SYSC 3303 Real-Time Concurrent Systems

The Nature of Real-Time and Embedded Systems

- revised January 6th, 2016

What are Real-Time Systems?

- "A real-time system must respond to unpredictable stimuli from its environment in a timely fashion while operating reliably and continuously in the presence of failures in its own (perhaps distributed) components and connections, and uncertainty about the state of its environment."
  - *An Introduction to Real-Time Systems: From Design to Multitasking with C/C++*,
    R.J.A. Buhr & D.L. Bailey,

What are Real-Time Systems?

- “Any information processing activity or system which has to respond to externally generated input stimuli within a finite and specified delay.”
- “The correctness of a real-time system depends not only on the logical result of the computation, but also on the time at which the results are produced.”
  - *Real-Time Systems and Programming Languages, 2nd Ed.* Alan Burns & Andy Wellings, copyright © 1997 Addison Wesley Longman Ltd.

What are Real-Time Systems?

- “Real-time problems are those for which timeliness, [defined as] ‘Occurring at a suitable or opportune time’, is a correctness criterion. If the process finishes late, it is wrong or at least noticeably less satisfactory than a process that completes on time.”
  - *Real-Time Java Platform Programming*
    Peter C. Dibble
    Sun Microsystems Press (A Prentice Hall Title)

What are Real-Time Systems?

- A computer-based system that must resolve a number of difficult issues simultaneously:
  - rapid response
  - continuous operation
  - bursty stimuli
  - failures of its components and connections
  - uncertainty about communications and processing delays
  - uncertainty about the state of the environment
  - geographical distribution
  - need to adapt over time while continuously operating

“Real-Time” Versus “Fast”

- All programs must be fast enough to meet their user’s requirements, so “real-time” means more than just “fast”
- Real-time means dealing directly with: physical distribution, unpredictable stimuli, failures in components and connections, uncertainty about the state of the environment, and their effects on the system’s performance and robustness

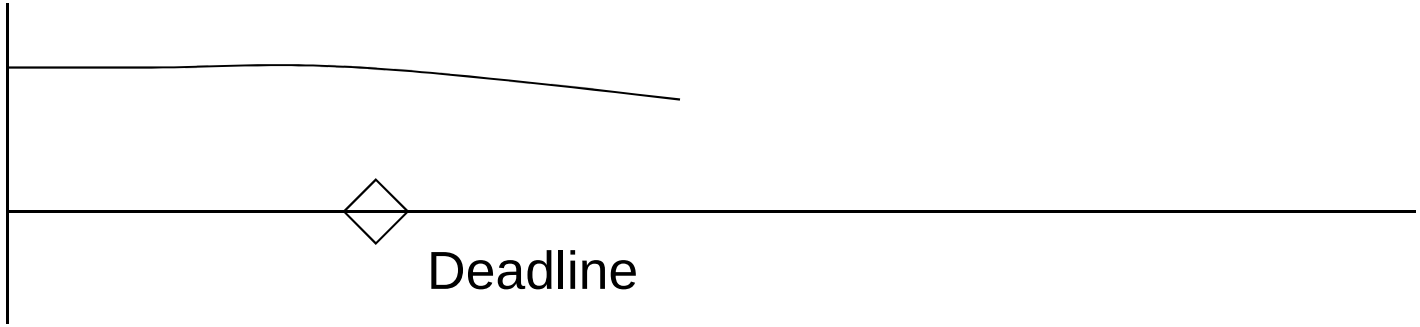
Hard vs. Soft Real-Time

- *Hard* real-time system - responses must occur within specified deadlines - failure to do this will likely have catastrophic results
- *Soft* real-time system - response times are important, but system functions correctly if a deadline is occasionally missed (i.e., responses occasionally delivered late or not at all)

Utility Function Curves (1)

