Project

Deadlines

1. Form Groups: **Friday, Oct 8th**, submit your group here .
2. Draft of Intro, Machine Description, and CPU Operations: **Tuesday, Oct 19th at 2:00pm** (start of class)
3. Draft of Memory Operations: **Tuesday, Nov 9th at 2:00pm** (start of class)
4. Final report with all measurements plus code: **Friday, Dec 3rd at 11:59pm---- Extended to Dec 5th at 11:59pm**

Overview

In building an operating system, it is important to be able to determine the performance characteristics of underlying hardware components (CPU, RAM, disk, network, etc.) and to understand how their performance influences or constrains operating system services. Likewise, in building an application, one should understand the performance of the underlying hardware and operating system, and how they relate to the user's subjective sense of that application's "responsiveness". While some of the relevant quantities can be found in specs and documentation, many must be determined experimentally. While some values may be used to predict others, the relations between lower-level and higher-level performance are often subtle and non-obvious.

In this project, you will create, justify, and apply a set of experiments to a system to characterize and understand its performance. In addition, you may explore the relations between some of these quantities. In doing so, you will study how to use benchmarks to usefully characterize a complex system. You should also gain an intuitive feel for the relative speeds of different basic operations, which is invaluable in identifying performance bottlenecks.

You have complete choice over the operation system and hardware platform for your measurements. You can use your laptop that you are comfortable with, an operating system running in a virtual machine monitor, or even a supercomputer.

You may work either alone or in 2-3 person groups. Groups do the same project as individuals. All members receive the same grade. Note that working in groups may or may not make the project easier, depending on how the group interactions work out. If collaboration issues arise, contact me as soon as possible: flexibility in dealing with such issues decreases as the deadline approaches.

This project has two parts. First, you will implement and perform a series of experiments. Second, you will write a report documenting the methodology and results of your experiments. When you finish, you will submit your report as well as the code used to perform your experiments.

Report

Your report will have a number of sections including an introduction, a machine description, and descriptions and discussions of your experiments.

1) Introduction

Describe the goals of the project and, if you are in a group, who performed which experiments. State the language you used to implement your measurements, and the compiler version and optimization settings you used to compile your code. If you are measuring in an unusual environment (e.g., virtual machine, Web browser, compute cloud, etc.), discuss the implications of the environment on the measurement task (e.g., additional variance that is difficult for you to control for). Estimate the amount of time you spent on this project.

2) Machine Description

Your report should contain a reasonably detailed description of the test machine(s). The relevant information should be available either from the system (e.g., `sysctl` on BSD, `/proc` on Linux, System Profiler on Mac OS X), or online . Gathering this information should not require much work, but in explaining and analyzing your results you will find these numbers useful. You should report at least the following quantities:

1. Processor: model, cycle time, cache sizes (L1, L2, instruction, data, etc.)
2. Memory bus
3. I/O bus
4. RAM size
5. Disk: capacity, RPM, controller cache size
6. Network card speed
7. Operating system (including version/release)

3) Experiments

Perform your experiments by following these steps:

1. Estimate the base hardware performance of the operation and cite the source you used to determine this quantity (system info, a particular document). For example, when measuring disk read performance for a particular size, you can refer to the disk specification (easily found online) to determine seek, rotation, and transfer performance. Based on these values, you can estimate the average time to read a given amount of data from the disk assuming no software overheads. For operations where the hardware performance does not apply or is difficult to measure (e.g., procedure call), state it as such.
2. Make a guess as to how much overhead software will add to the base hardware performance. For a disk read, this overhead will include the system call, arranging the read I/O operation, handling the completed read, and copying the data read into the user buffer. We will not grade you on your guess, this is for you to test your intuition. (Obviously you can do this after performing the experiment to derive an accurate "guess", but where is the fun in that?) For a procedure call, this overhead will consist of the instructions used to manage arguments and make the jump. Finally, if you are measuring a system in an unusual environment (e.g., virtual machine, compute cloud, Web browser, etc.), estimate the degree of variability and error that might be introduced when performing your measurements.
3. Combine the base hardware performance and your estimate of software overhead into an overall prediction of performance.
4. Implement and perform the measurement. In all cases, you should run your experiment multiple times, for long enough to obtain repeatable measurements, and average the results. Also compute the standard deviation across the measurements. Note that, when measuring an operation using many iterations (e.g., system call overhead), consider each run of iterations as a single trial and compute the standard deviation across multiple trials (not each individual iteration).
5. Use a low-overhead mechanism for reading timestamps. All modern processors have a cycle counter that applications can read using a special instruction (e.g., rdtsc). Searching for "rdtsc" in Google, for instance, will provide you with a plethora of additional examples. Note, though, that in the modern age of power-efficient multicore processors, you will need to take additional steps to reliably use the cycle counter to measure the passage of time. You will want to disable dynamically adjusted CPU frequency (the mechanism will depend on your platform) so that the frequency at which the processor computes is deterministic and does not vary. Use "nice" to boost your process priority. Restrict your measurement programs to using a single core.
   - More tips on measuring time from Prof. Voelker

