CPSL
Cyber-Physical Systems Laboratory

Real-Time Scheduling

Chenyang Lu
CSE 467S Embedded Computing Systems

Washington University in St. Louis

Readings

- Single-Processor Scheduling: Hard Real-Time Computing Systems, by G. Buttazzo.
  - ☐ Chapter 4 Periodic Task Scheduling
  - ☐ Chapter 5 (5.1-5.4) Fixed Priority Servers
  - ☐ Chapter 7 (7.1-7.3) Resource Access Protocols

- Optional further readings
  - ☐ A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems, by Klein et al.
  - ☐ Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms, by Stankovic et al.

Real-Time Scheduling

- What are the optimal scheduling algorithms?
- How to assign priorities to processes?
- Can a system meet all deadlines?

Benefit of Scheduling Analysis

- Schedulability analysis reduces development time by 50%!
  - Reduce wasted implementation/testing rounds
  - Analysis time << testing
- More reduction expected for more complex systems

→*Quick exploration of design space!*

VEST (UVA)		Baseline (Boeing)	
Design – one processor	**40**	**Design – one processor**	**25**
		Implementation – one processor	**75**
Scheduling analysis - MUF ×	**1**	**Timing test** ×	**30**
Design - two processors	**25**	**Design - two processors**	**90**
		Implementation – two processors	**105**
Scheduling analysis - DM/Offset √	**1**	**Timing test** √	**20**
“Implementation”	**105**		
Total composition time	**172**	**Total composition time**	**345**

J.A. Stankovic, et al., VEST: An Aspect-Based Composition Tool for Real-Time Systems, RTAS 2003.

Consequence of Deadline Miss

- Hard deadline
  - System fails if missed.
  - Goal: guarantee no deadline miss.

- Soft deadline
  - User may notice, but system does not fail.
  - Goal: meet most deadlines most of the time.

Comparison

- General-purpose systems
  - ☐ Fairness to all tasks (no starvation)
  - ☐ Optimize throughput
  - ☐ Optimize average performance

- Embedded systems
  - ☐ Meet all deadlines.
  - ☐ Fairness or throughput is not important
  - ☐ Hard real-time: worry about worst case performance

Terminology

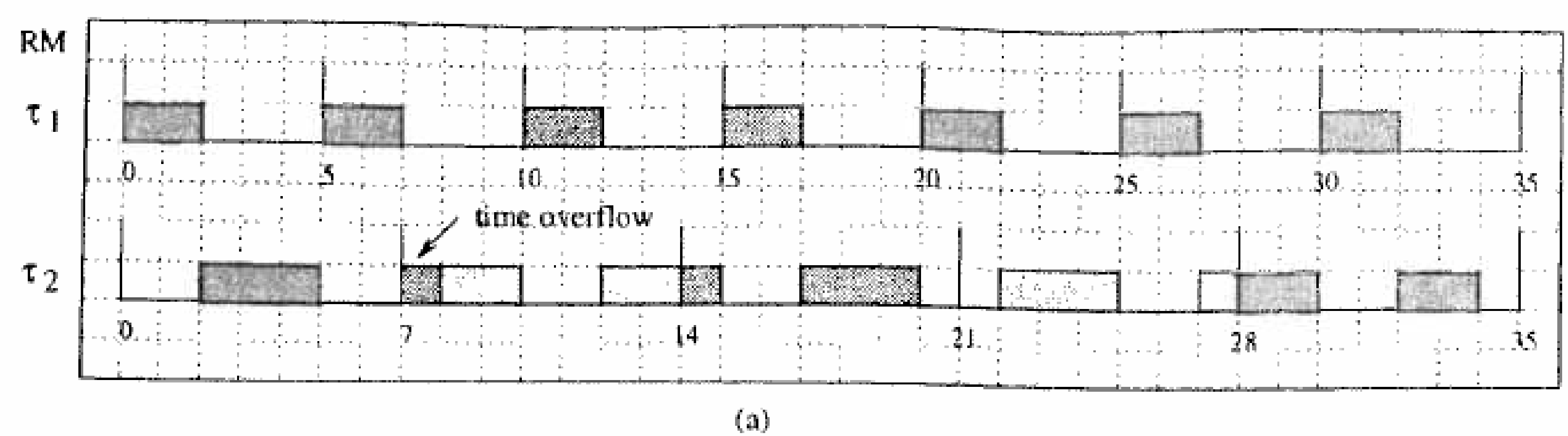
- Task
  - Map to a process or thread
  - May be released multiple times
- Job: an instance of a task
- Periodic task
  - Ideal: inter-arrival time = period
  - General: inter-arrival time >= period
- Aperiodic task
  - Inter-arrival time does not have a lower bound

Timing Parameters

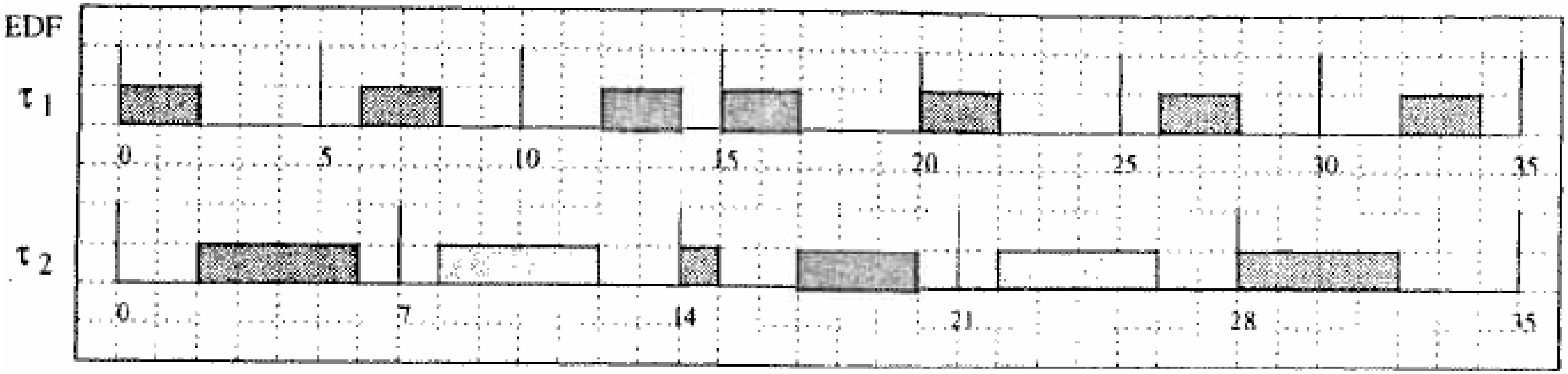
- Task T_i
  - Period P_i
  - Worst-case execution time C_i
  - Relative deadline D_i
- Job J_{ik}
  - Release time: time when a job is ready
  - Response time R_i = finish time – release time
  - Absolute deadline = release time + D_i
- A job misses its deadline if
  - Response time $R_i > D_i$
  - Finish time > absolute deadline

Example

- $P_1 = D_1 = 5, C_1 = 2; P_2 = D_2 = 7, C_2 = 4.$



(a)



(b)

Metrics

- A task set is schedulable if all jobs meet their deadlines.
- Optimal scheduling algorithm
  - If a task set is not schedulable under the optimal algorithm, it is not schedulable under any other algorithms.
- Overhead: Time required for scheduling.

