Real Time Systems

Subra Ganesan

Professor, Oakland University, Rochester, MI 48309

Rochester, MI 48309

Phone: (248) 370 2206

Fax: (248) 370 4625

Email: Ganesan@oakland.edu

Web: www.secs.oakland.eud/~ganesan

Abstract

• Real time embedded systems are used in many

applications widely.

• Theoretical analysis, Design , Use of advanced design tools, Use of the latest real time software, Performance monitoring,

Simulation and testing of real time embedded systems in a systematic way are vital for the future.

Introduction

Any system where a timely response by the computer to external input/condition is vital is a real-time system.

Timely: Meet the deadline.

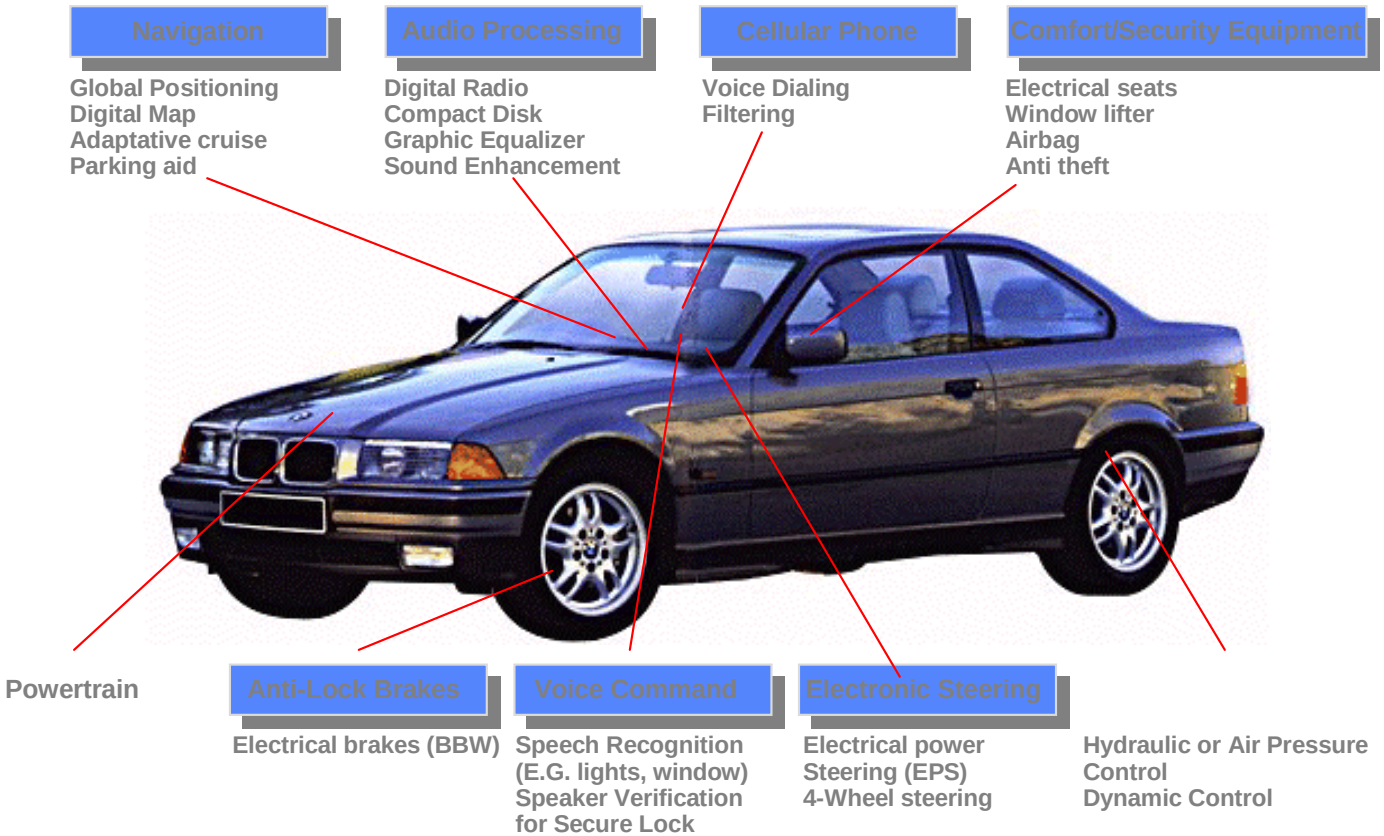
Deadline: Hard and soft

Complete task: Accurately or estimate within deadline.

Hard Real-Time Systems: Aircraft,Nuclear Reactor control.

Soft Real-time Systems: Multimedia,Internet access**.**

Which are the Hard/Soft Real Time Systems ?



A Car and Driver Example

Driver is the real time controller

Car is the controlled process

Road is the operating environment

Mission is to reach the destination

Performance Measure:

1)Time to reach the destination under various road conditions

2)Safety of the driver even if the destination is not reached

Task deadlines are not constants ,varies with the operating environment.

Writing formal specification and relating them is difficult.

Issues in Real-time Computing

Research areas cover: Computer architecture, fault tolerant computers, networks, embedded systems, standards, digital communication, operating system clock synchronization,..

Example:

Task-Scheduling:For a normal system the goal is fairness to all tasks. Round-robin scheduling.

For a RT system: Meet the deadlines for critical and high priority tasks is the goal.

Task-Execution time should be predictable in RT system. For a cache based RT system memory access time varies.

Structure of a Real-time System

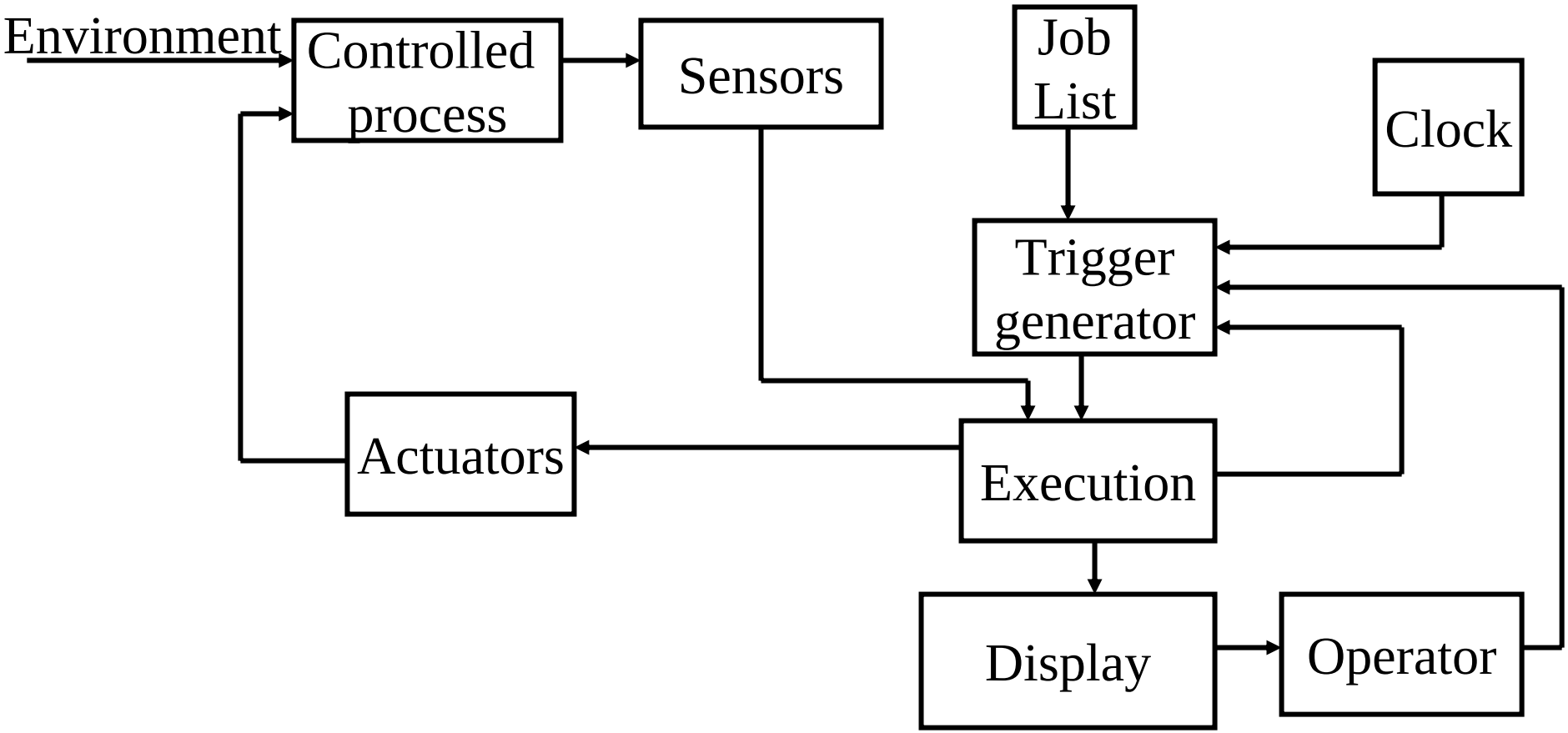


Figure: A Schematic Diagram of a real-time system

Task classes

- Periodic and aperiodic tasks

  Aperiodic tasks with a bounded inter-arrival time are called sporadic tasks
- Critical and non-critical tasks.