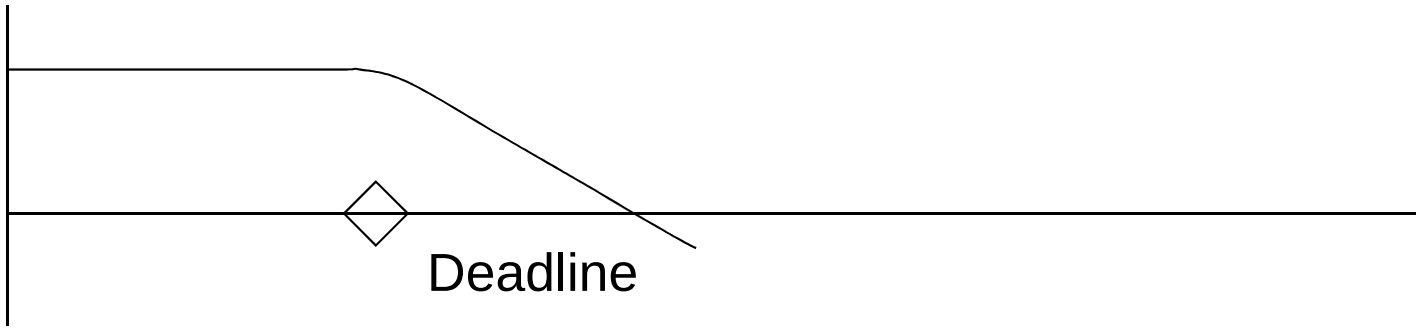
- As used by economists, the term *utility* refers to how valuable something is
- We can characterize different types of computing systems by plotting graphs that show the utility of completing some computation around a deadline
  - source: *Real-Time Java Platform Programming*
    Peter C. Dibble

Utility Function Curves (2)



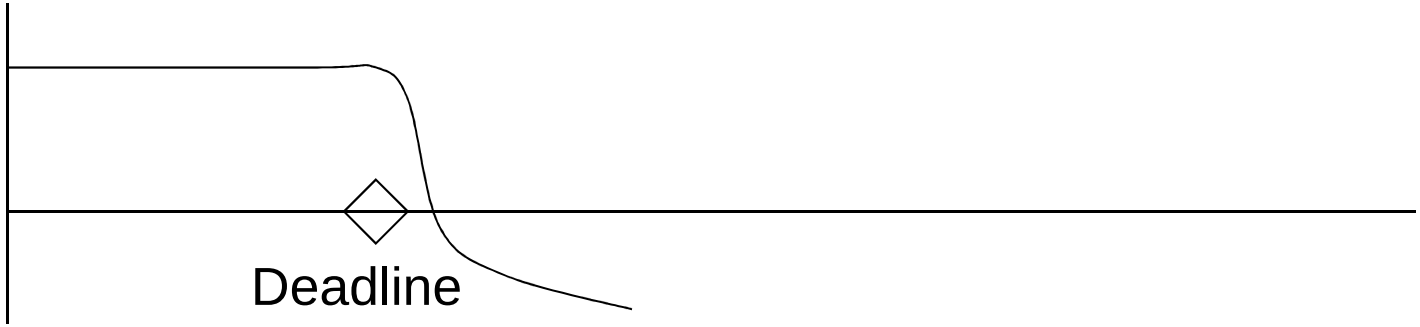
- Non-real-time
  - utility of completion declines slowly after deadline
  - little difference between completing by deadline and being very late

Utility Function Curves (3)



- Soft real-time
  - utility of completion declines after the deadline, but remains positive for a while after the deadline

Utility Function Curves (4)



- Recoverable real-time fault
  - utility of completion quickly becomes negative, but not catastrophic
  - even after the utility is negative, the system can take remedial action (slope of curve decreases) - nothing terrible happens

Utility Function Curves (5)

- Hard real-time
  - shortly after the deadline, utility of completion goes to negative infinity
  - no point in trying to recover from the missed deadline - system must be engineered to ensure that deadlines are not missed

Embedded Systems

- Many real-time systems are *embedded systems*
  - computer-based system designed to perform a specific dedicated application
  - computer hardware and software are usually not accessible/upgradeable by the system's users
  - software usually permanently stored in read-only memory (ROM)
    - some embedded systems download software from a network (so-called Internet appliances)
  - system may not have a keyboard, disk, graphics display
    - HCI devices tailored to the application

Embedded System Examples (1)

- Consumer products: kitchen appliances, audio/video (CD player, television, etc.), remote controls, watches, games & toys
- Communications products: telephones, cell phones, pagers
- Automotive systems: ABS braking, theft deterrent, electronic ignition, cruise control, instrumentation

