Project Part 2 Lab Report: CprE 308

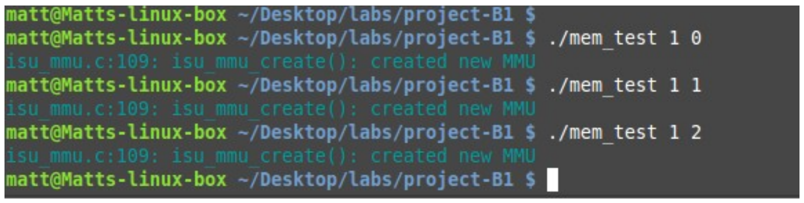
Matt McKillip & Lee Robinson

1. Introduction

In this lab, we will explore several different types of memory page replacement algorithms and compare their effectiveness. More specifically, we will be dealing with FIFO (First In First Out), LRU (Least Recently Used), and the Clock algorithms. We have explained these paging algorithms in lecture, so we're now familiar with the concepts we will be implementing in this lab. After the completion of this lab, we should have a better understanding of how paging algorithms work and how to test their effectiveness.

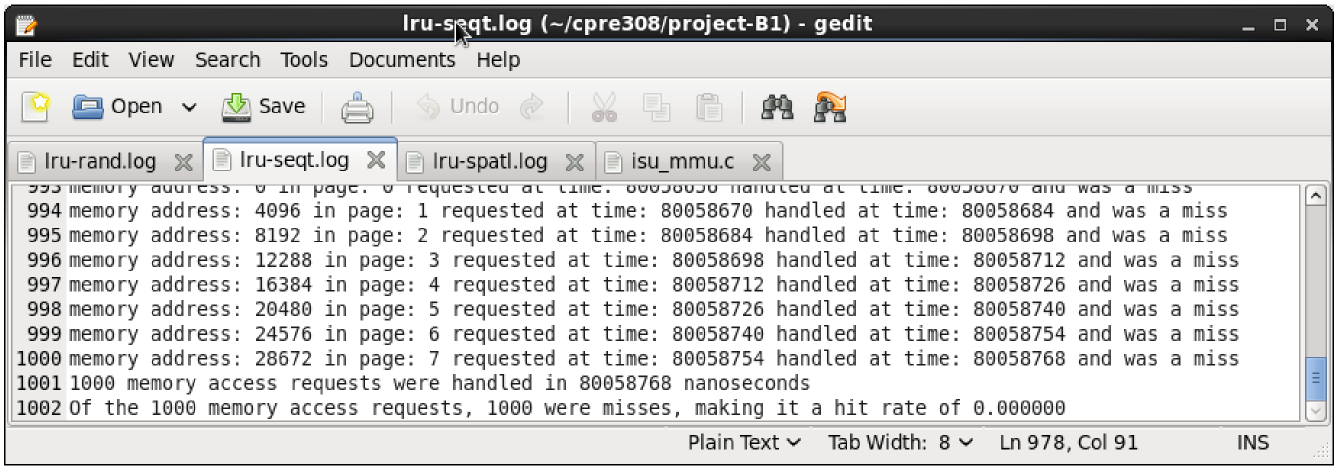
2. Results/Output

Least Recently Used



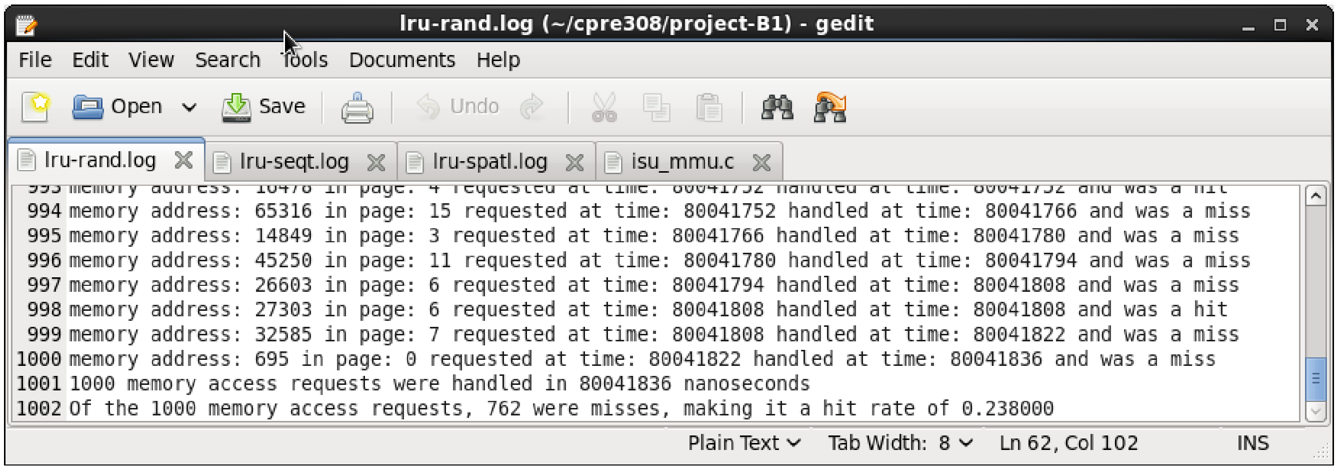
```
matt@Matts-linux-box ~/Desktop/labs/project-B1 $
matt@Matts-linux-box ~/Desktop/labs/project-B1 $ ./mem_test 1 0
isu_mmu.c:109: isu_mmu_create(): created new MMU
matt@Matts-linux-box ~/Desktop/labs/project-B1 $ ./mem_test 1 1
isu_mmu.c:109: isu_mmu_create(): created new MMU
matt@Matts-linux-box ~/Desktop/labs/project-B1 $ ./mem_test 1 2