OPERATING SYSTEM ISSUES

- Task assignment and scheduling
- Communications protocols
- Failure management and recovery
- Clock synchronization algorithms

Other issues

- Programming languages
- Databases
- Performance measure

Characteristics of RTS

Performance :

Validation process includes

(1) Checking design correctness using formal and informal methods,

(2) Characterizing performance

(3) Reliability

Choosing performance measure is crucial.

e.g : average time, peak time or variance.

Clock cycle may be misleading.

For soft RT, degraded response time is acceptable.

Traditional Performance Measures

- **Reliability** is the probability the system will not fail in an interval
- **Availability** is the fraction of time for which the system is up.
- **Throughput** is the average number of instructions per unit time the RTS can process.
- **Capacity Reliability(CR):** Interval time when reliability is high.
- **Expected capacity survival time** is the expected time for CR to drop to specific value.
- Mean computation before failure
- Computational reliability
- Computational availability
- Per formability
- Accomplishment levels
- Cost functions for hard deadlines: Cost Vs Response times

Definition of Real Time System

- There are many interpretation of a real time system; however, they all consider the notion of **response time** – the time take for the system to generate output from some associated input.

  "Any information processing activity or system which has to respond to externally generated input stimuli within a finite and specified period."

  "A real-time system is a system that is required to react to stimuli from the environment (including the passage of physical time) within time intervals dictated by the environment."
- The **correctness of a real-time system** depends not only on the logical result of the computation, but also on the time at which the results are produced.

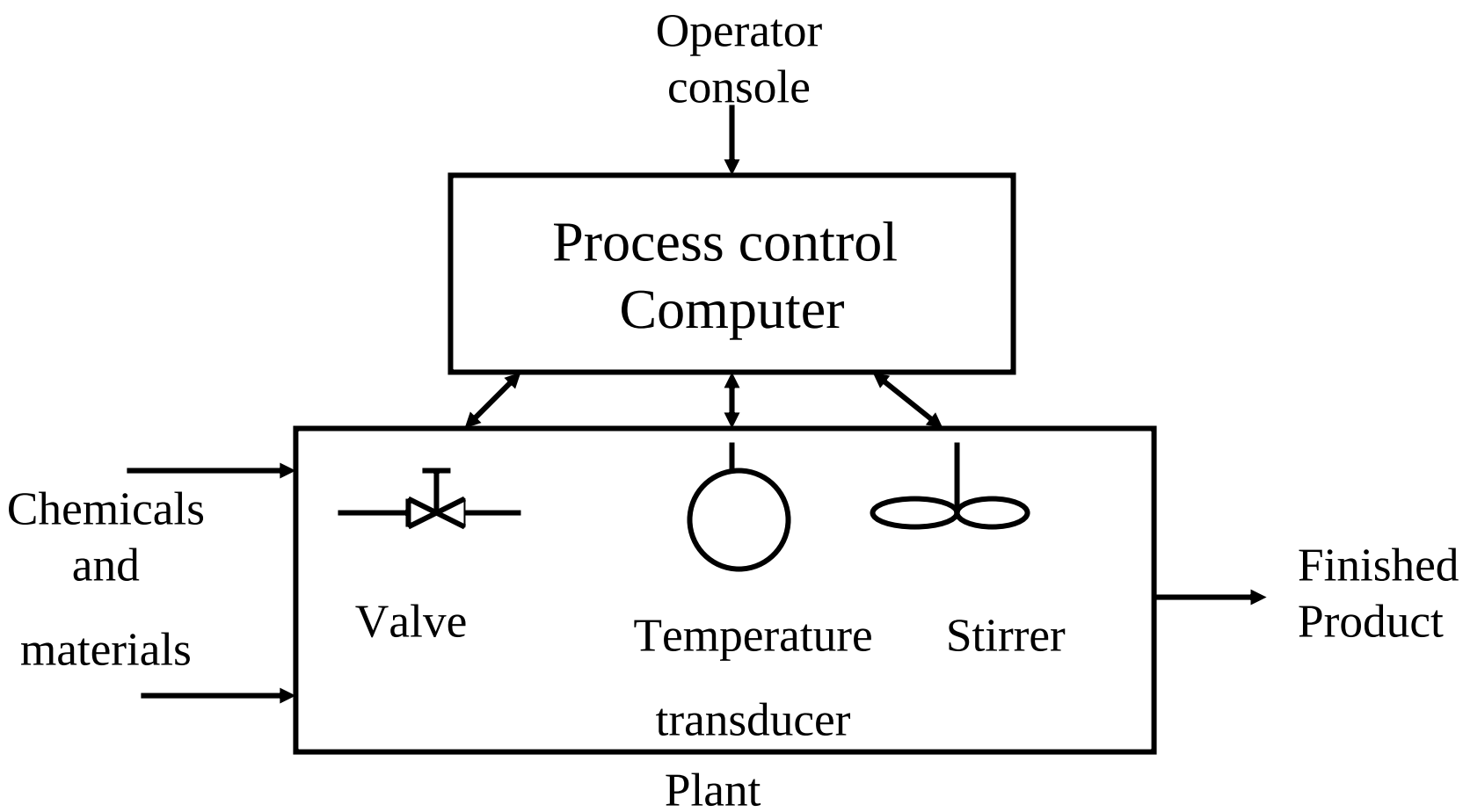
Definition of Real Time System

- **Hard real-time system and soft real-time systems:**

  (1) **Hard real-time systems** – are those where it is absolutely imperative the response occur within the specified deadline.

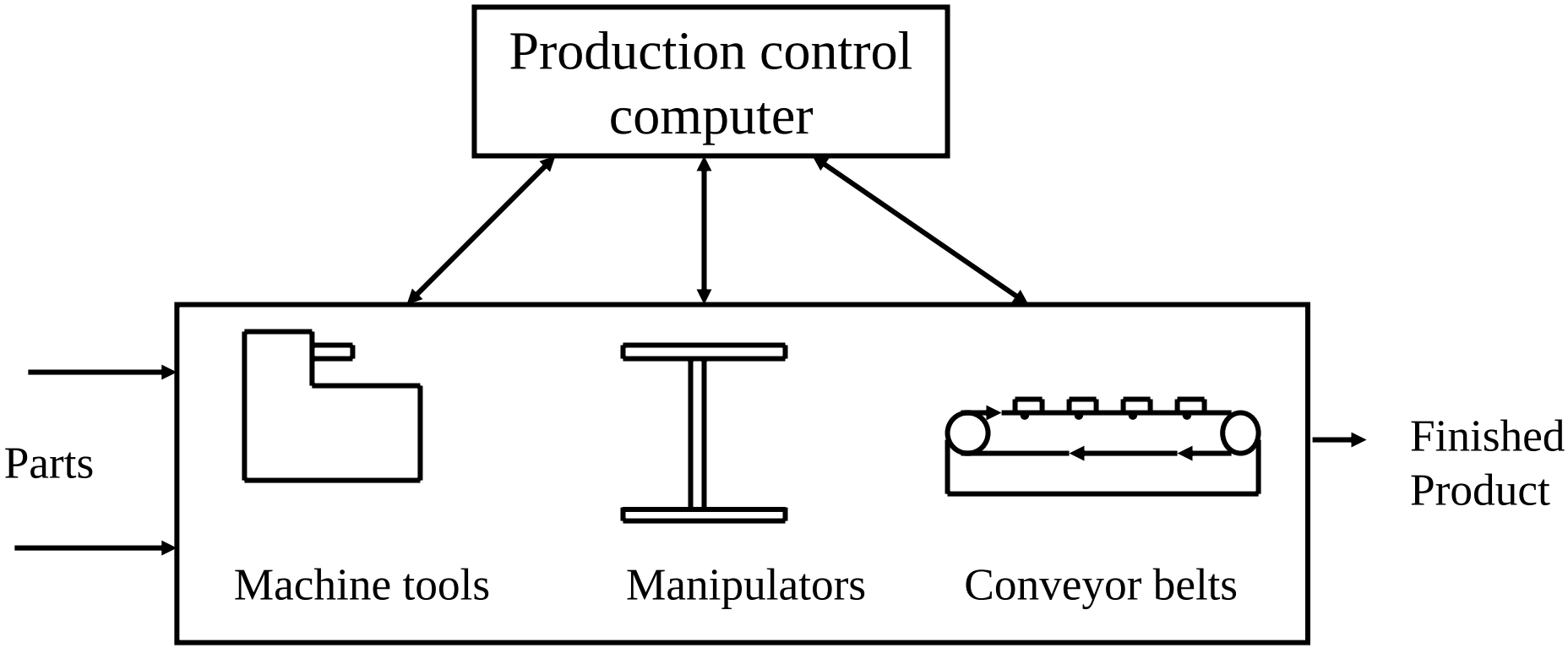
  (2) **Soft real-time systems** – are those where response time are important but the system will still function correctly if deadlines are occasionally missed.

