Module-5-Real Time Systems and scheduling-Techniques-RMS & EDF

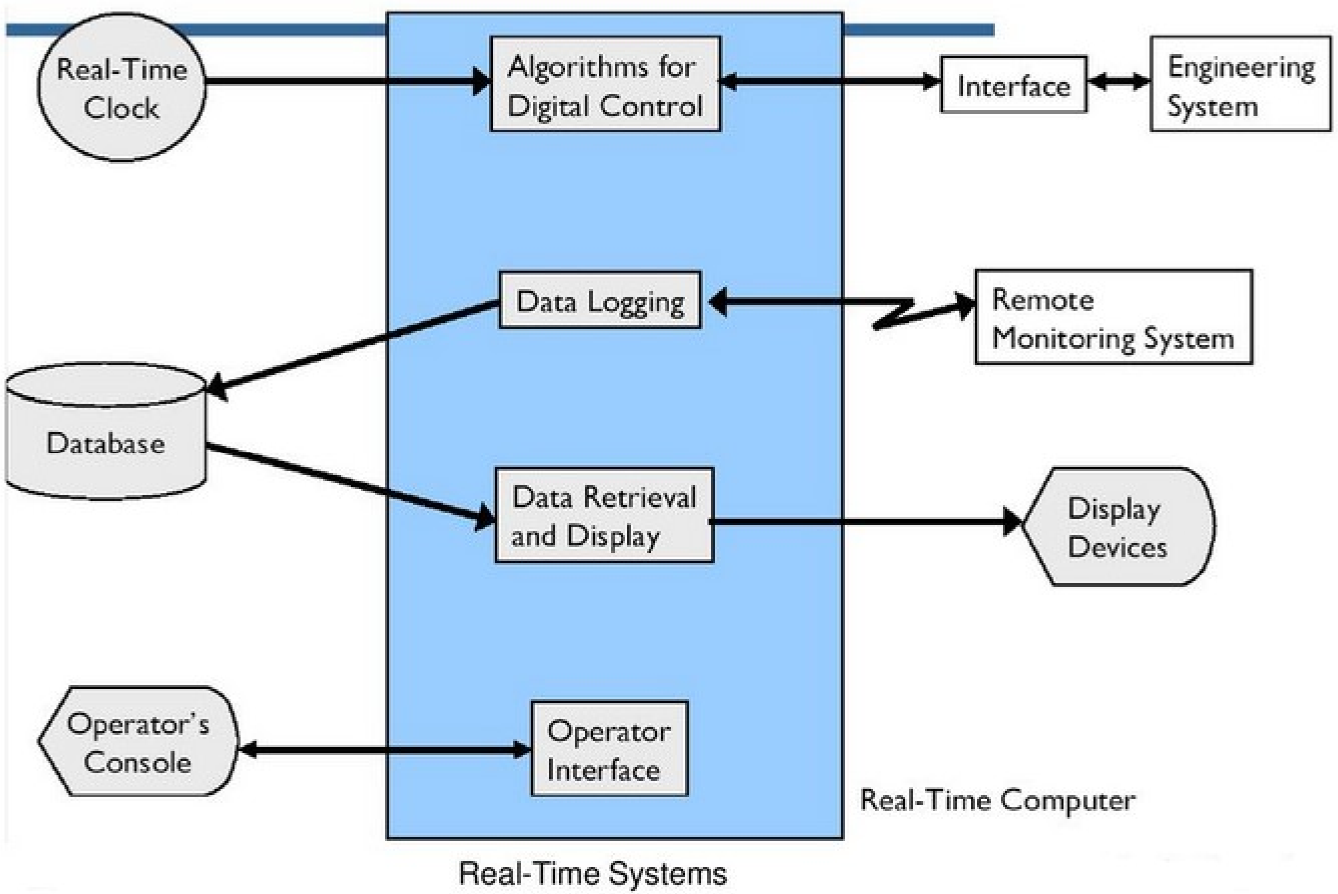
Characteristics of RTS

- Extreme reliability and safety
  - Embedded systems typically control the environment in which they operate
  - Failure to control can result in loss of life, damage to environment or economic loss
- Guaranteed response times
  - We need to be able to <u>predict with confidence</u> the <u>worst case response times</u> for systems
  - Efficiency is important but predictability is essential
    - In RTS, performance guarantees are:
      - Task- and/or class centric
      - Often ensured a priori
    - In conventional systems, performance is:
      - System oriented and often throughput oriented
      - Post-processing (... wait and see ...)