isu_mmu.c:109: isu_mmu_create(): created new MMU
matt@Matts-linux-box ~/Desktop/labs/project-B1 $
```

Sequential



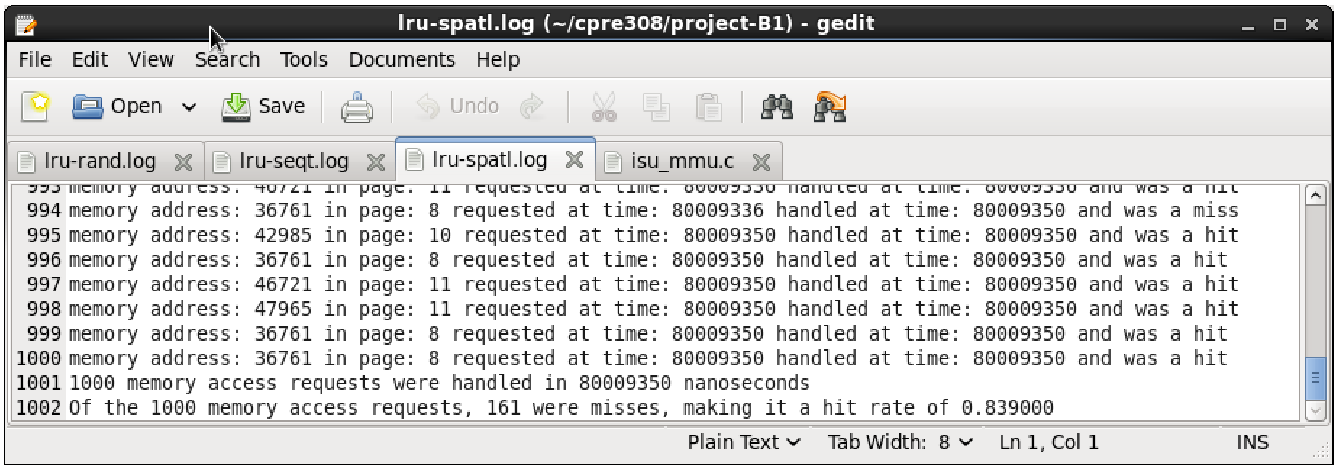
```
memory address: 0 in page: 0 requested at time: 80058656 handled at time: 80058670 and was a miss
memory address: 4096 in page: 1 requested at time: 80058670 handled at time: 80058684 and was a miss
memory address: 8192 in page: 2 requested at time: 80058684 handled at time: 80058698 and was a miss
memory address: 12288 in page: 3 requested at time: 80058698 handled at time: 80058712 and was a miss
memory address: 16384 in page: 4 requested at time: 80058712 handled at time: 80058726 and was a miss
memory address: 20480 in page: 5 requested at time: 80058726 handled at time: 80058740 and was a miss
memory address: 24576 in page: 6 requested at time: 80058740 handled at time: 80058754 and was a miss
memory address: 28672 in page: 7 requested at time: 80058754 handled at time: 80058768 and was a miss
1000 memory access requests were handled in 80058768 nanoseconds
Of the 1000 memory access requests, 1000 were misses, making it a hit rate of 0.000000
```

