Scheduling

Arijit Mondal
Dept. of Computer Science & Engineering
Indian Institute of Technology Patna
arijit@iitp.ac.in

Introduction

- In general there will be more number of tasks than the number of processors
  - Need a scheduler to run the tasks effectively
- Tasks may have precedence constraints
- Tasks may have hard timing constraints (Real time systems)
  - Typically referred as deadline
- Scheduling techniques are applicable in different domains

Scheduler

- Decides what task to execute next when faced with a choice in the execution of concurrent programs
  - Multiprocessor scheduler needs to decide which processor as well (Processor assignment)
- Scheduling decision
  - Assignments - which processor should execute
  - Ordering - in what order each processor should execute
  - Timing - the time at which each task executes
- Above parameters can be decided in design time (static scheduler) or at run time (dynamic scheduler)

Scheduler (contd.)

- Static scheduler - decides the parameter in design time
  - Does not require semaphore or lock in general
  - Predicting time for modern processor is extremely difficult (out-of-order execution)

  Dynamic scheduler
  - Performs all decision at run time
- Online vs Offline
- Preemption vs Non-preemption
  - Blocked - waiting for mutual exclusion lock

Task model

- Arrival of tasks - scheduler needs to know the task before scheduling
- Periodic, aperiodic, sporadic
- Execution of tasks - preemptive vs non-preemptive
- Precedence constraints
- Pre-condition
- Release time, Start time, Finish time, Execution time, Deadline
- Hard real time scheduling, Soft real time scheduling
- Priority - fixed, dynamic

Execution of task

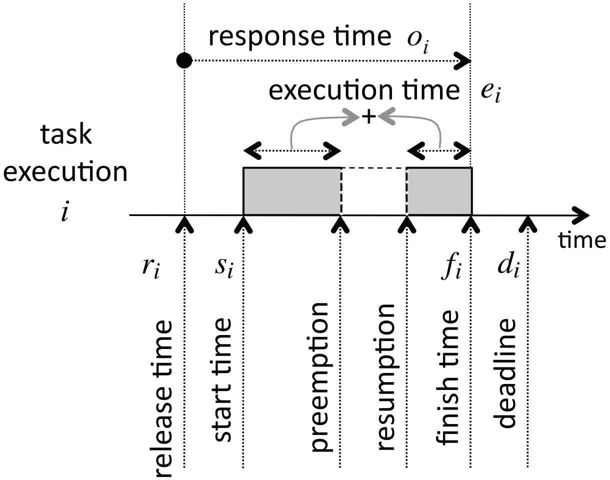


image source: Introduction to Embedded Systems book

Comparing scheduler

- Goal of any scheduler is to find any feasible schedule that is $f_i \leq d_i$ for all tasks
- A scheduler that yields feasible schedule for a task set when there is a feasible schedule is said to be optimal with respect to feasibility
- **Utilization** - Percentage of time that the processor spends executing tasks
  - Most popular metric
- **Maximum lateness** - It is defined as $L_{max} = \max(f_i - d_i)$
  - For feasible schedule it will be 0 or negative
- **Total completion time / Makespan** - It is defined as $M = \max_T f_i - \min_T r_i$

Implementation of scheduler

- Scheduler can be part of compiler or code generation
  - Decision made at design time
- Scheduler can be part of operating system or kernel
  - Decision made at run time
- It can be both as well
- For non-preemptive scheduling procedure is invoked when a task completes
- For preemptive scheduling procedure is invoked when several things occur
  - A timer interrupt occurs
  - An I/O interrupt occurs
  - AN OS service is invoked
  - Task attempts to get mutex
  - A task tests semaphore

Rate monotonic

- n tasks execute periodically
- Let p_i be the period for ith task and r_i be the release time
- Deadline for jth execution $r_i + j \times p_i$
- Fixed priority scheduling
- Scheduling strategy: higher priority to a task that has smaller period
  - Optimal with respect to feasibility for fixed priority

