CptS 464/564
Lecture 22

4th April 2003

Finish up real-time architectures from Lecture 21

Overview of Chapter 12 topics

- Specifying real-time systems - considerations and vocabulary
- Entitities and Representatives: Mapping the physical world onto a computational system
- Time-value duality: what is being measured and when
- Real-time communications
- Resource scheduling for real-time
- Real-time I/O

Temporal specification

There are a great many things one might want to specify about the temporal behavior of a system. Here we explore some aspects of temporal specification. We consider both what one wants to say about a system and the practical limitations on one's ability to say it.

Many (most?) real-time systems are *reactive* (at least at some level – recall the RT architectures). That is their primary purpose is to react to events in their environment (inputs) by producing some outputs.

One common specification and measurement of such systems is their *response time:* the amount of time between the occurence of an input event and the first related output event. We may be interested in both the maximum *and minimum* response times. Going along with the notion then, we have the notion of a *timed action:* an action that is

specified to occur within an interval T_A following some instant t_A. (Observe that t_A need not represent the beginning of execution of the action.)

Warning: semantic curves ahead. When we use the word "time" sometimes it means an absolute time ("What time does this class start? 4:15") and sometimes it means a duration or amount of time ("How much time is spent in lecture?" 1:15). The textbook's authors are not careful in their writing about this distinction.

Components of response time.

deferral time delay between an input event and beginning to process it

termination time total amount of time (duration) between the occurence of an event, also called the start time (absolute) and completion of its processing, also called the end time (absolute)

execution time the amount of time (duration) the event is actively being processed between its start and end. The execution time is less than or equal to the termination time, excluding the deferral time and any other *dead time*.

Suppose we have specified an action to react to a particular kind of input event, and that this action is expected to take amount of time T_A. If the input event occurs at t_A, when will the output event occur? $T_A + t_A$. Yes, but...in a real systems the action won't take exactly T_A, nor will we be able to measure exactly the time when it completes. To accomodate these uncertainties we introduce the notion of *jitter,* capturing both the *variance* in the execution time and the *imprecision* of measuring the times of the starting and terminating events.

Good RT system specifications *must* incorporate provision for jitter: reality can't be modelled with exact numbers.

Meeting a timed-action specification

There are essentially two approaches to meeting a timed action specification: event-triggered and time-triggered. In the event-triggered approach, the system begins processing when the event occurs. (Think interrupts.) In the time-triggered approach, the system begins processing at the next tick of the clock. The distinction may seem clear-cut, but as we look at different time scales we might see an event that was time-triggered at a fine granularity looking event-triggered at a coarser granularity and vice versa. For example, consider an interrupt for a keypress. At the level of the electrical pulses, there might be a clock that latches the signal for the keypress and allows the processor to be interrupted (time triggered). The processor immediately stops what it is doing and works to process the keypress (event-triggered), but that involves making a process ready. Actually switching to the process doesn't occur until the currently running process finishes its time-slice (time-triggered), but even so, at the time scale of

the person typing at the keyboard, the character appears on the screen "immediately" (event-triggered).

In addition to treating different events differently, as described for the hybrid case in the text, the same kind of events can be treated differently at different times, as the system adapts to the load. A fairly common design technique is to have a system operating under light load operate event-triggered which provides the best response latency in this case. But as the load increases it may switch to operating time-triggered, processing multiple events at each trigger point. Why? Processing multiple events per trigger reduces the overhead such as context switches and cache pollution associated with event processing.

Specifying load - Arrival Distributions

As we have previously noted in talking about synchronous systems, real systems cannot process an unlimited amount of input in a limited time. A specification for a real-time system *must* therefore include a temporal description of how work arrives at the system. In general, the amount of arriving work cannot be bounded unless the environment of the system can be constrained, so the specification may well be of the form "as long as the environment behaves in *such and such* a manner, the system will meet its timeliness specifcations." We say that an environmental specification ike this *covers* a portion of all the possible environmental behaviors. Specifications that cover more behaviors make it harder to design a system to meet the specification.

Classes of environmental specifications for input arrivals:

Periodic arrivals arrivals occur at regular intervals. Our design will need to accomodate jitter but the amount of work to do be done is predictable. Design for periodic arrivals is relatively easy.

Sporadic arrivals arrivals occur irregularly, but the arrival pattern can be characterized by parameters that bound the amount of work to be done in any interval. An example of such a pattern would be one that is characterized as bursty, where we know the maximum amount of data in each burst, N_B and bursts start with at least a minimum separation, T_B called the burst period. Specification and design for sporadic arrivals is harder. For the environment in which the system is to operate, you have to figure out how to characterize the arrival distribution in a useful way, and then you have to implement the system to meet this more complex specification.

Aperiodic arrivals inputs may arrive at any time. Even if we know the average arrival rate, it is impossible to guarantee that deadlines will be met if arrivals are aperiodic.

Timing Failure Detection - skipping

A few words on entities and representatives (12.3)

This section presents one model for interpreting the interactions between a real-time computational system and the physical world. In this model, physical world (environment) is characterized by states of physical phenomena and actions of physical objects, for example the temperature of a liquid, the degree to which a valve is open or closed, etc. We call these real-time *entities.*

These real-time entities are interfaced to the computational system through (multiple layers of) *representatives.* The first layer of representatives consists of the sensors and actuators that measure and affect the physical system, and these in turn are interfaced to an executing program through I/O boards, busses, processor interrupt systems, etc.

The advantage of this *entity-representative modelling* is that it encourages us to address phenomena that could otherwise easily be overlooked. For example, the temperature measured by a sensor may suffer from a time-lag (due to the thermal mass of the sensor) or from lack of precision. Treating the sensor as *only a representative* of the temperature encourages us carry the information about these effects through the design to the computational steps that will incorporate the temperature.