Scheduling

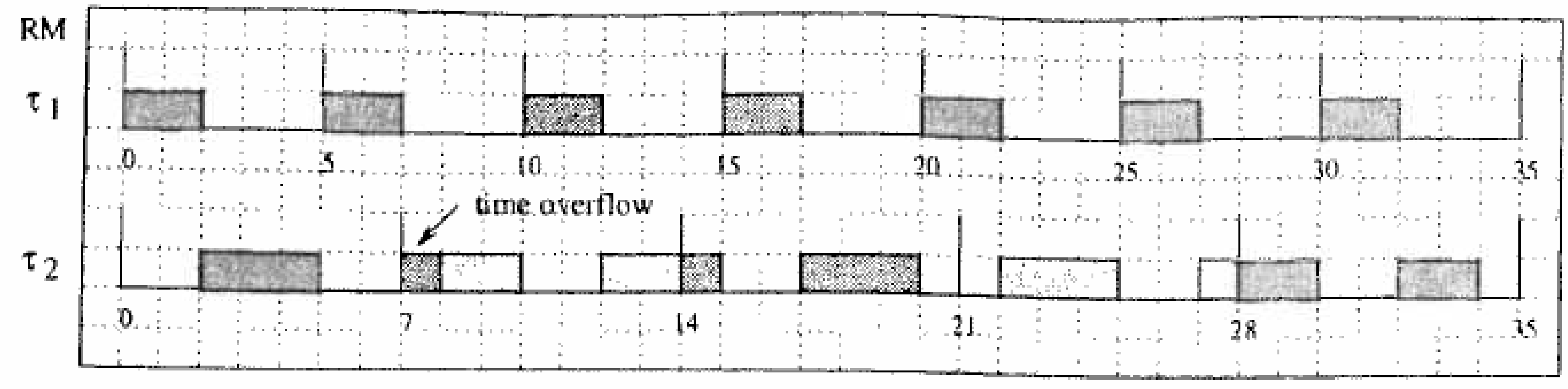
Single Processor

Optimal Scheduling Algorithms

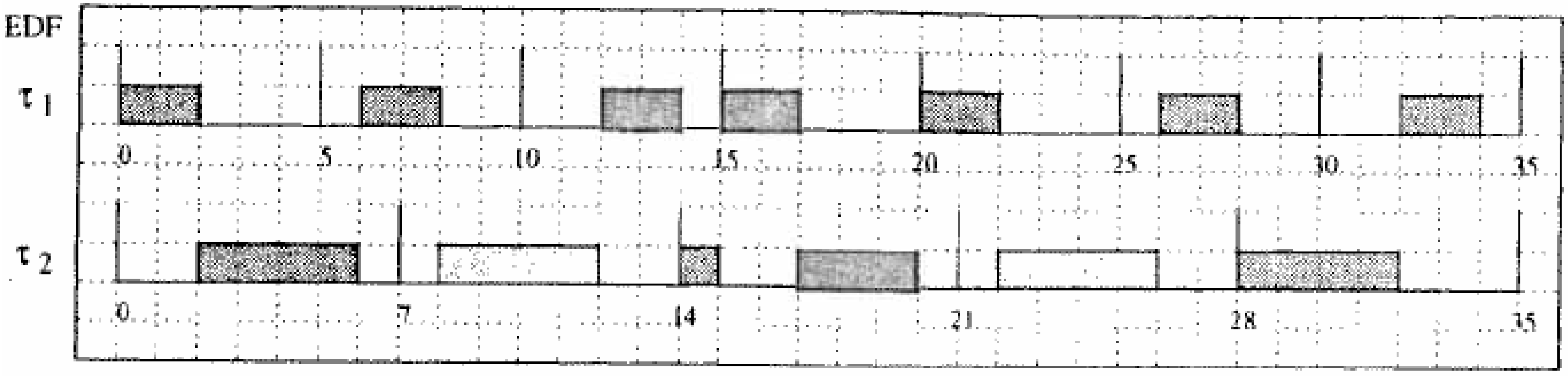
- Rate Monotonic (RM)
  - ☐ Higher rate (1/period) → Higher priority
  - ☐ Optimal preemptive static priority scheduling algorithm
- Earliest Deadline First (EDF)
  - ☐ Earlier absolute deadline → Higher priority
  - ☐ Optimal preemptive dynamic priority scheduling algorithm

Example

- $P_1 = D_1 = 5, C_1 = 2; P_2 = D_2 = 7, C_2 = 4.$



(a)



(b)

Assumptions

- Single processor.
- All tasks are periodic.
- Zero context switch time.
- Relative deadline = period.
- No priority inversion.

- RM and EDF have been extended to relax assumptions.

Schedulable Utilization Bound

- Utilization of a processor:

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i}$$

  - n: number of tasks on the processor.
- Utilization bound U_b: All tasks are guaranteed to be schedulable if $U \leq U_b$.
- No scheduling algorithm can schedule a task set if U>1
  - $U_b \leq 1$
  - An algorithm is optimal if its $U_b = 1$

RM Utilization Bound

- $U_b(n) = n(2^{1/n}-1)$
  - n: number of tasks
  - $U_b(2) = 0.828$
  - $U_b(n) \geq U_b(\infty) = \ln 2 = 0.693$

- $U \leq U_b(n)$ is a sufficient condition, but not necessary.

- $U_b = 1$ if all task periods are harmonic
  - Periods are multiples of each other
  - e.g., 1,10,100

Properties of RM

- RM may not guarantee schedulability even when CPU is not fully utilized.

- Low overhead: when the task set is fixed, the priority of a task never changes.
- Easy to implement on POSIX APIs.

EDF Utilization Bound

- $U_b = 1$
- $U \leq 1$: sufficient and necessary condition for schedulability.

- Guarantees schedulability if CPU is not over-utilized.
- Higher overhead than RM: task priority may change online.

Assumptions

- Single processor.
- All tasks are periodic.
- Zero context switch time.
- Relative deadline = period.
- No priority inversion.

