Issues in Building Real-Time Applications

郭大維 教授
ktw@cs.ccu.edu.tw
即時及嵌入式系統實驗室
(Real-Time and Embedded System Laboratory)
國立臺灣大學資訊工程系

Introduction to Real-Time Systems

- Checklist
  - What is a real-time system?
  - What is the way usually used to classify real-time tasks?
  - What are the issues and research for real-time systems?
  - Is there any misconception about real-time computing?
  - Is our current software development environments suitable to time-critical systems?
  - What kinds of software architectures are adopted or considered in current time-critical systems?

Introduction to Real-Time Systems

- What is a real-time system?

  Any system where a timely response by the computer to external stimuli is vital!

- Examples:
  - multimedia systems, virtual reality, games.
  - avionics, air traffic control, nuclear power plant
  - stock market, trading system, information access, etc.

What is a Real-Time System?

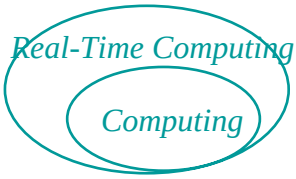
- Does the definition make every computer a real-time computer?

  Yes! It is if we need some response from a computer within a finite time!!

- Category of Real-Time Systems:
  - Hard Real-Time Systems - catastrophic if some deadlines are missed.
  - Soft Real-Time Systems - otherwise.

Issues in Real-time Computing

- The field of real-time computing is especially rich in research problems!

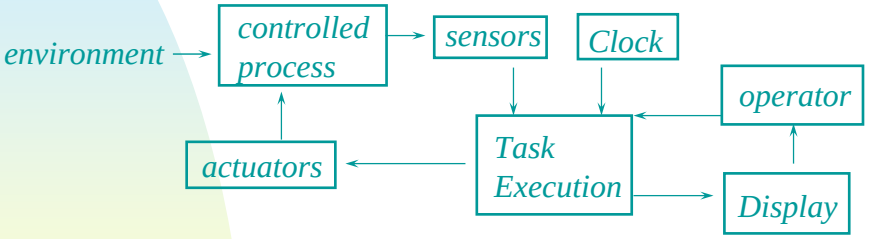


For example, CPU scheduling of tasks with different criticality!

- However, real-time computing systems often differ from their counterparts in two ways:
  - More specific in their applications.
  - More drastic for their failures.

Structure of A Real-Time System - An Example

- A control system



- Rates - sensors & actuators, peripheral, control program
- Phases - takeoff, cruise, and landing, etc.

Task Classes

- Ways to classify real-time tasks:
  - Predictability of their arrivals.
    - Periodic tasks have regular arrival times.
    - Aperiodic tasks have irregular arrival times.
      - bounded inter-arrival time -> Sporadic tasks.
  - Criticality - consequences of non-timely executions.
    - Critical tasks should have timely executions
      - Most of them are hard real-time transactions
    - Non-critical tasks are usually soft real-time tasks
      - minimize miss ratio, minimize response time, maximize values contributed to the system, etc.

Issues and Research

- Software engineering
  - System architecture, e.g., event-driven, time-line, time-driven, object-oriented, etc.
  - Network architecture, e.g., topology, predictability, and controllability.
  - Fault-tolerance and reliability evaluation, etc.
  - Tools for prototyping, simulation, code synthesis.
- Operating systems
  - Task assignment and scheduling
  - Communication protocols
  - Failure management and recovery
  - Clock Synchronization, etc.

