INTRODUCTION to REAL-TIME SYSTEMS

References [Kri974], [Nis97]

INTRODUCTION to REAL-TIME SYSTEMS

Operation of computer systems

FUNDAMENTALS
Real Time Systems
Structure of Real Time Systems
Constraints of Real Time Systems
Properties of Real Time Systems
Application of Real Time Systems

TYPES of REAL-TIME SYSTEMS
Clock- or Time-Based (or Periodic) Systems
Event- or Interrupt-Driven (Sporadic) Systems
Interactive Systems

TYPES of COMPUTER PROGRAMS
Sequential Programs
Multi-Tasking Programs
Real Time Programs

PERFORMANCE of REAL-TIME SYSTEMS
Performance Measures for Real Time Systems
Estimation of Run Times of Real Time Programs

DESIGN of REAL-TIME SYSTEMS
Design and Implementation Issues

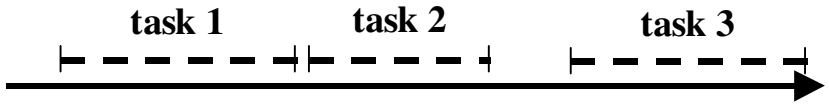
STATE of ART in REAL-TIME SYSTEMS
Specification and Verification.
Real Time Kernels and Operating Systems
Scheduling
Real Time Languages
Architectures and Structures
Fault Tolerance

INTRODUCTION to REAL-TIME SYSTEMS

Operation of computer systems

1. sequential operation:

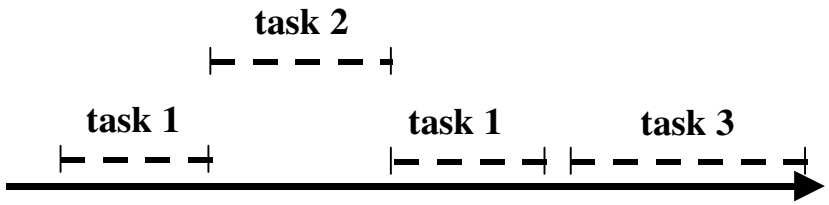
- performing tasks, handling inputs and outputs, processing functions and communication are being carried out in computer systems with single processor sequentially i.e. one-by-one,



2. pseudo-concurrent:

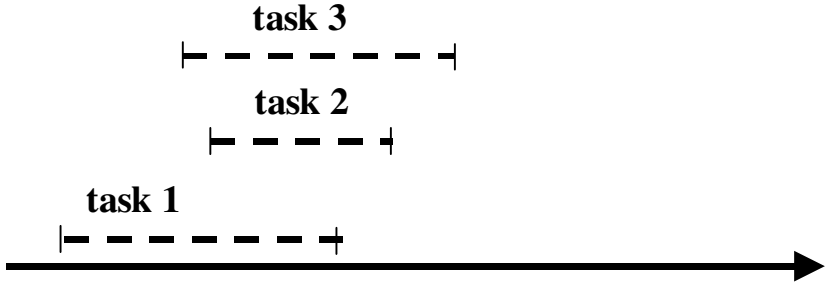
- single processor is distributed between several tasks using multi-tasking approach

- in a computer system with a single processing unit concurrent or parallel operation cannot be achieved,



3. concurrent operation:

- various tasks are being carried out by separate processors of the multiprocessor computer system parallel,



FUNDAMENTALS

Real Time Systems

- real-time systems require both the logical and temporal correctness contrary to sequential systems that need only logical correctness,

- time is the most valuable resource that should be managed by rts, for example task must be assigned and scheduled to be completed before their deadlines,

- a real-time system cannot perform its task either because of faults in hardware and/or software or because of not responding in time to meet timing requirements,

***logical correctness*: property that describes whether a system performs its tasks as required,**

- logical correctness may depend on the timing correctness of certain components, but separating the functional specification from timing specification is difficult,

***temporal correctness:* property that describes whether a system meets its timing constraints ,**

- it depends on the timing requirements of the application,

- an ordering /partial ordering/ of events taking into consideration their temporal precedence

***implicit precedence relation:* input-output relation where outputs follow inputs**

conformity of the behaviour with the specified input-output relation corresponds to logically correct behaviour

***explicit precedence relation* /temporal precedence relation/: events are patterns of interaction with its environment**

- in sequential systems logical correctness is defined by correct inputs and outputs,

- in concurrent systems the correctness is defined by correct pattern of behaviour and correct sequence of events,

- in real-time systems inputs correspond to some actions or events in the environment or the physical world, and outputs have to respond to these actions and events, taking into account timing constraints of the environment,

- rts read inputs from their environment and send output signals to their environment at times determined by operational considerations of their environment not at times limited by the capabilities of the computer system,

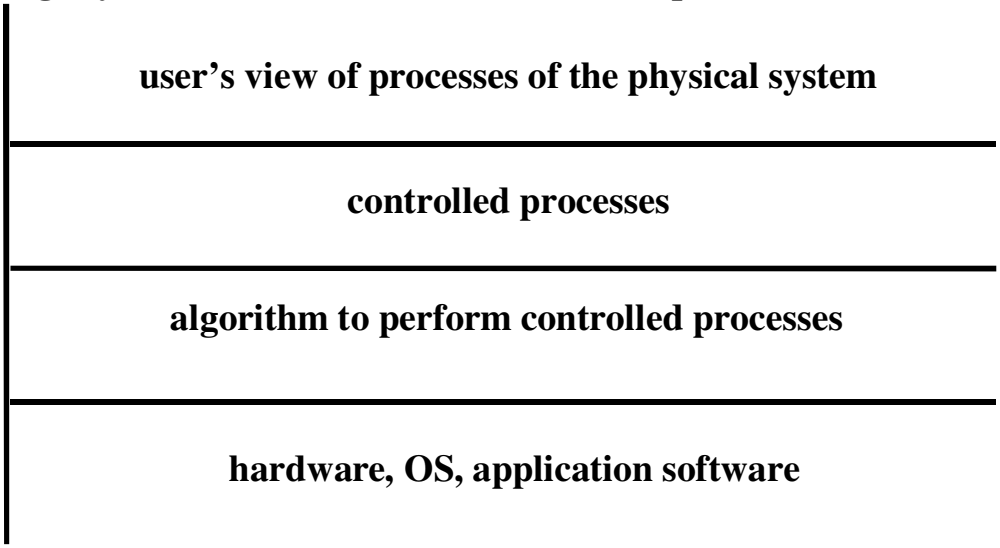
- rts should produce correct responses within a definite time limit, given by the environment of the rts,

- severe consequences will result if logical as well as temporal properties of the system are not satisfied, for example if responses exceed time bounds it may result in performance degradation and/or malfunction,