- What if relative deadline < period?

Optimal Scheduling Algorithms

Relative Deadline < Period

- Deadline Monotonic (DM)
  - ☐ Shorter relative deadline → Higher priority
  - ☐ Optimal preemptive static priority scheduling

- Earliest Deadline First (EDF)
  - ☐ Earlier absolute deadline → Higher priority
  - ☐ Optimal preemptive dynamic priority scheduling algorithm

DM Analysis

- Sufficient but pessimistic test

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{1/n} - 1)$$

- Sufficient and necessary test: response time analysis

Response Time Analysis

- Works for any fixed-priority preemptive scheduling algorithm.
- Critical instant
  - results in a task's longest response time.
  - when all higher-priority tasks are released at the same time.
- Worst-case response time
  - Tasks are ordered by priority; T_1 has highest priority

$$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{P_j} \right\rceil C_j$$

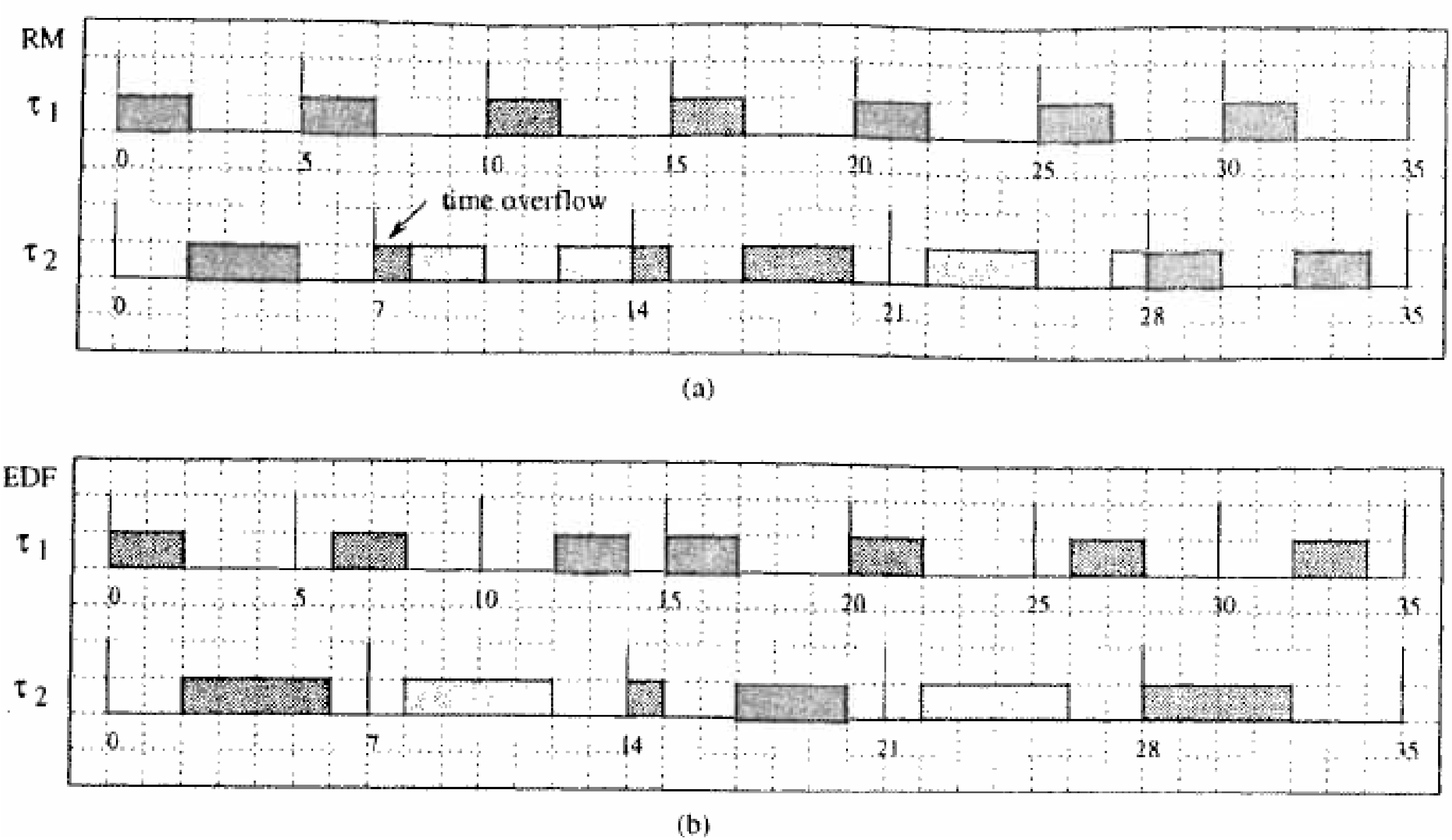
Response Time Analysis

```
Tasks are ordered by priority;
$T_1$ has the highest priority.

for (each task $T_j$) {
   I = 0; R = 0;
   while (I + $C_j$ > R) {
    R = I + $C_j$;
    if (R > $D_j$) return UNSCHEDULABLE;
        $I = \sum_{k=1}^{j-1} \left\lceil \frac{R}{P_k} \right\rceil C_k$;
   }
}
return SCHEDULABLE;
```

Example

- $P_1 = D_1 = 5, C_1 = 2; P_2 = D_2 = 7, C_2 = 4.$



(a)

(b)

EDF: Processor Demand Analysis

- To start, assume $D_i = P_i$
- Processor demand in interval [0, L]: total time needed for completing all jobs with deadlines no later than L.

$$C_P(0, L) = \sum_{i=1}^{n} \left\lfloor \frac{L}{P_i} \right\rfloor C_i$$

Schedulable Condition

- Theorem: A set of periodic tasks is schedulable by EDF if and only if for all L ≥ 0:

$$L \geq \sum_{i=1}^{n} \left\lfloor \frac{L}{P_i} \right\rfloor C_i$$

- There is enough time to meet processor demand at every time instant.

