

CprE 488 – Embedded Systems Design

Lecture 5 – Embedded Operating Systems

Phillip Jones

Electrical and Computer Engineering

Iowa State University

www.ece.iastate.edu/~phjones

rcl.ece.iastate.edu

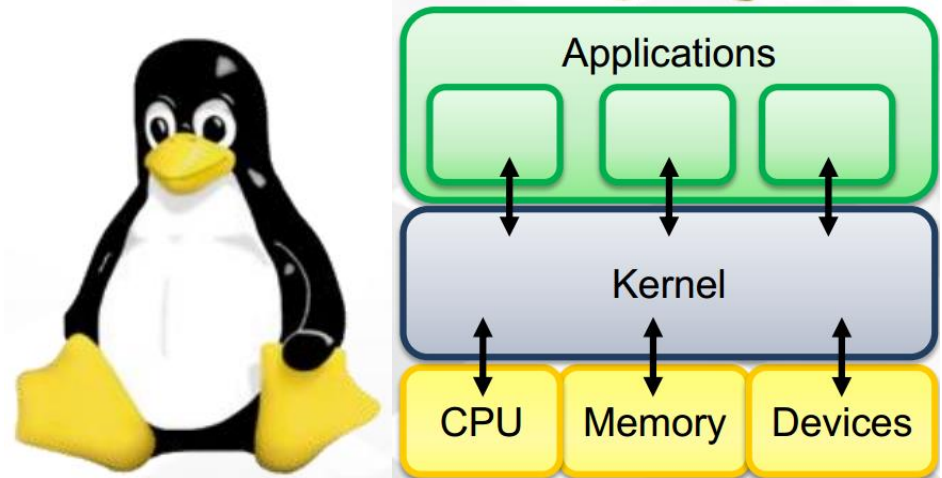
...the Linux philosophy is "laugh in the face of danger". Oops. Wrong one. "Do it yourself". That's it. – Linux Torvalds

Announcements

- **Exam 2: Tues 5/6**
 - Can have 1-side of one-page of **handwritten** notes.
 - There is a paper and a Canvas part of the Exam in class
 - Bring an electronic device for taking the Canvas part
 - You are encouraged to have a calculator
- **HW4: Tue 4/8**
- **Class Project Topic:**
 - ~~Project Idea draft proposal, and draft Rubric (due **Wed 4/2**)~~
 - Project Proposal Presentation (In class **Tue 4/8**)
 - Project Demos in Lab at Final Exam time: **Thur 5/15**, 9:45am)
 - Demos similar to normal MP demo day, with the option to **additionally** give a live demo
 - **Make project grading Rubric last slide of all presentations**
- **MP-3: Demo Day Thur 4/10**

Motivation

- We have already run into some limitations of the standalone process model:
 - Single application, growing in complexity quickly
 - Lots of polling loops, deep nested 'if' statements
- We could continue in this direction, but a modern Operating System (OS) provides streamlined mechanisms for:
 - Preemptive multitasking
 - Device drivers
 - Memory management
 - File systems



- It would be insane to try to cover all the major issues involved in embedded OS in a single lecture

This Week's Topic

- Embedded Operating System features
 - Processes and scheduling
 - Context switching
 - Scheduler policies
 - Real-Time Operating Systems (RTOS)
 - Atomic operations
 - Inter-processes communication
 - Virtual memory
 - Examples along the way:
 - Linux, POSIX, freeRTOS.org
 - ARM architecture support
- Reading: Wolf chapter 6, 3.5

Reactive Systems

- Respond to external events:
 - Engine controller
 - Seat belt monitor
- Requires real-time response:
 - System architecture
 - Program implementation
- May require a chain reaction among multiple processors

Tasks and Processes

Task:

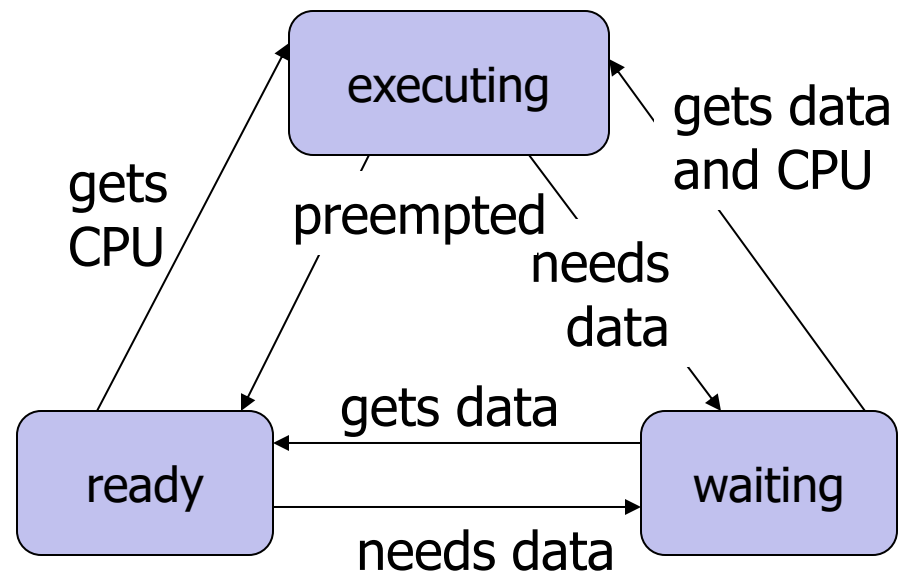
- A task is a functional description of a connected set of operations
- Task can also mean a collection of processes

Process:

- 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
 - memory
- The operating system manages processes

Process State

- A process can be in one of three states:
 - **executing** on the CPU
 - **ready** to run
 - **waiting** for data