- rts solve the problem of missing deadlines in ways specific to the requirements of the environment or the physical system,

- sooner a system determines that deadline is going to be missed, the more flexibility it will have in dealing with the exception,

- temporal properties of rts, specifically the execution times of the various tasks, tightly related to the hardware of the computer,



structure of the real-time system

TASK

- a process with a single thread that thinks it has the processor all the time,

- real time applications are split into tasks that perform some jobs of the application,

- each task has its own set of processor's registers, its own stack area, and a priority is assigned to it,

- tasks are invoked at either regular or irregular intervals, and have deadlines by which they must complete their execution,

- during the execution of a task it senses the state of the system, performs some computation, and if necessary, sends commands to change the state of the system,

types of tasks

time-critical tasks	non-time critical tasks
periodic tasks	aperiodic tasks

time critical tasks:

- task that has a timing requirements that should be met during the operation of computer,

- computer should execute time-critical tasks so that each task will meet its time requirements or deadlines, if deadlines are missed severe consequences may occur,

- need to meet the requirements of individual time-critical tasks is an issue that makes the problem of designing a real-time system a hard problem,

- deadlines or other explicit timing constraints are attached to tasks, the systems should make compromises in order to avoid faults that may cause catastrophic consequences,

- it is not always necessary that critical tasks be executed successfully and within the given deadlines during each execution cycle,

- time-critical task is often executed at a higher frequency than is absolutely necessary to achieve time redundancy and to ensure one successful completion every "n" iteration of the task which is sufficient to keep system alive,

non-time critical tasks:

- computer should execute the non-time critical tasks so that the average response time of these tasks is minimised,

- the goal of scheduling of these tasks is to maximise the number of tasks successfully executed within their deadlines,

- they deal with data that depends on time as a result they are useless if not completed within a deadline,

periodic tasks:

- they are time critical tasks, i.e. they cannot run further without completing in time,

- they are activated at regular intervals and should be completed before their deadlines

aperiodic(or non-periodic or sporadic) tasks:

- these tasks are activated only when certain events occur, because events may not always occur at regular intervals, the corresponding tasks also do not arrive at regular intervals,

- if the event is time-critical, then the corresponding aperiodic task will have a deadline,

- if the event is not time-critical, then the corresponding task will not have deadline, but it should be serviced as soon as possible without affecting the deadlines of the other tasks,

DEADLINE

- deadline is a time limit in the time space before a task should be completed,

- deadlines are defined by rts including the environment and the system,

types of deadlines:

hard	firm	soft

hard deadlines:

- deadline is hard if the consequences of not meeting it can be catastrophic,

- systems with hard deadlines must satisfy "hard" time constraints or deadlines on each and every occasion,

- in these systems temporal correctness is critically important and may not be sacrificed for other gains,

- for example in safety critical systems logical correctness may be relaxed in favour of achieving temporal correctness, this results in poorer quality, but on time, that may be preferable to a late but a highly accurate result,

firm deadlines:

- deadline is firm if the results produced by a task will not be useful as soon as the deadline expires, for example transactions in a database system,

soft deadlines:

- deadline is soft deadline if the utility of results produced by a task decreases over time after the deadline expires,

- systems with soft deadlines are systems for which an occasional failure to meet a deadline does not comprise the correctness of the system,

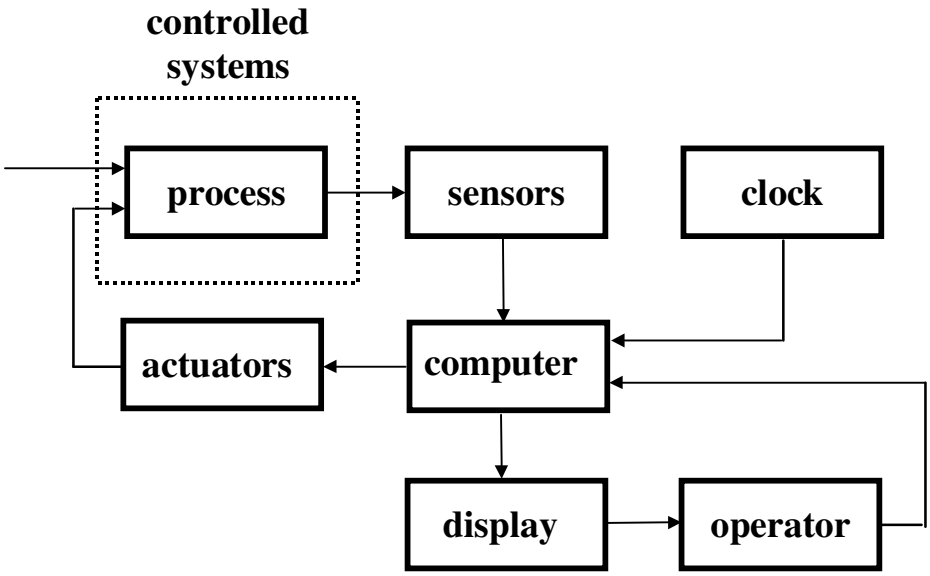
- temporal correctness is important but not critical because a failure should not result in serious consequence,

- soft real-time tasks are performed as fast as possible but are not constrained by absolute deadlines and their temporal correctness may be sacrified under special circumstances,

Structure of Real-Time Systems

- rts is a computer-based system that consists of a control system and a controlled system or environment,

- the computer system interacts with its environment through several or other subsystems,



structure of the real-time system

1. controlled system

- the environment or the physical system with that the computer communicates and interacts,

2. control system

- interacts with its environment based on the information available about the environment,

- temporal requirements and their correctness in a rts also arise as a result of the physical impact of the control systems' activities upon its environment,

- control system includes the control computer that must be capable to

work in harsh environments and in changing computational loads

- the computer is connected to the environment within which it is working by a wide range of interface devices and receives and sends a variety of stimuli,

- external processes /actions or events/ operate in their own time scales and the computer operates in real-time if actions carried out in the computer relate to the time-scales of the external processes.

