Uniprocessor Scheduling - II

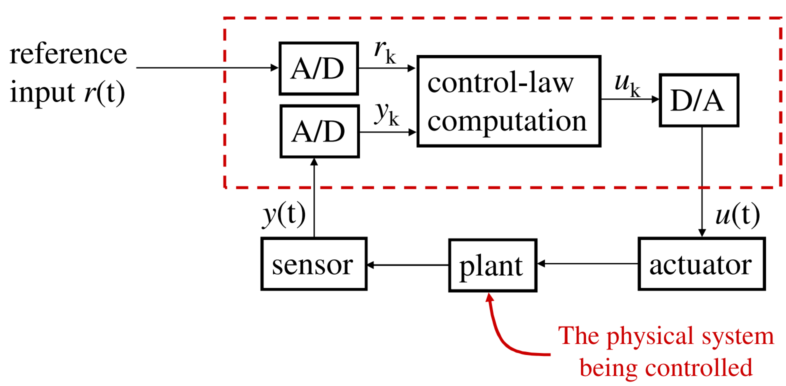
Sandeep D'souza

Lecture #4

Example Real-Time Applications

Many real-time systems are **control systems**.

Example 1: A simple one-sensor, one-actuator control system.



Simple Control System (cont'd)

Pseudo-code for this system:

set timer to interrupt periodically with period *T*;
at each timer interrupt, **do**
    do analog-to-digital conversion to get *y*;
    compute control output *u*;
    output *u* and do digital-to-analog conversion;
end do

T is called the **sampling period**. *T* is a key design choice. *T*ypical range for *T*: milliseconds to seconds.

Multi-rate Control Systems

More complicated control systems have multiple sensors and actuators and must support control loops of different rates.

Example 2: Helicopter flight controller.

Do the following in *each* 1/180-sec. cycle:
validate sensor data and select data source;
if failure, reconfigure the system

***Every sixth* cycle do:**
keyboard input and mode selection;
data normalization and coordinate transformation;
tracking reference update
control laws of the outer pitch-control loop;
control laws of the outer roll-control loop;
control laws of the outer yaw- and collective-control loop

Every *other* cycle do:
control laws of the inner pitch-control loop;
control laws of the inner roll- and collective-control loop

Compute the control laws of the inner yaw-control loop;

Output commands;

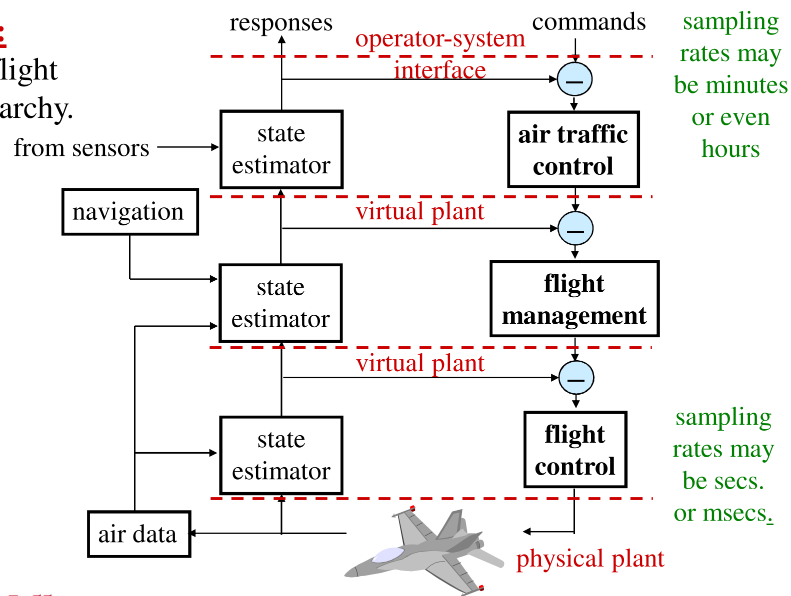
Carry out built-in test;

Wait until beginning of the next cycle

Note: Having only **harmonic** rates simplifies the system.

