CIS 452

Lab 11 Report

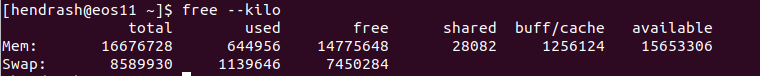
Ashley Hendrickson

Muna Gigowski

Fall 2019

Virtual Memory Performance Statistics

1.  Determine your system configuration:

****

* specify what eos system you are working on **#eos11**
  1. use the **free** memory utility program to determine:
     + the total amount of *physical* memory (KB) on your system

**15,653,306 KB**

* + - the current amount of *free* memory (KB)

**14,775,648 KB**

2. Examine and observe the memory demand of an executing process:

* 1. What is your estimate of the approximate memory demand of the Sample Program?

https://lh5.googleusercontent.com/hdYGbe_Q3t6UTOtsZuYoeFhrmm61A3ef8qXTNThwUUk2RmUClV3HET0fbPr-37kDNhkwSBuMZcBQXEVg8m_HvM9XtsL5py1F-sdI3e5ahYTNkarWEaeNikFMQyanJ8H-7dT0UqMX

**dim\*dim\*sizeof(int) = 4194332 Bytes = 16,777,216**

Or **16384 KB**

* 1. Approximately how much does the amount of free (idle) memory change?

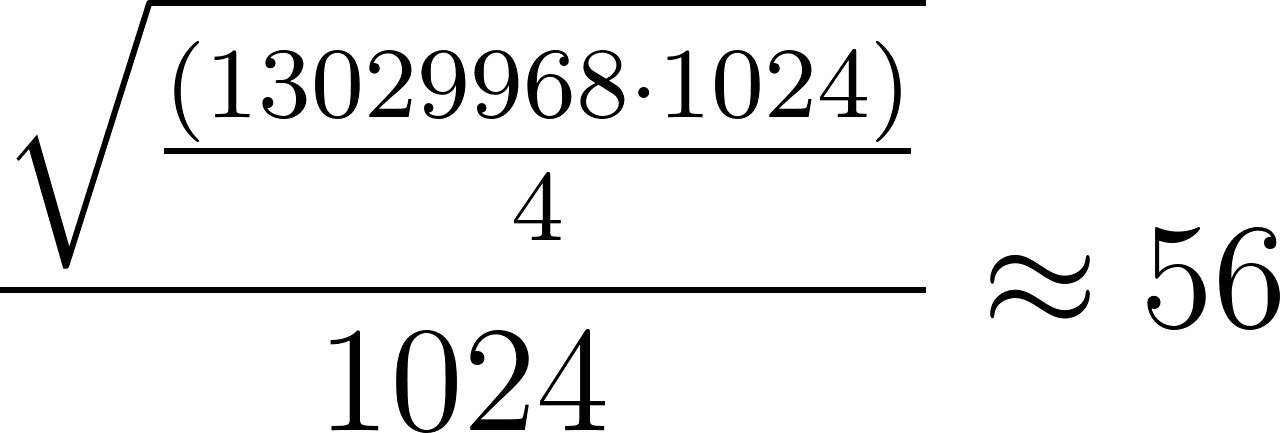
**3744236 - 3717592 = 26,644KB**

* 1. Considering your estimated memory demand of the Sample Program (question 2a), explain why the observed change is an expected result.

**\* This result is expected because the total memory demand of the Sample Program is equal to the memory that must be allocated, plus the individual variables, plus and the object file and the library.**

3. Examine the effect of increased demand for memory resources:

1. The computed value for COEFFICIENT will be different on different machines -- describe and *justify* your choice of the COEFFICIENT parameter.

**[](https://www.codecogs.com/eqnedit.php?latex=\frac%7b\sqrt%7b\frac%7b\left(13029968\cdot1024\right)%7d%7b4%7d%7d%7d%7b1024%7d\approx56%250)**

**We rounded up to 57 thought to be safe**

* 1. Observe what happens to the amount of free memory.  Given your computations and the results from experiment 2 above, is this what you expected to see?  Why or why not?

