Distributed Systems – Techs
11. Real time systems

What is a real-time system?

- real-time programs (and systems) interact with the external world in a way that involves time.
- When a stimulus appears, the system must respond to it in a certain way and before a certain deadline.
- If it delivers the correct answer, but after the deadline, the system is regarded as having failed.
- When the answer is produced is as important as which answer is produced.

Examples

- An audio compact disk player consists of a CPU that takes the bits arriving from the disk and processes them to generate music.
  - Suppose that the CPU is just barely fast enough to do the job.
  - Now imagine that a competitor decides to build a cheaper player using a CPU running at one-third the speed.
  - If it buffers all the incoming bits and plays them back at one-third the expected speed, people will wince at the sound, and if it only plays every third note, the audience will not be wildly ecstatic either.
- Time is inherently part of the specification of correctness here.

Other examples

- computers embedded in television sets and video recorders,
- computers controlling aircraft ailerons and other parts (so called fly-by-wire),
- automobile subsystems controlled by computers (drive-by-wire?),
- military computers controlling guided antitank missiles (shoot-by-wire?),
- computerized air traffic control systems,
- scientific experiments ranging from particle accelerators to psychology lab mice with electrodes in their brains,
- automated factories,
- telephone switches,
- robots,
- medical intensive care units,
- CAT scanners,
- automatic stock trading systems,

…

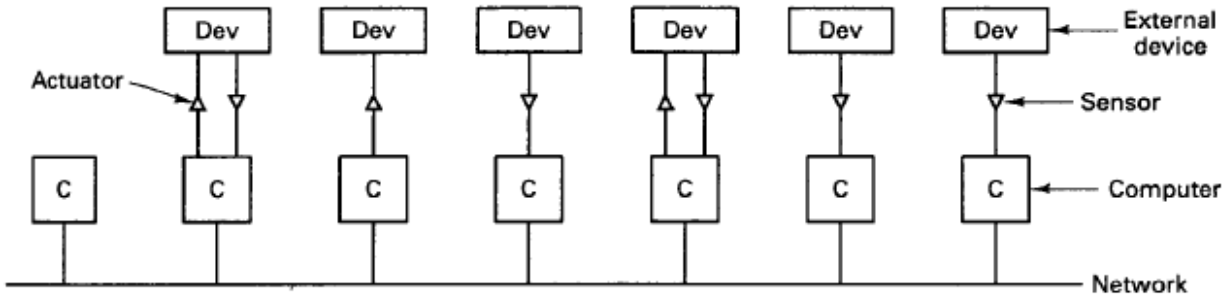
Stimulus

- An external device (possibly a clock) generates a *stimulus* for the computer, which must then perform certain actions before a deadline.
- When the required work has been completed, the system becomes idle until the next stimulus arrives.
- The stimulii are

1. Frequently, *periodic*, with a stimulus occurring regularly every AT seconds, such as a computer in a TV set or VCR getting a new frame every 1/60 of a second.
2. Sometimes, *aperiodic*, meaning that they are recurrent, but not regular, as in the arrival of an aircraft in a air traffic controller's air space.
3. Finally, some are *sporadic* (unexpected), such as a device overheating.

Structure

- Distributed real-time systs can often be structured as illustrated in Fig.
- Here we see a collection of computers connected by a network.
  - Some of these are connected to external devices that produce or accept data or expect to be controlled in real time.
  - The computers may be tiny microcontrollers built into the devices, or stand-alone machines.
  - In both cases they usually have sensors for receiving signals from the devices and/or actuators for sending signals to them.
  - The sensors and actuators may be digital or analog.



Two types

- two types depending on how serious their deadlines are and the consequences of missing one. These are:

1. Soft real-time systems.
   - missing an occasional deadline is all right.
   - For example, a telephone switch might be permitted to lose or misroute one call in under overload conditions and still be within specification.
2. Hard real-time systems.
   - even a single missed deadline in a hard real-time system is unacceptable, as this might lead to loss of life or an environmental catastrophe.

- In practice, there are also intermediate systems where missing a deadline means you have to kill off the current activity, but the consequence is not fatal.
  - For example, if a soda bottle on a conveyor belt has passed by the nozzle, there is no point in continuing to squirt soda at it, but the results are not fatal.
- In some real-time systems, some subsystems are hard real time whereas others are soft real time.

