REAL-TIME DISTRIBUTED SYSTEMS - DEFINITIONS AND EXAMPLES

These two types of systems are readily combined because:

- Most distributed systems (of any complexity) have some temporal performance constraints, and
- Most real-time systems (of any complexity) have multiple distributed processing elements.

Common features:

- We have sufficient complexity and/or sufficiently rigorous performance specifications to justify considerable design effort - e.g. safety/mission/life critical applications.
- Concurrency - co-operating concurrent processes or tasks that are mostly independent but must communicate at intervals.

Advantages of concurrent tasking:

- A natural model for many real-world applications - the real-world is concurrent.
- Separation of functionality - easier to understand and design.
- Overall reduction in execution time for the system can be achieved - parallel I/O operations and easy migration to physical parallelism.
- Improved scheduling flexibility - critical tasks can be assigned an appropriately high priority.

Disadvantages of concurrent tasking:

- Too many concurrent tasks can produce an overly complex design - increased scheduling and communication overhead.
- More difficult to model and verify performance.

REAL-TIME SYSTEMS

Significant feature - 'correctness' of operation is dependant on:

- logical and functional operation
- temporal properties

Oxford Dictionary of Computing Definition:

> "Any system in which the time at which output is produced is significant. This is usually because the input corresponds to some movement in the physical world, and the output has to relate to that same movement. The lag from input time to output time must be sufficiently small for acceptable *timeliness*"

Timeliness must be evaluated with respect to some timescale:

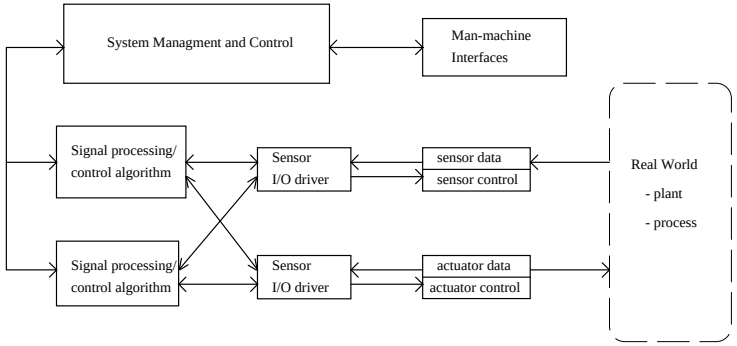
- avionics or nuclear power plant control - response times of msec are required.
- e-commerce - response times of seconds
- mineral processing - response times of several seconds

A broad classification of Real-Time Systems:

- *hard* - critical that responses occur within a deadline or the system fails
- *soft* - system still functions if deadlines are missed but some "responsiveness" is desired (we could call these *interactive* systems)
- *batch* - no coupling with Real-Time at all

Another important characteristic of RT systems is the strength of the coupling they have with the real physical world through instrumentation, e.g. usually doing some monitoring and/or control of some physical process.

Typically, the RT system is a component in a much larger engineering system - they have also become known as *embedded systems*:



A distinguishing feature of embedded systems is that they usually only provide an *execute-only* environment - the workload is well-defined and fixed. Transient inputs can be handled - their processing requirements are predefined and **are included** in the system design.

Real-Time systems are also *reactive systems*, i.e. they are event driven and respond to external stimuli - the response is usually state dependent as well, e.g. building lift-control.

Real-Time systems are also used in *control systems*, i.e. they make control decisions based on input data usually without any human intervention, e.g. automotive cruise-control.