Rate monotonic: Example

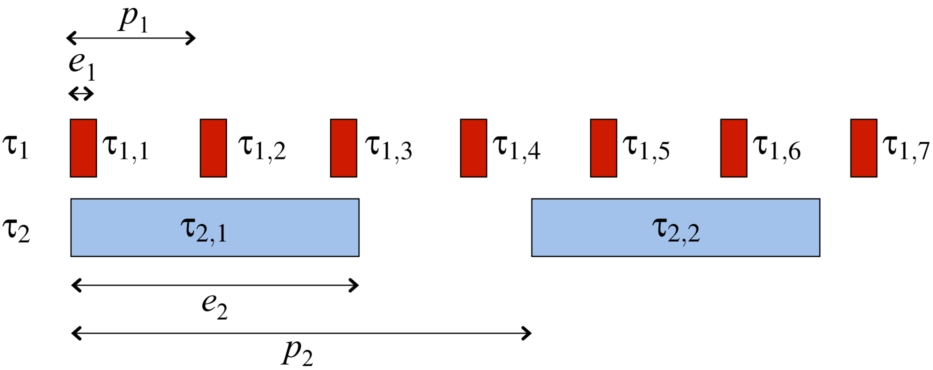


image source: Introduction to Embedded Systems book

Rate monotonic: Example

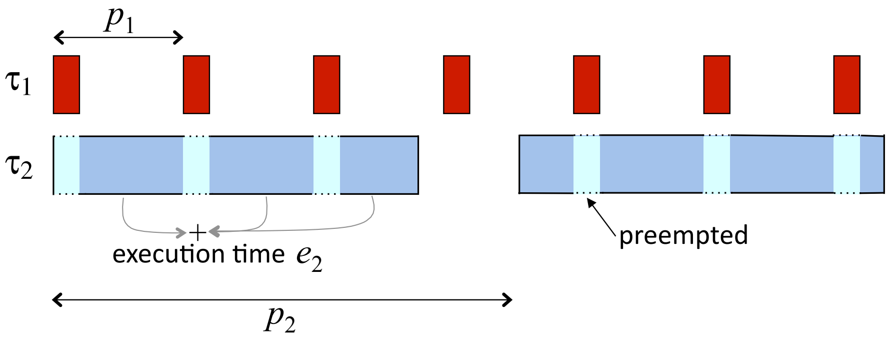


image source: Introduction to Embedded Systems book

Rate monotonic: Response time

- Response time of the lower priority task is worst when its starting time matches that of higher priority tasks
- Worst case scenario occurs when all start at the same time

