

EE445M/ECE380L.12

Embedded and Real-Time Systems/ Real-Time Operating Systems

Lecture 6: Real-Time Scheduling, Priority Scheduler

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

1

Real-Time Scheduling

- Tasks have deadlines
 - Some tasks are more important than others
 - In order to do something first, something else must be second
 - Priority scheduler
- Reactivity
 - When to run the scheduler?
 - Periodically, `systick` and `sleep`
 - On `OS_wait`
 - On `OS_signal`
 - On `OS_sleep`, `OS_kill`

Reference Book,
Chapter 6

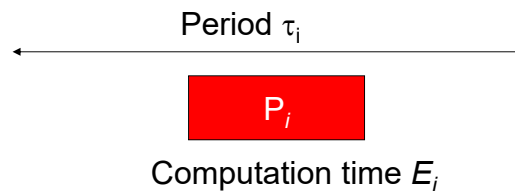
Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

2

Real-Time Scheduling Model

- E_i is execution time of task i
- Deadline τ_i is period of task i



- Response time r_i
 - Time from arrival until finish of task
- Lateness l_i
 - $r_i - \tau_i$

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

Source: M. Jacome, UT Austin

3

Priority Scheduling

- Execute highest priority first
 - Two tasks at same priority?
- Assign a dollar cost for delays
 - Minimize cost
 - Minimize latency on real-time tasks
 - Minimize maximum lateness (relative to deadline)

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

4

Priority Scheduler

- Assigns each thread a priority number
 - Reduce latency (response time) by giving high priority
 - Static (creation) or dynamic (runtime)
 - Performance measures (utilization, latency/lateness)
- Strictly run the ready task with highest priority at all times
 - Priority 2 is run only if no priority 1 are ready
 - Priority 3 only if no priority 1 or priority 2 are ready
 - If all have the same priority, use a round-robin system
- Blocking semaphores and not spinlock semaphores
- On a busy system, low priority threads may never be run
 - Problem: Starvation
 - Solution: Aging

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

5

How to find Highest Priority

- Search all for highest priority ready thread
 - Skip if blocked
 - Skip if sleeping
 - Linear search speed (number of threads)
- Sorted list by priority
 - Chain/unchain as ready/blocked
- Priority bit table (uCOS-II and uCOS-III)
 - See [OSUnMapTbl](#) in `os_core.c`
 - See [OS_Sched](#) (line 1606) Software\uCOS-II\Source
 - See [CPU_CntLeadZeros](#) in `cpu_a.asm`

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12Software\uC-CPU\Cortex-M3\RealView

6

Adaptive Priority- Aging

- Solution to starvation
- Real and temporary priorities in TCB
- Priority scheduler uses temporary priority
- Increase temporary priority periodically
 - If a thread is not running
- Reset temporary back to real when runs

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

7

I/O Centric Scheduler

- Automatically adjusts priority
 - Exponential queue
- High priority to I/O bound threads
 - I/O needs low latency
 - Every time it issues an input or output,
 - Increase priority by one
- Low priority to CPU bound threads
 - Every time it runs to completion
 - Decrease priority by one

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

8

Exponential Queue

- Exponential comes from doubling/halving
 1. Round robin with variable timeslices
 - Time slices 8,4,2,1 ms
 2. Priority with variable priority/timeslices
 - Time slices 8,4,2,1 ms
 - Priorities 0,1,2,3

Final exam 2006, Q5

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

9

Scheduling Metrics

- How do we evaluate a scheduling policy?
 - Ability to satisfy all deadlines
 - Minimize maximum lateness
 - CPU utilization $\sum_i E_i / \tau_i$
 - Percentage of time devoted to useful work
 - Scheduling overhead
 - Time required to make scheduling decision

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

Source: M. Jacome, UT Austin

10

Scheduling Algorithms

- Rate monotonic scheduling (RMS), static
 - Assign priority based on how frequent task is run
 - Lower *period* (more frequent) are higher priority
- Earliest deadline first (EDF), dynamic
 - Assign priority based on closest deadline
- Least slack-time first (LST), dynamic
 - Slack = (time to deadline)-(work left to do)
- ...

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

11

Scheduling Analysis

- Rate monotonic scheduling theorem
 - All n tasks are periodic
 - Priority based on period τ_i
 - Maximum execution time E_i
 - No synchronization between tasks (independent)
 - Execute highest priority task first
 - Guarantee deadlines if processor utilization:

$$\sum \frac{E_i}{\tau_i} \leq n \left(2^{1/n} - 1 \right) \leq \ln(2) \approx 69\%$$

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

12

Rate Monotonic Analysis (RMA)

- Optimal (fixed) priority assignment
 - Shortest-period process gets highest priority
 - priority based preemption can be used...
 - Priority inversely proportional to period
 - Break ties arbitrarily
- No fixed-priority scheme does better.
 - RMS provides the highest worst case CPU utilization while ensuring that all processes meet their deadlines

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

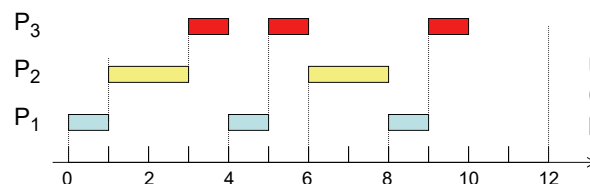
Source: M. Jacome, UT Austin 13

RMS Example 1

Process P_i	Execution Time E_i	Period T_i
P_1	1	4
P_2	2	6
P_3	3	12

Static priority: $P_1 \gg P_2 \gg P_3$

Critical instant
all tasks arrive at same time



Unrolled schedule
(least common multiple of process periods)

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

Source: M. Jacome, UT Austin 14

RMS Example 2

Process P_i	Execution Time E_i	Period T_i
P_1	1	4
P_2	6	8

Is this task set schedulable?? If yes, give the CPU utilization.