Random


lru-rand.log (~/cpre308/project-B1) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
lru-rand.log lru-seqt.log lru-spatl.log isu_mmu.c
Plain Text Tab Width: 8 Ln 62, Col 102 INS

```
memory address: 65316 in page: 15 requested at time: 80041752 handled at time: 80041766 and was a miss
memory address: 14849 in page: 3 requested at time: 80041766 handled at time: 80041780 and was a miss
memory address: 45250 in page: 11 requested at time: 80041780 handled at time: 80041794 and was a miss
memory address: 26603 in page: 6 requested at time: 80041794 handled at time: 80041808 and was a miss
memory address: 27303 in page: 6 requested at time: 80041808 handled at time: 80041808 and was a hit
memory address: 32585 in page: 7 requested at time: 80041808 handled at time: 80041822 and was a miss
memory address: 695 in page: 0 requested at time: 80041822 handled at time: 80041836 and was a miss
1000 memory access requests were handled in 80041836 nanoseconds
Of the 1000 memory access requests, 762 were misses, making it a hit rate of 0.238000
```

Spatial

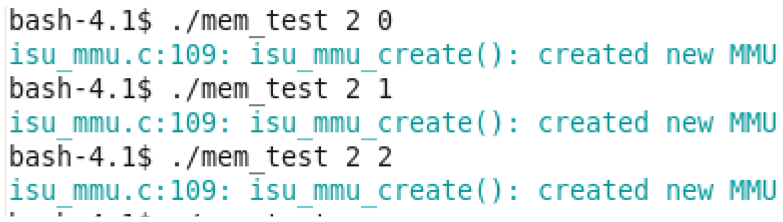

lru-spatl.log (~/cpre308/project-B1) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
lru-rand.log lru-seqt.log lru-spatl.log isu_mmu.c
Plain Text Tab Width: 8 Ln 1, Col 1 INS

```
memory address: 36761 in page: 8 requested at time: 80009336 handled at time: 80009350 and was a miss
memory address: 42985 in page: 10 requested at time: 80009350 handled at time: 80009350 and was a hit
memory address: 36761 in page: 8 requested at time: 80009350 handled at time: 80009350 and was a hit
memory address: 46721 in page: 11 requested at time: 80009350 handled at time: 80009350 and was a hit
memory address: 47965 in page: 11 requested at time: 80009350 handled at time: 80009350 and was a hit
memory address: 36761 in page: 8 requested at time: 80009350 handled at time: 80009350 and was a hit
memory address: 36761 in page: 8 requested at time: 80009350 handled at time: 80009350 and was a hit
1000 memory access requests were handled in 80009350 nanoseconds
Of the 1000 memory access requests, 161 were misses, making it a hit rate of 0.839000
```

Are those the expected outputs/behavior? Which seems to perform the best over all the memory access patterns?

