Homework 6

Erik Brakke

March 22, 2016

Answer 1

1. If a CPU bound process is running when a IO bound job arrives, the scheduler may realize that through HSN, this new IO bound job should be scheduled next. However because it is non-preemptive, the IO bound job must wait for the CPU bound job to finish. This will casue a huge slowdown for the IO bound job.

2. If there are already some jobs in the queue when an IO bound job arrives, The scheduler will attempt to schedule it at that moment. When the IO bound job arrives, it's slowdown is 0 and will be put at the end of the queue. If the IO bound job happened to take the least amount of CPU time, it could not preempt the current job that is running and thus would still have to wait.

3. Assuming a fixed timeout means calculating the slowdown after some fixed amount of time at arrival, this is a better solution because the scheduler is predicting what the slowdown of the job will be. This ensures that fast jobs will preempt the current job that is running. If, for example, many similar length IO bound jobs came in at the same time, each new one would preempt the old one. This is problematic because IO bounds jobs would in general not like to be interrupted as they're using the CPU for such a small amount of time.

4. A fix for this problem would be rather than having a fixed timeout, there could be some sort of weighted timeout. We want to make sure that IO bound jobs that arrive to the system are able to interrupt long jobs, but we do not want IO bound jobs to interrupt other IO bound jobs. So if on average more IO bound jobs are arriving, the timeout can shrink, but when CPU bound jobs start arriving, the timeout can start to grow.

Answer 2

1. Turn = 1, P0 sets flag[0] = 1, P0 interrupted before critical section, P1 sets flag[1] = 1, P1 sees that flag[0] == 1, but that turn == 1 so it will go into the critical section. P0 resumes in critical section and now both P0 and P1 are in the critical section

2. instead of an out if(flag[opposite] == true), we can turn this into a while loop. This ensures that before entering the critical second, the process will make sure the other processes will not enter the critical section

Answer 3

1. Uncomment the code for 'Process' to see the ouput
2. Uncomment the code for 'ProcessEnhancement' in the simulation to see the output
3. Yes, if you make process 0 sleep right after it sets it's flag to false, process 1 can continue to beat it (uncomment part c code in 'ProcessEnhancement')
4. It will never occur because there is never a spot in Peterson where you can make Process 0 sleep and have process 1 continue to run. Process 1 will always wait have to wait for process 0 to finish

Answer 4

Running the trial 1000 times with 95% confidence, the average number of busy waiting was 127382198 $\pm$902560.55