- computers in rts differ from computers in non-rts because they are more specific in applications, and consequences of their failure are more drastic,

- computers of rts is designed for specific applications, this approach provides an optimum performance,

- as a result of severe consequences of failures in rts computers, these computers should be specified more carefully, their performance should be better validated,

- these requires

1. specification languages to describe constraints of rts

2. performance measures to validate specifications

3. tool to model the reliability of hardware and software

4. support to allocate tasks to processors and to schedule tasks to meet deadlines

5. mechanisms that provide recovery from failures of components

- it also contains actuators and sensors i.e. the system receives information about the environment from sensors and affects the environment through actuators,

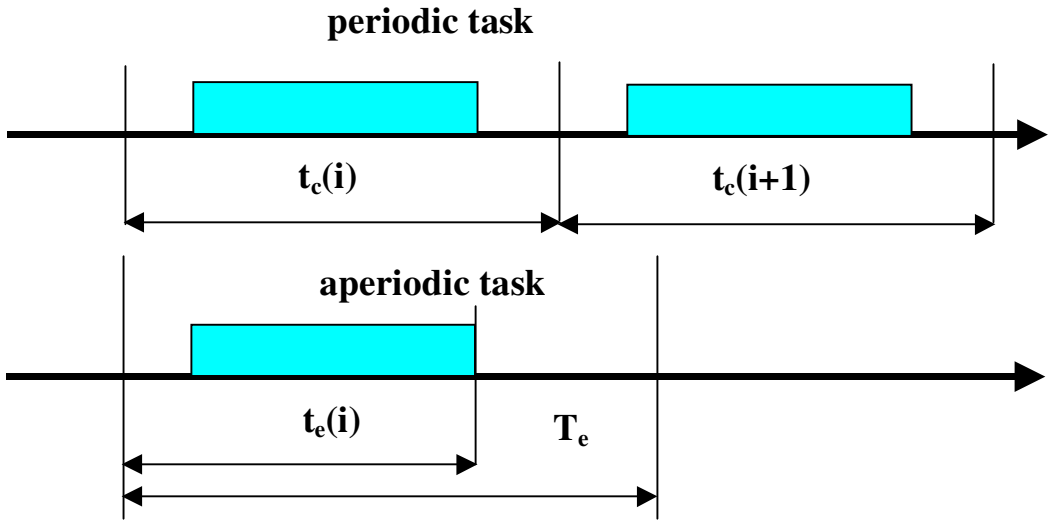
Constraints of Real Time Systems

types of constraints:

time constraints	**resource constraints**
precedence constraints	**concurrency constraints**
placement constraints	**performance constraints**
criticalness constraints	

1. time constraints:

a. hard time constraints



periodic (cyclic) $t_c(i) = t_s \pm a$ **(1.1)**

aperiodic (event) $t_e(i) << T_e$ **(1.2)**

where

t_c **(i)** **- the interval between the "i"and "i -1" cycles,**

t_e**(i)** **- the response time to the ith occurrence of event "e",**

t_s **- the desired periodic (cyclic) interval,**

T_e **- the maximum permitted response time to event "e",**

a **- time tolerance.**

- a periodic task is performed repetitively and has a time interval /period/ during that the periodic task should be completed,

- duration of the period of periodic tasks are known and they can be pre-scheduled,

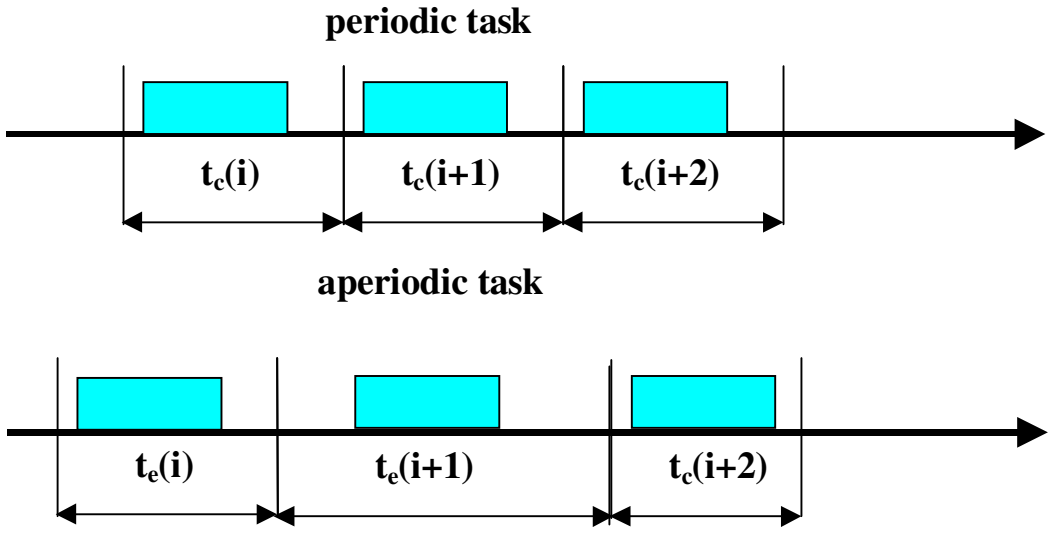
- periodic task is for example the monitoring that samples the environment in highly regulated and predictable manner,

- an aperiodic task occurs occasionally and it has a deadline by which it must finish or start, or it may have time constraints to both start and finish times,

- aperiodic tasks cannot be predicted and sufficient computing power must be held in reserve to execute them to meet their deadlines,

- aperiodic task is called into action for example upon detection of irregularities in the environment during monitoring,

b. soft time constraints:



periodic (cyclic) $$\frac{1}{n}\sum_{i=1}^{n} t_c(i) = t_s \qquad n = \frac{T}{t_s} \qquad (1.3)$$

aperiodic (event) $$\frac{1}{n}\sum_{i=1}^{n} t_e(i) \le T_a \qquad n = \frac{T}{t_s} \qquad (1.4)$$

where

T_a - the average permitted response time to event "e" measured over some time interval "T",

n - the number of occurrences of event "e" within the time interval "T", or the number of cyclic repetitions during the time interval "T",

- a hard time constraint represents a much more severe constraint to the performance of the system than a soft time constrains,

- systems with hard time constraints present a difficult challenge both to hardware engineers to software designers,

- system designer will attempt to reduce the number of activities (tasks) that are subject to a hard time constraint,

- most real-time systems contain a mixture of activities that can be classified as clock-based, event-based and interactive with both hard and soft time constraints (they will also contain activities which are not real time ones),

- characteristic of the various application are usually known a priori and tasks performing them may be scheduled statically or dynamically,

- static specification of schedules is typically the case for periodic tasks, the opposite is true for aperiodic tasks.

2. ***resource constraints:***

- task may require access to certain resources other than the CPU such as I/O devices, data structures, files, and databases,

3. ***precedence constraints:***

- a task may require results from one or more other tasks before it can starts its execution,

- complex task, for example, one requiring access to many resources, may be handled by decomposing it up into multiple subtasks taking into account precedence constraints,

- each subtask will require a subset of resources whose sequence is defined by precedence constraints,

4. ***concurrency constraints:***

- tasks should be allowed concurrent access to resources providing the consistency of the resources is not violated their temporal requirements,

5. ***dependability/performance constraints:***

- a task may have to meet certain availability, reliability and/or performance requirements

6. placement constraints:

- when multiple instances of a task are executed in order to achieve fault-tolerance, the different instances should be executed on different processors.