Issues and Research

- Programming languages
  - Better control over timing
  - Proper interface to special-purpose devices
- Database systems
  - Concurrency Control
  - Failure recovery
  - Availability
  - Query Optimization, etc.
- Specification and verification
  - Expressiveness and complexity

Issues for Programming Environments

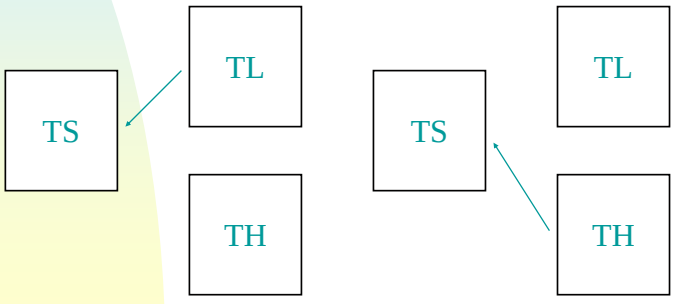
- Loop size, timer granularity, imprecise timer, sleep(), multi-programming, etc.
- Sequential programs, parallel programs, timely programs.
- Client-server priority assignments - priority inversion.
- Verification, analysis, and testing.

Issues for Programming Environments

```
Loop
    ……
    Sensor();
    ……..
    computation……
    ……..
    t = time();
    SleepTime := ReadyTime + PERIOD - t;
    ReadyTime = ReadyTime + PERIOD;
    Sleep(SleepTime);
EndLoop;
```


Issues for Programming Environments

- The priority assignment for a Server TS?
  - Processes TH and TL
  - Priority Inversion



Software Architectures and Fault Tolerance Issues for Real-Time Applications

郭大維 教授
ktw@cs.ccu.edu.tw
即時及嵌入式系統實驗室
(Real-Time and Embedded System Laboratory)
國立臺灣大學資訊工程系

Source: C. Douglass Locke & Farnam Jahanian, RTCSA'96 Talks Presentation.

Software Architectures for Real-Time Applications

- Popular architectures:
  - Timeline (i.e., cyclic executive or frame-based)
  - Event-driven (with both periodic and aperiodic activities)
  - Pipelined
  - Client-Server
- Impacts
  - performance and life-cycle cost
  - critical design decisions such as synchronization and exceptions.
- No restriction on parallel processing.

Timeline or Cyclic Executive

- A major cycle consists of a non-repeating set of minor cycles
  - Operations are implemented as procedures.
  - The timer calls each procedure in the list.
- No concurrency exists.
- Very high life-cycle cost but very predictable in run-time behavior!

Event-Driven

- Characteristics:
  - Trigger schedulable tasks by I/O completion and timer events.
- Task Priority:
  - Determined by timing constraints, e.g., RMS, or by semantic importance.
- Ways to avoid synchronization is needed for predictable response.
- Processor utilization is preserved for aperiodic events for response predictability.
- Prone to event shower! Good for systems with spare computation power!

Pipelined

- Characteristics:
  - Trigger schedulable tasks by I/O completion, timer events, and inter-task messages.
  - The system can be described as a set of pipelines of task invocations.
- Task priority
  - Increasing task priorities in a unidirectional pipeline will minimize message queue buildup.
  - Equal task priority setup is normal for bi-directional pipelines.
- Prone to event shower! Good for systems with spare computation power!

Client-Server

- Characteristics:
  - Trigger schedulable tasks by I/O completion, timer events, and inter-task messages.
  - Control flow for an event stays at a node while data flow is distributed.
- Task priority
  - Priority inheritance is used ideally. Practically task priorities are set equally, and message priorities are used instead to avoid bottlenecks.
- More message exchange but significantly easier in debugging than pipelined systems.

Fault Tolerance

- Definition:
  - A real-time fault-tolerance system is a system that can deliver its service even in the presence of faults.
- Timeliness versus Fault Tolerance
  - Possible Faults: Hardware/Software errors, violation of timing constraints because of the “environment”.

Fault Tolerance

- Use redundancy to detect errors and mask failures
  - Space Redundancy: replication of physical devices.
  - Time Redundancy: repetition of a computation or communication.
  - Information Redundancy: specific encoding scheme, e.g., parity bit.

Fault Tolerance

- Real-time systems
  - Time is scare -> methods should trade space/information redundancy for time.
- Possible Structures:
  - Active replicas:
    - Each request is processed by all replicas, and their results are “combined” to mask faults.
  - Passive replicas:
    - One primary and several backups.
    - Once the primary fails, a backup takes over.
  - Cooperating replicas/objects:
    - A client makes a request through a “broker” mechanism.