Design Issues

- **Clock Synchronization.** The first issue is the maintenance of time itself. With multiple computers, each having its own local clock, keeping the clocks in synchrony is a key issue.
- **Event-Triggered versus Time-Triggered Systems.**
- **Predictability.**
- **Fault Tolerance.**
- **Language Support.**

Event-triggered real-time system

- When a significant event in the out-side world happens, it is detected by some sensor, which then causes the attached CPU to get an interrupt.
- Event-triggered systems are thus interrupt driven.
- For soft real-time systems with lots of computing power to spare, this approach is simple, works well, and is still widely used.
- The main problem with event-triggered systems is that they can fail under conditions of heavy load, that is, when many events are happening at once.
  - For example, what happens when a pipe ruptures in a controlled nuclear reactor.
    - Temperature alarms, pressure alarms, radioactivity alarms, and other alarms will all go off at once, causing massive interrupts.
    - This *event shower* may overwhelm the computing system and bring it down, potentially causing problems far more serious than the rupture of a single pipe.

Triggered real-time system

- A clock interrupt occurs every d*T* milliseconds.
- At each clock tick (selected) sensors are sampled & (certain) actuators are driven
- No interrupts occur other than clock ticks.
- Example of the difference between these two approaches: consider the design of an elevator controller in a 100-story building.
  - Suppose that the elevator is sitting peacefully on the 60th floor waiting for customers.
  - Then someone pushes the call button on the first floor.
  - Just 100 msec later, someone else pushes the call button on the 100th floor.
  - In an event-triggered system:
    - The first call generates an interrupt, which causes the elevator to take off downward.
    - The second call comes in after the decision to go down has already been made, so it is noted for future reference, but the elevator continues on down.
  - A time-triggered elevator controller that samples every 500 msec.
    - If both calls fall within one sampling period, the controller will have to make a decision, for example, using the nearest-customer-first rule, in which case it will go up.
- In summary:
  - Event-triggered designs give faster response at low load but more overhead and chance of failure at high load.
  - Time-trigger designs have the opposite properties & are only suitable in a relatively static environment in which a great deal is known about syst behavior in advance
- Which one is better depends on the application

Predictability

- Important properties of any real-time system: its behavior is predictable.
- Ideally: the system meet all of its deadlines, even at peak load.
- Most DS designers are used to thinking in terms of
  - independent users accessing shared files at random
  - or numerous travel agents accessing a shared airline data base at unpredictable times.
- This kind of chance behavior rarely holds in a real-time system.
  - It is known that when event E is detected, process X should be run, followed by Y and Z, in either order or in parallel.
  - Furthermore, it is often known (or should be known) what the worst-case behavior of these processes is.
  - For example, if it is known that X needs 50 msec, Y and Z need 60 msec each, and process startup takes 5 msec, then it can be guaranteed in advance that the system can flawlessly handle five periodic type E events per second in the absence of any other work.

Fault Tolerance

- Many real-time systems control safety-critical devices in vehicles, hospitals, and power plants, so fault tolerance is frequently an issue.
- Active replication is sometimes used, but only if it can be done without extensive (and thus consuming) protocols to get everyone to agree on everything all the time.
- Primary-backup schemes are less popular because deadlines may be missed during after the primary fails.
- A hybrid approach is to follow the leader, in which one machine makes all the decisions, but the others just do what it says to do without discussion, ready to take over at a moment's notice.
- In a safety-critical system, it is especially important that the system be able to handle the worst-case scenario
  - it is not enough to say that the probability of three components failing at once is so low that it can be ignored.

Fail-safe

- Failures are not always independent.
  - For example, during a sudden electric power failure, everyone grabs the telephone, possibly causing the phone system to overload, even though it has its own independent power generation system.
  - The peak load on the system often occurs precisely at the moment when the maximum no. of components have failed because much of the traffic is related to reporting the failures.
- Consequently, fault-tolerant real-time systems must be able to cope with the maximum number of faults and the maximum load at the same time.
- Some real-time systems have the property that they can be stopped cold when a serious failure occurs.
  - E.g. when a railroad signaling system unexpectedly blacks out, it may be possible for the control system to tell every train to stop immediately.
  - If the system design always spaces trains far enough apart & all trains start braking more-or-less simultaneously, it will be possible to avert disaster & system can recover gradually when the power comes back on.
  - A syst that can halt operation like this without danger is said to be *fail-safe*.

