CS330A: Assignment 2

Jaya Meena: 200472
Kajal Deep: 200483
Kumar Arpit: 200532
Pratham Jain: 200712

1. Comparison between non-preemptive FCFS and preemptive round-robin:

   (a) Evaluate batch1.txt for SCHED_NPREEMPT_FCFS and SCHED_PREEMPT_RR. Report the differences in statistics. Explain the observation.

   Solution:

   - SCHED_NPREEMPT_FCFS:
     Batch execution time: 9073
     Average turn-around time: 9069
     Average waiting time: 8157
     Completion time: avg: 9197, max: 9202, min: 9193
   - SCHED_PREEMPT_RR:
     Batch execution time: 8961
     Average turn-around time: 8941
     Average waiting time: 8040
     Completion time: avg: 9007, max: 9026, min: 8977

   We see that the two schedulers behave relatively similarly. FCFS has a slightly larger turnaround time as well as average waiting time. Also, the difference between maximum and minimum completion times for FCFS is somewhat greater.

   Since batch1.txt calls testloop1.c which has large CPU bursts interspersed with small I/O bursts, FCFS behaves a bit like RR itself, albeit with larger "timer intervals" corresponding to the sleep(1) calls which puts the process to sleep and after which it goes back to the end of the ready queue of the RUNNABLE processes. Thus, both schedulers behave like fair schedulers with not so good turnaround times. The slightly greater difference in maximum and minimum completion times corresponds to it being less fair since it is non-premptime itself while the sleep(1) calls causing it to switch processes but slower than the timer interval.

   □

   (b) Evaluate batch2.txt for SCHED_NPREEMPT_FCFS and SCHED_PREEMPT_RR. Report the differences in statistics. Explain the observation.

   Solution:

   - SCHED_NPREEMPT_FCFS:
     Batch execution time: 8459
     Average turn-around time: 8456
     Average waiting time: 7610
     Completion time: avg: 8527, max: 8531, min: 8524
   - SCHED_PREEMPT_RR:
     Batch execution time: 8236
     Average turn-around time: 8233
     Average waiting time: 7409
     Completion time: avg: 8355, max: 8359, min: 8351

   Again, the two schedulers behave much similarly. The only apparent and somewhat significant difference between the two from the observed statistics is that the batch execution time, turn around time and waiting time for FCFS is larger compared to that for RR. Other than that, the difference between maximum and minimum completion time for both are relatively same.

   Again, the yield calls in testloop2.c processes created by batch2.txt act similar to timer intervals causing the currently running process to give up the scheduling round. RR having slightly higher

execution, turn-around and waiting times is probably just a random artifact. Also, the batch2.txt took relatively less time compared to batch1.txt probably because of overhead involved in sleeping and waking of a process. Thus, both the scheduler seem to act like fair ones (even though one is not!).

□

(c) Evaluate batch7.txt for SCHED_NPREEMPT_FCFS and SCHED_PREEMPT_RR. Report the differences in statistics. Explain the observation.

Solution:

- SCHED_NPREEMPT_FCFS:
  Batch execution time: 8217
  Average turn-around time: 4535
  Average waiting time: 3714
  Completion time: avg: 4679, max: 8362, min: 976
- SCHED_PREEMPT_RR:
  Batch execution time: 8403
  Average turn-around time: 8387
  Average waiting time: 7546
  Completion time: avg: 8502, max: 8518, min: 8478

This one is a rather interesting case. While the difference between maximum and minimum completion times for the processes is again small for RR, it is huge for FCFS. Similarly, the average waiting and completion times for FCFS is significantly smaller (roughly 50%) that of FCFS. It is not particularly difficult to explain the above behaviour. Since batch7.txt calls testloop4.c program which has no I/O burst at all except one at the end, once a process is scheduled, FCFS being non-premptive, does not prempt it and it continues it's execution until it exits after which a new process is scheduled. Also, since all arriving processes have same length, this in-fact maximizes throughput (by minimizing turnaround time) but at the cost of not being fair. On the other hand, RR behaves in a fair way as before and has a much larger turnaround time.

□

2. CPU burst estimation error using exponential averaging:

Evaluate batch2.txt and batch3.txt for SCHED_NPREEMPT_SJF and report any observed differences in the CPU burst estimation error. You should be paying attention to how the following ratio changes: (the average CPU burst estimation error per estimation instance)/(the average CPU burst length). Explain what you observe.

Solution:

- batch2.txt:
  Batch execution time: 8680
  Average turn-around time: 8114
  Average waiting time: 7246
  Completion time: avg: 8242, max: 8809, min: 7809
  CPU bursts: count: 59, avg: 147, max: 193, min: 1
  CPU burst estimates: count: 53, avg: 132, max: 171, min: 1
  CPU burst estimation error: count: 49, avg: 63
- batch3.txt:
  Batch execution time: 33554
  Average turn-around time: 24309
  Average waiting time: 20954
  Completion time: avg: 24401, max: 33648, min: 15917
  CPU bursts: count: 210, avg: 159, max: 197, min: 1

CPU burst estimates: count: 201, avg: 158, max: 191, min: 1
CPU burst estimation error: count: 200, avg: 20

There are several notable differences here. First we see that the batch execution time for batch3.txt is about 4 times that of batch2.txt as expected (since it has same number of processes and each process has 4 times the number of iterations in the outer loop). The average turnaround time for batch2.txt is similar to its execution time while it is significantly shorter for batch3.txt. Same for average waiting and completion times as well. The difference between maximum and minimum waiting times for batch2.txt is small however so is not the case for batch3.txt.