7. ***criticalness constraints:***

- depending on the functionality of a task, meeting the deadline of one task may be considered more critical than another,

Properties of Real-Time Systems

static/dynamic	**predictability**
reliability	**adaptability**
liveness	**fairness**
eventuality	**safety**

1. static or dynamic

2. predictability

- a system is predictable if it is possible to demonstrate at design time that constraints of all tasks can be met with 100% certainty,

- to prove predictability of a system, the total number of tasks, computation and resource requirements of all tasks and would-be changes in the environment over time should be known a priori,

- to answer whether a system is predictable the following questions relating to time must be answered:

- Will a task meet its temporal requirements?

- When will a task finish?

- How many tasks will finish before their deadline?

- What will be the maximum utilisation of a given resource?

- deadline guarantees are possible only if task characteristics like the execution and arrival times of tasks are given a priori,

- it is difficult to obtain exact information of task characteristics, so the worst case values are assumed or derived from extensive simulations, testing or other means,

- there are three types of predictability:

100% predictability

- there is 100% guarantee that constraints of some tasks will be satisfied,

probabilistic predictability

- either a certain number of tasks are guaranteed to meet their constraints or a task has a certain probability of meeting its constraints,

run-time deterministic predictability

- when a task is activated the system determines whether or not the task's constraints can be satisfied without affecting the guarantees provided to other tasks,

3. reliability:

- real-time constraints cannot be achieved if system components are not reliable,

- level of reliability should be specificable and predictability will define system performance under different levels of reliability.

4. adaptability:

- a real-time system should be adaptive to changes in the environment or the physical system (i.e. to changes in system dynamics including overloads and failures - short-term -, to changes in system configuration - long-term -,

- adaptability is particularly important for real-time systems because if a task's deadlines can be met only under a restricted system state/configuration, reliability and performance may be compromised,

- if a system is adaptive, one does not have to redefine the system or re-compute resource and task allocation for every small change, this reduces development and maintenance costs.

5. liveness:

- temporal property that guarantees the eventual occurrence of events,

6. fairness:

- guarantees occurrence of events sufficiently frequently but it is impossible to define "sufficient frequently",

7. eventuality:

- property that concerns both liveness and fairness and precludes the introduction of events or transitions that will never take place,

8. safety:

- guarantees the non-occurrence of certain events or their relegation to a very low probability of occurrence,

- it describes that "something bad never happens" i.e. certain events will not take place under certain global conditions

Application of Real Time Systems

- real-time systems are used in many applications with increasing complexity and economic and social importance,

- time scales of real-time applications may vary from milliseconds to microseconds,

INDUSTRY:

- process control in industrial plants,

- flight control in avionics,

- air traffic control,

- remote exploration of undersea, space and contaminated areas

- robotics with vision systems,

- defence and military applications,

- patient monitoring in health care,

INFORMATION TECHNOLOGY:

- computer networks and systems,

- database systems

- virtual reality,

- multimedia network,

- telecommunication,

1.2. TYPES of REAL-TIME SYSTEMS

- there are the following three types of rts:

clock- or time-based (or periodic) systems
event- or interrupt-driven (sporadic) systems
interactive systems

Clock- or Time-Based (or Periodic) Systems

- synchronisation in clock-based systems between the external processes and the internal tasks carried out by the computer is defined in terms of the relative time, or the actual time and the operations are carried out according to a time schedule,

- clock-based tasks are typically referred to as *cyclic or periodic tasks* where the terms can imply either that the task is to run once per time period "T" (or cycle time "T"), or is to run at exactly "T" unit intervals.

- time constant of the environment or the physical system is a measure of the time taken by environment or the physical system to respond to a change in input or load and is used as a characteristic of the environment or the physical system,

- the shorter the time constant of the process, the faster the required execution time,

- computer which is used to control an physical entity must therefore be synchronised to real time and must be able to carry out all the required operations within each time interval,

- completion of the operations within the specified time is dependent on the number of operations to be performed and the speed of the computer,

- synchronisation is usually obtained by adding a clock to the computer system normally referred to as *a real-time* clock - and using a signal from this clock to *interrupt* the operations of the computer at some predetermined fixed time interval,

- computer may carry out input, output and control tasks in response to the clock interrupt or, if the clock interrupt has been set at a faster rate than

the sampling rate, it may count each interrupt until it is time to run the tasks,

- in larger systems the tasks may be subdivided into groups for controlling different parts of the physical entity and these may need to run at different sampling rates,

- clock interrupt is frequently used to keep a clock and calendar so that the computer system is aware of both the time and the date,

Event- or Interrupt-Driven (Sporadic) Systems

- there are many systems where actions have to be performed not at particular times or time intervals but in response to some events,

- events occur at non-deterministic intervals and tasks, which handle them, are frequently referred to as *aperiodic* tasks,

- such tasks may have deadlines expressed in terms of having start times or end times (or even both),

- synchronisation in event-based systems between the external processes and the internal tasks carried out taking into consideration events occurred in the environment,

- specification of event-based systems usually includes a requirement that the system must respond within a given maximum time to a particular event,

- event-based systems are used for example to indicate alarm conditions and initiate alarm actions,

- event-based systems normally employ interrupts to inform the computer system that action is required,

- some small, simple systems may use polling; that is, the computer periodically asks (polls) the various inputs to see if action is required.

Remark: clock-based and event-based systems coexist in real-time applications,

Interactive Systems

- in interactive systems a set of operations in the computer should be completed within a predetermined time,

- interactive systems probably represent the largest class of real-time systems,

- real-time requirements are usually expressed in terms such as `the average response time must not exceed,

- similarly to the event-based system the interactive system apparently responds to a signal from the plant, but it is different because it responds at a time determined by the internal state of the computer and without an reference to the environment,

- many interactive systems give the impression that they are clock based in that they are capable of displaying the date and time; they do indeed have a real-time clock which enables them to keep track of time,

1.3. TYPES of COMPUTER PROGRAMS

- activities are divided into real-time and non-real-time tasks, and they are further divided into different types to support designing and implementing the different types of computer programs,

- definitions of programs are based on the kind of arguments which would have to be made in order to develop a formal proof of correctness for programs of each type,