Hierarchical Control Systems

Example 3:
Air traffic-flight control hierarchy.

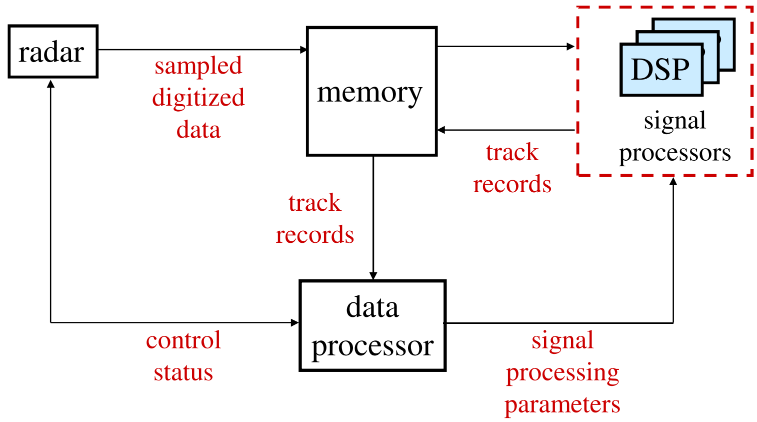


Signal-Processing Systems

Signal-processing systems transform data from one form to another.

- Examples:
  - Digital filtering.
  - Video and voice compression/decompression.
  - Radar signal processing.
- Response times range from a few milliseconds to a few seconds.

Example: Radar System



Other Real-Time Applications

- **Real-time databases.**
  - Transactions must complete by deadlines.
  - **Main dilemma:** Transaction scheduling algorithms and real-time scheduling algorithms often have conflicting goals.
  - Data may be subject to **absolute** and **relative temporal consistency** requirements.
- **Multimedia.**
  - Want to process audio and video frames at steady rates.
    - TV video rate is 30 frames/sec. HDTV is 60 frames/sec.
    - Telephone audio is 16 Kbits/sec. CD audio is 128 Kbits/sec.
  - **Other requirements:** Lip synchronization, low jitter, low end-to-end response times (if interactive).

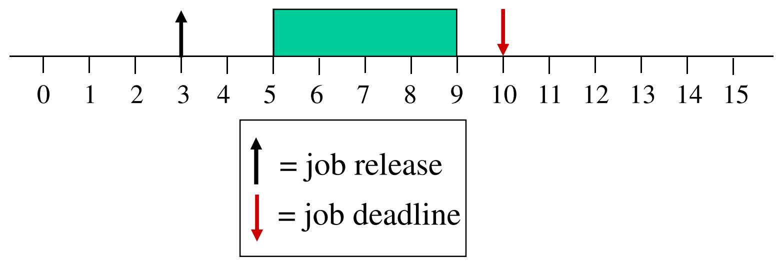
Are *All* Systems Real-Time Systems?

- **Question:** Is a payroll processing system a real-time system?
  - It has a time constraint: Print the pay checks (say) every two weeks.
- Perhaps it is a real-time system in a definitional sense, but it does *not* pay us to view it as such.
- We are interested in systems for which it is not *a priori* obvious how to meet timing constraints.
  - Wide variety of constraints
  - Really tight timing constraints
  - Different levels of criticality

Hard vs. *Soft* Real Time

- **Task:** A sequential piece of code.
- **Job:** Instance of a task.
- Jobs require **resources** to execute.
  - **Example resources:** CPU, network, disk, critical section.
  - We will simply call all hardware resources "processors".
- **Release time of a job:** The time instant the job becomes ready to execute.
- **Absolute Deadline of a job:** The time instant by which the job must complete execution.
- **Relative deadline of a job:** "Deadline – Release time".
- **Response time of a job:** "Completion time – Release time".