Examples of Real-time System: Process Control



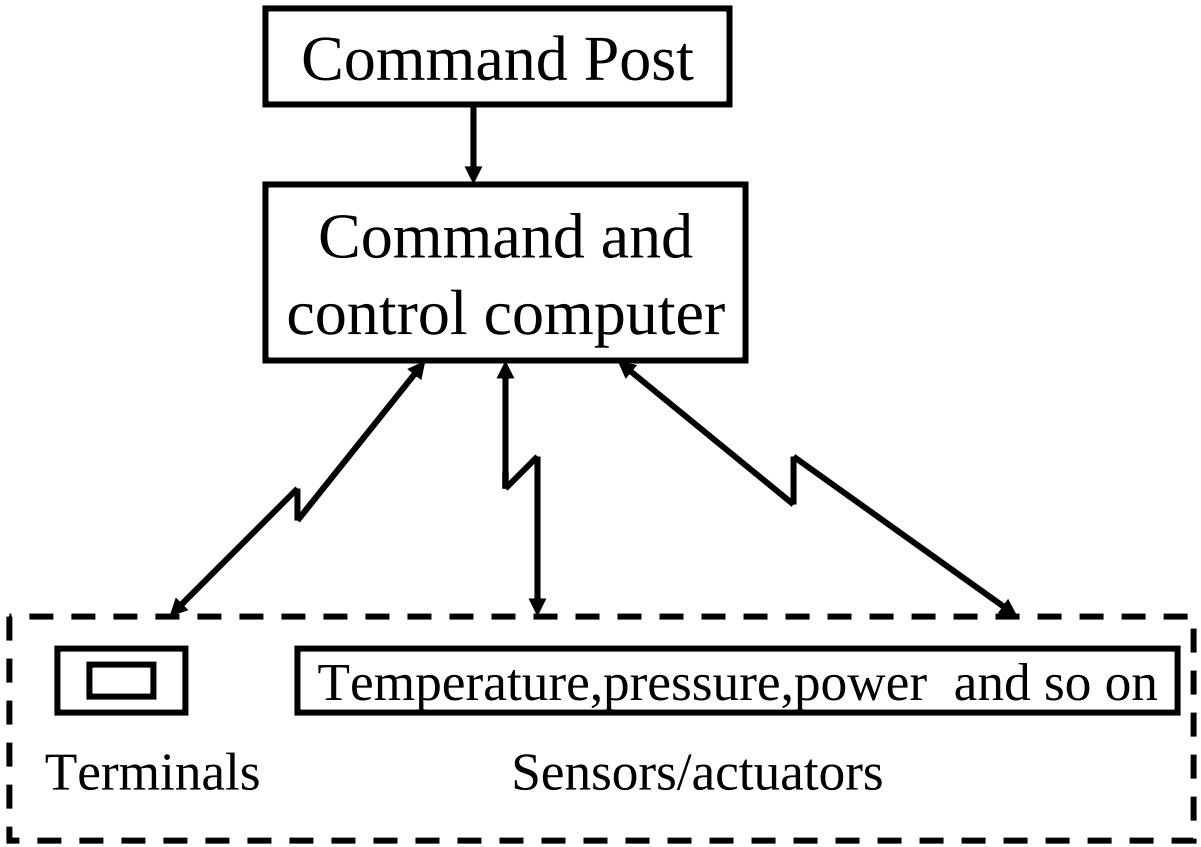
A Process Control System

Examples of Real-time Systems: Manufacturing



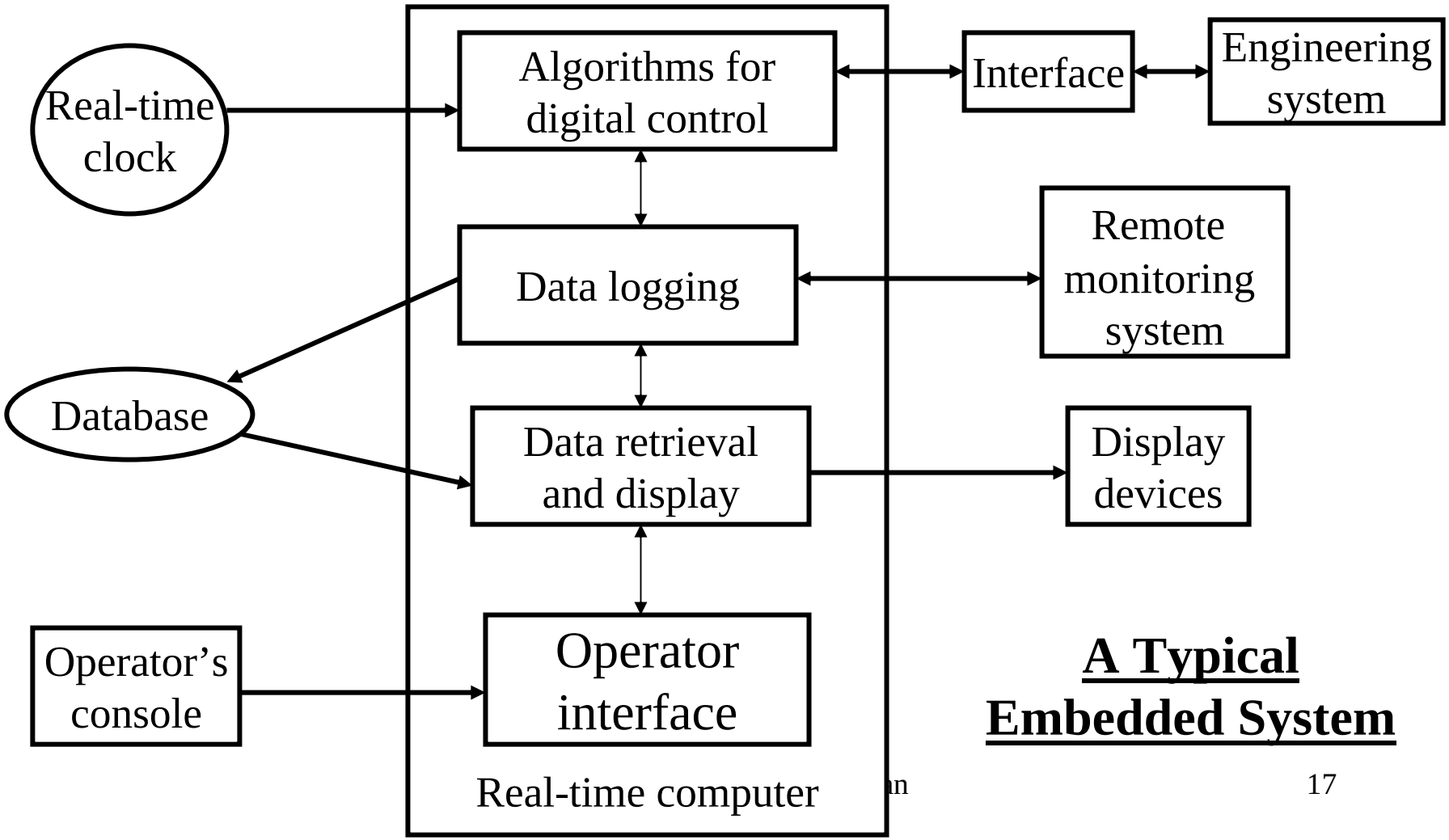
A Production Control System

Examples of real-time Systems: Communication,command and control



A Command and Control System

Examples of Real-time Systems- Generalized Embedded Computer Systems



A Typical Embedded System

Characteristics of Real-time Systems

- A real-time system has many special characteristics (either inherent or imposed). Clearly, not all real-time systems will exhibit the same characteristics. However, any general-purpose language(or operating system) which is to be used for the effective programming of real-time systems must have facilities which these characteristics.