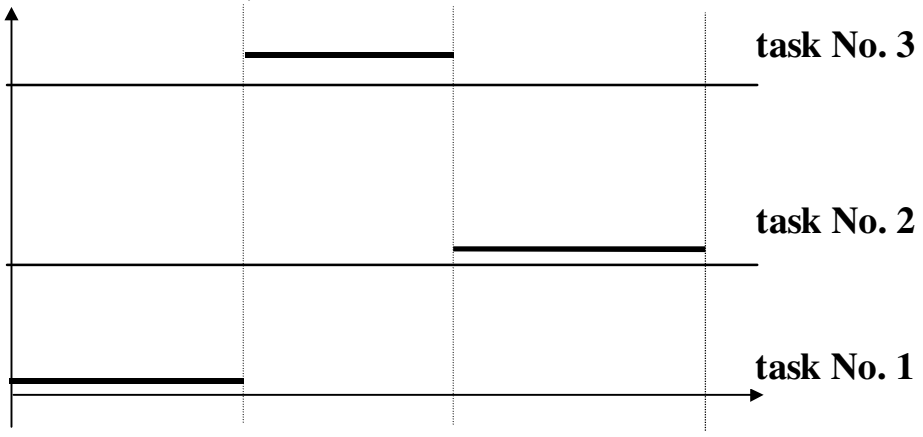
- types of computer programs:

sequential programs
multi-tasking programs
real-time programs

Sequential Programs

- actions, affecting a program, are strictly ordered by a time sequence,

- behaviour of the program depends only on the effects of the individual actions and their order,



time diagram of sequential program

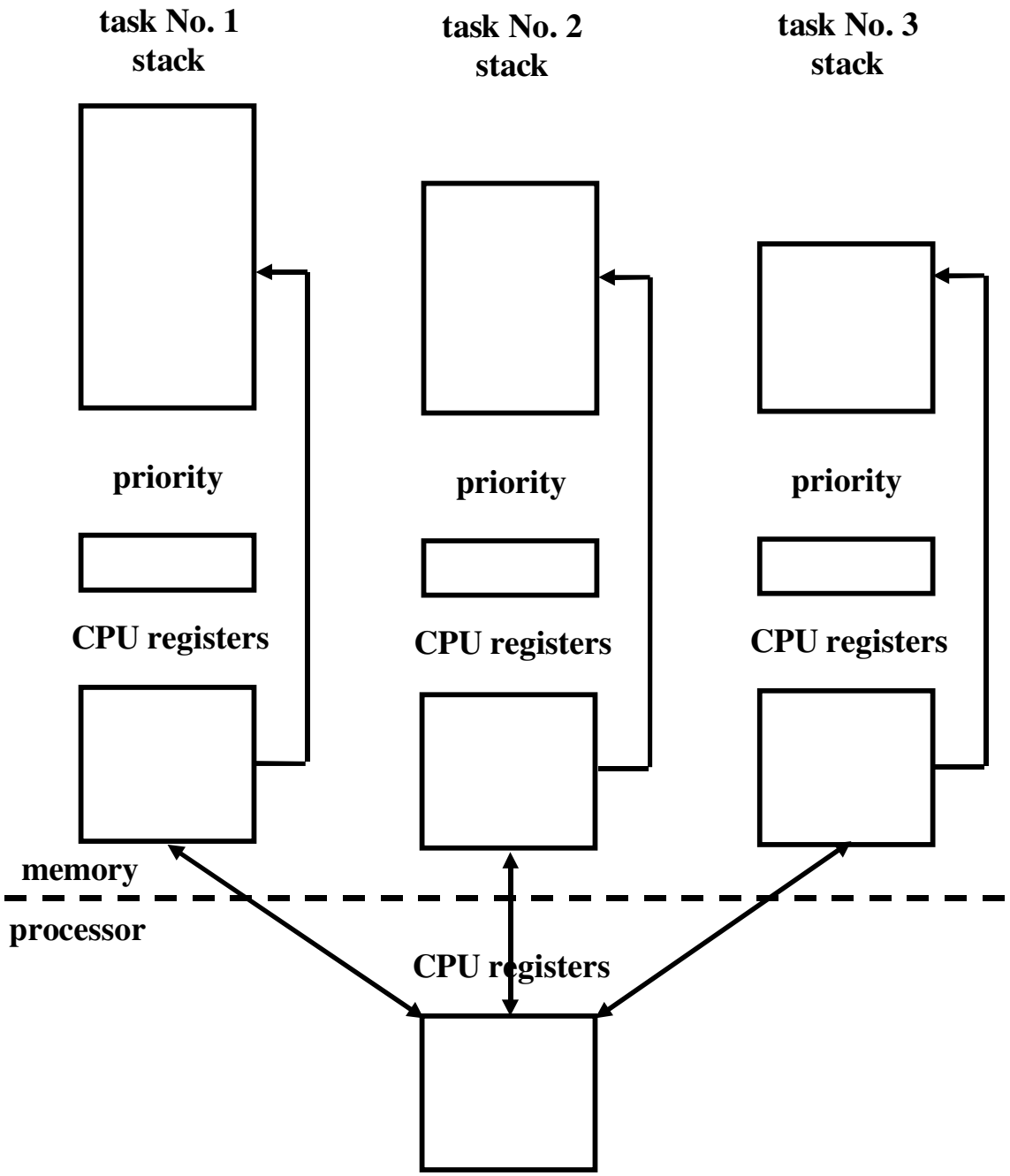
- verification of a sequential program requires the following two kinds of argument:

1. a particular statement defines a particular action; and

2. various program structures produce a corresponding sequence of events,

Multi-Tasking Programs

- actions to be performed are not necessarily disjoint in time, as a result it may be necessary for several actions to be performed in parallel,

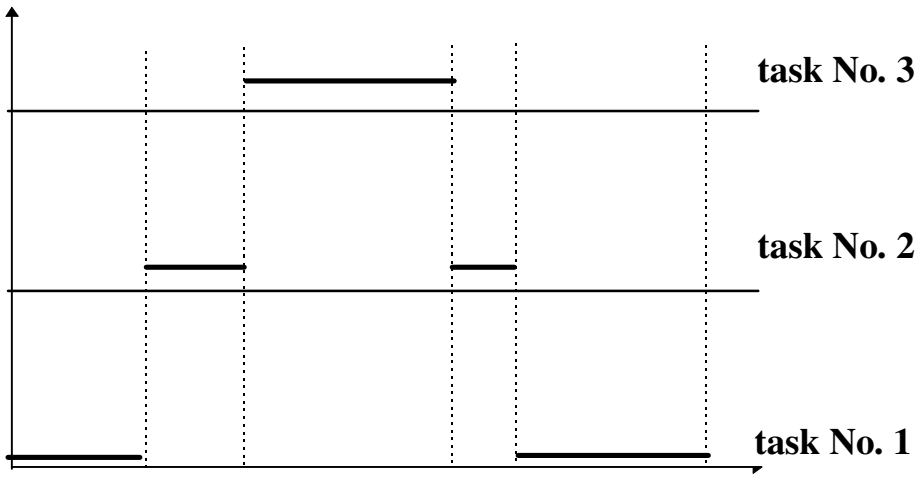


multitasking program

- a multi-tasking program may be built from a number of tasks which are themselves partly sequential, but which may be executed concurrently and which communicate through shared variables and synchronisation signals,

- multi-tasking is scheduling and switching the processor between several tasks i.e. the processor is switched between several sequential tasks,

- multi-tasking maximises the utilisation of the processor and supports modular structure of applications,



time diagram of multitasking program

- verification requires the application of arguments for sequential programs with some additions:

1. task (processes) can be verified separately only if the constituent variables of each task (process) are distinct

2. if the variables are shared, the potential concurrency makes the effect of the program unpredictable (and hence not capable of verification) unless there is some further rule that governs the sequencing of the several actions of the tasks (processes).

