Administrative Stuff

- We are now in week 11
- No class on Thursday
- About one month to go
  - Spend your time wisely
  - Make any major decisions w/ Client
# Real-Time and On-Line

<table>
<thead>
<tr>
<th>ON-Line</th>
<th>Real-Time</th>
<th>NOT Real-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flight avionics</td>
<td>Flight avionics</td>
<td>Our PCs</td>
</tr>
<tr>
<td>Analyze seismic data</td>
<td>Analyze seismic data</td>
<td>Rendering “Toy Story”</td>
</tr>
</tbody>
</table>
Distributed Systems
Basic System Types

- **Personal systems**
  - One, two, or four processors, very tightly coupled in one machine
  - Usually on-line (interactive) but not real-time

- **Embedded systems**
  - Moderately tightly coupled processors, specific to an appliance, device, or machine
  - Usually on-line, often real-time

- **Distributed systems**
  - Loosely integrated group of cooperating processors linked by a network
  - Usually on-line (interactive) but not real-time
  - Sometimes off-line and not real-time
Distributed Systems

- Virtually all large computer-based systems are now distributed systems
- Information processing is distributed over several computers rather than confined to a single machine
- Distributed software engineering is now very important
DS Characteristics

- Resource sharing
- Openness
- Concurrency
- Scalability
- Fault tolerance
- Transparency

(components)
Disadvantages

- Complexity
- Security
- Manageability
- Unpredictability
Issues

- Resource identification
- Communication
  - Quality of service
- Software architecture
Main DS Architectures

- **Client-server architectures**
  - Servers that provide services are treated differently from clients that use services.

- **Distributed object architectures**
  - No distinction between clients and servers. Any object on the system may provide and use services from other objects.
Middleware

- Manages and supports the different components of a distributed system
- Usually COTS (commercial off-the-shelf)
- Examples
  - Transaction processing
  - Data conversion
  - Distributed communication control
Multiprocessor Architectures

- Multiple related processes running on coupled processors, embedded or distributed
- Control (scheduling)
  - A priori vs. real-time
  - Both
- Common w/ real-time systems
Client-Server Architectures

- Servers offer a set of services, clients request/use these services
- Clients know of servers but servers need not know of clients
- Clients/servers are logical processes
- The mapping of processors to processes is not necessarily 1:1
Thin vs. Thick Client

- **Thin-client model**
  - All of the application processing and data management is carried out on the server.
  - Client is simply responsible for running the presentation software.
  - Heavy burden on server

- **Thick-client model**
  - Server is only responsible for data management.
  - Client maintenance is more complex.
CORBA

- International standard for an Object Request Broker—middleware to manage communications between distributed objects
- Several implementations available
- DCOM is an alternative approach by Microsoft to object request brokers
- CORBA has been defined by the Object Management Group
CORBA Standards

- Object model for application objects
  - A CORBA object is an encapsulation of state with a well-defined, language-neutral interface defined in an *interface definition language (IDL)*
- An *object request broker (ORB)* that manages requests for object services
- A set of *general object services* of use to many distributed applications
- A set of *common components* built on top of these services
CORBA Objects

- Comparable (in principle) to objects in C++ and Java
- Each object MUST have a separate interface definition that is expressed using a common language (IDL) similar to C++
- There is a mapping from this IDL to programming languages (C++, Java, etc.)
- As a result, objects written in different languages can communicate with each other
CORBA Object Request Broker

- Handles object communications
  - Knows all objects and interfaces in the system
- Calling object
  - Using an ORB, calling object binds an IDL “stub” interface of the called object
- Receiving object
  - ORB calls the required object through a published IDL “skeleton” interface to the service implementation
Inter-ORB Communication

- **Link-time objects**
  - ORBs are not usually separate programs but are a set of objects in a static library that are linked with an application when it is developed

- **One/same machine**
  - ORBs handle communications between objects executing on the same machine

- **Different/distributed machines**
  - Several ORBS may be available and each computer in a distributed system will have its own ORB
  - Inter-ORB communication services are used for distributed object calls
CORBA Services

- **Naming and trading**
  - These allow objects to discover and refer to other objects on the network

- **Notification**
  - These allow objects to notify other objects that an event has occurred

- **Transaction**
  - These support atomic transactions and rollback on failure
Real-Time Systems
Real-Time Systems

