420-326 – Operating Systems
Winter 2019

Worth 6%

Assignment #1: Caching

Notice

This is version #2 of this document. Be aware that revisions may be made to add details and/or requirements. If a revision is made, it will be posted to Lea. No revision will be made later than one week before the due date. You are responsible for meeting the latest requirement.

This document has a ton of detail. Read it from top-to-bottom and then re-read at least twice to make sure you don't miss anything.

Preamble

You will write a PowerShell script to simulate the job that the control unit (CU) of a CPU performs when it fetches data from cache, RAM and disk to load into the CPU registers. In real life, these operations are performed billions of times a second (with tens of millions of memory addresses). Our simulation will only track 10,000 "pieces" of data (memory addresses). It will simulate 10,000 data accesses by the CU – data will be stored throughout the hierarchy (registers, cache, RAM, disk).

The purpose of the assignment is twofold:

- To learn about the impact of the CPU cache on performance.
  - How does the cache replacement policy affect performance?
  - Increasing cache size will improve performance up to a point - where is the point of diminishing returns?
- To learn about caching algorithms and the tradeoffs each of them makes.
  - Compare and contrast LFU and FIFO caching algorithms (defined below) to determine which the best for our scenario is.

Requirements

Write a single-threaded process program to simulate moving data between storage mediums (primary/secondary storage) and implement caching.

Your program will track 10,000 "pieces" of data. It will simulate 10,000 data accesses by the CU – data will be stored throughout the hierarchy (registers, cache, RAM, disk)

Assumptions

- Spatial locality (https://en.wikipedia.org/wiki/Locality_of_reference) is in effect. In the data file I am providing you (dataset.txt), 99% of required data is located in close proximity (+/- 200 memory addresses) to the previous item.

Performance:

Storage Type	Size	Latency	Bandwidth
Registers	32	.25 NS	**Irrelevant**
L1 cache	1024	2 NS	700 GB/s
RAM	**Irrelevant**	100 NS	15 GB/s
Disk (SSD)	**Irrelevant**	40000 NS	200 MB/s

(Note: Using L1/L2/L3 cache would be more realistic, but also more complicated – assume only L1 cache exists).

Here are some constants to help you:

```
$TOTAL_OPS = 10000
$NUM_REGISTERS = 32
$CACHE_SIZE = 1024

# Response time of storage methods
$REGISTER_ACCESS_TIME = 0.25
$CACHE_ACCESS_TIME = 2
$RAM_ACCESS_TIME = 100
$DISK_ACCESS_TIME = 40000

# Time to transfer a single 64 bit word
$CACHE_BANDWIDTH_TIME    = ([Math]::Pow(10,9) / (700*[Math]::Pow(1024,3) / 8))
$RAM_BANDWIDTH_TIME      = ([Math]::Pow(10,9) / (15*[Math]::Pow(1024,3) / 8))
$DISK_BANDWIDTH_TIME     = ([Math]::Pow(10,9) / (200*[Math]::Pow(1024,2) / 8))

$RAM_HIT_PERCENTAGE = 90
$CACHE_REPLACEMENT_POLICIES = @('FIFO','LFU')
```

Sources (assumptions)