  (1) **Large and complex**.

  -Largeness of the system is related to variety which is caused by the continuous changing requirements of the real world. Traditional indicators, such as number of instructions and development effort, are just symptoms of variety. Therefore, real-time systems undergo constant maintenance and enhancements during their lifetimes. They must be extensible.

  -Although real-time software is often complex, features provided by real-time languages and environments enable these complex systems to be broken down into smaller components which can be managed effectively.

Characteristics of Real-time Systems

(2) **Manipulation of real numbers:**

-Many real-time systems involve the control of some engineering activity and most controllers are nowadays implemented by computers.

-The implementation of control algorithms using computers is often mathematically complex and require high degree of a real time programming language, therefore, is the ability to manipulate real, fixed- or floating-point numbers.

Characteristics of Real-time Systems

(3) **Extremely reliable and safe:**

-The more society passes the control of its vital functions to computers, the more important it becomes that those computers do not fail.

*failure of a system involved in automatic fund transfer between banks may cause the loss of millions of dollars.

*a faulty component in electricity generation could result in the failure of the vital life-support system in a intensive care unit.

-The large size and complexity of real-time systems make the reliability become worse; not only must one expect difficulties inherent in the application to be taken into account, but also those introduced by faulty software design.

Characteristics of Real-time Systems

(4) **Concurrent control of separate system components:**

-In a typical embedded computer example, the program has to interact with an engineering system(which will consist of many parallel activities such as robots, conveyor belts, sensors and so on) and the computers display devices, the operator's console, the database and the real-time clock.

Characteristics of Real-time Systems

-A major problem associated with the production of software for systems which exhibit concurrency is how to express that concurrency in the structure of the program.

Older real-time programming languages, for example RTL/2,Coral 66 and C, relied on operating system support for concurrency.

The more modern languages, such as Ada, Pearl and Occam, have direct support for concurrent programming.

Characteristics of Real-time Systems

(5) **Real-time facilities:**

-Response time is crucial in any real-time system.Unfortunately, it is very difficult to design and implement systems which will guarantee that the appropriate output will be generated at the appropriate times under all possible conditions. For this, reason real-time systems are usually constructed using processors with considerable spare capacity.

Characteristics of Real-time Systems

-Given adequate processing power, language and run-time support, the programmer is to give:

#specify times at which actions are to be performed.

specify times at which actions are to be completed.

#responses to situations where all or any of the timing requirements cannot be met.

responses to situations where all or any of the timing requirements are changed dynamically.

All these are called real-time control facilities. They enable program to synchronize with time itself.

Characteristics of Real-time Systems

(6) **Interaction with hardware interfaces:**

-Most of the real-time systems requires the computer components to interact with the external devices (e.g. sensors, actuators). These devices interface to the computer via input and output registers.Devices may also generate interrupts to signal to the processor.

-The methods used in the past is:

~under the control of operating system, **or**

~the application programmer is required to resort to assembly language control and manipulate the registers and interrupts.

-The methods used nowadays are:

~the facilities provided by modern high level real-time languages allows the control to be direct, and not through a layer of operating system.

Characteristics of Real-time Systems

(7) **Efficient implementation and execution environment:**

-In real-time systems, what is actually important is the meeting of deadlines of adequate response times in a particular execution environment.

The term “execution environment” is used to mean those components which are used together with the application’s code to make the complete system: the processors, networks, operating system and so on. Clearly, the more ‘efficient’ use of the execution environment, the more likely the requirements will be met.

Real Time Scheduling

Objectives

- Describe the major characteristic of real-time operating systems
- Identify and describe the 4 major classes of real-time scheduling algorithms.
- Describe Deadline Scheduling.
- Describe Rate Monotonic Scheduling.

Real-Time Scheduling-Background

- **Background**

  -Real-time computing is becoming an increasingly important discipline. The operating systems, and in particular the scheduler, is perhaps the most important component of a real-time system.

  -A real-time system can be defined by defining what is meant by a real-time processor, or task. Such a task may be classified as hard and soft:

  >A hard real-time task is one that must meet its deadline; otherwise it will cause undesirable damage or a fatal error to the system.

  >A soft real-time task has an associated deadline that is desirable but not mandatory; it still makes sense to schedule and complete the task even if it has passed its deadline.

Real-Time Scheduling-Background

Another characteristics of real-time tasks is whether they are **periodic or aperiodic**:

>An aperiodic task has a deadline by which it must finish or start, or it may have a constraint on both start and finish time.

>A periodic tasks has the requirement to be stated as “once per period T” or “exactly T units apart ”.

Characteristics of Real-time Operating Systems

- Real-time operating systems can be characterized as having unique requirements in five general areas:
  - Determinism
  - Responsiveness
  - User control
  - Reliability
  - Fail-soft operation

Characteristics of Real-time Operating Systems

(1) Determinism

An operating system is ***deterministic*** to the extent that it performs operations at fixed, per-determined times or within pre-determined time intervals. When multiple processes are competing for resources and processor time, no system will be fully deterministic.

The extent to which an operating system can deterministically satisfy requests depends on

(a) *the speed with which it can respond to interrupts*

(b) *whether the system has sufficient capacity to handle all requests within the required time.*

One useful measure of the ability of an operating system to function deterministically is the maximum delay from the arrival of a high-priority device interrupt to when servicing begins. For real-time Operating Systems the delay may have an upper bound of anywhere form a few.

Characteristics of Real-time Operating Systems

(2) **Responsiveness**

Responsiveness is concerned with how long, after acknowledgement, it takes an operating system to service the interrupt. Aspects of responsiveness include the following.

1.***The amount of time required to initially handle the interrupt and begin execution of the interrupt service routine*** (**ISR**). This depends on whether process switch or context switch is required.

2.***The amount of the time required to perform ISR***. This generally is depend on hardware platform.

3.***The effect of interrupt nesting***. If an ISR can be interrupted by the arrival of another interrupt, then the service will be delayed.

***Determinism and responsiveness together make up the **response time** to external events and response-time requirements are critical real-time systems.

Characteristics of Real-time Operating Systems

(3)**User control**

It is generally much broader in a real-time operating system than in ordinary operating systems. In a typical non-real-time operating system, the user either has no control over the scheduling function of the operating system or can only provide broad guidance, such as grouping users into more than one priority class. In a real-time system, **it is essential to allow the user fine-grained control over task priority**. The user should be able to distinguish between hard and soft tasks and to specify relative priority within each classes. **A real-time system may also allow to specify such characteristics** as the use of paging or process swapping must always be resident in main memory, what disk transfer algorithms are to be used, what rights the processes in various priority bands have, and so on.

Characteristics of Real-time Operating Systems

(4) **Reliability**

It is typically **far more important for real-time systems than non-real-time systems**. A transient failure in non-real-time system may solved by simple rebooting the system. But a real-time system is responding to and controlling events in real-time. **Loss and degradation of performance may have catastrophic consequences**, ranging from financial loss to major equipment damage and even loss of life.

Characteristics of Real-time Operating Systems

(5) **Fail-soft operation**

It is characteristic that **refers to the ability of a system to fail in such a way as to preserve as much capability and data as possible**.

For example:

In typical traditional UNIX system, when it detects a corruption of data within the kernel, issues a failure message on the system console, dumps the memory contents to disk for later failure analysis, and terminates execution of the system. In contrast, **a real-time system will attempt either to correct the problem or minimize its effects while continuing to run**. Typically, the system notifies a user or user process that it should attempt corrective action and then continues operation perhaps at a reduced level of service. In the event a shutdown is necessary, an attempt is made to maintain file and data consistency.

Characteristics of Real-time Operating Systems

An important aspect of fail- soft operation is referred to as stability. A real-time system is stable if, in cases where it is impossible to meet all task deadlines, the system will meet the deadlines of its most crucial, highest-priority tasks, even if some less critical task deadlines are not always met.

