CS330_Assignment:2

Pratyush Gupta

October 2022

1 Implementation Details

1.1 Implementation of SCHED_NPREEMPT_SJF

While implementing this scheduling policy firstly, the in the kerneltrap and usertrap the call of the yield function was disabled to prevent context switching at the end of the time quanta.

Then we iterate over all the processes in the process table that are currently in runnable state. If we find a process that is not batch process, we schedule it immediately, otherwise we keep a track of the shortest estimated burst length by using the burst length estimation of each process which has been stored in the process table entry of the process. Ties are resolved by scheduling the process that was found first in the process table. Then the subsequently found process with shortest estimated burst length is executed and its execution time is recorded.

With the help of this recorded time and the previously estimated time, the new burst length is estimated and updated in the process table entry of the process.

1.2 Implementation of SCHED_PREEMPT_UNIX

In the implementation of this process, we iterate over all the processes in the process table that are currently in the runnable state. If we find a process that is not batch process, we schedule it immediately otherwise we calculate the current priority of each process with the help of the base priority and the cpu usage, stored in the process table entry. Then the process with the least priority is saved and scheduled.

Ties are broken by comparing the wait time of each process in the process queue. The wait time has also been stored in the process table entry.

We also update cpu usage (cpu usage = cpu usage / 2) of every process in this iteration, if it has changed since the previous run of the scheduler. For this we store the previous cpu usage as well in the process table entry.

To update cpu usage of a process when it transfers from running to runnable state, the respective increment has been done in the yield function. Whereas

to update it when the process transfers from running to sleeping state, the respective increment has been done in the sleep function. s

2 Evaluation

Comparison between non-pre-emptive FCFS and pre-emptive RR

1. **batch1.txt**:

   FCFS
   Batch execution time: 8285
   Average turn-around time: 8284
   Average waiting time: 7450
   Completion time: avg: 9043, max: 9045, min: 9041

   RR

   Batch execution time: 8309
   Average turn-around time: 8294
   Average waiting time: 7457
   Completion time: avg: 8983, max: 8998, min: 8941

   Batch 1 consists of testloop1 (Long CPU bursts with small IO bursts). The start time for RR is same for the processes in the batch as expected. For FCFS the start time of processes differ by the length of 1st CPU burst of previously scheduled process. Since there is very little computation after the last sleep call, for both FCFS and RR all the processes roughly finish at the same time, as a result the batch execution and the average turnaround time of both is comparable. Slight advantage in average turnaround time and batch execution time for FCFS may be explained by the frequent context switch overhead involved in RR. The smaller difference between max and min times for FCFS can be explained by the fact that FCFS takes all the processes to the last IO burst before excuting the last IO burst for any process, since last IO burst takes very small amount of time, the differences in min, max is very low. No such gurantee is given by RR.

2. **batch2.txt**:

   FCFS

   Batch execution time: 8193
   Average turn-around time: 8191
   Average waiting time: 7372
   Completion time: avg: 9188, max: 9191, min: 9186

RR

Batch execution time: 8230
Average turn-around time: 8209
Average waiting time: 7385
Completion time: avg: 10630, max: 10651, min: 10579
Batch 2 consists of testloop2 (Long CPU bursts with yields). The analysis is identical to the previous case and just differs in the fact that yield is called instead of sleep. As above, the start time for RR is same for the processes in the batch as expected. For FCFS the start time of processes differ by the length of 1st CPU burst of previously scheduled process. Since there is very little computation after the last sleep call, for both FCFS and RR all the processes roughly finish at the same time, as a result the batch execution and the average turnaround time of both is comparable. Slight advantage in average turnaround time and batch execution time for FCFS may be explained by the frequent context switch overhead involved in RR. The smaller difference between max and min times for FCFS can be explained by the fact that FCFS takes all the processes to the last IO burst before excuting the last IO burst for any process, since last IO burst takes very small amount of time, the differences in min, max is very low. No such gurantee is given by RR.

In comparison to part (a), notice there is a minuscule advantage in the waiting time in batch 2. This may be explained by the fact that the sleep(1) is called, and this makes the process sleep for a very small duration. Note that every sleep call is accompanied by a corresponding wakeup call after very small duration. In comparison yield directs transitions the process into RUNNABLE state and saves on the time of wakeup call.

