[REPORT]

■과 목 명: 운영체제 (SWE3004-42)

■담 당 교 수: 신동군 교수님

■제 출 일: 2022년 4월 4일

■학　　과: 수학과

■학　　번: 2016314786

■성　　명: 김호진

Chap 21. vmstat

수학과 김호진 (2016314786)

과제를 수행하기에 앞서 vmstat의 결과 값이 의미하는 바에 대해 알아보았다. vmstat 명령어는 프로세스, 메모리, 페이징, I/O 블럭, CPU 활동 사항 등의 정보를 출력하는 기능을 한다.

출력 값 중 proc 필드의 r은 CPU에 접근 대기 중인 실행 가능한 프로세스의 수를 나타내며, b는 I/O 자원을 할당 받지 못해 블록 된 프로세스의 수를 나타낸다. 다음으로 Memory 필드의 swpd는 사용된 가상 메모리의 용량을, free는 사용가능한 여유 메모리의 용량을, buffer는 버퍼에 사용된 메모리의 용량을, cache는 페이지 캐시에 사용된 메모리의 용량을 각각 의미한다. swap 필드의 si는 디스크로부터 swap-in 된 메모리의 양(kb)이며, so는 디스크로 swap-out 된 메모리의 양(kb)이다. I/O 필드의 bi는 블록 디바이스로부터 입력된 블록 수를 나타내고, bo는 블록 디바이스로 보내진 블록 수를 말한다. system 필드의 in은 초당 발생한 interrupts의 수와 같고, cs는 초당 발생한 context switches 수와 같다. 마지막으로 CPU 필드의 us는 CPU가 사용자 수준 코드를 실행한 시간(%)을, sy는 CPU가 시스템 수준 코드를 실행한 시간(%)을, id는 idle 시간(%)을, wa는 IO wait 시간(%)을, st는 가상화를 이용 시 가상머신 CPU의 계산으로 대기 된 시간(%)을 의미한다.

1. First, open two separate terminal connections to the *same* machine, so that you can easily run something in one window and the other. Now, in one window, run vmstat 1**, which shows statistics about machine usage every second. Read the man page, the associated README, and any other information you need so that you can understand its output. Leave this window running** vmstat **for the rest of the exercises below. Now, we will run the program** mem.c **but with very little memory usage. This can be accomplished by typing** ./mem 1 **(which uses only 1 MB of memory). How do the CPU usage statistics change when running** mem**? Do the numbers in the** user time **column make sense? How does this change when running more than one instance of** mem **at once?**

./mem 1의 실행으로 인해 발생한 CPU 사용량의 변화를 알기 위해서 CPU 필드를 살펴본 결과, user time은 증가(0%→12~13%)하고 idle time은 감소(100%→86~87%)한 것을 확인하였다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch21$ vmstat 1
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      0 1366980  67904 2033580    0    0     0     2    2    0  0  0 100  0  0
 0  0      0 1366980  67904 2033580    0    0     0     0   46  264  0  0 100  0  0
 0  0      0 1365436  67904 2033580    0    0     0     0  105  350  0  0 99  0  0
 0  0      0 1365772  67904 2033580    0    0     0     0   69  321  0  0 100  0  0
 1  0      0 1365460  67904 2033580    0    0     0     0   77  317  2  0 98  0  0
 1  0      0 1364704  67904 2033580    0    0     0     0  146  432 12  2 86  0  0
 2  0      0 1365184  67904 2033580    0    0     0     0   93  372 13  1 87  0  0
 1  0      0 1364940  67904 2033580    0    0     0     0  132  407 13  1 86  0  0
 1  0      0 1364940  67904 2033580    0    0     0     0  103  358 13  2 86  0  0
 1  0      0 1364704  67904 2033580    0    0     0     0   76  330 13  0 87  0  0
 1  0      0 1365428  67904 2033580    0    0     0     0  155  459 13  1 86  0  0
```