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

Source: M. Jacome, UT Austin 15

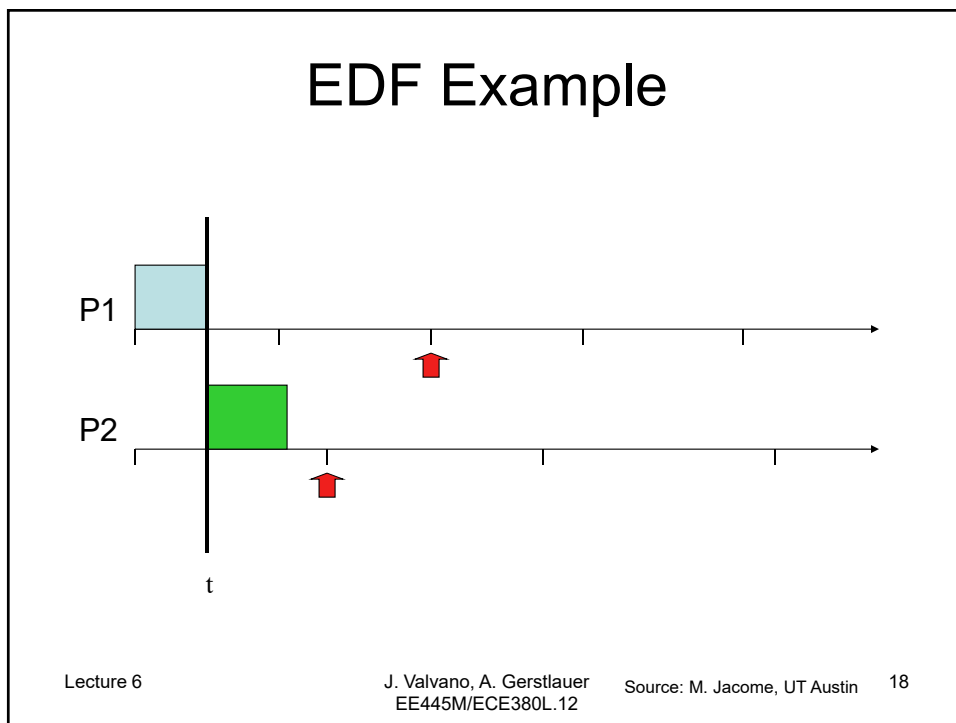
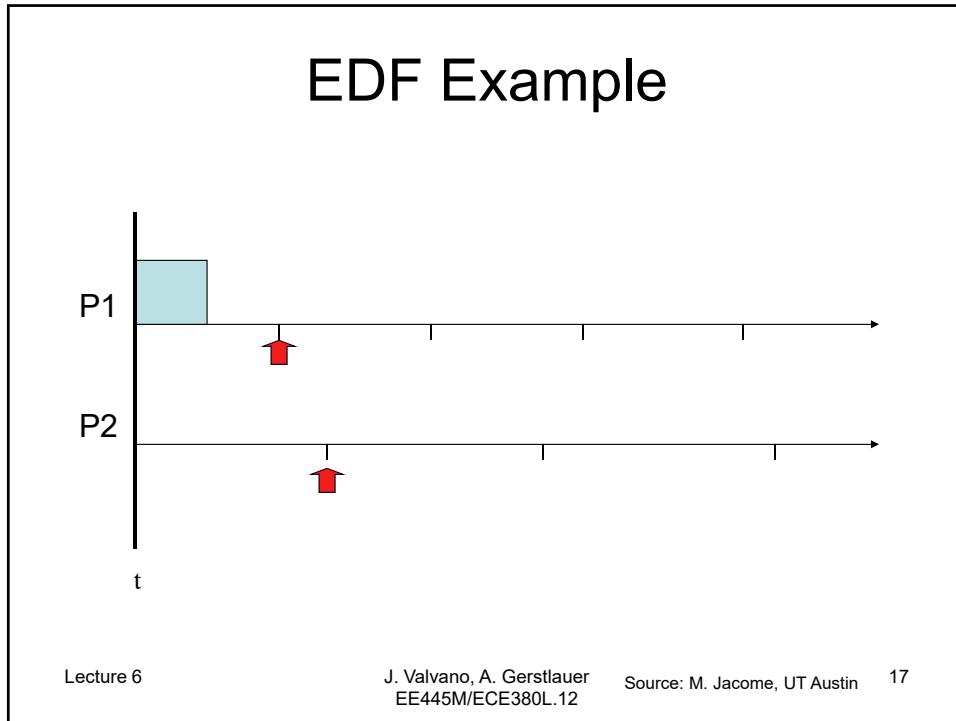
Earliest-Deadline-First (EDF)

