COMP 4550
Real-Time Systems

Autonomous Agents Lab, University of Manitoba
jacky@cs.umanitoba.ca
http://www.cs.umanitoba.ca/~jacky
http://aalab.cs.umanitoba.ca
Introduction

- Definition of Real-Time Systems
- Real-Time Operating Systems
  - Children Robotics Programming
Real-Time: Pertaining to the performance of a computation during the actual time that the related physical process transpires in order that the results of the computation can be used in guiding the physical process. [IEEE Standard Dictionary]
Real-Time Systems

- Real-time systems are defined as those systems in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced [Stankovic'88]
- Performance of a computation during the actual time that the related physical process transpires. Results of the computation can be used in guiding the physical process.
- Any system in which the time at which output is produced is significant. This is usually because the input corresponds to some movement in the physical world, and the output has to relate to the same movement. The lag from input time must be small enough to acceptable timeliness
Real-Time Systems

- Input corresponds to some movement in the physical world, and the output has to relate to this movement. The latency from input to output must be small enough or accounted for.
- The environment changes in real time and imposes timing constraints on the system. Current system output depends upon “timed behavior” of input.
Real-Time Systems

- Hard Real-Time Systems
  - Avionic control
  - Nuclear power plant
- Firm Real-Time Systems
  - Banking
- Soft Real-Time Systems
  - Video on demand
  - Youtube
Real-Time Definitions

- **Deadline**
  - Time when execution must be completed
  - Specific point in time
    - Earth closest to Mars for the next 50 years
    - Time driven
  - Time interval (delta time)
    - Clear the buffer before the next message arrives
    - Even driven
Real-Time Systems Identification

- Hard deadline
  - Penalty due to missing a deadline is a higher order of magnitude than the reward in meeting the deadline
  - Prevent mid-air collision
- Firm deadline
  - Penalty and reward are in the same order of magnitude
- Soft deadline
  - Penalty is of lesser magnitude than reward
  - Best attempt policy
Hard Real Time systems: Systems where it is absolutely imperative that responses occur within the specified deadline.

Soft Real Time systems: Systems where response time is important but the system will still function correctly if deadlines are occasionally missed.

From [www.embedded.com](http://www.embedded.com)
Real-Time System Characteristics

- Embedded system: a component of a larger hardware/software system.
  - Many people use the term embedded systems synonymous to real-time systems.
- Concurrent system
  - The system simultaneously controls and/or reacts to different aspects of the environment, many events that need to be processed in parallel.
- Safety critical system
  - Not only reliable but also safe, if it fails then without causing injury or loss of life. The development of safe system involves architectural redundancy.
- Reactive system
  - There is a continuous interaction with the environment, event-driven and must respond to external stimuli, system response is typically state dependent. Behavior of event-driven system primarily caused by reactions to external events.
Real-Time System Examples

- Microwave ovens,
- Medical monitoring systems,
- Car control systems: ABS systems and engine control,
- Air craft systems,
- Chemical factory control system,
- Networks of dozens of computers collaborating to control entire factories,
- Nuclear power plant control systems
- Mobile phones
- VoIP
- Game consoles
- Network routers
Real-Time System Definitions

- **Time Driven System**
  - primarily driven by periodic tasks rather than by the arrival of aperiodic events
- **Task or a Thread**
  - an encapsulated sequence of actions that executes independently and in parallel with other tasks. Actions are sequentially executing statements that execute at the same priority or perform cohesive functions.
- **Heavy weight thread or Process**
  - share resources (memory, IO descriptors)
- **In a multitasking system the tasks are scheduled to run via scheduling policy.**
- **Real Time Operating System (RTOS) is software that is responsible, among other activities, to implement the scheduling policy. Not all real-time systems have RTOS. A task priority is used by RTOS to control when it runs in relation to other tasks.**
 Definitions: Timeliness