***To meet the foregoing requirements (the characteristics of real-time operating systems), current real-time operating systems typically include the following features:**

-fast process or thread switch

-small size (with its associated minimal functionality)

-ability to respond to external interrupts quickly.

Characteristics of Real-time Operating Systems

-Multitasking with inter-process communication tools such as semaphores, signals, and events.

- Use of special sequential files that can accumulate data at a fast rate.

- Preemptive scheduling based on priority.

-Minimization of intervals during which interrupts are disabled.

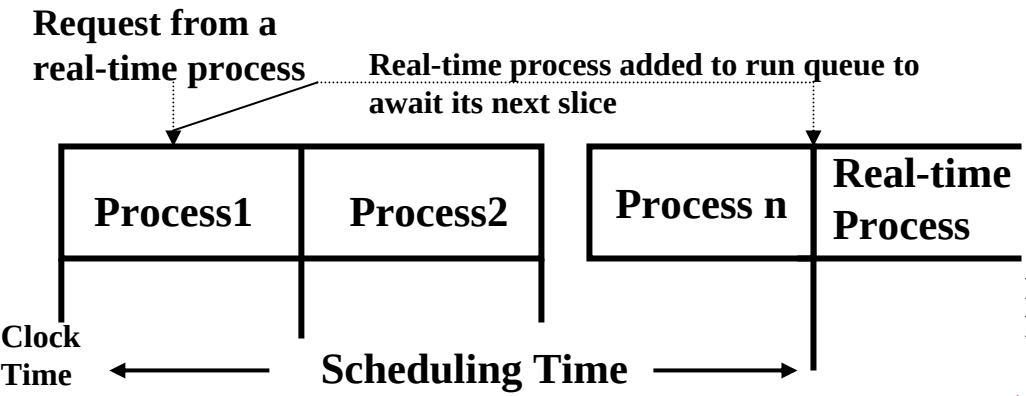
-Primitives to delay tasks for a fixed amount of time and to pause/resume tasks.

-Special alarms and time outs.

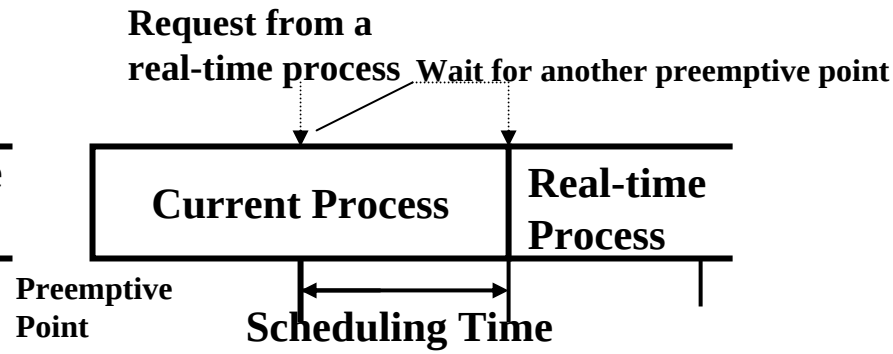
Characteristics of Real-time Operating Systems

- Most contemporary real-time operating systems are unable to deal directly with deadlines. Instead, they are designed to be as responsive as possible to real-time tasks so that, when a deadline approaches, a task can be quickly scheduled. From this point of view, real-time applications require deterministic response times in the several-millisecond to sub-millisecond span under a broad set of conditions; leading-edge applications - in stimulators for military aircraft, for example-often have constraints in the range of 10 to 100 us.

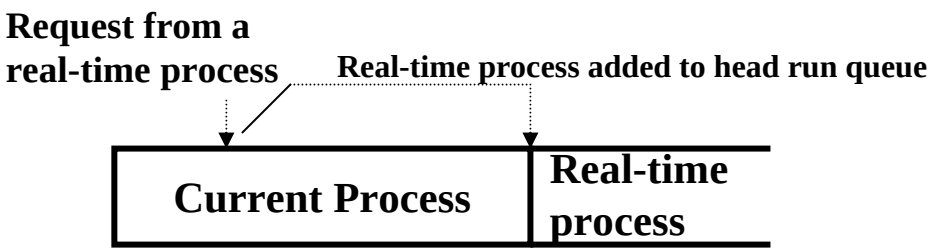
Characteristics of Real-time Operating Systems