In your report:

1. Clearly explain the methodology of your experiment.
2. Present your results:
   1. For measurements of single quantities (e.g., system call overhead), use a table to summarize your results. In the table report the base hardware performance, your estimate of software overhead, your prediction of operation time, and your measured operation time.
   2. For measurements of operations as a function of some other quantity, report your results as a graph with operation time on the y-axis and the varied quantity on the x-axis. Include your estimates of base hardware performance and overall prediction of operation time as curves on the graph as well.
3. Discuss your results:
   1. Cite the source for the base hardware performance.
   2. Compare the measured performance with the predicted performance. If they are wildly different, speculate on reasons why. What may be contributing to the overhead?
   3. Evaluate the success of your methodology. How accurate do you think your results are?
   4. For graphs, explain any interesting features of the curves.
   5. Answer any questions specifically mentioned with the operation.
4. At the end of your report, summarize your results in a table for a complete overview. The columns in your table should include "Operation", "Base Hardware Performance", "Estimated Software Overhead", "Predicted Time", and "Measured Time". (Not required for the draft.)
5. State the units of all reported values.

Do not underestimate the time it takes to describe your methodology and results.

4) Operations

1. CPU, Scheduling, and OS Services
   1. Measurement overhead: Report the overhead of reading time, and report the overhead of using a loop to measure many iterations of an operation.
   2. Procedure call overhead: Report as a function of number of integer arguments from 0-7. What is the increment overhead of an argument?
   3. System call overhead: Report the cost of a minimal system call. How does it compare to the cost of a procedure call? Note that some operating systems will cache the results of some system calls (e.g., idempotent system calls like getpid), so only the first call by a process will actually trap into the OS.
   4. Task creation time: Report the time to create and run both a process and a kernel thread (kernel threads run at user-level, but they are created and managed by the OS; e.g., pthread_create on modern Linux will create a kernel-managed thread). How do they compare?
   5. Context switch time: Report the time to context switch from one process to another, and from one kernel thread to another. How do they compare? In the past students have found using blocking pipes to be useful for forcing context switches.

2. Memory
   1. RAM access time: Report latency for individual integer accesses to main memory and the L1 and L2 caches. Present results as a graph with the x-axis as the log of the size of the memory region accessed, and the y-axis as the average latency. Note that the lmbench paper is a good reference for this experiment. In terms of the lmbench paper, measure the "back-to-back-load" latency and report your results in a graph similar to Fig. 1 in the paper. You should not need to use information about the machine or the size of the L1, L2, etc., caches when implementing the experiment; the experiment will reveal these sizes. In your graph, label the places that indicate the different hardware regimes (L1 to L2 transition, etc.).
   2. RAM bandwidth: Report bandwidth for both reading and writing. Use loop unrolling to get more accurate results, and keep in mind the effects of cache line prefetching (e.g., see the lmbench paper).
   3. Page fault service time: Report the time for faulting an entire page from disk (mmap is one useful mechanism). Dividing by the size of a page, how does it compare to the latency of accessing a byte from main memory?

3. Network
   1. Round trip time. Compare with the time to perform a ping (ICMP requests are handled at kernel level).
   2. Peak bandwidth.
   3. Connection overhead: Report setup and tear-down.

   Evaluate for the TCP protocol. For each quantity, compare both remote and loopback interfaces. Comparing the remote and loopback results, what can you deduce about baseline network performance and the overhead of OS software? For both round trip time and bandwidth, how close to ideal hardware performance do you achieve? In describing your methodology for the remote case, either provide a machine description for the second machine (as above), or use two identical machines.