Language Support

- While many RT systs and applications are programmed in languages such as C, *specialized real-time languages* can potentially be of great assistance.
- For example, in such a language, it should be easy to express the work as a collection of short tasks (e.g. threads) that can be scheduled independently, subject to user-defined precedence and mutual exclusion constraints.
- The language should be designed so that the maximum execution time of every task can be computed at compile time.
  - This requirement means that the language cannot support general while loops. Iteration must be done using for loops with constant parameters. Recursion cannot be tolerated either.
  - Even these restrictions may not be enough to make it possible to calculate the execution time of each task in advance since cache misses, page faults, affect performance, but they are a start.
- Real-time languages need a way to deal with time itself.
  - To start with, a special variable, clock, should be available, containing the current time in ticks.
  - One has to be careful about the unit that time is expressed in. The finer the resolution, the faster clock will overflow.

Real-Time Communication

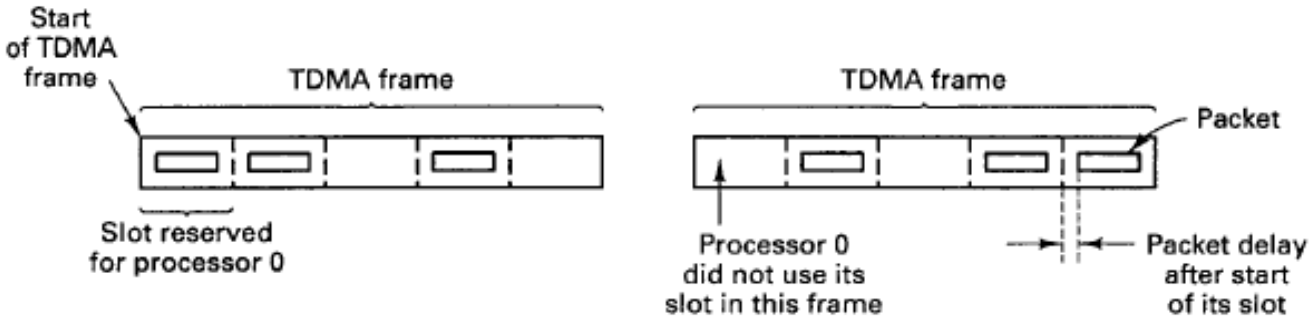
- Communication in RT DSs is different from communication in other DSs.
- While high performance is always welcome, predictability and determinism are the real keys to success.
- Achieving predictability in a DS means that communication between processors must also be predictable.
  - LAN protocols that are inherently stochastic, such as Ethernet, are unacceptable because they do not provide a known upper bound on transmission time.
- Consider a token ring LAN.
  - Whenever a processor has a packet to send, it waits for the circulating token to pass by, then it captures the token, sends its packet, and puts the token back on the ring so that the next machine downstream gets the opportunity to seize it.
  - Assuming that each of the k machines on the ring is allowed to send at most one n-byte packet per token capture, it can be guaranteed that an urgent packet arriving anywhere in the system can always be transmitted within kn byte times.
  - This is the kind of upper bound that a RT DS needs.

Token ring and priority

- Token rings can also handle traffic consisting of multiple priority classes.
- The goal here is to ensure that if a high-priority packet is waiting for transmission, it will be sent before any low-priority packets that its neighbors may have.
- For example, it is possible to add a reservation field to each packet, which can be increased by any processor as the packet goes by.
- When the packet has gone all the way around, the reservation field indicates the priority class of the next packet.
- When the current sender is finished transmitting, it regenerates a token bearing this priority class.
- Only processors with a pending packet of this class may capture it, and then only to send one packet.
- Of course, this scheme means that the upper bound of kn byte times now applies only to packets of the highest priority class.

TDMA (Time Division Multiple Access) protocol

- An alternative to a token ring
- Traffic is organized in fixed-size frames, each of which contains n slots
- Each slot is assigned to one processor, which may use it to transmit a packet when its time comes.
- In this way collisions are avoided, the delay is bounded, and each processor gets a guaranteed fraction of the bandwidth, depending on how many slots per frame it has been assigned.



Wide-area real-time distributed systems

- A potential problem is their relatively high packet loss rates.
- Standard protocols deal with packet loss by setting a timer when each packet is transmitted.
  - If the timer goes off before the acknowledgement is received, the packet is sent again.
- In RT systs, this kind of unbounded transmission delay is rarely acceptable.
- One easy solution is for the sender always to transmit each packet two (or more) times, preferably over independent connections if that option is available.
  - Although this scheme wastes at least half the bandwidth, if one packet in, say, is lost, only one time in will both copies be lost.
  - If a packet takes a millisecond, this works out to one lost packet every 4 months.
  - With 3 transmissions, one packet is lost every 30,000 years.
  - The net effect of multiple transmissions of every packet right from the start is a low and bounded delay virtually all the time.

