Schedulability in Real Time Systems

Paritosh Pandya

CS684, IIT Bombay
March 2022

Real time, Embedded Systems

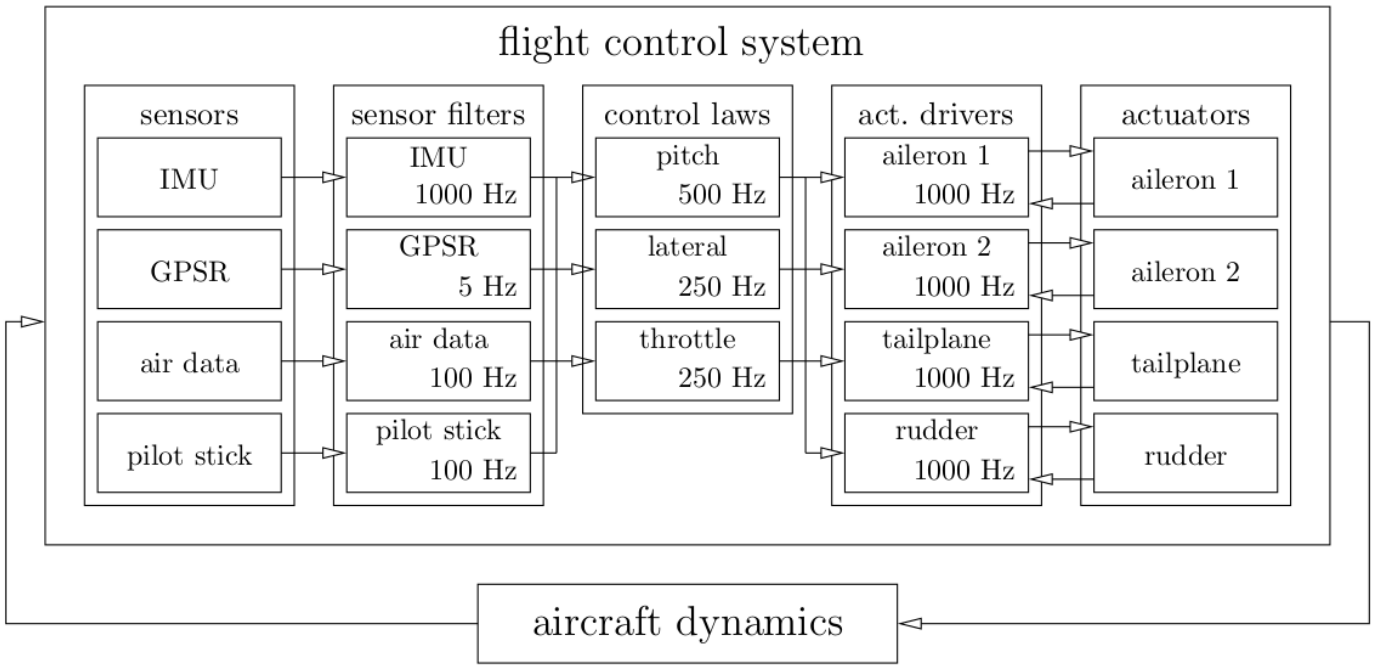
An assembly of electro-mechanical, optical, chemical components with sensors and actuators, connected to onboard computer.

- Program is typically orgainzed as a set of repeating tasks.
- A periodic task typically has the structure:

```
repeat every 10 ms
  { sense input;
    Compute;
    Actuate output;
  }
```

- (latancy) There are real-time requirements on delays between input and output.

Example: Flight Control System



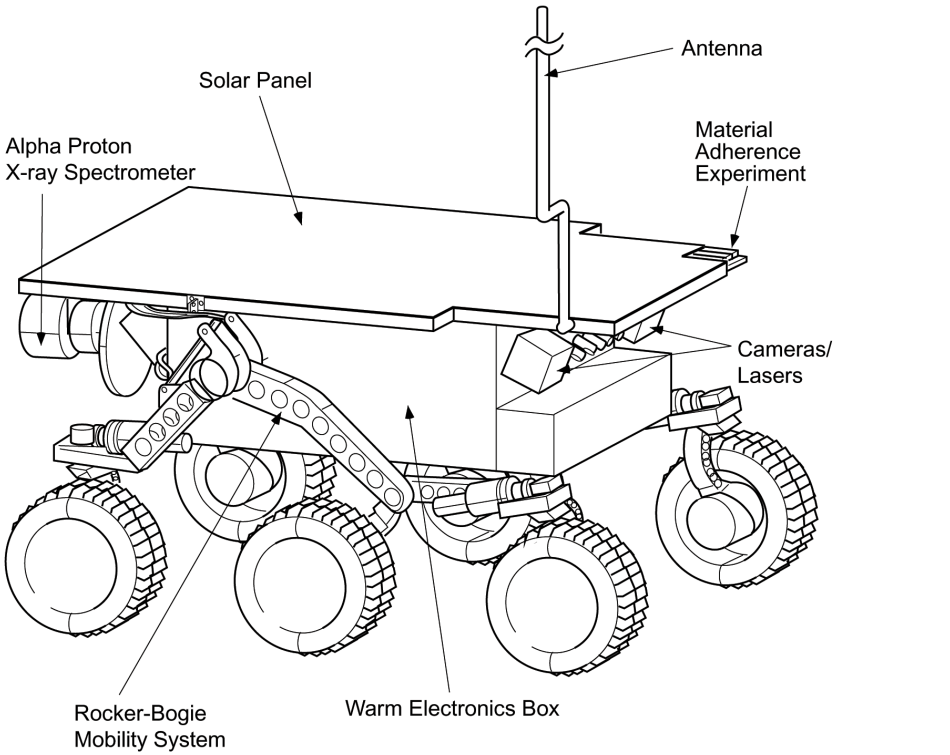
- A set of periodic processes.
- Interaction via shared memory.
- Execute on single microcontroller by sharing CPU
- Scheduling important to meet latency.

Schedulability Analysis

Given set of tasks what kind of scheduling policy will allow all tasks to meet their deadlines (latency requirements) ?

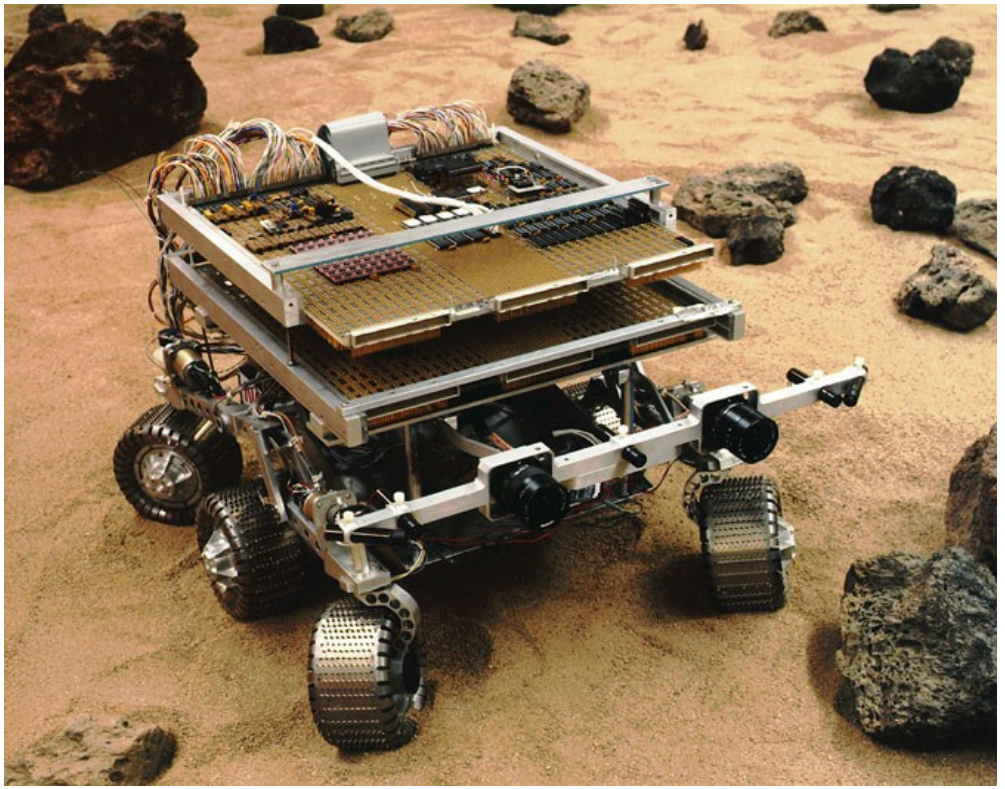
Example: Mars Pathfinder (1)

NASA Mars Mission 4 July, 1997.