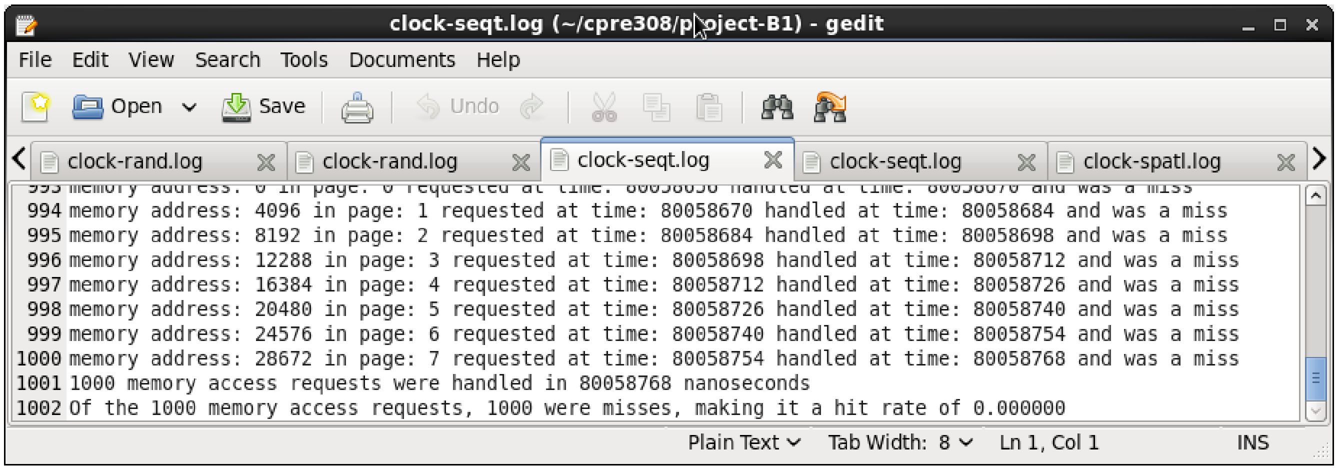
Yes, we expected a similar outcome. The least recently used algorithm performs best in the spatially located memory access pattern and worst in sequential memory access pattern. Sequential memory access pattern is the worst since once a page is accessed another time, all of the previous pages have been accessed, causing the algorithm to always page fault.

Clock Algorithm



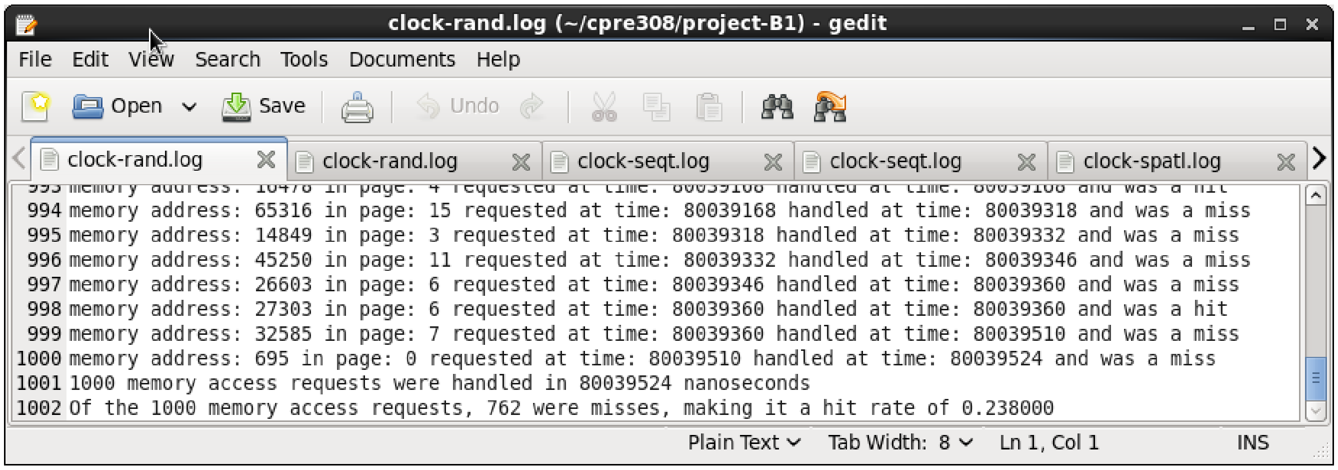
```
bash-4.1$ ./mem_test 2 0
isu_mmu.c:109: isu_mmu_create(): created new MMU
bash-4.1$ ./mem_test 2 1
isu_mmu.c:109: isu_mmu_create(): created new MMU
bash-4.1$ ./mem_test 2 2
isu_mmu.c:109: isu_mmu_create(): created new MMU
```