A Brief Historical Perspective

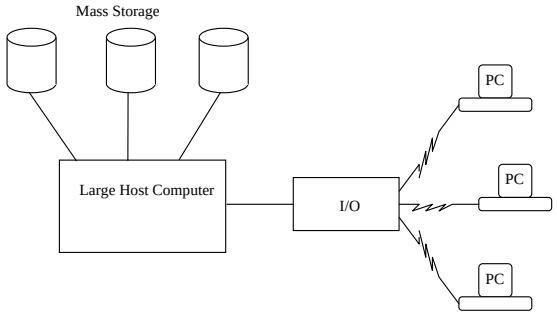
- Started with replacement of the analogue processing section of control systems with digital computers in the late 1950's, e.g. 1959 - replacement of the polymerization unit in an oil refinery using an RW-300 computer - 26 flows, 32 temperatures and 3 pressures.
- 1962 - Ferranti computer replaced analog instrumentation in an ICI chemical plant - 129 valves and measured 224 nodes.
- 1960's - Mercury, Gemini & Apollo programs - proprietary aerospace hardware and early development of fault-tolerant hardware and software.
- 1970's - Widespread commercial use of embedded systems exploded with the advent of the minicomputer, e.g. the DEC PDP family, Data General Nova, Honeywell Microdata 800, HP 2100, TI 960
- 1980's - Microprocessor technology $\rightarrow$ upsurge in number of real-time systems $\rightarrow$ growth in distributed RT systems, e.g. Space shuttle computer system - distributed, real-time & fault-tolerant.
- 1990's - distributed, real-time, fault-tolerant, object-orientated - e.g. Levi & Agrawala's MARUTI project, Sun Microsystems' DOE project $\rightarrow$ SPRING, CORBA, and many others.
- 2000's - move towards highly available, highly reliable and highly mobile global enterprise information portals with real-time responsiveness in both B2C and B2B applications.

DISTRIBUTED SYSTEMS

Concurrent applications executing on multiple nodes which are usually physically distributed. The huge increase in distributed systems applications is mostly due to the removal of constraints on computer/information systems designers:

- improved interconnection capability
- improved processor capacity
- improved software design methodology
- improved scalability
- improved cost-benefit ratio

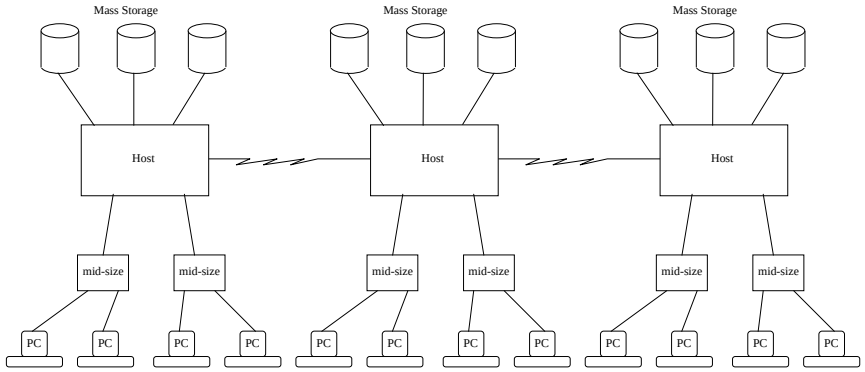
Consider the historical evolution via this basic configuration:



Distributed processing – initial aim was to *off-load* functionality from mainframes to PC's, e.g. word processing, spreadsheets, or private database products.

Many variations on this structure are possible, e.g. I/O could be a minicomputer → a hierarchy of processing systems.

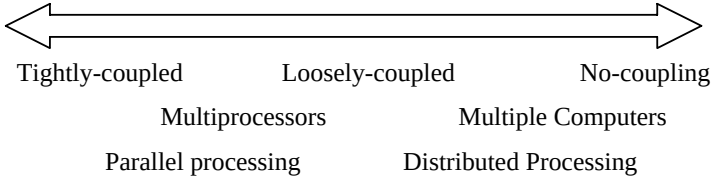
An common extension to this arrangement is the peer interconnection - i.e. similar capabilities on all interconnected systems - usually avoids a central point of control and synchronization. The *hierarchic peer interconnection* combines architectural features of both structures, e.g.:



Distributed processing or Parallel/Multi processing?

Only difference is the extent of *space-time coherence* - multiprocessors have higher space-time coherence than distributed systems:

- A typical tightly-coupled multiprocessor has processors that share devices and memory under the control of the same operating system.
- A typical distributed system has complete computers with separate operating systems and separate datasets.



Historical Perspective:

- mid 1970's - a realization that large centralized systems were no longer cost-effective for all applications, previously driven by "economies of scale" - a large number of the same components in a large machine.

- "Economies of scale" now applied to a large number of less specialized and lower component count systems.