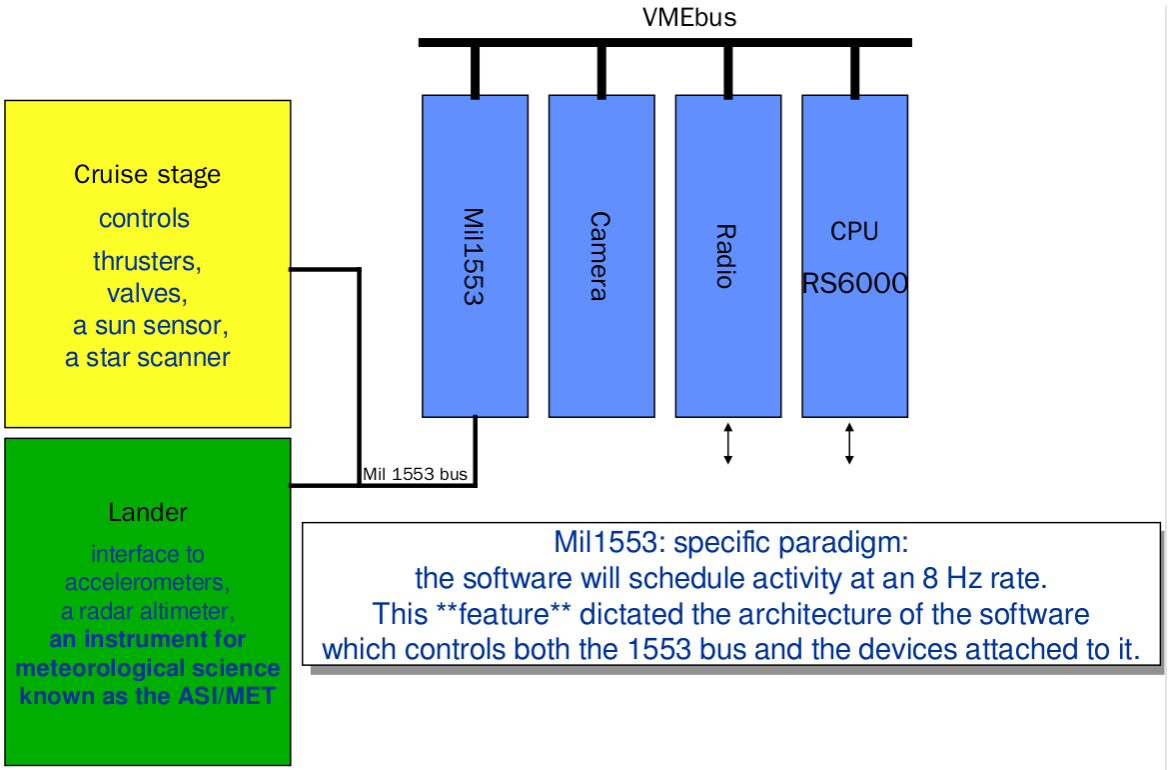


(Images courtsey NASA)

Example: Mars Pathfinder (2)



Example: Mars Rover (3)



Information Bus: A shared data structure for devices and processes of Lander and Cruise Control.

Example: Mars Pathfinder (4)

Pathfinder used VxWorks RTOS

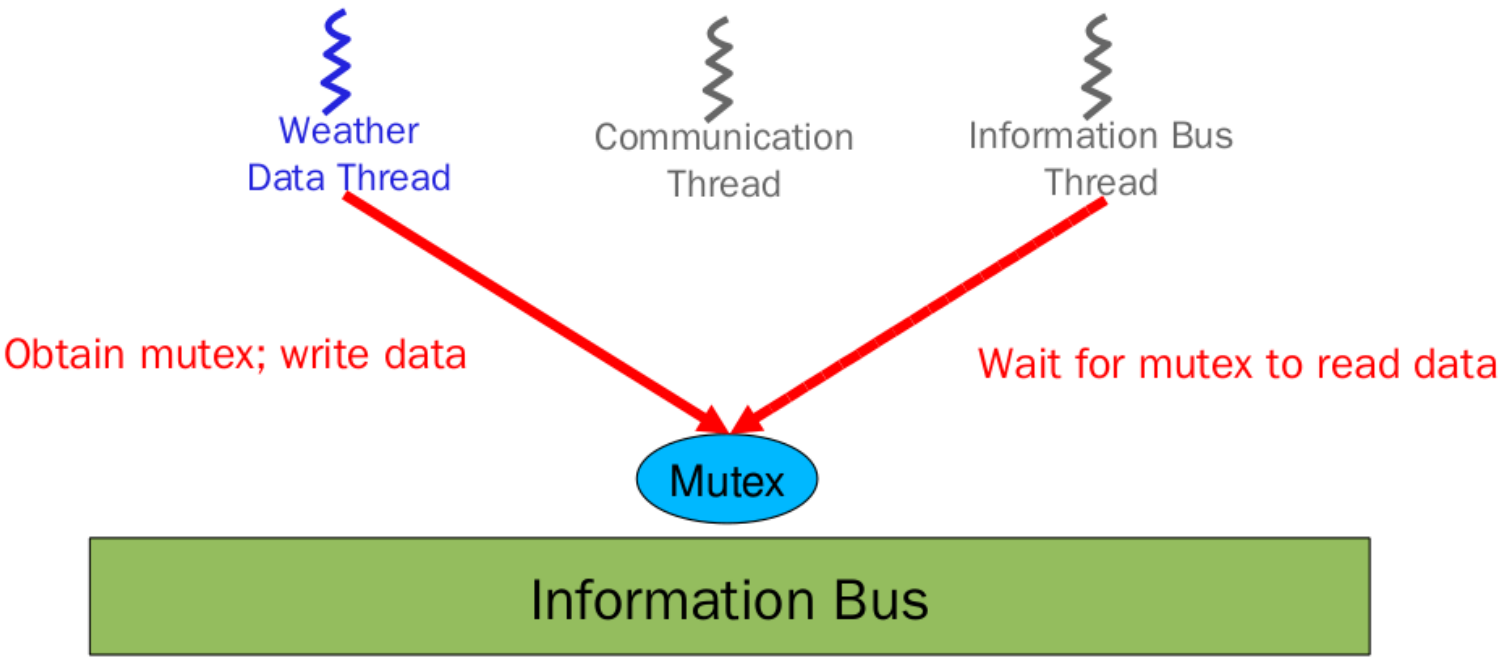
- Threads for the 1553 bus for data collection, scheduled on every 1/8th sec cycle.
- 3 periodic tasks
  - Task 1 – Information Bus Thread: Bus Manager

    high frequency, high priority
  - Task 2 – Communication Thread

    medium frequency / priority, high execution time
  - Task 3 – Weather Thread: Geological Data Gatherer

    low frequency, low priority
- Each checks if the other executed and completed in the previous cycle
  - If the check fails, this is a violation of a hard real-time guarantee and the system is reset

Example: Mars Pathfinder (5)



Mars Pathfinder Bug

NASA sent mars pathfinder Sojourn on 4 July 1997.

- Stopped working after 87 sol due to software error.

Mars Pathfinder Bug

NASA sent mars pathfinder Sojourn on 4 July 1997.

- Stopped working after 87 sol due to software error.
- Diagnosed as a rare schedulability problem called Priority Inversion.

Mars Pathfinder Bug

NASA sent mars pathfinder Sojourn on 4 July 1997.

- Stopped working after 87 sol due to software error.
- Diagnosed as a rare schedulability problem called Priority Inversion.
- Fixed by reloading patched code.

Mars Pathfinder Bug

NASA sent mars pathfinder Sojourn on 4 July 1997.

- Stopped working after 87 sol due to software error.
- Diagnosed as a rare schedulability problem called Priority Inversion.
- Fixed by reloading patched code.

Schedulability Analysis

Subsequent schedulability analysis found the problem in original design. It proved the correctness of modified design.