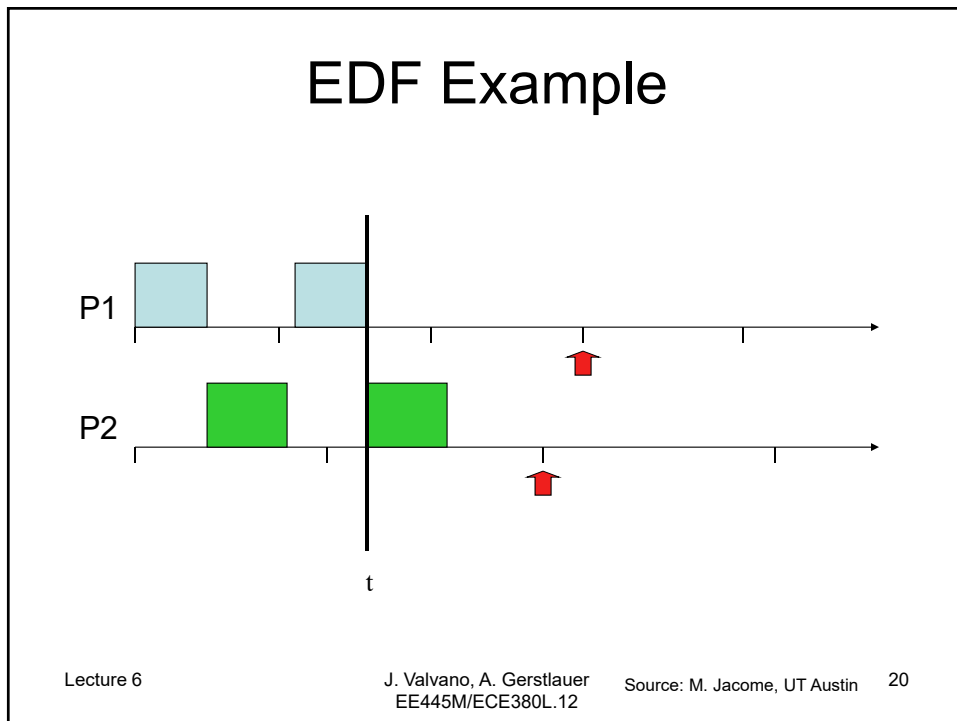
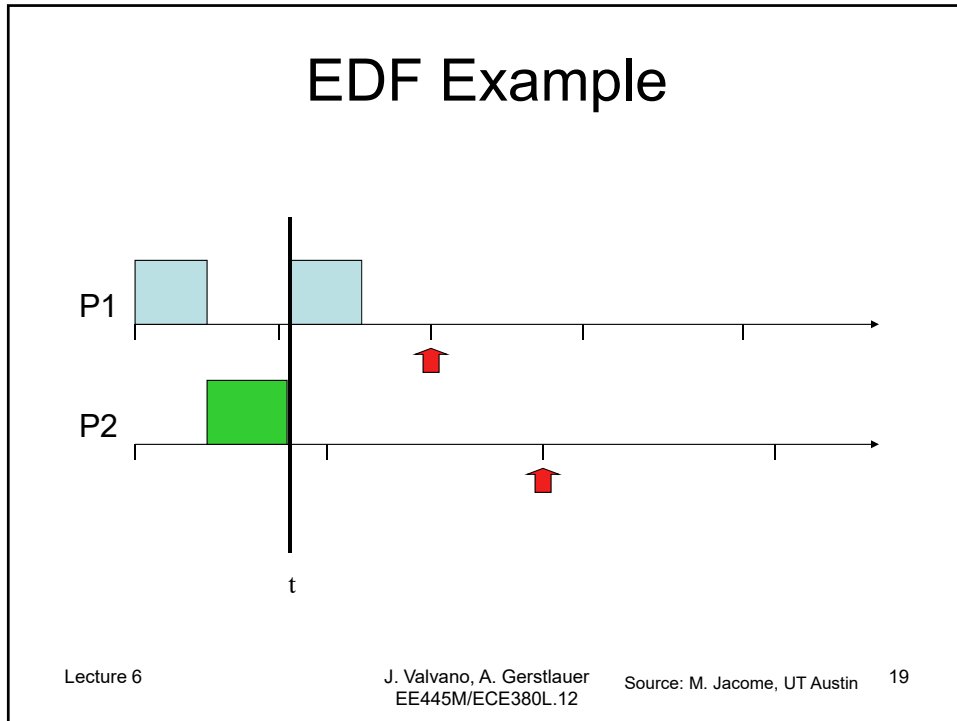
- *Dynamic* priority scheduling scheme
 - Process closest to its deadline has highest priority
- EDF is optimal
 - EDF can use 100% of CPU for worst case
- Expensive to implement
 - On each OS event, recompute priorities and resort tasks