```
borussen@DESKTOP-L834KLC:~/ostep_ch21$ ./mem 1
allocating 1048576 bytes (1.00 MB)
  number of integers in array: 262144
loop 0 in 2.18 ms (bandwidth: 458.29 MB/s)
loop 1509 in 0.11 ms (bandwidth: 8943.08 MB/s)
loop 3117 in 0.13 ms (bandwidth: 7810.62 MB/s)
loop 4799 in 0.10 ms (bandwidth: 9799.78 MB/s)
loop 6482 in 0.12 ms (bandwidth: 8388.61 MB/s)
loop 8166 in 0.10 ms (bandwidth: 9709.04 MB/s)
loop 9746 in 0.10 ms (bandwidth: 10010.27 MB/s)
loop 11404 in 0.10 ms (bandwidth: 9892.23 MB/s)
loop 13033 in 0.10 ms (bandwidth: 10082.46 MB/s)
loop 14662 in 0.10 ms (bandwidth: 10082.46 MB/s)
loop 16249 in 0.27 ms (bandwidth: 3663.15 MB/s)
```

여러 mem instance를 동시에 실행하면 user time은 instance의 수에 따라 선형적으로 증가한다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch21$ ./mem 1
allocating 1048576 bytes (1.00 MB)
  number of integers in array: 262144
loop 0 in 0.45 ms (bandwidth: 2241.74 MB/s)
loop 1477 in 0.11 ms (bandwidth: 8848.74 MB/s)
loop 3178 in 0.18 ms (bandwidth: 5584.96 MB/s)
loop 4767 in 0.10 ms (bandwidth: 9709.04 MB/s)
loop 6410 in 0.10 ms (bandwidth: 9709.04 MB/s)
loop 8112 in 0.11 ms (bandwidth: 8774.69 MB/s)
loop 9728 in 0.10 ms (bandwidth: 9709.04 MB/s)
loop 11406 in 0.14 ms (bandwidth: 7145.32 MB/s)
loop 13027 in 0.10 ms (bandwidth: 9892.23 MB/s)
loop 14662 in 0.11 ms (bandwidth: 8924.05 MB/s)
loop 16065 in 0.11 ms (bandwidth: 8924.05 MB/s)
```

```
borussen@DESKTOP-L834KLC:~/ostep_ch21$ ./mem 1
allocating 1048576 bytes (1.00 MB)
  number of integers in array: 262144
loop 0 in 2.97 ms (bandwidth: 336.59 MB/s)
loop 1424 in 0.17 ms (bandwidth: 5714.31 MB/s)
loop 2742 in 0.28 ms (bandwidth: 3509.88 MB/s)
loop 4033 in 0.19 ms (bandwidth: 5295.84 MB/s)
loop 5464 in 0.13 ms (bandwidth: 7928.74 MB/s)
loop 6979 in 0.17 ms (bandwidth: 5915.80 MB/s)
loop 8529 in 0.23 ms (bandwidth: 4271.19 MB/s)
loop 10104 in 0.16 ms (bandwidth: 6250.83 MB/s)
loop 11679 in 0.17 ms (bandwidth: 5777.28 MB/s)
loop 13167 in 0.17 ms (bandwidth: 5949.37 MB/s)
loop 14535 in 0.11 ms (bandwidth: 8756.38 MB/s)
```

```
borussen@DESKTOP-L834KLC:~/ostep_ch21$ ./mem 1
allocating 1048576 bytes (1.00 MB)
  number of integers in array: 262144
loop 0 in 0.64 ms (bandwidth: 1559.80 MB/s)
loop 1227 in 0.19 ms (bandwidth: 5405.03 MB/s)
loop 2623 in 0.11 ms (bandwidth: 8848.74 MB/s)
loop 3939 in 0.18 ms (bandwidth: 5497.12 MB/s)
loop 5108 in 0.18 ms (bandwidth: 5683.34 MB/s)
loop 6218 in 0.19 ms (bandwidth: 5295.84 MB/s)
loop 7431 in 0.18 ms (bandwidth: 5683.34 MB/s)
loop 8729 in 0.22 ms (bandwidth: 4524.60 MB/s)
loop 9958 in 0.15 ms (bandwidth: 6574.14 MB/s)
loop 11250 in 0.13 ms (bandwidth: 7810.62 MB/s)
loop 12423 in 0.13 ms (bandwidth: 7463.17 MB/s)
```

```
borussen@DESKTOP-L834KLC:~/ostep_ch21$ vmstat 1
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      0 1362408  67880 2033552    0    0     0     2    2   11  0  0 100  0  0
 2  0      0 1361628  67880 2033552    0    0     0     0   51  285  0  0 100  0  0
 0  0      0 1364096  67880 2033552    0    0     0     0  168  523  1  0 98  0  0
 0  0      0 1362336  67880 2033552    0    0     0     0  136  432  0  1 99  0  0
 1  0      0 1362100  67880 2033552    0    0     0     0   92  396  6  0 94  0  0
 1  0      0 1360120  67880 2033552    0    0     0     0  153  453 13  2 85  0  0
 2  0      0 1354728  67880 2033552    0    0     0     0  131  446 12  2 85  0  0
 1  0      0 1360084  67880 2033552    0    0     0     0  147  427 13  2 85  0  0
 2  0      0 1359848  67880 2033552    0    0     0     0  161  506 15  2 83  0  0
 2  0      0 1358824  67880 2033552    0    0     0     0  259  573 24  7 69  0  0
 2  0      0 1359992  67880 2033552    0    0     0     0  290  645 24  7 69  0  0
 3  0      0 1358816  67880 2033552    0    0     0     0  215  492 24  7 69  0  0
 3  0      0 1358548  67880 2033552    0    0     0     0  292  628 33  4 63  0  0
 3  0      0 1359060  67880 2033552    0    0     0     0  314  685 35  7 57  0  0
 3  0      0 1359768  67880 2033552    0    0     0     0  311  699 36  5 58  0  0
 3  0      0 1359044  67880 2033552    0    0     0     0  302  668 36  6 58  0  0
```

