Paging Algorithms

Write a program that implements both the First-In-First-Out (oldest resident page) and the Least-Recently-Used page replacement algorithms presented in Chapter 9 (Virtual Memory).

You have only 16 dynamic page frames of real memory; and 100 pages of virtual memory.

You have 10 processes running in your simulation. Each process has 10 pages (0 through 9) of executable code and data. Logical pages 8 and 9 (or any two pages) of each process have the section of logic that represents a loop and therefore exhibit a significantly higher degree of locality than the other pages. These highly-referenced pages are accessed 10 times as often as the other pages in the same process. (See pseudo-code below)

Processes (A through J) are round-robin scheduled. When a process is dispatched generate a random page-reference indicator where page numbers range from 0 to 9 for that process.

Each time a page is accessed its reference bit is changed from 0 to 1.

When a process runs there is a probability that a page access results in a content update 10 percent of the time. That means that content remains unchanged 90 percent of the time that that same page was accessed. When a page's content is altered the content bit is changed from 0 to 1.

A successful resident page access takes 1 unit of time.

A page is a candidate for removal when its reference bit is 0.

A page-fault required to page-out and page-in a page frame takes 20 units of time. A page-fault required to only page-in a page frame takes 10 units of time.

Whenever the dynamic page pool has 2 or less pages remaining not referenced then reset all reference bits to mark all pages in the pool as candidates for removal.

Apply the random page-reference indicator to each algorithm and record the number of page faults (both page-outs and page-ins) incurred by each algorithm.

Run at least 1 billion time units.

Determine under each algorithm:

- What is the average residence time for a page to remain in real storage.
- What is the difference between the number of page-outs and page-ins, why?
- When does thrashing significantly affect performance and why.
- Once you detect thrashing, what can be done by the system to remediate the problem?

Which method performs better.

Assumptions:

Demand paging is used.
Round-Robin scheduling.
No pages are initially in real memory (disregard use of memory by the OS itself).
All pages for the simulation are loaded from auxiliary storage.
Single CPU system.
Unlimited auxiliary storage for virtual storage paging slots.

Your group must demonstrate the program and explain the results found during their research. Be prepared for questions from both the instructor and students.

Project Grading

- Program Correctness – 30%
  - Adherence to instructed theory.
  - Compiles error free and executes without errors that cause abnormal termination.
  - User friendliness.
  - Efficiency of Algorithm.
- Proper implementation of solution – 20%
- Coding Style – 10%
  - Program should be well documented
  - Code should be clear and read well
- Project Report – 40%
  - Demonstrate understanding of the problem.
  - Analysis of the problems.
  - Discuss the advantages/disadvantages of solutions.

Completed Project – Due on the class meeting prior to the final exam.

- Complete electronic copies only -- including all source code and documentation required to successfully compile, link and execute programming.

HINTS:

Use Round-Robin queueing for selecting the next process to dispatch.

You need a Page Map Table to mark the owning process of a page frame along with a boolean reference bit and boolean change bit.

You also need an Auxiliary Page Table to track the 10 pages of each process.

Counters for Page-Ins and Page-Outs.

When a process is dispatched, you will need a random number routine that will choose a process' page for access to be distributed 90 percent on two pages and 10 percent on the remaining pages. Below is a suggestion:

```
// This routine will return an integer between 0 and 9 with 90% biasing
// of the times that the number is 8 and 9.
// First, a random number between 0 and 10 is selected.
// Then, if the number is in the 10% area another random number is chosen
// between 0 and 8. (representing 8 of 10 possibilities).
// But if the number is in the 90% area another randome number is chosen
// between 0 and 2, (representing 2 of 10 possibilities).
public int getBiasedRand()
{
   int biasednum = 0;
   int num = (int) (Math.random()* 10 );
   if (num >= 9)  // 10 percent area
   {
      num = (int) (Math.random()* 8 );
      biasednum = num;
   }
   else  // 90 percent area
   {
      num = (int) (Math.random()* 2 );
      if (num == 0)
      {
         biasednum = 8;
      }
      else
      {
         biasednum = 9;
      }
   }
   return biasednum;
}
```

A sample distribution based on the above code:

```
The count of 0 is 9
The count of 1 is 9
The count of 2 is 10
The count of 3 is 14
The count of 4 is 14
The count of 5 is 14
The count of 6 is 17
The count of 7 is 13
The count of 8 is 463
The count of 9 is 437
Sum of 8 or 9 is 900
```