Daniel Gooden
COSC240

Custom Scheduling Algorithm

Assignment 4

Description and Justification

My custom scheduling algorithm is based on an adaptive priority based scoring system that dynamically selects the next process to run, using three key factors:

1. **Wait Time:** Processes that have been waiting longer are prioritised to prevent starvation.
2. **Remaining Time:** Processes that are closer to finishing are prioritised to improve system responsiveness.
3. **Rescheduled Count:** Processes that have been rescheduled too many times are penalised to ensure fairness.

This algorithm operates with two priority queues:

1. **Fast Queue:** For processes needing quick response time, with shorter time quanta
2. **Smart Queue:** For processes that are managed based on their dynamic score

All processes are initially placed in the Fast Queue. Every process placed in the Fast Queue has a time quanta of 2. If it is rescheduled, it is then placed into the Smart Queue. If a process is selected from the Smart Queue, it is then given a time quanta of 4 to run on the processor. Any process that is given any time quanta will be allowed complete it's entire quanta before evaluating the next process to schedule. The scheduler will also prioritise emptying all processes from the Fast Queue before evaluating processes on the Smart Queue.

Scoring Formula

The score for each process is calculated based on the following formula:

$$score = \left(\frac{\text{waiting\_factor}}{\text{running\_factor}}\right) + \left(\text{remaining\_time\_factor} \times \text{RT\_WEIGHT}\right) - \left(\text{reschedule\_penalty} \times \text{RS\_WEIGHT}\right)$$

Where:

- waiting_factor = wait time of the process + 1
- running_factor = total time run + 1
- remaining_time_factor = (processing time +1) / (processing time – processed time + 1)
- reschedule_penalty = reschedule count + 1
- RT_WEIGHT = 1.2 to boost processes that are closer to completion
- RS_WEIGHT = 0.5 to penalise processes that have been rescheduled too many times

Why This Algorithm Works

1. **Balanced Prioritisation:** Processes that have been waiting longer get higher priority, but those that are closer to finishing are also favoured.
2. **Adaptability:** This system adapts to different process characteristics dynamically, based on the parameters allowed in the simulation, so that it prioritises the best process based on current conditions.
3. **Fairness and Efficiency:** By penalising resceduled processes, the algorithm ensures that no process is unfairly delayed, while still maintaining efficient CPU utilisation.

Results

In testing, the custom algorithm has shown a reduction in average turnaround and wait times compared to standard algorithms like FCFS, Round-Robin, Constant Feedback, and Exponential Feedback schedulers. I wrote a very long spec_schedule.txt with many long running and quick short running processes to test the efficiency while also trying to maintain a responsive scheduler that could be suitable for interactive systems. This is the results that I received after running my spec_schedule for all algorithms:

- **First Come, First Served (FCFS):**
  - Average turnaround time: **2099.76**
  - Average wait time: **2058.68**
- **Round Robin (RR):**
  - Average turnaround time: **1858.79**
  - Average wait time: **1817.70**
- **Constant Feedback:**
  - Average turnaround time: **1209.30**
  - Average wait time: **1168.21**
- **Exponential Feedback:**
  - Average turnaround time: **1157.56**
  - Average wait time: **1116.48**
- **Custom:**
  - Average turnaround time: **1110.55**
  - Average wait time: **1069.47**