- The **timeliness** of an action is how the action meets time constraints.
- The basic concept is that actions must begin in response to event or due time and must complete within a certain time after they begin.
- Usually, timeliness requirement is specified by determining the end to end performance requirement of an event response action sequence.
- Example 1 (Response to event): The light switching algorithm will start **after** a user has pressed one of the traffic light button.
- Example 2 (Due-time arrival): Every **100ms**, the system will read the electromagnetic sensor data and will update traffic status.
- Example 3 (Completion of an action): The camera will be positioned on the loaded traffic route **within 3 sec** from receiving an external command.
Definitions: Responsiveness

- Standard systems response primarily to the user, while real-time systems may interact with users, but primarily response to external sensors / actuators / devices. External events are often not predictable and the real-time system must react on time, this is called the responsiveness.

- The real world is not linear and often its elements are dependent. A small perturbation in the input can cause a major change in the output. Our conceptual and computer models often make assumptions of linearity and independence, and things go wrong when these assumptions are violated. This is the main difficulty.

- Examples: Flight control computers command engine thrust and wing and tail flap orientation to meet flight parameters. Chemical-process-control systems when, what kind, and the amounts of different reagents added to mixing vats. Pacemakers make the heart beat at appropriate intervals by means of electrical leads attached to the inside walls of the heart.
Costs

- Many embedded products are cost sensitive.
- Marketing and sales concerns push for smaller processors and less memory to lower the manufacturing costs.
- The per shipped item is called Recurring Cost (RE).
- The developmental (fixed) cost is called Non Recurring Cost (NRE).
- Reducing the hardware facilities may reduce recurring cost.
- This reduction influences and usually increases cost and effort of the development activities.
- Anecdote - Seagate engineer I met in Singapore saved one screw from design. Cost savings to Seagate meant that Seagate could have hired him for 20 years.
- Anecdote - My student Daniel Kao worked 5 months reducing memory footprint of DVD decoder from 6KB to 5KB (MediaTec Taiwan).
System on a Chip (SoC)

- Many real-time systems are designed using system-on-a-chip (SOC) strategy.
- SOC allows the CPU, memory, memory management unit, and attached peripheral ports (i.e. USB) to be contained in a single integrated circuit.
- Field Programmable Gate Arrays (FPGAs) are a large collection of logic gates that can be connected via programming:
  - Altera Stratix V 1.19 Million logic elements
  - Soft processors (i.e., built your own processor) powerful enough to run ARM or Sparc cores
  - Built your own processor
Fault Tolerance

● Not only reliable but also safe, if it fails then it causes injury or loss of life. The system should never fail!
● The development of safe system involves architectural redundancy
  ○ Planes have three control computers
  ○ Decision of the majority of the computers is executed
  ○ Run intermittent self-test patterns (similar to unit tests) and signal a shutdown if the output does not match the unit tests
● Architectural design patterns are used to capture and represent the redundancy.
Fault Tolerance

● A system is **correct** when it does the right thing all the time. A correctness of the system depends not only on the logical results of the computation, but also on the time at which the results are produced.

● A system is **robust** when it does the right thing not just in regular circumstances, but under unplanned circumstances, even in the presence of unplanned failures.

● A robust system must:
  ○ identify the fault
  ○ take an action like:
    ■ correct the failure and continue processing
    ■ repeat the previous computation to restore the correct system state
    ■ enter a fail-safe condition
Exceptional Conditions

- One way to achieve robustness is to use exception handling to improve system correctness.
- This requires active means to identify faults.
- This means that computational preconditional invariants must be explicitly verified during execution.
- If these are satisfied, normal processing continues.
- If the invariant condition tests fail, then an action is taken.
- In a robust system these invariants are checked whenever they are relied upon
Example

