Assignment #5 - Virtual Memory Simulator

Grade Weight: 100pts

Assignment Goals:

- Apply concepts of the memory management subsystem to manage the memory hierarchy, thereby using a small amount of expensive, fast memory and minimize access to the large, slow storage.

Your Task:

- Implement a simulated Memory Management Unit (MMU) in software
- When handling page faults, apply the Least Recently Used (LRU) page replacement algorithm
- Assume the following hardware configuration:
    - Physical Memory Size = 32KB
    - Disk size is 256KB
    - Page Size = 4KB (32 / 4 = 8 physical pages | 256/4 = 64 virtual pages)
- Assume that we start with nothing loaded into physical memory. The sample code uses a linked list with a virtual page frame set to -1 to indicate that the physical pages are not being used.
- Assume that we are ***not*** using a Translation Lookaside Buffer (TLB) and that all memory misses cause a page fault.

Test / Input Data: Each line in the input text contains a list of virtual addresses. These represent memory request from a process that happens in order.

Sample Code: A significant amount of sample code is provided:

- **feeder.c** → reads the input file (list of virtual addresses) and acts like the CPU by firing off requests to access virtual memory addresses.
- **queues.c** → contains very helpful functions for managing queues (you need to keep track of the least recently used process)
    - **push**: add to the end (tail) of the queue
    - **pop**: remove from head of the queue
    - **moveToTail:** given a node in the linked list, remove it and push it to the end of the queue
    - **findPage**: given a physical page number, find the node in the queue
- **memory.c** → initializes data structures (page table and linked list LRU queue)
- **utils.c** → contains an assortment of helper functions to make life easier. Including functions for easily outputting results to the standard output.
- **mmu.c** → for this simulation imagine that you are programming the mmu unit. You need to maintain the the queue data structure to easily find the least recently used (LRU) physical page. Implement the logic for both memory hits and page faults. **You only are**

required to add code to the mmu.c file and not any other C programs, although you can make changes to anything (if you wish…..)

SAMPLE CODE & TEST DATA: Available Here

TESTING: Example output from the code solutions is provided. Correct implementations should have output that is very similar to the output from the test data files provided.

See following pages for sample output…...

Submission Instructions:

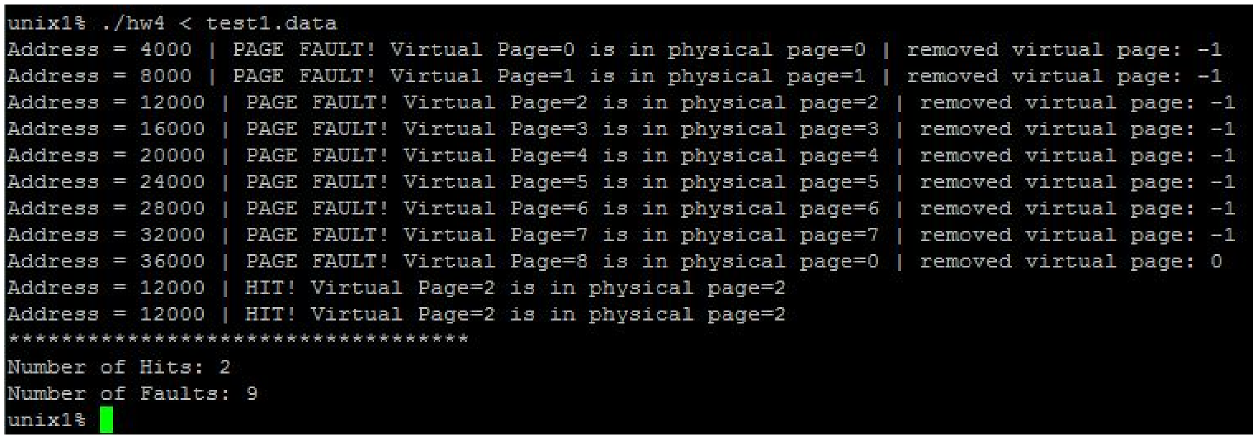
- Submit your code on Blackboard as a single well organized tar file.
- Your code should compile using the Makefile. Submission that do not compile will receive a grade of a zero.
- Make sure your code is well-formatted, commented and has appropriate error checking

