CS333 Spring 2016

Lab 10 (Take Home)

Goal

The goal of this lab is to understand memory management in Linux, and how memory-mapped files work.

Before you start

- Read up on the `proc` file system, especially those files that provide virtual memory statistics, both for the entire machine, as well as per process. Also learn about commands like `free` that present the output from `proc` in an easier-to-follow format.
- Understand how virtual memory and file systems interact with memory-mapped files. Read up on how to memory map a file using the `mmap` command.

Part A: Understanding virtual memory and resident set size

Create a file of size 10MB on your local disk. Write a program that memory maps this file, and then proceed to read and write from the file, in several steps, as mentioned in the exercises below. The program must also pause (e.g., by waiting for the user to hit "Enter", or by using breakpoints with `gdb`) after each step / exercise. The purpose of these pauses is for you to go observe the virtual memory statistics of the system after each step, and answer the questions in the exercises below.

Helpful tip: You may want to write the entire code with all the steps (or some subset of the steps) first, and test for correctness. For example, check that the code runs without any crashes correctly, reads and writes the file correctly, and so on. You can then add pauses and answer the exercises below.

NOTE: The exercises below ask you to measure various statistics related to system memory. For all exercises, you must report the measured value, explain how you obtained it (e.g., describe the specific file/line you referred to in the `proc` file system), and justify the value you see by providing a reasonable explanation. Avoid running any other program on your system concurrently, in order to avoid noise in your measurements. Take as many measurements as required to have confidence in your conclusions.

1. First, pause the code execution right after initialization, before the file is memory-mapped. Report how much virtual memory is allocated to the process (referred to as VM-Alloc henceforth) right after starting execution, but before memory mapping the 10 MB file. Also report how much of this is actually resident in the physical RAM of the machine (henceforth referred to as VM-RSS, short for virtual memory resident set size).

2. Next, memory map the file and pause. Report and justify the values of VMAlloc and VM-RSS. For example, you must explain if and by how much these values increase / decrease, and the reason behind the increase / decrease.

3. Next, read and print out the first character of the file, and pause. Report and justify the values of VMAlloc and VM-RSS (as in the previous exercise).

4. Next, read and print out the character at offset 10,000 bytes in the file, and pause. Report and justify the values of VMAlloc and VM-RSS.

Part B: Understanding the performance of memory-mapped files

Now, create 25 files of 10MB each on your local disk. Then, extend your program to memory map all the 25 files. Do the following exercises that involve reading and writing from the 25 10MB files. (You may use fewer than 25 files if you find that you are running out of memory when running any of these experiments.)

Note: when you measure performance access of reading / writing files from disk, the performance is highly dependent on whether the files are already in the disk buffer cache. So we will ask you to perform experiments by clearing the buffer cache in some exercises below. Have those commands handy.

Helpful tip: most of the exercises below can be solved by writing just one or two programs, and running them with different commandline options, instead of writing separate code for each exercise.

1. Clear disk buffer cache. Read (or, access) all the data in all the 25 memory-mapped files once (one or a small number of bytes at a time). There is no need to print out or do anything with the data, just accessing it will do. Calculate the average throughput of the reads of the memory-mapped files (in MB/s), by measuring the time taken for all the reads to complete. Provide a suitable explanation for this value of average throughput (e.g., why is it not lower or higher, what is the bottleneck, etc.).

2. Now clear the buffer cache. Read all the 25 files from disk by opening them as regular (not memory-mapped) files. You may reuse the disk benchmarking code you have written in earlier labs. You can read the files from disk in block sizes that are multiples of 512 bytes (e.g., 1024 bytes). What is the average disk read throughput (MB/s) in this case? Compare it with your answer in exercise 1, and explain why the throughput values are similar or different.

3. From the exercises above, comment on whether using memory-mapped files gives you any performance benefits over regular files, when your application workload consists mostly of disk reads, and when the workload file content is not usually found in the disk buffer cache.

4. Notice what happens to the size of the disk buffer cache before and after exercises 1 and 2. Using your observations, explain how the disk buffer cache is used when reading regular and memory-mapped files.

5. Clear the disk buffer cache. Next, write over all the data in all the 25 memory-mapped files once (one or a small number of bytes at a time). You can replace the file content by some other content; the exact content you write isn't important. You must not change the size of the files, however, but only rewrite the bytes. Calculate the average throughput of the writes of the memory-mapped files (in MB/s), by measuring the time taken for all the writes to complete. Provide a suitable explanation for this value.

6. Clear the disk buffer cache. Now, write over the contents of all the files by opening them as regular (not memory-mapped) files. You may reuse the disk benchmarking code you have written in earlier labs. You can write the files to the disk in block sizes that are multiples of 512 bytes (e.g., 1024 bytes). What is the average disk write throughput (MB/s) in this case? Compare it with your answer in exercise 5, and explain the reason for any differences.

7. From the exercises above, comment on whether using memory-mapped files gives you any performance benefits over regular files, when your application workload consists mostly of disk writes, and when the workload file content is not usually found in the disk buffer cache.

8. Repeat exercises 5–7 by using a small number of files (say, 2–3), and ensuring that the files are already present in disk buffer cache before you start writing (say, from a previous read). Do memory-mapped files give you any performance benefits over regular files when the application workload consists of frequent writes to a small amount of data that can mostly fit into the buffer cache?

Submission and Grading

You may solve this assignment in groups of one or two students. You must submit a tar gzipped file, whose filename is a string of the roll numbers of your group members separated by an underscore. For example, `rollnumber1_rollnumber2.tgz`. The tar file should contain the following:

- `report.pdf`, which contains your answers to the exercises above. Be clear and concise in your writing.

Evaluation of this lab will be based on reading your answers to the exercises in your report.