Scheduling

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Basic Scheduling
Non-preemptive Scheduling

- The running task voluntarily yields the CPU
- Force everybody to cooperate

Thread ping ()
{
    while (1) {
        printf ("ping\n");
        yield();
    }
}

Thread pong ()
{
    while (1) {
        printf ("pong\n");
        yield();
    }
}
Preemptive Scheduling

▪ The scheduler can interrupt a task and force a context switch
▪ Need to regain control of processor asynchronously \(\rightarrow\) periodic timer interrupt
▪ At each timer interrupt, the scheduler gains control and context switches as appropriate
▪ Timer tick vs. quantum (or timeslice)
Comparison

Preemptive:
Always runs the highest available task.

Cooperative (Non-preemptive):
Context switches only occur if a task blocks, or explicitly relinquishes CPU control.
Starvation

- A situation where a task is prevented from making progress because another task has the resource it requires
- A poor scheduling policy can cause starvation
- Synchronization can also cause starvation
Priority Scheduling (1)

- Choose task with highest priority to run next
- Round-robin or FIFO within the same priority
- Can be either preemptive or non-preemptive
- Priority is dynamically adjusted
  - Static priority vs. dynamic priority
Priority Scheduling (2)

- Starvation problem
  - If there is an endless supply of high priority tasks, no low priority task will ever run

- Aging
  - Increase priority as a function of wait time
  - Decrease priority as a function of CPU time
UNIX Schedulers

- Priority-based
  - Static priority + dynamic priority
- Preemptive
- Time-shared
- Aging
- Priority boost for I/O-bound tasks

- Priority vs. quantum?
Real-Time Scheduling
Real-Time Systems

- Perform a computation to conform to external timing constraints
- Deadline frequency: periodic vs. aperiodic
- Deadline type:
  - Hard: failure to meet deadline causes system failure
  - Soft: failure to meet deadline causes degraded response (best effort, statistical guarantees)
Periodic vs. Aperiodic Tasks

- Periodic task: executes on (almost) every period
- Aperiodic task: executes on demand
- Analyzing aperiodic task sets is harder
  - Must consider worst-case combinations of task activations
Real-Time Workload

- **Job (unit of work)**
  - A computation, a file read, a message transmission, etc.

- **Attributes**
  - Resources required to make progress
  - Timing parameters

![Graph showing real-time workload components: Released, Execution time, Relative deadline, Absolute deadline]
Real-Time Task

- Task: a sequence of similar jobs
- Periodic task \((p, e)\)
  - Its jobs repeat regularly
  - Period \(p = \text{inter-release time (} 0 < p \))
  - Execution time \(e = \text{maximum execution time (} 0 < e < p \))
  - Utilization \(U = e/p\)
Real-Time Scheduling

- **Schedulability**
  - Property indicating whether a real-time system (a set of real-time tasks) can meet their deadlines

- **Real-time scheduling**
  - Determines the order of real-time task executions
  - Static-priority scheduling: RM
  - Dynamic-priority scheduling: EDF
Simple Feasibility Test

- **Assume:**
  - No resource conflicts
  - Constant process execution times

- **Require:**
  - \( T \geq \sum_i T_i \)
  - Can’t use more than 100% of the CPU
- **Rate Monotonic**
  - Optimal static-priority scheduling
  - Assigns priority according to period
  - A task with a shorter period has a higher priority

![Diagram](image)
RM

- Rate Monotonic
  - Executes a job with the shortest period
RM

- Rate Monotonic
  - Executes a job with the shortest period

![Diagram showing execution of jobs with rate monotonic scheduling]

T₁(4,1)
T₂(5,2)
T₃(7,2)

Deadline Miss!
RM

- Utilization bound
  - Real-time system is schedulable under RM if
    \[ \sum U_i \leq n(2^{1/n} - 1) \]
  - Example: \( T_1(4,1), T_2(5,1), T_3(10,1) \)
    \[ \sum U_i = \frac{1}{4} + \frac{1}{5} + \frac{1}{10} = 0.55 \]
    \[ 3(2^{1/3} - 1) \approx 0.78 \]
    Thus, \( \{T_1, T_2, T_3\} \) is schedulable under RM.
RM

- Utilization bound (cont’d)

\[ \sum U_i \leq n(2^{1/n} - 1) \]

RM Utilization Bounds

\[ \sum U_i \leq n(2^{1/n} - 1) \]

~ 69%
RM

- As the number of tasks approaches infinity, the maximum utilization approaches 69%
- RM cannot use 100% of CPU, even with zero context switch overhead
- Must keep idle cycles available to handle worst-case scenario
- However, RM guarantees all tasks will always meet their deadlines
EDF

- Earliest Deadline First
  - Optimal dynamic priority scheduling
  - Task with a shorter deadline has higher priority
  - Executes a job with the earliest deadline

\[ T_1(4,1) \]
\[ T_2(5,2) \]
\[ T_3(7,2) \]
EDF

- Earliest Deadline First
  - Executes a job with the earliest deadline
**EDF**

- **Earliest Deadline First**
  - Executes a job with the earliest deadline
EDF

- Earliest Deadline First
  - Executes a job with the earliest deadline
EDF

- Optimal scheduling algorithm
  - If there is a schedule for a set of real-time tasks, EDF can schedule it.
EDF

- Utilization bound
  - Real-time system is schedulable under EDF if and only if
    \[ \sum U_i \leq 1 \]

RM vs. EDF (1)

- **Rate Monotonic**
  - Simpler implementation, even in systems without explicit support for timing constraints
  - Predictability for the highest priority tasks

- **EDF**
  - Full processor utilization
  - Implementation complexity and runtime overhead due to dynamic priority management
  - Misbehavior during overload conditions
RM vs. EDF (2)

Assumptions

- All tasks are periodic and fully preemptible
- All tasks are released at the beginning of period and have a deadline equal to their period
- All tasks are independent
- All tasks have a fixed computation time
- No task may voluntarily suspend itself
- All overheads are assumed to be 0
- There is just one processor