- A satellite system is subject to a high level of ionizing radiation that frequently modify values of bits in memory. If a correct value was written to memory it does not mean that the value remains correct.
- One can use the following class that verifies its values with redundant 1’s complement storage

```cpp
Class LaserPosition{
    long x, y, z, notX, notY, notZ;
    public:
    void set(a, b, c){
        x=a; notX=~a;
        y=b; notY=~b;
        z=c; notZ=~c;
    }
    long getX(void){
        if(x==~notX) return x;
        else throw("Corrupted X");
    }
};
```
Mars Climate Orbiter

Case study: Mars mission


Mars Climate Orbiter

- Cruise
  - 4 midcourse maneuvers
  - 10-Month Cruise

- Launch
  - Delta 7425
  - Launch 12/1/98

- Mapping/Relay
  - 12/3/99 – 3/1/00: Mars Polar Lander Support Phase
  - 3/00 – 1/02: Mapping Phase
  - PMR/R and MARCI Science
  - Relay for future landers

- Mars Orbit Insertion and Aerobraking
  - Arrival 9/23/98
  - MOI is the only use of the main [biprop] engine. The 16-minute burn deploys oxidizer and captures vehicle into 13-14 hour orbit.
  - Subsequent burn using hydrazine thrusters reduce orbit period further.
  - Aerobraking to be completed prior to MPL arrival [12/3/99].
Mars Climate Orbiter

- The Mars Climate Orbiter (MCO) was launched on December 11, 1998 and spent 9 months traveling toward Mars.
- Orbit Mars as the first interplanetary weather satellite.
- Provide communication relay for the Mars Polar Lander (MPL) which was scheduled to reach Mars three months later, in December of 1999.
- On September 23, 1999 the Mars Orbiter stopped communicating with NASA and it was presumed to have either been destroyed in the atmosphere by entering orbit too sharply or to have passed by Mars by entering orbit too shallowly.
- The root cause for this failure to approach orbit at the right angle was discovered to be an inconsistency in the unit of measure used by two modules created by two separate groups.
- Thruster performance data was computed in imperial units and fed into the module that computed small forces, but which expected data to be in metric units.
Mars Climate Orbiter

Entry/Descent/Landing Phase

- Guidance System Initialization (L = 13 min)
  - 4600 km
  - 5700 m/s
- Turn to Entry Attitude (L = 12 min)
  - 3900 km
  - 9990 m/s
- Cruise Ring Separation / Microprobe Separation (L = 10 min)
- Atmospheric Entry (L = 5 min)
  - 125 km
  - 6000 m/s
- Parachute Deployment (L = 2 min)
  - 8800 m
  - 490 m/s
- Heatshield Jettison (L = 110 s)
  - 7500 m
  - 250 m/s
- Radar Ground Acquisition (Altitude) (L = 39 s)
  - 2560 m
  - 93 m/s
- Radar Ground Acquisition (Doppler) (L = 36 s)
  - 1460 m
  - 89 m/s
- Lander Separation / Powered Descent (L = 35 s)
  - 1300 m
  - 86 m/s
- Touchdown 2.3 m/s
- Solar Panel / Instrument Deployments (L + 20 min)
Mars Polar Lander

Cruise
- RCS attitude control
- Four trajectory correction maneuvers,
  Site Adjustment maneuver 9/1/99,
  Contingency maneuver up to Entry – 7 hr.
- 11 Month Cruise
- Near-simultaneous tracking w/ Mars Climate
  Orbiter or MGS during approach

Launch
- Delta 7425
- Launch 1/3/99
- 576 kg Launch Mass

Entry, Descent, and Landing
- Arrival 12/3/99
- Jettison Cruise Stage
- Microprobes sep. from Cruise Stage
- Hypersonic Entry (6.9 km/s)
- Parachute Descent
- Propulsive Landing
- Descent Imaging [MARDI]

Landed Operations
- 76° S Latitude, 195° W Longitude
- Ls 256 (Southern Spring)
- 60–90 Day Landed Mission
- MVACS, LIDAR Science
- Data relay via Mars Climate
  Orbiter or MGS
- Commanding via Mars
  Climate Orbiter or
direct-to-Earth high-gain antenna
Mars Polar Lander