2. Let's now start looking at some of the memory statistics while running mem**. We'll focus on two columns:** swpd **(the amount of virtual memory used) and** free **(the amount of idle memory). Run** ./mem 1024 **(which allocates 1024MB) and watch how these values change. Then kill the running program (by typing control-c) and watch again how the values change. What do you notice about the values? In particular, how does the** free **column change when the program exits? Does the amount of free memory increase by the expected amount when** mem **exits?**

mem이 실행되는 동안 free 메모리의 양은 크게 감소하였고, 프로그램이 중지된 이후 초기와 비슷한 수준으로 다시 증가하였다. 사용된 메모리는 RAM으로 들어가기 때문에 Memory 필드의 다른 수치에서는 별다른 변화가 발견되지 않는다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch21$ vmstat 1
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      0 1363644  67904 2033576    0    0     0     2    2    0  0  0 100  0  0
 1  0      0 1360392  67904 2033576    0    0     0     0   76  320  0  0 100  0  0
 0  0      0 1363164  67904 2033576    0    0     0     0   97  319  0  0 100  0  0
 0  0      0 1363960  67904 2033576    0    0     0     0   89  353  0  0 99  0  0
 0  0      0 1363960  67904 2033576    0    0     0     0   65  297  0  0 100  0  0
 0  0      0 1362928  67904 2033576    0    0     0     0  105  358  0  0 100  0  0
 0  0      0 1363456  67904 2033576    0    0     0     0  117  418  0  0 99  0  0
 1  0      0 1051404  67904 2033576    0    0     0     0   78  320  1  2 98  0  0
 1  0      0 314660  67904 2033576    0    0     0     0   95  313  9  4 87  0  0
 1  0      0 316644  67904 2033576    0    0     0     0  160  431 12  3 85  0  0
 1  0      0 315612  67904 2033576    0    0     0     0  169  434 12  3 85  0  0
 1  0      0 316344  67904 2033576    0    0     0     0   91  301 13  1 86  0  0
 1  0      0 1365872  67904 2033576    0    0     0     0   91  298 12  0 88  0  0
 0  0      0 1365984  67904 2033576    0    0     0     0   37  223  0  1 99  0  0
 0  0      0 1363520  67904 2033576    0    0     0     0   89  320  0  0 100  0  0
 0  0      0 1363992  67904 2033576    0    0     0     0  127  385  0  0 99  0  0
 0  0      0 1364544  67904 2033576    0    0     0     0   55  254  0  0 100  0  0
```