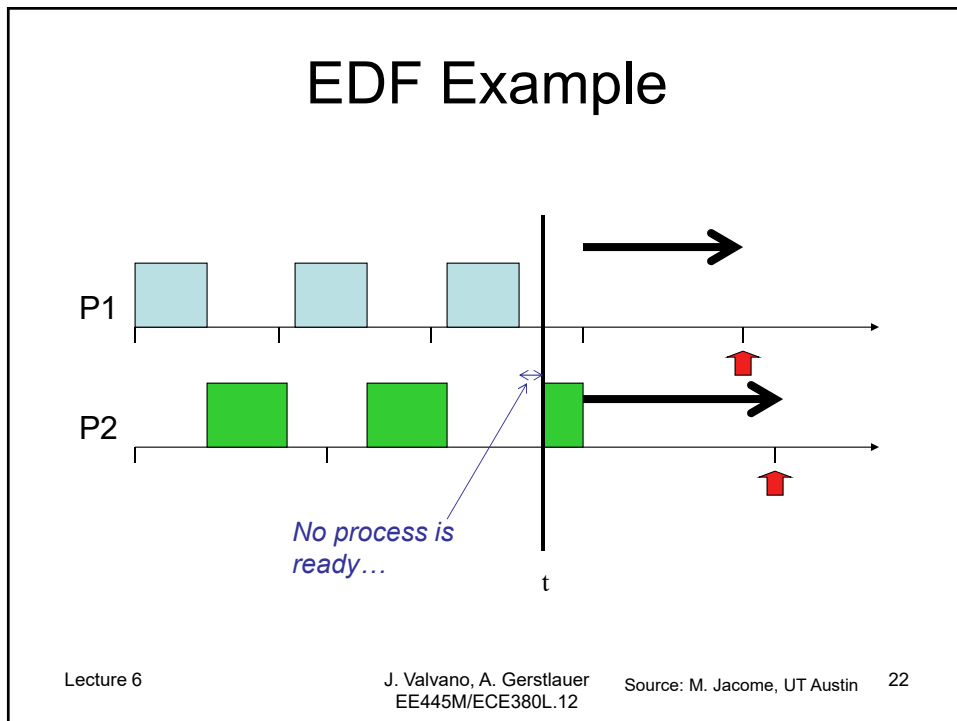
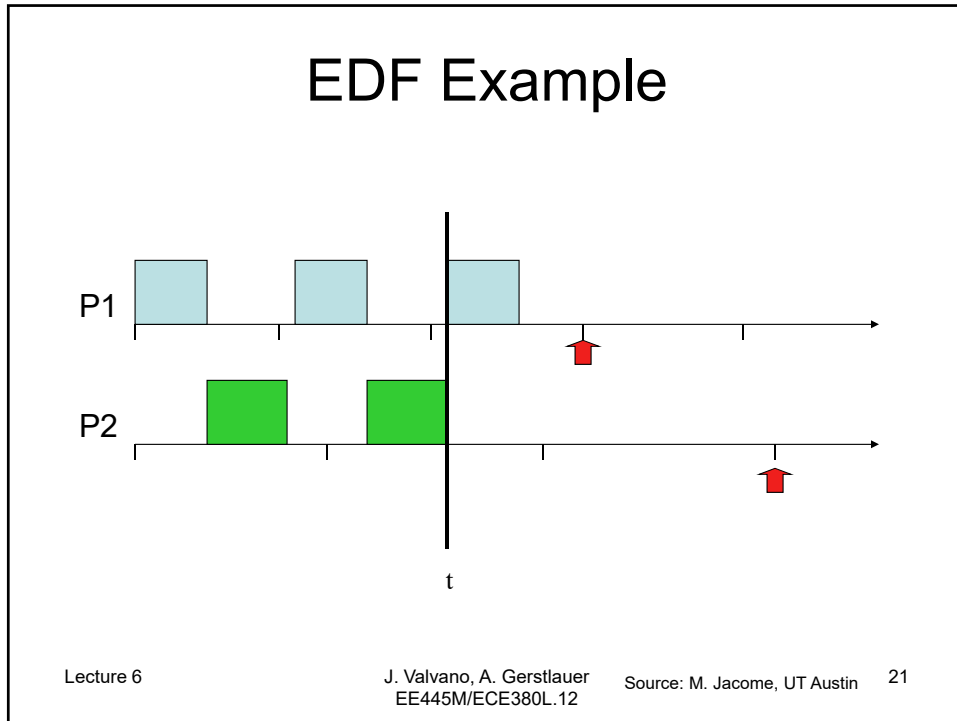
Lecture 6

