Problem Set 3 solutions **Due: March 28** **cs2me3 Winter 2007**

Instructions

1. You are encouraged to work in groups of two. If you cannot find a partner, you can work alone.
2. Please submit one copy of the assignment; if you are working with a partner, both names should appear on the assignment.
3. For **Part A** of the assignment, you must submit an electronic copy of your Java application via WebCT (by the time of the lecture on the due date of the assignment).

Part A

Write a program which does the following experiment: on input m, n, k it creates a random list of m requests, where each request is from among $[n] = \{1, 2, \ldots, n\}$ pages of memory, and it counts the number of misses on a cache of size $i \leq k$ of each of the following page replacement algorithms: LRU, CLOCK, FIFO, LIFO, LFU, LFD[1], and plots the result.

In other words, for each $0 < i \leq k$ it draws a point in position $(i, \text{misses}_{\mathcal{A}}(i))$, the number of misses on a cache of size i on algorithm $\mathcal{A}$. (For a better visual effect, plot the outcome of each algorithm in a different color, and provide a legend.)

Part B

1. For a general i, provide a sequence of page requests that illustrates Belady's Anomaly incurred by FIFO on cache sizes i and $i+1$. In your analysis, assume that the cache is initially empty.

   Solution: Consider the following list:

   $$\underbrace{1, 2, 3 \ldots, i, i+1}_{1}, \underbrace{1, 2, 3, \ldots, i-1}_{2}, \underbrace{i+2}_{3}, \underbrace{1, 2, 3, \ldots, i, i+1, i+2}_{4}$$

   If we have a cache of size $i+1$, then we incur $i+1$ faults in segment 1 (because the cache is initially empty), then we have $i-1$ hits in segment 2, then we have another page fault in segment 3 so we evict 1, and in segment 4 we lag behind by 1 all the way, so we incur $i+2$ page faults. Hence, we incur $i+1+1+i+2 = 2i+4$ page faults in total. Suppose now that we have a cache of size i. Then we incur $i+1$ page faults in segment 1, then we have $i-1$ page faults in segment 2, and one page fault in segment 3, hence $2i+2$ page faults before starting segment 4. When segment 4 starts, we already have pages 1 through $i-1$ in the stack, so we have hits, and then when $i+1$ is requested,

[1] See page 43 of the notes for the description of these algorithms.

we have a fault, and when $i+2$ is requested we have a hit, and hence only one fault in segment 4. Therefore, we have $2i+3$ page faults with a cache of size i. To understand this solution, make sure that you keep track of the contents of the cache after each of the four segments has been processed. Note that i has to be at least 3.

2. Show that FWF *is* a marking algorithm, and that FIFO *is not* a marking algorithm.

   Solution: FWF really implements the marking bit, so it is almost a marking algorithm by definiton. FIFO is not a marking algorithm because with $k = 3$, and the request sequence $1, 2, 3, 4, 2, 1$ it will evict 2 in the second phase even though it is marked.

3. We say that a page replacement algorithm ALG is *conservative* if it satisfies the following condition: On any consecutive input subsequence containing k or fewer distinct page requests, ALG will incur k or fewer page faults. Prove that LRU and FIFO are conservative, but FWF is not.

   Solution: For this exercise assume that the cache is of size k. Otherwise the claim is not true: for example, suppose that we have a cache of size 1, and the following sequences:

   $$1, 2, 1, 2$$

   Then, in that sequence of 4 requests there are only 2 distinct requests, yet with a cache of size 1, there would be 4 faults, for *any* page-replacement algorithm. With a cache of size k, LRU is never going to evict a page during this consecutive subsequence, once the page has been requested. Thus, each distinct page request can only cause one fault. Same goes for FIFO. Thus, they are both conservative algorithms. However, it is possible that half-way through the consecutive subsequence, the cache of FWF is going to get full, and FWF is going to evict everybody. Hence, FWF may have more than one page fault on the same page during this consecutive subsequence.