Grading Rubric

Code submitted correctly. Contains Makefile, driver and user space program contained in a single .tar file on Blackboard.	**10 points**
Code is well organized, contains reasonable error checking and contains useful comments	**10 points**
Successful Execution for test1.data: Your code executes the test1 data set correctly	**20 points**
Successful Execution for test2.data: Your code executes the test2 data set correctly	**20 points**
Successful Execution for test3.data: Your code executes the test3 data set correctly	**20 points**
Successful Execution for Additional Test: Your code executes correctly on another input data set similar to test #3 that is not shared with the class	**20 points**
TOTAL	**100 Points**

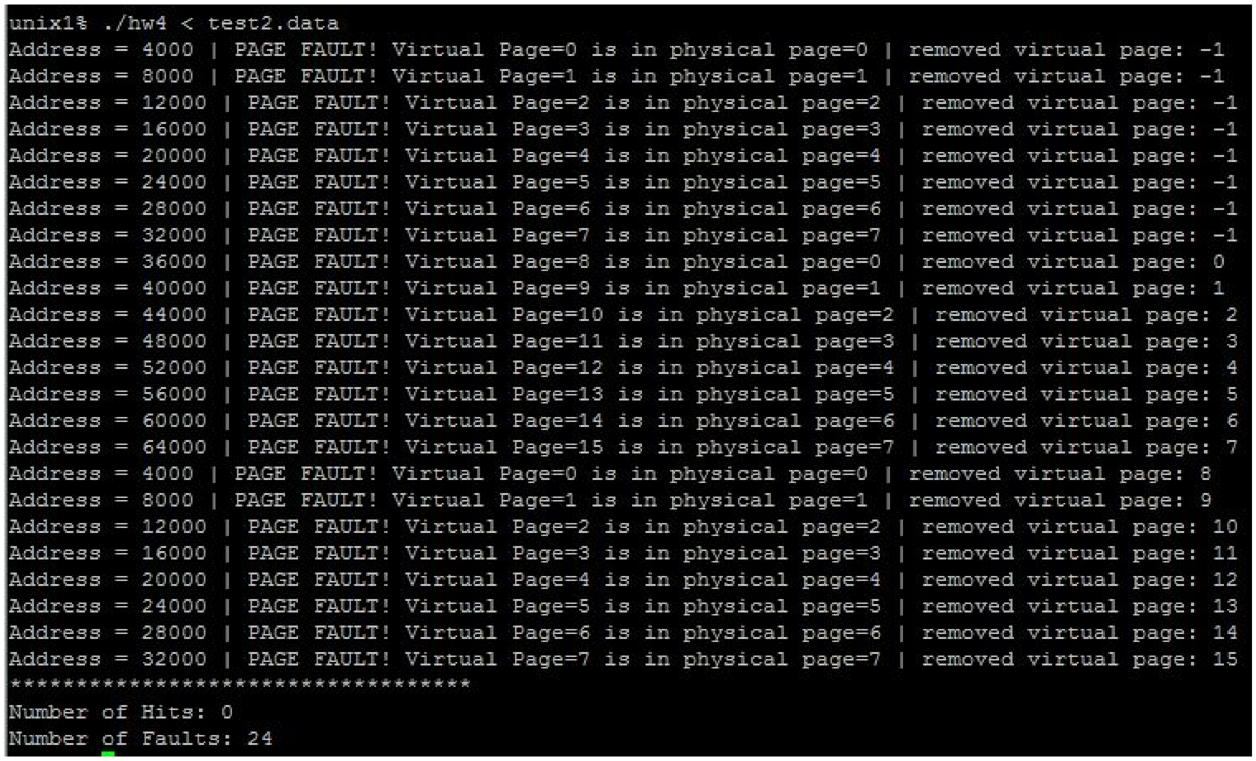
****NOTE: If the code does not compile, it will receive a grade of a zero*

test1.data -- **Initially fills memory and then generates an additional page fault.**