Embedded System Examples (2)

- Military: “smart” weapons, missile guidance systems
- Industrial applications: robotics, bar code readers, setback thermostats
- Medical applications: apnea monitors, cardiac monitors, drug delivery, pacemakers, prosthetic devices

Real-Time vs. Embedded

- What about a real-time application running under a real-time operating system on a desktop PC equipped with various I/O boards required by the application?
- Would you consider this to be an embedded system?
- Don’t get too hung up on the terminology
- In this course, we’ll sometimes use the terms *real-time* and *embedded* interchangeably

Real-Time Systems are Characterized by Diversity

- Applications: e.g., board-level communications products, consumer appliances, telephone switches, air traffic control systems, on-line banking systems, etc.
- Components: e.g., physical components (I/O components, I/O interface chips, etc.), software components (objects, concurrent processes, agents, etc.)

Real-Time Systems are Characterized by Diversity

- Programming languages: e.g., assembly languages, C, C++, Ada, occam2, Smalltalk (!), Java, etc.
- Operating systems: e.g., MQX, QNX, pSOS, Unix?, Linux?, NT?, or no O/S (just a language-specific runtime system)

Java for Real-Time Embedded Systems?

- In the hands-on part of this course, we'll focus on the design and implementation of systems that are organized as a set of concurrent collaborating threads
  - the programming technology we'll use is Java
  - this may seem an unusual choice for a course in real-time systems, but as we'll see, there are good reasons for using Java

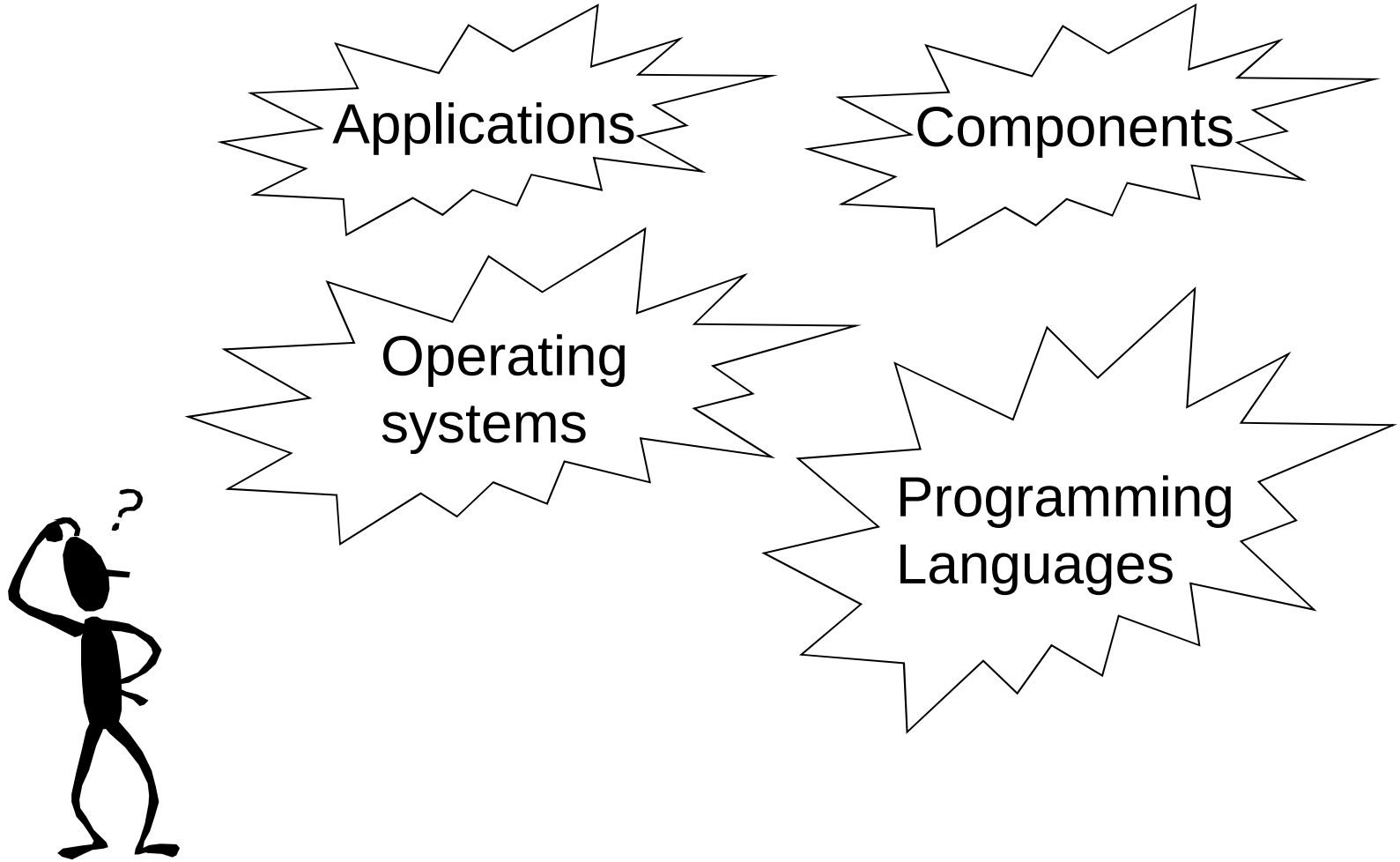
Java for Real-Time Embedded Systems?

