Marc-Andre Fichtel - 30014709
CPSC 457 - Fall 2017

Assignment 1

Due Friday, Sep 29, 2017 at 11:59pm.
Individual assignment. No group work allowed.
Weight: 5% of the final grade.

Question 1:

1 second (1s) = 1 billion nanoseconds (1.000.000.000ns). The average number of instructions that can be executed per second is determined by the pipeline bottleneck, which is the Fetch Unit. It takes 10ns to fetch an instruction. Since the pipeline allows the three units to work independently and in parallel, the Decode and Execute Units' time does not have an impact on the end result. So the number of instructions executed per second on average is 1.000.000.000 / 10 = 100.000.000 (one hundred million).

To improve the performance of this CPU, the Fetch Unit (bottleneck) has to be improved. Fetch Units fetch instructions from main memory, which has a typical access time of 10ns. If the instruction set could be stored in a set of CPU registers or the CPU cache (both of which have lower typical access times than main memory), then the overall performance could be significantly improved. However, the cache and especially registers are very expensive memory spaces with a very limited capacity, and are furthermore used for other purposes already. Thus, improving the Fetch Unit in this way seems unlikely with current technologies.

Question 2:

The benefits of using virtual machines from an OS perspective entail the following:

- The host system (i.e. OS) is protected from any VM it may be running. Thus, unsafe or potentially dangerous programs can be run in a VM without security concerns.
- By using multiple independent VMs, an OS can concurrently run applications that would conflict with one another, if they were being run without the use of multiple VMs. Likewise, by using VMs in this way, multiple OSes or OS versions can be run on the same machine at the same time.

From a user point of view, using virtual machines has the following advantages:

- Using VMs can potentially save a lot of money for big companies, who may choose to buy and operate one big server rather than many smaller ones. This is called 'System Consolidation'.
- VMs offer an ideal opportunity for OS research and development, because the underlying system's operations only rarely need to be disrupted from the system development.

Marc-Andre Fichtel - 30014709
CPSC 457 - Fall 2017

<u>Question 3:</u>

Interrupts (i.e. software interrupts) are external events that are delivered to the CPU by devices (often I/O devices). They may originate from I/O, timers, or user input, and are asynchronous with the current activity of the CPU. The time of the interrupt is unknown and unpredictable. Interrupts increase CPU performances, because a CPU doesn't need to wait for, for example, I/O, but can instead do other things until an interrupt is received.

Traps on the other hand are internal events. They usually originate from system calls, but may also do so from error conditions (such as a division by zero). As opposed to Interrupts, Traps are synchronous with the current activity of the CPU, and may occur during the execution of a machine instruction. A Trap switches from User to Kernel Mode and invokes a predefined kernel routine configured by the OS. Once the routine is completed, User Mode is restored.

The actions of both interrupts (such as I/O) and traps (executing a kernel routine) are performed by the kernel, so both interrupts and traps need to be handled in kernel mode. User mode would simply not allow for the actions to be performed.

<u>Question 4:</u>

Given that countLines.cpp was compiled into an executable called countLines, the following are the outputs of the given commands (as executed on my personal Laptop running Ubuntu 16.04.1).

- `time wc -l romeo-and-juliet.txt`
    - `4853 romeo-and-juliet.txt`
    - `real  0m0.003s`
    - `user  0m0.004s` → Time spent in user mode
    - `sys   0m0.000s` → Time spent in kernel mode
- `time ./countLines romeo-and-juliet.txt`
    - `4853 romeo-and-juliet.txt`
    - `real  0m0.046s`
    - `user  0m0.012s` → Time spent in user mode
    - `sys   0m0.036s` → Time spent in kernel mode

The wc program is faster than the C++ program, because it does not need to switch to kernel mode and back, as indicated by the line "`sys      0m0.000s` ". The C++ program, on the other hand, switches to kernel mode and back, which takes much longer.

Marc-Andre Fichtel - 30014709
CPSC 457 - Fall 2017

Question 5:

See file “myWC.cpp” for my improvements onto the countLines program.

The following is the runtime breakdown of “myWC.cpp” (again, run on my personal Laptop):

- `time ./myWC romeo-and-juliet.txt`
    - `4853 romeo-and-juliet.txt`
    - `real  0m0.005s`
    - `user  0m0.004s` → Time spent in user mode
    - `sys   0m0.000s` → Time spent in kernel mode

Compared to the other programs, “myWC” performs much closer to “wc” than “countLines”. The “countLines” program uses the “`read()`” function to read each character individually from the file. Every one of these function calls results in a system call (one for each character read!), which triggers the switch into (and later out of) kernel mode and thus makes this program take much longer.

My solution for this performance issue in “myWC” is to instead include the “`<fstream>`” header, which let's the program open the input file as an “`ifstream`”, and subsequently allows for the “`getline()`” command to be used on the stream to read each line individually. This, I believe, does not trigger a system call for each character read, but rather only once, when the file is opened as an input file stream. The “`getline()`” command reads a line from the input file stream, and so doesn’t cause a system call (as far as I can tell at this point). This significantly increases the performance, bringing the time taken by “myWC” much close to that of “wc”.