Collin Gros
10-04-2020
cs474

PROJECT 1 REPORT

CODE

> ./main.c

TRIAL DATA

TRIAL 1	P1	P2	P3	P4
COUNTER	118650	221141	325285	500000
PID	23424	23425	23426	23427
TRIAL 2	P1	P2	P3	P4
COUNTER	119841	219490	325707	500000
PID	23429	23430	23431	23432
TRIAL 3	P1	P2	P3	P4
COUNTER	119795	220435	326672	500000
PID	23439	23440	23441	23442

ANALYSIS

After viewing the results from my trial runs, it appears that the last process that is forked (P4) always stops exactly at its while-loop conditional, 500000, which is what I expected of the other 3 processes. However, in all three trials, the other processes were over their while conditional by quite a few numbers. I didn't expect this, but I hypothesize that it is due to the lack of synchronization between the processes. No process is able to increment 'value' slowly enough because all the other processes are incrementing it at the same time. This demonstrates a problem with shared memory, if the user doesn't strictly take control of it and set rules so that every process behaves in a way that is expected.

I also have observed that in my implementation of this code, the PIDs are always in sequential order. This is quite nice, but I am not sure this is what was expected of my code. I have a for loop in my main function where I call a helper function, doproc, to call the appropriate process, depending on the value of 'i', which means every process is in the correct order every time.

I also noticed that the PIDs of these processes are ALWAYS counting up between trials. Looking at the table, the order of trials that I executed was Trial 1 -> Trial 2 -> Trial 3, and the PIDs of every process are in order, from smallest to largest. I am guessing this is because Linux implements a process counter in their code to assign a PID to new processes, and their counter

never subtracts completed processes. I only hypothesize that and am not sure if that's what actually happens or not.