Real-Time Systems

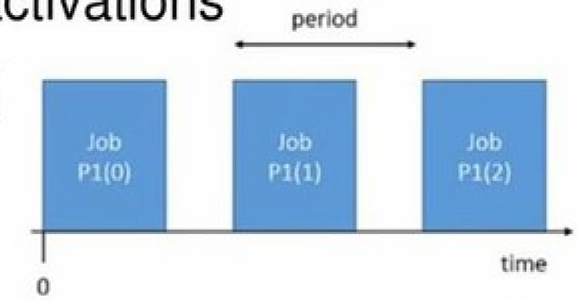
- Perform a computation to conform to external timing constraints
- Deadline frequency:
  - Periodic
  - Aperiodic
- Deadline type:
  - Hard: failure to meet deadline causes system failure
    - Car Airbag system, car brakes
  - Soft: failure to meet deadline causes degraded response
    - Room temperature control, car multimedia system
  - Firm: late response is useless; Infrequent deadline misses are tolerable, but may degrade the system's quality of service
    - A digital cable set-top box frame decoder

Components of RTS



Types of Process Timing Requirements

- Release time: time at which process becomes ready to execute
- Deadline: time at which process must finish execution
- Periodic process: a process executes every period
  - e.g. Firing the spark plugs
- Aperiodic process: executes on demand
  - Processing a button press
- Period : interval between process activations
- Initiation Interval or Rate = 1/period

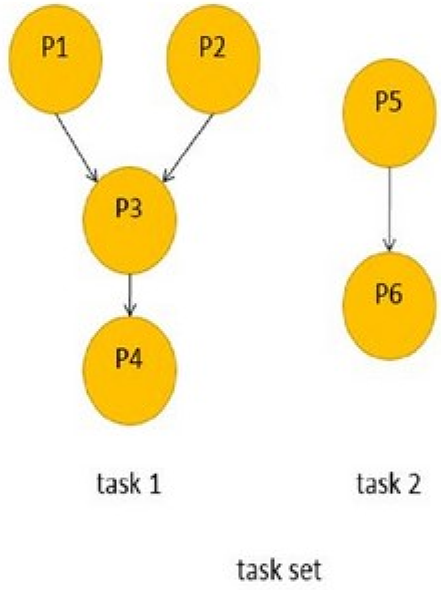


Types of Process Timing Requirements

- Jitter: Allowable variation in task completion time
  - Example: multimedia synchronization
- What happens when a process misses a deadline?
  - Can be catastrophic such as in an automotive control system
  - a missed deadline in a telephone system may cause a temporary silence on the line
- Example: Space Shuttle software error
  - Space Shuttle's first launch was delayed by a software timing error
  - Change to one routine added delay that threw off start time calculation

Task Graphs

- ❖ Tasks may have data dependencies---must execute in certain order
- ❖ Task graph shows data/control dependencies between processes
- ❖ Task: connected set of processes
- ❖ Task set: One or more tasks
- ❖ Task graph assumes that all processes at the same rate, tasks do not communicate
- ❖ In reality, some amount of inter-task communication is necessary



Process Execution Characteristics

- Process execution time T_i
  - Execution time in absence of preemption
  - Possible time units: seconds, clock cycles
  - Worst-case, best-case execution time may be useful in some cases
- Sources of variation:
  - Data dependencies
  - Memory system
  - CPU pipeline

Processes and Operating Systems

- Processes and operating system are abstractions
  - allow us to build complex applications on microprocessors
  - provide much greater flexibility to satisfy timing requirements
- Let us switch the state of the processor between multiple tasks
- Process defines the state of an executing program
- A process is a unique execution of a program
  - Several copies of a program may run simultaneously or at different times
- A process has its own state: registers and memory
  - Threads share the same address space

