***Pledge:*** *I pledge my honor that I have abided by the Stevens Honor System.* - **Eric Altenburg**

---

**1: Consider a multiprogrammed system with degree of 6 (i.e., six programs in memory at the same time). Assume that each process spends 40% of its time waiting for I/O. What will be the CPU utilization?**

---

$$\begin{aligned}\text{CPU Utilization} &= 1 - p^n \\ &= 1 - (0.40)^6 \\ &= 1 - (0.004096) \\ &= 0.995904 = 99.5904\%\end{aligned}$$

---

**2: What are the benefits of threads vs processes? What is the biggest advantage of implementing threads in user space? What is the biggest disadvantage?**

---

The biggest benefit to threads is that they are considerably faster than processes in creation time (upwards of 100 times faster). An example of this speed is that for every process it has its own PCB which contains several characteristics of the specific process, and so when switching between processes, that PCB has to be loaded which can take a long time; threads do not have this overhead. Additionally, threads share the same address space which allows them to communicate more easily (specifically with global variables) as opposed to processes which are not in the same shared address space, so communication is much more difficult.

When implementing threads in the user space, the biggest advantage it offers is the speed for switching between threads since they do not have to go through the kernel which can be slow. However, there are disadvantages to implementing in the user space, one of which is the blocking of system calls. An example of this can be with a web server's I/O: Since the kernel does not see the threads, it only sees the processes, if one thread within a process $P_0$ performs a system call, what will end up happening is the kernel will put $P_0$ in the block queue until the system call is completed thus putting the entire web server to sleep.

---

**3: Assume we have a system with a single-core CPU. Multiple jobs can be 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 parallel? Assume 50% I/O time.**

---

**(Sequential)**

$$\begin{aligned}\text{CPU Utilization} &= 1 - (0.5)^1 \\ &= 0.50 = 50\%\end{aligned}$$

Since CPU utilization and I/O wait are both 50%, and the total CPU time is 20 minutes, then that means for each job it will take 20 minutes for both the CPU utilization and 20 minutes for the I/O

wait totaling 40 minutes for each job.
Total amount of time it will take for the second job to finish after the first one started is $40+40 = 80$ minutes.

**(Parallel)**

$$\begin{aligned}\text{CPU Utilization} &= 1 - (0.5)^2 \\ &= 1 - 0.25 \\ &= 0.75 = 75\%\end{aligned}$$

This 75% CPU Utilization is for both of the jobs, therefore, each job has the CPU for 35.7%. From this, the total amount of time the second job will take to complete its execution is $20 * 0.375 = 53.33$ minutes.

---

**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?**

---

This is not a problem for single-threaded processes because once the process is blocked for keyboard input, the process cannot fork.

---

**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?**

---

The set up for downloading such file with threads would be to split up the 2GB file into $n$ equal parts where $n$ is the total number of threads which is also the same as the amount of mirror servers; each thread will therefore take on $\frac{2\text{GB}}{n}$ amount of bytes to download. Then from there, since the file can be hosted on multiple mirror servers, each of the threads can begin to download their respective part of the file from separate mirror servers starting and stopping at the bytes specified by the servers. Once each thread has finished downloading their respective parts of the file, they can begin to stitch together the file based on the start and end bytes.

A limiting factor that might pose as a bottleneck to this design would be the total amount of bandwidth allowed for downloading files. If multiple threads are requesting info from mirror servers, then since these threads are all on the same network, the overall speed would potentially be reduced due to the total amount of traffic coming out from $n$ amount of threads attempting to download a file. Generally, the more threads being used, the more of an issue this poses.

**6: What are the two main functions of an operating system? What is the main function of a hypervisor? What are the differences between hypervisors and operating systems?**

The two main functions of an operating system are that it makes use of the hardware as an almost "extended machine" and as a resource manager for multiple applications that might be open at once. In general, the OS bridges the gap between hardware and application programs, so it makes the hardware usable/manageable. In terms of resource managing, it allows multiple programs to run at once while also managing and protecting the memory, I/O devices, and other resources. It does so by sharing resources in two different ways, in space and time; caching is also heavily used as it drastically increase the performance of the machine.

The hypervisor is mainly used for server farms as it allows multiple virtual machines to be run on one machine.

Although, an operating system and hypervisor might have some overlap, they are generally considered to be very different. An operating system is a layer that runs on the computer that manages and even protects the machine's resources so that multiple applications can be run; it gives the machine an application interface. A hypervisor on the other hand, is a "lighter" layer to the machine that allows it to present a computer appearance in terms of software; a virtual computer. The hypervisor does not add any additional core functionality to the computer like an operating system does, it only fundamentally breaks up the machine into smaller virtual computers.