IMPLEMENTATION OF SCHEDULING ALGORITHMS

---

1. SCHED_NPREEMPT_SJF

In the scheduler() function, we check if the current policy is SCHED_NPREEMPT_SJF and proceed as follows:

We keep a variable estim_shortest to keep track of the running shortest estimated time. To store the process corresponding to the value in estim_shortest, we have the variable job (struct proc *). We keep on checking the prev_estimate field of the struct proc and update the variables estim_shortest and job accordingly.
The field prev_burst_start stores the start time of the last CPU burst of a process, which is used to calculate the actual CPU burst stored finally in the prev_burst field.
The prev_estimate field of a process is our current estimation of the process' next CPU burst. This is updated when the last CPU burst ends (i.e. in the functions yield(), sleep()).

We iterate over all the RUNNABLE processes and do the following:
If it comes across a process that is not a batch process (checked using the batch_proc field added in the struct proc), it is selected for scheduling immediately by switching context between the found process and the FIRST context.
If it comes across a batch process, we check if its CPU burst estimate is less than estim_shortest. We update the variables estim_shortest and job accordingly and go to the next process.

When we are done with iterating over all the RUNNABLE processes, we schedule the process in the variable job. Before switching contexts, we update the prev_burst_start field of the process by the current time so that the actual CPU burst can be calculated in case the user process calls yield() or sleep().

After this process completes execution or gives up the CPU willingly, we call the sched() function from exit() or sleep(), yield() functions respectively, which switches the context back to the scheduler() function. We then repeat the entire procedure for the updated process table..

2. SCHED_PREEMPT_UNIX

In the scheduler() function, we check if the current policy is SCHED_PREEMPT_UNIX and proceed as follows:

We first update the CPU usage (cpu_usage field) and dynamic priority (priority field) of all the processes using the following rule:

$$\text{cpu\_usage} = (\text{cpu\_usage})/2;$$
$$\text{priority} = (\text{base\_priority}) + (\text{cpu\_usage})/2;$$

We keep a variable smallest to keep track of the process with the least priority value. To store the process corresponding to the value in smallest, we have the variable job (struct proc *). We keep on checking the priority field of the struct proc and update the variables smallest and job accordingly.

The fields cpu_usage and priority are updated in the functions yield() and sleep() according to the following rules:

cpu_usage = cpu_usage + (SCHED_PARAM_CPU_USAGE)/2 in sleep()
cpu_usage = cpu_usage + (SCHED_PARAM_CPU_USAGE) in yield()

We iterate over all the RUNNABLE processes and do the following:
If it comes across a process that is not a batch process (checked using the batch_proc field added in the struct proc), it is selected for scheduling immediately by switching context between the found process and the FIRST context.
If it comes across a batch process, we check if its priority value is less than smallest. We update the variables smallest and job accordingly and go to the next process.

When we are done with iterating over all the RUNNABLE processes, we schedule the process in the variable job.

After this process completes execution or gives up the CPU willingly, we call the sched() function from exit() or sleep(), yield() functions respectively, which switches the context back to the scheduler() function. We then repeat the entire procedure for the updated process table.

EVALUATIONS AND STATISTICS

---

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.

   SCHED_NPREEMPT_FCFS

   ```
   Batch execution time: 15209
   Average turn-around time: 15209
   Average waiting time is : 13681
   Completion time : avg: 15843, max: 15851, min : 15836
   ```

   SCHED_PREEMPT_RR

   ```
   Batch execution time: 15367
   Average turn-around time: 15294
   Average waiting time is : 13765
   Completion time : avg: 16001, max: 16042, min : 15904
   ```

   batch1.txt consists of testloop1 files. Hence, the batch can be viewed as a sequence of large CPU bursts interspersed with small I/O bursts (which are sleep(1) calls).

   We observe that the performance of SCHED_NPREEMPT_FCFS and SCHED_PREEMPT_RR algorithms is more or less the same. This is because if all the processes go to sleep, then the CPU remains idle without any CPU running on it. This also explains the much higher execution time in this part(around 15000) as compared to part (b) and part(c) (around 10000). This idle CPU time also increases the turnaround, waiting and completion times of both the schedulers.

   Since the processes are the same and the CPU remains idle in both the schedulers, their performance metrics are roughly equal.

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

SCHED_NPREEMPT_FCFS

```
Batch execution time: 9624
Average turn-around time: 9624
Average waiting time is : 8660
Completion time : avg: 10077, max: 10082, min : 10073
```

SCHED_PREEMPT_RR

```
Batch execution time: 10286
Average turn-around time: 10281
Average waiting time is : 9252
Completion time : avg: 10824, max: 10835, min : 10809
```

batch2.txt consists of testloop2 files. Hence, the batch can be viewed as a sequence of large CPU bursts interspersed with yield() calls.