```
borussen@DESKTOP-L834KLC:~/ostep_ch21$ ./mem 1024
allocating 1073741824 bytes (1024.00 MB)
  number of integers in array: 268435456
loop 0 in 577.21 ms (bandwidth: 1774.04 MB/s)
loop 1 in 209.54 ms (bandwidth: 4886.96 MB/s)
loop 3 in 167.64 ms (bandwidth: 6108.15 MB/s)
loop 4 in 215.64 ms (bandwidth: 4748.67 MB/s)
loop 5 in 233.39 ms (bandwidth: 4387.51 MB/s)
loop 7 in 155.22 ms (bandwidth: 6597.18 MB/s)
loop 8 in 212.71 ms (bandwidth: 4814.09 MB/s)
loop 10 in 208.31 ms (bandwidth: 4915.84 MB/s)
loop 12 in 163.35 ms (bandwidth: 6268.78 MB/s)
loop 14 in 140.47 ms (bandwidth: 7289.86 MB/s)
loop 16 in 146.86 ms (bandwidth: 6972.53 MB/s)
loop 18 in 159.33 ms (bandwidth: 6426.74 MB/s)
loop 20 in 151.51 ms (bandwidth: 6758.81 MB/s)
loop 22 in 154.06 ms (bandwidth: 6646.89 MB/s)
loop 24 in 148.26 ms (bandwidth: 6906.75 MB/s)
loop 26 in 146.03 ms (bandwidth: 7012.17 MB/s)
^C
```

3. We'll next look at the swap **columns (**si **and** so**), which indicate how much swapping is taking place to and from the disk. Of course, to activate these, you'll need to run** mem **with large amounts of memory. First, examine how much free memory is on your Linux system (for example, by typing** cat /proc/meminfo**; type** man proc **for details on the** /proc **file system and the types of information you can find there). One of the first entries in** /proc/meminfo **is the total amount of memory in your system. Let's assume it's something like 8 GB of memory; if so, start by running** mem 4000 **(about 4 GB) and watching the swap in/out columns. Do they ever give non-zero values? Then, try with** 5000, 6000**, etc. What happens to these values as the program enters the second loop (and beyond), as compared to the first loop? How much data (total) are swapped in and out during the second, third, and subsequent loops? (do the numbers make sense?)**

Linux system의 free momory 크기를 확인하기 위해 cat /proc/meminfo 명령어를 입력하였고, 다음의 결과를 얻었다. 전체 메모리의 크기는 3.9GB 정도이며 사용가능한 메모리의 크기는 1.4GB 정도이다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch21$ cat /proc/meminfo
MemTotal:        3904468 kB
MemFree:         1367972 kB
MemAvailable:    3210168 kB
Buffers:           67912 kB
Cached:          1915768 kB
SwapCached:            0 kB
Active:           794216 kB
Inactive:        1518276 kB
```

mem 4000을 실행한 결과, 첫 번째 루프에서 대량의 데이터가 swapped out 되었고 상대적으로 소량의 데이터가 swap-in 되었다. 그 후 시간이 지나면서 swap out 값이 감소하였고 최종적으로 0이 되었다. Swap in 수치 또한 줄어들기는 했지만 swap out처럼 일정 수치(0)로 안정화되지는 않았다.