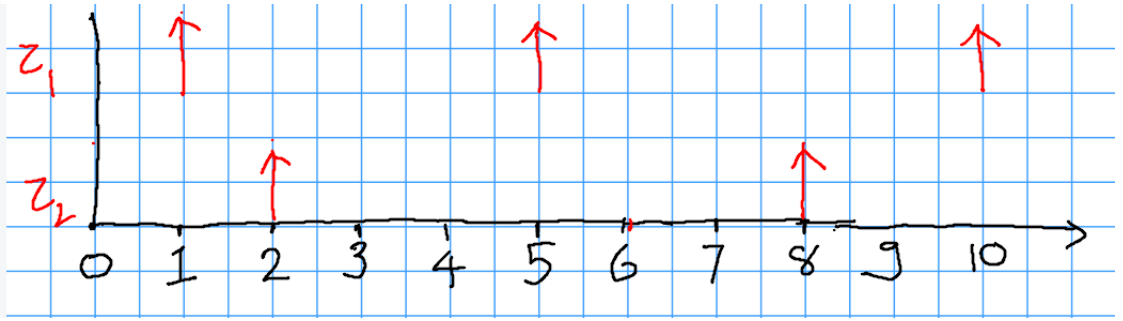
IEEE TCRTS Test Of Time Awards 2020

Instituted by IEEE in 2020 for papers having lasting impact on the field.

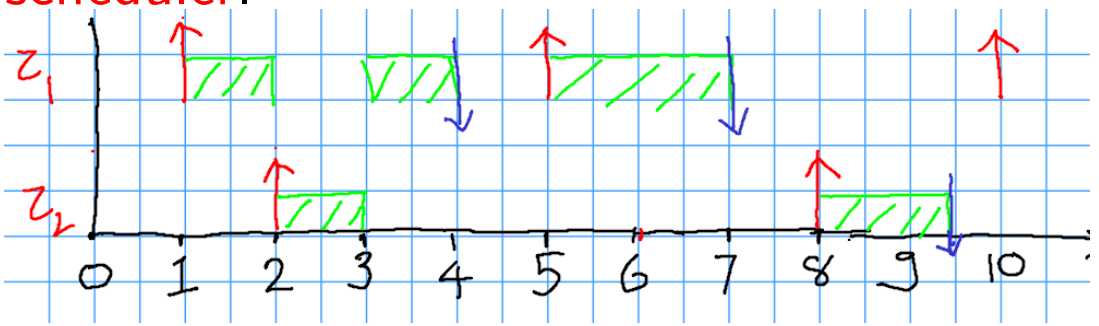
- Chung. L. Liu and James W. Layland
  Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment
  Journal of the Association for Computing Machinery, Vol. 20, No. 1, pp. 46-61, January 1973.
  For pioneering the way towards a formal analysis of real-time scheduling algorithms.
- Mathai Joseph and Paritosh Pandya
  Finding Response Times in a Real-Time System
  The Computer Journal, Vol. 29, Issue 5, pp. 390–395, 1986.
  For first proposing an exact method for computing worst-case response times under fixed priority pre-emptive scheduling.
- John A. Stankovic
  Misconceptions about Real Time Computing: A Serious Problem for Next Generation Systems
  IEEE Computer, Vol. 21, pp. 10-19, October 1988.
  For clearly highlighting the unique characteristics of real-time computing and motivating research in this field.
- Lui Sha, Raj Rajkumar and John P. Lehoczky
  Priority Inheritance Protocols: an Approach to Real-Time Synchronization
  IEEE Transactions on Computers, Vol. 39, No. 9, pp. 1175-1185, September 1990.
  For introducing the now-standard methods for controlling priority inversion on uniprocessors and their impact on the Mars Pathfinder mission.

Framework for Schedulability

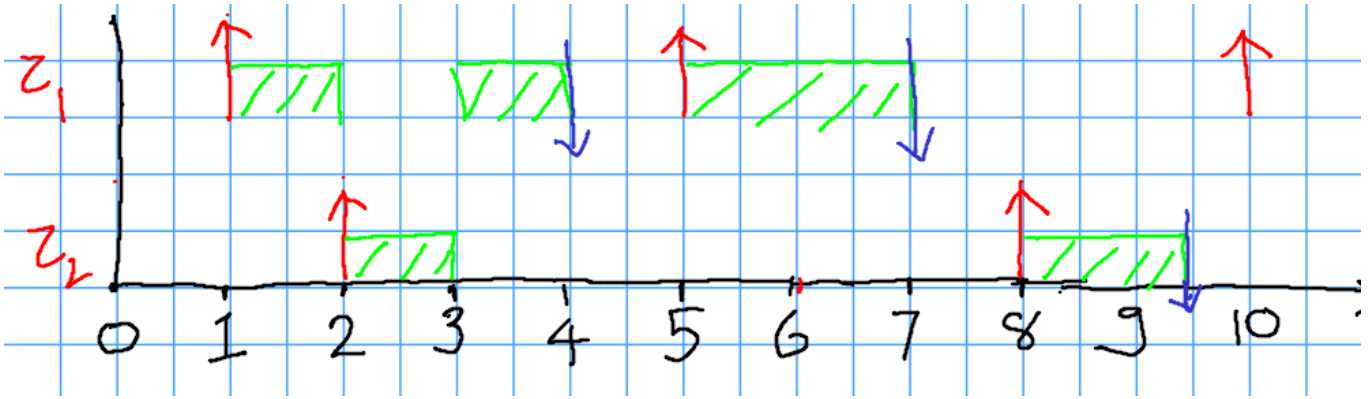
- A set of repeating tasks $\tau_1, \ldots, \tau_n$
- Arrival Pattern $\sigma = (\Gamma_i, \Theta_i)$



- $\Gamma_i(j)$ gives time of arrival of jth instance of task i.
  $\Theta_i(j)$ gives cpu time needed to execute jth instance of task i.
- Tasks are executed on Single CPU under the control of a scheduler.



Framework for Schedulability (2)



$pr(\tau_2) > pr(\tau_1)$

- Preemptive scheduling versus Non-preemptive scheduling.
- Priority Based Preemptive Scheduling Each process τ_i has a unique priorty $pr(\tau_i)$. Processes can be ordered by their priority.
- For a taskset $\tau_1, \ldots, \tau_n$, arrival pattern blue $\sigma = (\Gamma_i, \Theta_i)$ and priority assignment $pr(\tau_i)$ there is unique execution diagram.
- Response time (local) $RTL_i(j)$ is the time between release and completion of jth instance of task i.
  Example: $RTL_1(1) = 3$, $RTL_2(1) = 1$, and $RTL_2(1) = 2$.
- Deadline D_i maximum permitted response time.
  Execution meets deadline D_i if $\forall i, j.\ \ RTL_i(j) \leq D_i$.

Sporadic Tasks

A set of sporadic tasks $\tau_1, \ldots, \tau_n$.

$\tau_i = (T_i, C_i, D_i)$ with $D_i \leq T_i$.

- (Period T_i) Each task τ_i is repeatedly invoked with a minimum period of T_i.
  $\Gamma_i(j+1) - \Gamma_i(j) \geq T_i$ for all i, j.
- (Load C_i) Each invocation needs at most C_i seconds processor time.
  $\Theta_i(j) \leq C_i$ for all i, j.
- (Deadline D_i) Each invocation must finish within D_i seconds of its arrival.

Worst Case Response time RT_i under priority assignment pr

Let Σ be set of arrival patterns satisfying sporadic constraints.

$$RT_i = \max_{\sigma \in \Sigma} \max_j RTL_i(j)$$

Thus worst case response time RT_i is the maximum of $RTL_i(j)$ over all instances and all permitted arrival patterns Σ.