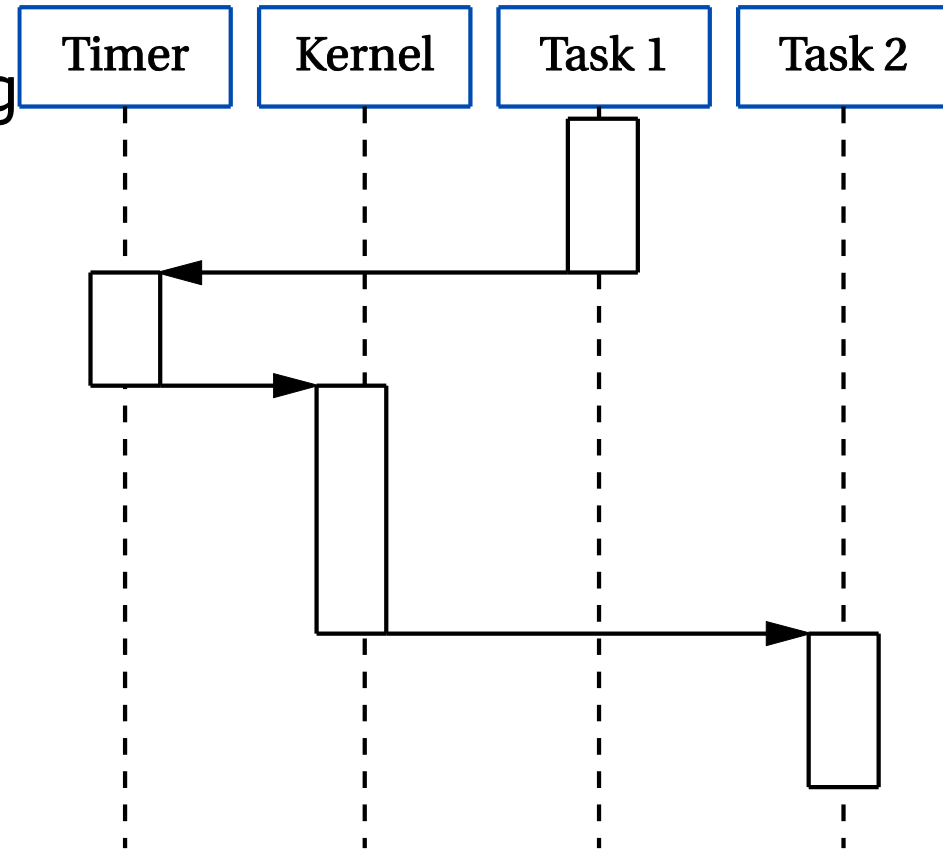


Embedded vs. General-Purpose Scheduling

- 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

Preemptive Scheduling

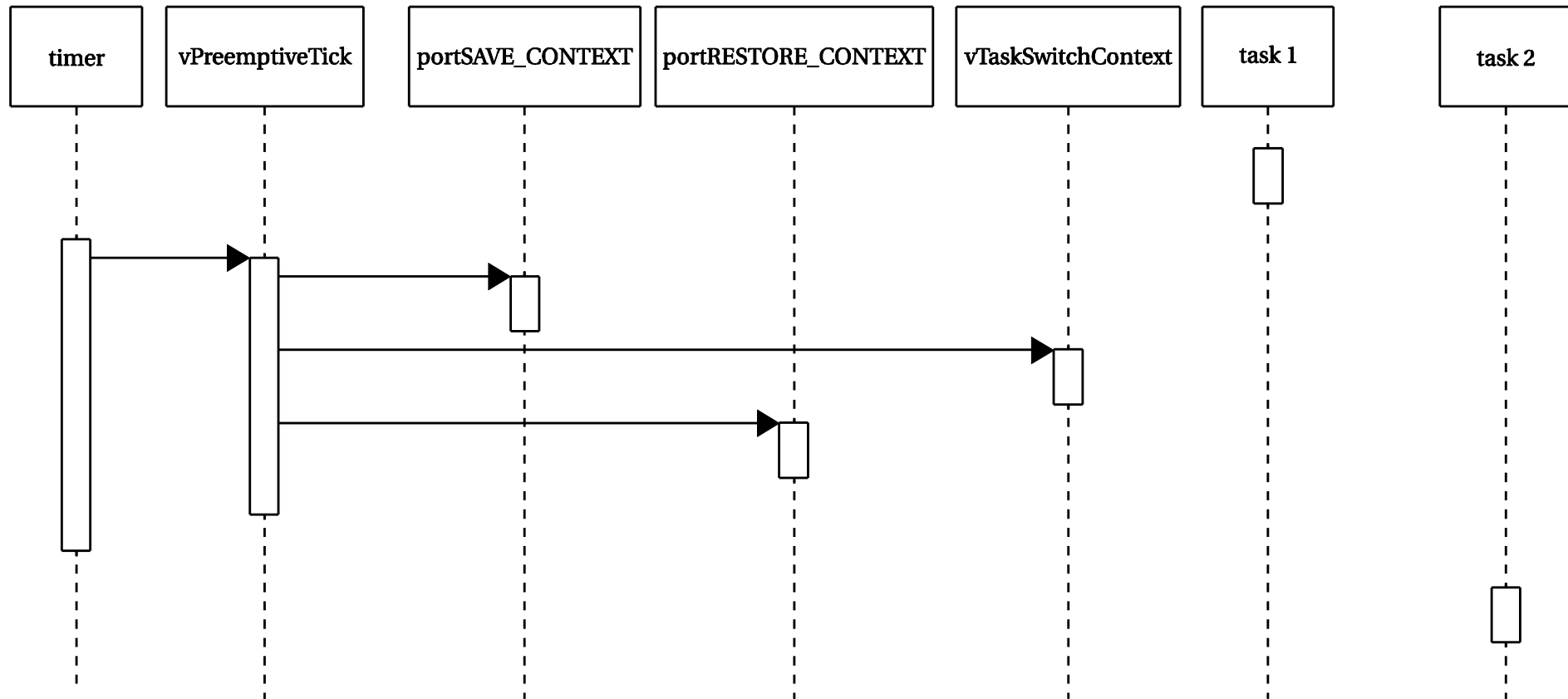
- Timer interrupt gives CPU to kernel
 - Time quantum is smallest increment of CPU scheduling time
- Kernel decides what task runs next
- Kernel performs context switch to new context



Context Switching

- Set of registers that define a process's state is its context
 - Stored in a record
- Context switch moves the CPU from one process's context to another
- Context switching code is usually assembly code
 - Restoring context is particularly tricky

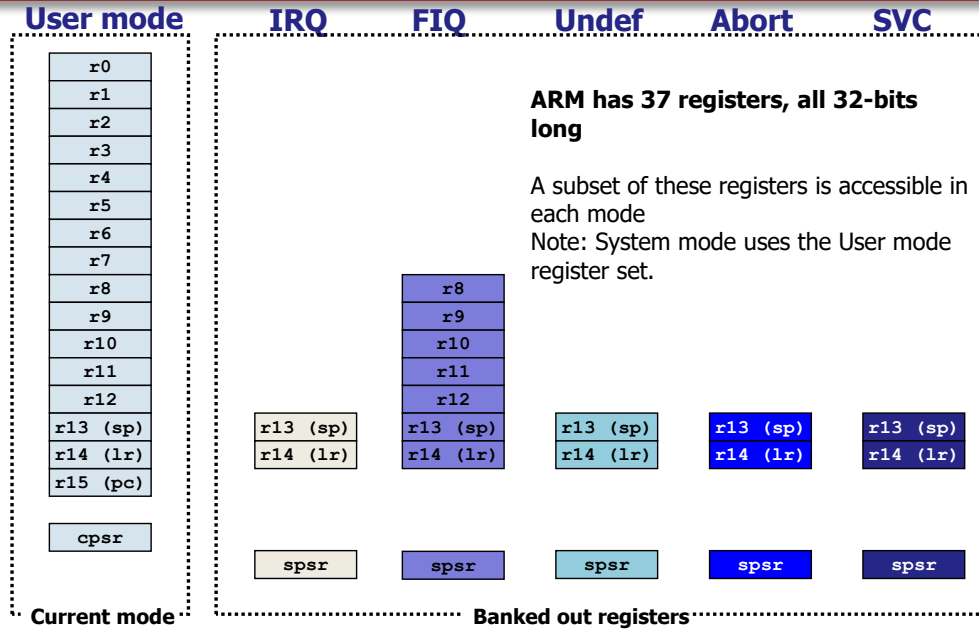
freeRTOS.org Context Switch



freeRTOS.org Timer Handler

```
void vPreemptiveTick( void ) {
    /* Save the context of the current task. */
    portSAVE_CONTEXT();
    /* Increment the tick count - this may wake a task. */
    vTaskIncrementTick();
    /* Find the highest priority task that is ready to run. */
    vTaskSwitchContext();
    /* End the interrupt in the AIC. */
    AT91C_BASE_AIC->AIC_EOICR = AT91C_BASE_PITC->PITC_PIVR;
    portRESTORE_CONTEXT();
}
```

ARM Context Switch



```

STM      sp, {R0-lr}^      ; Dump user registers above R13.
MRS      R0, SPSR          ; Pick up the user status
STMDB   sp, {R0, lr}      ; and dump with return address below.
LDR      sp, [R12], #4    ; Load next process info pointer.
CMP      sp, #0           ; If it is zero, it is invalid
LDMDBNE sp, {R0, lr}     ; Pick up status and return address.
MSRNE   SPSR_cxsf, R0    ; Restore the status.
LDMNE   sp, {R0-lr}^    ; Get the rest of the registers
NOP                                           ; and return and restore CPSR.
SUBSNE  pc, lr, #4       ; Insert "no next process code" here.
    
```

Real-Time Systems

- What is a real-time system?
- Which of the following is real-time?
 - Program that processes 100 video frames per sec?
 - Program that that processes 1 video frame per 10 secs?
- A better name
 - “Get things done on time” Systems
- They are about getting things done **on time**, not getting things done fast

Real-Time Systems: Key Terms/Concepts

- Task
 - Cost: time for processor to complete task without interruptions
 - Release time: when task is ready to be run
 - Deadline: time by which task needs to be completed
 - Period: time between release times
- Task-set schedule: order in which tasks are allocated the CPU
- Scheduling policy (algorithm): means by which (i.e. rules followed) to create a task-set schedule

Scheduling: Period vs Aperiodic

- **Periodic process**: executes on every period
- **Aperiodic process**: executes on demand
- Analyzing aperiodic process sets is harder---must consider worst-case combinations of process activations

Timing Requirements on Processes

- **Period**: interval between process activations
- **Initiation interval**: reciprocal of period
- **Initiation time**: time at which process becomes ready
- **Deadline**: time at which process must finish

- What happens if a process doesn't finish by its deadline?
 - **Hard deadline**: system fails if missed
 - **Soft deadline**: user may notice, but system doesn't necessarily fail

Priority-driven Scheduling

- Each process has a priority
- CPU goes to highest-priority process that is ready
- Priorities determine scheduling policy:
 - Fixed (Static) priority
 - Time-varying (Dynamic) priorities

