Advanced Operating Systems - Real Time Systems

Mridul Sankar Barik

Jadavpur University

2024

Real-Time Systems

- The operating system, and in particular the scheduler, is the most important component
- Correctness of the system depends not only on the logical result of the computation but also on the time at which the results are produced
- Tasks or processes attempt to control or react to events that take place in the outside world
- These events occur in "real time" and tasks must be able to keep up with them

Soft Real-Time Systems

- In a soft real-time system, it is considered undesirable, but not catastrophic, if deadlines are occasionally missed
- Also known as "best effort" systems
- Most modern operating systems can serve as the base for a soft real time systems
- Examples:
  - Multimedia transmission and reception
  - Networking, Telecom (cellular) networks
  - Web sites and services
  - Computer games

Hard Real-Time Systems

- A hard real-time system has time-critical deadlines that must be met; otherwise a catastrophic system failure can occur
- Requires formal verification/guarantees of being able to always meet its hard deadlines (except for fatal errors)
- Examples:
  - Vehicle Subsystems Control
  - Nuclear Power Plant Control
  - Process Control in Industrial Plants
  - Robotics
  - Air Traffic Control
  - Military Command and Control Systems

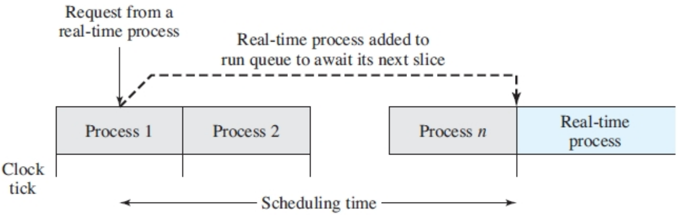
Real-Time Tasks

- Hard real-time task
  - One that must meet its deadline
  - Otherwise it will cause unacceptable damage or a fatal error to the system
- Soft real-time task
  - Has an associated deadline that is desirable but not mandatory
  - It still makes sense to schedule and complete the task even if it has passed its deadline

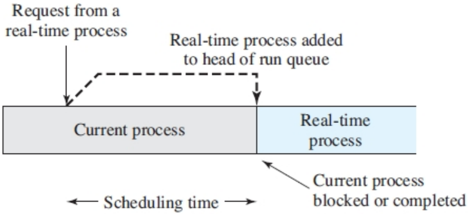
Periodic and Aperiodic Tasks

- Periodic tasks
  - Requirement may be stated as:
    - Once per period T
    - Exactly T units apart
- Aperiodic tasks
  - Has a deadline by which it must finish or start
  - May have a constraint on both start and finish time

Scheduling of Real-Time Process I

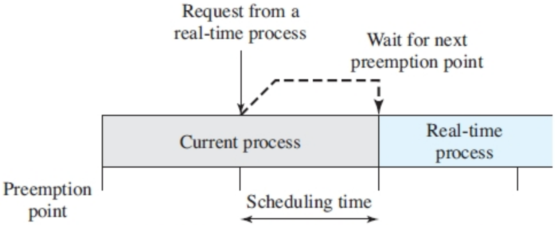


(a) Round-robin preemptive scheduler

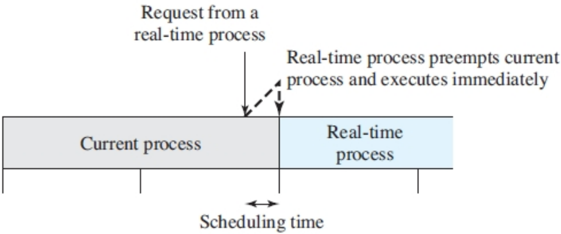


(b) Priority-driven nonpreemptive scheduler

Scheduling of Real-Time Process II



(c) Priority-driven preemptive scheduler on preemption points



(d) Immediate preemptive scheduler

Real-Time Scheduling

- Scheduling approaches depend on:
  - Whether a system performs schedulability analysis
  - If it does, whether it is done statically or dynamically
  - Whether the result of the analysis itself produces a scheduler plan according to which tasks are dispatched at run time

Classes of Real-Time Scheduling Algorithms

- Static table-driven approaches
  - Performs a static analysis of feasible schedules of dispatching
  - Result is a schedule that determines, at run time, when a task must begin execution
- Static priority-driven preemptive approaches
  - A static analysis is performed but no schedule is drawn up
  - Analysis is used to assign priorities to tasks so that a traditional priority-driven preemptive scheduler can be used
- Dynamic planning-based approaches
  - Feasibility is determined at run time
  - An arriving task is accepted for execution only if it is feasible to meet its time constraints
- Dynamic best effort approaches
  - No feasibility analysis is performed
  - System tries to meet all deadlines and aborts any started process whose deadline is missed