Priority assignment pr is feasible if $RT_i \leq D_i$ for all i

Hard Real Time Systems [Liu and Layland 1973]

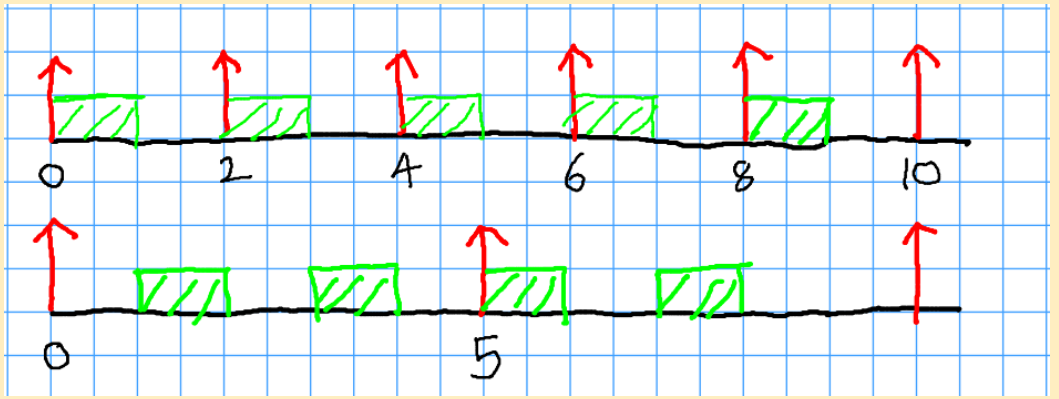
- A set of sporadic tasks $\tau_1, \ldots, \tau_n$ with $\tau_i = (T_i, C_i, D_i)$
- (Period T_i) Each task τ_i is repeatedly invoked at a minimum period of T_i.
  (Load C_i) Each invocation needs atmost C_i seconds processor time.
- (Deadline D_i) Each invocation must finish within D_i seconds of its arrival.
- Tasks are independant (no synchronization).
- Tasks execute on a single processor. CPU is shared between tasks.
- Priority based pre-emptive scheduling:
  Tasks are assigned unique priorities.
  Invocation of higher priority task switches processor to it from currently executing lower priority task.

A priorty assignment is feasible if for all possible task arrival patterns all deadlines are met. Taskset is feasible if there exists a feasible priority assignment.

Hard Real-time System Example

Task set $\tau_1 = (2, 1, 2)$ and $\tau_2 = (5, 2, 4)$

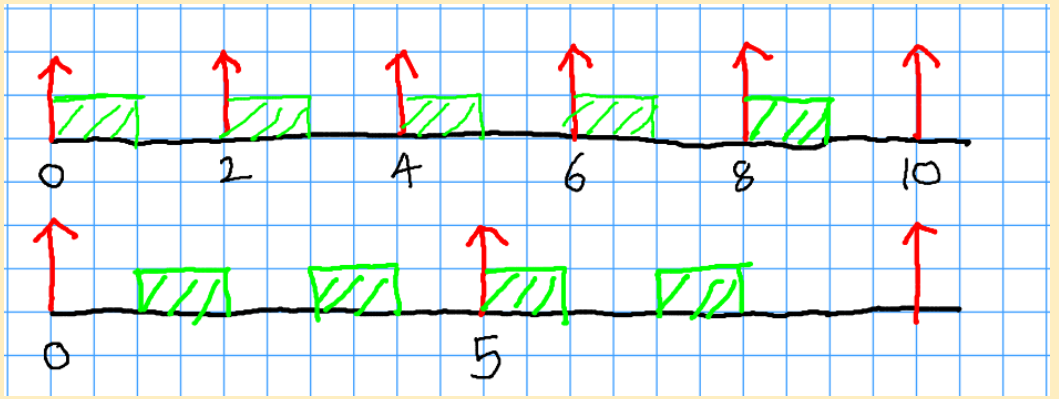
Execution with $pr(\tau_1) > pr(\tau_2)$



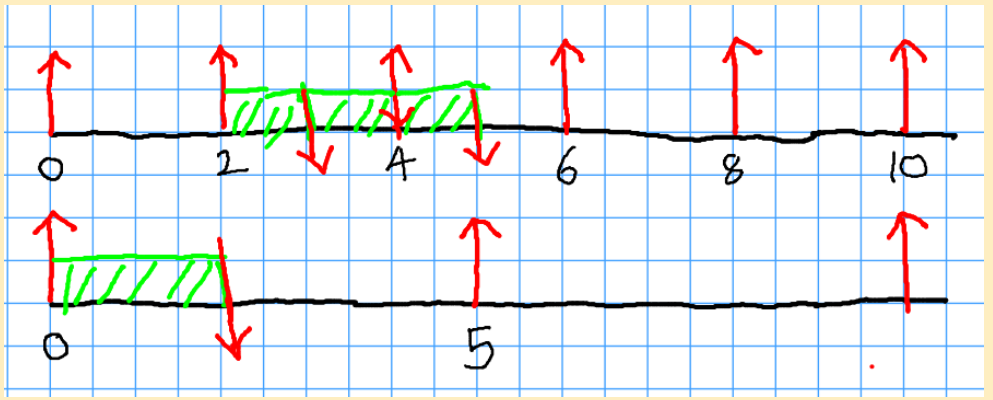
Hard Real-time System Example

Task set $\tau_1 = (2, 1, 2)$ and $\tau_2 = (5, 2, 4)$

Execution with $pr(\tau_1) > pr(\tau_2)$



Execution with $pr(\tau_2) > pr(\tau_1)$



Analysis of Hard Real Time Systems

[Liu and Layland 1973]
For a given sporadic task set

Feasibility

Given a priority assignment *pr*, how to check feasibility (i.e. deadlines are always met under all permitted task arrival patterns)?

Priority Assignment

How to assign priorities to the tasks to ensure feasibility? How to compute *pr* which is feasible?

Critical Instance: Planning for the worst

Given a taskset $\tau_1, \ldots, \tau_n$ with $\tau_i = (T_i, C_i, D_i)$, the critical instance is the unique arrival pattern $\sigma = (\Gamma_i, \Theta_i)$ where

- All tasks are invoked simultaneously at time = 0. Thus, $\Gamma_i(1) = 0$
- All tasks always arrive exactly after period T_i. Thus, $\Gamma_i(j+1) - \Gamma_i(j) = T_i$ for all i, j.
- Each task invocation takes maximum permitted load C_i. Thus, $\Theta_i(j) = C_i$ for all i, j.

Under a given priority assignment, the critical instance has unique execution.

Critical Instance: Planning for the worst

Given a taskset $\tau_1, \ldots, \tau_n$ with $\tau_i = (T_i, C_i, D_i)$, the critical instance is the unique arrival pattern $\sigma = (\Gamma_i, \Theta_i)$ where

- All tasks are invoked simultaneously at time $= 0$. Thus, $\Gamma_i(1) = 0$
- All tasks always arrive exactly after period T_i. Thus, $\Gamma_i(j+1) - \Gamma_i(j) = T_i$ for all i, j.
- Each task invocation takes maximum permitted load C_i. Thus, $\Theta_i(j) = C_i$ for all i, j.

Under a given priority assignment, the critical instance has unique execution.

Theorem (Liu, Layland 73)

If critical instance gives feasible execution, then the priority assignment is feasible.

Naive Feasibility Test

Given sporadic task set $\tau_1, \ldots, \tau_n$ with $\tau_i = (T_i, C_i, D_i)$, the hyper-period $HP = lcm(T_1, \ldots, T_n)$.

Given priority assignment pr to check if it is feasible,

- Observation: Execution of the critical instance under pr for the interval $[0 : HP)$ repeats without any change.
- Simulate the execution of critical instance only upto HP and compute observed worst case reponse times for each task.
- If each of these $RT_i \leq D_i$ then priority assingment is feasible.

Difficulty Hyperperiod can grow exponentially with number of tasks and hence simulation is often not practicable.

Static Priority Assignment Schemes (scheduling policies)

Rate Monotonic Scheduling

Assign priorities in the order of rate (inverse of period). Shortest period gets highest priority.

Theorem (Liu, Layland 73)

For tasksets where $T_i = D_i$ for all i, rate monotonic scheduling is optimal. If any arbitrary priority assignment is feasible then so is rate monotonic assignment.

Deadline Monotonic Scheduling

Assign priorities in order of inverse of deadlines. Shortest deadline gets highest priority.

Theorem (Leung, Whitehead, 1982)

For tasksets with $D_i \leq T_i$ forall i, Deadline monotonic scheduling is optimal. If any arbitrary fixed priority assignment is feasible (meets deadlines) then so is deadline monotonic priority assignment.

Example: Rate and Deadline Monotonic Priority Assignments

Task	*T*	*C*	*D*
τ_1	10	1	3
τ_2	5	1	5
τ_3	6	2	4

- Rate monotonic Priorities: $\tau_2 > \tau_3 > \tau_1$.
  Infeasible by naive test as it violates deadline.
- Deadline monotonic priorities: $\tau_1 > \tau_3 > \tau_2$
  Feasible by Naive test.