**We expected the free memory to get close to zero, but it never reached zero completely; at its lowest free memory was 164,684. The original amount of free memory was 13,681,600 KB.**

**The amount of free memory drops because it’s being allocated, however it will never go down to zero because it’s being used by other processes and is needed for other essential buffers; instead memory from the disk and cache will be allocated. Free memory changes during the process from 13,681,600 KB to 164,684 KB.**

* 1. Reference the man pages for **vmstat** to understand exactly what is being displayed.  What other *memory* field(s), if any, changed during execution?  How has the amount of memory free changed before/after running the test program?  Speculate: *why* have these fields changed? In other words, explain how the system is adapting to the large memory demand of the program.

**The amount of free memory drops because it’s being allocated, but will never go down to zero because it’s being used by other processes and is needed for other essential buffers and the OS. Instead, memory from the disk, cache, and some buffer memory will be allocated. Free memory changes during the process from 13,681,600 KB to 164,684 KB. Active memory goes up from 753,432 to 14,809,964 inactive memory also goes up from 693024 to 860824.**

4.  Examine the effect of memory access patterns:

* Change the COEFFICIENT and LOOP parameters back to their original values.  Read the man pages for the **time** utility program.  Then use /usr/bin/time together with command-line arguments as described for **time** to obtain complete statistics (i.e. run in *verbose* mode).  Execute and time the Sample Program.
  1. obtain basic statistics

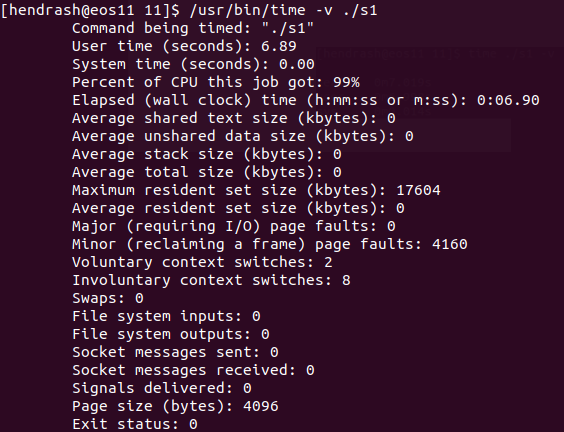
getconf PAGE\_SIZE

* + - what is the size of a page in Linux?

**4096 bytes**

* + - how long does the program take to run?

**6.89 seconds**



* 1. *Precisely*, how does this change alter the program's memory *access* pattern (i.e. what memory objects get "touched", and in what order)? A diagram will help here.

**Before memory was getting accessed row by row then moving down:**

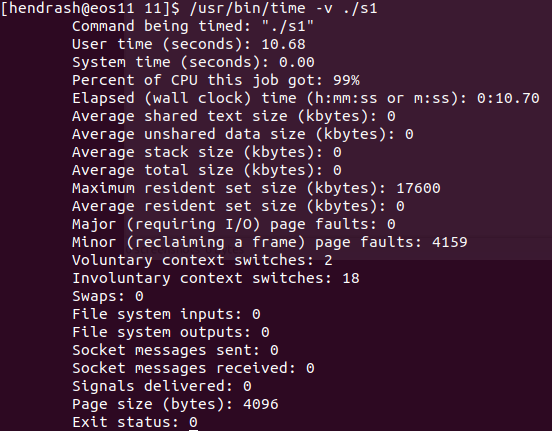
|  |  |  |
| --- | --- | --- |
| intPtr[i \* dim + j] | | |
| 1 | 2 | 3 |
| ... | ... | .... |
| dim + j | Dim + j | Dim + j |

**Now memory is being accessed column by column:**

|  |  |  |
| --- | --- | --- |
| intPtr[j \* dim + i] | | |
| 1 | ... | Dim + i |
| 2 | ... | Dim + i |
| 3 | ... | Dim + i |

* 1. How does this change affect the program's execution time?

**It increases by 3.89 seconds to become 10.68 seconds.**



* 1. *Precisely*, why does the change have the observed effect (your answer must incorporate an important concept related to virtual memory)?