Busy Period B_p

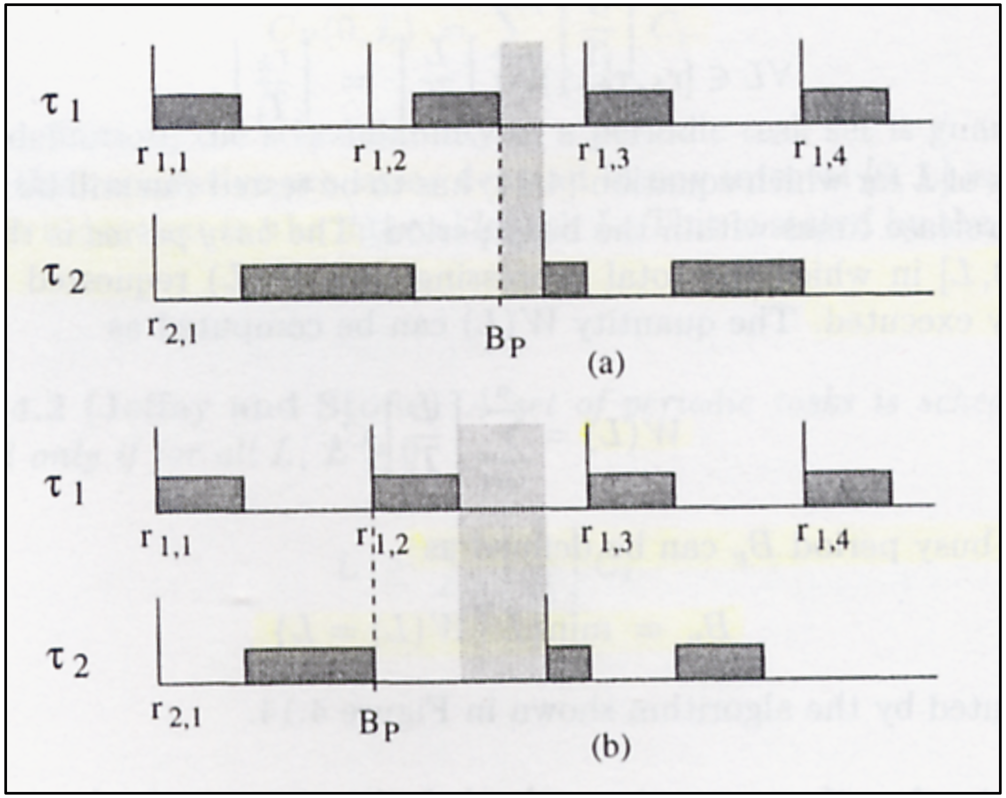
- End at the first time instant L when all the released jobs are completed
- W(L): Total execution time of all tasks released by L.

$$W(L) = \sum_{i=1}^{n} \left\lceil \frac{L}{P_i} \right\rceil C_i$$

$$B_p = \min\{L \mid W(L) = L\}$$

Properties of Busy Period

- CPU is fully utilized during a busy period.
- The end of a busy period coincides with the beginning of an idle time or the release of a periodic job.



Schedulable Condition

- All tasks are schedulable if and only if

$$L \geq \sum_{i=1}^{n} \left\lfloor \frac{L}{P_i} \right\rfloor C_i$$

at all job release times before min(B_p, H)

Compute Busy Period

```
busy_period
{
    H = lcm($P_1$,…,$P_n$); /* least common multiple */
    L = $\sum C_i$;
    L' = W(L);
    while (L' != L and L' <= H) {
        L = L';
        L' = W(L);
    }
    if (L' <= H)
        $B_p$ = L;
    else
        $B_p$ = INFINITY;
}
```

Processor Demand Test: $D_i < P_i$

- A set of periodic tasks with deadlines no more than than periods is schedulable by EDF if and only if

$$\forall L \in D, \quad L \geq \sum_{i=1}^{n} \left[\left(\left\lfloor \frac{L - D_i}{P_i} \right\rfloor + 1 \right) C_i \right]$$

where $D = \{D_{i,k} \mid D_{i,k} = kP_i + D_i, D_{i,k} \leq \min(B_p, H), 1 \leq i \leq n, k \geq 0\}$.

- Note: only need to test all deadlines before $\min(B_p, H)$.

Schedulability Test Revisited

	D = P	D < P
Static Priority	RM Utilization bound Response time	DM Response time
Dynamic Priority	EDF Utilization bound	EDF Processor demand

Assumptions

- Single processor.
- All tasks are periodic.
- Zero context switch time.
- Relative deadline = period.
- No priority inversion.

Questions

- What causes priority inversion?
- How to reduce priority inversion?
- How to analyze schedulability?

Priority Inversion

- A low-priority task blocks a high-priority task.

- Sources of priority inversion
  - ☐ Access shared resources guarded by semaphores.
  - ☐ Access non-preemptive subsystems, e.g., storage, networks.

Semaphores

- OS primitive for controlling access to shared variables.
  - ☐ Get access to semaphore S with wait(S).
  - ☐ Execute critical section to access shared variable.
  - ☐ Release semaphore with signal(S).

- Mutex: at most one process can hold a mutex.

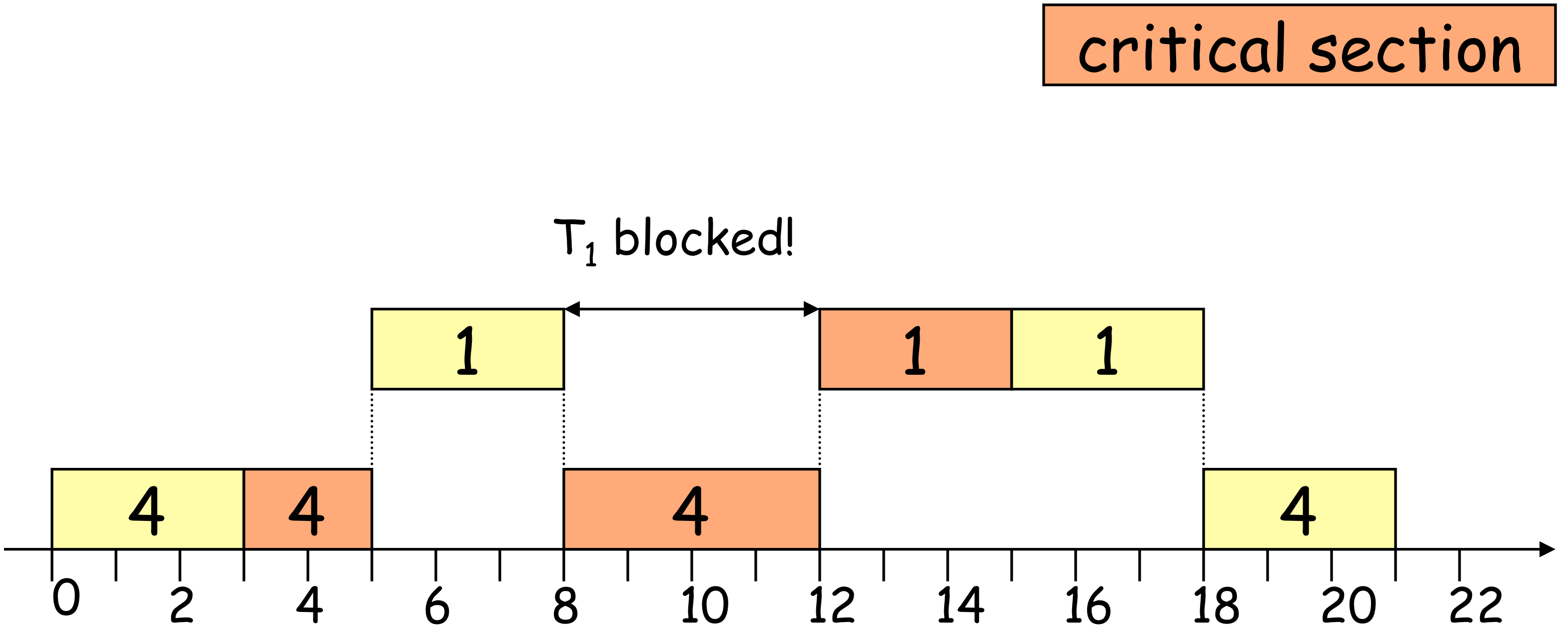
```
wait(mutex_info_bus);
Write data to info bus;
signal(mutex_info_bus);
```

