Due Thursday May 20 in class. No late homeworks accepted.

GROUP PROBLEMS: #1 and #2  25 pts each
INDIVIDUAL PROBLEMS: #3 and #4  25 pts each

1. Page Replacement Algorithms (25 pts)

Compare the performance of the following page replacement algorithms by filling in the table below. Assume a process is allowed a fixed number of frames = F. Define parameters you use in your overhead analysis such as M = length of the whole memory reference string.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Data structure needed</th>
<th>Overhead per memory reference</th>
<th>Overhead per page fault</th>
<th>Rank from smallest to largest page fault rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPT</td>
<td>an array containing the full memory reference string</td>
<td>none</td>
<td>search future reference string for each page in memory O(F) * O(M)</td>
<td>#1</td>
</tr>
<tr>
<td>FIFO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LRU</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LRU -approx</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2nd chance</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. Demand Paged Memory Management (25 pts)

Mrs. Science's application program involves matrix computations. In particular, one part of the program computes

\[ \vec{Y} = \vec{A} \times \vec{X} \]

where \( \vec{X} \) and \( \vec{Y} \) are one-dimensional vectors of \( n \) integers and \( \vec{A} \) is an \( n \times n \) matrix of integers. Recall that each element of \( \vec{Y} \) is computed as

\[ y_i = \sum_{j=1}^{n} a_{ij} \times x_j \]

Assume \( n = 1024 = 2^{10} \) and that an integer takes 4 bytes. Also, suppose the application is run on a machine with virtual memory of size \( 2^{48} \) bytes, physical memory of \( 2^{28} \) bytes, and page size of \( 2^{10} \) bytes.

a. Fill in the table below by computing the number of pages (of virtual address space) occupied by array \( \vec{A} \), vector \( \vec{X} \), and vector \( \vec{Y} \).

<table>
<thead>
<tr>
<th>Contents</th>
<th>Starting address (virtual)</th>
<th>Number of pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Vector X</td>
<td>1024</td>
<td></td>
</tr>
<tr>
<td>Vector Y</td>
<td>5120</td>
<td></td>
</tr>
<tr>
<td>Array A</td>
<td>9216</td>
<td></td>
</tr>
</tbody>
</table>

b. As seen in the table above, Vector \( \vec{X} \) starts at virtual address 1024. If Vector \( \vec{X} \) is stored in frame 24 in main memory, what is the physical address of \( x_i \)? Show work below. You can give your answer in either binary or decimal.
3. A New (Imaginary) Page Replacement Algorithm (25 pts)

The Expected Forward Distance (EFD) Page Replacement Algorithm is a variation on OPT which guesses which page will be referenced farthest in the future based on probabilities about the likelihood of certain reference strings.

Consider a very simplified example. Suppose a process has a total of 3 pages, pages 1, 2, and 3. Let the frame allocation be 2 physical frames. Suppose there are only two possible reference strings for this process with the respective probabilities known:

1 2 3 3 3 3 3 3 3 1 2 2 with probability \( p \)
1 2 3 3 2 2 1 1 1 2 2 2 1 with probability \( (1-p) \)

Now suppose this process starts executing and issues three memory references:

1 2 3

When page 3 is referenced, a page fault will occur and we must replace either page 1 or page 2. However, we don’t know for sure which of the two possible reference strings is being executed since they both start with 1 2 3. We only know the likelihood of each one.

a. How does the performance of EFD compare to that of OPT?

___ better than OPT  ___ worst than OPT  ___ same as OPT  ___ can’t predict

Explain.

b. How does the performance of EFD compare to that of LRU?

___ better than LRU  ___ worst than LRU  ___ same as LRU  ___ can’t predict

Explain.

c. If \( p = .75 \), which page should be replaced? Show work. Hint: you need to compute EFD for page 1 and then EFD for page 2.
4. PFF Dynamic Page Replacement (25 pts)

The Page Fault Frequency dynamic page allocation scheme operates as follows:
* The page fault rate is monitored for each process. If the rate becomes higher than
  the "high threshold value," the process is given more frames. If there are no more free
  frames, the process is blocked until more frames are free.
* If the rate becomes lower than the "low threshold value," the process loses frames.

Pre-defined for you will be the usual data structures and procedures given in earlier
problems.
In addition, the following new data structures and algorithms are pre-defined:

Allocate(PCB,F): allocates F more frames to the process
with the specified PCB.
Deallocate(PCB,F): deallocates F frames from the
process with the specified PCB.

Low_PFR: the low threshold value, a constant.
High_PFR: the high threshold value, a constant.
F: the number of frames allocated or deallocated, a constant.

Use the same structure for the scheduler as in previous problems:

scheduler() { /* I am a Round Robin Scheduler */
  if (called by a process that is finished) {
    /* current process is done */
    //your code goes here

  } else
    if (called by I/O routines) {
      /* current process requests I/O */
    //your code goes here

}
} else
    if (called by the I/O interrupt handler) {
        /* I/O done for process IO_DonePID */

        //your code goes here
    }
} else
    if (called by the timer interrupt handler) {
        /* current process has used up its time quantum */

        //your code goes here
    }
} else
    if (called by page fault interrupt handler) {

        //your code goes here
    }