Introduction to Real-Time Process Scheduling

郭大維 教授
ktw@cs.ccu.edu.tw
即時及嵌入式系統實驗室
(Real-Time and Embedded System Laboratory)
國立臺灣大學資訊工程系

Introduction to Real-Time Process Scheduling

- Q: Many theories and algorithms in real-time process scheduling seem to have simplified assumptions without direct solutions to engineers' problems. Why should we know them?
- A:
  - Provide insight in choosing a good system design and scheduling algorithm.
  - Avoid poor or erroneous choices.

Introduction to Real-Time Process Scheduling

- Checklist
  - ⊕ What do we really know about the rate monotonic (RM) and the earliest deadline first (EDF) scheduling?
  - ⊕ What is known about uniprocessor real-time scheduling problems?
  - ⊕ What is known about multiprocessor real-time scheduling problems?
  - ⊕ What task-set characteristics cause NP-hard?
  - ⊗ What is the impact of overloads on the scheduling results?
  - ⊗ What do we really know about theories for off-line schedulability such as the rate monotonic analysis?

Introduction to Real-Time Process Scheduling

Introduction to Real-Time Process Scheduling

Uniprocessor Process Scheduling

- *Rate Monotonic Scheduling*
- *Earliest Deadline First Scheduling*
- *Priority Ceiling Protocol*
- *Important Theories*

Reading: Stankovic, et al., "Implications of Classical Scheduling Results for Real-Time Systems," IEEE Computer, June 1995, pp. 16-25.
Krishna and Shin, "Real-TimeSystems," McGRAW-HILL, 1997.

Process Model

- Periodic process
  - each periodic process arrives at a regular frequency - a special case of demand.
    - **r**: ready time, **d**: relative deadline, **p**: period, **c**: maximum computation time.
  - For example, maintaining a display
- Sporadic process
  - An aperiodic process with bounded inter-arrival time p.
  - For example, turning on a light
- Other requirements and issues:
  - process synchronization including precedence and critical sections, process value, etc.

Performance Metrics

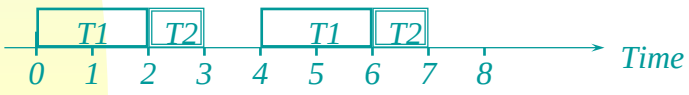
- Metrics for hard real-time processes:
  - Schedulability, etc.
- Metrics for soft real-time processes:
  - Miss ratio
  - Accumulated value
  - Response time, etc.
- Other metrics:
  - Optimality, overload handling, mode-change handling, stability, jitter, etc.
  - Combinations of metrics.

Basic definitions:

- Preemptive scheduling: allows process preemptions. (vs non-preemptive scheduling)
- Online scheduling: allocates resources for processes depending on the current workload. (vs offline scheduling)
- Static scheduling: operates on a fixed set of processes and produces a single schedule that is fixed at all time. (vs dynamic scheduling)
- Firm real-time process: will be killed after it misses its deadline. (vs hard and soft real-time)
- Fixed-priority scheduling: in which the priority of each process is fixed for any instantiation. (vs dynamic-priority scheduling)

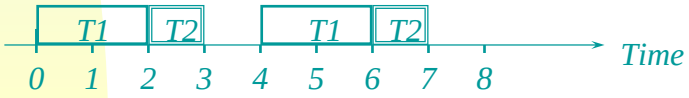
Rate Monotonic Scheduling Algorithm

- Assumptions:
  - all periodic fixed-priority processes
  - relative deadline = period
  - independent process - no non-preemptable resources
- Rate Monotonic (RM) Scheduling Algorithm
  - RM priority assignment: priority ~ 1/period.
  - preemptive priority-driven scheduling.
- Example: T1 (p1=4, c1=2) and T2 (p2=5, c1=1)