시스템은 막대한 크기의 메모리를 할당하기 위해서 우선적으로 많은 데이터를 swap out 한다. 이후 최소한의 free space를 유지하기 위해 swap out을 지속적으로 수행하다가 목표가 달성되면 swap out을 멈춘다. 그러므로 swap out과 swap in 수치는 적절하다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch21$ vmstat 1
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  1 198380 3450788   8764 162376    2    3     3     5    2    1  0  0 100  0  0
 0  0 198380 3450016   8764 162372    0    0     0     0   46  225  0  0 100  0  0
 0  0 198380 3449976   8764 162388   12    0    12     0  132  377  0  1 99  0  0
 1  0 198380 2285016   8764 162380   24    0    24     0  214  542  3 10 87  0  0
 1  0 197612 1849924   8764 162376  892    0   892     0  378  740  2 12 86  0  0
 1  0 197612 1218288   8764 162376    8    0     8     0  128  334  1 12 87  0  0
 1  0 197612 686848   8772 162376    0    0     0    16   40  248  2  9 89  0  0
 1  0 197612 243472   8772 162376    0    0     0     0   59  242  2 12 87  0  0
 2  0 197880 116708   8744 110192    8  268   816   268  196  375  1 19 80  0  0
 1  0 197880 115532    600  43716    8    0   148   220  198  564  1 16 83  0  0
 2  2 553984 185780    604  29484   88 355912 16860 355912  648 2104  1 15 67 17  0
 1  0 553728 116676    600  33640  284    0  6524     0  282  640  1 16 83  0  0
 1  0 806968 157680    416  32628  100 256000   552 256000  831 2031  0 16 82  2  0
 1  0 820996 119784   1944  37568 25052 43008 34460 43008 1204 2534  1 14 84  0  0
 2  1 262152 3544728   1948  40712 69912 65536 78744 65536 2819 6038  4  7 87  2  0
 1  0 261896 3673788   1948  41424  680    0  1468     0  291  676  0  5 94  0  0
 0  0 261896 3676148   1948  41520   20    0   112     0   69  270  0  2 98  0  0
 0  0 261896 3675912   1948  41516   12    0    12     0   31  181  0  0 100  0  0
 1  0 261128 3673944   1948  42612  840    0  1932     0  314  755  0  0 100  0  0
 0  0 261128 3673040   1948  42608   36    0    36     0  150  417  0  1 99  0  0
```

4. Do the same experiments as above, but now watch the other statistics (such as CPU utilization and block I/O statistics). How do they change when mem **is running?**

mem이 실행되면서, user time과 system time의 점유율(%)은 증가하는 반면, idle time의 점유율(%)은 감소하였다. I/O 필드의 bi와 bo의 경우, 둘 다 증가하는 것을 확인할 수 있다. 이러한 통계적 수치들은 첫 번째 루프 이후 초기와 비슷한 수준으로 돌아갔다. 또한 Swap이 이루어지기 전, 시스템은 swap out 할 페이지를 결정하기 위해서 몇 가지 처리를 진행해야 하고 해당 작업이 완료되면 디스크에 페이지가 기록되는 동안 약간의 대기 시간이 발생하게 되므로 wa가 간헐적으로 나타났다.

5. Now let's examine performance. Pick an input for mem **that comfortably fits in memory (say** 4000 **if the amount of memory on the system is 8 GB). How long does loop 0 take (and subsequent loops 1, 2, etc.)? Now pick a size comfortably beyond the size of memory (say** 12000 **again assuming 8 GB of memory). How long do the loops take here? How do the bandwidth numbers compare? How different is performance when constantly swapping versus fitting everything comfortably in memory? Can you make a graph, with the size of memory used by** mem **on the x-axis, and the bandwidth of accessing said memory on the y-axis? Finally, how does the performance of the first loop compare to that of subsequent loops, for both the case where everything fits in memory and where it doesn't?**

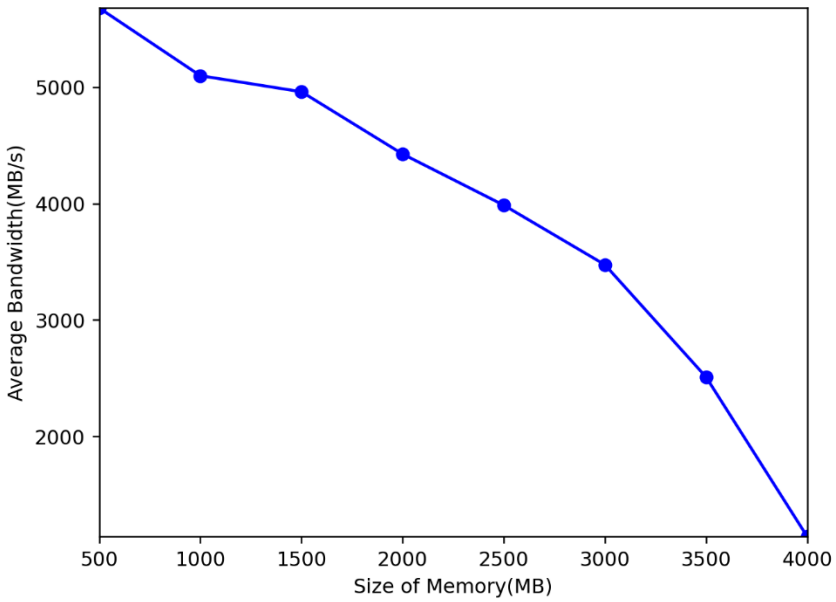
메모리에 완벽하게 맞는 값을 mem의 입력으로 넣게 되면, 다음의 결과를 얻는다. Loop 0에서 약간의 시간이 소요되기는 했지만 그 후의 Loop들은 매우 빠르게 진행되었다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch21$ ./mem 2000
allocating 2097152000 bytes (2000.00 MB)
  number of integers in array: 524288000
loop 0 in 1056.99 ms (bandwidth: 1892.16 MB/s)
loop 1 in 397.89 ms (bandwidth: 5026.55 MB/s)
loop 2 in 391.18 ms (bandwidth: 5112.75 MB/s)
loop 3 in 390.08 ms (bandwidth: 5127.20 MB/s)
loop 4 in 424.64 ms (bandwidth: 4709.87 MB/s)
loop 5 in 375.28 ms (bandwidth: 5329.31 MB/s)
```