Priority-driven Scheduling Example

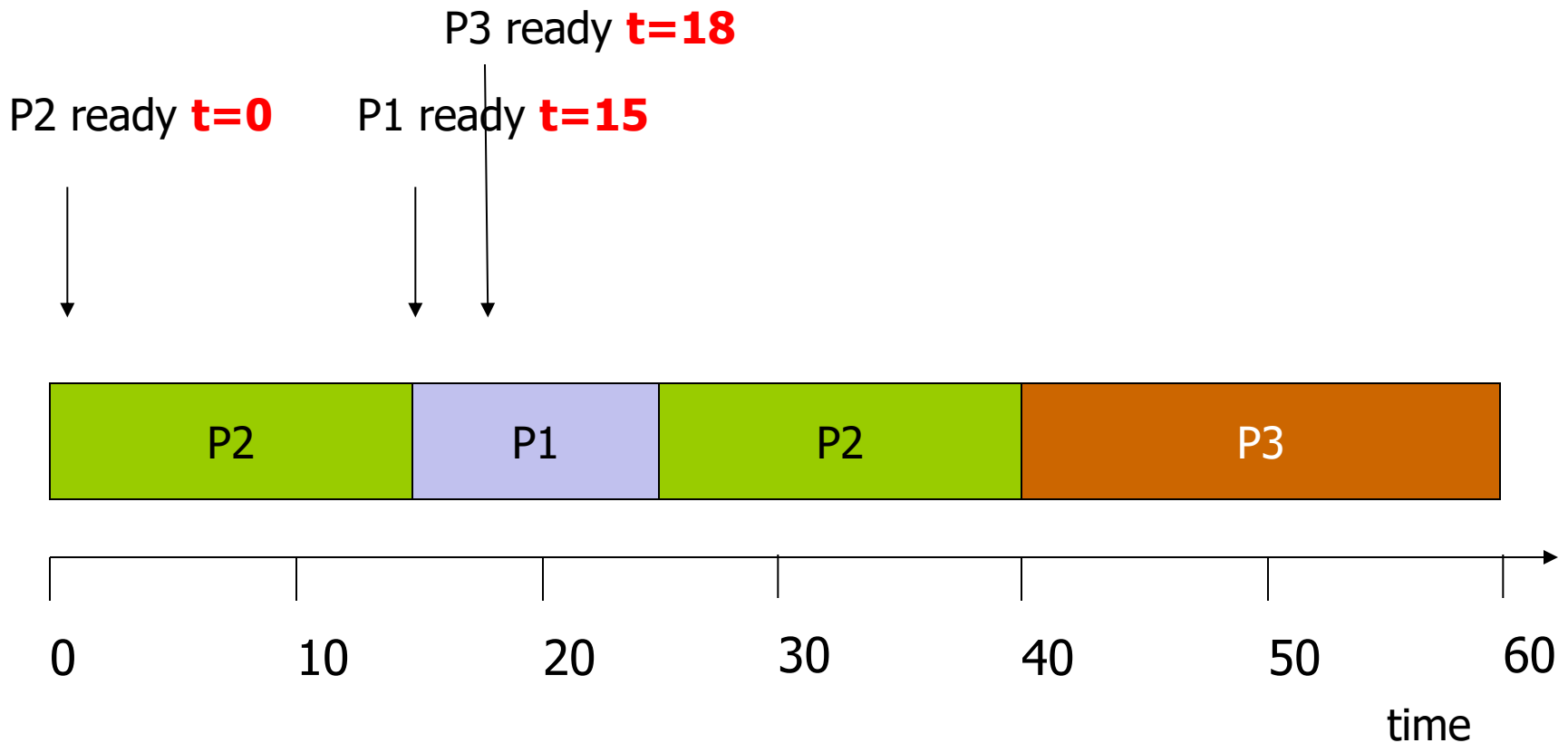
- Rules:
 - Each process has a fixed priority (1 highest)
 - Highest-priority ready process gets CPU
 - Process will not self stop (i.e. block) until done or is preempted by a high priority process
- Processes
 - P1: priority 1, execution time 10, release time 15
 - P2: priority 2, execution time 30, release time 0
 - P3: priority 3, execution time 20, release time 18

Priority-driven Scheduling Example (cont.)

P1: priority 1, execution time 10, release time 15

P2: priority 2, execution time 30, release time 0

P3: priority 3, execution time 20, release time 18



The Scheduling Problem

- Can we meet all deadlines?
 - Must be able to meet deadlines in all cases
- How much CPU horsepower do we need to meet our deadlines?

CPU Utilization

- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- What is the CPU utilization of T1?

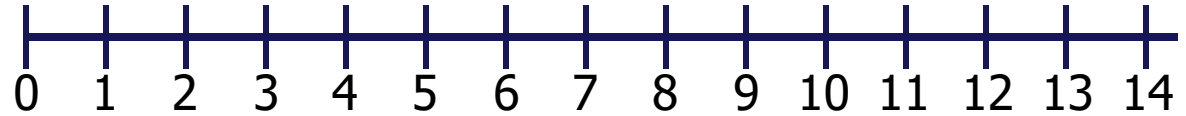
$$\mathbf{Task_CPU_Utilization} = \frac{\mathbf{Task_Cost}}{\mathbf{Task_Period}}$$

Scheduling Example (no preemption)

- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms
- T3: Check Sensors
 - Cost = 6 ms
 - Deadline = 12 ms
 - Period = 12 ms
- What rules to follow for scheduling
 - Let's say that the more often a Task needs to run, the higher the priority (allow **NO** preemption)
 - Is there a release pattern that can cause a task to miss a deadline?

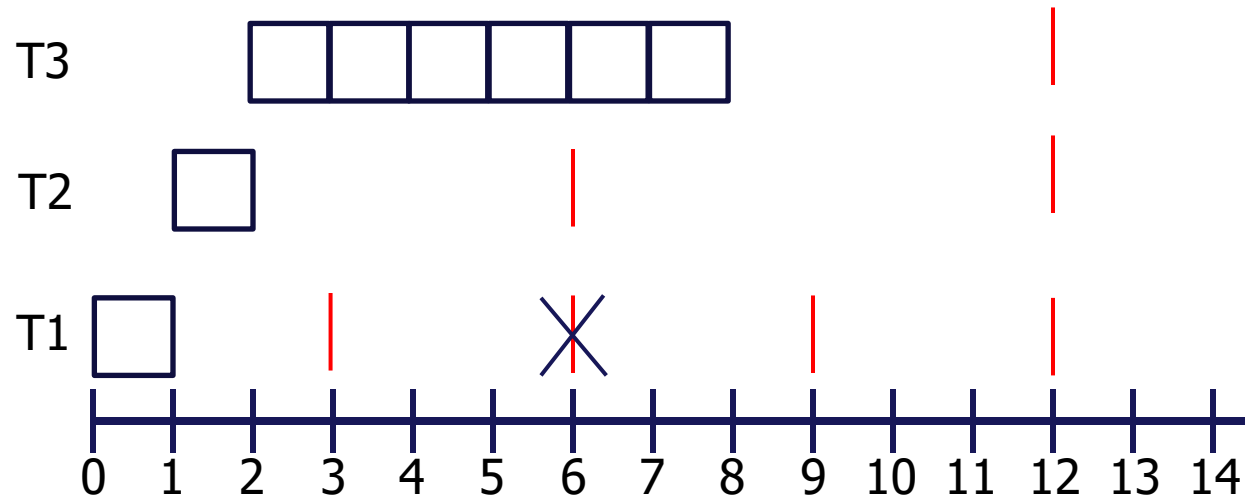
Scheduling Example (no preemption)

- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms
- T3: Check Sensors
 - Cost = 6 ms
 - Deadline = 12 ms
 - Period = 12 ms
- What rules to follow for scheduling
 - Let's say that the more often a Task needs to run, the higher the priority (allow **NO** preemption)
 - Is there a release pattern that can cause a task to miss a deadline?



Scheduling Example (no preemption)

- T1: PPM update
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms
- T2: Video processing
 - Cost = 1 ms
 - Deadline = 12 ms
 - Period = 12 ms
- T3: Check Sensors
 - Cost = 6 ms
 - Deadline = 12 ms
 - Period = 12 ms



- What rules to follow for scheduling
 - Let's say that the more often a Task needs to run, the higher the priority (allow **NO** preemption)
 - Is there a release pattern that can cause a task to miss a deadline?