Processes and Operating Systems

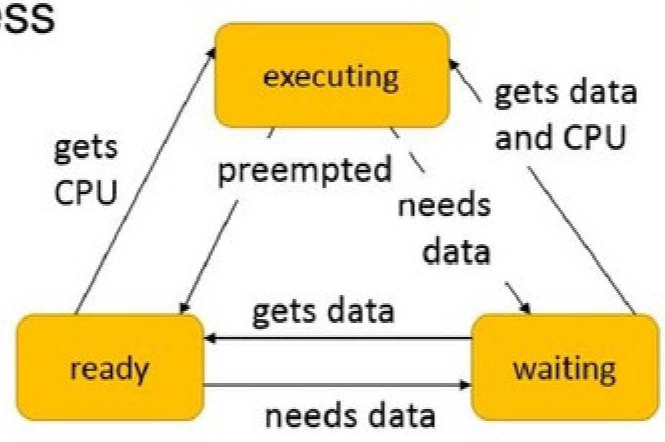
- The Operating System (OS) manages processes
- OS provides the mechanism for switching execution between processes
- Real-Time Operating System (RTOS) is OS that provides facilities for satisfying real-time requirements
  - Allocates resources based on real-time requirements
  - General-purpose OSs use other criteria, e.g. fairness
- RTOS helps build more complex systems using several programs that run concurrently

Real-Time Operating Systems

- Solves the main problems of a cooperative multitasking system
- Executes processes based on timing requirements provided by system designer
- Based on two basic concepts:
  - Preemption: the ability to interrupt a process to switch to another
  - Context switching: switching execution and CPU state between processes

State of a Process

- A process can be in one of three states:
  - executing on the CPU
  - ready to run
  - waiting for I/O, another process, timer, next period
- The operating system selects the next executing process
- At most one executing process



Context Switching

- Context: The set of registers that define a process
- Context Switching: Switching the registers from one process to another
  - Timer interrupt: transfer control from a process to kernel
  - Kernel saves current process context
  - Kernel selects next process (scheduling)
  - Kernel restores next process context

The Scheduling Problem

- Choosing the order of running processes is known as scheduling
- Workstations try to avoid starving processes of CPU access
  - Fairness = access to CPU
- Embedded systems must meet deadlines
  - Low-priority processes may not run for a long time
- Scheduling feasibility
  - Resource constraints make schedulability analysis NP-hard
  - Must show that the deadlines are met for all timings of resource requests

Scheduling Metrics

- How do we evaluate a scheduling policy:
  - Ability to satisfy all deadlines
  - CPU utilization---percentage of time devoted to useful work
  - Scheduling overhead---time required to make scheduling decision

Scheduling Approaches (Hard RTS)

- **Off-line scheduling / analysis** (static analysis + static scheduling)
  - All tasks, times and priorities given a priori (before system startup)
  - Time-driven; schedule computed and hardcoded (before system startup)
  - E.g., Cyclic Executives
  - Inflexible
  - May be combined with static or dynamic scheduling approaches
- **Fixed priority scheduling** (static analysis + dynamic scheduling)
  - All tasks, times and priorities given a priori (before system startup)
  - Priority-driven, dynamic(!) scheduling
    - The schedule is constructed by the OS scheduler at run time
  - For hard / safety critical systems
  - E.g., RMA/RMS (Rate Monotonic Analysis / Rate Monotonic Scheduling)
- **Dynamic priority scheduling**
  - Tasks times may or may not be known
  - Assigns priorities based on the current state of the system
  - For hard / best effort systems
  - E.g., Least Completion Time (LCT), Earliest Deadline First (EDF), Least Slack Time (LST)

Rate Monotonic Scheduling (RMS)

- RMS is widely-used, analyzable scheduling policy
- A static scheduling policy: processes have fixed priorities
- RMS Model
  - All processes run on single CPU
  - Context switching time is ignored
  - No data dependencies between processes
  - Process execution time is constant
  - All deadlines are at the end of the period
  - Highest-priority ready process runs first
- Priority assignment: The process with the shortest period is assigned the highest priority

Rate Monotonic Analysis: Assumptions

A1: Tasks are periodic (activated at a constant rate).
Period P_i = Interval between two consecutive activations of task T_i