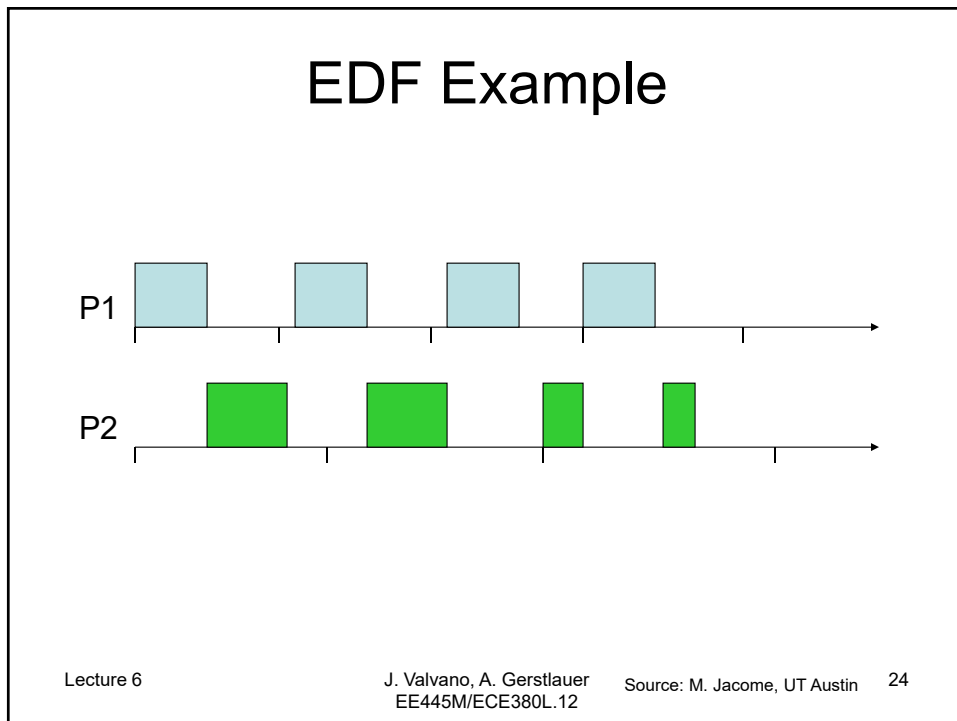
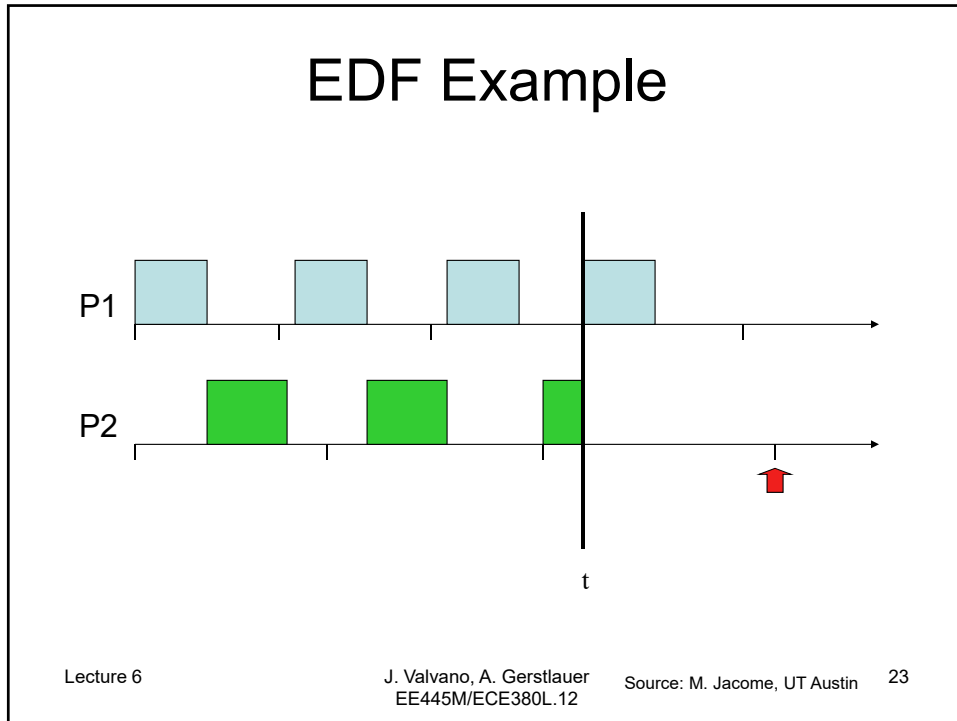
J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

Source: M. Jacome, UT Austin 16









Scheduling Anomalies

The New York Times
Monday, February 16, 2009

Archives

WORLD | U.S. | N.Y. / REGION | BUSINESS | TECHNOLOGY | SCIENCE | HEALTH | SPORTS

Mars Craft Again Halts Transmission

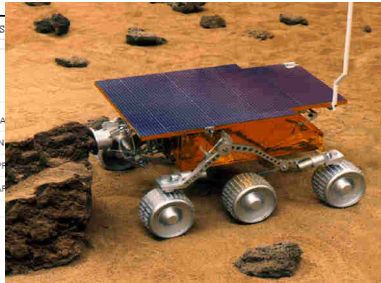
Published: July 15, 1997

The computer aboard the Mars Pathfinder **overloaded and reset itself** early today for the second time in just over three days, interrupting the transmission of a full-color panoramic scene.

The mishap delayed chemical analysis of a tubby rock named Yogi, but no information was lost, and controllers will be able to resume transmission where it was left off, officials said.

Mary Beth Murrill, a spokeswoman for NASA's Jet Propulsion Laboratory, said transmission of the panoramic shot took "a lot of processing power." She likened the data overload to what happens with a personal computer "when we ask it to do too many things at once."

The project manager, Brian Muirhead, said that to prevent a recurrence, controllers would schedule activities one after another, instead of at the same time. It was the second time the Pathfinder's computer had reset itself while trying to carry out several activities at once.



