ECE3140 / CS3420
Embedded Systems

Conventional Scheduling Algorithms

Prof. José F. Martínez
Scheduling Algorithms

- Preemptive or Non-preemptive

- Static or Dynamic
  - Are the scheduling decisions based on parameters that change with time?
  - Fixed-priority vs. dynamic-priority

- Online or Offline
  - Are the decisions made a priori with knowledge of task activations, or are they taken at run time based on the set of active tasks?

- Optimal or Heuristic
  - Can you prove that the algorithm is optimal in terms of a certain criteria or not?
Outline: Conventional Scheduling

Scheduling algorithms for non-real-time systems

- A metric: average response time
- First Come First Served (FCFS)
- Shortest Job First (SJF)
- Priority Scheduling
- Round Robin (RR) Scheduling
Scheduling Algorithm

- Scheduling algorithm: the strategy used to pick a ready task for execution

- Let us first consider non-preemptive scheduling
  - *Preemptive*: The running task can be temporarily suspended to execute another task
  - *Non-preemptive*: The running task cannot be suspended until completion or until it is blocked
Metric: Average Response Time

- Goal: minimize the average time that a customer (job) waits in the line (ready queue)
- Response time: \( R_i = f_i - r_i \)
- Average response time:
  \[
  \frac{1}{n} \sum_{i=1}^{n} R_i = \frac{1}{n} \sum_{i=1}^{n} (f_i - r_i)
  \]
First Come First Served (FCFS)

- One of the most popular classical scheduling policies
Predictability of FCFS

- FCFS is rather unpredictable: response time depends strongly on task arrival times
  → *not suitable for real-time systems*
Shortest Job First (SJF) Policy

- Pick the task with the shortest computation time

  - Optimal
    - SJF minimizes the average response time

  - Non-preemptive
    - Preemptive version is often called Shortest Time-to-Completion First (STCF)
Optimality: Proof Sketch?

Given \( n \) tasks, show that \( \sigma_{SJF} = \tau_1 \tau_2 \ldots \tau_n \) where \( C_1 \leq C_2 \leq \ldots \leq C_n \) minimizes the average response time \( \left( \frac{1}{n} \sum_{i=1}^{n} f_i \right) \). Assume that the arrival time is zero for all tasks \( (r_i = 0) \).

Consider a schedule \( \sigma = \tau_1 \tau_2 \ldots \tau_k \tau_l \tau_1 \tau_2 \ldots \) where the first \( k \) tasks are the same as \( \sigma_{SJF} \) but \( \tau_l_1 > \tau_l_2 \)

Now consider the schedule \( \sigma' = \tau_1 \tau_2 \ldots \tau_k \tau_l \tau_1 \tau_2 \ldots \) that is identical to \( \sigma \) except for switching \( \tau_l_1 \) and \( \tau_l_2 \). If \( \bar{R}(\sigma) \) is the average response time for \( \sigma \), how do \( \bar{R}(\sigma) \) and \( \bar{R}(\sigma') \) compare?
What about Real-Time Constraints?

- SJF is NOT optimal for real-time in the sense of feasibility!
Priority Scheduling

- Each task is assigned a priority
  - Example: $p_i \in [0,255]$ (one byte to store priority)

- Task with the highest priority is selected first

- Tasks with the same priority are scheduled using FCFS
More on Priority Scheduling

- Starvation
  - Low priority tasks may experience very long delays due to preemption by higher priority tasks

- Common approach to handle starvation
  - Aging: priority increases with waiting time

- Priority scheduling can be used to implement other policies
  - If $p_i \propto 1/C_i$: a preemptive version of shortest job first!
  - If $P_i = 1/r_i$: FCFS (assume $r_i > 0$)
Round Robin (RR) Scheduling

- The ready queue is FCFS

- However . . .
  - Each task cannot execute more than Q time units (the quantum)
  - When Q time units have elapsed, the task is put back into the ready queue
Round Robin Scheduling Example

- What are the advantages of RR over SJF/STCF?
Round Robin Scheduling Properties

- If there are $n$ tasks in the system,
  - Each repeating sequence in the schedule is $nQ$ in length
  - In each repeating sequence, a task gets $Q$ units of time
  - Suppose context switch time $\delta$

- Hence,
  \[
  R_i = f_i - r_i \approx n(Q + \delta) \frac{C_i}{Q} = nC_i \left(1 + \frac{\delta}{Q}\right)
  \]

- For small $Q$ and negligible $\delta$:
  - Each task runs as if it were executing on a virtual processor that is $n$ times slower than the real one

- If $Q$ is very large, then RR $\equiv$ FCFS