Sequential



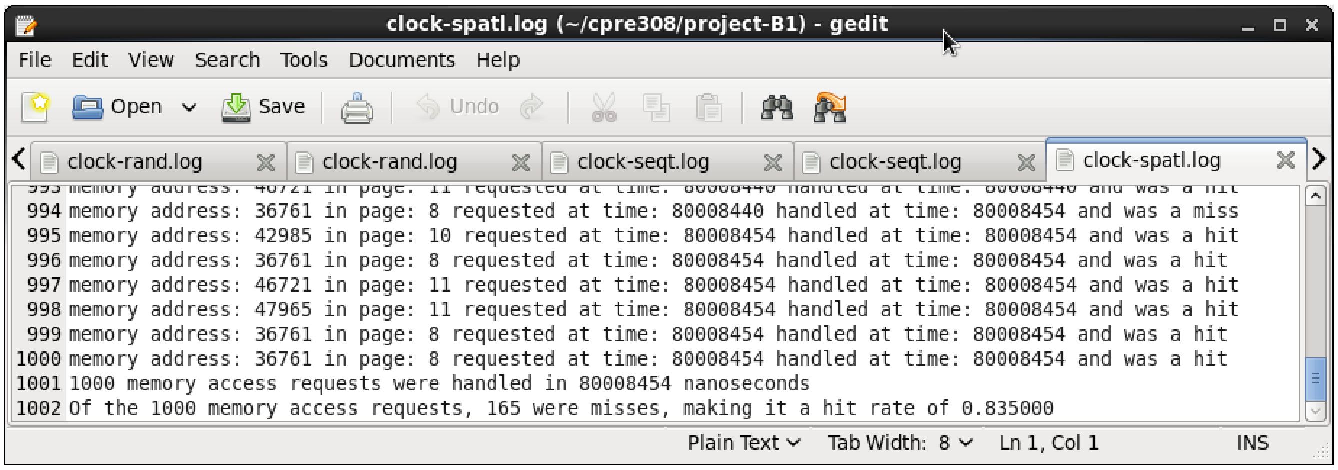
```
memory address: 0 in page: 0 requested at time: 80058656 handled at time: 80058670 and was a miss
memory address: 4096 in page: 1 requested at time: 80058670 handled at time: 80058684 and was a miss
memory address: 8192 in page: 2 requested at time: 80058684 handled at time: 80058698 and was a miss
memory address: 12288 in page: 3 requested at time: 80058698 handled at time: 80058712 and was a miss
memory address: 16384 in page: 4 requested at time: 80058712 handled at time: 80058726 and was a miss
memory address: 20480 in page: 5 requested at time: 80058726 handled at time: 80058740 and was a miss
memory address: 24576 in page: 6 requested at time: 80058740 handled at time: 80058754 and was a miss
memory address: 28672 in page: 7 requested at time: 80058754 handled at time: 80058768 and was a miss
1000 memory access requests were handled in 80058768 nanoseconds
Of the 1000 memory access requests, 1000 were misses, making it a hit rate of 0.000000
```