- Support for concurrent threads is part of the language specification (`synchronized` methods and blocks), the class library (primarily `Object` and `Thread`), and the JVM
- Industrial application of Java in embedded systems is still limited; however, an embedded system version of Java has been released by Sun; and classes to better support real-time Java programming have been released

Real-Time Systems are Characterized by Diversity

- Although we'll be using Java, it is important to keep in mind that in the real-time world, there is no one dominant platform that is the "best" one to learn to make your resume look good
- It's important to understand the principles independently of any particular implementation technology

How Do We See the R-T Forest for the Trees?



How Do We See the R-T Forest for the Trees?

- Some researchers and practitioners believe that visual notations can help us view real-time systems as a collection of concurrent collaborating components, at a level of abstraction above the coding details
- Different shapes represent different types of components and different types of interactions between components
- Different notations have been proposed for r-t design (e.g., Booch diagrams; Machine Charts, Collaboration Graphs and Use Case Maps (Buhr); ROOM modelling language (ObjecTime))

R-T Design and the UML

- The UML is becoming increasingly important in the real-time field (although it's more widely used for modelling non-real-time OO systems)
  - plusses: UML is a defacto standard, widely adopted by industry, and increasingly supported by software CAD tools
  - minuses: UML is big (too big?), yet appears to have some significant limitations wrt. modelling concurrent systems (we'll look at these in this course)

Concurrency: Active System Components

- If a system is comprised of collaborating components, then some of those components must be *active*; i.e, have autonomy and operate concurrently
- In some systems, all concurrency is at the hardware level; e.g., I/O devices operate concurrently with the CPU
- Concurrency can be introduced into the software in a limited way through interrupt service routines (ISRs) that asynchronously handle low-level I/O

Concurrency: Active System Components

- In many systems, active system components are typified by concurrent processes that are supported by different programming technologies under a variety of names (e.g., process, thread, task, actor)
  - processes do not guarantee that a program will be fast
  - processes allow performance to be tuned (process priorities, allocation of processes to processors, etc.)

Concurrent System Design

- How do we discover processes and process architectures from application requirements?
- It is often difficult for novices to discover processes, build failure/recovery mechanisms into process architectures, decide on interprocess communication (IPC) mechanisms, and integrate components other than processes into the architectures
- Traditional structured and OO design methodologies don’t normally deal with this
- We’ll introduce you to a lightweight technique for developing process architectures that is supported by visual notations (UCMs)

Emerging Design Methodologies

- There are a number of real-time design methodologies based on the UML (e.g., see the books by Gomaa and Douglass that are listed in the course outline)
  - we won't cover either of these methodologies in this course, but once you understand the design techniques presented in this course, you should have no problem learning the more heavyweight design methodologies, including the ones presented in these books

Analytical Techniques

- There are several analytical techniques for proving that a hard real-time system can meet its deadlines
- We'll look at some of the simpler techniques in this course
- See *Real-Time Systems*, Jane W.S. Liu, Prentice-Hall, 2000, for in-depth coverage of algorithms for scheduling and validating hard real-time systems