Katie Prescott
Homework 1
CS 492
9 February 2018

1. A computer system has enough room to hold five programs in its main memory. These programs are idle waiting for I/O half the time. What fraction of the CPU time is wasted?

Wait time: 0.5
Number of programs: 5
CPU utilization:

$1 - (0.5)^5 = 0.96875 = 31/32$

CPU time wasted: $1/32 = 3.125\%$

2. On all current computers, at least part of the interrupt handlers are written in assembly language. Why?

Some code, which is implemented during an interrupt, cannot be written in a higher level programming language. It needs to be able to communicate with the kernel.

3. Assume we have a system with a single-core CPU. Multiple jobs can run in parallel (multiprogramming) and finish faster than if they had run sequentially. Suppose that two jobs, each needing 20 minutes of CPU time, start simultaneously. How long will the last one take to complete if they run sequentially? How long if they run in parallel? Assume 50% I/O wait.

Sequentially:

50% wait time -> time needed = $20/0.5 = 40$
The 2nd program will finish 80 minutes (or 1 hour, 20 minutes) after the first program starts.

Parallel:

$1 - (0.5)^2 = 0.375$
Time needed = $20/0.375 = 53.33$ minutes.
The programs will finish in approximately 53.33 minutes; since they are running parallel, they will finish at the same time.

4. If a multithreaded process forks, a problem occurs if the child gets copies of all the parent's threads. Suppose that one of the original threads was waiting for keyboard input. Now two threads are waiting for keyboard input, one in each process. Does this problem ever occur in single-threaded processes?

A single-threaded process cannot fork if it is blocked, waiting for keyboard input.

5. Assume that you are trying to download a large 2-GB file from the Internet. The file is available from a set of mirror servers, each of which can deliver a subset of the file's bytes; assume that a given request specifies the starting and ending bytes of the file. Explain how you might use threads to improve the download time. Are there any possible bottlenecks in your proposed solution?

You can use threads to improve the download time by creating a new thread to download a subset of the files bytes, one thread for each of the mirror servers. Each thread can download their own portion of the file, and spread the CPU utilization across the mirror servers. This solution could bottleneck if one of the threads is working on a slower server, and the other threads will have to wait for that one to have the whole file.