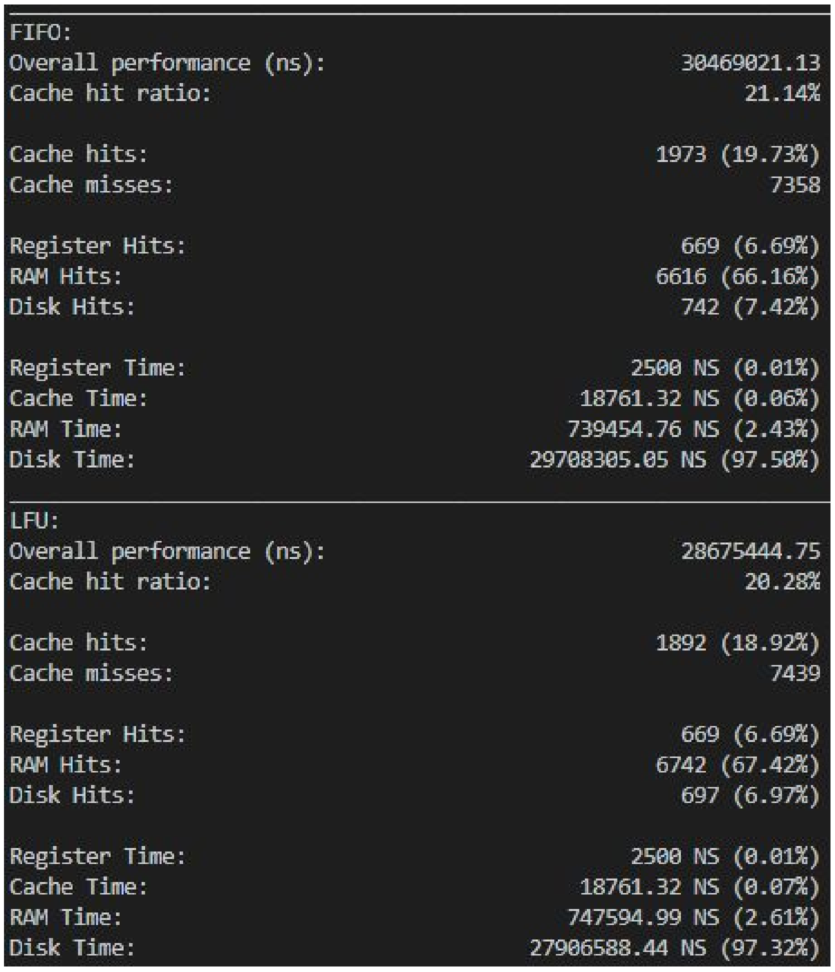
https://en.wikipedia.org/wiki/List_of_device_bit_rates
https://en.wikipedia.org/wiki/Memory_hierarchy
http://stackoverflow.com/questions/14504734/cache-or-registers-which-is-faster

Input

A data file (dataset.txt) with 10000 items (99 % spatial locality, +/- 200) is included with the assignment on Lea. You need to read the file and process each of the address requests.

Output

- For each policy, run your code and output the following:
    - Policy name
    - Overall performance of the of the operations measured in nanoseconds
    - A breakdown of performance by each storage location:
        - Total time, total time for registers, cache, RAM, disk.
        - % of operations that were resolved in registers, cache, RAM, disk – should add to 100%.
        - NOTE: **Cache hit ratio = cache hits / (cache hits + cache misses)** (to avoid counting register hits)
    - Your output layout should look **EXACTLY** like this:
        - NOTE: Total output width = 64



```
FIFO:
Overall performance (ns):                            30469021.13
Cache hit ratio:                                          21.14%

Cache hits:                                        1973 (19.73%)
Cache misses:                                               7358

Register Hits:                                       669 (6.69%)
RAM Hits:                                          6616 (66.16%)
Disk Hits:                                           742 (7.42%)

Register Time:                                   2500 NS (0.01%)
Cache Time:                                  18761.32 NS (0.06%)
RAM Time:                                   739454.76 NS (2.43%)
Disk Time:                               29708305.05 NS (97.50%)

LFU:
Overall performance (ns):                            28675444.75
Cache hit ratio:                                          20.28%

Cache hits:                                        1892 (18.92%)
Cache misses:                                               7439

Register Hits:                                       669 (6.69%)
RAM Hits:                                          6742 (67.42%)
Disk Hits:                                           697 (6.97%)

Register Time:                                   2500 NS (0.01%)
Cache Time:                                  18761.32 NS (0.07%)
RAM Time:                                   747594.99 NS (2.61%)
Disk Time:                               27906588.44 NS (97.32%)
```

Specifications

1. Each line of the input file represents the memory address of some data that the CU will be looking for. Most data accesses in real life demonstrate a high rate of spatial locality (https://en.wikipedia.org/wiki/Locality_of_reference), so most memory addresses are pretty close together.
2. It may not be clear, but you never actually deal with actual data – just the memory addresses (which I generated with a random number generator) is fine.
3. You do not need to store items for RAM or disk -
    1. See assumptions section:
        - RAM is assumed to contain **90%** of all data – a simple call to `Get-Random` can be used to determine if RAM contains the item you need (don't forget to seed the RNG first!)
        - Disk is assumed to contain all data – if the RAM doesn't have the data, disk will.