Fesibility Checking

[Liu and Layland 73]

Utilization

CPU Utilization by task i is $U_i = C_i/T_i$
Total Untilization $U = C_1/T_1 + C_2/T_2 + \ldots + C_n/T_n$
Total utilization gives the fraction of time the CPU is kept busy.

Necessary condition

$$U \leq 1$$

Necessary but not sufficient.

Sufficient Condition: For Tasksets with $D_i = T_i$ for all i.

$$U \leq B(n) \text{ where } B(n) = n \times (2^{1/n} - 1)$$

$B(n)$ has limit $\ln(2)$ as $n \to \infty$.
Sufficent but not necessary.

Utilization Bound Table

B(1)=1.0	B(4)=0.756	B(7)=0.728
B(2)=0.828	B(5)=0.743	B(8)=0.724
B(3)=0.779	B(6)=0.734	$U(\infty)=0.693$

Note that $U(\infty)=0.693$!

Examples: Utilization based feasibility

- $(U > 1)$ Infeasible: Taskset $(12, 8),\ (6, 3)$.
- $(U < ln(2))$ Feasible: Taskset $(12, 2),\ (6, 1)$.
- $(U = 1)$ Inconclusive: Taskset $(12, 4), (6, 4)$. Consider naive test with rate monotonic assignment.
- Taskset $(100, 20),\ (150, 40),\ (350, 10)$.

Examples: Utilization based feasibility

$4/12 + 4/6 = 1/3 + 2/3 = 1$

- $(U > 1)$ Infeasible: Taskset $(12, 8),\ (6, 3)$. $U = 8/12 + 3/6 > 1$
- $(U < ln(2))$ Feasible: Taskset $(12, 2),\ (6, 1)$.
- $(U = 1)$ Inconclusive: Taskset $(12, 4), (6, 4)$. Consider naive test with rate monotonic assignment.
- Taskset $(100, 20),\ (150, 40),$ ~~$(350, 10)$~~. $3 + 10/350$
  $U = 20/100 + 40/150 = 0.753 > ln(2)$.
  But, $B(3) = 3 * (2^{1/3} - 1) = 0.779$. Hence, $U < B(3)$.
  Feasible by rate monotonic.

Exact test for feasibility: The Response Time Approach

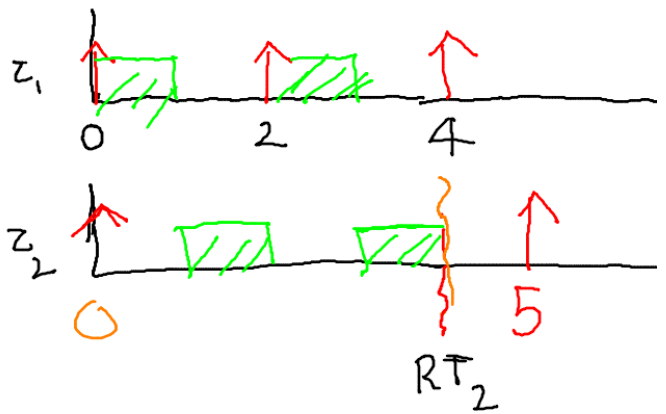
[Joseph, Pandya 86]
For a given priority assignment, let RT_i denote the worst case response time of task τ_i.

Equational characterization of RT_i

Let $hp(i)$ denote the set of tasks with priority higher than i.

$$RT_i = C_i + \Sigma_{j \in hp(i)} (\lceil RT_i / T_j \rceil \times C_j) \quad (1)$$

Task set $\tau_1 = (2, 1, 2)$ and
$\tau_2 = (5, 2, 4)$



Exact test for feasibility: The Response Time Approach

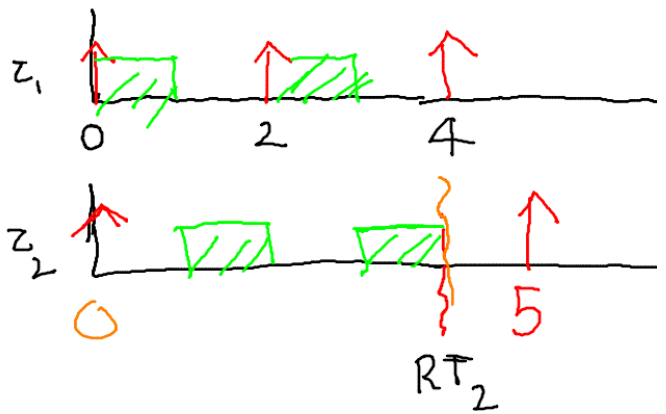
[Joseph, Pandya 86]
For a given priority assignment, let RT_i denote the worst case response time of task τ_i.

Equational characterization of RT_i

Let $hp(i)$ denote the set of tasks with priority higher than i.

$$RT_i = C_i + \Sigma_{j \in hp(i)} (\lceil RT_i / T_j \rceil \times C_j) \quad (1)$$

Task set $\tau_1 = (2, 1, 2)$ and
$\tau_2 = (5, 2, 4)$



$$RT_1 = C_1 + 0$$

Exact test for feasibility: The Response Time Approach

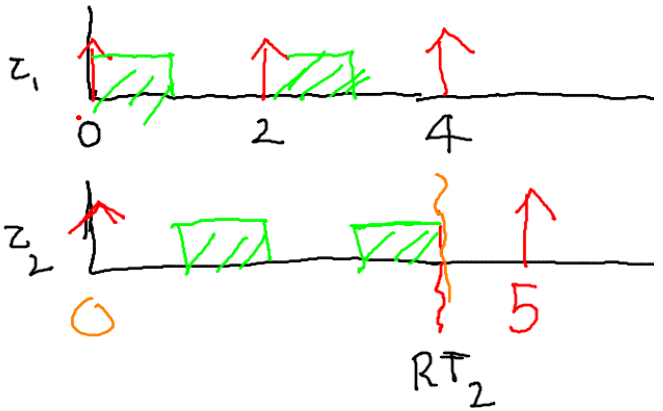
[Joseph, Pandya 86]
For a given priority assignment, let RT_i denote the worst case response time of task τ_i.

Equational characterization of RT_i

Let $hp(i)$ denote the set of tasks with priority higher than i.

$$RT_i = C_i + \Sigma_{j \in hp(i)} (\lceil RT_i / T_j \rceil \times C_j) \quad (1)$$

Task set $\tau_1 = (2, 1, 2)$ and $\tau_2 = (5, 2, 4)$



$$RT_2 = C_2 + I_1$$

$$I_1 = (\text{Count of } \tau_1 \text{ in } [0, RT_2)) * C_1$$
$$= (\lceil RT_2 / T_1 \rceil) * C_1$$

$$RT_2 = 2 + (\lceil RT_2 / 2 \rceil) * 1$$

Interference due to τ_j in RT_i is

$$I_i^j = (\lceil RT_i / T_j \rceil) * C_j$$

$RT_3 = C_3 + I_3^1 + I_3^2$

$\lceil x \rceil$ Ceiling of real x.

$\lceil 3.2 \rceil = 4$

$\lceil 3.0 \rceil = 3$

$$RT_2 = C_2 + \sum_{j \in h(2)} RT_2 / C_j$$

Exact Test: Necessary and Sufficient Condition

Theorem (Joseph-Pandya 1986)

- *The smallest solution of the equation (1) gives exact value of worst case response time RT_i.*
- *The sporadic taskset with $D_i \leq T_i$ with given priority assignment pr is feasible if and only if $RT_i \leq D_i$. (Necessary and Sufficient condition).*

Solving the equation iteratively

$C_1 + C_2 \quad + C_i \leq RT_i$

$$r_i^1 = C_1 + C_2 + \ldots + C_i$$

$$r_i^{n+1} = C_i + \Sigma_{j \in hp(i)} \left(\lceil r_i^n / T_j \rceil \times C_j \right)$$

Solving the equation iteratively

$$r_i^1 = C_1 + C_2 + \ldots + C_i$$

$$r_i^{n+1} = C_i + \Sigma_{j \in hp(i)} \left(\lceil r_i^n / T_j \rceil \times C_j \right)$$

- Start with r_i^1 as above. (An under-estimate of RT_i).
- Iteratively compute r_i^{n+1} from r_i^n.