Deadline Scheduling

- Real-time operating systems are designed with the objective of starting real-time tasks as rapidly as possible and emphasize rapid interrupt handling and task dispatching
- Real-time applications are generally not concerned with sheer speed but rather with completing (or starting) tasks at the most valuable times
- Priorities provide a crude tool and do not capture the requirement of completion (or initiation) at the most valuable time

Information Used for Deadline Scheduling I

- Ready time
  - Time at which task becomes ready for execution
  - For a repetitive or periodic task, this is actually a sequence of times that is known in advance
  - For an aperiodic task, this time may be known in advance, or the operating system may only be aware when the task is actually ready
- Starting deadline
  - Time by which a task must begin
- Completion deadline
  - Time by which task must be completed
  - Typical realtime application will either have starting deadlines or completion deadlines, but not both
- Processing time
  - Time required to execute the task to completion
  - In some cases, this is supplied
  - In others, the OS measures an exponential average

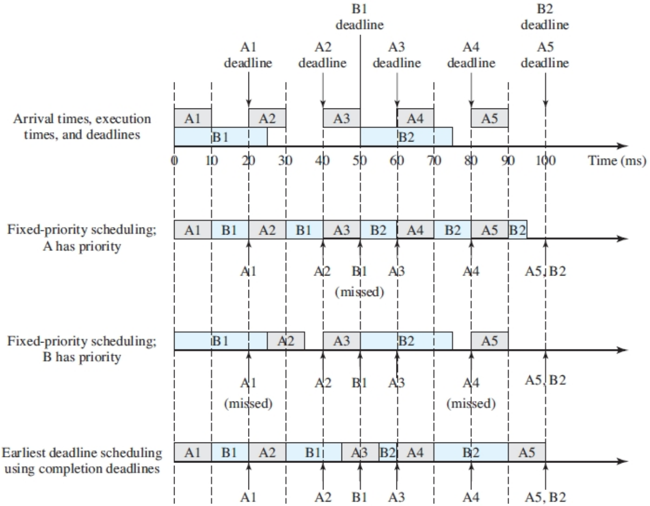
Information Used for Deadline Scheduling II

- Resource requirements
  - Set of resources (other than the processor) required by the task while it is executing
- Priority
  - Measures relative importance of the task
  - Hard real-time tasks may have an "absolute" priority, with the system failing if a deadline is missed
  - If the system is to continue to run no matter what, then both hard and soft realtime tasks may be assigned relative priorities as a guide to the scheduler
- Subtask structure
  - A task may be decomposed into a mandatory subtask and an optional subtask
  - Only the mandatory subtask possesses a hard deadline

Execution Profile of Two Periodic Tasks

Process	Arrival Time	Execution Time	Ending Deadline
A(1)	0	10	20
A(2)	20	10	40
A(3)	40	10	60
A(4)	60	10	80
A(5)	80	10	100
• • •	• • •	• • •	• • •
B(1)	0	25	50
B(2)	50	25	100
• • •	• • •	• • •	• • •

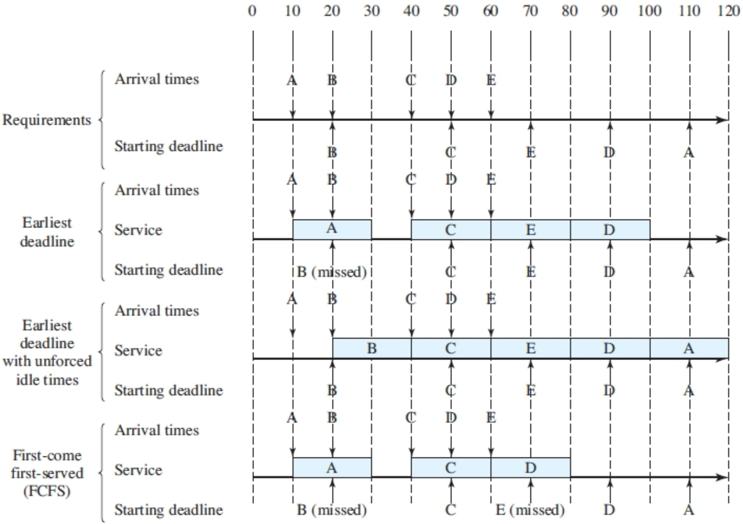
Scheduling of Periodic Real-Time Tasks with Completion Deadlines



Execution Profile of Five Aperiodic Tasks

Process	Arrival Time	Execution Time	Starting Deadline
A	10	20	110
B	20	20	20
C	40	20	50
D	50	20	90
E	60	20	70

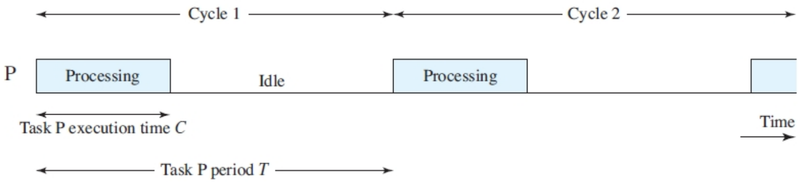
Scheduling of Aperiodic Real-Time Tasks with Starting Deadlines