Real Time Programs

- actions of real-time programs are not necessarily being disjoint in time,

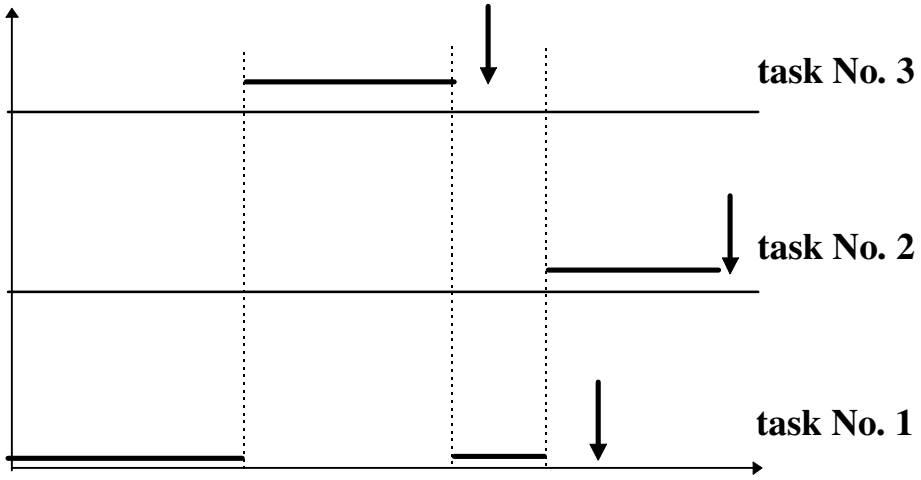
- sequence of some of its actions is not determined by the designer but by the environment - that is, by events occurring in the outside world which occur in real time and without reference to the internal operations of the computer,

- real-time program can still be divided into a number of tasks but communication between the tasks cannot necessarily wait for a synchronisation signal of the environment i.e. task cannot be delayed.

- in real-time programs, in contrast to the two previous types of program, the *actual time taken* by an action is an essential factor in the process of verification.

- design of real-time software is significantly more difficult than the design of sequential software,

- real-time features had to be built into the operating system which was written in the assembly language of the machine by teams of specialist programmers,



time diagram of real-time program

1.4. PERFORMANCE of REAL-TIME SYSTEMS

- designers or users have to estimate the worst-case run-times of real-time programs for a given application, source code and target architecture because it is necessary to estimate how long a particular program takes to run and whether it can meet deadlines of tasks

Performance Measures for Real-Time Systems

- rts-s are critical applications, and must be carefully designed and validated before being put into operation,

- validation includes checking design correctness using formal and informal methods and characterising performance and reliability,

properties of performance measures:

1. represent an efficient encoding of relevant information

- real-time systems often have high volume of information and interactions with the environment,

- there is required a context or framework in order first, to define relevant data secondly, to surpass irrelevant data,

- to have an efficient specification performance measure should be congruent to the application,

- the simpler the specification, the more likely it is to be correct and internally consistent,

2. provide an objective basis for the ranking of candidates for a given application

- performance measures must evaluate the "goodness" of computer systems in a given application,

- as a result, performance measures rank computer systems for the same application,

3. provide objective optimisation criteria for design

- the more complex a system, the more difficult to optimise it because there are numerous links between hardware and software components of the system,

4. represent verifiable data and facts

- performance measure that is impossible to define does not support design of rtss, thus, performance measures should have some extent accuracy

performance measures

reliability	**availability**
throughput	**performability**

<u>reliability:</u>

- probability that the system will not fail during a given interval of time,

- there are several types of reliability in rtss:

1. capacity reliability = probability of the system that it is not in failure states throughout some interval of time

- *expected capacity survival time* describes the expected time for the capacity reliability to drop to a specific value,

- *mean computation before failure* defines the mean amount of computation done before failure,

2. computational reliability = probability that the system can start a task "T" at time "t" and successfully execute it to completion,

<u>availability:</u>

- interval of time during that a system is available

computational availability = the expected value of the system computation capacity at any given time

<u>throughput:</u>

- average number of instructions per unit time that the system can

process

performability:

- **probability that the computer system will allow each performance level to be met,**

- **if there are *n* performance levels *A1, A2, .., An,* the performability of a real-time system is given by vector [P(A1), P(A2), ...,P(An)]**

  where *P(Ai)* - probability that the computer functions in such a way that the controlled process reach performance level *i*

- **performability connects real-time computer to the performance of the process that it controls**

- **the controlled process has several performance levels that are associated with the execution of control tasks**

drawbacks of performance measures:

- **reliability, availability and throughput does not afford interactions among hardware, systems software and application software**

- **availability cannot evaluate ability how a real-time systems meets deadline,**

- **throughput does not provide information on response time of tasks to several stimuli**

costs functions

- **functions that are obtained by comparing the performability of a real-time system with a zero response time to a system with given positive response time**

Estimation of Run Times of Real Time Programs

- it is necessary in rtss to accurately estimate run times of programs to answer question whether a real-time system meets deadlines,

- run-time of a program depends on the following factors:

- source code	**optimisation and tuning**
- compiler	**mapping between source code and machine code**
- operating system	**task scheduling, memory management, interrupt handling**
- hw architecture	**interactions between processor, memory, I/O devices, bus of the computer,**

- to define run-time or its time bounds these factors should taken into account, but this analysis is difficult because it should include every aspects of the execution and interactions between several components of the computer system,

- there is only a partial solution to the problem of estimation of run-times of programs, because to estimate run-time of a program on a target hardware in a real-time environment inputs, outputs, external and internal events, their time constraints,

analysis of source code:

- execution time of straight-line code depends on hardware architecture, and the compiler,

