Kristy Nguyen
Dr. Ed Jorgensen
CS 219-1002: Computer Organization
12 April 2021

CS 219 – Assignment #9

Purpose: Become familiar with caching and virtual memory implementation
Points: 100

Reading/References:

Chapter 5

page table if the page table entry is not in the TLB. Furthermore, if the page is not in main

Assignment:

Answer the following questions:

1) If all misses are classified into one of three categories (compulsory, capacity, or conflict) which misses are likely to be reduced when: [9 pts, 3 pts each]
   a) The program is rewritten so as to require less memory? conflict and capacity
   b) If the clock rate of the processor executing the program is increased? none
   c) If the associativity of the existing cache is increased? conflict

2) What is the purpose of ***swapping***? [5 pts]
   If there is not enough memory available to keep all running processes in memory at the same time, then some processes who are not currently using the CPU, may have their memory swapped out to a secondary storage called the backing store.
3) For a virtual memory environment:
   a) Is it necessary for all of the pages of a process to be in main memory while the process is executing? No [3 pts]
   b) Must the pages of a process in main memory be contiguous? No [3 pts]
   c) In what order are the the pages of a process in main memory? [3 pts]
      The pages are not needed to be in sequential order. The processor executes whichever comes in from the main memory.
4) What is the purpose of a Translation Lookaside Buffer (TLB)? [5 pts]
   Improve performance by reducing having to go to disk to retrieve a table entry.
5) Consider a fixed partitioning scheme with equal-size partitions of 2^{16} bytes and a total main memory size of 2^{24} bytes. A process table is maintained that includes a pointer to a partition for each resident process. How many bits are required for the pointer? [7 pts]
   8 bits are required for the pointer.
6) Give two reasons why the page size in a virtual memory system should ***not*** be very small.
   1. Requires larger page table so it uses more RAM [7 pts]
   2. Likely to increase overhead due to additional page swapping.
7) Give one reason why the page size in a virtual memory system should ***not*** very large.
   1. Some of page may not be used so it is wasteful. [7 pts]
   2. Inefficient use of primary storage
8) Consider a processor with a 16-entry TLB that uses 2 KB pages. What are the performance consequences of this memory system if a program accesses at least 2 MB of memory at a time? Can anything be done to improve the performance? [10 pts]
   For a 16-entry TLB that uses 2 KB pages, there will be more TLB misses since the program is accessing a large amount of memory at a time. Since there is a larger amount of memory being accessed at a time, there will be more pages for the CPU to check the

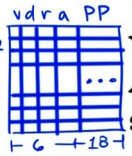
9) Consider a virtual memory system with the following properties:

- 36-bit virtual byte address
- 16 KB pages
- 32-bit physical byte address

VIRTUAL: | 22 | 14 |

PHYSICAL: | 18 | 14 |

Page Table



What is the total size of the page table for each process on this processor, assuming that the valid, protection, dirty, and use bits take a total of 6 bits and that all the virtual pages are in use. Assume that disk addresses are not stored in the page table. Must sow work, final answer should be in megabytes[1]. [15 pts]

16 KB = 2^{14} bytes → 14 bit page offset

2^{22} × 24 bits = 100,663,296 bits → 12,582,912 bytes → 12.58 MB

10) Compile (**g++ -o progA progA.cpp**), execute and time (**time ./progA**) each below programs. Explain, *in detail*, any difference in execution times and ***why***. [10 pts]

a) Program A

```
#include <iostream>
#include <cstdlib>
using namespace std;

int main(int argc, char *argv[])
{
        int     ROWS = 500000, COLS = 8192;
        int     cnt = 0, **arr;

        arr = new int*[ROWS];
        for (int r=0; r < ROWS; r++)
                arr[r] = new int[COLS];

        for (int r=0; r < ROWS; r++)
                for (int c=0; c < COLS; c++)
                        arr[r][c] = cnt++;

        return  0;
}
```

b) Program B

```
#include <iostream>
#include <cstdlib>
using namespace std;

int main(int argc, char *argv[])
{
        int     ROWS = 500000, COLS = 8192;
        int     cnt = 0, **arr;

        arr = new int*[ROWS];
        for (int r=0; r < ROWS; r++)
                arr[r] = new int[COLS];

        for (int c=0; c < COLS; c++)
                for (int r=0; r < ROWS; r++)
                        arr[r][c] = cnt++;

        return  0;
}
```

	Program A	Program B
real:	0m 24.139s	1m 59.153s

The reason why Program A runs faster than Program B is because Program A takes advantage of spatial locality since the arrays are read by rows. Program B takes longer because it does not take advantage of spatial locality since it needs to access the array by column instead of by row.