General Logic Structure

Once you determine the next item to retrieve, generally speaking you do the following:

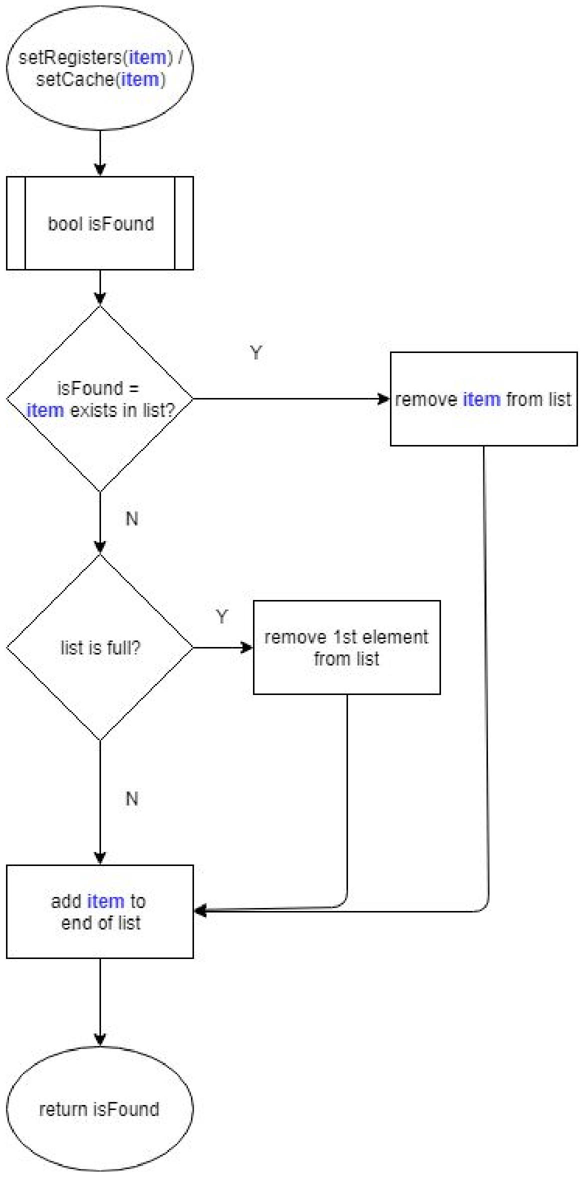
1. Add the register latency time to the overall time variable and check if the registers contain the data already.
2. If it does, move the data to the newest block of the register (i.e. ArrayList methods `removeAt()` and `add()`). See the Registers section for details. Go to the next operation (i.e. step #1)
3. If it doesn't, add the cache latency time to the overall time variable and check if the cache contains the data already.
4. If it's in cache, add the cache bandwidth time to the overall time variable. Modify the cache (if necessary) as per the cache retention policy. Also copy the value to the registers (you might opt to do this in step #2 - be careful not to do it twice!). Go to the next operation (i.e. step #1)
4. If it's not in cache, add the RAM latency time to the timer variable and then check RAM (and so on…)

To summarize the time for a "***hit***" scenario:

Where was the Data?	How much time to Record
Register	Latency (Reg)
Cache	Latency (Reg+cache) + BW (cache)
RAM	Latency (Reg+cache+RAM) + BW (cache + RAM)
Disk	Latency (Reg+cache+RAM+disk) + BW (cache + RAM + disk)