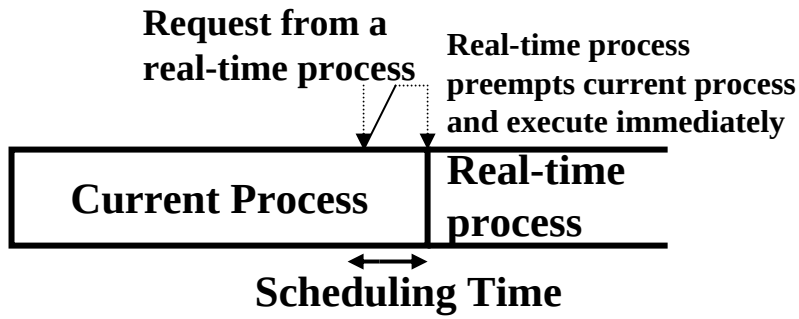
(a) Round-Robin Preemptive Scheduler



(c) Priority-drive, Preemptive Scheduler on Preemptive Points

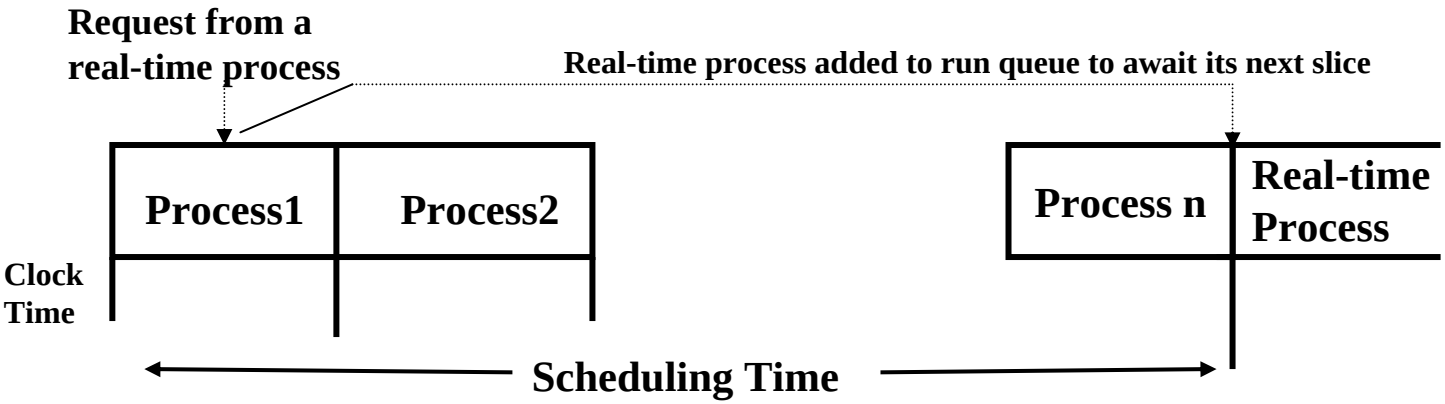


(b) Priority-driven Non-Preemptive Scheduler

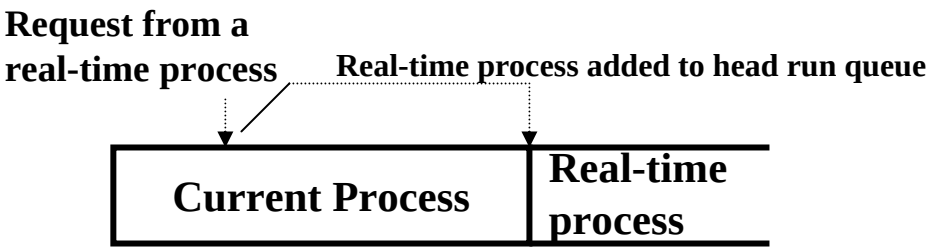


(d) Immediate Preemptive Scheduler

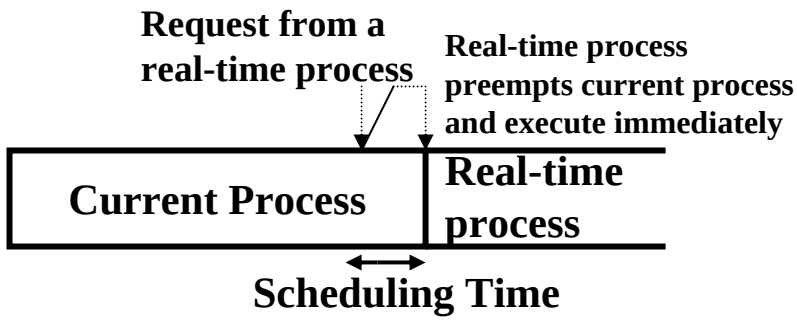
Scheduling of a Real-time Process



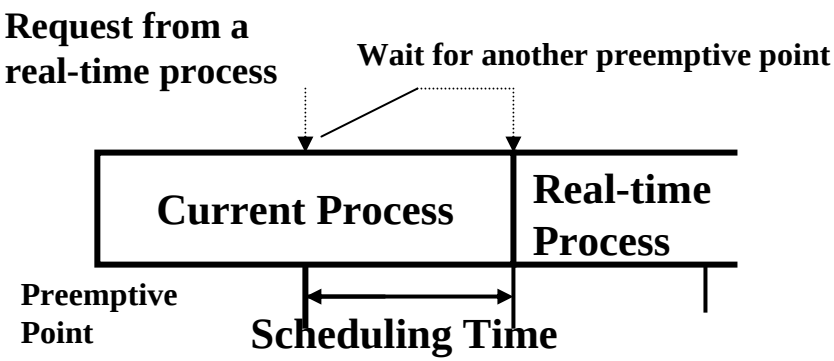
(a) Round-Robin Preemptive Scheduler



(b) Priority-driven Non-Preemptive Scheduler



(d) Immediate Preemptive Scheduler



(c) Priority-drive, Preemptive Scheduler on Preemptive Points

Real –time Scheduling

- ***The heart of a real-time system is the short-term task scheduler*** & it is one of the most active areas of research in computer science. **In designing such a scheduler, what is important is that the hard real-time tasks complete (or start) by their deadline and that as many as possible soft real-time tasks also complete (or start) by their deadline.**
- In a survey of real-time scheduling algorithms, it was observed that the various scheduling approaches depend on:

  (i) whether a system performs schedulability analysis,

  (ii) if it does, whether it is done statically or dynamically, and

  (iii) whether the result of the analysis itself produces a schedule or plan according to which tasks are dispatched at run time.

Real –time Scheduling

Based on the considerations as described before, the following classes of algorithms are identified:

- **Static table-driven approaches**:

  ~ ***Performs a static analysis of feasible schedules of dispatching. The result of the analysis is a schedule that determines, at run time, when a task must begin execution.***

  ~ ***It is applicable to tasks that are periodic***. Input to the analysis consists of the periodic arrival time, execution time, periodic ending deadline, and relative priority of each task. The scheduler attempts to develop a schedule that enables it to meet the requirements of all periodic tasks.

  ~***It is predictable approach but inflexible*** because any change to any task requirements requires that the schedule be redone.

  ~***The examples are earliest-deadline-first or other periodic deadline techniques.***

Real –time Scheduling

- **Static priority-driven preemptive scheduling:**

  # **A static analysis is performed, but no schedule is drawn. The analysis is used to assign priorities to tasks.**

  # It makes use of the priority-driven preemptive scheduling mechanism common to most non-real-time multiprogramming systems. In a real-time system, priority assignment is related to the time constraints associated with each task.

  # **One example of this approach is the rate monotonic algorithm** which assigns static priorities to tasks based on their periods.

- **Dynamic planning-based scheduling:**

  # **Feasibility is determined at run time (dynamically)** rather than offline prior to the start of execution (statically).

  # **An arriving task is accepted for execution only if it is feasible to meet its time constraints**. If the new arrival can be scheduled in such a way its deadlines are satisfied and that no currently scheduled task misses a deadline, then the schedule is revised to accommodate the new task.

Real –time Scheduling

- **Dynamically best effort scheduling**
  - ***No feasible analysis is performed.***
  - ***The system tries to meet all deadlines and aborts any started process whose deadline is missed.***
  - ***Is the approach used by many real-time systems that are currently commercially available***. When a task arrives, the system assigns priority based on characteristics of the task. Some form of deadline scheduling, such as earliest-deadline scheduling, is typically used. Typically, the tasks are aperiodic and so on static scheduling analysis is possible. With this type of scheduling, until a deadline arrives or until the task completes, we do not know whether a timing constraint will be met. This is the major disadvantage of this form of scheduling. Its advantage is that it is easy to implement.

Deadline Scheduling

- Most contemporary real-time operating systems are designed with the objective of starting real-time tasks as possible, and hence emphasize rapid interrupt handling and task dispatching. In fact, this is not a particularly useful metric in evaluating real-time operating systems.
- Real-time applications are generally not concerned with sheer speed but rather with completing (or starting) tasks at the most valuable times, neither too early nor too late, despite dynamic resource demands and conflicts, processing overloads, and hardware or software faults.
- There have been a number of proposals for more powerful and appropriate approaches to real-time task scheduling. All of these are based on having additional information about each task. In its most general form, the following information about each tasks might be used:

Deadline Scheduling

- **Ready time :** ***Time at which task becomes ready for execution.*** In the case of
  - **periodic task –** a sequence of times that is known in advance
  - **aperiodic task –** this time may be known in advance, or the operating system may only be aware when the task is actually ready.
- **Stating deadline :** ***Time by which a task must begin.***
- **Completion deadline :** ***Time by which task must be completed***. The typical real-time application will either have starting deadlines or completion deadlines, but not both.
- **Processing time :** ***Time required to execute the task to completion.*** In some cases, this is supplied. In others, the operating system measures an exponential average. For still other scheduling systems, this information is not used.

Deadline Scheduling

- **Resource requirements : Set of resources (other than the processor) required by the task while it is executing.**
- **Priority : Measures relative importance of the task.** Hard real-time tasks may have an “absolute” priority, with the system failing if a deadline is missed. If the system is to continue to run no matter what, then both hard and soft real-time may be assigned relative priorities as a guide to the scheduler.
- **Subtask structure : A task may be decomposed into a mandatory subtask and an optional subtask**. Only the mandatory subtask processes a hard deadline.

Deadline Scheduling

- There are several dimensions to the real-time scheduling function when deadlines are taken into account:

  -Which task to schedule next ?

  -Which sort of preemption is allowed ?

  -It can be shown, for a given preemption strategy and using either starting or completion deadline, that a policy of scheduling the task with the earliest deadline minimizes the fraction of tasks that miss their deadlines.

  -When starting deadlines are specified, then a non-preemptive scheduler make sense. In this case, it would be responsibility of the real-time task to block itself after completing the mandatory or critical portion of its execution, allowing other real-time starting deadlines to be satisfied.

  -For a system with completion deadlines, a preemptive strategy is most appropriate.

