University of Calgary

CPSC 457 - Principles of Operating Systems

Professor: Dr. Pavol Federl

Assignment #1

Evan Loughlin

Student ID: 00503393

Tutorial Section: T04

TA: Sina Keshvadi

January 29, 2018

Q1

Assume that a CPU cycle has three stages, and each stage is handled by a separate unit. Namely: fetch unit, decoding unit, and execute unit. For every instruction, the fetch unit takes 10 nsec, the decoding unit takes 0.5 nsec, and the execute unit takes 1 nsec.

a) How many instructions per second can this CPU execute on average if the three stages are not parallelized?

If the three stages are not parallelized, the instructions must be executed sequentially.

$$t_{cycle} = t_{fetch} + t_{decode} + t_{execute}$$
$$t_{cycle} = 10ns + 0.5ns + 1ns = 11.5ns = 0.0000000115s$$
$$f = \frac{1}{t_{cycle}} = \frac{1cycle}{0.0000000115s} = 86,956,521\frac{cycles}{s}$$

b) How many instructions per second can this CPU execute on average if the three stages are operating in parallel?

If the instructions are parallelized, then the length of each cycle can be shortened as some units can be completed at the same time as other units. In this instance, the fetch unit takes the majority of the total time of the cycle, so the fetch unit can be executed in parallel with the other two units.

$$t_{cycle} = min(t_{fetch} + t_{decode} + t_{execute})$$
$$t_{cycle} = t_{fetch} = 10ns = 0.00000001s$$
$$f = \frac{1}{t_{cycle}} = \frac{1cycle}{0.00000001s} = 100,000,000\frac{cycles}{s}$$

Q2

a) What are the benefits of using a virtual machine from the operating system's perspective?

Some of the benefits of using a virtual machine, from the operating system's perspective, are as follows:

- Generally speaking, the operating system being run doesn't "know" whether it's running on actual hardware, or virtualized hardware. So from the OS's point of view, nothing needs to be changed or adapted.
- The host system is (in most cases) protected from the Virtual Machines. For instance, it's possible (although no foolproof) to run unsafe programs inside a VM without it impacting the host system. Furthermore, this allows for more options in testing bugs, OS development, research, through (but not limited to) intentionally "failing" the operating system.
- The operating system can run independent on the system level architecture and hardware of the host system. This means that if the OS is designed for particular hardware, that the necessary hardware can be virtualized to fit the OS.

- It's possible to run multiple different OS's simultaneously on the same hardware.
- The hypervisor (software or hardware) which manages VMs can be done bare-metal, hosted (running on top of another OS), or as a "hybrid", for example using the linux kernel to function as a hypervisor.

b) What are the benefits of using a virtual machine from the user's perspective?

There are several benefits of using a virtual machine, from the user's perspective. They are (in addition to some listed above) as follows:

- Not necessary for the user to have specific hardware in order to a run a particular OS, as a Virtual Machine will create a virtual version of the required hardware. And as said previously, multiple different operating systems can run using the same hardware.
- Scalability factor - if you're an enterprise or business using a centralized computer system running multiple different OS's simultaneously, fixing bugs, updating OS, or installing new software can be completed all at once, rather than needing to deploy said updates / upgrades to hundreds or thousands of individual computing devices.
- It's possible to run a virtual machine within another operating system, reducing the difficulty of the user to access said OS.

Q3

a) Define interrupts.
A signal sent by hardware (devices such as the hard disk, I/O devices, graphics card, etc.) to the processor indicating that an event needs immediate attention. An interrupt is generally used to alert the processor of a high-priority condition requiring the processor to attend to it immediately. The processor responds by temporarily suspending its current activity, saving its state, and executing an interrupt handler (a function used to deal with the event). Once the interrupt has been handled, the processor resumes normal activities.

b) Define traps.
A trap is a software-generated interrupt, which can be used to call operating system routines or catch arithmetic errors. It is typically caused by a an exceptional condition (such as division by zero, or invalid memory access), and generally results in a switch from *user mode* to *kernel mode.* At this point, the operating system performs some action before returning control to the originating process. System calls are generally implemented as traps in order to execute code in a privileged (kernel) mode that allows for manipulation of critical resources, like peripherals, memory management hardware, etc.

c) Describe the differences between interrupts and traps.

Interrupts and traps are similar in that they both cause the processor to enter kernel mode and execute a service routine, they both save the current state of the CPU, and they both resume the original operations when done.

However, they differ in that interrupts are typically caused by peripheral devices sending asynchronous events. The origins of an interrupt can be I/O devices, timer, or user input. The time of the event is not known and it is not predictable. On the other hand, traps are caused by synchronous operation of the currently executing process. They are internal events (system calls or error conditions, such as division by zero). They occur as a result of execution of a machine instruction.

d) Explain why interrupts and traps are handled in kernel mode instead of user mode.

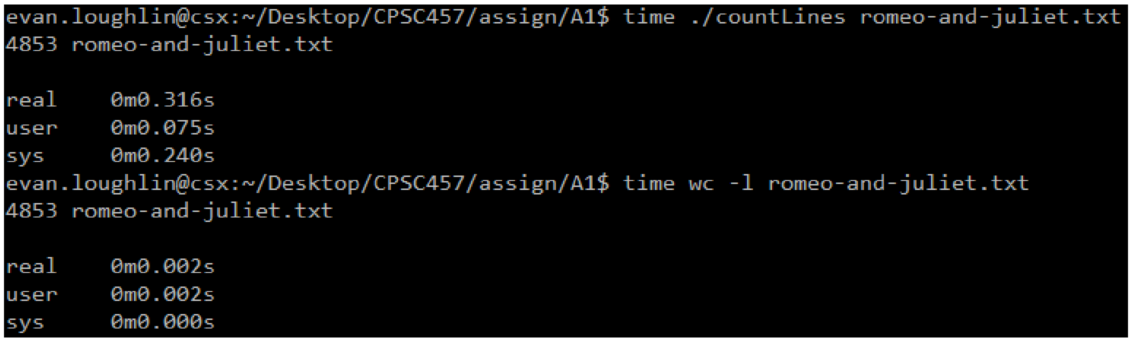
In Kernel Mode, the executing code has unrestricted access to the underlying hardware, and can execute and CPU instruction and/or reference to memory address. Kernel mode is generally reserved for the lowest level, most trusted functions of the operating system.