Registers

- Relevant function(s) that you must code: `setRegisters()`
- Assume that there are 32 registers (from the assumptions section).
- Implements FIFO algorithm to determine which register to overwrite once all the registers are full. Note that FIFO includes moving "register hits" to the back of the queue.
    - If the item exists, move it to the end of the queue (enqueue)
    - If the item does not exist, dequeue the front of the queue, enqueue the current item. Discard the dequeued item.
    - If the queue is not yet full, don't dequeue (do move if item exists!)


setRegisters(item) / setCache(item)
bool isFound
isFound = item exists in list?
Y
remove item from list
N
list is full?
Y
remove 1st element from list
N
add item to end of list
return isFound

Cache

- Relevant function(s) that you must code: `setCache()`, `setFIFOCache()`, `setLFUCache()`.
- Pretend you are a CPU designer; for the cache replacement policy (https://en.wikipedia.org/wiki/Cache_replacement_policies), try FIFO and LFU (use separate functions for each)
    - For FIFO - use the following logic the same logic as FIFO for the registers
    - Note that for LFU, you will need to store a counter for number of accesses.
        - **HINT**: My own implementation is done "backwards" because I'm lazy: I used a 2D array where the outer index represents the frequency and the inner array are the items used that number of times (sorted by last access time - actually just a queue). I.e. $cache[3][0] contains the oldest item accessed 4 times (because of zero-indexing, 3 -> 4) (**We'll discuss this and alternatives in class**)
        - Use whatever mechanism you find simplest (KISS principle).
- Cache misses obviously will cause the cache to change.
- Cache hits also affect the cache (FIFO -> move the item to the back of the queue, LRU -> augment the frequency count).
- With LFU – there will constantly for items with a frequency of 1. For ties of this type, remove the oldest item.

RAM

- There is no need to keep track of what is in RAM! Simply assume that 90% of what is needed from RAM is found in RAM. In other words, if the data is not in cache, there is a 90% chance it is in RAM and a 10% chance that you must go to disk.

NOTE: You must set the initial random number generator seed like this (seed = 2019):

```
Get-Random -maximum 100 -SetSeed 2019 | out-null
```

(https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/get-random?view=powershell-6)

Important details

- If your script uses filenames (to load or write files), you <u>must</u> use the `$PWD` environment variable or no path at all. The penalty for failing to do so will be <u>harsh</u>!
- You should test your script under PowerShell 5/5.1 (run `$PSVersionTable.PSVersion.Major` to check). Our computer labs run PowerShell 5. If you have an older version installed at home, simply update it (PowerShell is a free download: http://aka.ms/wmf5download).
  - Using an older version is going to make your life much harder; there are numerous features in PowerShell 5 that aren't in early versions.
- Use Visual Studio Code with the PowerShell extension, you need a debugger!
  - When using the debugger, make sure the debug mode is set to "PowerShell Launch Current File in Temporary Console", otherwise your **`Set-Variable`** constant declarations will fail between runs (you did use constants, right?!)
- Your script must use "**`set-strictmode -version latest`**" right at the top of the script.
- We may build off of this assignment in the future, it is important that you complete it so as to not jeopardize your future grades.
- While programming style is not the primary focus of this class, you are expected to produce quality code. Please use common sense (i.e. functions where appropriate, liberal use of whitespace, consistent indentation, comments where needed, but not excessive, etc.). Grades will be deducted for inconsistency and/or incompetence re: coding style. See handout re: coding style.

Submission

A <u>zip</u> file containing your .ps1 file along with a screenshot of an instance of your program output (to prove that it worked on your PC when it inevitably fails on mine).

Late Penalty

10% per day to a maximum of three days late. If you have a legitimate reason, ask for an extension (a few days in advance) rather than losing grades for nothing.

Grading Rubric

To earn 100% on this assignment you must correctly implement both cache policies.
You may however elect to submit a partial assignment (for partial marks), implementing just FIFO is worth up to (but no more than) 60%.