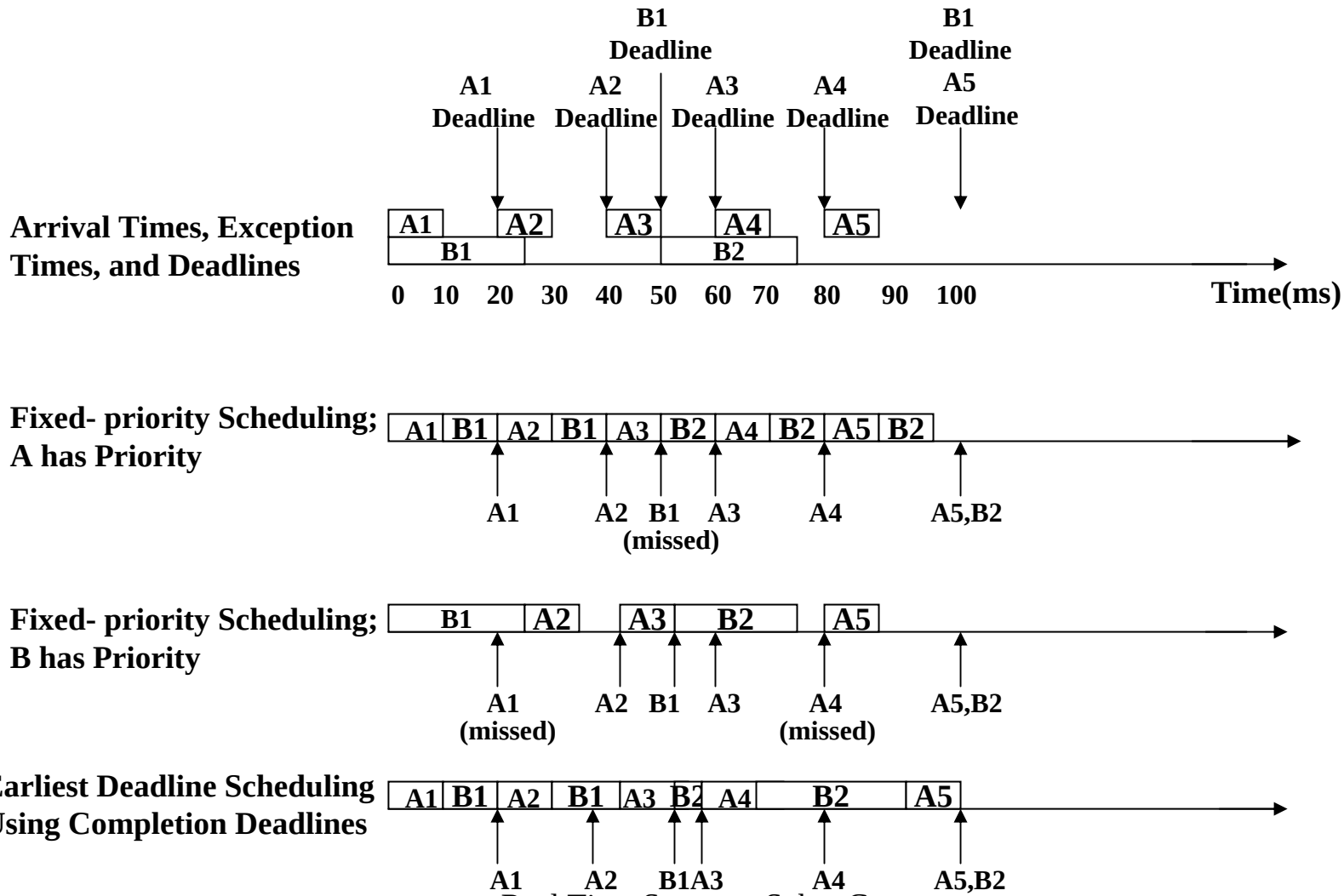
An example of Scheduling Periodic Tasks with completion deadlines

- Consider a system that collects and processes data from two sensors, A and B. the deadline for collecting data from sensor A must be ,met every 20 ms, and that for B every 50 ms. It takes 10 ms, including operating system overhead, to process each sample of data from A and 25ms to process each sample of data from B. The following table summaries the execution profile of the two tasks.

An example of Scheduling Periodic Tasks with completion deadlines

Process	Arrival Time	Execution Time	Ending Deadline
A (1)	0	10	20
A (2)	20	10	40
A (3)	40	10	60
A (4)	60	10	80
⋮	⋮	⋮	⋮
B (1)	0	25	50
B (2)	50	25	100
⋮	⋮	⋮	⋮

An example of Scheduling Periodic Tasks with completion deadlines

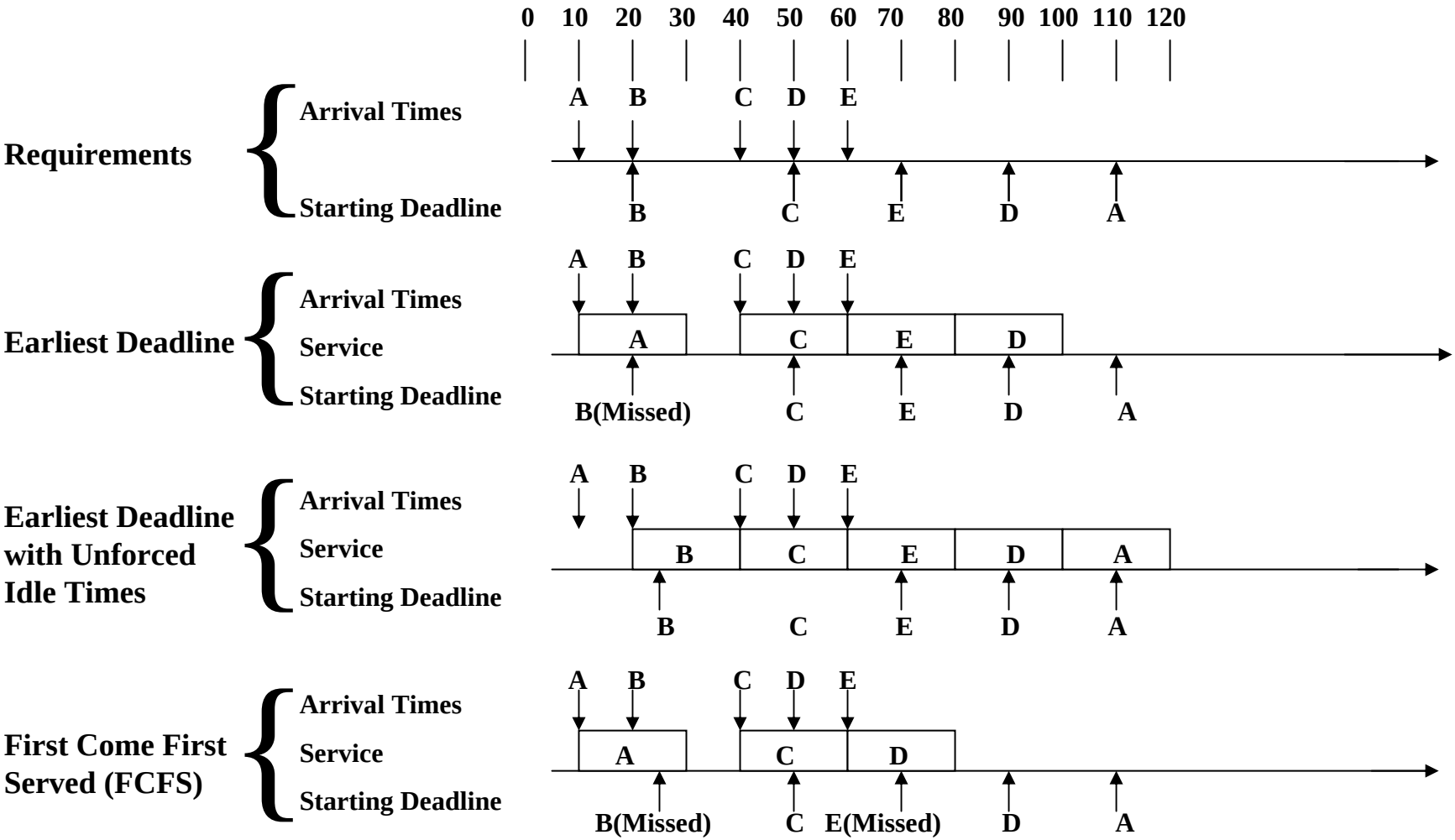


An example of Scheduling Periodic Tasks with completion deadlines

- This examples shows the scheme for dealing with aperiodic tasks with starting deadlines.

Process	Arrival Time	Execution Time	Starting Time
A	10	20	110
B	20	20	20
C	40	20	50
D	50	20	90
E	60	20	70

An example of Scheduling Periodic Tasks with completion deadlines



Rate Monotonic Scheduling (RMS)

- One of the more promising methods of resolving multitask scheduling conflicts for periodic tasks.
- The relevant parameters for periodic tasks:

  - **The task's period, T,** is the amount of time between the arrival of one instance of the task and the arrival of the next instance of the task.

  - **The task's rate (in Hertz)** is simply the inverse of its period (in seconds). For example, a task with a period of 50 ms occurs at the rate of 20 Hz. Typically, the end of a task's period is also the task 's hard deadline, although some tasks may have earlier deadlines.