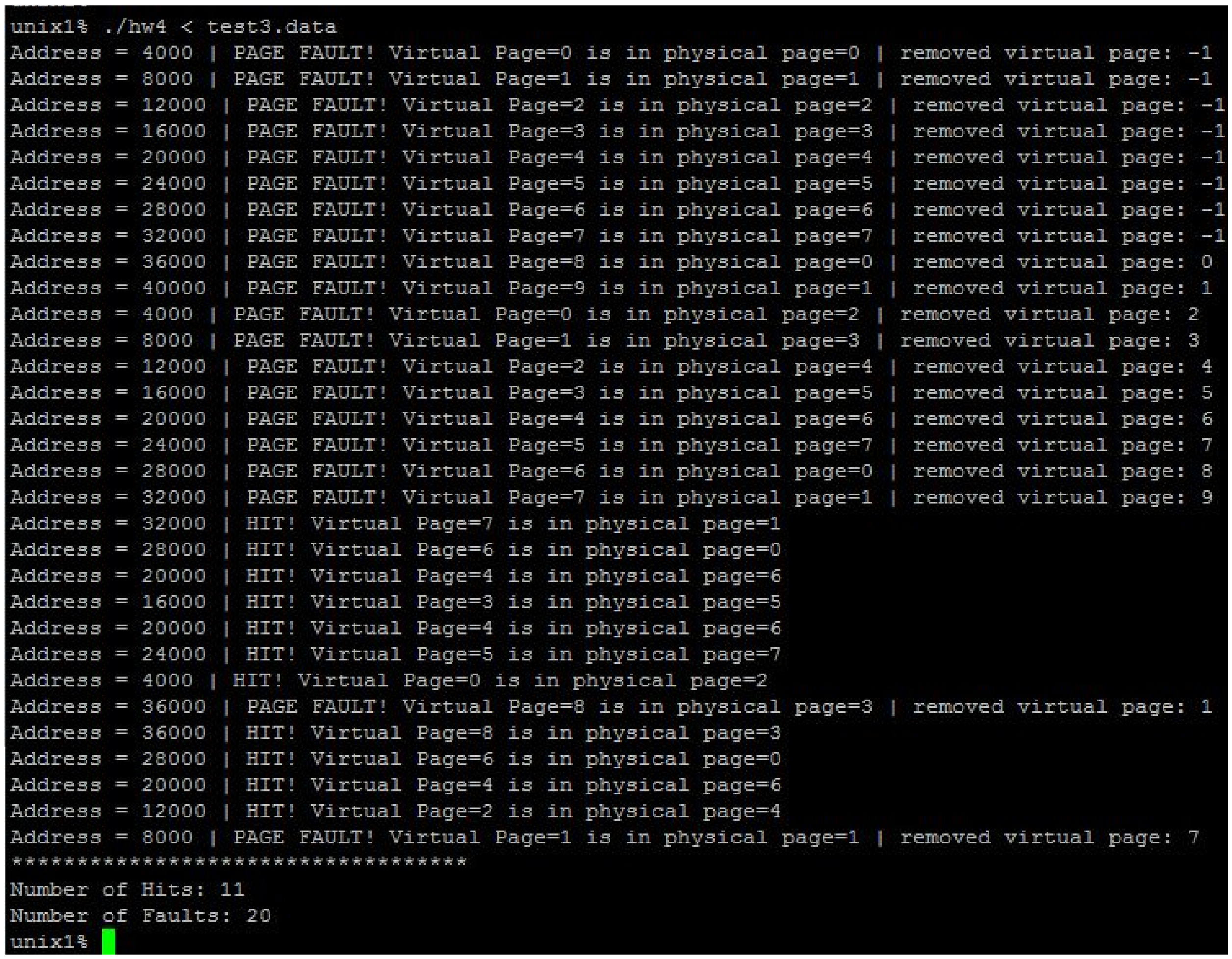
```
unix1% ./hw4 < test1.data
Address = 4000 | PAGE FAULT! Virtual Page=0 is in physical page=0 | removed virtual page: -1
Address = 8000 | PAGE FAULT! Virtual Page=1 is in physical page=1 | removed virtual page: -1
Address = 12000 | PAGE FAULT! Virtual Page=2 is in physical page=2 | removed virtual page: -1
Address = 16000 | PAGE FAULT! Virtual Page=3 is in physical page=3 | removed virtual page: -1
Address = 20000 | PAGE FAULT! Virtual Page=4 is in physical page=4 | removed virtual page: -1
Address = 24000 | PAGE FAULT! Virtual Page=5 is in physical page=5 | removed virtual page: -1
Address = 28000 | PAGE FAULT! Virtual Page=6 is in physical page=6 | removed virtual page: -1
Address = 32000 | PAGE FAULT! Virtual Page=7 is in physical page=7 | removed virtual page: -1
Address = 36000 | PAGE FAULT! Virtual Page=8 is in physical page=0 | removed virtual page: 0
Address = 12000 | HIT! Virtual Page=2 is in physical page=2
Address = 12000 | HIT! Virtual Page=2 is in physical page=2
************************************
Number of Hits: 2
Number of Faults: 9
unix1%
```