Solving the equation iteratively

$$r_i^1 = C_1 + C_2 + \ldots + C_i$$

$$r_i^{n+1} = C_i + \Sigma_{j \in hp(i)} \left(\lceil r_i^n / T_j \rceil \times C_j \right)$$

- Start with r_i^1 as above. (An under-estimate of RT_i).
- Iteratively compute r_i^{n+1} from r_i^n.
- (Convergence) Stop when $r_i^{n+1} = r_i^n$. This r_i^n gives the worst case response time RT_i for Task τ_i.

Solving the equation iteratively

$$r_i^1 = C_1 + C_2 + \ldots + C_i$$

$$r_i^{n+1} = C_i + \Sigma_{j \in hp(i)} \left(\lceil r_i^n / T_j \rceil \times C_j \right)$$

- Start with r_i^1 as above. (An under-estimate of RT_i).
- Iteratively compute r_i^{n+1} from r_i^n.
- (Convergence) Stop when $r_i^{n+1} = r_i^n$. This r_i^n gives the worst case response time RT_i for Task τ_i.
- If $U \leq 1$ then the computation will converge.

Solving the equation iteratively

$$r_i^1 = C_1 + C_2 + \ldots + C_i$$

$$r_i^{n+1} = C_i + \Sigma_{j \in hp(i)} \left(\lceil r_i^n / T_j \rceil \times C_j \right)$$

- Start with r_i^1 as above. (An under-estimate of RT_i).
- Iteratively compute r_i^{n+1} from r_i^n.
- (Convergence) Stop when $r_i^{n+1} = r_i^n$. This r_i^n gives the worst case response time RT_i for Task τ_i.
- If $U \leq 1$ then the computation will converge.
- Fail if for any n, we get $r_i^n > D_i$. Priority assignment is infeasible.

Example: Exact feasibility Test

Taskset $(10, 1, 3)$, $(6, 2, 4)$, $(5, 1, 5)$.

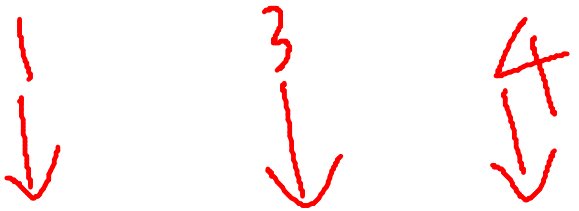
- $r_1 = C_1 + 0 = 1$.

Example: Exact feasibility Test

Taskset $(10, 1, 3)$, $(6, 2, 4)$, $(5, 1, 5)$.

- $r_1 = C_1 + 0 = 1$.
- $r_2^1 = C_1 + C_2 = 1 + 2 = 3$.
  $r_2^2 = C_2 + \lceil r_2^1 / T_1 \rceil * C_1 = 2 + \lceil 3/10 \rceil * 1 = 2 + 1 * 1 = 3$.
  $r_2^1 = r_2^2$ (convergence). Hence $RT_2 = 3$.

Example: Exact feasibility Test



Taskset $(10, 1, 3)$, $(6, 2, 4)$, $(5, 1, 5)$.

- $r_1 = C_1 + 0 = 1.$
- $r_2^1 = C_1 + C_2 = 1 + 2 = 3.$
  $r_2^2 = C_2 + \lceil r_2^1 / T_1 \rceil * C_1 = 2 + \lceil 3/10 \rceil * 1 = 2 + 1 * 1 = 3.$
  $r_2^1 = r_2^2$ (convergence). Hence $RT_2 = 3$.
- $r_3^1 = C_1 + C_2 + C_3 = 1 + 2 + 1 = 4$
  $r_3^2 = C_3 + (\lceil r_3^1 / T_1 \rceil * C_1) + (\lceil r_3^1 / T_2 \rceil * C_2)$
  $= 1 + (\lceil 4/10 \rceil * 1) + (\lceil 4/6 \rceil * 2)$
  $= 1 + (1 * 1) + (1 * 2) = 4.$
  $r_3^1 = r_3^2$ (convergence). Hence $RT_3 = 4$.

Example: Exact feasibility test

Task set (with $T_i = D_i$) is $\tau_1 = (7, 3)$ $\tau_2 = (12, 3)$ $\tau_3 = (20, 5)$.
Compute the response time for τ_3.

- $r_3^1 = C_1 + C_2 + C_3 = 3 + 3 + 5 = 11$

Example: Exact feasibility test

Task set (with $T_i = D_i$) is $\tau_1 = (7, 3)$ $\tau_2 = (12, 3)$ $\tau_3 = (20, 5)$.
Compute the response time for τ_3.

- $r_3^1 = C_1 + C_2 + C_3 = 3 + 3 + 5 = 11$
- $r_3^2 = C_3 + I_3^1 + I_3^2 = 5 + \lceil 11/7 \rceil * 3 + \lceil 11/12 \rceil * 3 = 14$

Example: Exact feasibility test

Task set (with $T_i = D_i$) is $\tau_1 = (7, 3)$ $\tau_2 = (12, 3)$ $\tau_3 = (20, 5)$.
Compute the response time for τ_3.

- $r_3^1 = C_1 + C_2 + C_3 = 3 + 3 + 5 = 11$
- $r_3^2 = C_3 + I_3^1 + I_3^2 = 5 + \lceil 11/7 \rceil * 3 + \lceil 11/12 \rceil * 3 = 14$
- $r_3^3 = 5 + \lceil 14/7 \rceil * 3 + \lceil 14/12 \rceil * 3 = 17$

Example: Exact feasibility test

Task set (with $T_i = D_i$) is $\tau_1 = (7, 3)$ $\tau_2 = (12, 3)$ $\tau_3 = (20, 5)$.
Compute the response time for τ_3.

- $r_3^1 = C_1 + C_2 + C_3 = 3 + 3 + 5 = 11$
- $r_3^2 = C_3 + I_3^1 + I_3^2 = 5 + \lceil 11/7 \rceil * 3 + \lceil 11/12 \rceil * 3 = 14$
- $r_3^3 = 5 + \lceil 14/7 \rceil * 3 + \lceil 14/12 \rceil * 3 = 17$
- $r_3^4 = 5 + \lceil 17/7 \rceil * 3 + \lceil 17/12 \rceil * 3 = 20$

Example: Exact feasibility test

Task set (with $T_i = D_i$) is $\tau_1 = (7, 3)$ $\tau_2 = (12, 3)$ $\tau_3 = (20, 5)$.
Compute the response time for τ_3.

- $r_3^1 = C_1 + C_2 + C_3 = 3 + 3 + 5 = 11$
- $r_3^2 = C_3 + I_3^1 + I_3^2 = 5 + \lceil 11/7 \rceil * 3 + \lceil 11/12 \rceil * 3 = 14$
- $r_3^3 = 5 + \lceil 14/7 \rceil * 3 + \lceil 14/12 \rceil * 3 = 17$
- $r_3^4 = 5 + \lceil 17/7 \rceil * 3 + \lceil 17/12 \rceil * 3 = 20$
- $r_3^5 = 5 + \lceil 20/7 \rceil * 3 + \lceil 20/12 \rceil * 3 = 20$

Example: Exact feasibility test

Task set (with $T_i = D_i$) is $\tau_1 = (7, 3)$ $\tau_2 = (12, 3)$ $\tau_3 = (20, 5)$.
Compute the response time for τ_3.

- $r_3^1 = C_1 + C_2 + C_3 = 3 + 3 + 5 = 11$
- $r_3^2 = C_3 + I_3^1 + I_3^2 = 5 + \lceil 11/7 \rceil * 3 + \lceil 11/12 \rceil * 3 = 14$
- $r_3^3 = 5 + \lceil 14/7 \rceil * 3 + \lceil 14/12 \rceil * 3 = 17$
- $r_3^4 = 5 + \lceil 17/7 \rceil * 3 + \lceil 17/12 \rceil * 3 = 20$
- $r_3^5 = 5 + \lceil 20/7 \rceil * 3 + \lceil 20/12 \rceil * 3 = 20$
- Convergence. Hence $RT_3 = 20$. Meets deadline.

Complexity

Naive Test Exponential

- Feasibility is in NP.
- Feasibility has pseudo-polynomial upper bound.
  - Each iteration takes constant amount of time.
  - Number of iterations is bounded by the value $\mathcal{O}(\Sigma_i\ (T_i)^2)$.