What happened to Pathfinder?

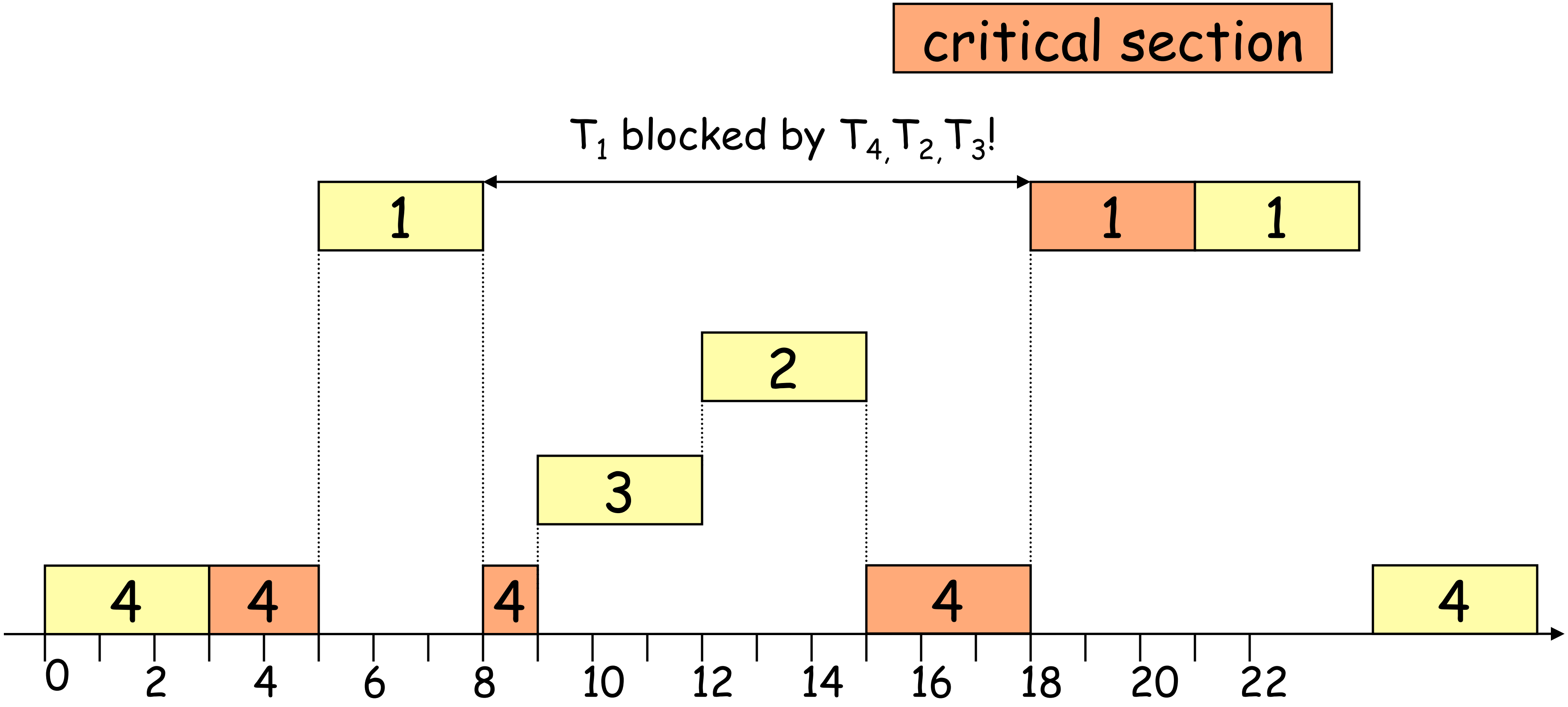
- …But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data…

Real-World (Out of This World) Story: Priority inversion almost ruined the path finder mission on MARS! http://research.microsoft.com/~mbj/

