CSE 325 **Spring 2020**

Computer Project #10

Assignment Overview

This assignment focuses on memory management within an operating system. You will design and implement additional functionality to extend the previous project, as described below.

It is worth 40 points (4% of course grade) and must be completed no later than 11:59 PM on Thursday, 4/9.

Assignment Deliverables

The deliverables for this assignment are the following files:

`proj10.makefile` – the makefile which produces `proj10`
`proj10.student.c` – the source code file for your solution

Be sure to use the specified file names and to submit your files for grading via the CSE Handin system before the project deadline.

Assignment Specifications

The program will simulate some of the steps that an operating system takes to manage a virtual memory system which uses demand paging and a fixed allocation of page frames. The program will gather statistics about the behavior of a single user process under a specified page replacement algorithm.

The system to be simulated contains 1,048,576 bytes of RAM which is divided into 256 page frames. Virtual addresses are 16 bits in length.

1. Your program will accept command-line arguments to specify the file which contains the memory references (the "-refs" option), and to control the display of debugging information (the "-debug" option). In addition, your program will input the simulation parameters from a file named "config" in the current directory.

The first line of the "config" file will contain the page replacement algorithm to be studied (the character string "LRU").

The second line will contain information about the page frames allocated to the process. The first token on the line will be a decimal integer representing the number of page frames allocated; the remaining tokens on the line will be the free frame list. The frame numbers in the free frame list will be given in hexadecimal (with one space between) and will be listed in order (the first frame number listed will be the first frame assigned to hold a page).

The third line will contain information about the valid pages in the process. The first token on the line will be a decimal integer representing the number of valid pages; the remaining tokens on the line will be the valid page numbers (given in hexadecimal, with one space between).

Items in a line will be separated by exactly one space, and each line will terminate with a newline. For example:

```
LRU
4 a7 03 1d c3
7 0 1 2 3 d e f
```

If the “config” file exists, your program may assume that the lines in the file are properly formatted.

2. The "-refs" option will be followed by the name of the file which contains zero or more memory references. Each line of the file will contain the following information:

a) virtual address being referenced (four hexadecimal digits, with leading zeroes)
b) operation being performed (one character; 'R' for read and 'W' for write)

Items in the line will be separated by exactly one space, and the line will terminate with a newline. For example:

```
3608 R
0e90 W
10d4 R
```

If the file exists, your program may assume that the lines in the file are properly formatted.

3. For each valid memory reference in the file, your program will display one line with the following information:

a) virtual address being referenced (four hexadecimal digits, with leading zeroes)
b) operation being performed (one character; 'R' for read and 'W' for write)
c) page number (one hexadecimal digit)
d) page offset (three hexadecimal digits, with leading zeroes)
e) page fault flag (one character; 'F' for page fault, blank otherwise)
f) write back flag (one character; 'B' for write back, blank otherwise)
g) physical address (five hexadecimal digits, with leading zeroes)

Items in the line will be separated by exactly one space, and the line will terminate with a newline. For example:

```
9052 R 9 052 F   7c052
0c1f W 0 c1f F B dfc1f
a004 R a 004     0c004
```

An appropriate error message will be displayed for each invalid memory reference.

4. After the simulation is completed, your program will display the contents of the page table. The display will contain one line for each page table entry:

a) index of the page table entry (one hexadecimal digit)
b) V bit (one character; '0' for not valid, '1' for valid)
c) P bit (one character; '0' for not present, '1' for present)
d) R bit (one character; '0' for not referenced, '1' for referenced)
e) M bit (one character; '0' for not modified, '1' for modified)
f) frame number stored in that page table entry (two hexadecimal digits, with leading zeroes)

Items within a line will be separated by exactly one space, and each line will terminate with a newline. The page table display will begin and end with a blank line, and will include appropriate column headers.

5. After the simulation is completed, your program will display the simulation parameters and the following:

a) total number of read operations
b) total number of write operations
c) total number of page faults
d) total number of write backs

The summary information will be appropriately labeled and formatted and the counts will be displayed in base 10.

6. If the "-debug" option has been selected, your program will display:

   a) the contents of the page table at the start of the simulation
   b) the contents of the page table after each memory reference is processed

7. The program will include appropriate error-handling.

Assignment Notes

1. As stated above, your source code file will be named "proj10.student.c"; that source code file may contain C or C++ statements.

2. You must use "g++" to translate your source code file in the CSE Linux environment.

3. Valid executions of the program might appear as follows:

```
proj10 -refs test1 -debug
proj10 –debug –refs test2
proj10 –refs test3
```

4. Your program must create a data structure representing the page table and set all of the entries to zero at the start of the simulation. Your program will actively manage the page table based on the memory references (and thus the contents of the page table will change over time).