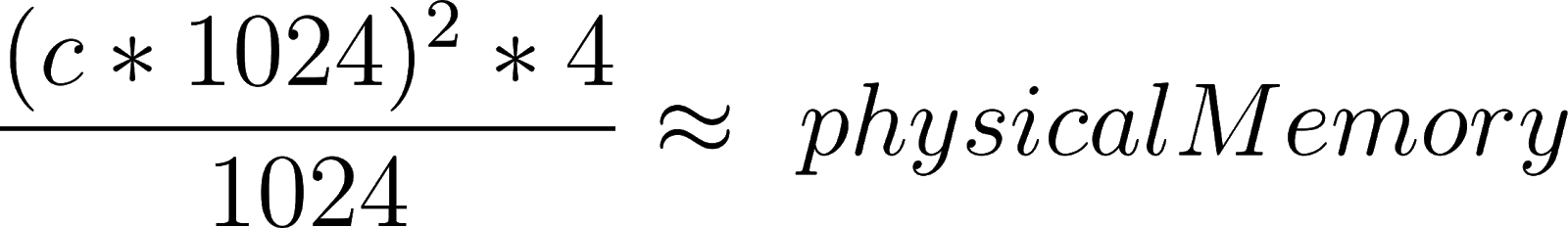
**This is similar to the analogy we talked about in class; when someone has a shop and a customer comes in and asks for something that’s not on the shelves the store clerk has to go to the truck and retrieve it, which takes longer than if it were on the shelf. In this case the memory that is being accessed is so large that the MMU can’t find the address in the TLB so the kernel must find the page in the compressed RAM, then decompress it and put it into physical memory - then the MMU will search for it.**

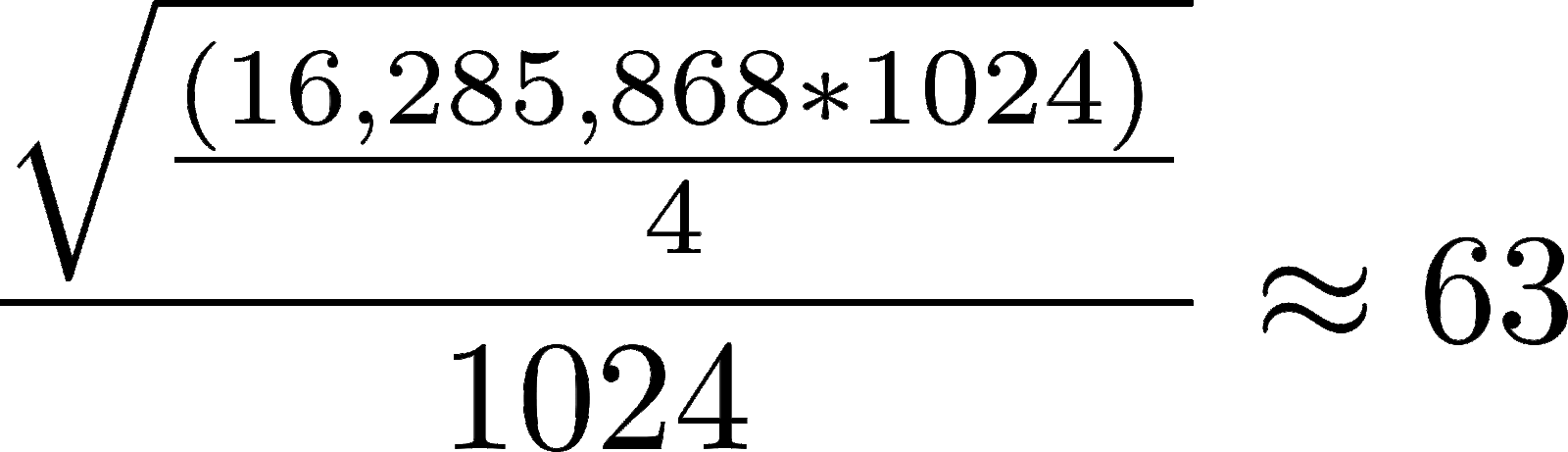
5.  Examine the use of virtual memory:

* Change the memory access pattern for the Sample Program back to its original form.  Change the **LOOP** value to 1.  Adjust the COEFFICIENT parameter in the Sample Program to a value that causes the memory demand of the program to exceed the total amount of *physical* memory on your machine (as determined in question 1 above).