Random



```
memory address: 16478 in page: 4 requested at time: 80039168 handled at time: 80039168 and was a hit
memory address: 65316 in page: 15 requested at time: 80039168 handled at time: 80039318 and was a miss
memory address: 14849 in page: 3 requested at time: 80039318 handled at time: 80039332 and was a miss
memory address: 45250 in page: 11 requested at time: 80039332 handled at time: 80039346 and was a miss
memory address: 26603 in page: 6 requested at time: 80039346 handled at time: 80039360 and was a miss
memory address: 27303 in page: 6 requested at time: 80039360 handled at time: 80039360 and was a hit
memory address: 32585 in page: 7 requested at time: 80039360 handled at time: 80039510 and was a miss
memory address: 695 in page: 0 requested at time: 80039510 handled at time: 80039524 and was a miss
1000 memory access requests were handled in 80039524 nanoseconds
Of the 1000 memory access requests, 762 were misses, making it a hit rate of 0.238000
```

Spatial


clock-spatl.log (~/cpre308/project-B1) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
clock-rand.log
clock-rand.log
clock-seqt.log
clock-seqt.log
clock-spatl.log
Plain Text Tab Width: 8 Ln 1, Col 1 INS

```
memory address: 46721 in page: 11 requested at time: 80008440 handled at time: 80008440 and was a hit
memory address: 36761 in page: 8 requested at time: 80008440 handled at time: 80008454 and was a miss
memory address: 42985 in page: 10 requested at time: 80008454 handled at time: 80008454 and was a hit
memory address: 36761 in page: 8 requested at time: 80008454 handled at time: 80008454 and was a hit
memory address: 46721 in page: 11 requested at time: 80008454 handled at time: 80008454 and was a hit
memory address: 47965 in page: 11 requested at time: 80008454 handled at time: 80008454 and was a hit
memory address: 36761 in page: 8 requested at time: 80008454 handled at time: 80008454 and was a hit
memory address: 36761 in page: 8 requested at time: 80008454 handled at time: 80008454 and was a hit
1000 memory access requests were handled in 80008454 nanoseconds
Of the 1000 memory access requests, 165 were misses, making it a hit rate of 0.835000
```

Are those the expected outputs/behavior? Which seems to perform the best over all the memory access patterns?

Again the sequential was the worst, and spacial was the best. The results are very similar to the LRU algorithm above. We did have discrepancy between our results and the results in answers, but we were withing a reasonable difference, our results had 2 less page faults. Other than the discrepancy, our results were what we expected, similar to the LRU.

Going Further

After the page replacement algorithms have been implemented, it is recommended that the students play around with the size of L1 cache(or if desired, L2 cache, and RAM as well) and observe the difference in total time it took to handle all memory requests, and the difference in hit rate. Comment on what was experimented with and what observations were made in regards to the effects of changing the cache sizes.

Doubling the L1 size to 8 resulted in higher hit rates and lower running time for every algorithm. This intuitively makes sense, since there are more places to hold pages, more pages can be held in L1. Since there are more pages, there is a better probability for a page hit. Since a page fault takes longer than a page hit, if the hit rate increases the running time will decrease.

Doubling the L2 size seems to not affect the hit rate, but does decrease the running time.

Doubling the RAM size does not effect the hit rate or the running time.

3. Design Decision

Least Recently Used

We decided to keep it simple and use the code from the FIFO algorithm, but replace placement_time with access_time.

Clock Algorithm

To implement the clock algorithm we used a while loop which terminates when the page the hand is pointing to has a ref bit == 0. Then inside the while loop we set the current pages reference bit to 0 then advance the hand.

4. Issues

Most of our issues were implementing the clock algorithm. Our first problem was using the dirty bit in our algorithm, instead of the reference bit. This was a simple oversight, which was simple to fix. Another issue we ran into was we didn't increment our clock hand once we had found the replace index. This gave us some issues, since the output was very similar to the answers, but slightly off.

5. Conclusion

Overall, this lab was a success. We got to see the applications of paging algorithms and observe their effectiveness via test programs. We also gained more experience using Makefiles and compiling libraries. This lab gave us an insight into some of the core features of a real operating system, which helped give the lecture material some value.

6. Suggestions

We have no suggestions for this lab due to the well documented example files given and the straight-forward instructions given in the lab.