3. **batch7.txt**:

FCFS

Batch execution time: 8419
Average turn-around time: 4608
Average waiting time: 3765
Completion time: avg: 5256, max: 9068, min: 1473

RR

Batch execution time: 8367
Average turn-around time: 8344
Average waiting time: 7506
Completion time: avg: 36032, max: 36055, min: 36002

testloop4 consists of a single cpu followed by an io burst. As expected for FCFS, each process runs and completes after which next process is scheduled. Each of process roughly takes the same amount of time. For RR, all

processes start roughly at the same time and end at the same time. This explains the variations in min,max,avg. Since FCFS completes a process before starting a new process, the execution time of only the processes before the execution of that process is accounted in the corresponding waiting time, for RR the execution time of all the processes contributes to the waiting time of every process. Hence as expected the waiting time and turnaround time for FCFS is significantly lower than RR. Alternatively FCFS behaves bit like SJF in this case because all the processes in batch are identical resulting in small waiting and turnaround times for FCFS.

Burst Estimation Error

1. **batch2.txt** CPU bursts: count: 55, avg: 156, max: 188, min: 1
   CPU burst estimates: count: 51, avg: 135, max: 170, min: 1
   CPU burst estimation error: count: 45, avg: 54

2. **batch3.txt** CPU bursts: count: 208, avg: 152, max: 179, min: 1
   CPU burst estimates: count: 201, avg: 149, max: 168, min: 1
   CPU burst estimation error: count: 157, avg: 18

Explanation

Let r be (the average CPU burst estimation error per estimation instance)/(the average CPU burst length).

The average CPU burst length for both the cases is close to each other. The value of r for *batch2.txt* is 0.4 and the value of r for *batch3.txt* is 0.12. In batch3.txt, we run the program *testlooplong.c*, which has a larger number of CPU bursts. We can see that *batch3.txt* runs for 201 burst compared to just 51 for *batch2.txt*. Exponential averaging gets more and more accurate as it gets more data, i.e. the value of r decreases. Since it gets to run for much longer in *batch3.txt*, the accuracy of the predictions gets better. The predictions which are made towards the end will be much more accurate than earlier. The error in such predictions will be low and that will bring down the overall average error.

It is a better idea to compare r for both the cases rather than the absolute error. For two processes with constant but different burst lengths, the r value will be same after the same number of updates but the absolute errror may be different.

Comparison between non-pre-emptive FCFS and non-pre-emptive SJF

1. **batch4.txt**:
   FCFS:
   Batch execution time: 6380
   Average turn-around time: 6379
   Average waiting time: 5740
   Completion time: avg: 10082, max: 10085, min: 10080

SJF:
Batch execution time: 6473
Average turn-around time: 4723
Average waiting time: 4076
Completion time: avg: 5414, max: 7165, min: 3544
CPU bursts: count: 56, avg: 115, max: 182, min: 1
CPU burst estimates: count: 50, avg: 104, max: 169, min: 43
CPU burst estimation error: count: 46, avg: 41

We note that SJF has considerably smaller waiting and turnaround times as compared to FCFS, since the short CPU bursts (corresponding to test-loop3) are completed earlier for the case of SJF and processes are not left hanging for completion like FCFS. This, in turn reduces the turn-around time and waiting time for these processes leading to the reduced average turn-around time and waiting time.

Comparison between pre-emptive RR and pre-emptive Unix scheduler

1. **batch5.txt**:
   RR
   Batch execution time: 8426
   Average turn-around time: 8401
   Average waiting time: 7553
   Completion time: avg: 41811, max: 41836, min: 41779

   UNIX
   Batch execution time: 7715
   Average turn-around time: 5071
   Average waiting time: 4295
   Completion time: avg: 5702, max: 8347, min: 2936

   Round Robin ignores the priorities that were given to each process. It also has very high waiting and turnaround times. However the UNIX scheduler takes the priorities into account, so the processes which are highly prioritized will get scheduled more often and have earlier end times. But the UNIX scheduler also ensures that lower base priority processes also get some CPU time. This means a shorter turn around and waiting time.

   Round robin has a smaller spread in the completion time. However this makes it less fair than the UNIX scheduler because it ignores the base priorities. The UNIX scheduler has a larger spread in completion times because the processes with higher priority get more CPU time and hence finish first.

2. **batch6.txt**:

RR
Start time: 51468, End time: 59931, Total time: 8463
Batch execution time: 8470
Average turn-around time: 8448
Average waiting time: 7601
Completion time: avg: 59912, max: 59931, min: 59870

UNIX
Batch execution time: 7767
Average turn-around time: 5072
Average waiting time: 4295
Completion time: avg: 6340, max: 9036, min: 3576

The explanation remains the same apart from one more observation. The batch execution time for *batch6.txt* is slightly greater than *batch5.txt*. The difference between the two is that of a *yield* and *sleep* call respectively. The reason for the time difference is the same as in Q1.