a. Describe and *justify* your computation

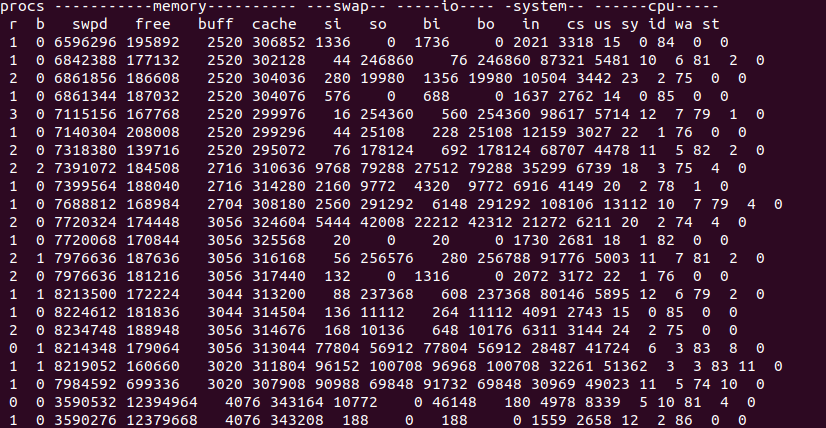
**We have to solve for the coefficient using our given equation:**

**[](https://www.codecogs.com/eqnedit.php?latex=%5Cfrac%7B(c*1024)%5E2*4%7D%7B1024%7D%20%5Capprox%20%5C%20physicalMemory%250)**

**[](http://www.texrendr.com/?eqn=%5Cfrac%7B%5Csqrt%7B%5Cfrac%7B%5Cleft(16%2C285%2C868*1024)%7D%7B4%7D%7D%7D%7B1024%7D%5Capprox63%250)**

**We rounded up to 64**

* configure and run **vmstat** to display statistics once every second and use **/usr/bin/time** in verbose mode to execute and time the program
  1. Observe **vmstat** system statistics as the program executes.  What happens to the amount of free memory (during and after the run)?  Describe *all* the other fields that have changed (including non-memory fields), and describe why they have changed?



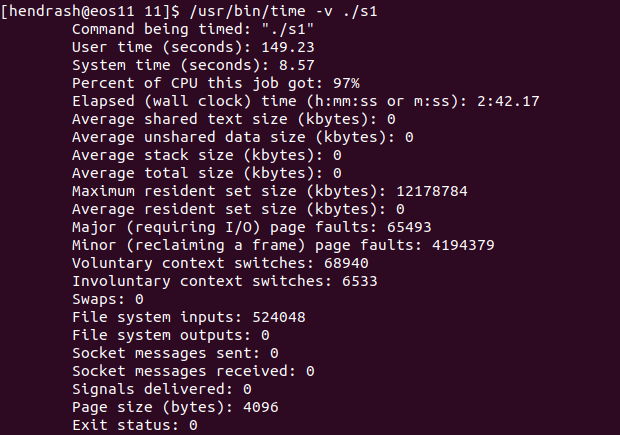
**Cache and buffer memory decrease as the program runs because, as available memory decreases it pulls from these two, so they each also decrease. Understandably, these two then increase after the program is done running.**

**Swap memory goes up during program runtime because it keeps a total of all swapped memory, and as the program runs more and more memory is getting swapped around, which adds to the swap memory total. After the program completes, swap memory decreases**

**Free memory decreases as the program runs since the program itself is pulling from the free memory pool, and increases after the program finishes because the program releases the memory it was using when it completes. Cache and buffer increase after program finishes.**

**Because our program exceeded the amount of physical memory available, RAM memory gets utilized as backup memory, which is technically an I/O operation, so the I/O fields are also altered when the program runs, and after it finishes.**

* 1. Explain how the operating system is adapting to the increased memory demand of the Sample Program.  Include a brief discussion of the execution time and the number of page faults incurred. Your explanation should demonstrate that you

**The OS is adapting by swapping in RAM memory in place of the used up physical memory, as the running program reaches its limit. There were 65, 493 major page faults and 4,194,379 minor page faults. The “User Time” listed above is the total running time of the program in seconds, at 149.23 seconds.**