We observe that the FCFS algorithm performs better in comparison to the RR algorithm. Due to the use of yield() calls, the processes now don't leave the ready queue as compared to the previous part, where the processes left the ready queue when they called sleep(). Hence, in both scheduling algorithms we can always find a process to schedule, i.e. the CPU doesn't remain idle. Now, the RR algorithm, being fair, has more overhead due to the repeated timer interrupts (and the resulting context switches) which results in more execution, turnaround and waiting time as compared to FCFS.

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

SCHED_NPREEMPT_FCFS

```
Batch execution time: 10163
Average turn-around time: 5578
Average waiting time is : 4559
Completion time : avg: 6104, max: 10695, min : 1579
```

SCHED_PREEMPT_RR

```
Batch execution time: 10528
Average turn-around time: 10500
Average waiting time is : 9447
Completion time : avg: 11384, max: 11416, min : 11356
```

Batch7.txt calls the `testloop4.c` code, which has no I/O calls except one at the end.
This would mean that in non-preemptiveFCFS, a process once it starts execution does not stop until it exits(completes execution). This is why we can observe a low turnaround time(since a process gets the CPU continuously once it starts, i.e. very low `endtime-ctime`). The completion times are spread far apart since each process is executed individually.

The preemptive RR is a more fair scheduling algorithm, it switches context through timer interrupts, the processes are given equal quanta of time and finish (more or less) at the same time, this gives a low range between max and min completion times. This also means a much higher average 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.

```
Batch execution time: 9954
Average turn-around time: 9708
Average waiting time is : 8710
Completion time : avg: 10260, max: 10511, min : 9746
CPU bursts: count: 58, avg: 171, max: 225, min: 1
CPU burst estimates: count: 58, avg: 137, max: 204, min: 1
CPU burst estimation error: count: 48, avg: 71
```

BATCH 2

$$AvgBurstError/AvgBurstLength = \frac{71}{171} = 0.415$$

```
Batch execution time: 42405
Average turn-around time: 37537
Average waiting time is : 33294
Completion time : avg: 54663, max: 59536, min : 48089
CPU bursts: count: 209, avg: 202, max: 229, min: 1
CPU burst estimates: count: 209, avg: 192, max: 226, min: 1
CPU burst estimation error: count: 199, avg: 24
```

BATCH 3

$$AvgBurstError/AvgBurstLength = \frac{24}{202} = 0.118$$

Batch 2 calls the *testloop2.c* file, which has some CPU bursts(58) with 'yield' to end each burst.
Batch 3 calls *testlooplong.c* file, which has a much larger number of CPU bursts(209).
The exponential averaging formula:

$s(n+1) = at(n) + (1-a)s(n)$, *where* $t(n)$: *Actual* n^{th} *burst and* $s(n)$: *Estimated* n^{th} *burst*

Here we have taken, $a = \frac{1}{2}$ *and* $s(0) = 0$.

We have to 'guess' the initial estimate (s(0)), which we have taken to be 0.
It can be seen that, in successive iterations of the formula, the 'weight' of the term $s(0)$ decays exponentially, thereby our guess has little effect after a large number of CPU bursts.

Another observation is that the $(n+1)^{th}$ burst gives maximum weight to the actual n^{th} burst and weight of the previous actual burst is exponentially reduced, this would mean that any inaccuracy present would be reduced as the number of iterations increase.
In other words, as the number of iterations (CPU bursts) increase, the estimate would get closer to the real CPU burst.

This is clearly observed in the outputs of Batch2 and Batch3. Batch3 has significantly higher number of CPU bursts, giving a much better CPU burst Estimate(an error of only 24), which means a much lower $BurstError/BurstLength$ value. Batch2 has a comparatively lower number of CPU bursts, giving a relatively worse CPU burst Estimate(an error of 71) thus it has a higher $BurstError/BurstLength$ value.

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.

The statistics obtained for non-preemptive SJF are as follows :

```
Batch execution time: 12195
Average turn-around time: 9049
Average waiting time is : 7807
Completion time : avg: 9922, max: 13079, min : 6054
CPU bursts: count: 61, avg: 199, max: 364, min: 1
CPU burst estimates: count: 62, avg: 158, max: 313, min: 1
CPU burst estimation error: count: 51, avg: 87
```

The statistics obtained for non-preemptive FCFS are as follows :

```
Batch execution time: 11865
Average turn-around time: 11866
Average waiting time is : 10678
Completion time : avg: 13841, max: 13850, min : 13833
```

batch4.txt contains testloop2 and testloop3 files. Hence, the batch can be viewed as a sequence of CPU bursts interspersed with yield() calls.