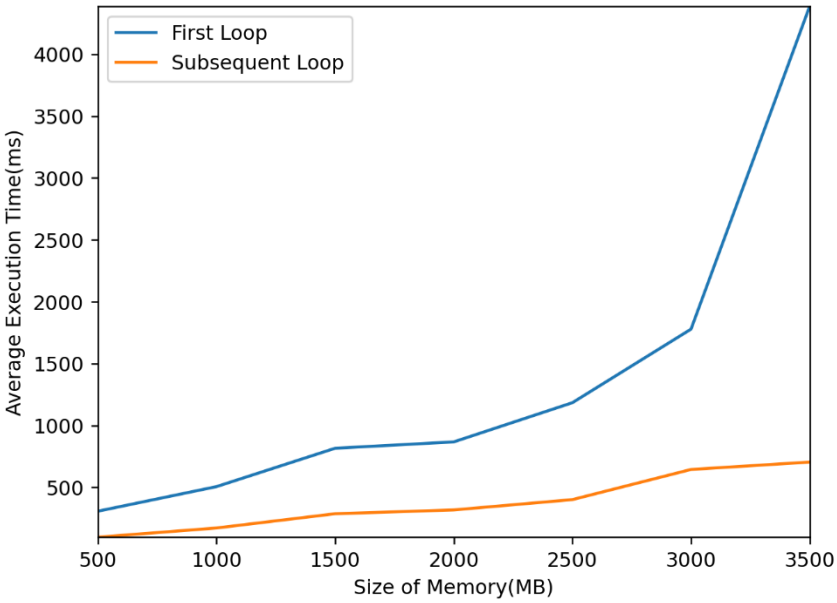
메모리 사이즈를 넘는 값을 mem의 입력으로 넣으면, 다음의 결과를 얻는다. Loop 0 이외에 모든 Loop들에서 상당한 시간이 소요되었으며 bandwidth가 크게 감소하였다. 이를 통해 해당 경우의 성능이 메모리에 완벽하게 맞는 값을 입력으로 넣었을 때에 비해서 확연하게 떨어졌음을 확인하였다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch21$ ./mem 4000
allocating 4194304000 bytes (4000.00 MB)
  number of integers in array: 1048576000
loop 0 in 13037.51 ms (bandwidth: 306.81 MB/s)
loop 1 in 46513.22 ms (bandwidth: 86.00 MB/s)
loop 2 in 44961.97 ms (bandwidth: 88.96 MB/s)
loop 3 in 49934.34 ms (bandwidth: 80.11 MB/s)
loop 4 in 52749.58 ms (bandwidth: 75.83 MB/s)
loop 5 in 53007.32 ms (bandwidth: 75.46 MB/s)
```

x축을 mem이 사용하는 메모리 사이즈로, y축을 bandwidth의 크기로 나타낸 그래프는 다음과 같다. 사용하는 메모리의 크기가 커질수록 average bandwidth가 작아졌으며, 특히 메모리 사이즈를 넘는 값이 입력되었을 때 큰 폭으로 감소하였다.



마지막으로 첫 번째 루프의 성능과 후속 루프들의 성능을 비교하기 위해서 두 경우를 분리해서 측정한 뒤 x축을 mem이 사용하는 메모리 사이즈로, y축을 실행시간으로 하여 그래프로 표현하였다. 이 때, 후속 루프의 성능은 loop 1부터 loop 4까지의 평균 실행 시간을 구해 측정하였다. 두 경우 모두 사용하는 메모리의 크기가 커질수록 Average Execution Time이 오래 소요되었으며, 같은 메모리 크기에 대해 첫 번째 루프가 후속 루프들에 비해서 더 긴 Execution time을 가졌다.



6. Swap space isn't infinite. You can use the tool swapon **with the** -s **flag to see how much swap space is available. What happens if you try to run** mem **with increasingly large values, beyond what seems to be available in swap? At what point does the memory allocation fail?**

다음과 같이 swapon -s를 통해 swap space를 확인하였다. 출력 값을 통해 Swap space의 크기는 1GB정도이고 대략 250MB가 사용되었다는 것을 알 수 있다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch21$ swapon -s
Filename                                Type            Size    Used    Priority
/swap/file                              file            1048576 257996  -2
```

