CIS 452
Lab 11 Report

Ashley Hendrickson
Muna Gigowski
Fall 2019

Virtual Memory Performance Statistics

1. Determine your system configuration:

```
[hendrash@eos11 ~]$ free --kilo
              total        used        free      shared  buff/cache   available
Mem:       16676728      644956    14775648       28082     1256124    15653306
Swap:       8589930     1139646     7450284
```

- specify what eos system you are working on **#eos11**
  a. 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:

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

```
(gdb) p dim * dim *sizeof(int)
$4 = 16777216
```

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

Or **16384 KB**

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

3744236 - 3717592 = 26,644KB

c. 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:

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

$$\frac{\sqrt{\frac{(13029968 \cdot 1024)}{4}}}{1024} \approx 56$$

We rounded up to 57 thought to be safe

b. 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.

c. 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.

  b. 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

```
[hendrash@eos11 11]$ /usr/bin/time -v ./s1
        Command being timed: "./s1"
        User time (seconds): 6.89
        System time (seconds): 0.00
        Percent of CPU this job got: 99%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:06.90
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 17604
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 4160
        Voluntary context switches: 2
        Involuntary context switches: 8
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

b. *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

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

It increases by 3.89 seconds to become 10.68 seconds.

```
[hendrash@eos11 11]$ /usr/bin/time -v ./s1
        Command being timed: "./s1"
        User time (seconds): 10.68
        System time (seconds): 0.00
        Percent of CPU this job got: 99%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:10.70
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 17600
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 4159
        Voluntary context switches: 2
        Involuntary context switches: 18
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

d. *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:

  $$\frac{(c * 1024)^2 * 4}{1024} \approx physicalMemory$$

  $$\frac{\sqrt{\frac{(16{,}285{,}868 * 1024)}{4}}}{1024} \approx 63$$

  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

  b. 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?

```
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0 6596296 195892   2520 306852 1336    0  1736     0 2021 3318 15  0 84  0  0
 1  0 6842388 177132   2520 302128   44 246860    76 246860 87321 5481 10  6 81  2  0
 2  0 6861856 186608   2520 304036  280 19980  1356 19980 10504 3442 23  2 75  0  0
 1  0 6861344 187032   2520 304076  576    0   688     0 1637 2762 14  0 85  0  0
 3  0 7115156 167768   2520 299976   16 254360   560 254360 98617 5714 12  7 79  1  0
 1  0 7140304 208008   2520 299296   44 25108   228 25108 12159 3027 22  1 76  0  0
 2  0 7318380 139716   2520 295072   76 178124   692 178124 68707 4478 11  5 82  2  0
 2  2 7391072 184508   2716 310636 9768 79288 27512 79288 35299 6739 18  3 75  4  0
 1  0 7399564 188040   2716 314280 2160 9772  4320  9772 6916 4149 20  2 78  1  0
 1  0 7688812 168984   2704 308180 2560 291292  6148 291292 108106 13112 10  7 79  4  0
 2  0 7720324 174448   3056 324604 5444 42008 22212 42312 21272 6211 20  2 74  4  0
 1  0 7720068 170844   3056 325568   20    0    20     0 1730 2681 18  1 82  0  0
 2  1 7976636 187636   3056 316168   56 256576   280 256788 91776 5003 11  7 81  2  0
 2  0 7976636 181216   3056 317440  132    0  1316     0 2072 3172 22  1 76  0  0
 1  1 8213500 172224   3044 313200   88 237368   608 237368 80146 5895 12  6 79  2  0
 1  0 8224612 181836   3044 314504  136 11112   264 11112 4091 2743 15  0 85  0  0
 2  0 8234748 188948   3056 314676  168 10136   648 10176 6311 3144 24  2 75  0  0
 0  1 8214348 179064   3056 313044 77804 56912 77804 56912 28487 41724  6  3 83  8  0
 1  1 8219052 160660   3020 311804 96152 100708 96968 100708 32261 51362  3  3 83 11  0
 1  0 7984592 699336   3020 307908 90988 69848 91732 69848 30969 49023 11  5 74 10  0
 0  0 3590532 12394964   4076 343164 10772    0 46148   180 4978 8339  5 10 81  4  0
 1  0 3590276 12379668   4076 343208  188    0   188     0 1559 2658 12  2 86  0  0
```

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.

c. 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

```
[hendrash@eos11 11]$ /usr/bin/time -v ./s1
        Command being timed: "./s1"
        User time (seconds): 149.23
        System time (seconds): 8.57
        Percent of CPU this job got: 97%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 2:42.17
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 12178784
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 65493
        Minor (reclaiming a frame) page faults: 4194379
        Voluntary context switches: 68940
        Involuntary context switches: 6533
        Swaps: 0
        File system inputs: 524048
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```

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.