test2.data -- **Initially fills memory, then replaces all pages, and the replaces all pages once again**



```
unix1% ./hw4 < test2.data
Address = 4000 | PAGE FAULT! Virtual Page=0 is in physical page=0 | removed virtual page: -1
Address = 8000 | PAGE FAULT! Virtual Page=1 is in physical page=1 | removed virtual page: -1
Address = 12000 | PAGE FAULT! Virtual Page=2 is in physical page=2 | removed virtual page: -1
Address = 16000 | PAGE FAULT! Virtual Page=3 is in physical page=3 | removed virtual page: -1
Address = 20000 | PAGE FAULT! Virtual Page=4 is in physical page=4 | removed virtual page: -1
Address = 24000 | PAGE FAULT! Virtual Page=5 is in physical page=5 | removed virtual page: -1
Address = 28000 | PAGE FAULT! Virtual Page=6 is in physical page=6 | removed virtual page: -1
Address = 32000 | PAGE FAULT! Virtual Page=7 is in physical page=7 | removed virtual page: -1
Address = 36000 | PAGE FAULT! Virtual Page=8 is in physical page=0 | removed virtual page: 0
Address = 40000 | PAGE FAULT! Virtual Page=9 is in physical page=1 | removed virtual page: 1
Address = 44000 | PAGE FAULT! Virtual Page=10 is in physical page=2 | removed virtual page: 2
Address = 48000 | PAGE FAULT! Virtual Page=11 is in physical page=3 | removed virtual page: 3
Address = 52000 | PAGE FAULT! Virtual Page=12 is in physical page=4 | removed virtual page: 4
Address = 56000 | PAGE FAULT! Virtual Page=13 is in physical page=5 | removed virtual page: 5
Address = 60000 | PAGE FAULT! Virtual Page=14 is in physical page=6 | removed virtual page: 6
Address = 64000 | PAGE FAULT! Virtual Page=15 is in physical page=7 | removed virtual page: 7
Address = 4000 | PAGE FAULT! Virtual Page=0 is in physical page=0 | removed virtual page: 8
Address = 8000 | PAGE FAULT! Virtual Page=1 is in physical page=1 | removed virtual page: 9
Address = 12000 | PAGE FAULT! Virtual Page=2 is in physical page=2 | removed virtual page: 10
Address = 16000 | PAGE FAULT! Virtual Page=3 is in physical page=3 | removed virtual page: 11
Address = 20000 | PAGE FAULT! Virtual Page=4 is in physical page=4 | removed virtual page: 12
Address = 24000 | PAGE FAULT! Virtual Page=5 is in physical page=5 | removed virtual page: 13
Address = 28000 | PAGE FAULT! Virtual Page=6 is in physical page=6 | removed virtual page: 14
Address = 32000 | PAGE FAULT! Virtual Page=7 is in physical page=7 | removed virtual page: 15
************************************
Number of Hits: 0
Number of Faults: 24
```