swap으로 사용할 수 있는 수치보다 큰 값으로도 mem이 실행되는 경우가 존재한다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch21$ ./mem 3000
allocating 3145728000 bytes (3000.00 MB)
  number of integers in array: 786432000
loop 0 in 3074.89 ms (bandwidth: 975.64 MB/s)
loop 1 in 6807.73 ms (bandwidth: 440.68 MB/s)
loop 2 in 630.82 ms (bandwidth: 4755.71 MB/s)
loop 3 in 678.13 ms (bandwidth: 4423.96 MB/s)
^C
```

그러나, 메모리의 크기와 swap space의 크기의 합보다 더 큰 값으로 mem을 실행하면 memory allocation에 실패하고 프로세스가 중단된다.

```
borussen@DESKTOP-L834KLC:~/ostep_ch21$ ./mem 5000
allocating 5242880000 bytes (5000.00 MB)
memory allocation failed
```

7. Finally, if you're advanced, you can configure your system to use different swap devices using swapon **and** swapoff**. Read the man pages for details. If you have access to different hardware, see how the performance of swapping changes when swapping to a classic hard drive, a flash-based SSD, and even a RAID array. How much can swapping performance be improved via newer devices? How close can you get to in-memory performance?**

Flash-based SSD를 사용할 경우, classic hard drive(HDD)를 사용하는 경우에 비해서 swap 속도가 훨씬 빠르기 때문에 더 좋은 성능을 보인다. 그러나 SSD는 HDD에 비해서 용량대비 매우 높은 가격을 가지며, 단일 HDD는 데이터를 빠른 속도로 처리하고 장기간 보관하는 데 있어 많은 제약을 받는다. 이러한 문제점을 해결하기 위해 여러 개의 디스크를 묶어 하나의 디스크처럼 사용하는 기술인 RAID array가 등장하였고 어느 정도의 속도와 안정성을 확보하게 되었다.