The CPU burst count and burst estimate count are similar and comparable (when we take account the length of processes being 4 times in batch3.txt), however, the average CPU burst estimation error and (the average CPU burst estimation error per estimation instance)/(the average CPU burst length) in case of batch2.txt is much smaller. Also, for batch3.txt, the average CPU Burst estimate is extremely close to the actual average CPU burst different from that in batch2.txt.

batch2.txt executes testloop2.c while batch3.txt executes testlooplong.c. Both these are same with long CPU bursts interspersed with yield() calls with the difference that the outer loop in testlooplong.c runs much longer, 4 times than that in testloop2.c. Also, the processes themselves are quite monotonous with evenly dispersed yield() calls. Thus, given the longer time, the SJF scheduler is able to estimate the CPU burst much better and it actually converges pretty close to the actual CPU burst in case of batch3.txt. So, even though the initial estimates by SJF are not so good, they converge, and for batch3.txt processes, the error falls off pretty quickly. Infact, we see that (the average CPU burst estimation error per estimation instance)/(the average CPU burst length) for batch3.txt is roughly one third that of batch2.txt which implies that by the time the outer loop has executed 5 times, the scheduler has already an excellent estimate of the CPU burst and in the next 15 iterations, the burst error is pretty small. (It would have been one fourth in the ideal case when the scheduler was able to estimate the CPU bursts with complete accuracy).

Other than the burst estimation error, we see that the average turn-around, waiting and completion times did not exactly increase linearly with the length of the process/execution time for batch3.c. This is probably because after the burst estimate converges for a process which was chosen because it had the minimum estimate at the time, the other processes still continue to have the higher estimate and the first process is scheduled until completion and this happens for other processes as well. This also affects the fairness of the scheduler and thus the difference between the minimum and maximum completion time of the processes is much higher.

□

3. Comparison between non-preemptive FCFS and non-preemptive SJF:

   Evaluate batch4.txt for SCHED_NPREEMPT_FCFS and SCHED_NPREEMPT_SJF. Report the differences in statistics. Explain the observation.

   Solution:

   - SCHED_NPREEMPT_FCFS:
     Batch execution time: 6242
     Average turn-around time: 6238
     Average waiting time: 5614
     Completion time: avg: 6327, max: 6332, min: 6324
   - SCHED_NPREEMPT_SJF:
     Batch execution time: 6595
     Average turn-around time: 4870
     Average waiting time: 4211
     Completion time: avg: 4978, max: 6705, min: 2786

CPU bursts: count: 59, avg: 111, max: 210, min: 1
CPU burst estimates: count: 50, avg: 106, max: 181, min: 42
CPU burst estimation error: count: 49, avg: 47

We see that the average turnaround and waiting times for SJF are much shorter compared to FCFS. Also, the difference in minimum and maximum completion times for FCFS is negligible while it is extremely large for SJF case.

batch4.txt alternately contains 5 each of processes, testloop2.c and testloop3.c which themselves are almost identical except that testloop3.c has half the number of iterations in the inner loop (and thus roughly half the CPU burst time). Also, it is worth noting that both have same number of outer loop iterations (and also the fact that both schedulers are non premptive). Now since both the processes have same number of outer loop, the yield() call causes the FCFS scheduler to essentially schedule them in a round robin fashion and thus both kinds of processes end almost together in the end and thus such a small difference in minimum and maximum waiting times (and the large turn-around and waiting times as well). However, SJF is able to estimate burst lengths of the process. So, after the first yield() call by each of the processes (until with they all have estimated burst length as 0), the testloop3.c processes gets scheduled first as they have shorter burst lengths and thus all five testloop3.c get scheduled in a round robin like fashion and end after which the testloop2.c programs gets scheduled again (after only one or two initial iterations of the outer loop) and then they end up getting scheduled in a round robin way. We can even justify the numerical observation on average waiting and turn around times on the basis of above arguments (provided we give some leverage because of the execution of the first couple of loops of testloop2.c).

□

4. Comparison between preemptive round-robin and preemptive UNIX:

   (a) Evaluate batch5.txt for SCHED_PREEMPT_RR and SCHED_PREEMPT_UNIX. Report the differences in statistics. Explain the observation.