Example



- Job is released at time 3.
- Its (absolute) deadline is at time 10.
- Its relative deadline is 7.
- Its response time is 6.

Hard Real-Time Systems

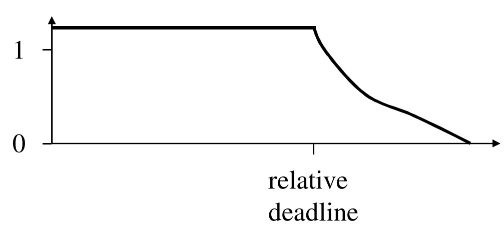
- A **hard deadline** *must* be met.
  - If *any* hard deadline is *ever* missed, then the system is **incorrect**.
  - Requires a means for **validating** that deadlines are met.
- **Hard real-time system:** A real-time system in which all deadlines are hard.
  - We mostly consider hard real-time systems in this course.
- **Examples:** Nuclear power plant control, flight control.

Soft Real-Time Systems

- A **soft deadline** may *occasionally* be missed.
  - **Question:** How to define "occasionally"?
- **Soft real-time system:** A real-time system in which some deadlines are soft.
- **Examples:** Telephone switches, multimedia applications.

Defining "Occasionally"

- **One Approach:** Use probabilistic requirements.
  - For example, 99% of deadlines will be met.
- **Another Approach:** Define a "usefulness" function for each job:



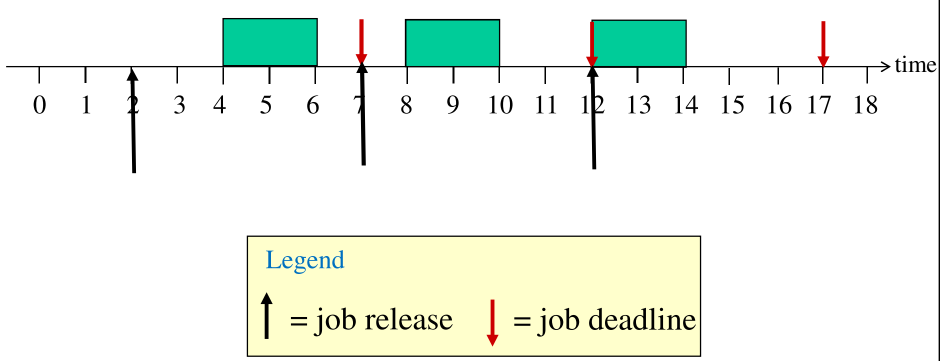
- **Note:** Validation is *much* trickier here.

Periodic, Sporadic, Aperiodic Tasks

- **Periodic task:**
  - We associate a **period T_i** (as in $1/f$) with each task τ_i.
  - T_i is the interval between job releases of a task τ_i.
- **Sporadic and Aperiodic tasks:** Released at arbitrary times.
  - **Sporadic:** Has a hard deadline.
  - **Aperiodic:** Has no deadline or a soft deadline.

Examples

A periodic task τ_i with $r_i = 2$, $T_i = 5$, $C_i = 2$, $D_i = 5$ may execute like this:



Classification of Scheduling Algorithms

All scheduling algorithms

- static scheduling (or offline, or clock-driven)
- dynamic scheduling (or online, or priority-driven)
  - fixed-priority scheduling
  - dynamic-priority scheduling

Real Time Systems and You

- Embedded real time systems enable us to:
  - manage the vast power generation and distribution networks,
  - control industrial processes for chemicals, fuel, medicine, and manufactured products,
  - control automobiles, ships, trains and airplanes,
  - conduct video conferencing over the Internet and interactive electronic commerce, and
  - send vehicles high into space and deep into the sea to explore new frontiers and to seek new knowledge.

Real-Time Systems

- Timing requirements
  - meeting deadlines
- Periodic and aperiodic tasks
- Shared resources
- Interrupts

What's Important in Real-Time

Metrics for real-time systems differ from that for time-sharing systems.