Courtesy NASA/JPL-Caltech

➤ **Priority inversion**

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

25

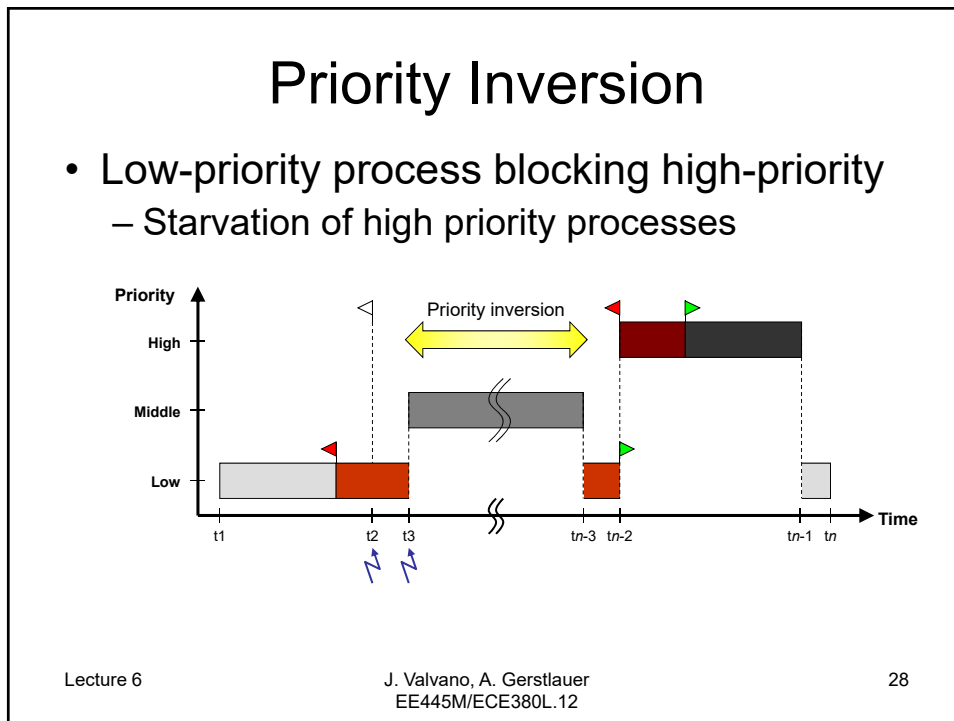
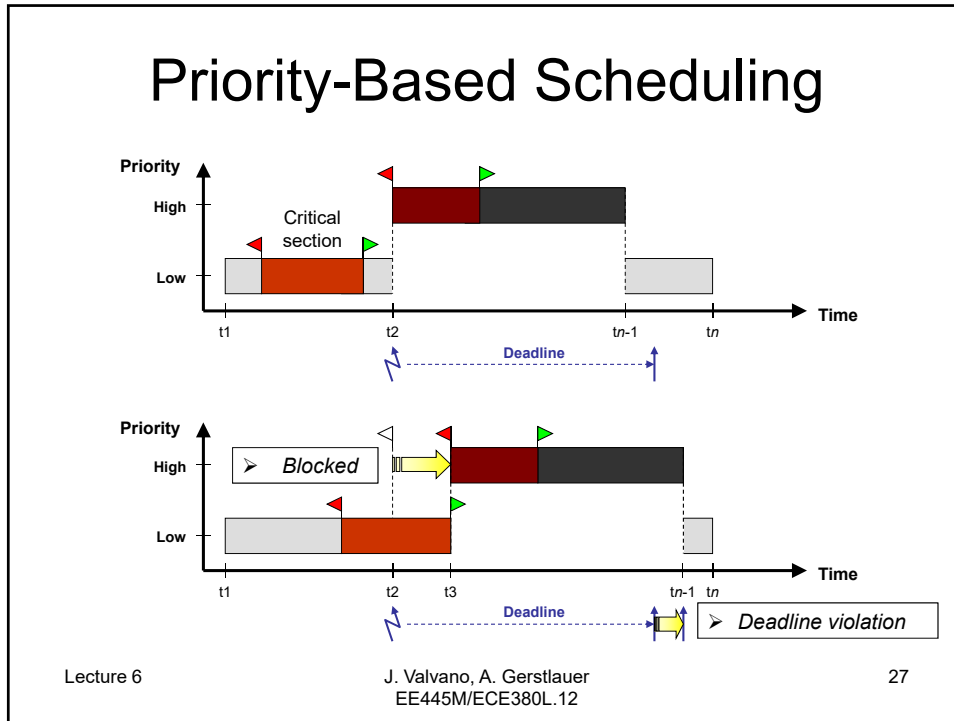
Priority Inversion

- Low-priority process keeps high-priority process from running.
 - Low-priority process grabs resource (semaphore)
 - High-priority device needs resource (semaphore), but can't get it until low-priority process is done.
- Can cause deadlock

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

26



Priority Inversion Solutions

- Avoid preemption in critical sections
 - Interrupt masking
 - Priority Ceiling Protocol (PCP)
 - Priority Inheritance Protocol (PIP)

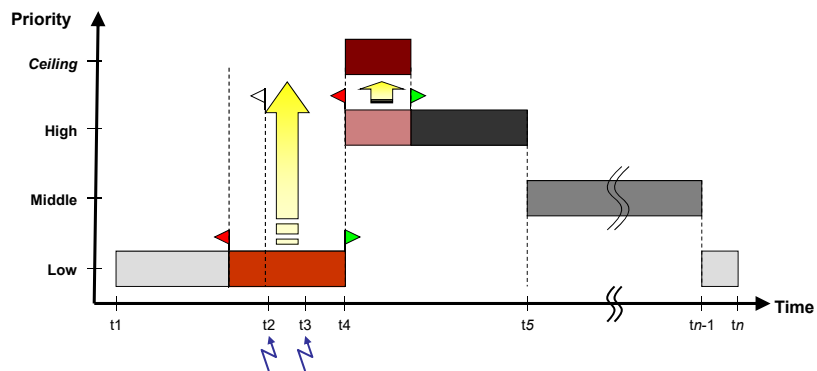
Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