- Due to underlying technology reasons, I/O devices also achieved economies of scale → allowed the minicomputer to move from process control to commercial data processing

- Late 70's - prediction that centralized systems would disappear - but the software/security complexity of distributed systems allowed the role of centralized systems to be reestablished into the 1980's

- 1980's - advent of the PC has also extended the life of mainframe systems → encouraged the shift of the major distributed processing model from inter-connected mid-sized machines to large mainframes surrounded by 100's of PC's. Mid-sized machines are often interposed at the departmental level between organizational level mainframe computers

- late 1980's-90's - expansion in network capabilities - distributed processing systems now approaching I/O system capability → encouraged connection of large numbers of mid-range machines

- 2000's - heterogenous mix of processing systems at all levels from mobile PDAs to large server clusters

- Software support for distributed systems <u>is still</u> the major problem area - some important issues:
  - different end user 'views' of different systems
  - management of heterogeneous database structures
  - functional decomposition of software

REAL-TIME DISTRIBUTED SYSTEMS - A FEW GOOD (BAD) EXAMPLES

PATRIOT missile battery radar system
(ref: Science, March 1992)

25 Feb. 1991, Dharan, Saudi Arabia - SCUD missile kills 28 US servicemen when the PATRIOT battery fails to intercept the missile due to software error.

PATRIOT (mil-spec) processor - 20 years old (24 bits).

Reason for failure: **TIME** handling fault

Time - maintained by internal clock in 1/10 secs as integer from start of system operation.

To track SCUD location - time & velocity are converted to reals (conversion is no more precise than 24 bits).

The effect on the "range-gate" accuracy is proportional to the target's velocity and length of time the system has been running → the range-gate progressively shifts away from the expected target position with increasing system run time.

Typically: 8 hours run time → 55 m error
100 hours run time → 687 m error

Error was found by the Israeli military through testing - fault was notified but the correction to the software arrived one day too late!

Lesson: not just a real-time related software coding error, but a design flaw, QA/QC testing failure, and total 'systems' failure.

THERAC-25 Accidents (1985-1987)
(ref: IEEE Computer, July 1993)

- Controller for a dual-mode (photons/electrons) linear accelerator (25 Mev) for medical radiation treatment
- Single PDP-11/23, using assembly language, one programmer over several years
- Software performs these functions:
  - monitors machine status
  - setup machine for treatment - moves beam servos
  - activates the beam on/off
- A proprietary operating system was not used - real-time executive specially written - uses preemptive scheduling for critical tasks
- Major problems:
  - software allowed concurrent access to shared variables
  - test & set not indivisible
  - race conditions occurred
  - operator actions within narrow time frames was necessary to cause the accidents
- Results:
  Several deaths and serious injuries due to over irradiation
- Lessons:
  - too much faith put in software (hardware interlocks should not have been removed).
  - poor documentation & test plan (minimal module level testing done)
  - getting balance of safe versus "user-friendly" wrong
  - "... for complex interrupt-driven software, timing is of crucial importance"

Collins class submarines (1993-)
(ref: 25 March 1998 ANAO report #34 www.anao.gov.au/reports.html)

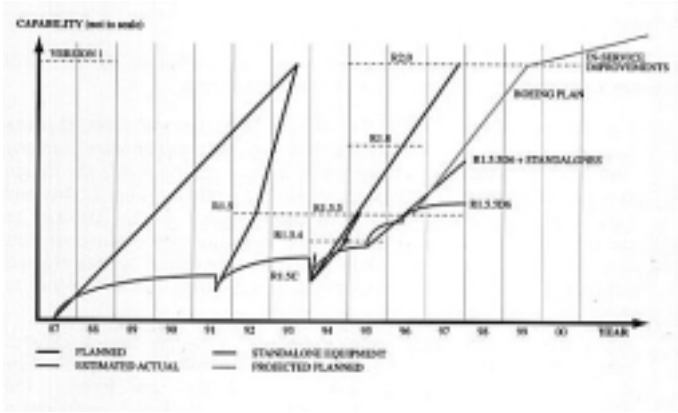
September 1987: ASC contract with Rockwell Ship Systems Australia (now known as Boeing Australia Limited) for six Collins-class submarines.

September 1993: Combat system software was first scheduled for delivery and integration into Collins (contract price $837 million). The Tactical Data Handling Subsystem (TDHS) manages all the electronic data and provides the means by which targets are engaged.

The Collins-class submarine combat system design utilises multi-function operator consoles to overcome the disadvantages of dedicated process-unique operator consoles found in earlier combat systems.