- One month after the Mars Orbiter was launched, on January 3, 1999, NASA launched three spacecraft using a single launch vehicle: the Mars Polar Lander (MPL) and two Deep Space 2 (DS2) probes.
- The Mars Lander was to land on the surface of the planet and perform experiments for 90 days. The DP2 were to be released above the planet surface and drop through the atmosphere, embedding themselves beneath the surface.
- According to plan, these three spacecraft ended communications as they prepared to enter the atmosphere of Mars on December 3, 1999. After arriving on the planet they were to resume communication on the evening of December 4, 1999.
Mars Polar Lander

- Communication was never reestablished. After a thorough investigation, the most probable cause seems to be the generation of spurious signal when the Lander’s legs were deployed during descent. This signal could give the Lander a false indication that it had landed, causing the engines to shut down. This would lead to a crash into the surface.
- The report claims that there was no software requirement to clear spurious signals prior to using the sensor information.
- During the test of the lander system, the sensors were incorrectly wired due to a design error. As a result the spurious signals were not identified by the systems test and the systems test was not repeated with properly wired touchdown sensors. While the most probable direct cause of the failure is premature engine shutdown, the underlying cause is inadequate software design and system test.
Pathfinder Mars Rover

- Landing: July 4, 1997; initial successes
- Intermittent software system resets
  - Delay of mission, serious loss of data
  - Happens when “too much” data are sent over a shared information bus
  - Low priority data collection task locks the bus,
  - Gets interrupted by medium priority tasks
  - High priority data distribution task fails to complete: cannot get shared bus
  - Scheduler detects pending high priority task and resets all the hardware and software
- Lots of tasks using a common bus
Pathfinder Mars Rover

● Problem modeling
● Priority inversion
  ○ high priority task delayed in a critical section by low priority tasks
  ○ More details later in the course
● Solutions proposed
  ○ Priority inheritance: low priority tasks entering critical section will inherit the highest priority of waiting tasks
● Solved the Pathfinder reset problem
Real-Time Operating System

- Often the need for an OS is downplayed
  - Just program it, no need for an OS!
  - The OS gets in the way!
  - RTOS aren't!

- Function of an OS
  - Manage threads, tasks, processes
  - Manage resources: memory, files, ports, ...

- Meeting with company in Taiwan (Jan. 2012) for computer vision research project
  - Company boss told engineers that they should use DOS only to get faster performance out of the system
  - Claims overhead of Linux/Windows is 40%
  - Camera captures image, cleans up the image, converts into RGB colour space, and stores in RAM
  - How to run the ethernet stack/networking in DOS?
Need for an OS (Threads)

- Introductory robotics workshop for Children
- Auckland NZ, Explorers’ Club – Mensa for children
- Lego Mindstorms Robotics Invention Kit
  - Actuators (3): DC Motors
  - Sensors (3): Light, touch
  - NQC programming language
Children Robotics Programming

- Use robotics to teach programming
  - Variables
  - Sequence
  - Selection (if .. then .. else)
  - Iteration (for, while)
  - Subroutines
- Build a line tracker using
  - Differential drive (right and left motor)
  - Light sensor
Children Robotics Programming

- Task 1: Drive in a “straight” line for one 1m
- Task 1b: Drive forward – backward, turns

```c
void main() {
  OnFwd(OUT_A);
  OnFwd(OUT_C);
  Wait(400);
  Off(OUT_A);
  Off(OUT_C);
  Wait(500);
  OnRev(OUT_A);
  OnRev(OUT_C);
  Wait(200);
  Off(OUT_A);
  Off(OUT_C);
}
```
Children Robotics Programming