A2: All instances of a periodic task T_i have the same computation time C_i

A3: All instances of a periodic task T_i have the same relative deadline, which is equal to the period $(D_i = P_i)$

A4: All tasks are independent
(i.e., no precedence constraints and no resource constraints)

Implicit assumptions:

A5: Tasks are preemptable

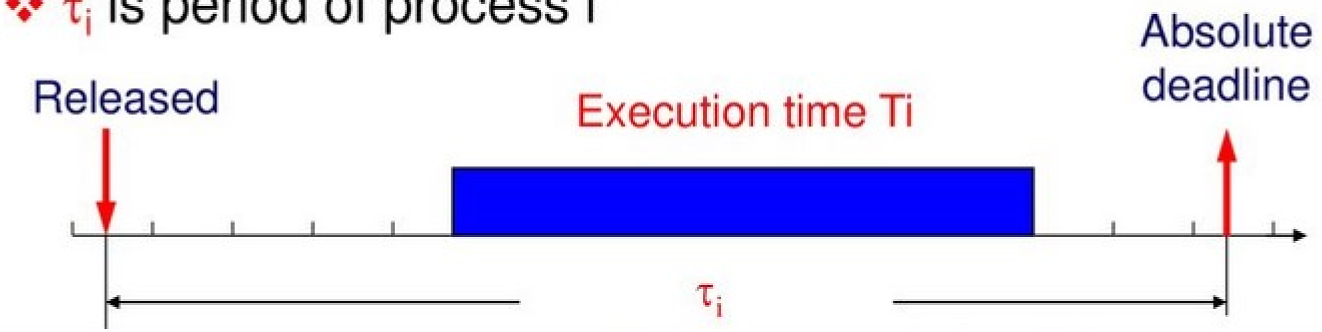
A6: No task can suspend itself

A7: All tasks are released as soon as they arrive

A8: All overhead in the kernel is assumed to be zero (or part of C_i)

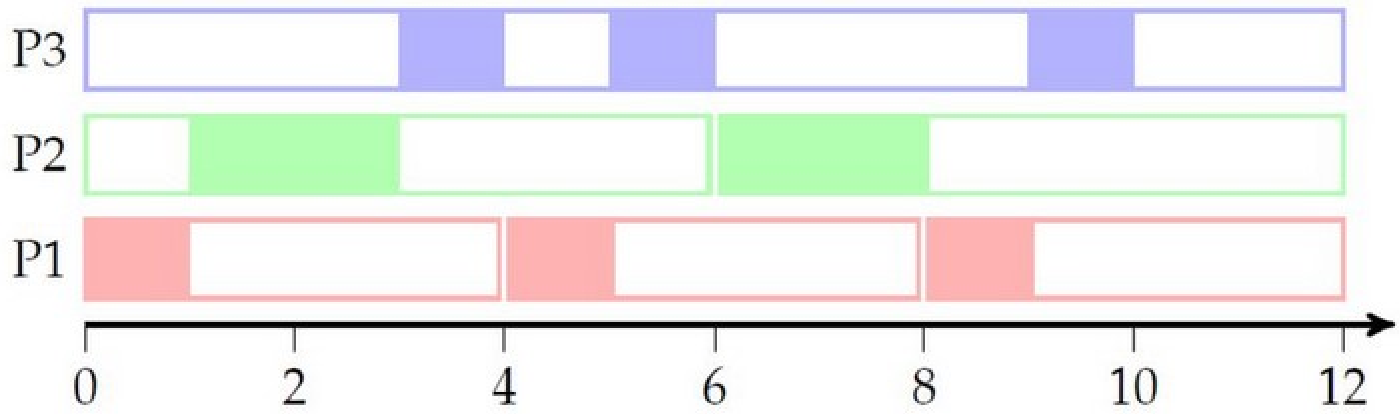
Rate Monotonic Scheduling (RMS)

- RMS always provides a feasible schedule if such a schedule exists with fixed priority
- Optimal static assignment
  - No fixed-priority scheme does better
  - Highest CPU utilization while ensuring that all processes meet their deadlines
- T_i is computation time of process i
- τ_i is period of process i



Rate Monotonic Scheduling Example

Process	Execution time	Period
P1	1	4
P2	2	6
P3	3	12



❖ What if execution times become 2, 3, 3?