Priority Inversion

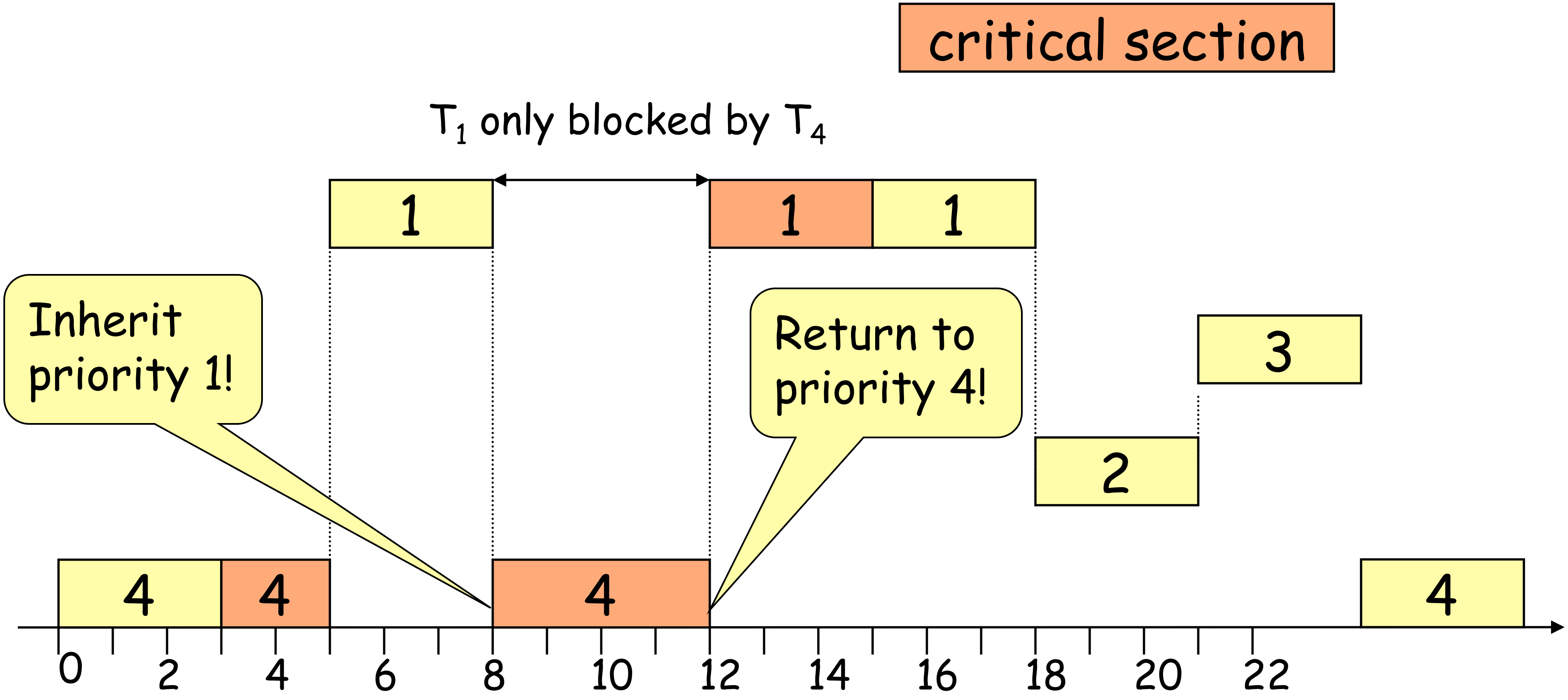


Unbounded Priority Inversion



Solution

- The low-priority task ***inherits*** the priority of the blocked high-priority task.



Priority Inheritance Protocol (PIP)

- When task T_i is blocked on a semaphore held by T_k
  - If prio(T_k) is lower than prio(T_i), prio(T_i) → T_k

- When T_k releases a semaphore
  - If T_k no longer blocks any tasks, it returns to its normal priority.
  - If T_k still blocks other tasks, it inherits the highest priority of the remaining tasks that it is blocking.

- Priority Inheritance is transitive
  - T_2 blocks T_1 and inherits prio(T_1)
  - T_3 blocks T_2 and inherits prio(T_1)

How was Path Finder saved?

- When created, a VxWorks mutex object accepts a boolean parameter that indicates if priority inheritance should be performed by the mutex.
  - The mutex in question had been initialized with the parameter FALSE.

- VxWorks contains a C interpreter intended to allow developers to type in C expressions/functions to be executed on the fly during system debugging.

- The initialization parameter for the mutex was stored in global variables, whose addresses were in symbol tables also included in the launch software, and available to the C interpreter.

- A C program was uploaded to the spacecraft, which when interpreted, changed these variables from FALSE to TRUE.

- No more system resets occurred.

Bounded Number of Blocking

- Assumptions of analysis
  - Fixed priority scheduling
  - All semaphores are binary
  - All critical sections are properly nested

- Task T_i can be blocked by at most min(m,n) times
  - m: number of distinct semaphores that can be used to block T_i
  - n: number of lower-priority tasks that can block T_i

Extended RMS Utilization Bound

- A set of periodic tasks can be scheduled by RMS/PIP if

$$\forall i, \qquad 1 \le i \le n, \qquad \sum_{k=1}^{i} \frac{C_k}{P_k} + \frac{B_i}{P_i} \le i(2^{1/i} - 1)$$

  - Tasks are ordered by priorities (T_1 has the highest priority).
  - B_i: the maximum amount of time when task T_i can be blocked by a lower-priority task.

Extended Response Time Analysis

- Consider the effect of blocking on response time:

$$R_i = C_i + B_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{P_j} \right\rceil C_j$$

- The analysis becomes sufficient but not necessary.

Priority Ceiling

- $C(S_k)$: Priority ceiling of a semaphore S_k
  - Highest priority among tasks requesting S_k.

- A critical section guarded by S_k may block task T_i only if $C(S_k)$ is higher than prio(T_i)

Compute B_i

```
Assumption: no nested critical sections.

/* potential blocking by other tasks */
B1=0; B2=0;
for each $T_j$ with priority lower than $T_i$ {
    b1 = longest critical section in $T_j$ that can block $T_i$
    B1 = B1 + b1
}

/* potential blocking by semaphores */
for each semaphore $S_k$ that can block $T_i$ {
    b2 = longest critical section guarded by $S_k$ among lower priority tasks
    B2 = B2 + b2
}
return min(B1, B2)
```

Priority Ceiling Protocol

- Priority ceiling of the processor: The highest priority ceiling of all semaphores currently held.

- A task can acquire a resource only if
  - the resource is free, AND
  - it has a higher priority than the priority ceiling of the system.

- A task is blocked by at most one critical section.

- Higher run-time overhead than PIP.

Assumptions

- Single processor.
- All tasks are periodic.
- Zero context switch time.
- Relative deadline = period.
- No priority inversion.

Hybrid Task Set

- Periodic tasks + aperiodic tasks

- Problem: arrival times of aperiodic tasks are unknown

- Sporadic task with a hard deadline
  - Inter-arrival time must be lower bounded
  - Schedulability analysis: treated as a periodic task with period = minimum inter-arrival time → can be very pessimistic.

- Aperiodic task with a soft deadline
  - Possibly unbounded inter-arrival time
  - Maintain hard guarantees on periodic tasks
  - Reduce response time of aperiodic tasks

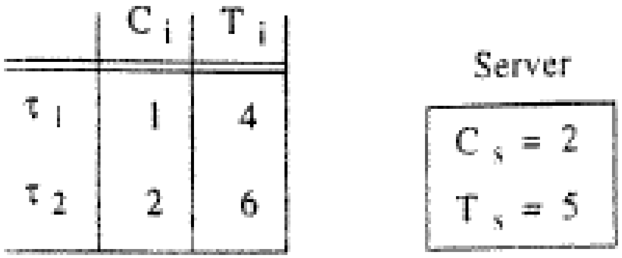
Background Scheduling

- Handle aperiodic requests with the lowest-priority task
- Advantages
  - Simple
  - Aperiodic tasks usually has no impact on periodic tasks.
- Disadvantage
  - Aperiodic tasks have very long response times when the utilization of periodic tasks is high.
- Acceptable only if
  - System is not busy
  - Aperiodic tasks can tolerate long delays