- Search pattern for the line
- Introduce command blocks {}
- Introduce constants
- Introduce repeat loops

```c
#define TEN_DEGREE_WAIT 3
#define TEN_CM_WAIT 50

task main() {
    /* Make 10 Squares */
    repeat(10) {
        repeat(4) {
            /* Drive 1m then turn 90 o*/
            OnFwd(MOTOR_LEFT);
            OnFwd(MOTOR_RIGHT);
            Wait(10*TEN_CM_WAIT);
            OnRev(MOTOR_LEFT);
            OnFwd(MOTOR_RIGHT);
            Wait(9 * TEN_DEGREE_WAIT);
        }
    }
    Off(MOTOR_LEFT);
    Off(MOTOR_RIGHT);
}
```
Children Robotics Programming

- Introduce variables
- Mathematical computation
- While loops with variable terminating conditions

```c
// Parameters
int square_length = 1;
int square_add = 1;

/* Do this until the length is less than 0.5m */
while (square_length < 5) {
    repeat(4) {
        /* Drive forward then turn*/
        OnFwd(MOTOR_LEFT);
        OnFwd(MOTOR_RIGHT);
        Wait(square_length * TEN_CM_WAIT);
        OnRev(MOTOR_LEFT);
        OnFwd(MOTOR_RIGHT);
        Wait(9 * TEN_DEGREE_WAIT);
    }
    /* Make the next square larger */
    square_length = square_length + square_add;
}
Off(MOTOR_LEFT);
Off(MOTOR_RIGHT); }
```
Children Robotics Programming

- How to add the line sensor? LIGHT_SENSOR
- Memory mapped IO
- Stop when the robot finds the line
- Try 1:

```c
while (LIGHT_SENSOR >= 42) {
    OnFwd( MOTOR_LEFT);
    OnFwd(MOTOR_RIGHT);
    Wait(square_length*TEN_CM_WAIT);
    OnRev(MOTOR_LEFT);
    OnFwd(MOTOR_RIGHT);
    Wait(9 * TEN_DEGREE_WAIT);
    ...
}
```
Children Robotics Programming

- Need to check the light sensor continuously
- Try 2:
  - Works but
    - Simplicity
    - Abstraction, Data Hiding
    - What about if we have found the line?

```c
while ( ! done ) {
  OnFwd( MOTOR_LEFT);
  OnFwd(MOTOR_RIGHT);
  repeat(square_length * TEN_CM_WAIT) {
    if (LIGHT_SENSOR >= 42) {
      done = true;
      break; // No continue
    }
  }
  if ( ! done ) {
    OnRev(MOTOR_LEFT);
    OnFwd(MOTOR_RIGHT);
    repeat(9 * TEN_DEGREE_WAIT) {
      if (LIGHT_SENSOR >= 42) {
        done = true;
        break;
      }
    }
    ...
  }
```
Children Robotics Programming

- Separate task state machine from simple behaviours
- Use separate tasks
- Proper information hiding
- Change logic easily

```c
role task main() {
    start drive_spiral;
    start check_playing_field;
}

role task check_playing_field() {
    /* Initialize the light sensor */
    SetSensor(LIGHT_SENSOR, SENSOR_LIGHT);
    SelectDisplay(DISPLAY_SENSOR_1);

    while (true) {
        if (LIGHT_SENSOR >= 42) {
            stop drive_spiral;
            PlaySound(SOUND_DOUBLE_BEEP);
            break;
        }
    }
    Off(MOTOR_LEFT);
    Off(MOTOR_RIGHT);
}
```
Children Robotics Programming

- Track the line
- If on the line => straight
- Else sweep for the line

```c
task search_line() {
    int angle = 1; int turn = 1;
    Off(MOTOR_LEFT); Off(MOTOR_RIGHT);
    while (angle_searched < 18) {
        if (turn == 0) { /* Right Turn - do left next*/
            OnRev(MOTOR_LEFT); OnFwd(MOTOR_RIGHT);
            Wait(angle * TEN_DEGREE_WAIT);
            turn = 1;
        } else { /* Left Turn - do right next*/
            OnFwd(MOTOR_LEFT); OnRev(MOTOR_RIGHT);
            Wait(9 * TEN_DEGREE_WAIT);
            turn = 0;
        }
        angle = angle + 2;
    }
    Off(MOTOR_LEFT); Off(MOTOR_RIGHT);
}```
Children Robotics Programming

- State 1: Drive square
- State 2: Drive straight
- State 3: Search line
- Concurrency is easy
  - when explained with robots
  - when you do not focus on synchronization problems