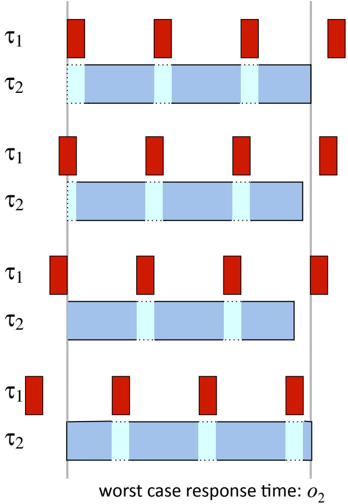


image source: Introduction to Embedded Systems book

Rate monotonic: Optimality

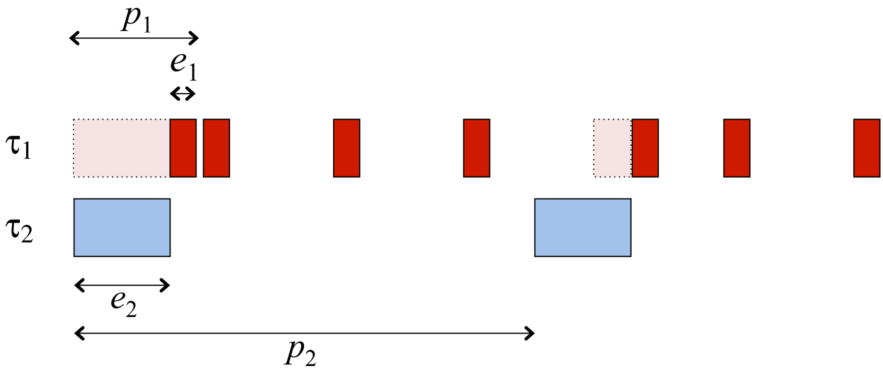


image source: Introduction to Embedded Systems book

Rate monotonic: Optimality

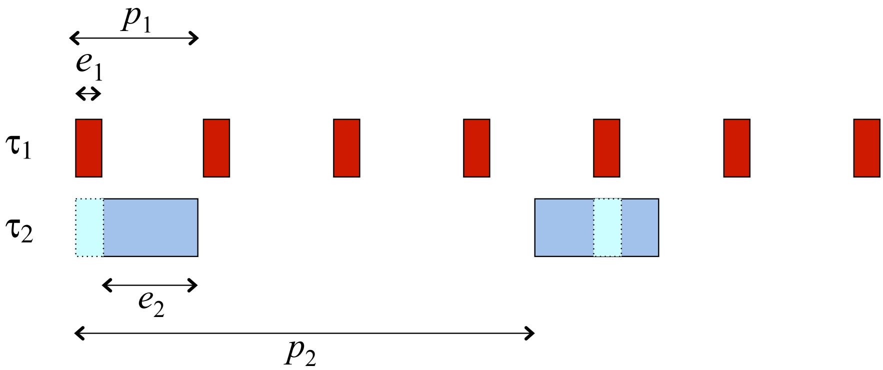


image source: Introduction to Embedded Systems book

Rate monotonic: Utilization

- May not achieve 100% utilization
- Utilization is defined as $\mu = \sum_{i=1}^{n} \frac{e_i}{p_i}$
- Utilization bound $\mu \leq n\left(2^{\frac{1}{n}} - 1\right)$
  - For $n = 2$ maximum utilization can be achieved as 82.8%
  - When n is very large, maximum utilization can be achieved as 69.3%

Earliest Deadline Due

- Given a set of non-preemptive non-repeating tasks with deadlines and no precedence constraints
- Executes tasks in the same order as their deadline
- EDD is optimal in a sense that minimizes maximum lateness
- Does not support arrival of tasks

Earliest Deadline First

- Given a set of n independent tasks $T = \{\tau_1, \tau_2, \ldots, \tau_n\}$ associated with deadlines $d_1, d_2, \ldots, d_n$ and arbitrary arrival time
- Scheduling strategy: at any instant executes the task with earliest deadline among all arrival tasks
- EDF is optimal in a sense that minimizes maximum lateness
- Dynamic priority scheduling algorithm
- If a task repeatedly executed, it may be assigned different priorities
- Complex to implement
- More expensive to implement than RM but performance is superior

RM vs EDF

- RM is optimal with fixed priority
- EDF is optimal with dynamic priority
  - Also minimizes maximum lateness
  - Results in less preemption, less overhead
- Any EDF schedule with less than 100% utilization can tolerate increase in execution time and/or reduction in period and still feasible

EDF with precedence

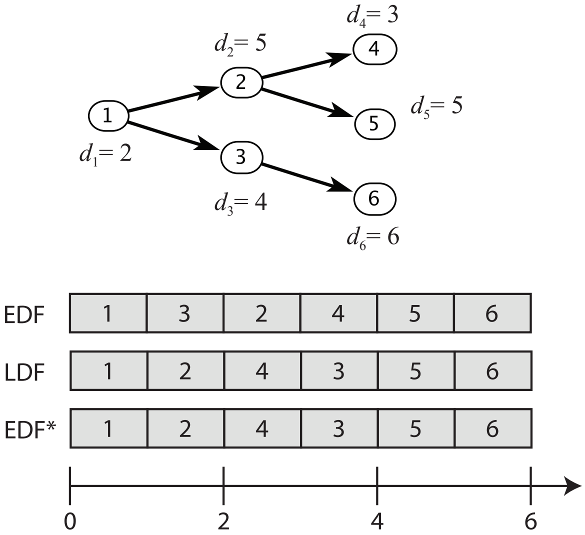


image source: Introduction to Embedded Systems book

LDF, EDF*

- Latest Dedline First (LDF)
  - Construct the scheduling backward
  - The last task is chosen first and which has latest deadline
  - Does not support arrival of tasks
- EDF*
  - Support arrival of tasks and minimizes maximum lateness
  - For a task i, let $D(i)$ be the set of task execution that immediately depend on i in precedence graph
  - Modified deadline $d'_i = \min\left(d_i, \min_{j \in D(i)} (d'_j - e_i)\right)$

Scheduling and mutual exclusion

- Priority inversion
  - Priority is based preemptive scheduler enables high priority task
  - Using mutual exclusion, a task may become blocked

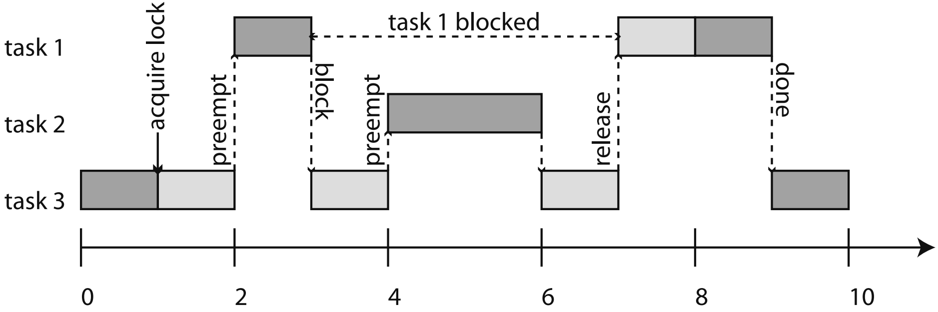


image source: Introduction to Embedded Systems book

Priority inheritance protocol

- When a task blocks attempting to acquire a lock, then the task that holds the lock inherits the priority of the blocked task