	Time-Sharing Systems	Real-Time Systems
Capacity	High throughput	Schedulability
Responsiveness	Fast average response	Ensured worst-case response
Overload	Fairness	Stability

- schedulability is the ability of tasks to meet all hard deadlines
- latency is the worst-case system response time to events
- stability in overload means the system meets critical deadlines even if all deadlines cannot be met

Real-Time Scheduling Policies

- CPU scheduling policy: a rule to select task to run next
  - cyclic executive
  - Rate-monotonic/deadline-monotonic
  - earliest deadline first
  - least laxity first
- Assume preemptive, priority scheduling of tasks
  - analyze effects of non-preemption later

Rate Monotonic Scheduling (RMS)

- Priorities of periodic tasks are based on their rates: the highest rate gets the highest priority.
- Theoretical basis
  - optimal fixed scheduling policy (when deadlines are at end of period)
  - analytic formulas to check schedulability
- Must distinguish between scheduling and analysis
  - Rate-monotonic scheduling forms the basis for rate-monotonic analysis
  - however, we consider later how to analyze systems in which rate-monotonic scheduling is *not* used
  - any scheduling approach may be used, but all real-time systems should be analyzed for timing

Rate Monotonic Analysis (RMA)

- Rate-monotonic analysis is a set of mathematical techniques for analyzing sets of real-time tasks.
- Basic theory applies only to independent, periodic tasks, but has been extended to address
  - priority inversion
  - task interactions
  - aperiodic tasks
- Focus is on RM**A**, not RM**S**

Why Are Deadlines Missed?

- For a given task, consider
  - **preemption**: time waiting for higher priority tasks
  - **execution**: time to do its own work
  - **blocking**: time delayed by lower priority tasks
- The task is schedulable if the sum of its preemption, execution, and blocking is less than its deadline.
- **Focus**: identify the biggest hits among the three and reduce, as needed, to achieve schedulability

Carnegie Mellon 18-648: Embedded Real-Time Systems Electrical & Computer ENGINEERING

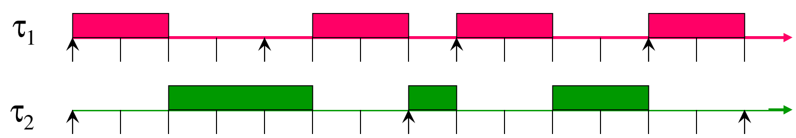
REAL-TIME SCHEDULING POLICIES

Scheduling Policies

- CPU scheduling policy: a rule to select task to run next
  - cyclic executive
  - Rate-monotonic/deadline-monotonic
  - earliest deadline first
  - least laxity first
- Assume preemptive, priority scheduling of tasks
  - analyze effects of non-preemption later

Earliest Deadline First (EDF) Scheduling

- Can be used to schedule periodic tasks
- Uses dynamic priorities and preemptive scheduling
  - Higher priority to task with earlier deadline
- **Example**: 2-task set where each task is (C, T=D) → {(2, 4), (3, 7)}



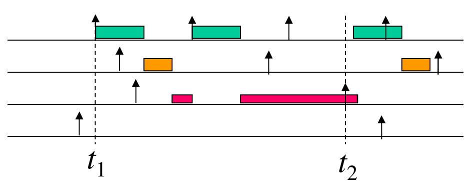
- Utilization U of task set {τ_i} for i = 1, …, n:

$$U_i = \frac{C_i}{T_i} \qquad U = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

EDF Schedulability Condition

Theorem: A task set is schedulable under EDF if and only if $U \le 1$.

