need to explain
                     how you implement the LRU and FIFO algorithm
```

Your assignment is to fill the FIFO and LRU functions in `vmalgorithm.c` to simulate FIFO and LRU page replacement algorithm, respectively. Feel free to add helper functions, global variables and etc if needed.

The `testvm.c` is the test program, so DO NOT modify it unless for test purpose (You may need to comment/uncomment a few lines during your test, will elaborate later).

There are several global variables defined `vmalgorithm.h` should be used in your code, please understand each of them before coding:

```
#define AccessPatternLength 20  // total number of memory access

typedef struct
{
    int *PageFrameList;     // The dynamically-allocated array representing
                               memory frames
    int nextToReplaced;       // point to the next frame to be replaced
}PageFrame;


int *accessPattern;         // point to an dynamic-allocated array indicating
                               page access pattern. For example,
                               int accessPattern[5] = [4,2,2,0,1] means page
                               4,2,2,0,1 will be accessed one by one in order

int ReferenceSZ;            // range of pages to be accessed, if this number
                               is 4, which indicates the maximum page index to
                               be accessed is 4.

int FrameNR;                // the number of page frames in memory

PageFrame memory;           // representing the simulated memory
```

You need to understand the differences between page (virtual memory) and page frame (physical memory). For example, if `ReferenceSZ = 7` and `FrameNR = 3`, then the program has 7 virtual pages but the physical memory only has 3 page frames, so paging is mandatory.

Suppose the access pattern `accessPattern = [5 2 2 0 4]`, initially, the memory is empty, so `PageFrameList: -1, -1, -1.`

By following the access pattern, if page fault, the page will be loaded into memory, so `PageFrameList` should be changed in the following order:

```
 5 -1 -1    //access page 5, page fault
 5  2 -1    //access page 2, page fault
 5  2 -1    //access page 2, hit
 5  2  0    //access page 0, page fault
 4  2  0    //access page 4, since the page 5 is LRU, evict it and load in
              Page 4.
```

TESTING

The code will take two arguments: one is the page range (virtual memory), the other is the number of page frames in memory. (The access pattern size is harden coded as 20)

The command Format:

```
./testvm  [reference page range] [number of frames]
```

For example:

```
./pagefault  7  3
```

The displayed information should look as below:

```
The Access Pattern: 5 2 2 0 4 6 4 4 1 3 4 5 0 4 5 6 2 0 0 1
Running program using FIFO algorithm ... ...
 5 -1 -1
 5  2 -1
 5  2 -1
 5  2  0
 4  2  0
    .
    .
    .
 0  6  2
 1  6  2
 page fault of FIFO: 13


The Same Access Pattern: 5 2 2 0 4 6 4 4 1 3 4 5 0 4 5 6 2 0 0 1
Running program using LRU algorithm ... ...

 5 -1 -1
 5  2 -1
 5  2 -1
 5  2  0
 4  2  0
    .
    .
    .
 2  6  0
 2  1  0
 page fault of LRU: 13
```

Note: The `ReferenceSZ` and `FrameNR` are hard-coded to `7` and `3` in the `testvm.c`, strongly recommend you follow the instructions to change `testvm.c` to further verify your code (You only need to comment/uncomment a few lines there).

REQUIREMENTS:

1. Your program should work for any random value of `accessPattern`, `ReferenceSZ` and `FrameNR;`
2. The screen output should be similar as the example shown above, you must print out `PageFrameList` after each memory access;
3. The total number of page fault must be calculated and printed out.

DIRECTIONS TO COMPLETE YOUR ASSIGNMENT

1. Download the source code four.zip from piazza
2. Upload four.zip to Edoras using sftp to programming folder

3. Unzip the four.zip on edoras using the commands:
   unzip four.zip
   so you will have one more folders: four (source files) under programming directory on edoras machine
4. Modify source files under folder four to complete this assignment (Note: you should add appropriate level of comments in your code, otherwise, up to 15% penalty may apply.)
5. Test your program to make sure it works correctly,
6. Please elaborate your implementation in readme file.

HOW TO SUBMIT YOUR ASSIGNMENT

- The source files under the *four* folder on edoras machine will be considered for grading.
- Please finish your coding by **11:59pm Aug 9th** and make sure your program must execute correctly on edoras. Your grading may be started immediately after the deadline unless noticed otherwise, so please make your files ready by the deadline.
- Excuses of "but it worked on my machine" will not be accepted, so if you develop elsewhere, plan to leave time for any migration problems that might arise.