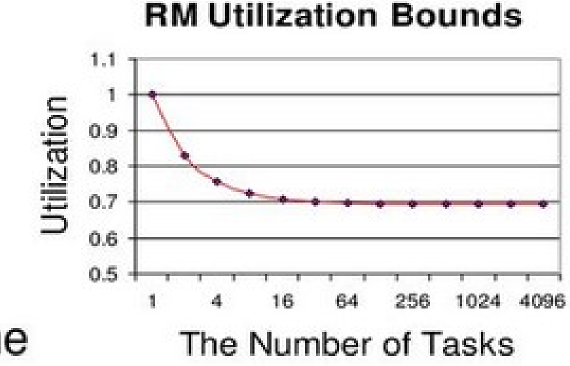
RMS CPU utilization

- Utilization for n processes is: $U = \sum_{i=1}^{n} \frac{T_i}{\tau_i}$

- Given *n* processes and ratio between any two periods less than 2, RMS CPU utilization upper bound:

$$U \leq n(2^{1/n} - 1)$$

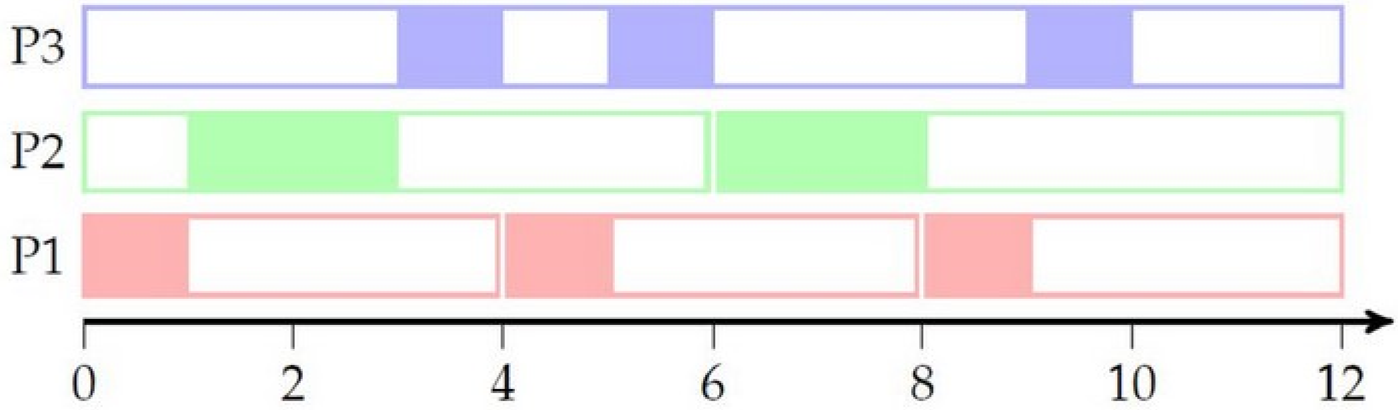
  - n = 2; $U \leq 0.83$
  - n = 3; $U \leq 0.78$
  - $n \to \infty$; $U \leq \ln 2 \approx 0.69$, 31% idle time