   Solution:

   - SCHED_PREEMPT_RR:
     Batch execution time: 8721
     Average turn-around time: 8705
     Average waiting time: 7828
     Completion time: avg: 8821, max: 8837, min: 8801
   - SCHED_PREEMPT_UNIX:
     Batch execution time: 8245
     Average turn-around time: 5421
     Average waiting time: 4592
     Completion time: avg: 5493, max: 8319, min: 2547

   We see that the average turn-around and waiting times in UNIX case is less than that in case of RR. Also, the difference between the maximum and minimum completion times of UNIX scheduler is large while for RR its extremely small.

   batch5.txt contains 10 instances of testloop1.c with varying priorities from 10 to 100. testloop1.c itself has large CPU bursts interspersed with small I/O bursts. We know that RR is an (almost) fair scheduler. Thus it has statistics as expected from a fair scheduler which has a large turnaround time. UNIX scheduler on the other hand, gives preference to processes based on their dynamic priority. Thus, it starts by executing the processes with least base priority (and thus the highest PID) and only executes another process when the dynamic priority value of processes with lower base priority increases above another. However, everytime a process goes to sleep(1) in testloop1.c, its priority increases less than what it would have increased in case of a timer interrupt. (Note

that it recovers from the sleep(1) in just 1 cycle and gets ready to be scheduled again and thus will probably be the one to get picked in the next scheduling round since its priority increased

□

(b) Evaluate batch6.txt for SCHED_PREEMPT_RR and SCHED_PREEMPT_UNIX. Report the differences in statistics. Explain the observation.

Solution:

- SCHED_PREEMPT_RR:
  Batch execution time: 8531
  Average turn-around time: 8512
  Average waiting time: 7658
  Completion time: avg: 8601, max: 8620, min: 8563
- SCHED_PREEMPT_UNIX:
  Batch execution time: 8829
  Average turn-around time: 5860
  Average waiting time: 4978
  Completion time: avg: 5957, max: 8927, min: 2799

Again, this is much similar to the above case of batch5.txt. We see that the average turn-around and waiting times in UNIX case is less than that in case of RR. Also, the difference between the maximum and minimum completion times of UNIX scheduler is large while for RR its extremely small.

The RR scheduler behaves almost in the same way as above (in case of batch5.txt) since for it sleep(1) and yield() calls are pretty much equivalent. UNIX scheduler also behaves in pretty much the same way, however, with a small difference. The yield() call here increases CPU time and thus the dynamic priority value of the process more and thus the probability of a program with larger priority to get scheduled is more. Thus the average turnaround time and waiting time of the processes are "slightly" larger.

□

Implementation Details of SCHED_NPREEMPT_SJF:

- As the scheduler is non-preemptive, yield() has been disabled in usertrap() and kerneltrap() in trap.c so that yield() is not called on timer-interrupts.
- Two fields have been added in the process table structure, struct proc in proc.h.:
  - btime - to store the burst start time
  - blength - to store the estimated CPU burst length, initialized to 0 in allocproc().
- The CPU burst of a process starts when the process gets scheduled. Thus, the btime of a process is set whenever it is scheduled i.e, it's state is set to RUNNING in the scheduler(). The CPU burst comes to an end when the process state changes to RUNNABLE in yield(), SLEEPING in sleep(), or ZOMBIE in exit(). Thus, the actual burst length is estimated as current ticks minus btime and stored in the variable t. Then, it is used to update the estimated burst length, blength, of the process using the formula given as $s(n+1) = (1-a) * t(n) + a * s(n)$.
- In the scheduler() function, a variable minlength initialized with INT_MAXis declared to store the minimum burst length estimated and a new process structure is declared as chosenprocess to keep account for the process with the minimum estimated burst length.
- A for loop goes over all the processes. If the process is in the RUNNABLE state, it is scheduled for RUNNING if it is not a batch process, else the minlength and chosenprocess are updated if the blength of the process is less than the minlength.
- In case the scheduler is unable to find any process in the RUNNABLE state, the while loop continues to loop until the scheduler finds one.
- After the loop ends, the chosenprocess is scheduled by the scheduler by setting it's state to RUNNING.

Implementation Details of SCHED_PREEMPT_UNIX:

- Two fields have been added in the process table structure, struct proc in proc.h.:
  - cpuusage - to store CPU usage of the process, initialized to 0 in allocproc().
  - priority - to store dynamic priority of the process
- The CPU usage of the process is updated when it transitions from RUNNING to RUNNABLE in yield() and from RUNNING to SLEEPING in sleep(). In case of yield(), the cpuusage is incremented by 200 whereas in case of sleep(), the increment value is 100.
- In the scheduler() function in proc.c, a for loop runs over all the processes and cpuusage and priority value is updated if the process is in RUNNABLE state using the formula,

  CPU usage = (CPU usage)/2;

  priority = (base priority) + (CPU usage)/2;
- A variable minpriority is declared to store the minimum priority value and a new process structure is declared as chosenprocess to keep account for the process with the minimum dynamic priority value.
- A for loop is again run to choose the process with the minimum priority. In case if a RUNNABLE process is not a batch process, it is scheduled immediately.
- In case the scheduler is unable to find any process in the RUNNABLE state, the while loop continues to loop until the scheduler finds one.
- After the loop ends, the chosenprocess is scheduled by the scheduler by setting it's state to RUNNING.