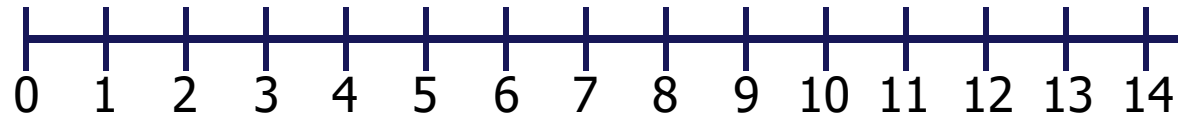
Scheduling Example (with preemption)

- T1: PPM update

- Cost = 1 ms
- Deadline = 3 ms T3
- Period = 3 ms T2

- T2: Video processing

- Cost = 1 ms T1
- Deadline = 6 ms
- Period = 6 ms



- T3: Check Sensors

- Cost = 6 ms
- Deadline = 12 ms
- Period = 12 ms

- What rules to follow for scheduling

- Let's say that the more often a task needs to run, the higher the priority (allow preemption)
- Draw out schedule and see if we miss a deadline

Scheduling Example (with preemption)

- T1: PPM update

- Cost = 1 ms
- Deadline = 3 ms
- Period = 3 ms

- T2: Video processing

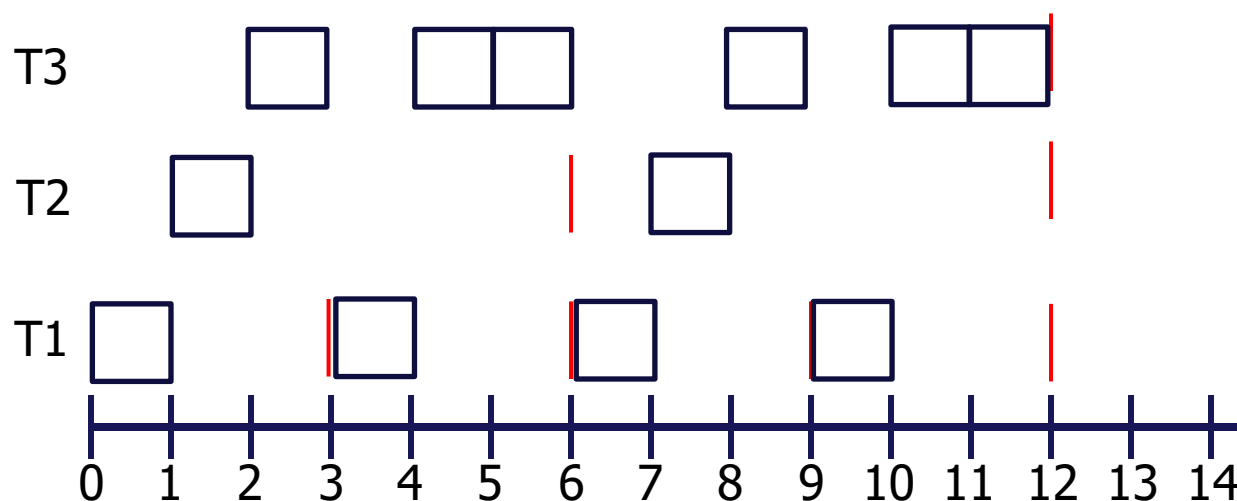
- Cost = 1 ms
- Deadline = 6 ms
- Period = 6 ms

- T3: Check Sensors

- Cost = 6 ms
- Deadline = 12 ms
- Period = 12 ms

- What rules to follow for scheduling

- Let's say that the more often a task needs to run, the higher the priority (allow preemption)
- Draw out schedule and see if we miss a deadline



Scheduling Metrics

- How do we evaluate a scheduling policy:
 - Ability to satisfy all deadlines (Feasibility)
 - Often a trade-off between:
 - Task set CPU utilization (i.e., time doing “useful” work)
 - Scheduling overhead (time to make scheduling decision)

Scheduling Metrics: Feasibility

- For previous preemption example
 - How long do we have to draw out the schedule before we know we will never miss a deadline?
 - What if we had 3 tasks with period 3ms, 4ms, and 7ms?
 - For a general task set, how long do we have to draw out the schedule?

Scheduling Metrics: Feasibility

- For previous preemption example
 - How long do we have to draw out the schedule before we know we will never miss a deadline?
 - What if we had 3 tasks with period 3ms, 4ms, and 7ms?
Answer: 84 ms
 - For a general task set, how long do we have to draw out the schedule?
Answer: Lowest common multiple of Task periods (a task set's *Hyper Period*). This is the time it takes before all Tasks release times synchronize after time = 0
- Is there a better way to determine is feasible (i.e. schedule using a given policy)? Yes! **RMA / RMS**

Rate Monotonic Scheduling (RMS)

- **RMS** (Liu and Layland [1973]): widely-used, analyzable scheduling policy
- Analysis is known as **Rate Monotonic Analysis (RMA)**
 - All processes run on single CPU
 - Zero context switch time
 - No data dependencies between processes
 - Process execution time is constant
 - Process deadline is at end of its period
 - Highest-priority ready process runs
 - Processes with lower periods have higher priority

RMS Priorities

- RMS gives an **optimal** (fixed/static) priority assignment:
 - Shortest-period process gets highest priority
 - Priority inversely proportional to period
 - Break ties arbitrarily
- No fixed-priority scheme does better:
 - In other words, if RMS does not give you a feasible schedule, then no static priority assignment approach can

RMS CPU Utilization

- CPU utilization for a Task set of n tasks:

$$\text{CPU Utilization of Task Set} = \sum_{i=1}^n \left(\frac{\text{Cost}_i}{\text{Period}_i} \right) = \sum_{i=1}^n (\text{CPU_Utilization_Task}_i)$$

- **For RMS**: Independent of number of tasks in the set, RMS is guaranteed to give a feasible scheduled if CPU Utilization is less than 69%:
 - If the Task set Utilization $\leq 69\%$, then RMS is guaranteed to meet all deadlines
 - If Utilization $> 69\%$, then must draw schedule for the Lowest Common Multiple (LCM) of the Task set periods.
 - Positive: Quick way to determine Feasibility
 - Negative: Gives up about 30% of CPU Utilization

Earliest-Deadline-First Scheduling

- EDF: dynamic priority scheduling scheme
 - Process closest to its deadline has highest priority
 - Must recalculate process priorities at every timer interrupt

- EDF guaranteed to give a feasible schedule, if a Task-set's CPU utilization is less than or equal to 100%

Scheduling Example (First try RMS)

- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms

Scheduling Example (First try RMS)

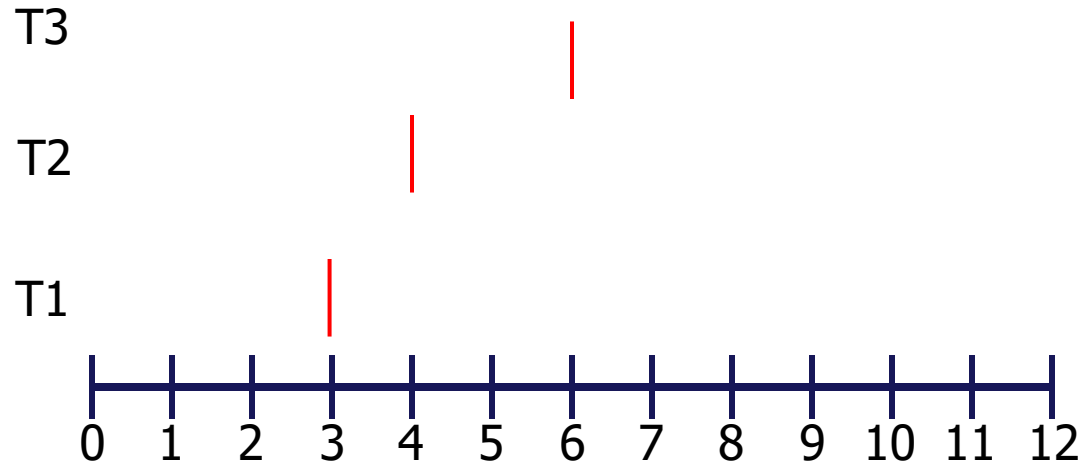
- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms

Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$

Hyper-Period = 12

Scheduling Example (First try RMS)

