Ahmed El Maliki

COP 4600

Spring 2022

Assignment 2: Measuring the Performance of Page Replacement Algorithms on Real Traces.

Introduction:

In this assignment we are to experiment and better understand how the OS and hardware deal with virtual addresses, and the concept of unlimited memory when it comes to the OS (virtual addresses), and limited physical memory when it comes to the hardware.

In computer science this problem has been addressed with the paging process.

Paging is a process of reading data from, and writing data to, the secondary storage (DISK).

It is a memory management system that is used to retrieve processes from the secondary memory(disk) and putting them in the primary memory.

If the page is in main memory and the last operation to that page was not a write, then it is a hit.

The W/R bit is then updated accordingly.

Otherwise, if the page is in main memory and its bit is W, then we do a read disk and write disk operation. If the page is not in main memory, we then just do a read disk operation.

Page table is updated accordingly after a miss or hit according to the conditions set above.

We are to write a program that simulates this memory management scheme with real life traces that implements different paging replacements algorithms. These algorithms are FIFO, LRU, and Segmented FIFO from the paper given in this assignment. All of this was done to compare how these algorithms perform under different circumstances such as different memory sizes.

Methods:

FIFO: A loop with the size of the file traces was used to mimic the CPU calls. A vector was used to mimic the size of the physical memory, the size of the physical memory is a variable entered by the user.

As Explained previously, vector is updated depending on hits or misses and dirty bit.

The vector data structure was also used a FIFO data structure by using C++ STL vector, we push from the back if it is a miss and pop from the front.

LRU: A loop with the size of the file was used to mimic the CPU calls. A vector was used to mimic the size of the physical memory, the size of the physical memory is a variable entered by the user.

As Explained previously, vector is updated depending on hits or misses and dirty bit.

The vector data structure was also used a LRU data structure by using C++ STL vector, we push from the back and pop the least recently used one by always updating the index if it is a hit.

SFIFO:

This is a mixture of both LRU and FIFO, this is 2 level page table.

Both LRU and FIFO have a vector data structure.

The size of LRU and FIFO is done by the equation presented in this assignment.

If the value p==0 or p==100 or even 1(testing showed 1 does not work), then the algorithm becomes FIFO.

Otherwise:

Page is FIFO, it is a hit. Dirty bit is updated accordingly.

Page is not in FIFO and FIFO is not FULL: check size and availability of page in LRU- if it is there, pop value from LRU and push to FIFO, also pop from FIFO and pushback to LRU. If page not in LRU: if LRU full, pop before pushing from FIFO, other wise push from FIFO, also when a miss we push back to FIFO from trace(disk)

Both full, Both not there: pop from LRU, pop from fifo, push fifo to lru, push trace to FIFO from disk.

Results:

All this is done with tracenumber=1,000,000

File: bzip.trace - p=50

	No FRAMES:64		**No Frames:32**	
algo	**Hit rate**	**Miss rate**	**Hit rate**	**Miss rate**
FIFO	**99.85%**	**0.15%**	**99.75%**	**0.25%**
LRU	**99.87%**	**0.13%**	**99.79%**	**0.21%**
SFIFO	**99.87%**	**0.13%**	**99.77%**	**0.23%**

File: sixpack.trace - p=50

	No FRAMES:64		**No Frames:32**	
algo	**Hit rate**	**Miss rate**	**Hit rate**	**Miss rate**
FIFO	**95.17%**	**4.83%**	**91.48%**	**8.52%**
LRU	**95.88%**	**4.12%**	**93.22%**	**6.78%**
SFIFO	**95.83%**	**4.17%**	**93.14%**	**6.86%**

We can clearly see that regardless of algorithm and memory size that these 2 files are completely different. Regardless of algorithm bzip.trace ‘s data seems to have a pattern meaning the traces or CPU calls seem to be not changing pages as much as sixpack.trace.

Conclusion:

It is very intuitive that the more physical memory present in the system, the more hits we get from any algorithm. With that said, these algorithms are primitive compared to real life paging algorithms being used in today’s OS. Apart from memory size, these algorithms will give different results based on the trace file being fed to them.