- RMS may not be able to use 100% of CPU, even with zero context switch overhead

RMS CPU Utilization Example

Process	Execution time	Period
P1	1	4
P2	2	6
P3	3	12



❖ Utilization = 1/4 + 2/6 + 3/12 = 0.83

RMS- Schedulability Check

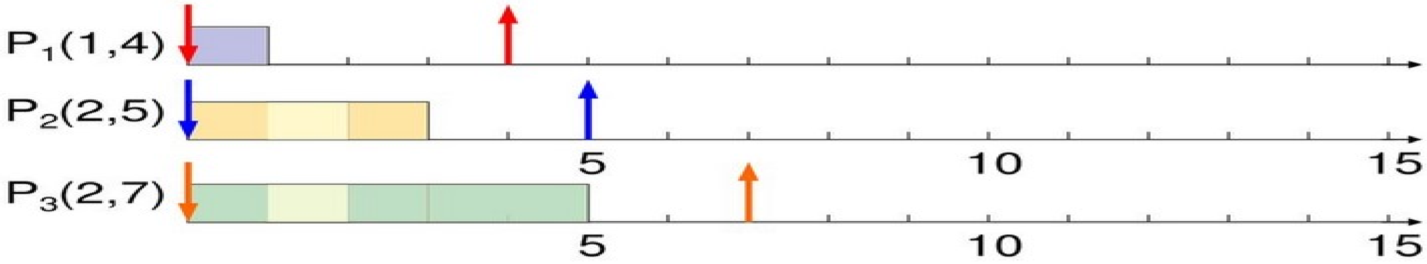
- A set of n processes is schedulable on a uniprocessor by the RMS algorithm if the processor utilization (utilization test):

$$U \leq n(2^{1/n} - 1)$$

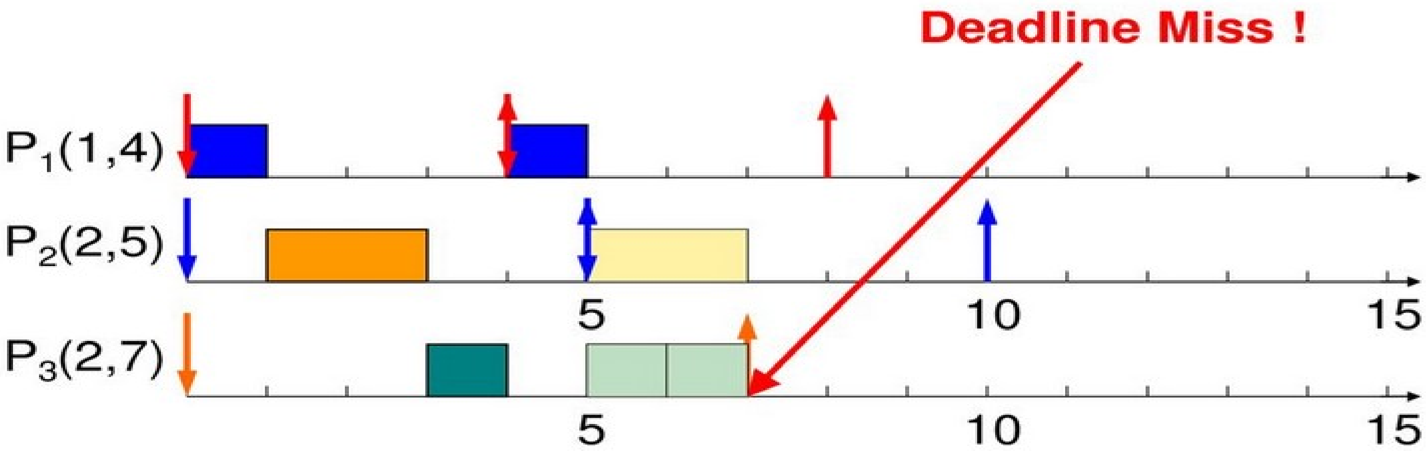
- This condition is sufficient, but not necessary
  - If U is less than or equal to the given bound, a schedule exists
  - If there is a schedule, U could be greater than the bound
- Example:
  - P1: ($T1=1$, $\tau1=2$), P2: ($T2=2$, $\tau2=4$)
  - There is a schedule with U=100%

Another RMS Example

P1: T1=1, τ1=4 P2: T2=2, τ2=5 P3: T3=2, τ3=7



$n = 3;\ U \le 0.78$ $U = 0.936$



$n = 3;\ U \le 0.78$ $U = 0.936$

Earliest-Deadline-First Scheduling

- EDFS: dynamic priority scheduling scheme
- Process closest to its deadline has highest priority
- Requires recalculating processes priorities at every timer interrupt
- Can achieve 100% utilization;higher utilization than RMS
- On each timer interrupt
  - compute time to deadline; choose process closest to deadline
- Optimal scheduling algorithm
  - if there is a schedule for a set of real-time tasks, EDF can schedule it
  - Real-time system is schedulable under EDFS iff $\sum U_i \leq 1$

EDF: Assumptions

A1: Tasks are periodic or aperiodic.

