CIS 452 02 – Lab 9 – Gabrielle Munson and Gabe Baksa

Sample Program 1

1. Determine your system configuration:

- **Specify what eos system you are working on**

  Gabbi's PC

  a. use the free memory utility program to determine and report:

  - **the total amount of physical memory (KB) on your system**

    7965052 KB

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

    7112844 KB

  - **the total amount of swap space**

    2097152 KB

```
gmunson@gabbis-laptop:/mnt/c/Users/munso/OneDrive/Desktop/OS/cis452$ free -k
               total        used        free      shared  buff/cache   available
Mem:         7965052      626584     7112844        3548      225624     7099532
Swap:        2097152           0     2097152
```

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

- **Study the Sample Program**

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

  We estimate that it will be **16,777,216** bytes, or **16 MB** (dim * dim * sizeof(int) or 2048 * 2048 * 4).

- **Start vmstat, make your window appropriately wide, and configure it to display statistics once per second; let the system stabilize**

  a. Note: these experiments are best performed on a "quiet" system (i.e., not many active users)

```
 0  0      0 7470400  11316 153012    0    0     0     0    3   28  0  0 100  0  0
 0  0      0 7470400  11316 153012    0    0     0     0    3   26  0  0 100  0  0
 1  0      0 7454292  11316 153032    0    0     0     0   66   92  4  0 96  0  0
 1  0      0 7454292  11316 153032    0    0     0     0   13   26  6  0 94  0  0
 1  0      0 7454292  11316 153032    0    0     0     0   15   26  6  0 94  0  0
 0  0      0 7469604  11316 153032    0    0     0     0   22   58  5  0 95  0  0
 0  0      0 7469604  11316 153032    0    0     0     0    4   24  0  0 100  0  0
```

- **In another window, execute the Sample Program**

  a. approximately how much does the amount of free (idle) memory change?

The free memory fluctuates by approximately 16 MB, ranging from 7454292 KB to 7470400 KB across the samples

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

This is exactly what we would expect because we allocated for about 16 MB in line 10:

```
10      if ((intPtr = malloc(dim * dim * sizeof(int))) == 0)
```

3. Examine the effect of memory access patterns:

• 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.

a. obtain basic statistics

- **What is the size of a page in Linux?**

  4096 bytes

- **How long does the program take to run?**

  3.05 seconds

• Change the memory access statement in the Sample Program to read:

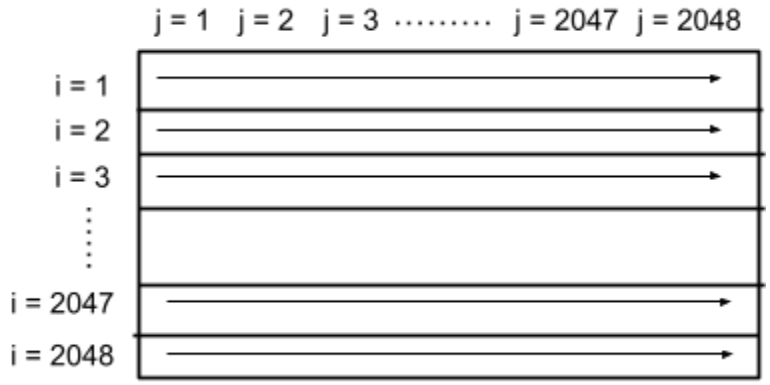
```
intPtr[j * dim + i] = (i + j) % count;
```

then re-compile and re-run the program, collecting statistics on execution time.

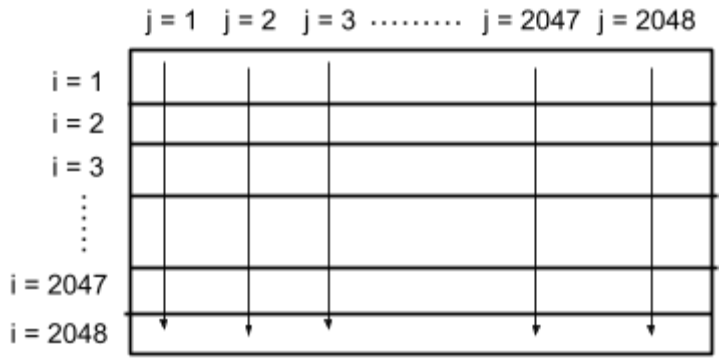
a. Precisely, how does this change alter the program's memory access pattern (i.e. what memory objects get "touched", and in what order)?