Rate Monotonic Scheduling Algorithm

- Critical Instant [1]
  - An instant at which a request of the process have the largest completion/response time.
  - An instance at which the process is requested simultaneously with requests of all higher priority processes
- Usages
  - Worst-case analysis
  - Fully utilization of the processor power
  - Example: T1 (p1=4, c1=2) and T2 (p2=5, c1=2)



[1] *Liu and Layland, "Scheduling Algorithms for multiprogramming in a hard real-time Environment," JACM, vol. 20, no. 1, January 1973, pp. 46-61.*

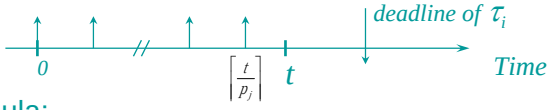
Rate Monotonic Scheduling Algorithm

- Schedulability Test:
  - A sufficient but not necessary condition
  - Achievable utilization factor α
    - of a scheduling policy P -> any process set with total utilization factor $\sum \frac{c_i}{p_i}$ no more than α is schedulable.
  - Given n processes, $\alpha = n\left(2^{1/n} - 1\right)$
- Stability:
  - Let processes be sorted in RM order. The ith process is schedulable if $\sum_{j=1}^{i} \frac{c_j}{p_j} \le i\left(2^{1/i} - 1\right)$
- An optimal fixed priority scheduling algorithm

Rate Monotonic Scheduling Algorithm

- Rate Monotonic Analysis (RMA) [2]
  - Basic Idea:

    Before time t after the critical instance of process τ_i, a high priority process τ_j may request $c_j\left\lceil \frac{t}{p_j} \right\rceil$ amount of computation time.



  - Formula:

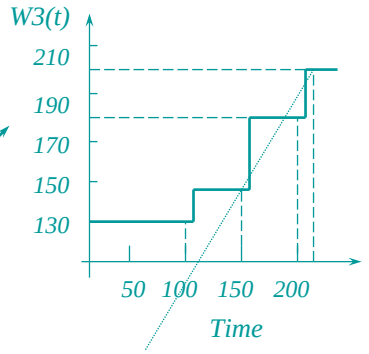
$$W_i(t) = \sum_{j=1}^{i} c_j \left\lceil \frac{t}{p_j} \right\rceil \le t \le d_i \quad \text{for some } t \text{ in } \{kp_j \mid j = 1,\ldots,i;\ k = 1,\ldots,\lceil p_i / p_j \rceil\}$$

  - A sufficient and necessary condition and many extensions...

[2] *Sha, "An Intorduction to Rate Monotonic Analysis," tutorial notes, SEI, CMU, 1992*

Rate Monotonic Scheduling Algorithm

- A RMA Example:
  - T1(20,100), T2(30,150), T3(80, 210), T4(100,400)
  - T1
    - c1 <= 100
  - T2
    - c1 + c2 <= 100 or
    - 2c1 + c2 <= 150
  - T3
    - c1 + c2 + c3 <= 100 or
    - 2c1 + c2 + c3 <= 150 or
    - 2c1 + 2c2 + c3 <= 200 or
    - 3c1 + 2c2 + c3 <= 210
  - T4
    - c1 + c2 + c3 + c4 <= 100 or
    - 2c1 + c2 + c3 + c4 <= 150 or
    - ....

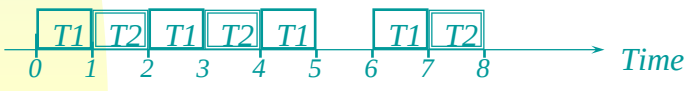


Rate Monotonic Scheduling Algorithm

- RM was chosen by
  - Space Station Freedom Project
  - FAA Advanced Automation System (AAS)
- RM influenced
  - the specs of IEEE Futurebus+
- RMA is widely used for off-line analysis of time-critical systems.