Time-Triggered Protocol (TTP)

- A node consists of at least one CPU, but often two or three work together to present the image of a single fault-tolerant, fail-silent node to the outside world.
- The nodes are connected by 2 reliable and independent TDMA broadcast networks.
  - All packets are sent on both networks in parallel.
  - The expected loss rate is one packet every 30 million years.
- Clock synchronization is critical.
  - Time is discrete, with clock ticks generally occurring every microsecond.
  - TTP assumes that all the clocks are synchronized with a precision on the order of tens of microseconds.
  - This precision is possible because the protocol itself provides continuous clock synchronization and has been designed to allow it to be done in hardware to extremely high precision.

TTP

- All nodes are aware of the programs being run on all the other nodes.
  - In particular, all nodes know when a packet is to be sent by another node and can detect its presence or absence easily.
  - Since packets are assumed not to be lost, the absence of a packet at a moment when one is expected means that the sending node has crashed.
- For example, suppose that some exceptional event is detected and a packet is broadcast to tell everyone else about it.
  - Node 6 is expected to make some computation and then broadcast a reply after 2 msec in slot 15 of the TDMA frame.
  - If the message is not forthcoming in the expected slot, the other nodes assume that node 6 has gone down, and take whatever steps are necessary to recover from its failure.
  - This tight bound and instant consensus eliminate the need for time-consuming agreement protocols and allow the system to be both fault tolerant and operate in real time.

TTP – Global state

- Every node maintains the global state of the system.
  - These states are required to be identical everywhere.
  - It is a serious (and detectable) error if someone is out of step with the rest.
- The global state consists of three components:

1. The current mode.
   - The mode is defined by the application and has to do with which phase the system is in.
     - For example, in a space application, the countdown, launch, flight, and landing might all be separate modes.
     - Each mode has its own set of processes and the order in which they run, list of participating nodes, TDMA slot assignments, message names and formats, and legal successor modes.
2. The global time.
   - Its granularity is application defined, but in any event must be coarse enough that all nodes agree on it.
3. A bit map giving the current system membership.
   - keeps track of which nodes are up and which are down.

TTP - Messages

- Unlike the OSI and Internet protocol suites, the TTP protocol consists of a single layer that handles end-to-end data transport, clock synchronization, and membership management.
- The control field contains
  - a bit used to initialize the system,
  - a bit for changing the current mode, and
  - a bit for acknowledging the packets sent by the preceding node (according to the current membership list).
- The purpose of this field is to let the previous node know that it is functioning correctly and its packets are getting onto the network as they should be.
  - If an expected acknowledgement is lacking, all nodes mark the expected sender as down and expunge it from the membership bit maps in their current state.
  - The rejected node is expected to go along with being excommunicated without protest.
- The data field contains whatever data are required.
- The CRC field provides a checksum over not only the packet contents, but over the sender's global state as well.
  - This means that if a sender has an incorrect global state, the CRC of any packets it sends will not agree with the values the receivers compute using their states.
  - The next sender will not acknowledge the packet, and all nodes, including the one with the bad state, mark it as down in their membership bit maps.

TTP - Checks

- Periodically, a packet with the initialization bit is broadcast.
  - This packet also contains the current global state.
  - Any node that is marked as not being a member, but which is supposed to be a member in this mode, can now join as a passive member.
  - If a node is supposed to be a member, it has a TDMA slot assigned, so there is no problem of when to respond (in its own TDMA slot).
  - Once its packet has been acknowledged, all the other nodes mark it as being active (operational) again.

TTP - Clock synchronization

- Because each node knows the time when TDMA frames start and the position of its slot within the frame, it knows exactly when to begin its packet.
- This scheme avoids collisions.
- However, it also contains valuable timing information.
- If a packet begins n microseconds before or after it is supposed to, each other node can detect this tardiness and use it as an estimate of the skew between its clock and the sender's clock.
- By monitoring the starting position of every packet, a node might learn, for example, that every other node appears to be starting its transmissions 10 microseconds too late.
  - In this case it can reasonably conclude that its own clock is actually 10 microseconds fast and make the necessary correction.
- By keeping a running average of the earliness or lateness of all other packets, each node can adjust its clock continuously to keep it in sync with the others without running any special clock management protocol.