- In general
  - Systems that interact with (monitor and control) their environment
- Inevitably associated with hardware devices
  - Sensors: collect data from the system environment
  - Actuators: change (in some way) the system's environment
- Time is critical
  - MUST respond within specified times
Definitions

- A real-time system is a software system where the **correct functioning** of the system depends on the **results** produced by the system and the **time at which these results are produced**
- “Soft” real-time system
  - A system whose operation is **degraded** if results are not produced according to the specified timing requirements
- “Hard” real-time system
  - A system whose operation is **incorrect** if results are not produced according to the timing specification
Stimulus/Response

- Given a stimulus, the system must produce a response within a specified time

- Periodic stimuli
  - Regular/predictable time intervals
  - For example, a temperature sensor may be polled 10 times per second

- Aperiodic stimuli
  - Stimuli which occur at unpredictable times
  - For example, a system power failure may trigger an interrupt which must be processed by the system
Architectural Considerations

- System architecture must allow for fast switching between stimulus handlers.
- Timing demands of different stimuli are different so a simple sequential loop is not usually adequate.
- Real-time systems are usually designed as cooperating processes with a real-time executive controlling the processes.
Design Considerations

- Identify stimuli to be processed and the required responses to these stimuli
- For each stimulus and response, identify the timing constraints
- Aggregate the stimulus and response processing into concurrent processes
  - A process may be associated with each class of stimulus and response
Design Considerations
(Continued)

- Algorithms to process each class of stimulus and response
  - These must meet the given timing requirements
- Design a scheduling system
  - Ensure that processes are started in time to meet their deadlines
- Integrate
  - Use a real-time executive or operating system (RTOS)
Timing Constraints

- Extensive simulation and experiment
  - Ensure that these are met by the system
- OOD might be OUT
  - Certain design strategies such as object-oriented design cannot be used because of the additional overhead involved
- ASM might be IN
  - May mean that low-level programming language features have to be used for performance reasons
Real-Time Executives

- Specialized operating systems that manage the processes in the RTS
- Responsible for process management and resource (processor and memory) allocation
- May be based on a standard RTE kernel which is used unchanged or modified for a particular application
- Usually does not include facilities such as file management
Executive Components

- **Real-time clock**
  - Provides information for process scheduling
- **Interrupt handler**
  - Manages requests for service
- **Scheduler**
  - Chooses the next process to be run
- **Resource manager**
  - Allocates memory and processor resources
- **Despatcher**
  - Starts process execution
- **Communications**
  - Send/receive process/processor messages
The processing of some types of stimuli must sometimes take priority.

- **Interrupt level priority**
  - Highest priority allocated to processes requiring a very fast response
  - Handle quickly

- **Clock level priority**
  - Allocated to periodic processes

- Within these, further levels of priority may be assigned
Interrupts

- Control is transferred automatically to a pre-determined memory location
- This location contains an instruction to jump to an interrupt service routine
- Further interrupts are disabled, the interrupt serviced and control returned to the interrupted process
- Interrupt service routines MUST be short, simple and fast
Periodic Processing

- **Classes**
  - periods, execution times, and deadlines
- **Timing interrupts**
  - Each real-time clock tick causes an interrupt that effects the scheduling of periodic processes
- The **process manager** selects a process that is ready for execution
Process Scheduling

- Non pre-emptive scheduling
  - Once a process has been scheduled for execution, it runs to completion or until it is blocked for some reason (e.g. waiting for I/O)

- Pre-emptive scheduling
  - The execution of an executing processes may be stopped if a higher priority process requires service

- Scheduling algorithms
  - Round-robin
  - Rate monotonic
  - Shortest deadline first
Monitor and Control (Class)

- Important class of real-time systems
- Continuously check sensors and take actions depending on sensor values
- Monitoring systems examine sensors and report their results
- Control systems take sensor values and control hardware actuators
- Estimation and control theory
Mutual Exclusion

- **Producer-consumer**
  - Producer processes collect data and add it to a buffer. Consumer processes take data from the buffer and act on them.

- **Producer and consumer processes must be mutually excluded** from accessing the same element.

- **Boundary conditions**
  - A buffer must stop producer processes adding information to a full buffer and consumer processes trying to take information from an empty buffer.
Software Engineering for Distributed and Real-Time Sys

- Notion of distribution impacts design
  - Would generally be part of requirements
- Periods for COTS evaluation
  - Spec, find, assess, choose
- Schedules might allow for simulations
  - Timing
  - Transaction order and robustness
- Stochastic testing of each