Proof:



- Assume that "overflow" occurs at time t_2.
- Let t_1 be the latest time before t_2 such that
  - the processor is fully utilized in the interval $[t_1, t_2]$
  - only instances with deadlines before t_2 execute in $[t_1, t_2]$
- If such a t_1 cannot be found, then set $t_1 = 0$.
- Let C_d be the computational demand in $[t_1, t_2]$

$$C_d = \sum_{r_i \ge t_1, d_i \le t_2} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor * c_i \le \sum_{i=1}^{n} \frac{t_2 - t_1}{T_i} * c_i = (t_2 - t_1)U$$

- But an overflow implies that $C_d > (t_2 - t_1)$: a contradiction if $U \le 1$.

Points to Note

- If deadlines are shorter than periods, the necessary condition for schedulability under EDF is an open problem.
- If $U > 1$, which task will first miss its deadline is unpredictable.

Rate-Monotonic Scheduling (RMS)

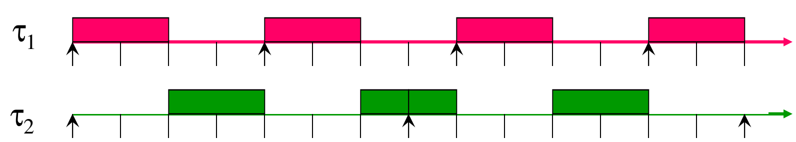
- The priorities of periodic tasks are based on their rates: a higher rate gets a higher priority.
- Theoretical basis
  - optimal fixed-priority scheduling policy (when deadlines are at the end of period)
  - analytic formulas to check schedulability
- Must distinguish between scheduling and analysis
  - Rate-Monotonic Scheduling (RMS) forms the basis for Rate-Monotonic Analysis (RMA)
  - However, we will consider later how to analyze systems in which rate-monotonic scheduling is *not* used
  - Any scheduling approach may be used, but all real-time systems should be analyzed for timing

Rate Monotonic Analysis (RMA)

- Rate-monotonic analysis is a set of mathematical techniques for analyzing sets of real-time tasksets.
- The original RMS theory applied only to independent, periodic tasks, but has been extended to address
  - priority inversion
  - task interactions
  - aperiodic tasks
- Focus is on RM**A**, not RM**S**

Rate-Monotonic Scheduling (RMS)

- Higher (fixed) priority to higher frequency task
  - i.e. higher priority to task with shorter period
- **Example**: 2-task set where each task is (C, T=D) → {(2, 4), (3, 7)}



Liu and Layland proved that RMS leads to a feasible schedule if $U \le n\,(2^{1/n} - 1)$

- If $C_2 = 3.1$, then C_2 will miss its deadline although the utilization is (2/4)+(3.1/7) = 0.5+0.443=0.943, which is < 1.
- The above bound is sufficient for schedulability but not necessary
- RMS is an <u>optimum</u> fixed-priority assignment algorithm
  - if a taskset cannot be scheduled by RMS, <u>all</u> other fixed-priority schedules will also be unschedulable.

More on the RMS Bound

- The RMS bound is a sufficient condition, not a necessary condition
- Occurs only under mathematically extreme conditions
- In practice
  - Many tasksets are **harmonic**
    - In a harmonic taskset, every period is an integral multiple (or sub-multiple) of all other periods in the task set
  - For harmonic tasksets, RMS is schedulable if $U \leq 1$.
  - Or for nearly harmonic tasksets
    - The schedulable utilization is about 0.92
- Mathematically, if *C*'s and *T*'s are chosen randomly, the average schedulable task set utilization is 88%.
- Remaining utilization can still be used for other purposes
  - Background tasks, testing, etc.

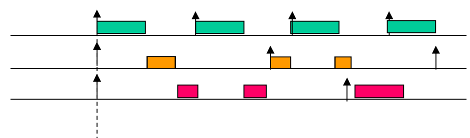
n	$n(2^{1/n} - 1)$
1	1
2	0.828
3	0.780
4	0.757
5	0.743
6	0.735
7	0.729
8	0.724
9	0.721
10	0.718
…	…
∞	$\ln 2 = 0.693$

The Derivation of the "RMS" Least Upper Bound

- **Step 1**: Find the worst-case relative phasing between tasks
- **Step 2**:
  - Assume that the ratio of the largest to the smallest period (T_n / T_1) is no more than 2.
  - For *n* tasks, find U_{lub} such that if a task set has $U \leq U_{lub}$, then it can be feasibly scheduled by RMS.
- **Step 3**: Generalize the result for an arbitrary T_n / T_1 ratio.

