CptS 464/564
Lecture 21

1st April 2003

Project 2 demo

If you have not contacted Vaishali about a project 2 demo, you *must* do so today. We would like to be able to give them back on Thursday.

Project 3 notes

don't use –no-type-codes when IDL compiling Widget.idl

If you get hundreds of c++ compiler error messages naming various < <= or > >= operators and types, the problem is almost certainly that either Widget.idl was not compiled with –no-type-codes or the type of the variable into which you are extracting is wrong. Be careful of const – the type of the variable must match the type of the operator which you can see in the generated Widget.h.

Finishing up chapter 4

Chapter 11 - Introduction to real-time systems

Why study real-time systems in a distributed systems class? One important class of distributed systems is those that interact with the physical world. Useful interactions with the physical world often require actions to be taken on-time. Real-time systems are designed to meet these requirements for timeliness. Examples: manufacturing process control, machinery operation (consider the laser or inkjet printer on your desk or down the hall), modern automobile engines, traffic signals, air traffic control, ...

Recall that we previously discussed problems of clock synchronization in distributed systems. Timeliness suggests something more than clock synchronization: we are now concerned with how long it takes to perform an action, how long it takes for messages to be delivered, how long it takes to respond to an event in the environment.

In order to succeed in meeting timeliness requirements in the physical world we will have to have

- effective models for the time taken for interactions between computers and the physical world
- ability to make accurate time measurements (of both intervals and absolute times)
- design techniques that allow us to manage the load placed on a system
- an understanding of what constitutes timeliness in a particular system, in particular recognizing that some tasks may be more *urgent* than others
- understanding of what *faults* must be tolerated while maintaining timeliness

Definition: Real-time system: a system whose progress is specified in terms of timeliness requirements dictated by its environment. Alternatives: correctness of its computation requires both correct logical results and timely delivery of those results; delivers at least one real-time service (not a good definition – circular).

Terminology: "Real-time" (note the hyphen (not "slash" which is what the book says) refers to systems and techniques concerning timeliness of action. "Real time" refers an abstract, universal, time reference.

Recall: in asynchronous systems, we can say *nothing* about time. Since in real-time systems we are explicitly concerned with timeliness, it must be the case that real-time systems are synchronous systems. Thus, we are concerned now with techniques for specifying and building synchronous systems.

Classes of RT

You should be familiar with a number of terms concerning RT systems that are in common use. There are not hard boundaries between these classes. Think of each RT system as occupying a point in a two-dimensional space. On one axis we have

- Hard real-time systems: designed to avoid timing failures provided the environment behaves as expected.
- Soft real-time systems: occasional timing failures (how many, how often) are contemplated by the design, even when the environment behaves as expected

and on the other

- critical real-time systems: failure to meet specification results in a large loss of money, injury, death, etc. To avoid disaster, the specification and implementation of critical systems might be expanded to incorporate notions of how exceptional conditions are to be handled and mechanisms to handle them.

- non-critical systems: failure to meet specification carries low cost (e.g. computer game)

Draw diagram of the space. Place several systems on the diagram. Note that this treatment of criticality is somewhat different from the treatment in the book.

What RT is not

- RT is not merely about going fast; RT is about predictability of timeliness; meeting deadlines. A non-RT system *cannot* be made RT by increasing the speed of the processors and networks!
- RT is not created by low-level hacking – assembly language programming, tweak it 'til it works; there is an established body of principles and architectures that guide modern real-time design and implementation.
- RT does not require hardwired circuitry; modern RT systems can involve general-purpose processors and may be distributed; they can be reprogrammed to meet new requirements.
- Achieving RT in the face of faults is not hopeless. An RT system can be designed to meet its deadlines even though hosts may crash and messages may be lost.

Achieving RT in distributed systems

- Model or framework that enforces timeliness (synchronous models such as event- and time-triggered)
- Scheduling on each node must support RT
- Communications infrastructure must support RT
- I/O devices must support RT
- Fault tolerance techniques required to achieve RT in the presence of faults
  - Example: suppose a message must be delivered over a LAN within a certain time bound in order for the system to meet its specification
  - LAN always either delivers the message within the bound or drops the message
  - This system cannot meet its specification: a dropped message is not delivered within the required time.
  - Solving this problem requires a *fault model* to describe the assumptions made (or to be made) in the design of the system concerning the kinds and number of faults to be tolerated.

Real-time communications networks

Primitive RT network: a piece of wire, strung point to point. Delay = speed of light in the wire. The old, analog, dial-up telephone network is also a primitive example of a RT network, not much different from the wire.

Now think about computers hooked to that wire with two RT applications running. Somehow, the two applications' use of the wire must be managed so that both can meet their delay requirements. This is a simple example of the kinds of resource management needed in RT systems. More sophisticated networks bring more challenging problems.

For example, consider a bus architecture: several hosts connected to one piece of wire. Now, not only must the applications on each host be managed, the hosts' access to the shared medium must also be managed. Examples:

- master/slave: a master node periodically polls each slave for its data
- TDMA (time division multiple access): each node has a slot in which it is allowed to transmit
- on-demand virtual circuit: the network allows nodes to request a "virtual wire" with certain bandwidth and delay properties connecting it to some other node.

Distributed Real-time architectures

Distributed real-time control

- sensors
- actuators
- controlling system: computers and software
- controlled system: physical system monitored by the sensors controlled via the actuators
- our earlier discussion about temporal order is quite relevant here as there are feedback paths through actuators, controlled system, and sensors. Designing control systems requires
  - solid understanding of the controlled system
  - knowledge of real-time system design
  - understanding of control theory (EE 489)

Real-time producer-consumer

In this architecture we usually think of the producer as producing a stream of data items that have to be delivered to the consumer with timeliness constraints. For example, playing a DVD to your eyes or CD to your ears. In addition to our earlier measures of *throughput (i.e. bandwidth)* and *latency,* we typically will also find a requirement to bound the *jitter* – how much the time between one data item and the next will vary.

Real-time client server

Mixed hard and soft real-time? Consider phone number translation example mentioned in the book. I have an 800 number from a third-party phone company, but my actual phone is from Verizon. When someone calls the 800 number it has to be translated to the real phone number. The phone company of the caller will want the 800 number translated quickly so its resources aren't tied up. But it will have a measure of goodness that says the longer the delay the less happy it is. That is an example of soft real-time. But beyond a certain delay it will not be willing to wait and will disconnect the call. That is an example of hard real-time.