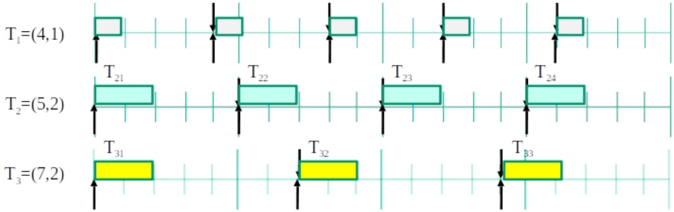
Rate Monotonic Scheduling

- Assigns priorities to tasks on the basis of their periods
  - Highest priority to shortest period

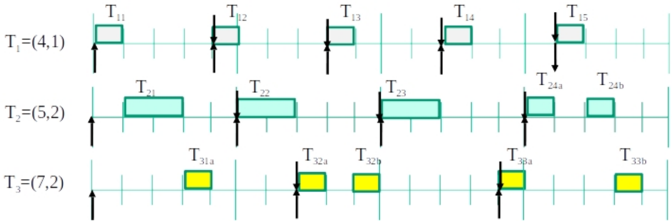
Periodic Task Timing Diagram



Rate Monotonic Scheduling



Periodic Tasks
T_i=(P_i, C_i)
P_i = period
C_i= processing time



RMS
Scheduling

Priority Inversion

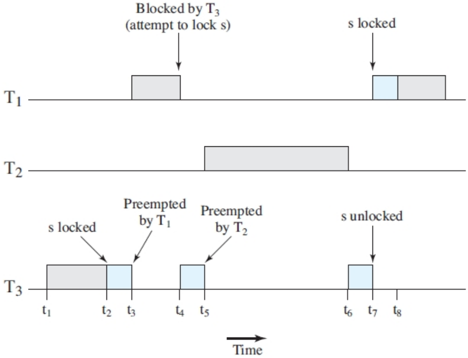
- Can occur in any priority-based preemptive scheduling scheme
- Particularly relevant in the context of real-time scheduling
- Best-known instance involved the Mars Pathfinder mission
- Occurs when circumstances within the system force a higher priority task to wait for a lower priority task
- Unbounded Priority Inversion
    - The duration of a priority inversion depends not only on the time required to handle a shared resource, but also on the unpredictable actions of other unrelated tasks

Mars Pathfinder Mission

- Three tasks with decreasing order of priority
  - T_1: Periodically checks the health of the spacecraft systems and software
  - T_2: Processes image data
  - T_3: Performs an occasional test on equipment status
- T_1 and T_3 share a common data structure protected by a binary semaphore.

Unbounded Priority Inversion

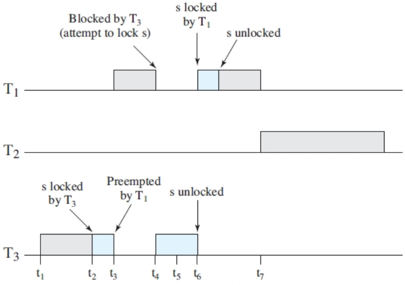
- t_1: T_3 begins executing.
- t_2: T_3 locks semaphore s and enters its critical section.
- t_3: T_1, which has a higher priority than T_3, preempts T_3 and begins executing.
- t_4: T_1 attempts to enter its critical section but is blocked because the semaphore is locked by T_3; T_3 resumes execution in its critical section.
- t_5: T_2, which has a higher priority than T_3, preempts T_3 and begins executing.
- t_6: T_2 is suspended for some reason unrelated to T_1 and T_3;T_3 resumes.
- t_7: T_3 leaves its critical section and unlocks the semaphore. T_1 preempts T_3, locks the semaphore and enters its critical section.



Priority Inheritance

- A lower-priority task inherits the priority of any higher-priority task pending on a resource they share

- t_1: T_3 begins executing.
- t_2: T_3 locks semaphore s and enters its critical section.
- t_3: T_1, which has a higher priority than T_3, preempts T_3 and begins executing.
- t_4: T_1 attempts to enter its critical section but is blocked because the semaphore is locked by T_3. T_3 is immediately and temporarily assigned the same priority as T_1. T_3 resumes execution in its critical section.
- t_5: T_2 is ready to execute but, because T_3 now has a higher priority, T_2 is unable to preempt T_3.
- t_6: T_3 leaves its critical section and unlocks the semaphore: its priority level is downgraded to its previous default level. T_1 preempts T_3, locks the semaphore, and enters its critical section.
- t_7: T_1 is suspended for some reason unrelated to T_2, and T_2 begins executing.



(b) Use of priority inheritance

Priority Ceiling

- A priority is associated with each resource
- The priority assigned to a resource is one level higher than the priority of its highest-priority user
- The scheduler dynamically assigns this priority to any task that accesses the resource
- Once the task finishes with the resource, its priority returns to normal