29

Priority Ceiling Protocol (PCP)

- Elevate priorities in critical sections
 - Assign priority ceilings to semaphore/mutex



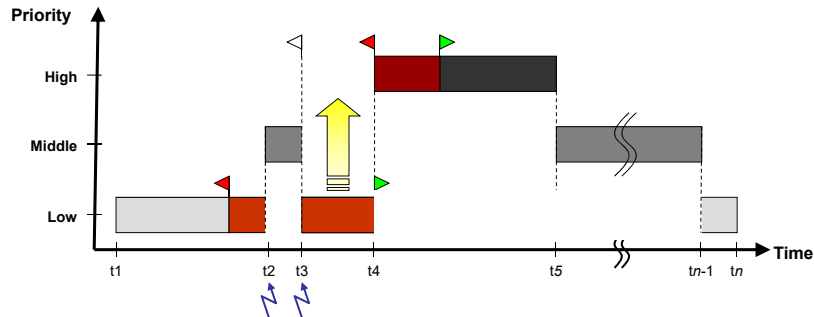
Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

30

Priority Inheritance Protocol (PIP)

- Dynamically elevate only when needed
 - Raise priorities to level of requesting task



Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

31

Fixed Scheduling

- Time-driven scheduler
 - In advance, a priori, during the design phase
 - Thread sequence
 - Allocated time-slices
 - Like
 - Creating the city bus schedule
 - Routing packages through a warehouse
 - Construction project
 - TDMA in communication networks

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

32

Fixed Scheduler Design (1)

- Fundamental principles
 - Gather reliable information about the tasks
 - Build slack into the plan
 - Expect delays
 - Anticipate problems
 - Just in time
- Consider resources required vs. available
 - Processor, memory, I/O channels, data

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

33

Fixed Scheduler Design (2)

- Create a list of tasks to perform
 1. Assign a priority to each task,
 2. Define the resources required for each task,
 3. Determine how often each task is to run, and
 4. Estimate how long each task will require.
- Objectives
 - Guarantee performance (latency, bandwidth)
 - Utilization
 - Maximize profit

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

34

Fixed Scheduler Design (3)

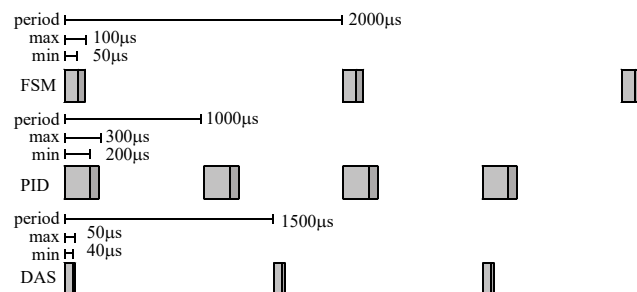
- Design strategy
 - Schedule highest priority tasks first
 - 100% satisfaction guaranteed
 - Then schedule all real-time tasks
 - Shuffle assignments like placing pieces in a puzzle
 - Maximizing objectives
 - The tasks that are not real-time can be scheduled in the remaining slots.

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

35

Fixed Scheduler Example (1)



- Four tasks
 - Finite state machine (**FSM**)
 - Proportional-integral-derivative controller (**PID**)
 - Data acquisition system (**DAS**)
 - Non-real-time task (**PAN**)

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

36

Fixed Scheduler Example (2)

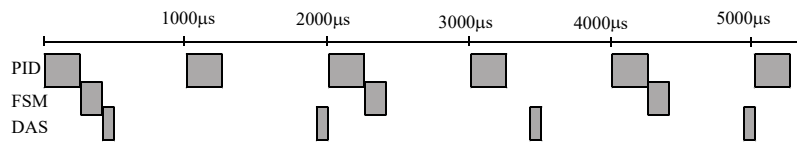
- To guarantee tasks will run on time

- Consider the maximum times

$$\sum_{i=0}^{n-1} \frac{E_i}{T_i} = \sum \frac{100}{2000} + \frac{300}{1000} + \frac{50}{1500} = 0.38 \leq n(2^{1/n} - 1) = 3(2^{1/3} - 1) = 0.78$$

- Design process (critical instant)

- Repeating pattern of least common multiple
- Start with the most frequent (priority) task
- Time-shift the second and third tasks (no overlap)



Lecture 6

Figure 4.17. Repeating pattern to schedule these three real-time tasks.