The Worst-Case Relative Phasing

- **Critical instant**: the instant or relative phasing at which a task arrival encounters its worst-case response time
- **Critical zone**: the duration between the critical instant of a task instance and its completion



- For a periodic task set $\{(C, T)\}$, the critical zone for a task τ occurs when τ arrives simultaneously with *all* other higher priority tasks.
  - So, all tasks arrive together
- A taskset is feasible if the first instance of a task arrives along with all other tasks and completes by its deadline

The Least Upper Schedulable Bound for 2 Tasks

Goal: Among all the feasibly scheduled task sets $\{\tau_1, \tau_2\}$, we want to find for any given combination of periods $\{T_1\ T_2\}$, the maximum feasible schedulable utilization U_{ub}.

- Find the minimum value of U_{ub}, U_{lub}, for all possible combinations of T_i.

Method:

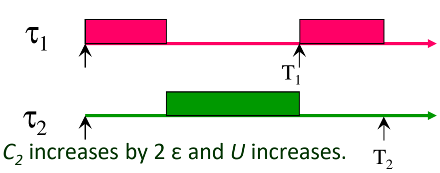
- Fix T_2 and consider all possible values of T_1, C_1 and C_2 (relative to T_2).
- For any T_1, assuming a fully utilized processor, the minimum U occurs when $C_1 = T_2 - T_1$

  Thus, $C_2 = T_1 - C_1$

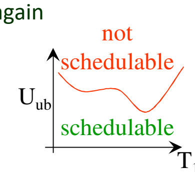
  $U = U_1 + U_2$

  $= (T_2 - T_1) / T_1 + (T_1 - T_2 + T_1) / T_2$

  $= T_2 / T_1 + 2\, T_1 / T_2 - 2$



- If $C_1 = T_2 - T_1 - \varepsilon$, then, for full utilization, C_2 increases by $2\,\varepsilon$ and U increases.
- If $C_1 = T_2 - T_1 + \varepsilon$, then, for full utilization, C_2 decreases by ε and U again increases (remember that $T_2 / T_1 < 2$).
- Next, differentiate U w.r.t. T_1 and equate the result to zero
  - Obtain $\dfrac{T_2}{T_1} = \sqrt{2}$



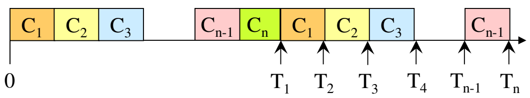
- Hence, among all possible values of T_1, the minimum utilization is

$$2(\sqrt{2} - 1)$$

Least Upper Schedulable Bound for *n* Tasks

- For given values of $T_1, \ldots, T_n$, the upper bound, U_{ub}, on the utilization of feasible sets is obtained when

$C_1 = T_2 - T_1$
$C_2 = T_3 - T_2$
….
$C_{n-1} = T_n - T_{n-1}$
$C_n = T_1 - (C_1 + \ldots + C_{n-1}) = 2T_1 - T_n$



- For the above values, $$U = \sum_{i=1}^{n-1} R_i + \frac{2}{R_1 R_2 \ldots R_{n-1}} - n$$

  where $R_i = T_{i+1} / T_i$ and $R_1 \ldots R_{n-1} = T_n / T_1$
- Differentiate U w.r.t. $R_1, \ldots, R_{n-1}$ and equate the result to zero to obtain

  $R_1 = \ldots = R_{n-1} = 2^{1/n}$
- Hence, among all the T's and C's, the minimum utilization for feasibility is given by $U = n(2^{1/n} - 1)$