Earliest Deadline First Scheduling Algorithm

- Assumptions (similar to RM):
  - all periodic dynamic-priority processes
  - relative deadline = period
  - independent process - no non-preemptable resources
- Earliest Deadline First (EDF) Scheduling Algorithm:
  - EDF priority assignment: priority ~ absolute deadline. i.e.,arrival time **t** + relative deadline **d**.
  - preemptive priority-driven scheduling
- Example: T1(c1=1, p1=2), T2(c2=2, p2=7)



Earliest Deadline First Scheduling Algorithm

- Schedulability Test:
  - A sufficient and necessary condition
  - Any process set is schedulable by EDF iff

$$\sum_{j=1}^{i} \frac{c_j}{p_j} \leq 1$$

- EDF is optimal for any independent process scheduling algorithms
- However, its implementation has considerable overheads on OS's with a fixed-priority scheduler and is bad for (transiently) overloaded systems.

Priority Ceiling Protocol

- Assumptions (as the same as RM for the first two):
  - all periodic fixed-priority processes
  - relative deadline = period
  - Non-preemptable resources guarded by semaphres
- Basic Ideas and Mechanisms:
  - Bound the priority inversions by early blocking of processes that could cause them, and
  - Minimize a priority inversion's length by allowing a temporary rise in the blocking process's priority.
- Contribution of the Priority Ceiling Protocol
  - Efficiently find a suboptimal solution with a clever allocation policy, guaranteeing at the same time a minimum level of performance.

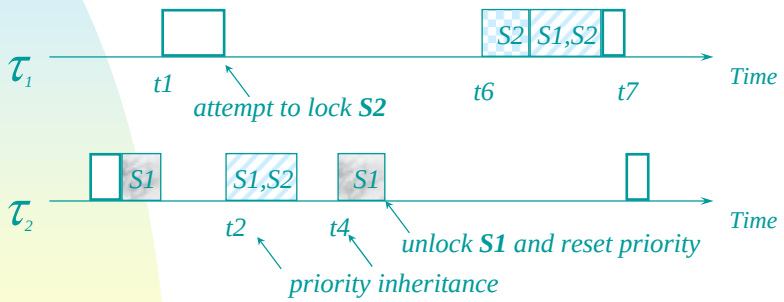
Priority Ceiling Protocol

- Pre-requirements: nested critical sections!
- Priority Ceiling Protocol (PCP):
  - Define a semaphore's priority ceiling as the priority of the highest priority process that may lock the semaphore.
  - Lock request for a semaphore is granted only if the requesting process's priority is higher than the ceiling of all semaphores concurrently locked by other processes.
  - In case of blocking, the task holding the lock inherits the requesting process's priority until it unlocks the corresponding semaphore. (Def: priority inheritance)

[1] *Sha, Rajkumar, and Lehoczky, "Priority Inheritance Protocols: an Approach to Real-Time Synchronization," IEEE Transactions on computers, Vol. 39, No. 9, Sept. 1990, pp. 1,175-1,185.*

