CMPT 330 Assign 1

Tyler Kuipers

September 25, 2015

Quesiton 1

Multiprogramming is when the computer runs multiple programs at the 'same' time. They cannot actually be running at the exact same time unless your computer has multiple cores in it's processor, or multiple processors. To simulate multiple programs running at once, the processor switches between tasks so quickly that it gives the illusion that they are running at the same time. Arch Linux is the operating system on my personal computer. It is a multiprogramming system because I don't have to wait for one process to complete before I start another one.

Question 2

When every byte of data written or read is handled by the CPU, it makes the CPU have to work a lot more. If everything that wants to access the memory needs to go through the CPU, the CPU ends up doing a lot of extra work. In a multiprogramming system this is bad. It means that we have to wait for every read/write command to complete before we can continue doing what we were doing. Because of DMA (Direct Memory Access), we can use interrupt driven hardware to read/write memory. Through this, we are able to let the CPU block programs and do other things while we read/write from memory.

Question 3

1. Disable all interrupts - This should be handled by the kernel and not the user. Disabling all interrupts could have big implications on the system and directly interacts with the hardware of the system, generally things that directly interact with system hardware should be handled by the kernel and not the user.

2. Read the time of day - This should be run in user mode. The time of day is something that a lot of users check to see. It is not something that the hardware needs to run and needs to be protected by layers of abstraction.

3. Set the time of day - This should be run in user mode. This parameter is not something that needs to be protected by a lot of abstraction because changing the time will not cause the operating system to break.

4. Change the memory map - This should be done with the kernel. Changing the memory map involves hardware, is extremely complicated, and needs to be done by the OS only because if it is not done correctly, it will crash the computer.

Question 4

Superscalar processing is when there are multiple parts of a CPU that are built for certain functions. The CPU then, instead of just pipelining it's processes, will decode the processes, put them in a holding buffer, and then the parts of CPU that specialize in certain areas will go back and check the holding buffer for things that they can do. Through this, we can simulate parallel processing within the CPU. The cost for this operation is complexity. The program can run out f the order defined by the developer. it is up to the hardware to make sure that the results are still the same.

1. One CPU - This program will have a run-time of 5 ms for P0, 10 for P1, and 20 for P2. We can assume that this is our baseline for the following questions. This makes the total time 35ms.

2. One superscalar CPU - These programs will have a maximum runtime of 5ms for P0, 10ms for P1, and 20ms for P2. In a superscalar CPU there may be no speedup if the programs are a repeated set of instructions. The minimum running time these programs have depends on the instruction set. If there are n different sets of instructions and n different areas of the CPU, and the normal running time is t, the run-time for each of the programs is t/n. This makes the total maximum time 35 ms, and the minimum time 35/n.

3. Two CPUs - The programs will each have a run-time of 5ms for P0, 10ms for P1, and 20ms for P2. In a two CPU machine, it will load P0 and P1 into each of the CPUs. P0 will then finish executing and P2 will be loaded into the fist CPU. P1 will then finish and nothing will happen. P2 will then finish and we will have a total time of 25ms. CPU 0: P0 + P2 = 25ms. CPU 1: P1 = 10ms.

4. Two superscalar CPUs - These programs will have a maximum runtime of 5ms for P0, 10ms for P1, and 20ms for P2. In a two CPU superscalar machine there may be no speedup. The minimum running time these programs have depends on the instruction set. If there are n different sets of instructions and n different areas of the CPU, and the normal running time is t, the run-time for each of the programs is t/n. If everything has the minimum speedup, in this machine, it will load P0 and P1 into each of the CPUs. P0 will then finish executing and P2 will be loaded into the fist CPU. P1 will then finish and nothing will happen. P2 will then finish and we will have a total time of 25ms. CPU 0: P0 + P2 = 25ms. CPU 1: P1 = 10ms. If there is a substantial speedup, P0 and P1 will be loaded into their respective CPU's. The time for P0 will be P0/n, and the time for P1 will be P1/n. P2 will then be loaded into whichever of these is finished first, its time

will be P2/n. The minimum time will then be either (P0 + P2)/n or (P1+P2)/n. Whichever is less.

Question 5

How many different instructions this CPU can execute depends on what you consider executing. If executing an instruction is only 1 section of the pipeline, then this pipeline can execute 1000000000-4 instructions per second. If you consider each stage of the pipeline fetching an instruction, then this setup can execute 4000000000 - 4 instructions. The -4 in each of these equations is valid if the pipeline is empty at the start.

Question 6

The trap instruction is an exception in user mode. It is often used when an item in user mode tries to access something in kernel mode. When this happens, the operating system takes over control, runs the command, and after it completes, the user program continues to run. The key difference between a trap and an interrupt is that an interrupt is hardware driven and a trap is user driven.

Question 7

The paper is 50 lines of 80 characters on 1 page. It is 300 pages long. 50*80*300 = 1200000. This paper is 1200000 characters long. Assuming 1 char = 1 byte, this paper is 1200000 bytes, 1200 kilobytes, or 1.2 megabytes long.

1. Registers - If the paper is stored in registers, it will take 1 nanosecond per byte. This paper is 1200000 bytes long giving us a lookup time of 1200000 nanoseconds, or 0.0012 seconds.

2. Cache - If the paper is stored in cache, the lookup time will be 2 nanoseconds per byte. This paper is 1200000 bytes long giving us a lookup time of 2400000 nanoseconds, or 0.0024 seconds.

3. Main Memory - If the paper is stored in memory, the lookup time will be 10 nanoseconds per byte. This paper is 1200000 bytes long giving us a lookup time of 12000000 nanoseconds, or 0.012 seconds.

4. Magnetic Disk - If the paper is stored in disks, the lookup time will be 10 milliseconds per kilobyte. This paper is 1200 kilobytes long giving us a lookup time of 12000 milliseconds, or 12 seconds.

5. Magnetic Tape - If the paper is stored in tapes, the lookup time will be 100 seconds to get to the beginning of the file, and then 10 milliseconds per kilobyte.

This paper is 1200 kilobytes long giving us a lookup time of 12000 milliseconds + 100 seconds, or 112 seconds.

Question 8

The first command will offset the file descriptors by 4 bytes. The entire buffer contains the string 'CMPT 330 Rocks' If the file contains a C string, the buffer will contain the C string without the first 4 characters and is 3 characters in length, the result being ' 330'.

Question 9

Blocking also occurs when writing the disk. The driver first tells the disk what to do when writing, then the controller starts the device, then when they controller is finished writing, it throws an interrupt using the bus-lines that it was given. If we did not block the program during this time, the CPU would end up sitting idle during a large portion of the disk writing process, making the entire computer less efficient.

Question 10

First the arm must move from track 100 to track 50. The arm takes 1 millisecond per track, giving us a total of 50 milliseconds. The disk will then spin for 5 milliseconds to move to the beginning of the file on the disk. This gives us a total of 55 milliseconds to find the file. We then read at 100 mb/s for a 10 megabyte file, this will take 0.1 seconds = 100 milliseconds. 100 milliseconds of finding plus 55 milliseconds of reading will take a total of 155 milliseconds, or 0.155 seconds.