- execution time of a straight-line code is given by the equation if there are no interrupts:

$$\sum_{i}^{n} T_{exec}(L_i)$$

where $T_{exec}(L_i)$ - time needed to execute instruction L_i

- unfortunately, the execution time is data-dependent i.e. execution time some operation depends on their input data

estimation procedure is divided into the following steps:

pre-processing = produces assembly code and marks off blocks to be analysed

parser = analyses the input source code

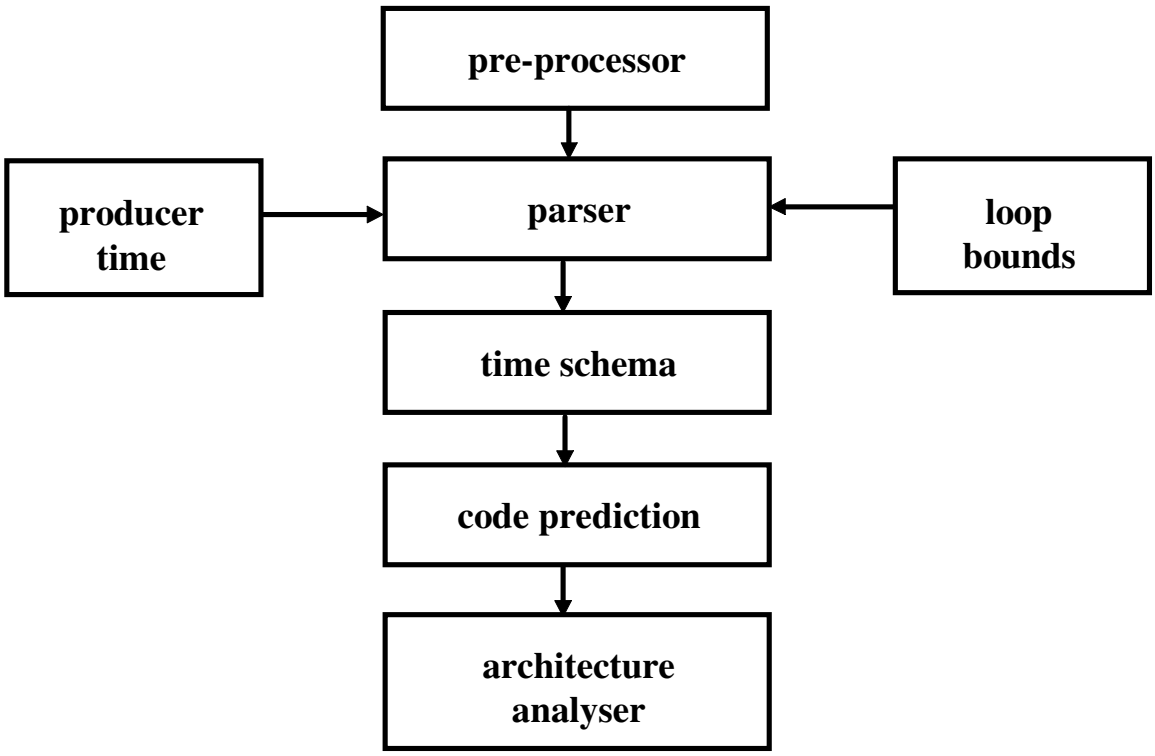
procedure time = maintains a table of procedures and their execution time

loop bounds = obtains bounds on the number of iteration of loops

time schema = computes execution time of each block using the execution time estimates computed by code prediction module

code prediction = computes execution time estimates of each block using the code generated by the pre-processor

architecture analyser = takes into account the influence of the computer architecture



1.5. DESIGN of REAL-TIME SYSTEMS

- design of real-time systems requires the accommodation of diverse task constraints,

- determination of task constraints requires a good understanding of the problem and an optimal solution to a number of closely related high level problems that are as follows:

1. problem decomposition at system level

2. identification of tasks

3. determination of system architecture

4. allocation of tasks to processors

5. scheduling tasks taking into account resource and time constraints

Design and Implementation Issues

constraints on system behavior	**resource allocation**
architecture	**criticality**

1. constraints on system behavior

- demand for processing in rts may change over time as a result of number of periodic and sporadic tasks,

- tasks of rts have strict time constraints for example deadlines that allocate computing to several time interval,

- access to resources as processor, peripheral and other resources, may be subject to various forms of constraints for example mutual exclusion,

- tasks may have precedence constraints affecting their order of execution imposed by the processing requirements of the application,

- placement constraints on tasks are defined by security reasons or fault tolerance,

2. resource allocation

- optimum use of available resources is closely related to the granularity of tasks, timing constraints of the application, processor capabilities, resource limitations and the communication network,

- the granularity of tasks defines the overall requirements of communication, performance and processing,

- problem decomposition and grouping of computational activities into tasks have major influence on optimal resource utilisation,

- to achieve optimum resource utilisation the following steps have to be performed during the design and application phase:

1. design phase:

- software partitioning that provides the best distribution of computational activities,

- task allocation takes into consideration the hardware of the computer and improves resource utilisation,

2. application phase

- task scheduling orders tasks in time for execution on the allocated processors in such a way that time constraints are all satisfied

3. architecture

- several hardware architectures are used in rts, selection of a particular architecture depends on the application,

- rts are implemented more and more as distributed systems that provide effective use of resources though load sharing, improve fault tolerance,

4. criticality

- computer systems are too complex for guaranteeing their fail-safe operation,

- as a result, there must be safeguards against failures that ensure ability of the system to continue to provide the service fully or partially or to guarantee the safety of any human life involved, the environment and the system itself,

- reliability and safety are achieved through fault tolerance using recovery mechanisms and redundancy methods,

1.6. STATE of ART in REAL-TIME SYSTEMS

Specification and Verification.

- existing practices for designing, validating, and implementing real-time systems are rather ad hoc today,

-design and application engineers are interested not only in the logical properties of safety and liveness but also in the physical property of timeliness.

- problem of specification, design, and analysis of real-time programs is more difficult than in sequential programs as a result of

- **parallelism inherent in real-time applications,**
- **presence of hard or soft real-time constraints,**

- specification for real-time systems is a kind of contract between the system and the environment,

- as a result of specification, based on certain assumed properties of the environment, the system guarantees to provide the specified performance,

- design of the system should not only meet the specifications but also guard the system against faults in the behaviour of the environment (i.e., against deviations from its assumed behaviour),

- specification and verification techniques are required to provide answers to the following questions:

- What is the worst-case response time of the system to a particular type of external event?

- Is the system performance under peak arrival rates adequate?

- What is the effect of different operating system policies, such as scheduling policies on safety and timeliness?

- Given the specification of a system, do the specifications conform to the

safety properties of interest to the system?

- Given an implementation, how can one ensure that the implementation conforms to the specifications?