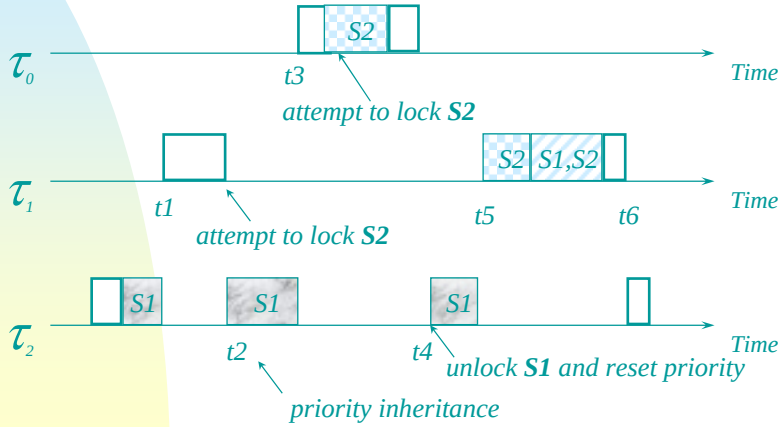
Priority Ceiling Protocol

- A PCP Example: avoid deadlock



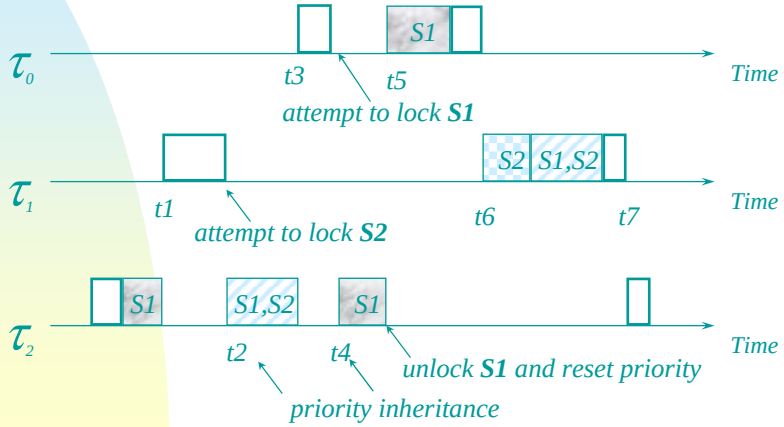
Priority Ceiling Protocol

- A PCP Example: avoid chain blocking



Priority Ceiling Protocol

- A PCP Example: one priority inversion



Priority Ceiling Protocol

- Important Properties:
  - A process is blocked at most once before it enters its critical section.
  - PCP prevents deadlocks.
- Schedulability Test of τ_i
  - worst case blocking time B_i - an approximation!
    - $S_j = \{ S \mid \text{semaphore } S \text{ is accessed by } \tau_j \}$
    - $BS_i = \{ \tau_j \mid j > i \ \& \ Max_{(s \text{ in } Sj)} (\text{ceiling}(s)) >= \text{priority}(\tau_i) \}$
    - $B_i = Max_{(\tau j \text{ in } BSi)} |\text{critical section}|$
  - Let processes be sorted in the RM priority order

$$\sum_{j=1}^{i-1} \left(\frac{c_j}{p_j}\right) + \frac{c_i + B_i}{p_i} \le i\left(2^{1/i} - 1\right)$$

Priority Ceiling Protocol

- Variations of PCP:
  - Stack Resource Policy - not permitted to start unless resources are all available.
    - multi-units per resource
    - dynamic and fixed priority assignments
  - Dynamic Priority Ceiling Protocol
    - extend PCP into an EDF scheduler.

[2] *Baker, "Stack-Based Scheduling of Real-Time Processes," J. Real-Time Systems, Vol. 3, No. 1, March 1991, pp. 67-99.*

[3] *Chen and Lin, "Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-time Systems," J. Real-Time Systems, Vol. 2, No. 4, Nov. 1990, pp. 325-340.*

Introduction to Real-Time Process Scheduling

Multiprocessor Process Scheduling

- *Important Theories*
- *Basic Approaches*

Reading: Stankovic, et al., "Implications of Classical Scheduling Results for Real-Time Systems," IEEE Computer, June 1995, pp. 16-25.
Krishna and Kang, "Real-TimeSystems," McGRAW-HILL, 1997.

Multiprocessor Process Scheduling

- Checklist
  - ⊕ Understand the boundary between polynomial and NP-hard problems to provide insights into developing useful heuristics.
  - ⊕ Understand the fundamental limitations of on-line algorithms to create robust system and avoid misconceptions and serious anomalies.
  - ⊕ Know the basic approaches in solving multiprocessing scheduling
- Remark: It is the area which we have very limited knowledge because of its complexity and our minimal experiences with multiprocessor systems.

Nonpreemptive Multiprocessor Scheduling

- Important Theorems[1]:
  - Conditions:
    - ☞ Single deadline, identical processors, ready at time 0
  - Theorems: ("_"-marked items causes NP-completeness!)

