Abstraction: The Process

Annalise Tarhan

January 25, 2021

1 Run $process - run.py$ with the following flags: $-1\ 5:100, 5:100$. What should the CPU utilization be? Why do you know this?

CPU utilization should be 100% because there are no I/O calls, only CPU instructions.

2 Now run with these flags: $./process - run.py\ \ -l\ 4:100, 1:0$. These flags specify one process with four instructions (all to use the CPU), and one that simply issues an I/O and waits for it to be done. How long does it take to complete both processes?

The time it takes for both processes to finish equals the time for the four CPU instructions plus the time an I/O takes, since the processes run sequentially. The time for an I/O process is actually the I/O time, five time units, plus two units for CPU instructions. The total time is 4+5+2=11 time units.

3 Switch the order of the processes: $-l\ 1:0,\ 4:100$. What happens now? Does switching the order matter? Why?

Yes, the switching matters, because the first process' four CPU instructions can execute during the five time units it takes for the second process' I/O to complete. The total time now is 5+2=7 time units.

4 We'll now explore some of the other flags. What happens when you run the following two processes ($-l\ 1:0, 4:100\ \ -c\ \ -S\ SWITCH\_ON\_END$), one doing I/O and the other doing CPU work?

This causes the processes to execute sequentially instead of concurrently, making the total time 11 time units again.

5 Now, run the same processes, but with the switching behavior set to switch to another process whenever one is WAITING for I/O. What happens now?

Now, they execute concurrently, reducing the total time back to seven time units.

6 One other important behavior is what to do when an I/O completes. With $-I\ IO\_RUN\_LATER$, when an I/O completes, the process that issued it is not necessarily run right away; rather, whatever was running at the time keeps running. What happens when you run this combination of processes? Are system resources being effectively utilized?

System resources are being utilized effectively, but not efficiently. If the first process had been restarted as soon as the first I/O completed, it could have started the second one sooner, reducing the total time.

7 Now run the same processes, but with $-I\ IO\_RUN\_IMMEDIATE$ set, which immediately runs the process that issued the I/O. How does this behavior differ? Why might running a process that just completed an I/O again be a good idea?

This time, the total time is 21 units instead of 31 because all of the CPU instructions were executed while another process was waiting for an I/O to complete. Running a process that just completed an I/O again is a good idea because it might be waiting for the result of that I/O to start another I/O. Execution would be more efficient if the other process' CPU instructions ran at the same time.

8 Now run with some randomly generated processes: $-s\ 1\ \ -l\ 3:50,3:50$ or $-s\ 2\ \ -l\ 3:50,3:50$ or $-s\ 3\ \ -l\ 3:50,3:50$. See if you can predict how the trace will turn out.

The trace is not predictable because the threads could execute CPU instructions or I/O calls depending on the random values chosen by the seed.

8.1 What happens if you use the flag $-I\ IO\_RUN\_IMMEDIATE$ vs $-I\ IO\_RUN\_LATER$?

This doesn't actually make a difference in any of the three cases. With a different combination of instructions, IO_RUN_IMMEDIATE would be faster.

8.2 What happens when you use $-S\ SWITCH\_ON\_IO$ vs $-S\ SWITCH\_ON\_END$?

Switch on IO is significantly faster in all cases. When one thread is waiting for an I/O to complete, the other thread can either complete CPU instructions or start its own I/Os, which means they wait for I/Os to complete concurrently instead of sequentially.