OS Project 1

r08922141

April 2020

1 Design

We run a simulation in the "scheduling loop", where each iteration increments the simulated time by one unit and does the scheduling. The scheduler process only synchonizes with child processes on creation and completion. By design, the scheduler and child processes must run on different cores.

Within each iteration, we check that if any process is ready and creates them. We also check if any scheduling decisions must be made, depending on the policy chosen. Round robin has a queue and (preemptive) shortest job first has a heap to determine the next process to be scheduled. After all has been done, we clear exited processes and increments the simulated time.

The scheduling is based entirely on the presumption that SCHED_IDLE and SCHED_OTHER can "preempt" and "schedule" the child process. This is mostly fine because the only time no child is SCHED_OTHER is in the (relatively small) overhead of the scheduling loop.

The reason that child and scheduler processes doesn't do further synchronization is because the only way to guarantee perfect simulation is by blocking in the scheduling and child processes (which is against the specs), all other forms of synchronizations are all flawed. The scheduling process has no way of estimating how long a simulated time unit is other than actually spinning, and even then, a simulated time unit is highly unstable. This potentially creates an inconsistent state where the simulated time differs in the scheduler and child process, and will lead to completely different scheduling decisions.

Since none of the synchronization options are satisfactory, the simplest form is chosen, which is to not synchronize. This is fine as long as the scheduler and child processes doesn't differ by too much, where the error margin is determined by how coarse the time is in the input.

2 uname -a

Linux linuxlite-virtual-machine 4.15.0-22-generic #24-Ubuntu SMP Wed May 16 12:15:17 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux

3 Comparisons

On the tests provided, the ordering is always one of the possible theoretical orderings. The reason that there is multiple theoretical orderings is because processes with equal scheduling priority may be chosen arbitrarily. The timing is *very* unstable. Consecutive runs results in vastly different timings, this is to be expected due to the nature of spinning and the fact that we do no synchronization during execution.