Processors	Resources	Ordering	Computation Time	Complexity
2	0	Arbitrary	Unit	Polynomial[2]
2	0	Independent	Arbitrary	NP-Complete[3]
2	0	Arbitrary	1 or 2 units	NP-Complete[3]
2	1	Forest	Unit	NP-Complete[3]
3	1	Independent	Unit	NP-Complete[3]
N	0	Forest	Unit	Polynomial[4]
N	0	Arbitrary	Unit	NP-Complete[5]

1. Stankovic, et al., "Implications of Classical Scheduling Results for Real-Time Systems," IEEE Computer, June 1995, pp. 16-25.
2. Coffman and Graham, "Optimal Scheduling for Two-Processor Systems," ACTA Information, 1, 1972, pp.200-213.
3. Garey and Johnson, "Complexity Bounds for Multiprocessor Schedulingwith Resource Constraints," SIAM J. Computing, Vol. 4, No.3, 1975, pp. 187-200.
4. Hu, "Parallel Scheduling and Assembly Line Problems," Operating Research,, 9, Nov. 1961, pp. 841-848.
5. Ullman, "Polynomial Complete Scheduling Problem," Proc. fourth Symp. Operating System Principles, ACM, 1973, pp. 96-101.

Preemptive Multiprocessor Scheduling

- Theorem of McNaughton in 1959.
  - Goal: Compare preemption and non-preemption.
  - Conditions:
    - ☞ identical processors.
  - Theorem 0: Given the metric to minimize the weighted sum of completion times, i.e., Sum(w_jc_j), there exists a schedule with no preemption for which the performance is as good as for any schedule with a finite number of preemptions.
  - Note: It is NP-hard to find an optimal schedule! If the metric is to minimize the sum of completion times, the shortest-processing-time-first greedy approach is optimal.

McNaughton, "Scheduling with Deadlines and Loss Functions," Management Science, Vol. 6, No. 1, Oct. 1959, pp.1-12.

Preemptive Multiprocessor Scheduling

- Theorem of Lawler in 1983.
  - Goal: Show that heuristics are needed for real-time multiprocessor scheduling.
  - Conditions:
    - identical processors, different deadlines for processes.
  - Theorem 0: The multiprocessing problem of scheduling P processors with process preemption allowed and with minimization of the number of late processes is NP-hard.

Lawler, "Recent Results in the Theory of Machine Scheduling," Mathmatical Programming: The state of the Art, A. Bachen et al., eds., Springer-Verlag, New York, 1983, pp. 202-233.

Preemptive Multiprocessor Scheduling

- Theorems of Mok in 1983
  - Goal:Understand the limitations of EDF.
  - Conditions:
    - different ready times.
  - Theorem 0: Earliest-deadline-first scheduling is not optimal in the multiprocessor case.
  - Example, T1(c=1,d=1), T2(c=1,d=2), T3(c=3,d=3.5), two processors.
  - Theorem 1: For two or more processors, no deadline scheduling algorithm can be optimal without complete a priori knowledge of deadlines, computation times, and process start times.

A.K. Mok, "Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment," Ph.D. Thesis, Dept. of Electrical Engineering and Computer science, MIT, May 1983.

