CS 452 Lab: Week #10
Virtual Memory Concepts

Overview

The purpose of this lab assignment is to conduct and interpret experiments that introduce virtual memory concepts. Specifically, the lab investigates the effect of memory-intensive process execution, frame allocation policies, and page replacement by observing periodic system reports of virtual memory statistics.

Hand-in:

- A word document containing the requested information
- Any requested screenshots imbedded into the word document
- Program source code (no zip files)

Virtual Memory Performance Statistics

Self-monitoring is a task performed by most operating systems - it is useful for administration and auditing purposes. Since there are many subsystems in a typical OS, there are many different performance auditing utilities. This lab concerns virtual memory, hence the primary tool used will be **vmstat** (virtual memory statistics). The utility programs **free** and **top** also give memory usage statistics. Begin by reading the man pages for these utilities to get an idea of the types of information they provide and the various output options available via command-line parameters.

The **vmstat** utility is typically used by system administrators to characterize the performance of a system. For example, to an administrator a large number of swapped processes and a large amount of active virtual memory is indicative of a system with a severe memory shortage (and high paging activity) => time to install more RAM.

Note: there are several different types of memory, and memory measures, referred to in this lab. Be sure you understand the difference and the correct use of each metric. Also, pay special attention to the units; i.e., does a utility report memory usage in bytes, kilobytes (KB), megabytes (MB), etc.

Study the following test program; in particular, note its memory demand requirements (the `malloc()` request) and its memory access profile (the loop). The value `dim` (dimension), is expressed as `COEFFICIENT` Kilobytes, and is used to control memory demand by creating a two-dimensional `dim × dim` matrix of integers. The value `LOOP` is used solely to keep the program executing long enough to observe its memory demand with the **vmstat** utility.

Sample Program

```
#include <stdio.h>
#include <stdlib.h>

#define COEFFICIENT 2
#define KB 1024
#define LOOP 200

int main()
{
   int count, *intPtr;

   long int i, j, dim = COEFFICIENT * KB;

   if ((intPtr = malloc (dim * dim * sizeof(int))) == 0) {
      perror ("totally out of space");
      exit (1);
   }
   for (count=1; count<=LOOP; count++)
      for (i=0; i<dim; i++)
         for (j=0; j<dim; j++)
            intPtr[i * dim + j] = (i + j) % count;

   free (intPtr);
   return 0;
}
```

1. Determine your system configuration:

- Specify what eos system you are working on
  a. use the **free** memory utility program to determine and report:
     - the total amount of *physical* memory (KB) on your system
     - the current amount of *free* memory (KB)
     - the total amount of *swap* space

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?
- 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)
- In another window, execute the Sample Program
    a. approximately how much does the amount of free (idle) memory change?
    b. considering your estimated memory demand of the Sample Program (question 2a), explain why the observed change is an expected result.

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?
        - how long does the program take to run?
- 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)? A diagram will help here.
    b. How does this change affect the program's (user) execution time?
    c. *Precisely*, why does the change have the observed effect (your answer must incorporate an important concept related to paging and virtual memory)?

4. Examine the use of virtual memory:

- Perform the following
    - Change the memory access pattern for the Sample Program back to its original form
    - 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 that (RAM + Swap).

a. What value did you use, *justify* your computation

- 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?

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.