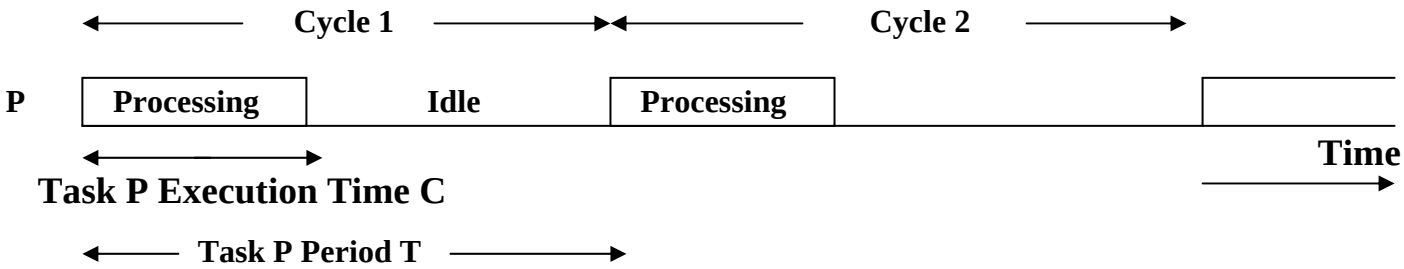
  - **The execution time, C,** is the amount of processing time required for each occurrence of the task. In a uniprocessor system, the execution time must be no greater than the period **(must have C<=T).**

  - If a periodic task is always run to completion – that is , if no instance of the task is ever denied service because of insufficient resources – then the utilization of the processor by this task is **U=C/T.**

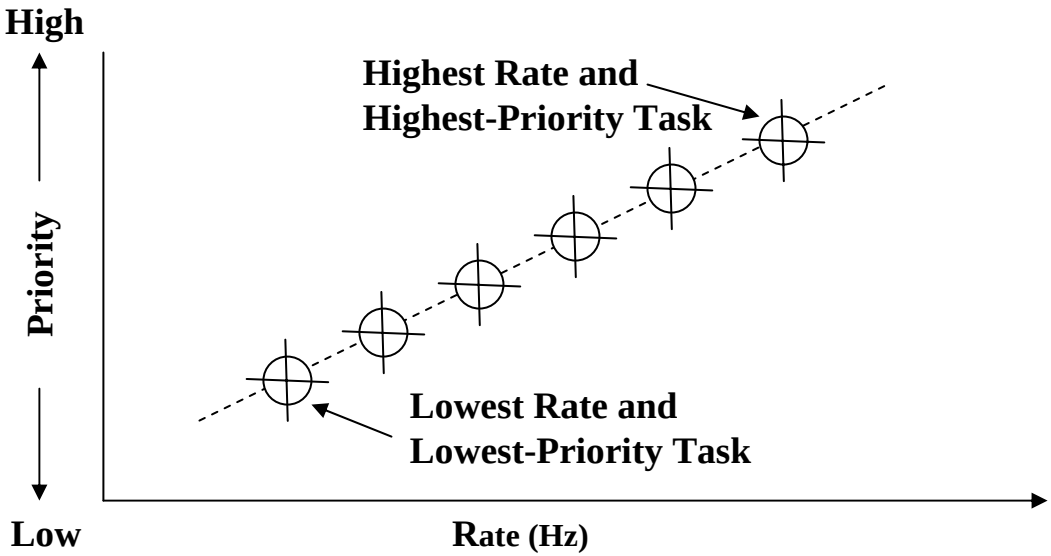
Rate Monotonic Scheduling (RMS)

- For **RMS**, the highest-priority task is the one with the shortest period, the second highest-priority task is the one with the second shortest period, and so on. When more than one task is available for execution, the one with the shortest period is serviced first. If we plot the priority of tasks as a function of their rate, the result is monotonic scheduling.

Rate Monotonic Scheduling (RMS)



Periodic Task Timing Diagram



A Task Set with RMS [WARR01]

Rate Monotonic Scheduling (RMS)

- One measure of the effectiveness of a periodic scheduling algorithm is whether or not it guarantees that all hard deadlines are met. Suppose that we have n tasks, each with the fixed period and execution time. Then for it to be possible to meet all deadlines, the following inequality must hold:

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots\cdots + \frac{C_n}{T_n} <= 1 \qquad \text{(Equation 4-1)}$$

For RMS, it can be shown that the following inequality holds:

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots\cdots + \frac{C_n}{T_n} <= 1 \qquad \text{(Equation 4-2)}$$

Rate Monotonic Scheduling (RMS)

The following table gives some values for this upper bound.

n	$n(2^{1/n} - 1)$
1	1.0
2	0.828
3	0.779
4	0.756
5	0.743
6	0.734
.	.
.	.
.	ln 2= 0.693

Rate Monotonic Scheduling (RMS)

As the number of tasks increases, the scheduling bound converges to

$\ln 2 \approx 0.693$.

As an example, consider the case of 3 periodic tasks where

$$U_i = C_i / T_i$$

>> Task P_1: $C_1 = 20$; $T_1 = 100$; $U_1 = 0.2$

>> Task P_2: $C_2 = 40$; $T_2 = 150$; $U_2 = 0.267$

>> Task P_3: $C_3 = 100$; $T_3 = 350$; $U_3 = 0.286$

Rate Monotonic Scheduling (RMS)

- The total utilization of these three tasks is 0.2+0.267+0.286=0.753, the upper bound of the schedulability of these three tasks using RMS is

$$\frac{C_1}{T_1}+\frac{C_2}{T_2}+\frac{C_3}{T_3} <= 3(2^{1/3}-1)=0.779$$

Because the total utilization required for the 3 tasks is less that the upper bound for RMS(0.753<0.799), we know that if RMS is used, all tasks will be successfully scheduled.

It is also be shown that upper bound of equation 4-1 holds for earliest deadline scheduling. This, it is possible to achieve greater overall processor utilization and therefore accommodate more periodic tasks with earliest deadline scheduling.

Rate Monotonic Scheduling (RMS)

Nevertheless, RMS has been widely adopted for use in industrial applications offers the following explanation:

1. **The performance difference is small in practice**. The upper bound of equation 4-2 is conservative. In practice, utilization as high as 90% is often achieved.

2. Most hard real-time systems also have **soft real-time components**, such as certain non-critical displays and built-in self-tests that **can execute at lower priority levels to absorb the processor time that is not used with RMS scheduling of hard-time tasks.**

Rate Monotonic Scheduling (RMS)

3. **Stability is easier to achieve with RMS**. When a system cannot meet all deadlines because of overload or transient errors, the deadlines of essential tasks need to be guaranteed provided that this subset of tasks is schedulable. In a static priority assignment approach, one only needs to ensure that essential tasks have relatively high priorities. This can be done in RMS by structuring essential tasks to have sort periods or modifying the RMS priorities to account for essential tasks. With earliest deadline scheduling, a periodic task's priority changes from one period to another. This makes it more difficult to ensure that essential tasks meet their deadlines.

If the total utilization of the tasks is no greater than

$n (2^{1/n} – 1)$ where n = the number of tasks, then

RM algorithm will schedule all of the tasks to meet the dead lines

Critical Section Handling:

- **Use Semaphore**
- **Dead Lock while sharing resources**
- **Priority Inversion Problem**
- **Use Priority Inheritance solution**

Sporadic Tasks

Consider it as periodic task with period equals minimum inter arrival time of the Sporadic task

Critical Tasks

These with long period can be broken to small period tasks to get high priority in RM scheduling algorithm.

Preemptive Earliest Dead Line First Algorithm (EDF)

Real Time communication

List of Protocols

Protocol	Dead line Guarantee	Network
VTCSMA	No	Broadcast
Window	No	Broadcast
Timed Token	Yes	Ring
IEEE 802.5	Yes	Ring
Stop and Go	Yes	Point to Point
Polled Bus	No	Bus
Round Robin (Hier)	Yes	Point to Point
Dead line Based	No	Point to Point

Real Time Databases

Real Time Fault Tolerant Techniques

- **Hardware design**
- **Software Architecture**

References

1. C.M. Krishna and R.G. Shin, "Real time system" McGraw Hill 1997

Micro C/OS-II, The real time kernel, A complete portable, ROMable, scalable preemptive RTOS by Jean J. Labrosse, R&D books, Miller Freeman inc., ISBN: 0-87930-543-6; Phone: 785 841 1631.

2. Embeded Systems Building Blocks, 2nd edition, Complete and ready to use modules in C, by Jean J. Labrosse, R&D books, Miller Freeman inc., ISBN: 0-87930-604-1; Phone: 1-800-788-3123.

3. Jeffrey Tsai and Steve Yang, " Monitoring and Debugging of Distributed real time system" IEEE computer Soceity press, ISBN 0-8186-6537-8

4. Java for Embedded System, by Ingo Cyliax, Circuit Cellar magazine, December 2000 and January 2001.

5. Real Time JVM, New Monics Inc., www.newmonics.com

6. Jworks, Windriversystems, Inc, www.wrs.com

7. Java Chip, ajile systems inc., www.ajile.com

8. Valvano, " Embedded microcontroller system- real time interfacing" Brooks/Cole publisher

Ronald Jurgen " Automotive Handbook", McGrawHill Handbook, second