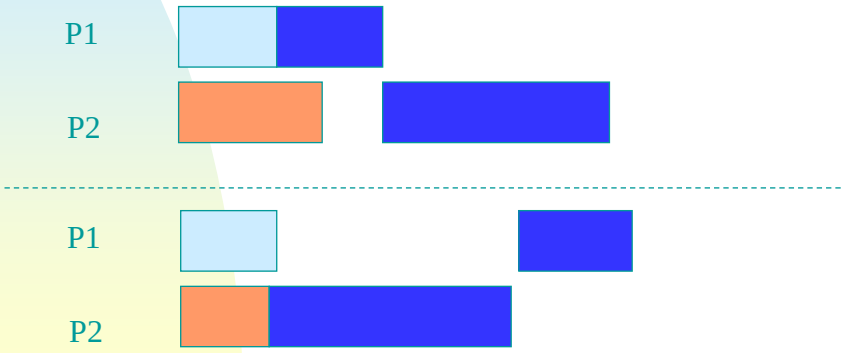
Multiprocessor Anomalies

- Theorem of Graham in 1976.
  - Goal:Notice anomaly and provide better design.
  - Conditions;
    - A set of processes is optimally scheduled on a multiprocessor with some priority order, fixed execution times, precedence constraints, and a fixed number of processors.
  - Theorem 0: For the stated problem, changing the priority list, increasing the number of processors, reducing execution times, or weakening the precedence constraints can increase the schedule length.

R. Graham, "Bounds on the Performance of Scheduling Algorithms," Computer and Job Shop Scheduling theory, E.G. Coffman, ed., John Wiley and Sons, 1976, pp. 165-227.

Multiprocessor Anomalies

- An Example



Multiprocessor Scheduling - Contemporary Approach

- Motivation:
  - The multiprocessor scheduling problem is NP-hard under any but the most simplifying assumptions.
  - The uniprocessor scheduling problem is usually tractable.
- Common Approach - 2 Steps
  - Assign processes to processors
  - Run a uniprocessor scheuling algorithm on each processor.
- Metrics:
  - Minimize the number of processors, fault tolerance, etc.

Multiprocessor Scheduling - Contemporary Approach

- However, the process assignment problem is again NP-hard in most cases.
- Heuristics:
  - Utilization balancing - balance workload of processors.[1]
  - Next-fit algorithm - used with RM. [2]
  - Bin-packing algorithm - set with a threshold and used with EDF [3], etc.
- Other considerations:
  - precedence constraints, dynamic overload handling, etc.

1. J.A. Bannister and K.S. Trivedi, "Task Allocation in Fault-Tolerance Distributed systems," Acta Informatica 20:261-281, 1983.
2. S. Davari and S.K. Dhall, "An On Line Algorithm for Real-Time Tasks Allocation," IEEE Real-Time Systems Symposium, 1986, pp.194-200, Dhall's Ph.D. thesis, UI.
3. D.S. Johnson, Near-Optimal Bin-Packing Algorithms," Ph.D. thesis, MIT, 1974.

Multiprocessor Scheduling

- Current Research
  - Classification: Migration(/Partition) & Static or Dynamic Priorities
  - Most Recent Results:
    - Utilization Bound = 42% by a bin-packing partitioning approach (JRTS, 1999)
    - Utilization Bound = 37.482% by RM-US – processes with a utlization > bound is given the highest priority; otherwise RM is adopted.
    - Utilization Bound = m – [(m-1)*Umax] if Umax <= 0.5, where Umax = max Ui. Or Utilization Bound = (m+1)/2+Umax if Umax > 0.5 – M-CBS (RTAS02)
    - Utilization Bound = 75% - EZDL (to appear)

Papers to Study

- J. Stankovic, M. Spuri, M.D. Natale, G.C. Buttazo,"Implications of Classical Scheduling Results for Real-Time Systems," IEEE Computer, 1995
- C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environments," Journal of ACM, 1973.
- L. Sha, R. Rajkumar, J.P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," IEEE Transactions on Computers, 1990.
- http://140.112.28.119

Papers to Study

- A.K. Mok, “The Design of Real-Time Programming Systems Based on Process Models,” IEEE Real-Time Systems Symposium, Dec 1994.
- T.W. Kuo, Y.H. Liu, K.J. Lin, “Efficient On-Line Schedulability Tests for Priority Driven Real-Time Systems," the IEEE Real-Time Technology and Applications Symposium, June 2000.
- A.K. Mok, “A Graph-Based Computation Model for Real-Time Systems,” IEEE International Conference on Parallel Processing, Aug 1985.