```cpp
 task check_playing_field() {
    /* Initialize the light sensor */
    ...
    while( true ) {
        if (LIGHT_SENSOR >= 42) {
            stop drive_square;
            PlaySound(SOUND_DOUBLE_BEEP);
            break;
        }
    }
    while( true ) {
        if (LIGHT_SENSOR >= 42) {
            stop search_line;
            start drive_straight;
        } else {
            stop drive_straight;
            start search_line;
        }
    }
    Off(MOTOR_LEFT); Off(MOTOR_RIGHT);
}
```
Real-Time Systems Programming

Program usually consists of

- **Program logic**
  - Modelled as an extended finite state machine
- **Synchronous/periodic tasks**
  - Polling a sensor
  - Controlling a motor
- **Asynchronous/aperiodic tasks**
  - Process an incoming character from the serial port
- Operating system manages these various tasks and provides logical separation
Real-Time Tasks

- **Periodic tasks**
  - Time-driven. Characteristics are known a priori
  - Task $Ti$ is characterized by $(pi, ci)$
- **E.g.**: Task monitoring temperature of a patient in an ICU.
- **Aperiodic tasks**
  - Event-driven. Characteristics are not known a priori
  - Task $Ti$ is characterized by $(ai, ri, ci, di)$
  - **E.g.**: Task activated upon detecting change in patient’s condition.
- **Sporadic Tasks**
  - Aperiodic tasks with known minimum inter-arrival time.
- $pi$ : task period $ai$ : arrival time $ri$ : ready time
- $di$ : deadline $ci$ : worst case execution time.
Task Constraints

- Deadline constraint
- Resource constraints
  - Shared access (read-read)
  - Exclusive access (write-x)
- Precedence constraints
  - T1 -> T2: Task T2 can start executing only after T1 finishes its execution
- Fault-tolerant Requirements
  - To achieve higher reliability for task execution
  - Redundancy in execution
- Scheduable
  - A schedule can be found for all the tasks that satisfies all deadlines
Predictability

● The most common denominator that is expected from a real-time system is predictability.
  ○ The behavior of the real-time system must be predictable which means that with certain assumptions about workload and failures, it should be possible to show at “design time” that all the timing constraints of the application will be met.
● For static systems, 100% guarantees can be given at design time.
● For dynamic systems, 100% guarantee cannot be given since the characteristics of tasks are not known a priori.
● In dynamic systems, predictability means that once a task is admitted into the system, its guarantee should never be violated as long as the assumptions under which the task was admitted hold.
Common Misconceptions (Stankovic 1988)

- Real-time computing is equivalent to fast computing
  - Faster hardware implies all deadlines will be met
  - Fast is relative. More important that system is fast enough
  - Deterministic and predictable
  - Worst-case response times of interest rather than average-case
- Real-time programming is assembly coding, priority interrupt programming, and writing device drivers
  - RT systems are low-level coding done using ad-hoc methods
Common Misconceptions (Stankovic 1988)

- Real-time systems operate in a static environment.
- The problems in real-time system design have all been solved in other areas of computer science.
- Just hacking — no science/math is needed
  - Scheduling theory, software design, formal methods and RTOS are changing things
References

Lecture slides taken from various sources

● Dr. Michael Nolin

● Dr. G. Manimaran
  ○ http://www.ee.iastate.edu/~gmani/cpre558/lecture_notes.htm

● B.P. Douglass book
  ○ “Doing Hard Time”.

● Dr. Il Marina
  ○ http://cs.haifa.ac.il/courses/rtos/RTCourse-lec1.ppt