37

Fixed Scheduler Implementation

- OS_Suspend
 - Cooperatively stops a real-time task
 - Runs a non real-time task
- Timer interrupt
 - Occurs when it is time to run a real-time task
 - Suspends a non-real-time task
 - Runs the next real-time task

```

*****Real-Time Task*****
void Task1(void){ unsigned char in, out;
Task1_Init();      // Initialize
for(;;) {
    OS_Suspend();  // Runs every Mms
    in = Task1_In(); // read input
    out = Task1_Calc(in);
    Task1_Out(out); // send output
}
}

*****Non-Real-Time Task*****
void Task2(void){ unsigned char input;
Task2_Init();      // Initialize
for(;;) {
    input = Task2_In(); // input
    Task2_Out(input); // process
}
}

```

Lecture 6

J. Valvano, A. Gerstlauer
EE445M/ECE380L.12

38

Fixed Scheduler Data Structure

```

struct Node{
    struct Node *Next;           // circular linked list
    TCBType *ThreadPt;         // which thread to run
    unsigned short TimeSlice;   // how long to run it
};
// Thread array
TCBType tcbs[4];
// thread currently running
TCBType *RunPt;

struct Node Schedule[22]={
    { &Schedule[1], ThePID, 300}, // interval    0, 300
    { &Schedule[2], TheFSM, 100}, // interval  300, 400
    { &Schedule[3], TheDAS,  50}, // interval  400, 450
    { &Schedule[4], ThePAN, 550}, // interval  450, 1000
    { &Schedule[5], ThePID, 300}, // interval 1000, 1300
    { &Schedule[6], ThePAN, 600}, // interval 1300, 1900
    { &Schedule[7], TheDAS,  50}, // interval 1900, 1950
    { &Schedule[8], ThePAN,  50}, // interval 1950, 2000
    { &Schedule[9], ThePID, 300}, // interval 2000, 2300
    { &Schedule[10], TheFSM, 100}, // interval 2300, 2400
    { &Schedule[11], ThePAN, 600}, // interval 2400, 3000
    { &Schedule[12], ThePID, 300}, // interval 3000, 3300
    { &Schedule[13], ThePAN, 100}, // interval 3300, 3400
    { &Schedule[14], TheDAS,  50}, // interval 3400, 3450
    { &Schedule[15], ThePAN, 550}, // interval 3450, 4000
    { &Schedule[16], ThePID, 300}, // interval 4000, 4300
    { &Schedule[17], TheFSM, 100}, // interval 4300, 4400
    { &Schedule[18], ThePAN, 500}, // interval 4400, 4900
    { &Schedule[19], TheDAS,  50}, // interval 4900, 4950
    { &Schedule[20], ThePAN,  50}, // interval 4950, 5000
    { &Schedule[21], ThePID, 300}, // interval 5000, 5300
    { &Schedule[0], ThePAN, 700} // interval 5300, 6000
};
// Threads
#define TheFSM &tcbs[0]
#define ThePID &tcbs[1]
#define TheDAS &tcbs[2]
#define ThePAN &tcbs[3]

```

Run timer interrupt
every 1 μ s and switch

Could this be solved with
regular periodic interrupts?

Rate Monotonic
Scheduling?

39

Fixed Scheduling Algorithm

- Find schedule with minimum jitter
- Inputs
 - Period for each task T_i
 - Maximum execution for each task E_i
- Fundamental issues
 - Find the largest Δt , and convert T_i and E_i specifications to integers
 - Find time at which the pattern repeats, least common multiple of T_i

<http://www.ece.utexas.edu/~valvano/EE345M/ScheduleFinder.c>

Example 1

- $T_i = \{1.0\text{ms}, 1.5\text{ms}, 2.5\text{ms}, 3.0\text{ms}\}$, $E_i = 0.1\text{ms}$
 - Time quanta = $\Delta t = 0.1 \text{ ms}$
 - **LCM** of 10, 15, 25 and 30 is 150
 - $E1/T1 + E2/T2 + E3/T3 + E4/T4 = 0.24$
- **ScheduleFinder (10, 15, 25, 30)**
 - Schedule Task A at times $n*10$
 - Schedule Task B at times $n*15 + j$
 - Schedule Task C at times $n*25 + k$
 - Schedule Task D at times $n*30 + l$
 - About $(15)*(25)*(30)=11250$ possible schedules (j,k,l)
 - Slide factors $j=1, k=2, l=3$ to minimize overlap (jitter=0):

```

abcd      a      b  a      c  ab d      a      b  a  c      ab d      a
012345678901234567890123456789012345678901234567890123456789012345678901234
bc a      ab d      a  c  b  a      ab d  c  a      b  a
567890123456789012345678901234567890123456789012345678901234567890123456789
    
```

Example 2

- $T_i = \{0.4\text{ms}, 0.6\text{ms}, 1.0\text{ms}, 1.5\text{ms}\}$, $E_i = 0.1\text{ms}$
 - Time quanta = 0.1ms, pattern repeats every 6ms
 - $E1/T1 + E2/T2 + E3/T3 + E4/T4 = 0.58$
- **ScheduleFinder (4, 6, 10, 15)**
 - Schedule Task A at times $n*4$
 - Schedule Task B at times $n*6 + 1$
 - Schedule Task C at times $n*10 + 1$
 - Schedule Task D at times $n*15 + 14$
 - Jitter = 5

```

abC a  ba  cabd a  bac  ab  ad baC  ab  ac baD  ab c
01234567890123456789012345678901234567890123456789012345678901
a  ba  d
23456789
    
```

Lecture 6 Red means one time quanta late
 Blue means two time quanta late (or one time quanta early)