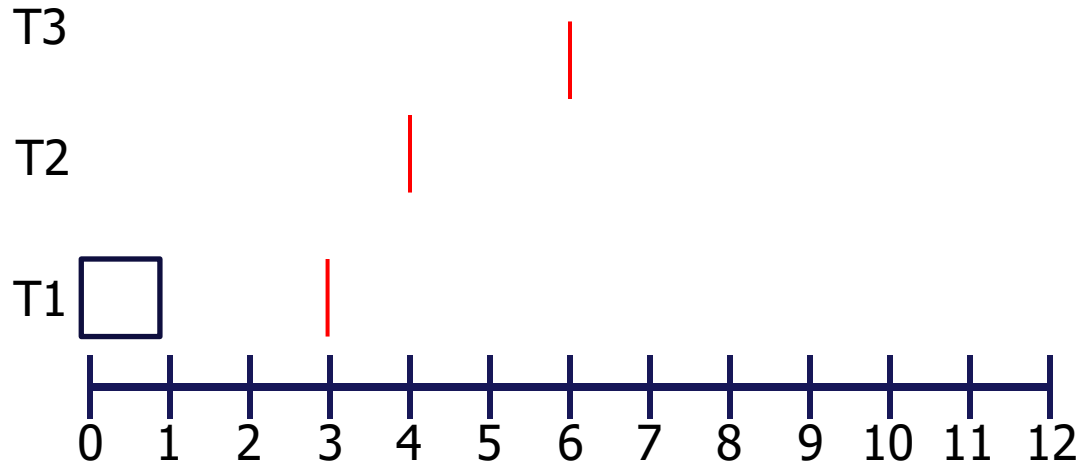
- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms



Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$
Hyper-Period = 12

Scheduling Example (First try RMS)

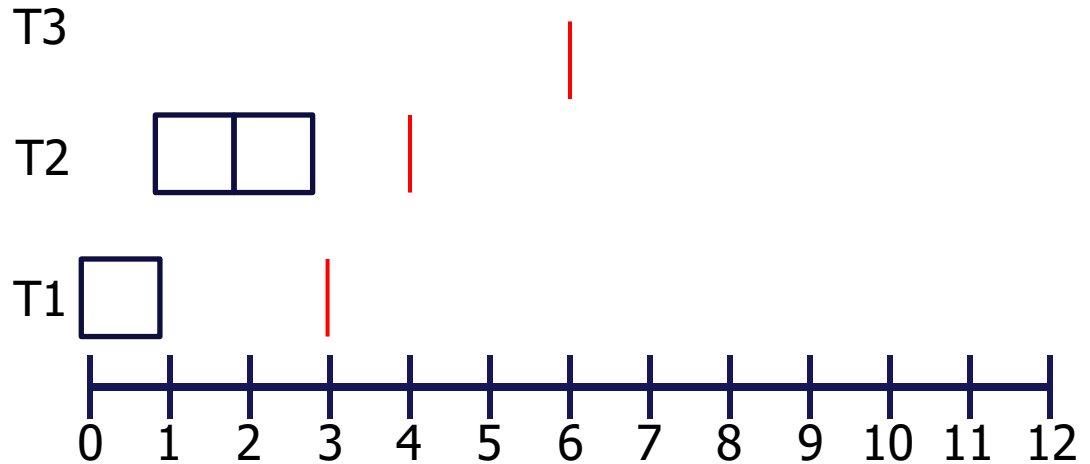
- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms



Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$
Hyper-Period = 12

Scheduling Example (First try RMS)

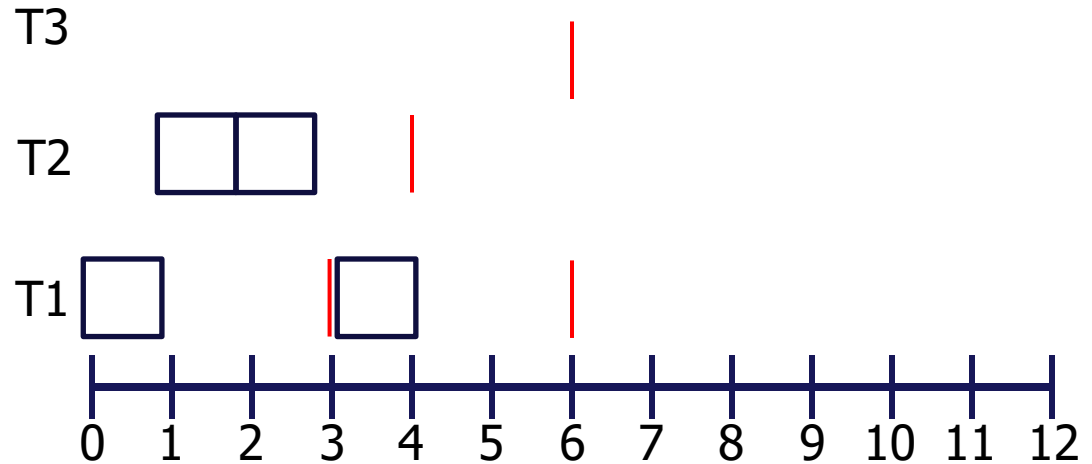
- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms



Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$
Hyper-Period = 12

Scheduling Example (First try RMS)

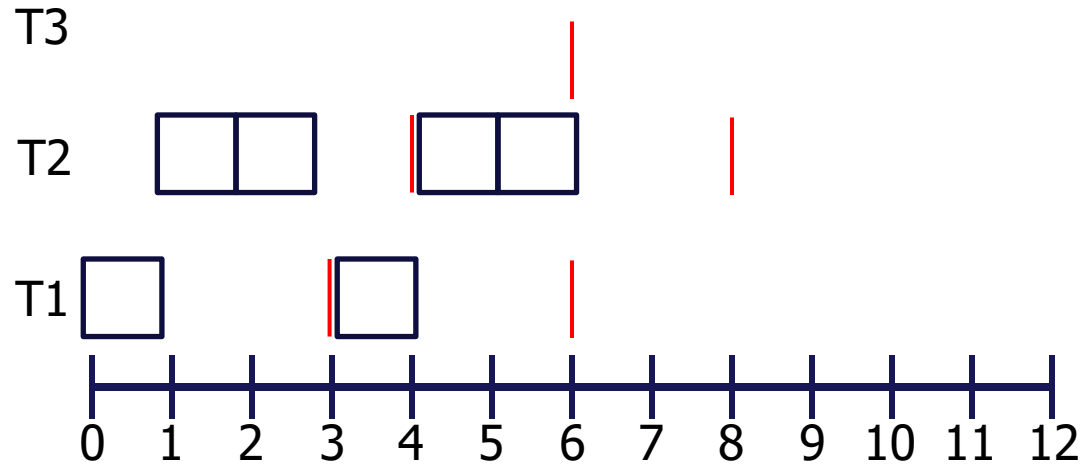
- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms



Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$
Hyper-Period = 12

Scheduling Example (First try RMS)

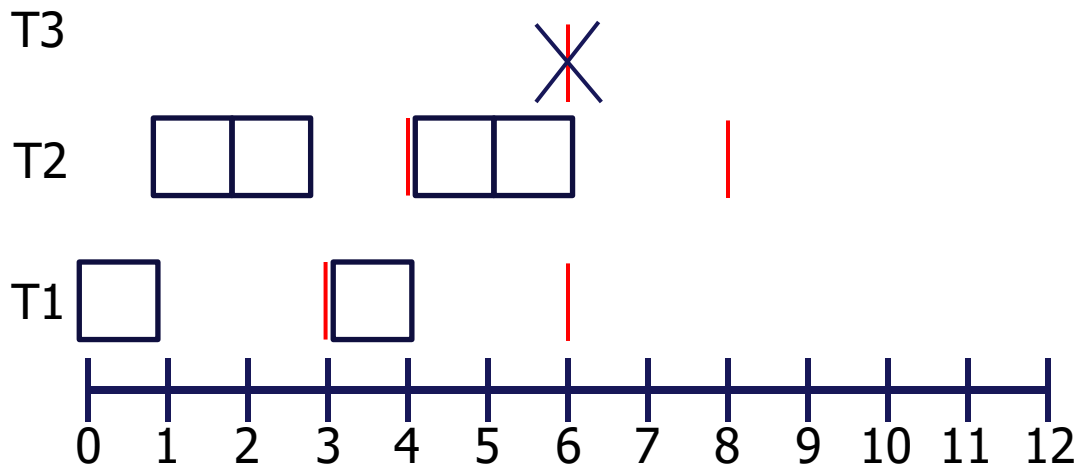
- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms



Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$
Hyper-Period = 12

Scheduling Example (First try RMS)

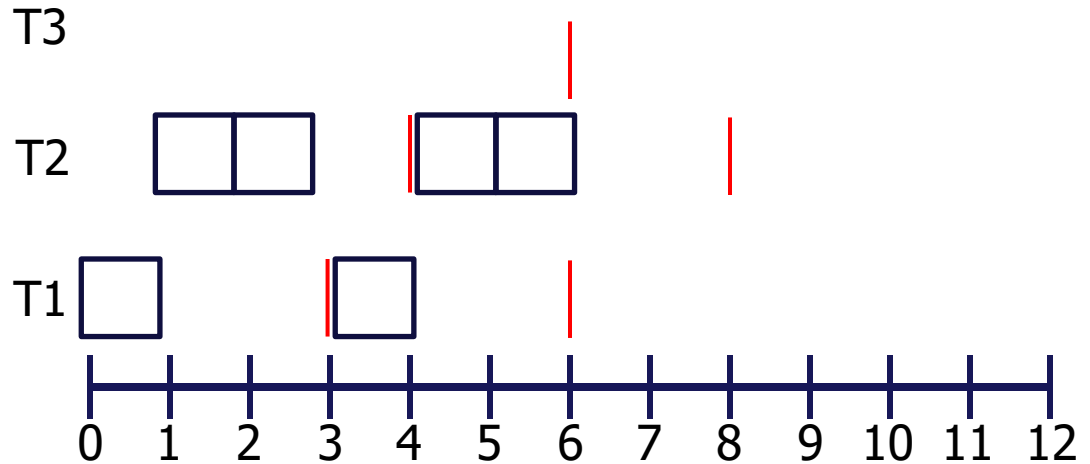
- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms



Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$
Hyper-Period = 12

Scheduling Example (Now try EDF)

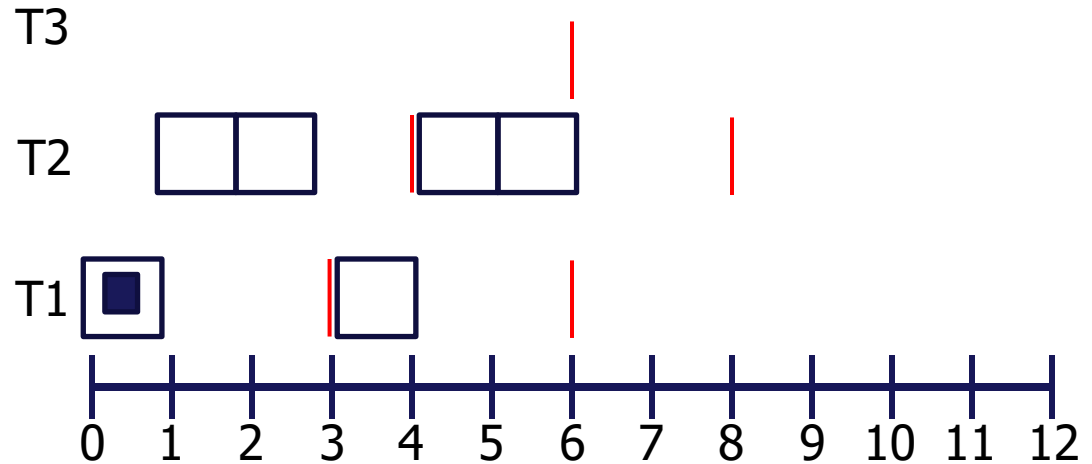
- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms



Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$
Hyper-Period = 12

Scheduling Example (Now try EDF)

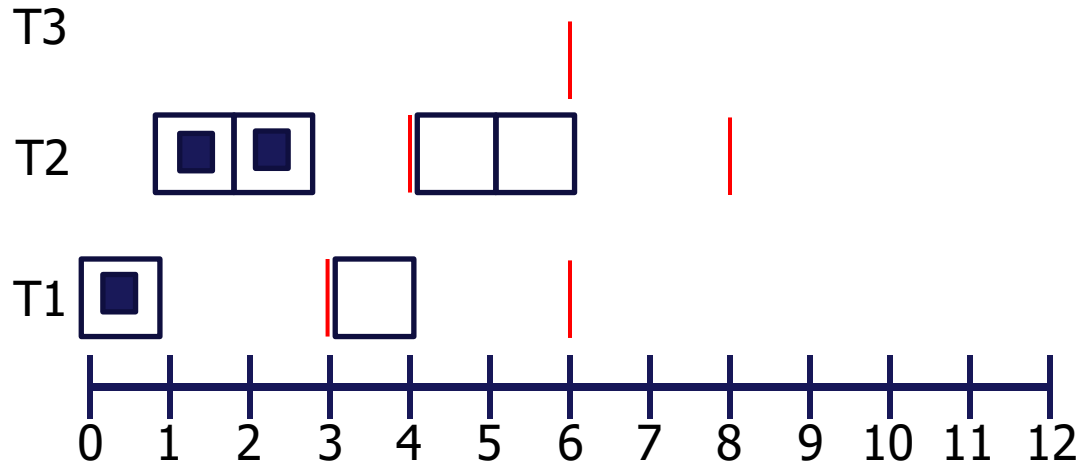
- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms



Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$
Hyper-Period = 12

Scheduling Example (Now try EDF)

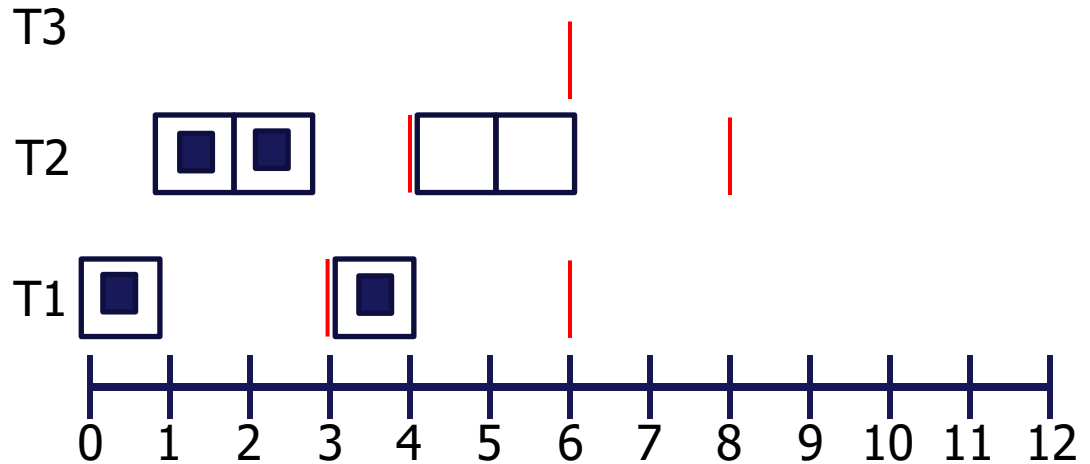
- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms



Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$
Hyper-Period = 12

Scheduling Example (Now try EDF)

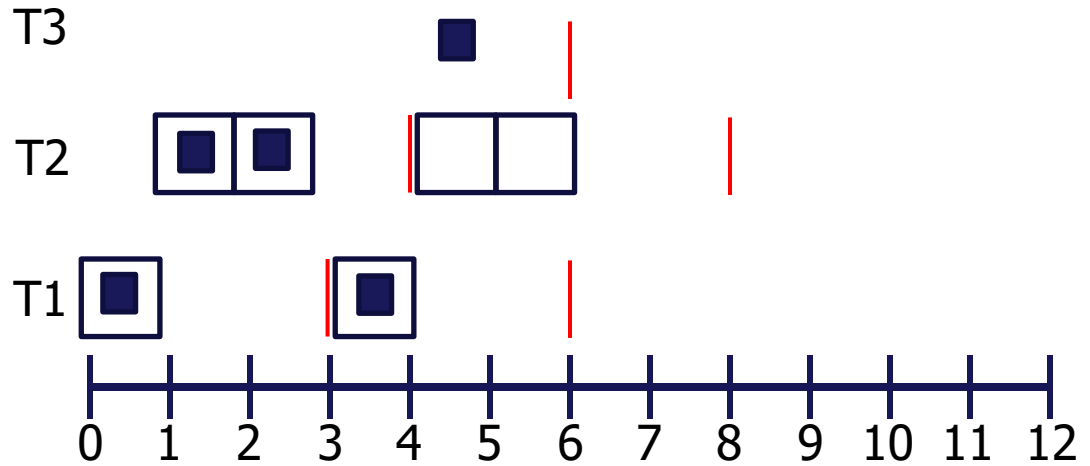
- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms



Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$
Hyper-Period = 12

Scheduling Example (Now try EDF)

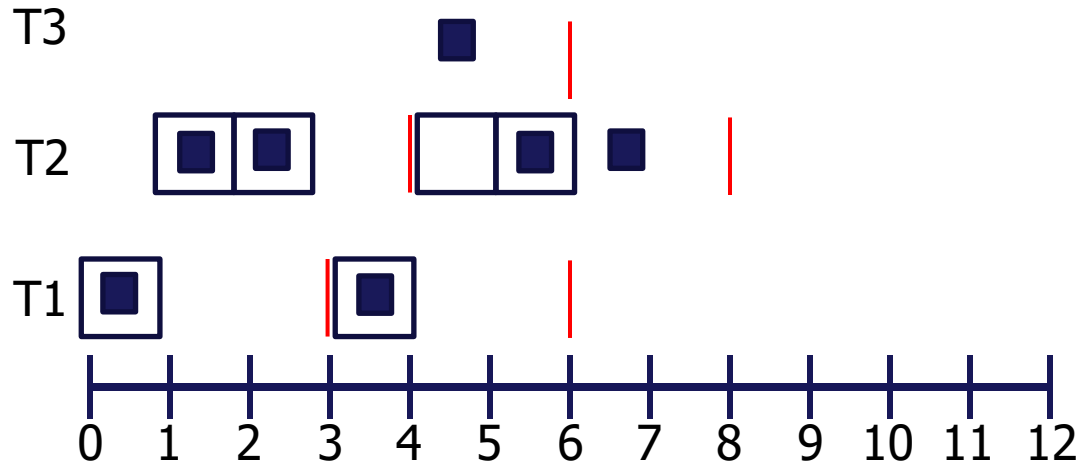
- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms



Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$
Hyper-Period = 12

Scheduling Example (Now try EDF)

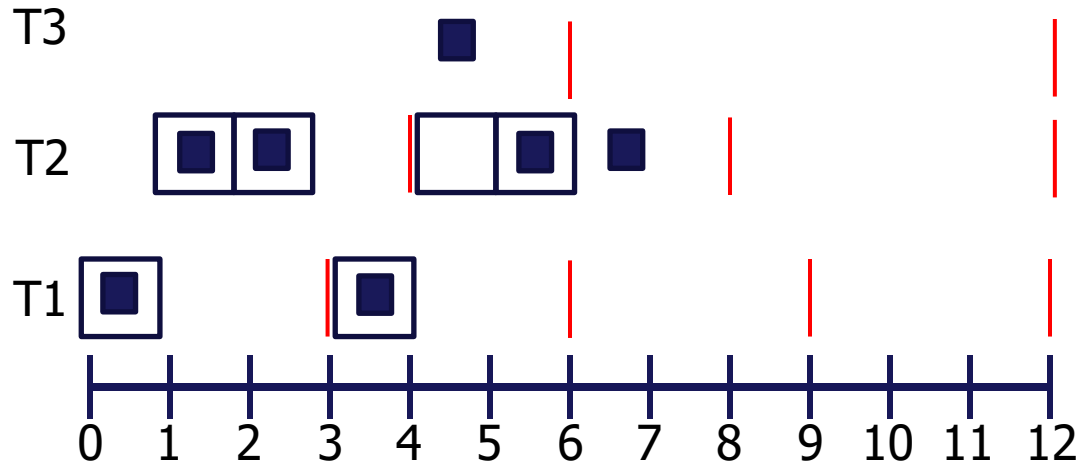
- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms



Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$
Hyper-Period = 12

Scheduling Example (Now try EDF)

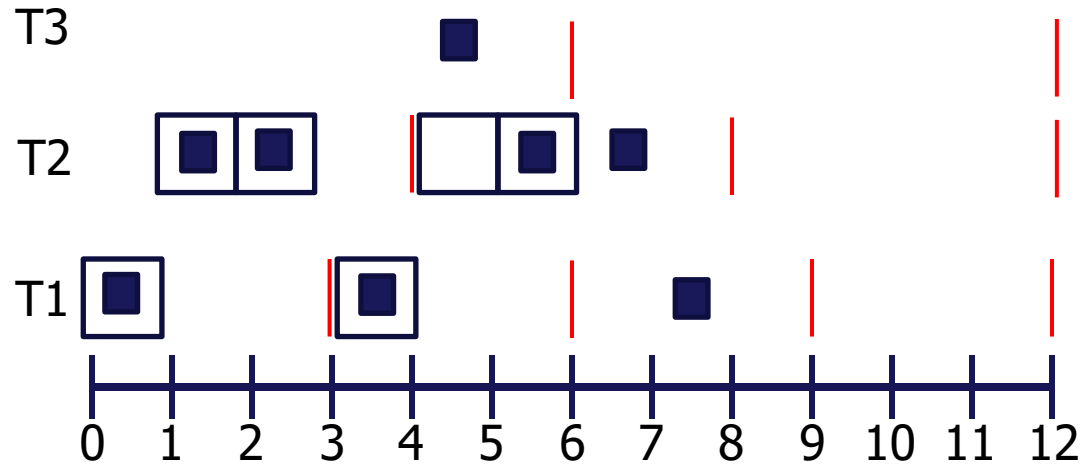
- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms



Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$
Hyper-Period = 12

Scheduling Example (Now try EDF)

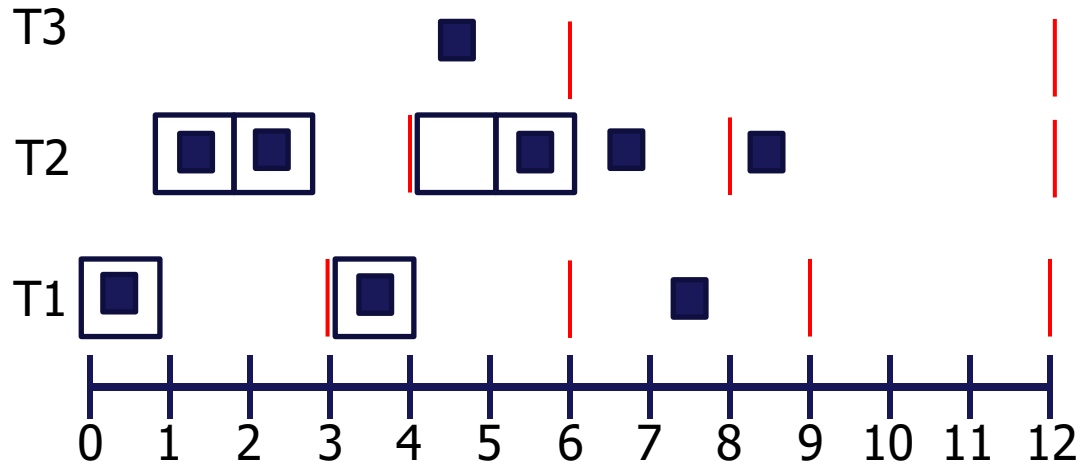
- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms



Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$
Hyper-Period = 12

Scheduling Example (Now try EDF)

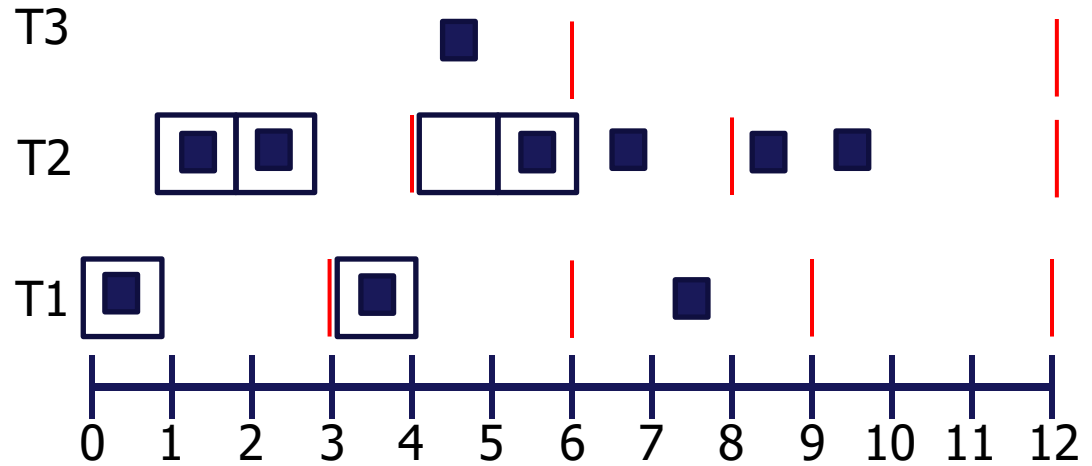
- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms



Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$
Hyper-Period = 12

Scheduling Example (Now try EDF)

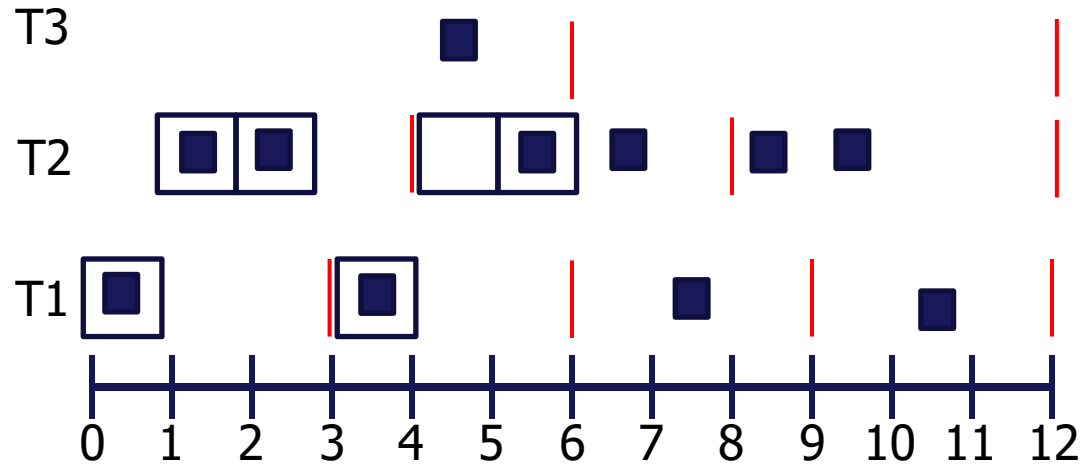
- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms



Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$
Hyper-Period = 12

Scheduling Example (Now try EDF)

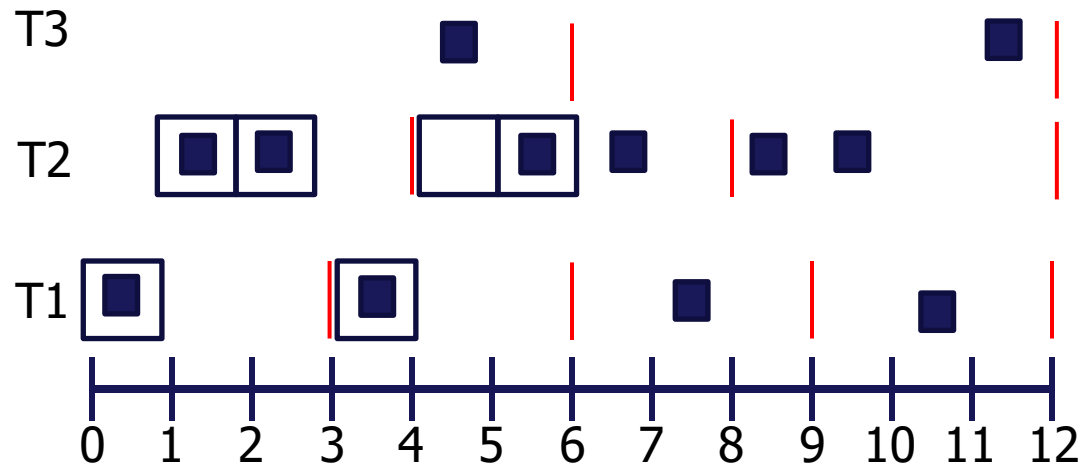
- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms



Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$
Hyper-Period = 12

Scheduling Example (Now try EDF)

- T1: PPM update
 - Cost = 1 ms
 - Deadline = 3 ms
 - Period = 3 ms
- T2: Video processing
 - Cost = 2 ms
 - Deadline = 4 ms
 - Period = 4 ms
- T3: Check Sensors
 - Cost = 1 ms
 - Deadline = 6 ms
 - Period = 6 ms



Task-set CPU Utilization: $4/12 + 6/12 + 2/12 = 12/12 = 100\%$
Hyper-Period = 12

Scheduling Metrics

- How do we evaluate a scheduling policy:
 - Ability to satisfy all deadlines (Feasibility)
 - Often a trade-off between:
 - Task set CPU utilization (i.e., time doing “useful” work)
 - Scheduling overhead (time to make scheduling decision)

EDF Implementation

- On each timer interrupt:
 - Compute time to deadline for each task
 - Choose process closest to deadline
- Generally considered too expensive to use in practice
 - EDF guarantees meeting all task deadlines if task set CPU utilization is less than or equal to 100%, but a major underlining assumption is that updating priorities at each timer interrupt take zero CPU time.
 - For fine grain timer ticks (i.e., short time between timer interrupts), the overhead for re-computing priorities can easily cost more CPU time than the task set!!
- **Note for RMS**: If task periods of a task set are “**Harmonic**” (i.e., period of each task is a multiple of each task with a smaller period), then RMS is **guaranteed** to schedule a task set with up to **100%** utilization

Scheduling Concerns

- **Task Context Switching Time**: Both EDF and RMS assume zero time for task context switching. In a real system one must be careful that indeed the time for context switching is **MUCH** smaller than the period of the tasks in the system. This can be a little tricky to account for in a real system.
 - Non-zero context switch time can push limits of a tight schedule
 - Hard to calculate effects---depends on order of context switches
 - In practice, OS context switch overhead is often small (hundreds of clock cycles) relative to many common task periods (10's ms – sec's)
 - Some microcontrollers (e.g. ARM), have special instructions, and hardware mechanisms to help keep context switch time small.
- What if your set of processes is not schedulable?
 - Change deadlines in requirements
 - Reduce execution times of processes
 - Get a faster CPU

Fixed Priority Concern: Priority Inversion

- **Priority inversion**: low-priority process keeps high-priority process from running. Thus the low-priority task indirectly acts as though it is a high-priority process.
 - This can happen if a low-priority process has a lock (control) of a resource that a higher priority process needs.
 - Can cause deadlock
- Example from Textbook, chapter 6
 - Assume three processes, with P1 highest priority, P2 next highest, and P3 lowest priority.
 - 1) P3 takes control of resource A (e.g. a Network card)
 - 2) P2 then preempts P3, while P3 still has control of resource A
 - 3) P1 preempts P2, and runs until it needs resource A, then blocks
 - 4) P2 run to completion, since P1 is no longer ready to run
 - 5) P3 runs until it releases control of resource A
 - 6) Now P1 can stop blocking on resource A and run

Solving Priority Inversion

- **Priority Inheritance**: Have process inherit the priority of the highest process that may ever need a given System resource, while using System resource.
 - Assume three processes, with P1 highest priority, P2 next highest, and P3 lowest priority. Now use Priority Inheritance.
 - 1) P3 takes control of resource A (e.g. a Network card)
 - Since P1 may need resource A, give P3 the priority level of P1
 - 2) P2 becomes ready, but will not preempt P3
 - 3) P3 finishes with resource A, and has its priority set back to its original level.
 - 4) P2 can now preempt P3, and P2 runs until
 - 3) P1 preempts P2, and P1 runs to completion
 - 4) P2 runs to completion, assuming P1 does not restart
 - 5) P3 runs to completion, assuming P1 or P2 do not restart
 - Note: can still have deadlock occur.
- **Priority Ceilings** (summary of idea): Process PA can only enter a critical section of code (e.g. lock a resource), if no other higher priority process has locked a resource that Process PA may need.
 - Used with Priority Inheritance solves deadlock, but is a more complex protocol

Acknowledgments

- These slides are inspired in part by material developed and copyright by:
 - Marilyn Wolf (Georgia Tech)
 - Fred Kuhns (Washington University in St. Louis)
 - Steve Furber (University of Manchester)
 - Ed Lee (UC-Berkeley)