4. File System
   1. Size of file cache: Note that the file cache size is determined by the OS and will be sensitive to other load on the machine; for an application accessing lots of file system data, an OS will use a notable fraction of main memory (GBs) for the file system cache. Report results as a graph whose x-axis is the size of the file being accessed and the y-axis is the average read I/O time. Do not use a system call or utility program to determine this metric except to sanity check.
   2. File read time: Report for both sequential and random access as a function of file size. Discuss the sense in which your "sequential" access might not be sequential. Ensure that you are not measuring cached data (e.g., use the raw device interface). Report as a graph with a log/log plot with the x-axis the size of the file and y-axis the average per-block time.
   3. Remote file read time: Repeat the previous experiment for a remote file system. What is the "network penalty" of accessing files over the network? You can either configure your second machine to provide remote file access, or you can perform the experiment on a department machine (e.g., APE lab). On these machines your home directory is mounted over NFS, so accessing a file under your home directory will be a remote file access (although, again, keep in mind file caching effects).
   4. Contention: Report the average time to read one file system block of data as a function of the number of processes simultaneously performing the same operation on different files on the same disk (and not in the file buffer cache).

References

During the quarter you will have read a number of papers describing various system measurements, including V, Sprite, microkernels, Scheduler Activations, LRPC, LFS, and IO-Lite. You may find these papers useful as references.

In addition, other papers you may find useful for help with system measurement are:

- John K. Ousterhout, Why Aren't Operating Systems Getting Faster as Fast as Hardware? , Proc. of USENIX Summer Conference, pp. 247-256, June 1990.
- J. Bradley Chen, Yasuhiro Endo, Kee Chan, David Mazieres, Antonio Dias, Margo Seltzer, and Michael D. Smith, The measured performance of personal computer operating systems , Proc. of ACM SOSP, pp. 299-313, December 1995.
- Larry McVoy and Carl Staelin, lmbench: Portable Tools for Performance Analysis , Proc. of USENIX Annual Technical Conference, January 1996.
- Aaron B. Brown and Margo I. Seltzer, Operating system benchmarking in the wake of lmbench: a case study of the performance of NetBSD on the Intel x86 architecture , Proc. of ACM SIGMETRICS, pp. 214-224, June 1997.
- John Ousterhout, Always Measure One Level Deeper , Communications of the ACM, Vol. 61, No. 7, July 2018, pp. 74–83.

You may read these papers, or other references, for strategies on performing measurements, but you may not examine code to copy or replicate the implementation of a measurement. For example, reading the lmbench paper is fine, but downloading and looking at the lmbench code violates the intent of the project.

Finally, it goes almost without saying that you must implement all of your measurements. You may not download a tool to perform the measurements for you.

Grading

We will grade your project on the relative accuracy of your measurement results (disk reads performing faster than the buffer cache are a bad sign) as well as the quality of your report in terms of methodology description (can we understand what you did and why?), discussion of results (answering specific questions, discussing unexpected behavior), and the writing (lazy writing will hurt your grade).

In the past, a frequent issue we see with project reports is that they do not clearly explain the reasoning behind the estimates, methodology, results, etc. As a result, we do not fully understand what you did and why you did it that way. Be sure to explain your reasoning as well.

Checkpoint 1: In the first stage of the project, we would like you to submit an early draft of the first part of the project. What should you cover in the draft? The first two parts of the report (Introduction and Machine Description), and the first set of operations (CPU, Scheduling, and OS Services). For this step only submit a draft of the report, not your code.

What percentage of the project grade does it form? It will only be worth 5% of your grade. Why so little? The idea with the initial draft is that it is primarily for your own benefit: it will get you started on the project early, and it will give you a sense for how long it will take you to complete the project by the end of the quarter (in the past, students have reported that it has taken them 40-120 hours on the project). As a result, you should be able to better budget your time as the end of the quarter arrives. How rough can the draft be? Your call. Again, this is primarily for your benefit.

Checkpoint 2: In the second stage of the project, extend your report draft with results for the second set of operations (Memory).

Paper/report format: It is recommended to use the new USENIX paper template (2019 conferences).

Submission guidelines: please submit it on Canvas. No need to submit code.

For the final project reports, submit them with your code on Canvas, packaged either as a tar.gz or a zip file. In the root directory of the code package, provide a `README` file on how to compile and run your code (e.g. `Makefile` and a shell script to run the experiments) and briefly describes the functionality of each code file.