From the ANAO report:
"*.... there were high-risks that development past 60 per cent of the specified requirement would be hampered by memory and data processing and distribution limitations already built into the TDHS system*."

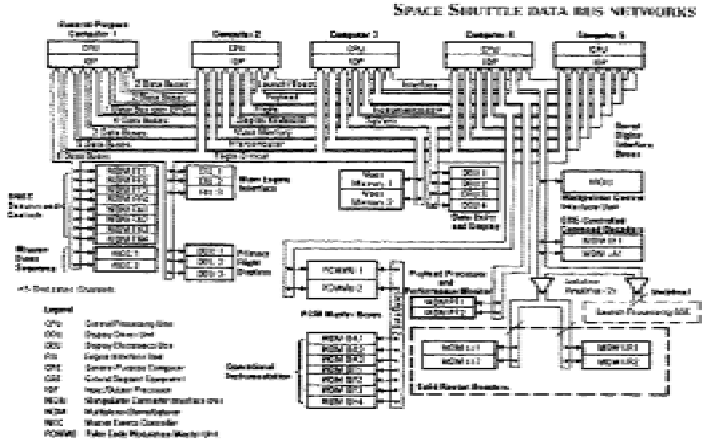
Combat System Capability Over Time - Commonwealth's Assessment



One finding: Need a cost/benefit analysis for the replacement of the current Tactical Data Handling System with products which are more technologically advanced and less costly to maintain and enhance.

STS-1 delayed launch (April 1981)

(ref: www.history.nasa.gov/sts1)



The first launch of the space shuttle was delayed for 2 days, at T-20 minutes before launch.

The backup computer system was unable to synchronize with the redundant set of primary computers (all AP-101B's).

A change 2 years earlier to a module performing bus initialization → record containing a time 40 msec in error was sent to the synchronization data area (which was not picked up by testing or caused a problem). One year later this time was increased to 65 msec due to another change and this resulted in a 1 in 67 chance of synchronization failure.

For the first launch this bus synchronization timing error occurred and three of the four primary computers were synchronized one cycle late relative to the first computer → they could not agree.

A fail-safe device prevented the fifth backup computer initializing when the other four were not in agreement → failed to initialize → launch automatically aborted.

AND importantly the difference in the synchronization module on the five computers was related to the different origins of the software – IBM for the primary avionics software and Rockwell for the backup avionics software (running on the fifth computer).

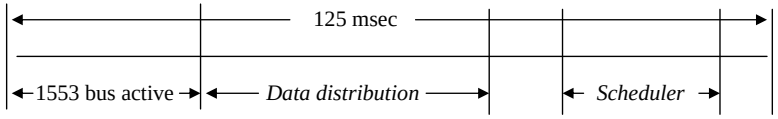
"What really happened on Mars?" (Dec 1997)

(ref: http://www.research.microsoft.com/~mbj/Mars_Pathfinder/Authoritative_Account)

The NASA Mars Pathfinder vehicle used a single processor (RS6000) on a VME bus for all spacecraft control. The VME bus contained interface cards for the radio transceiver, camera, and a 1553 high speed Mil-spec interface bus. The 1553 bus was used to interface to various spacecraft systems: e.g: thrusters, sensors and the meteorological (*ASI/MET*) payload.

The software was implemented on VxWorks (a commercial RTOS) using pre-emptive priority scheduling of tasks. An 8 Hz primary cycle rate was inherited from hardware and software reused from an earlier spacecraft (Cassini).

Tasking timeline



Task priorities: *Scheduler*
Spacecraft entry and landing
Data distribution
Other spacecraft functions & then science functions

Data collected from the 1553 bus is collected in a double buffered shared memory area and delivered via IPC mechanisms (i.e. *pipe()*). Tasks wait on various IPC queues using a *select()* mechanism to wait for messages.

Under high data load the *Data distribution* task was still running when the *Scheduler* task was executed which caused a deliberate hardware reset and termination of the commanded activity for the day.

The key cause of the failure was **priority inversion** (the *Data distribution* task was blocked by a much lower priority *ASI/MET* science task that held a shared resource). Several medium priority tasks ran pre-empting the *ASI/MET* task delaying the return of the shared resource and causing the intermittent priority inversion → system reset → loss of data.