Carnegie Mellon 18-648: Embedded Real-Time Systems Electrical & Computer ENGINEERING

Response Time (RT) Test

- Also, referred to as "Exact Schedulability Test" or "Completion Time Test"
- Can be used for computing response times with *any* fixed-priority preemptive scheduling scheme
- Let a_k^i **= the worst-case response time of task** τ_i. a_k^i of task τ_i may be computed by the following iterative formula:

$$a_{k+1}^i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{a_k^i}{T_j} \right\rceil C_j \qquad \text{where } a_0^i = \sum_{j=1}^{i} C_j$$

- Test terminates when $a_{k+1}^i = a_k^i$
- Task *i* is schedulable if its response time is before its deadline: $a_k^i \le D_i$
  - Stop test once current iteration yields a value of a_k^i beyond the deadline (else, you may never terminate).
- This test determines the schedulability of only **task** τ_i
- Repeat for other tasks as needed**:** *i* will change with the task

The 'square bracketish' thingies represent the 'integer ceiling' function, NOT brackets

Carnegie Mellon 18-648: Embedded Real-Time Systems Electrical & Computer ENGINEERING

Example: Testing for Schedulability

	C_i	T_i	U_i
Task τ_1:	40	100	0.4
Task τ_2:	40	150	0.267
Task τ_3:	100	350	0.286

- Utilization of first two tasks: 0.667 < LUB(2) = 0.828
  - The first two tasks are schedulable by UB test
- Utilization of all three tasks: 0.953 > LUB(3) = 0.779
  - UB test is inconclusive for the third task
  - Need to apply RT test

Applying RT Test

Use RT test to determine if τ_3 meets its first deadline: $\boxed{i = 3}$

$$a_0^3 = \sum_{j=1}^{3} C_j = C_1 + C_2 + C_3 = 40 + 40 + 100 = 180$$

$$a_1^3 = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{a_0^3}{T_j} \right\rceil C_j = C_3 + \sum_{j=1}^{2} \left\lceil \frac{a_0^3}{T_j} \right\rceil C_j$$

$$= 100 + \left\lceil \frac{180}{100} \right\rceil (40) + \left\lceil \frac{180}{150} \right\rceil (40) = 100 + 80 + 80 = 260$$

Applying the RT Test (contd.)

$$a_2^3 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{a_1^3}{T_j} \right\rceil C_j = 100 + \left\lceil \frac{260}{100} \right\rceil (40) + \left\lceil \frac{260}{150} \right\rceil (40) = 300$$

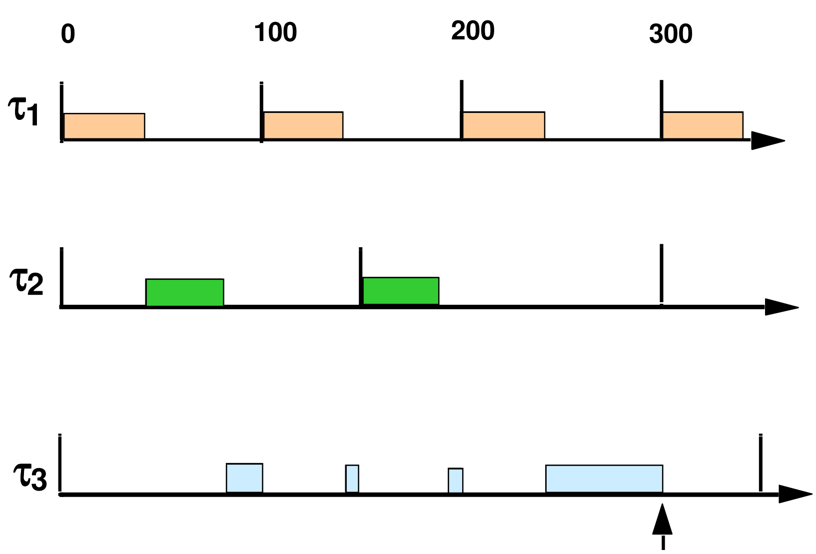
$$a_3^3 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{a_2^3}{T_j} \right\rceil C_j = 100 + \left\lceil \frac{300}{100} \right\rceil (40) + \left\lceil \frac{300}{150} \right\rceil (40) = 300$$