Recap of 1st and 2nd Lecture

Independent Task

- Hard Real-time System model.
- Sporadic task set $\tau_1, \ldots, \tau_n$ with $\tau_i = (T_i, C_i, D_i)$.
- Priority Assignment: Rate Monotonic and Deadline Monotonic.
- Feasibility of Priority Assignment: Analyse Critical Instance.
  - Naive Feasibility Test
  - Utilization bound tests
  - Exact response time test

$T_i = D_i$

Initialization;

Task 1

C_1 { PC_1, B_1 }

Private Comp 1

lock(q)

Critical Sect 1

unlock(q)

||

Task 2

C_2 { PC_2, B_2 }

Private Comp 2

lock(q)

Critical Sect 2

unlock(q)

Mars Pathfinder Bug

NASA sent mars pathfinder Sojourn on 4 July 1997.

- Stopped working after 87 sol due to software error.
- Diagnosed as a rare schedulability problem called Priority Inversion
- Fixed by reloading patched code.

Example: Mars Pathfinder (4)

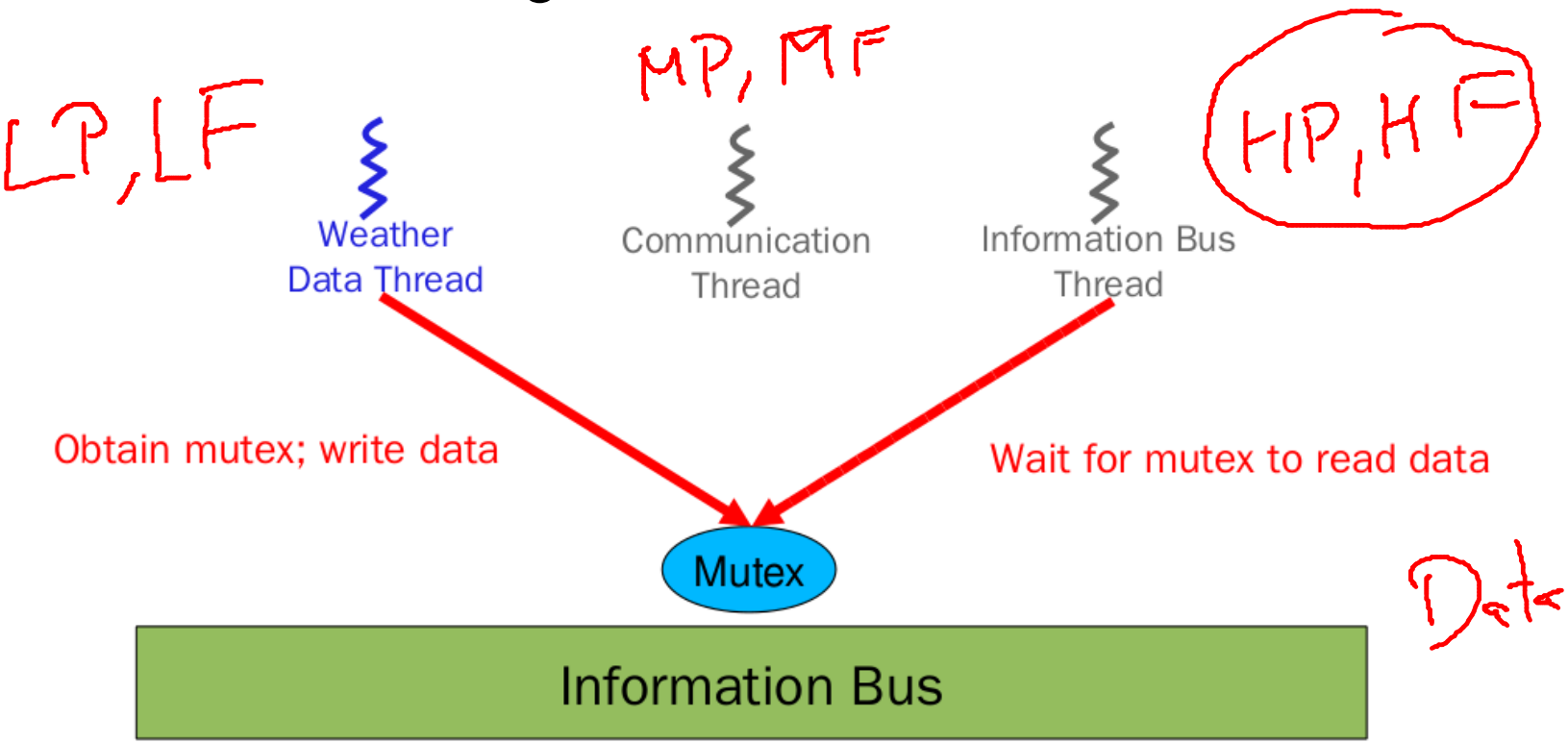
Pathfinder used VxWorks RTOS

- Threads for the 1553 bus for data collection, scheduled on every 1/8th sec cycle.
- 3 periodic tasks
  - Task 1 – Information Bus Thread: Bus Manager
    high frequency, high priority
  - Task 2 – Communication Thread
    medium frequency / priority, high execution time
  - Task 3 – Weather Thread: Geological Data Gatherer
    low frequency, low priority
- Each checks if the other executed and completed in the previous cycle
  - If the check fails, this is a violation of a hard real-time guarantee and the system is reset

Information Bus: A shared data structure for devices and processes of Lander and Cruise Control.

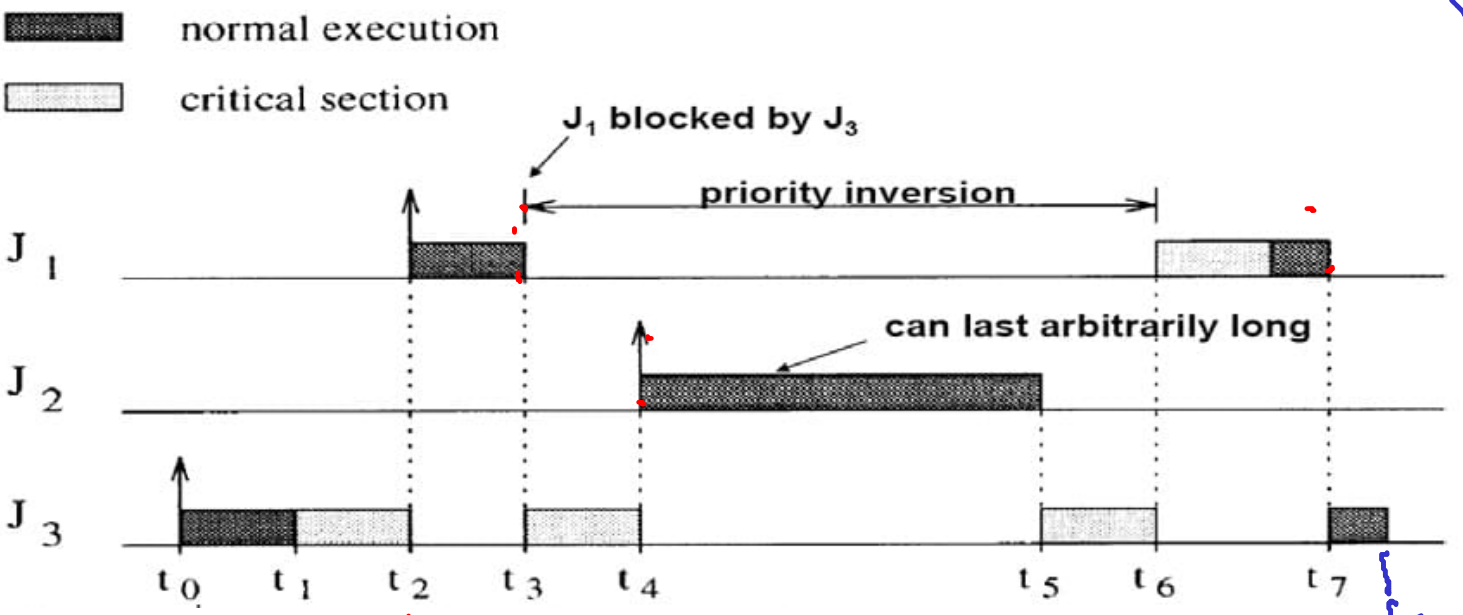
Processes with Shared Resources

- Shared resources with mutually exclusive access
- Tasks block waiting for shared resource.



