3.11.1: Describe the actions taken by a kernel to context-switch between processes.

In a nutshell, the most basic sequence follows: enter the kernel context, save the current outgoing process, update the outgoing process's current status, call the kernel's scheduler to manage the queue and select the next ready incoming process, restore the incoming process, update the incoming process's current status, and return to the user context to resume execution.

In more details, a context switch is triggered by an interrupt, which then saves all the CPU registers that might have been used. It then changes the status from running to ready or blocked on the PCB, which would place it in the appropriate queue. The scheduler is then called to determine, based on its policy (round-robin, etc.), which process is to run next. That process's state (incoming), which is saved in its PCB, is then loaded back into the CPU registers. Its status is then updated to running, and the kernel then switches back into the user mode resuming where it left off.

3.11.5: Explain the circumstances under which the line of code marked printf("LINE J") in the figure below will be reached.

The child print "LINE J" will only be called if the line "execlp("/bin/ls", "ls", NULL);" fails. This is because on a successful call, the execlp function does not return to the code. Instead, it replaces the child's memory with the new program defined as "/bin/ls" who will start running from its own address. Thus, anything after execlp function call will be skipped if it is successful.

4.10.7: Using Amdahl's Law, calculate the speedup gain for the following applications:

The formula follows 1/[(1-P) + P/N].

(a) 40 percent parallel with (a) eight processing cores and (b) sixteen processing cores

P = 0.4, N = 8, N = 16

S(8) = ~1.54, S(16) = 1.6

(b) 67 percent parallel with (a) two processing cores and (b) four processing cores

P = 0.67, N = 2, N = 4

S(2) = ~1.5, S(4) = ~2.01

(c) 90 percent parallel with (a) four processing cores and (b) eight processing cores

P = 0.9, N = 4, N = 8

S(4) = ~3.08, S(8) = ~4.71

4.10.9: A system with two dual-core processors has four processors available for scheduling. A CPU-intensive application is running on this system. All input is performed at program start-up, when a single file must be opened. Similarly, all output is performed just before the program terminates, when the program results must be written to a single file. Between start-up and termination, the program is entirely CPU-bound. Your task is to improve the performance of this application by multithreading it. The application runs on a system that uses the one-to-one threading model (each user thread maps to a kernel thread).

(a) How many threads will you create to perform the input and output? Explain.

One thread for the input because it is to open a single file. Similarly, one thread for the output to write the results into a file. So, there will be just two total threads since the two tasks are not concurrent, so there is no benefit in adding additional threads.

(b) How many threads will you create for the CPU-intensive portion of the application? Explain.

When the application is CPU-bound, with four logical processors available, the maximum speedup is possible when all four threads are running, where N = 4.