In User Mode, the executing code is not capable of directly accessing memory or hardware, and must do so using system calls to the underlying API.

This distinct difference is why interrupts must be handled in kernel mode; because interrupts deal with direct communication with memory and hardware. Traps are often conducted as system calls to the underlying API, and call operating system routines or catch arithmetic errors. In both scenarios, they both require access to the underlying system, and need unrestricted access. Thus it's necessary for them to run in kernel mode.

Q4

a) What are the outputs of the time commands?



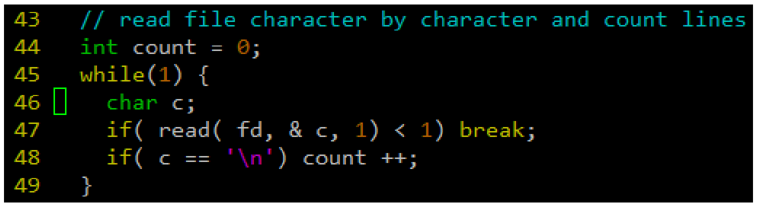
```
evan.loughlin@csx:~/Desktop/CPSC457/assign/A1$ time ./countLines romeo-and-juliet.txt
4853 romeo-and-juliet.txt

real    0m0.316s
user    0m0.075s
sys     0m0.240s
evan.loughlin@csx:~/Desktop/CPSC457/assign/A1$ time wc -l romeo-and-juliet.txt
4853 romeo-and-juliet.txt

real    0m0.002s
user    0m0.002s
sys     0m0.000s
```

b) How much time did the C++ program and 'wc' spend in the kernel mode and user mode, respectively?

The C++ program spent 0.075s in User Mode, and 0.240s in Kernel Mode.
The 'wc' command spent 0.002s in User Mode, and no time in Kernel Mode.

c) Why is the 'wc' program faster than the C++ program?

Because the C++ program needed to use the "read" function, which utilizes a system call to change to kernel mode, for each character that was in the text document. Referring to this code in particular:

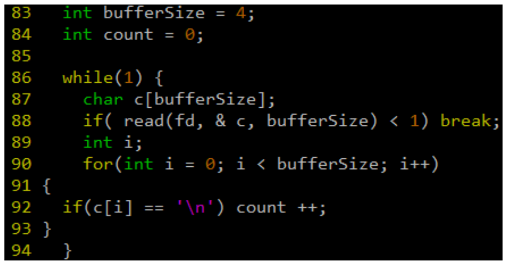


```
// read file character by character and count lines
int count = 0;
while(1) {
  char c;
  if( read( fd, & c, 1) < 1) break;
  if( c == '\n') count ++;
}
```

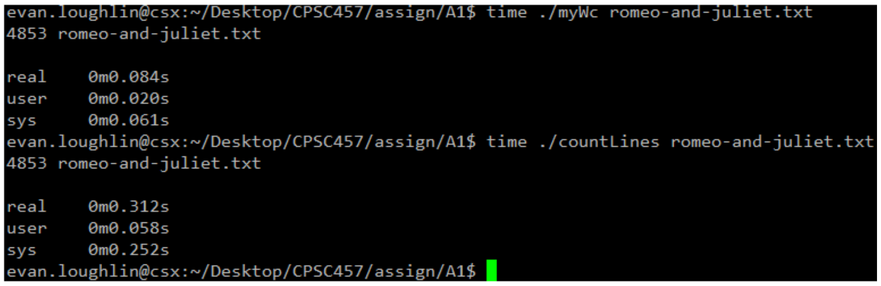
So for each character in the text document, a system call was used via the "read" function. However, the wc command (similar source code for wc found here: `https://www.gnu.org/software/cflow/manual/html_node/Source-of-wc-command.html`), the file is opened only once and all of the contents of the document are read into a single buffer. This allows the countlines operation to be conducted entirely in User Mode, negating the need for additional system calls.

Q5

The countlines.cpp program was modified to myWc.cpp, using only open(), and read(). The program's speed was substantially increased, as seen below. This was done by opening a buffer, then using the read() system call to read more bytes at a time into a char array. Next, the program looped through the array, and counted the total number of times that '/n' was in the array.



```
int bufferSize = 4;
int count = 0;

while(1) {
  char c[bufferSize];
  if( read(fd, & c, bufferSize) < 1) break;
  int i;
  for(int i = 0; i < bufferSize; i++)
{
if(c[i] == '\n') count ++;
}
  }
```



```
evan.loughlin@csx:~/Desktop/CPSC457/assign/A1$ time ./myWc romeo-and-juliet.txt
4853 romeo-and-juliet.txt

real    0m0.084s
user    0m0.020s
sys     0m0.061s
evan.loughlin@csx:~/Desktop/CPSC457/assign/A1$ time ./countLines romeo-and-juliet.txt
4853 romeo-and-juliet.txt

real    0m0.312s
user    0m0.058s
sys     0m0.252s
evan.loughlin@csx:~/Desktop/CPSC457/assign/A1$
```

This solution reduced the time to (USER: 0.084s, SYS: 0.020s) from (USER: 0.058s, SYS: 0.252), since it could read more bytes during each system call, thus reducing the number required.