Priority Inversion

[Lampson and Redell, 1980]



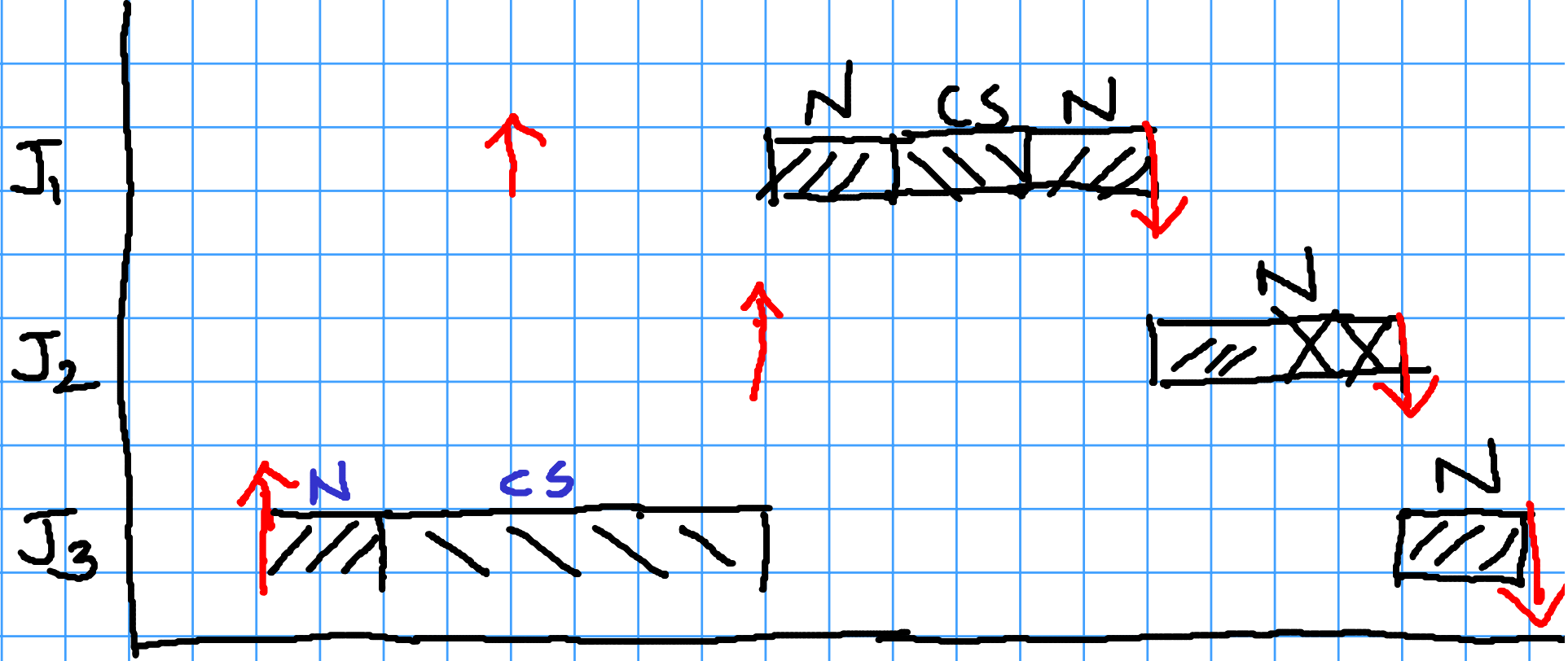
Priority Inversion

occurs when a lower priority task can block a higher priority task with which it is not sharing (locking) any resource.

Priority Inheritence Protocol (PIP)

- Ceiling Level of shared resource = highest priority of study process
- Temporarily increase the priority of task acquiring resource to high (ceiling) level.
- The critical section gets executed at high priority without blocking.
- Revert the task to original low priority on exiting the critical section.

Implemented in all major Kernels including POSIX threads, Java and VxWORKS.



Execution of tasks originally giving priority inversion (4 slides before) now using the priority ceiling protocol.

Response time of Tasks with Blocking and PIP

[Sha, Rajkumar, Lehoczky, 1990]

$$RT_i = C_i + B_i + \Sigma_{j \in hp(i)} \lceil RT_i / T_j \rceil \times C_j \qquad (2)$$

- Task τ_i can be blocked at most once by a lower level critical section.
- B_i is the worst case execution time of the longest critical section from lower level tasks sharing resources under PIP.

Dynamic Priority Assignment

Priority of a task can change during the execution as decided by the priority assignment algorithm.

Some Dynamic Priority Assignment Schemes

- Earliest Deadline First (EDF)
- Value based Scheduling (VBS)

Earliest Deadline First (EDF)

- Every time a task instance arrives, its time-to-deadline (TTD) is set to Deadline D_i.
- Conceptually, TTD decreases with passing of time.
- Everytime a task arrives or a task finishes execution,
  - TTD values of tasks are revissed.
  - Tasks are ordered by TTD values. The task with smallest TTD (also called earliest deadline) is scheduled to run.

Theorem (Liu, Layland 73)

A ~~sporadic~~ taskset with $T_i = D_i$ is schedulable with EDF iff $U \leq 1$.

Theorem (Liu Layland 73)

For tasksets with $D_i \leq T_i$ forall i, Earliest Deadline First (EDF) is optimal. If any arbitrary dynamic priority assignment scheme is feasible (meets deadlines) then so is EDF dynamic priority assignment.

EDF feaures

Advantages

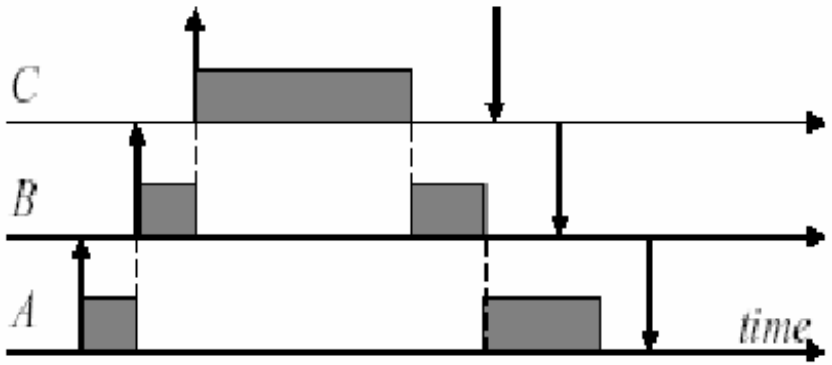
- Better CPU Utilization
- Can handle dynamic creation of tasks

Disadvantages

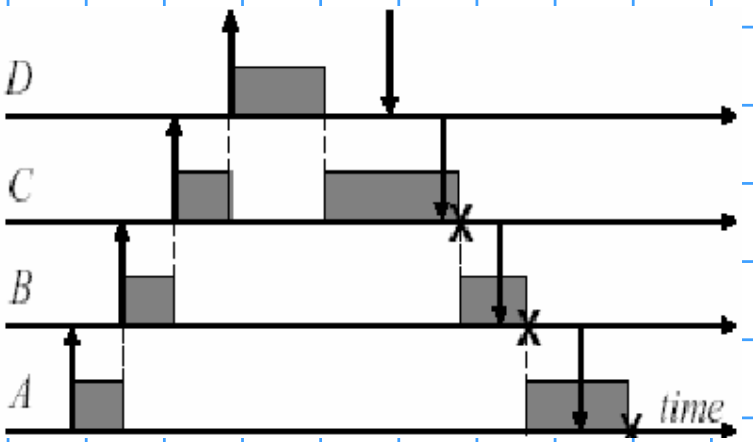
- Implmentation is more involved.
  Requires complex runtime mechanism to maintain and compute priorities.
- Very hard to predict actual response time of tasks.
- Domino effect under overload when large number of tasks miss their deadline.
- By contrast, For fixed priority assingment, low priority tasks miss their deadline first under overload. Response time of high priorty tasks is not affected.

Fixed priority assignment is widely used as compared to EDF.

Example of Unstability (Domino effect) in EDF scheduling.



EDF scheduling
all deadlines
met



With one new
arrival, Most
deadlines missed.

Applications and Extensions

- Jitter
- Task dependancies and synchronizations
- Multi-processors
- Used widely in analysis of systems with fixed priority scheduling:
  Including Mars Pathfinders.
  ISRO VSSC Launch Vehicle Task Scheduling

Thank You