Period P_i = Interval between two consecutive activations of task T_i

A2: All instances of a periodic task T_i have the same computation time C_i

A3: All instances of a periodic task T_i have the same relative deadline, which is equal to the period $(D_i = P_i)$

A4: All tasks are independent (i.e., no precedence constraints and no resource constraints)

Implicit assumptions:

A5: Tasks are preemptable

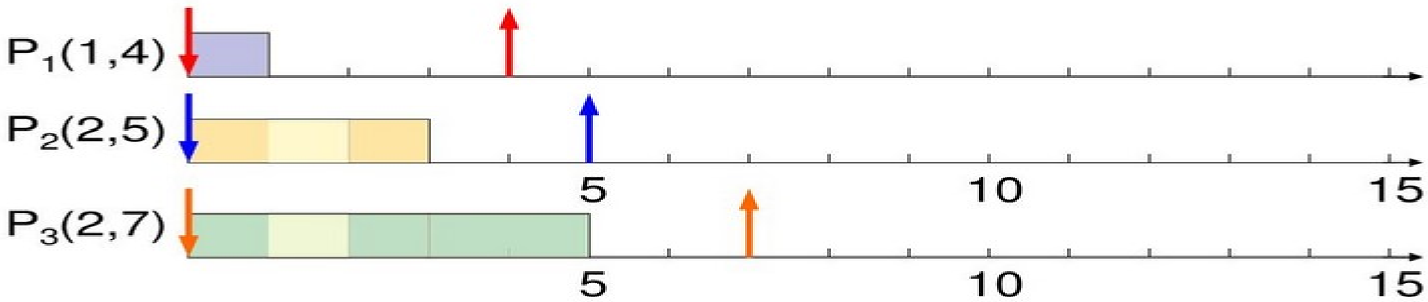
A6: No task can suspend itself

A7: All tasks are released as soon as they arrive

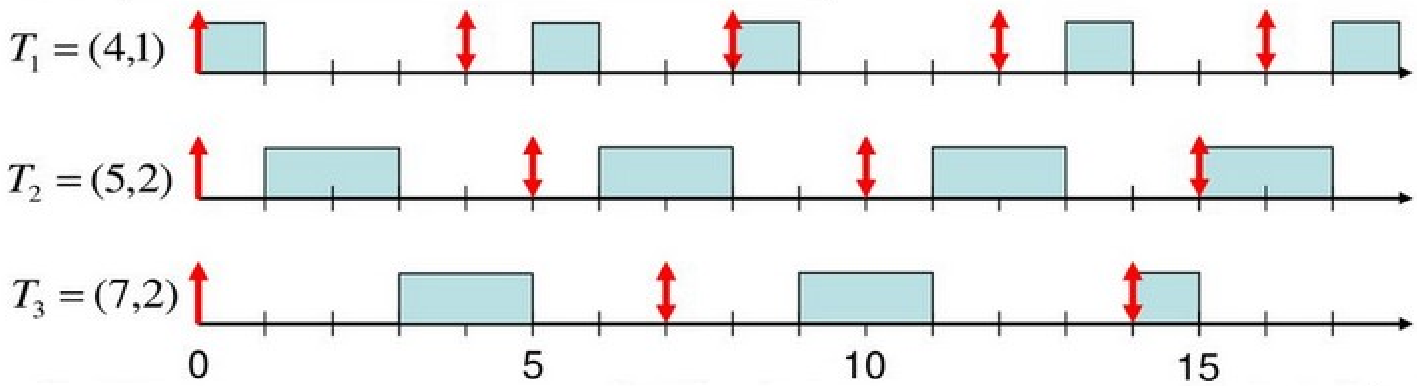
A8: All overhead in the kernel is assumed to be zero (or part of C_i)

EDFS Example

P1: T1=1, τ1=4 P2: T2=2, τ2=5 P3: T3=2, τ2=7



- Preemptive priority-based dynamic scheduling
- Each task is assigned a (current) priority based on how close the absolute deadline is.
- The scheduler always schedules the <u>active task</u> with the <u>closest absolute deadline</u>.



RM vs. EDF

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

- EDF
  - Full processor utilization
  - Misbehavior during overload conditions