In the original code, the memory was accessed row by row whereas in the revised code, the elements are accessed column by column.

Original code accessing the memory:



Altered code accessing the memory:



b. How does this change affect the program's (user) execution time?

It lengthened it from 3.05 seconds to 10.35 seconds

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

The observed effect of the change is that the revised code must access the backing store more frequently than the original code. In the original code, each row is accessed sequentially, which means that the program typically needs to load a page from the backing store only once per row. This leads to fewer page faults and faster execution. In contrast, the revised code accesses memory in a column-major order. As a result, each iteration of the loop often attempts to access memory that is not currently loaded in RAM, leading to frequent page faults. When a page fault occurs, the operating system must take specific actions to retrieve the required page from the backing store (typically a hard disk or SSD) and load it into memory. This process is significantly slower than accessing data already in RAM, which causes the overall execution time of the revised code to be considerably longer than that of the original code.

4. Examine the use of virtual memory:

- **Perform the following**
  - **o Change the memory access pattern for the Sample Program back to its original form**
  - **o Change the LOOP value to 1 (or be prepared to wait a looong time)**
- **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). Note: memory demand should exceed the amount of RAM on your system, but must be less than (RAM + Swap).**

a. What value did you use, justify your computation

Total Allocated Memory=(C×1024)^2 × 4

Due to the following lines 10 + 11:

```
long int i, j, dim = COEFFICIENT * KB;
if ((intPtr = malloc(dim * dim * sizeof(int))) == 0)
```

7965052kb x 1024=(C x 1024)^2 × 4

8156213248**/4**=(C x 1024)^2 × 4**/4**

2039053312= (C x 1024)^2

sqrt(2039053312**)** = **sqrt(**(C×1024)^2**)**

45155.87**/1024** = C×1024**/1024**

44.09 = C

The coefficient of the program must be greater than 44 kb to use up all the physical memory and go into the swap

• 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?

During the running of the program, the amount of free memory slowly declines with each second. After the run, the free memory jumps back up to almost the amount it had before the run and eventually increases back to that amount. The si, so, bi, and bo fields all stay at zero until the swapping of memory begins, same with the cache field. The si and so fields change indicating that memory is being swapped in or out, respectively. The bi field shows some activity nearing the end of the process's execution, indicating some data is being read from a

block device. bo shows a large increase in the final row of the process's execution(288812), suggesting that data is being written to a block device. In the system field, the context switching value begins to make a large jump nearing the end of the process. Once the process starts, you can see the id field of the cpu go down since it is no longer idle and some of the usage moves to the user and the system.

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 understand what is happening from a virtual memory Viewpoint.

The operating system adapts to increased memory demand by using virtual memory, which allows the program to allocate more memory than is physically available through paging. During execution, it experienced **14 major page faults** (requiring loading pages from disk) and **336,782 minor page faults** (accessing pages in virtual memory), leading to a total execution time of **17.03 seconds**. This indicates that the program frequently accessed pages not currently in RAM, significantly impacting performance due to the overhead of page swapping.

```
gmunson@gabbis-laptop:/mnt/c/Users/munso/OneDrive/Desktop/OS/cis452/labs/lab9
 /usr/bin/time -v ./a.out
        Command being timed: "./a.out"
        User time (seconds): 7.51
        System time (seconds): 9.48
        Percent of CPU this job got: 99%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:17.03
        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): 7097788
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 14
        Minor (reclaiming a frame) page faults: 336782
        Voluntary context switches: 32
        Involuntary context switches: 35
        Swaps: 0
        File system inputs: 136
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
```