Polling Server

- A periodic task (server) serves aperiodic requests.
  - Period: P_s
  - Capacity: C_s
- Released periodically at period P_s
- Serves any pending aperiodic requests
- Suspends itself until the end of the period if
  - it has used up its capacity, or
  - no aperiodic request is pending
- Capacity is replenished to C_s at the beginning of the next period

Example: Polling Server

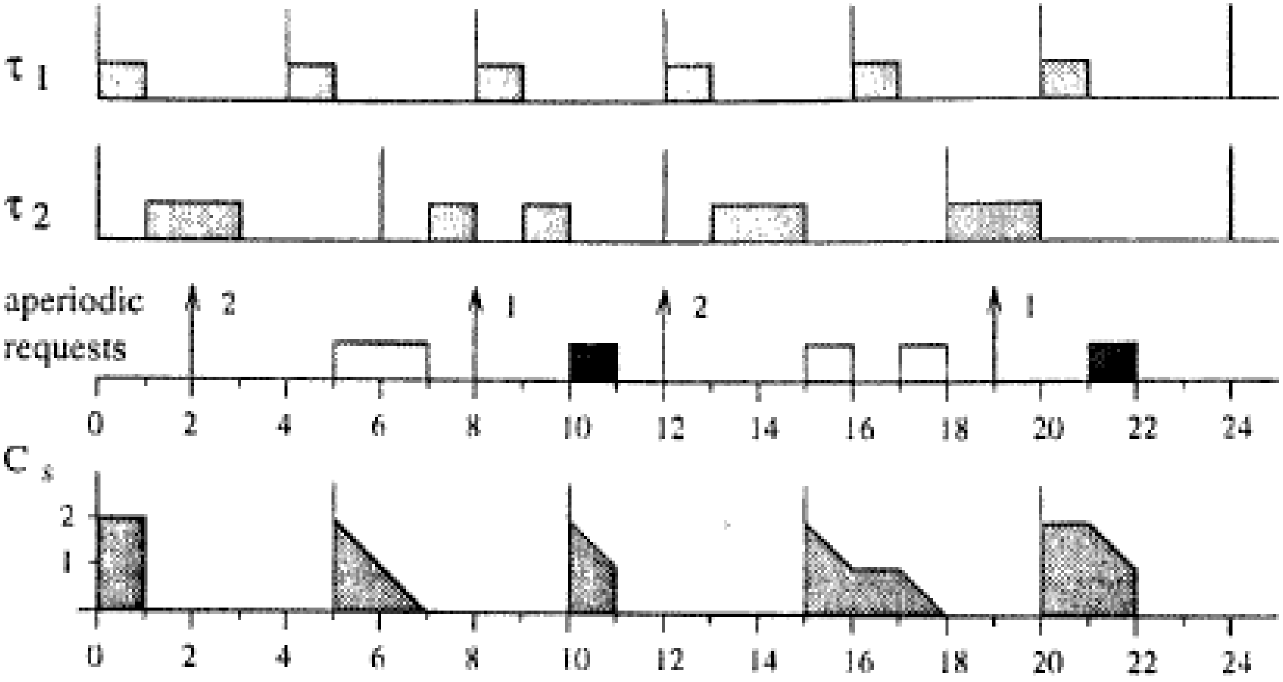


	C_i	T_i
τ_1	1	4
τ_2	2	6

Server

$C_s = 2$

$T_s = 5$



Schedulability

- Polling server has the same impact on periodic tasks as a periodic task.
  - n tasks with m servers: $U_p + U_s \leq U_b(n+m)$
- Disadvantage: If an aperiodic request "misses" the server, it has to wait till the next period. → long response time.
- Can have multiple servers (with different periods) for different classes of aperiodic requests

Deferrable Server (DS)

- Preserve unused capacity till the end of the current period → shorter response to aperiodic requests.
- Impact on periodic tasks differs from a periodic task.

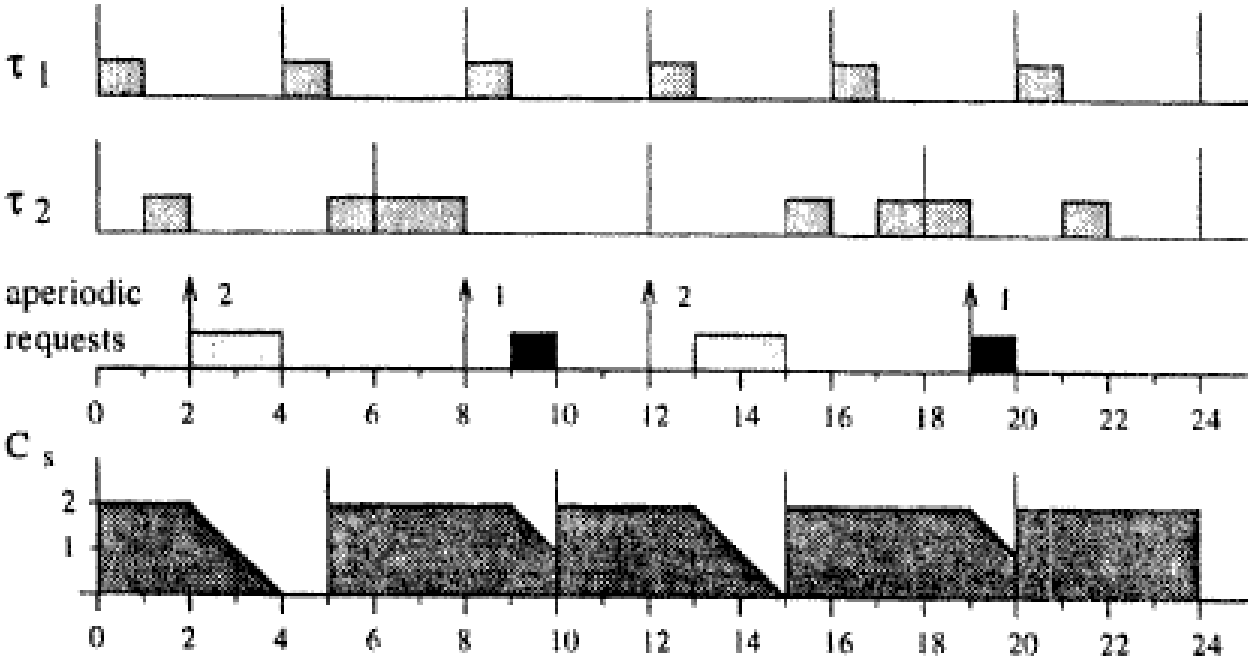
Example: Deferrable Server

	C_i	T_i
τ_1	1	4
τ_2	2	6

Server

$C_s = 2$

$T_s = 5$



RM Utilization Bound with DS

- Under RMS

$$U_b = U_s + \ln\left(\frac{U_s + 2}{2U_s + 1}\right)$$

- As n → ∞:

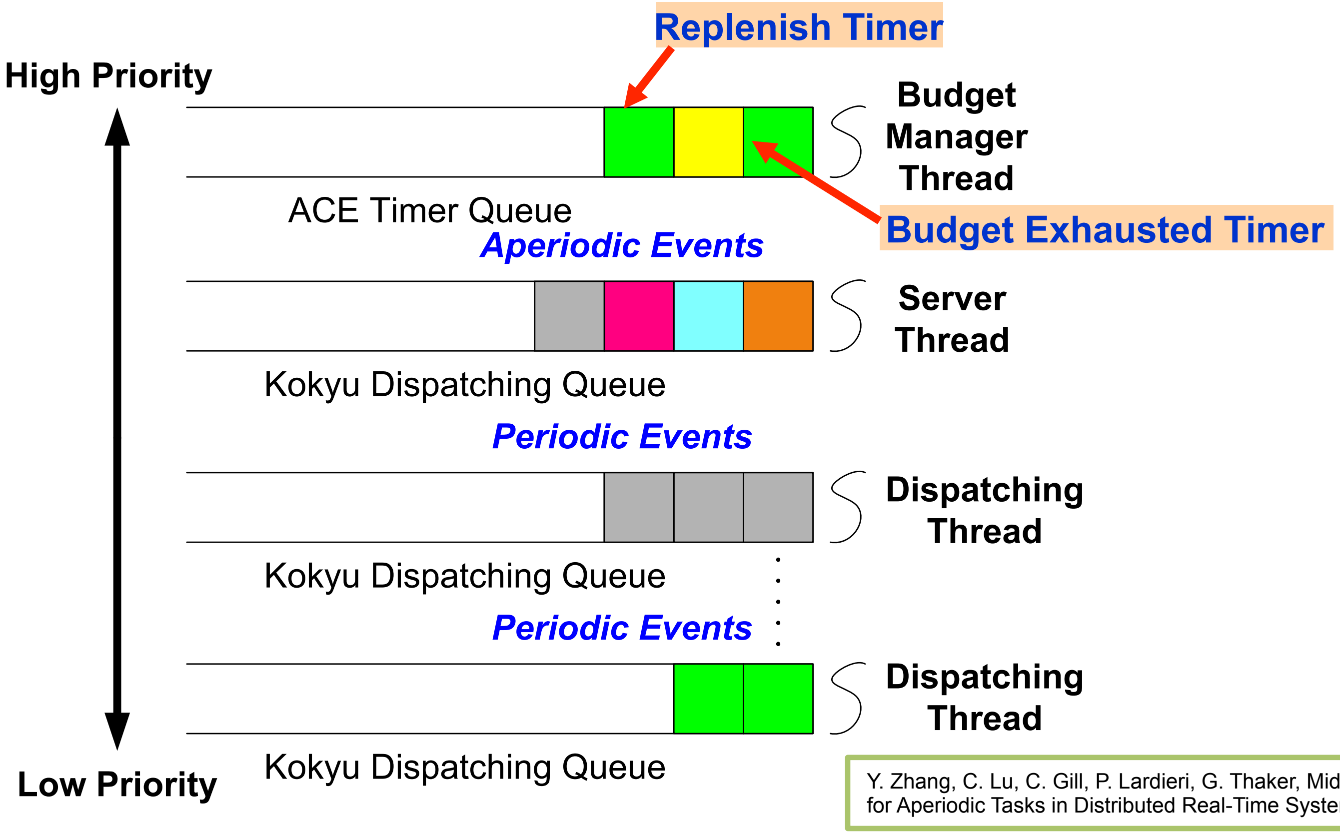
$$U_b = U_s + n\left[\left(\frac{U_s + 2}{2U_s + 1}\right)^{1/n} - 1\right]$$

  - When U_s = 0.186, min U_b = 0.652

- System is schedulable if $U_p \le \ln\left(\frac{U_s + 2}{2U_s + 1}\right)$

DS: Middleware Implementation

- First DS implementation on top of priority-based OS (e.g., Linux, POSIX)
- Server thread processes aperiodic events (2nd highest priority)
- Budget manager thread (highest priority) manages the budget and controls the execution of server thread



Replenish Timer

High Priority

Budget Manager Thread

ACE Timer Queue

Aperiodic Events

Budget Exhausted Timer

Server Thread

Kokyu Dispatching Queue

Periodic Events

Dispatching Thread

Kokyu Dispatching Queue

Periodic Events

Dispatching Thread

Low Priority

Kokyu Dispatching Queue

Y. Zhang, C. Lu, C. Gill, P. Lardieri, G. Thaker, Middleware Support for Aperiodic Tasks in Distributed Real-Time Systems, RTAS'07.

Assumptions

- Single processor.
- All tasks are periodic.
- Zero context switch time.
- Relative deadline = period.
- No priority inversion.

Context Switch Time

- RTOS usually has low context switch overhead.

- Context switches can still cause overruns in a tight schedule.
  - Leave margin in your schedule.

- Techniques exist to reduce number of context switches by avoiding certain preemptions.

- Other forms of overhead: cache, thread migration, interrupt handling, bus contention, thread synchronization…

Fix an Unschedulable System

- Reduce task execution times.
- Reduce blocking factors.
- Get a faster processor.
- Replace software components with hardware.
- Multi-processor and distributed systems.

Final

- 1-2:30 April 21st
- Open book/note
- Scope: Operating Systems, Real-Time Scheduling

Final Demo

- April 23rd, 1pm-2:30pm
- 20 min per team
- Set up and test your demo in advance
- All expected to attend the whole session
- Return devices to Rahav
- It’ll be fun! ☺

Project Report

- Submit report and materials by 11:59pm April 30th.
- Email to Rahav
- Report
  - Organization: See conference papers in the reading list.
  - 6 pages, double column, 10 pts fonts.
  - Use templates on the class web page.
- Other materials
  - Slides of your final presentation
  - Source code
  - Documents: README, INSTALL, HOW-to-RUN
  - Video (Youtube is welcome!)

Suggested Report Outline

- Abstract
- Introduction
- Goals
- Design: Hardware and Software
- Implementation
- Experiments
- Related Work
- Lessons Learned
- Conclusion and Future Work

Peer Review

- For fairness in project evaluation.

- Email me individually by 11:59pm, April 30th
  - ☐ Estimated percentage of contribution from each team member.
  - ☐ Brief justification.