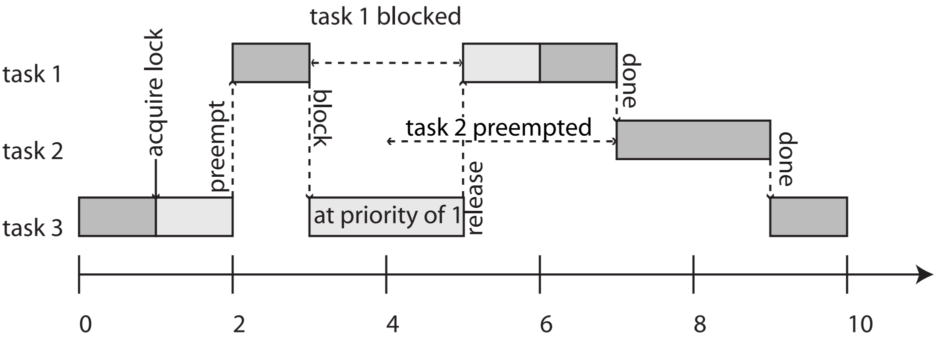


image source: Introduction to Embedded Systems book

Priority ceiling protocol

- Every lock is assigned a priority ceiling equal to the priority of the highest priority task that can lock it

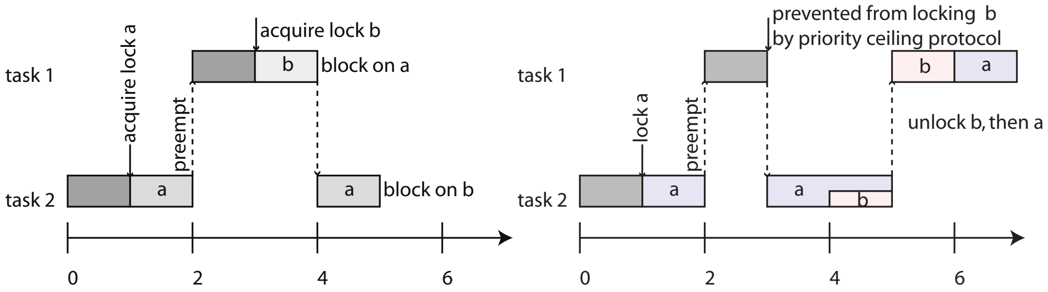


image source: Introduction to Embedded Systems book

Multiprocessor scheduling

- Scheduling on a single processor is hard, scheduling on multiprocessor is harder
- Scheduling of fixed finite set of tasks with precedence on a finite number of processors with goal to minimize makespan
  - NP-Hard problem
- Hu level scheduling algorithm
  - Assigns priority to each task based on the level
  - Greatest sum of execution times of tasks on a path in the precedence graph from τ to another task with no dependents

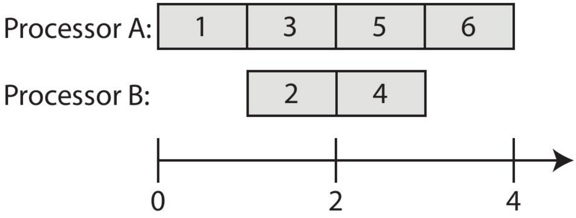


image source: Introduction to Embedded Systems book

Scheduling anomalies

- Multiprocessor scheduling are non-monotone
  - Improvement in local performance can degrade over all performance
- Richard's anomalies
  - *If a task set with fixed priorities, execution times, and precedence constraints is scheduled on a fixed number of processors in accordance with the priorities, then increasing the number of processors, reducing execution times, or weakening precedence constraints can increase the schedule length.*