The first observation is that the Average Turnaround time is very close to the Batch Execution time in non-preemptive FCFS scheduling. This is not the case for non-preemptive SJF. This is because in FCFS scheduling, each process is created at the start of the execution of the scheduling algorithm. The processes are all created at the start of the execution of the FCFS algorithm. Also, the repeated yield() calls ensure that processes exit (complete execution) at almost the same time, since forced yield() calls ensure fairness by repeatedly taking processes to the end of the ready queue. So, all processes are created and end at almost the same time. This explains the almost identical Batch Execution time and Average Turnaround time in FCFS.

In SJF, the processes need not end at roughly the same times, since the scheduler takes the process with lowest CPU burst for execution, ignoring the position in the ready queue. Hence, processes end at very different times, which leads to a substantial difference in Batch Execution time and Average Turnaround time.

Another takeaway is that the variance of Completion time is very high in SJF scheduling as compared to FCFS. This is expected since the use of yield() calls makes the FCFS fair in this case, which leads to processes finishing execution in a small time window. SJF, however, compromises on fairness, since it tries to maximise performance using the estimation of CPU bursts. Hence, processes finish at quite different times, which leads to a large variance in the Completion times in SJF.

Also, we observe that the average waiting time in SJF is much lower than that of FCFS, this is expected since SJF gives the least average waiting time among all scheduling algorithms.

The difference between the Average Turnaround Time and Average Waiting Time is given by :
FCFS : $11866 - 10678 = 1188$ and SJF : $9049 - 7807 = 1242$
The burst time (=Average Turnaround Time - Average Waiting Time) is almost the same for both schedulers.

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.

SCHED_PREEMPT_RR

```
Batch execution time: 13570
Average turn-around time: 13543
Average waiting time is : 12190
Completion time : avg: 14691, max: 14700, min : 14680
```

SCHED_PREEMPT_UNIX

```
Batch execution time: 13838
Average turn-around time: 8771
Average waiting time is : 7382
Completion time : avg: 11479, max: 16553, min : 6659
```

The batch5.txt file has 10 processes with large CPU bursts interspersed with small I/O bursts (just sleep(1) calls).

The average waiting time for SCHED_PREEMPT_UNIX is lower than SCHED_PREEMPT_RR. This is because a process that went to sleep in SCHED_PREEMPT_UNIX will get scheduled sooner than if it went to sleep in SCHED_PREEMPT_RR. This is based on the fact that SCHED_PREEMPT_UNIX considers the cpu_usage of a process which is not increased much when a process goes to sleep.

The average turn-around time is significantly low for SCHED_PREEMPT_UNIX. This is because SCHED_PREEMPT_RR chooses the processes randomly while SCHED_PREEMPT_UNIX chooses the processes based on priority (which depends on the overall cpu_usage).

The average turn-around time and batch execution time for SCHED_PREEMPT_RR are very close. This is mainly because the SCHED_PREEMPT_RR algorithm is fair. It will schedule jobs in an order and keep switching. Thus, almost all the processes will finish close to each other at the end. This can also be justified by observing that the completion times (average, max and min are also very close to each other).

This is not the case with SCHED_PREEMPT_UNIX. The completion times have significant variance. This is because it considers the cpu_usage time i.e the fact that a certain process went to sleep and thus its cpu_usage is not increased much. Thus, the scheduler favours processes that don't use up the complete quantum and these processes get scheduled again quickly. Hence, the minimum completion time is 6659 whereas the maximum is 16553, indicating a significant difference.

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

SCHED_PREEMPT_RR

```
Batch execution time: 13892
Average turn-around time: 13854
Average waiting time is : 12474
Completion time : avg: 14522, max: 14542, min : 14488
```

SCHED_PREEMPT_UNIX

```
Batch execution time: 13945
Average turn-around time: 8861
Average waiting time is : 7466
Completion time : avg: 9833, max: 14923, min : 5182
```

The batch6.txt file has 10 processes with large CPU bursts interspersed with yield() calls. Hence, the only difference from the previous part is the fact that now we have yield() calls instead of sleep().
We know that when yield() is called, the cpu_usage is increased by 200 whereas sleep() increases it by 100 only.

The statistics for SCHED_PREEMPT_RR are almost similar to the previous part. This is expected as SCHED_PREEMPT_RR doesn't care if the CPU was yielded or the process went to sleep. The current process goes to the end of the queue in both the cases.

The differences between SCHED_PREEMPT_RR and SCHED_PREEMPT_UNIX in the previous part are seen here as well and are due to the same reasons as well.

One critical thing to notice is that in SCHED_PREEMPT_UNIX for the current part, the completion time (average, max and min) has decreased as compared to the previous part. This is due to the fact that since the `cpu_usage` now increases by 200, hence the current process is scheduled later (the priority value increases quickly in this case as compared to sleep() calls, hence the priority decreases). There will be less yield() calls as the process will be scheduled a bit later as compared to sleep(). Less yield() calls implies less time wasted in context switches.