A note about strategy:

- Students should implement FIFO completely before moving on to LFU policies. This will ensure that you complete at least 60% of the assignment.
    - This involves coding `setRegisters()`, `setFIFOCache()` and `$stats.show()` if you use the starter code I provided.
- LFU is the harder policy to code, don't even start it until you feel confident that FIFO works.
- The number of operations per policy is 10,000 for the submission. For the sake of your own sanity, while coding use a much smaller number! (e.g. 100). 10,000 operations should complete in less than a minute; if your code is significantly slower, you've done something silly.

% of Grade	Description
20%	Demonstrated mastery of introductory PowerShell (conditionals, loops, data structures).
20%	Output format respected.
30%	Code quality: KISS principle respected, commenting where appropriate, proper and consistent formatting (indenting, constants, etc.), **code re-use**.
30%	Bug-free code, logical approach

NOTE: Code that does not work will clearly violate several of the above categories (e.g. broken code is clearly not of good quality, probably does not respect the output format, etc).

Things to Think About (potential test questions)

1. Why does modifying the register set size have relatively little effect on the register hit rate? Changing the spatial locality range, spatial locality odds or number of items all have a larger effect on the hit rate - do you understand the relationship between these things?

2. Why is the cache hit ratio correlated to the number of operations? In other words, why does raising the number of operations seem to raise the cache hit ratio? (to a point of diminishing returns)

3. Why does increasing the cache size alone not have much effect on the cache hit ratio past a certain point? For example, when I evaluate a FIFO cache with a cache size of 4196, I see a ~50% cache hit ratio, but with a cache size of 16,784 (a 4x increase), I only see a small increase in the cache hit ratio (from ~50% to ~54%). Quartering the cache size to 1,024 (a 4x decrease) seems to more than halve the cache hit ratio however (~21%). Explain.

4. Fill in the blanks using the words "increases" or "decreases":
   a. Raising the number of registers _______________ the cache hit ratio
   b. Reducing the % of closely located data (spatial locality) _____________ the cache hit ratio.
   c. Increasing the range of addresses considered closely located ___________ the cache hit ratio.
   d. Lowering the possible # of data items _____________ the cache hit ratio.
   e. Raising the number of operations ______________ the cache hit ratio.
   f. Eliminating the cache _______________ the RAM hit ratio.

5. Your LFU and FIFO implementations should have similar cache hit ratios, yet intuitively we assume LFU should have better performance. Why don't we see it in the code we've produced? What feature would we need to add to showcase the expected better LFU performance? Ignore stale item flushing (#6), the answer is much simpler.

6. (Unrelated to question #5) LFU usually has a mechanism to flush the cache so that very stale items are removed. Why is this an important performance enhancement?

PowerShell Language Features

(things I never remember and constantly have to google. This might not be relevant to you, or it might give you an extra tool or two)

- ErrorAction property: https://stackoverflow.com/questions/15545429/erroractionpreference-and-erroraction-silentlycontinue-for-get-pssessionconfigur
- get-random: http://ss64.com/ps/get-random.html (maximum is exclusive, minimum is inclusive *sigh*)
- Static arrays vs dynamic lists: https://www.jonathanmedd.net/2014/01/adding-and-removing-items-from-a-powershell-array.html
- Columnar output: https://ss64.com/ps/syntax-f-operator.html
- Rounding output to 2 decimals: https://stackoverflow.com/questions/24037021/powershell-round-format-float-to-max-2-decimals
- Powershell shortcut for displaying percentages: `https://blogs.technet.microsoft.com/heyscriptingguy/2015/01/29/powertip-express-percentage-in-powershell/`

Miscellaneous

(feel free to ignore)

To make debugging my own script easier, I created functions that do nothing but test my real functions. I included them in the starter script for no other reason than it might help you structure your thinking:

function testSetRegisters()

function testFetchNextItem()

function testSetFIFOCache()

function testSetLFUCache()