Multiprocessor scheduling

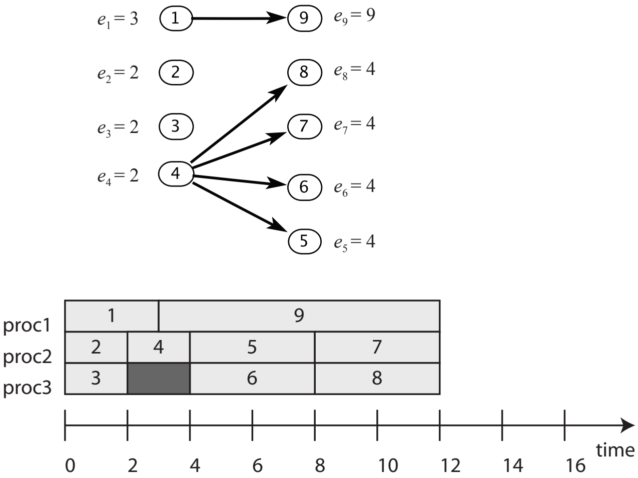


image source: Introduction to Embedded Systems book

Richard's anomalies: Reducing execution time

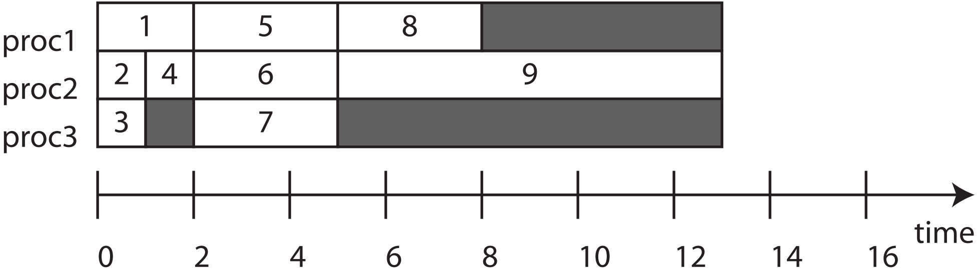


image source: Introduction to Embedded Systems book

Richard's anomalies: More processor

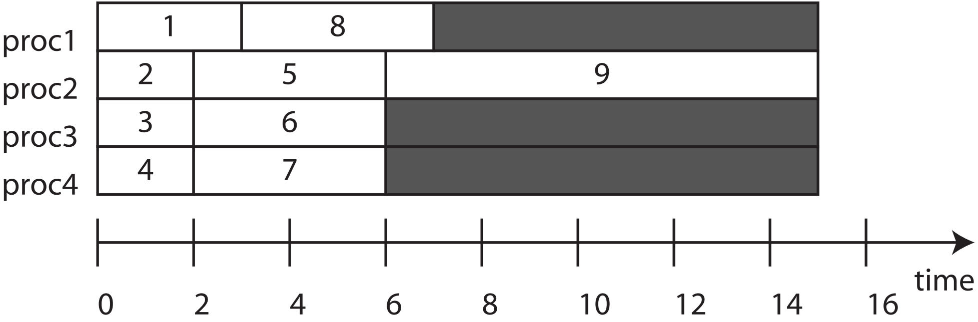


image source: Introduction to Embedded Systems book

Richard's anomalies:: Removing precedence

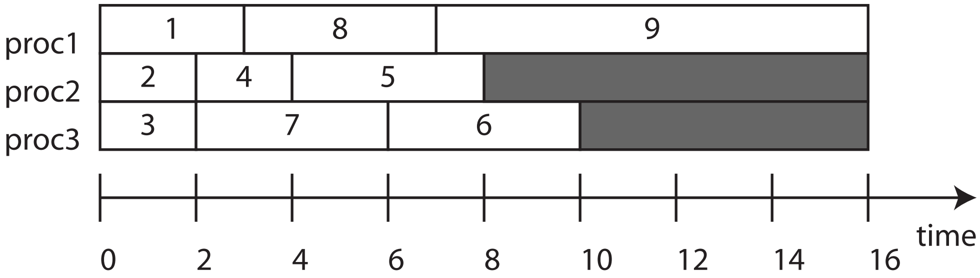
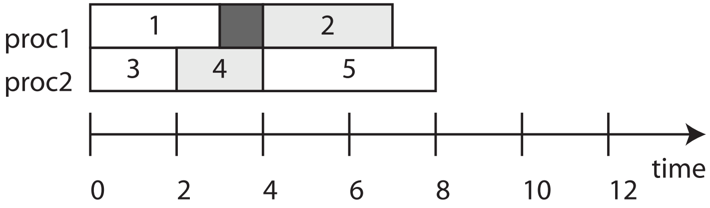


image source: Introduction to Embedded Systems book

Anomaly due to mutex



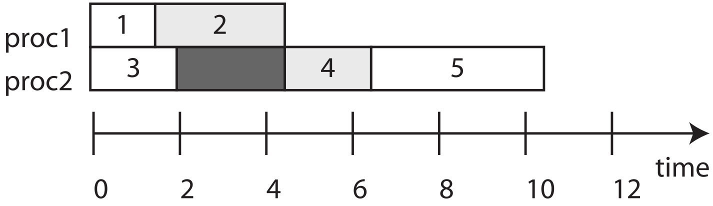


image source: Introduction to Embedded Systems book