Introduction

This homework is comprised of two parts. In both parts, we are going to measure certain properties of the hardware or kernel. In the first part, the focus is on the computer's CPU and memory hierarchy, namely the hardware cache and main memory. In the second part, we are going to focus on Linux's signals.

Part 1

As you know from class, CPUs operate at a faster rate compared to the main memory. In some implementations, this speed discrepancy is of several orders of magnitude.

To run a program, the CPU needs to fetch an instruction and possibly data from the memory at each clock cycle. Without caching instruction and data on a faster memory, the CPU has to stall on every instruction until the data is available to it. This greatly diminishes the speed of a program. That's why most commercial CPUs contain a faster memory called CPU cache that stores instruction and data.

CPU caches are transparent to the application programmer. However, almost all programs, to varying degree, benefit from the inclusion of the cache. If a programmer is aware of the cache layout and the principles of spatial and temporal locality in memory references, she can write her program in a way to benefit more from the cache. This can result in a noticeable decrease of the running time of the program.

Caches are characterized by three parameters which are cache line, capacity/size, and associativity. It is assumed that you know about these parameters. For more information about these parameters you can check any introductory textbook on computer architecture.

In this part you should determine the
- Cache line (aka cache block size)
- Cache size
- Cache miss penalty

Below I outline how this will be done.

- Since the cache is transparent to the application programmer, there is no direct method to measure the cache size. Therefore, the cache should be measured indirectly.
- The way we achieve this is by running a program that accesses the memory with certain patterns. These accesses lead to capacity misses on the cache. By measuring the running time
of the program we can infer the cache size.

- Assume that the cache line and size are a power of two.
- Make sure that the memory used in the program is already mapped into the page table before measuring the running time.

For this part send the programs with a makefile. You should also include a short report about your methodology and results (namely, the size of the cache line and the size of the cache, and how you determined them). This report should also contain a short description about each c file and the main function's input argument(s).

Part 2

As you know from class, a signal, aka software interrupt/exception, is a notification to a process that an event has occurred. Just like hardware interrupts, signals interrupt the normal flow of execution of a program. A signal can be sent by different sources:

- The kernel
- Some process in the system
- A process sending a signal to itself

In this assignment, we are going to focus on signals sent by the kernel to the process. Different types of events can cause the kernel to send a signal to a process:

- A hardware exception: For example executing malformed instructions, dividing by 0, or referencing inaccessible memory locations.
- Typing the terminal’s special characters: For example the interrupt character (usually Ctrl-C) and the suspend character (usually Ctrl-Z)
- A software event: For example a timer going off or a child of the process terminating.

There are different options when handling a signal: the default action occurs (for example terminating the process upon receiving SIGINT), the signal is ignored, or a user specified signal handler is executed. There are a lot of details about signals, which you can look up online.

In this part, you will write a program to measure the cost of a software interrupt. Since you do not have access to the kernel, what you can measure from the user space is the cost of a signal handler (which most likely will include the cost of exception handling in the kernel plus the preparation of the signal stack for the signal handler call and the cost of executing the handler itself).

To achieve this, you will have to set a handler for a signal (say for the divide by zero exception). If the handler does nothing to fix the cause of the exception, the process will return to the faulty instruction after the execution of the signal. This causes the same exception and another call to the handler. The running time of a single exception handler is too small to be measured accurately; therefore, you have to measure the time to execute a large number of exception handlings and calculate an average. To accomplish this, the handler should count the number of times it’s called. After the handler is executed the desired number of times, the handler must force the process to skip the faulty C
instruction (note that a C instruction is most likely translated into more than one assembly instruction). Skipping over the faulty instruction can be achieved by changing the value of the return address on the signal frame. The return address sets the value of the IP (Instruction Pointer) after the execution of the handler. Without modification the return address’ value is the address of the faulty instruction. The return address is located at a certain offset from top of the stack, which can be determined by using the signal frame structure (found on some header file) or, empirically, by inspecting the handler stack during the handler’s execution.

For this part send your program with a makefile. Also include a short report about your results and the program’s description.

Notes

- For part one use of any instruction or file (e.g. /proc/cpuinfo) that gives away the cache size is prohibited. In principle your solution should be portable to any CPU and OS pair. Use of said methods is not portable.
- Since the programs in this homework make use of time functions, you should not use a VM (Virtual Machine) for running the programs.
- Because of Architectural complexities, it is suggested to run your programs in the 32-bit version of Linux.
- There is no need to measure the size of the L1 cache.
- Use of setjmp and longjmp() is prohibited in part 2.
- All the programs should be written in the C programming language.