test3.data -- A mix of faults and hits



```
unix1% ./hw4 < test3.data
Address = 4000 | PAGE FAULT! Virtual Page=0 is in physical page=0 | removed virtual page: -1
Address = 8000 | PAGE FAULT! Virtual Page=1 is in physical page=1 | removed virtual page: -1
Address = 12000 | PAGE FAULT! Virtual Page=2 is in physical page=2 | removed virtual page: -1
Address = 16000 | PAGE FAULT! Virtual Page=3 is in physical page=3 | removed virtual page: -1
Address = 20000 | PAGE FAULT! Virtual Page=4 is in physical page=4 | removed virtual page: -1
Address = 24000 | PAGE FAULT! Virtual Page=5 is in physical page=5 | removed virtual page: -1
Address = 28000 | PAGE FAULT! Virtual Page=6 is in physical page=6 | removed virtual page: -1
Address = 32000 | PAGE FAULT! Virtual Page=7 is in physical page=7 | removed virtual page: -1
Address = 36000 | PAGE FAULT! Virtual Page=8 is in physical page=0 | removed virtual page: 0
Address = 40000 | PAGE FAULT! Virtual Page=9 is in physical page=1 | removed virtual page: 1
Address = 4000 | PAGE FAULT! Virtual Page=0 is in physical page=2 | removed virtual page: 2
Address = 8000 | PAGE FAULT! Virtual Page=1 is in physical page=3 | removed virtual page: 3
Address = 12000 | PAGE FAULT! Virtual Page=2 is in physical page=4 | removed virtual page: 4
Address = 16000 | PAGE FAULT! Virtual Page=3 is in physical page=5 | removed virtual page: 5
Address = 20000 | PAGE FAULT! Virtual Page=4 is in physical page=6 | removed virtual page: 6
Address = 24000 | PAGE FAULT! Virtual Page=5 is in physical page=7 | removed virtual page: 7
Address = 28000 | PAGE FAULT! Virtual Page=6 is in physical page=0 | removed virtual page: 8
Address = 32000 | PAGE FAULT! Virtual Page=7 is in physical page=1 | removed virtual page: 9
Address = 32000 | HIT! Virtual Page=7 is in physical page=1
Address = 28000 | HIT! Virtual Page=6 is in physical page=0
Address = 20000 | HIT! Virtual Page=4 is in physical page=6
Address = 16000 | HIT! Virtual Page=3 is in physical page=5
Address = 20000 | HIT! Virtual Page=4 is in physical page=6
Address = 24000 | HIT! Virtual Page=5 is in physical page=7
Address = 4000 | HIT! Virtual Page=0 is in physical page=2
Address = 36000 | PAGE FAULT! Virtual Page=8 is in physical page=3 | removed virtual page: 1
Address = 36000 | HIT! Virtual Page=8 is in physical page=3
Address = 28000 | HIT! Virtual Page=6 is in physical page=0
Address = 20000 | HIT! Virtual Page=4 is in physical page=6
Address = 12000 | HIT! Virtual Page=2 is in physical page=4
Address = 8000 | PAGE FAULT! Virtual Page=1 is in physical page=1 | removed virtual page: 7
************************************
Number of Hits: 11
Number of Faults: 20
unix1%
```

NOTE - In real systems, the hit rate is typically >90% (some architectures it might be closer to a 97% hit rate)