- What are the restrictions imposed on the external environment for the system to react safely?

- What are the rates at which stimuli can arrive at the system?

- What will be the effect of failures in the external environment?

specification & verification methods:

- temporal logic,

- finite state machines,

- Petri nets,

- concurrent sequential processes /CSP logic/

- first order logic,

Real-Time Kernels and Operating Systems

- RTOS must be able to perform integrated CPU scheduling and resource allocation so that collections of co-operating tasks can obtain the resources they need, at the right time, in order to meet timing constraints,

- kernels of RTOSs perform multi-linked, inter-task communication and synchronisation that are achieved via mailboxes, proxies, events and semaphores,

- key issue that has emerged in rts-s is the need to provide predictability, that requires clean operating system primitives, some knowledge of the application, proper scheduling algorithms, and a viewpoint based on a team attitude between the operating system and the application

- to reduce the run-time overheads incurred by the kernel and to make the system fast the following conditions should be met:

1. the kernel underlying a current real-time system has a fast context switch that has a small size with associated minimal functionality,

2. it is provided with the ability to respond to external interrupts quickly to minimise intervals during which interrupts are disabled,

3. it also provides fixed or variable sized partitions for memory management, i.e., no virtual memory as well as the ability to lock code and data in memory,

4. it has special sequential files that can accumulate data at a fast rate,

5. to deal with timing requirements, it maintains a real-time clock. The kernel provides special alarms and time-outs tasks that can invoke primitives to delay by a fixed amount of time and to pass and/or resume execution.

- it is often the case that existing real-time systems are supported by stripped-down and/or optimized versions of time-sharing operating systems,

Scheduling

- primary criteria in task scheduling are the execution times of tasks taking into consideration time constraints,

- time constraints are an indication of the level of urgency of a real-time task, and the criticalness of a task is indicative of the level of importance,

- task execution times are, in general, stochastic, the worst-case execution time of a task may be much higher than its average execution time,

- providing enough processing power so that even at peak loads all tasks will meet their timing requirements will result in a highly under-utilised system, but real-time systems are called upon to perform efficiently and correctly especially in overloaded situations,

solution: to perform scheduling such that the highly critical tasks are always guaranteed to meet their timing constraints,

- today's real-time systems continue to use priority-driven schedulers for scheduling tasks with time constraints, that have to take into considerations two competing components: time constraints and criticalness values of tasks, that are mapped into one factor, the priority of a task,

- this is done through an iterative process involving extensive simulation and it continues until a satisfactory performance is achieved in two steps:

1. task priorities are adjusted if some critical tasks miss their deadlines or if the resource utilisation is poor,

2. optimisation of task code is often performing order to meet time constraints,

- priority-based scheduling technology involves a lot of code and priority tuning accompanied by simulations to gain confidence in the system,

- rts "design philosophy" forces designers and engineers to deviate from structured system development techniques whereby the resulting system is neither adaptive nor extensible,

Real Time Languages

- minimal requirements that should be satisfied by any language used in real-time systems:

- rts languages should be high-level so that a user need not be very intimately involved with the underlying hardware architecture,

- modules of rts languages should be programmable such that they have (a priori determinable) bounded execution times,

- use of dynamic structures and unbounded loops should be disallowed or carefully conceded,

- all program constructs and modules should be analysable with respect to the schedulability of modules, availability of resources, and satisfaction of system-timing constraints,

- analysability of programs written in a real-time language,

problems:

- no appropriate rts language and lack of appropriate automatic analysis techniques,

- designers and programmer explicitly controls parallelism, priority, and timing at "low level,

Architectures and Structures

- rts architectures are divided into two levels:

1. node level

- each node may include application, system and network processors; shared memory segment and I/O interfaces,

- each processor must provide speed and predictability in executing real-time tasks, handling interrupts and interacting with their environment,

- it is provided by making instruction execution, memory accesses and context switching as predictable as possible,

- to achieve predictability rts do not use virtual memory and caches because first, page faults and/or uncertainty of cache hit and miss may result in unpredictable or long delays in memory access,

2. system level

- communication among nodes and fault tolerance makes difficult to achieve predictability

- distributed rts contains a set of nodes communicating with each other through a communication network

- requirements to hw architectures and structures:

- hardware architectures must provide meeting of real-time constraints and fault tolerance,

- application modules or processors are special purpose off-the-shell hardware modules that do should not change with each application,

- system and network processors or modules are custom designed ones to provide the support required for real-time applications,

- network processors supervise communication, i.e. they manage forwarding messages from a source to a destination to support for

meeting deadlines, time management and house-keeping

- capacity and size of real-time systems are strongly influenced by the growth of overhead as modules are added,

- program code of real-time computers might have to be in ROM to withstand harsh environments,

- memory modules should provide a fast and reliable communication between the processors at a node,

- to decrease system overhead of processors and to keep application tasks free local memories can be used for read-only code,

- global memory can contain shared data because dynamically loaded local memories and caches cause many problems difficult to deal with under time constraints,

- functional partitioning of program code should be performed, but too rigid partitioning and static schedule would be not suitable in many systems,

- the communication network for small and "old" systems is a custom-designed broadcast bus with redundancy, while for large and "new" systems is a high-speed token ring or a point-to-point network

- I/O devices are sensors, actuators and displays that are distributed in real-time systems and managed by I/O controllers

- real-time computers must also include on-line testability to increase reliability and support fault tolerance,

Fault Tolerance

- the real-time system must be predictable even in the presence of faults because if the system is unable to meet the timing constraints can be viewed as a fault,

- fault tolerance must be designed into real-time systems and it must encompass both hardware and software fault tolerance taking into account time constraints,

- fault tolerance is achieved by predictable redundancy management for example a set of servers on multiple processors provide fault tolerance for crash or performance failures,

- predictable redundancy management is based on synchronisation of servers, agreement on the order of request messages and the cost of failure detection and recovery,

- to meet timing constraints in rts after appearing a fault imposes additional demands on the scheduling algorithms, because traditional scheduling algorithms do not consider fault tolerance,

- primary and back-up schedules computed off line are often found in hard real-time systems to provide fault tolerance,

- on-line schedulers predict that time constraints will be missed, enabling early action on such faults, it leads to dynamic re-configurability,

- in real-time environment all constraints but timing constraints are traded because to meet timing constraints is essential to ensure correct system behaviour, it may lead to graceful degradation, i.e. a fault in rts could result in a temporary reduction in the quality of services provided to allow to system to meet critical task deadlines,

- highly-reliable systems need to employ proof-of-correctness techniques to ensure fault tolerance properties,

- fault tolerance techniques that explicitly take into account timing of application fault tolerance in real-time systems are lacking,