$a_3 = a_2 = 300$ **Done!**

Task τ_3 is schedulable using the RT test

$$\boxed{a_3 = 300 < D = T = 350}$$

Timeline for Example



τ_3 completes its work at t = 300

Exercise: Applying RT Test

Task τ_1: $C_1 = 1$ $T_1 = 4$
Task τ_2: $C_2 = 2$ $T_2 = 6$
Task τ_3: $C_3 = 2$ $T_3 = 10$

a) Apply the LUB test
b) Draw timeline
c) Apply RT test

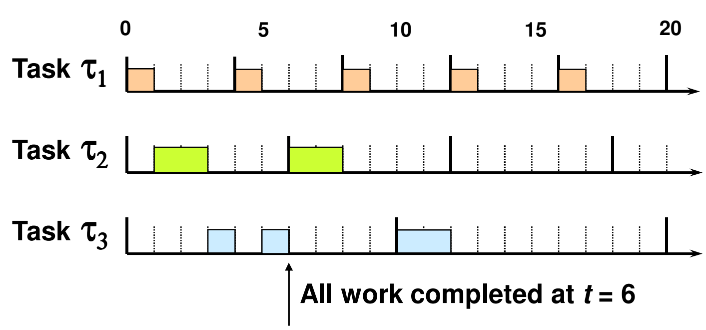
Solution: Applying RT Test

a) UB test

τ_1 and τ_2 OK -- no change from previous exercise

.25 + .34 + .20 = .79 > .779 ==> Test inconclusive for τ_3

b) RT test and timeline



Solution: Applying RT Test (cont.)

c) RT test

$$a_0^3 = \sum_{j=1}^{3} C_j = C_1 + C_2 + C_3 = 1 + 2 + 2 = 5$$

$$a_1^3 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{a_0^3}{T_j} \right\rceil C_j = 2 + \left\lceil \frac{5}{4} \right\rceil 1 + \left\lceil \frac{5}{6} \right\rceil 2 = 2 + 2 + 2 = 6$$

$$a_2^3 = C_3 + \sum_{j=1}^{2} \left\lceil \frac{a_1^3}{T_j} \right\rceil C_j = 2 + \left\lceil \frac{6}{4} \right\rceil 1 + \left\lceil \frac{6}{6} \right\rceil 2 = 2 + 2 + 2 = 6$$

Done

Why Are Deadlines Missed?

- For a given task, consider
  - **preemption**: time waiting for higher priority tasks
  - **execution**: time to do its own work
  - **blocking**: time delayed by lower priority tasks
- The task is schedulable if the sum of its preemption, execution, and blocking is less than (or equal to) its deadline.
- **Focus**: identify the biggest hits among the three and reduce, as needed, to achieve schedulability

Summary

- Dynamic-priority EDF scheduling can schedule periodic task sets with total utilization ≤ 100%
- Under fixed-priority Rate-Monotonic Scheduling (RMS),
  - Pathological tasksets with about 70% utilization could miss deadlines
    - Nearly impossible to find in practice
  - Least-Upper Bound tests are simple but conservative
    - Also referred to as a “utilization bound”
  - The Response time test is more exact but needs more calculations.
    - Easily automated
  - Harmonic tasksets can be scheduled up to 100%
  - Nearly harmonic tasksets (found in practice very often) are scheduled upwards of 92%
  - Randomly chosen tasksets have an average schedulable utilization of 88%
- RMS priority assignments are used widely in practice.