Roadmap

- Overview of the RTSJ
- Memory Management
- Clocks and Time
- **Scheduling and Schedulable Objects**
- Asynchronous Events and Handlers
- Real-Time Threads
- Asynchronous Transfer of Control
- Resource Control

Scheduling and Schedulable Objects

Lecture aims:

- To give an overview of fixed priority schedule
- To present the RTSJ Basic Scheduling Model

Introduction

- Real-time systems must be able to interact with their environment in a timely and predictable manner
- Designers must engineer analyzable systems whose timing properties can be predicted and mathematically proven correct
- Scheduling is the ordering of thread executions so that the underlying hardware resources and software resources (shared data objects) are efficiently and predictably used

Scheduling

- In general, scheduling consists of three components
  - an algorithm for ordering access to resources (scheduling policy)
  - an algorithm for allocating the resources (scheduling mechanism)
  - a means of predicting the worst-case behaviour of the system when the policy and mechanism are applied (schedulability analysis or feasibility analysis)
- Once the worst-case behavior of the system has been predicted, it can be compared with the system's timing requirements to ensure that all deadlines will be met

Fixed Priority Scheduling: Policy

- FPS requires
  - statically allocating schedulable objects to processors
  - ordering the execution of schedulable objects on a single processor according to a priority
  - assigning priorities to schedulable objects at their creation time — although no particular priority assignment algorithm is mandated, usually the shorter the deadline, the higher the priority
  - priority inheritance when accessing resources

FPS: Mechanism and Analysis

- **Mechanism**: FPS requires preemptive priority-based dispatching of processes — the processing resource is always given to the highest priority runnable schedulable object
- **Feasibility analysis**: There are many different techniques for analyzing whether a fixed priority-based system will meet its deadlines. Perhaps the most flexible is response time analysis

Information Needed for Analysis

- View the system as consisting of a number of schedulable objects
- Each schedulable object is characterized by its
  - release profile
  - processing cost per release
  - other hardware resources needed per release
  - software resources per release
  - deadline
  - value

Release Profile

- Typically after a schedulable object is started, it waits to be **released**
- When released it performs some computation and then waits to be released again (its completion time)
- The release profile defines the frequency with which the releases occur; they may be time-triggered (periodic) or event triggered
- Event triggered releases are further classified into sporadic (meaning that they are irregular but with a minimum inter-arrival time) or aperiodic (meaning that no minimum inter-arrival assumptions can be made)

Processing cost per release

- This is a measure of how much time is required to execute the computation associated with the schedulable object's release
- This may be a worst-case value or an average value depending on the feasibility analysis

Other resources

- Hardware: (other than the processor)
  - For networks, it is usually the time needed or bandwidth required to send messages
  - For memory, it is the amount of memory required by the schedulable objects (and types of memory)
- Software resources: a list of the non shareable resources that are required and the cost of using each resource
  - Access to non-shareable resources is a critical factor when performing schedulability analysis
  - Non shareable resources are usually non preemptible. Consequently when a schedulable tries to acquire a resource it may be blocked if the resource is already in use
  - This blocking time has to be taken into account

Deadline

- The time which the schedulable object has to complete the computation associated with each release
- As usually only a single deadline is given, the time is a relative value rather than an absolute value

Value

- A metric which indicates the schedulable objects contribution to the overall functionality of the application. It may be
  - a very coarse indication (e.g, safety critical, mission critical, non critical)
  - a numeric value giving a measure for a successful meeting of a deadline
  - a time-valued function which takes the time at which the schedulable object completes and returns a measure of the value (for those systems where there is no fixed deadline)

Online versus Off-line Analysis

- A key characteristic of schedulability (feasibility) analysis is whether the analysis is performed off-line or on-line
- For safety critical systems, where the deadlines associated with schedulable objects must always be met, off-line analysis essential
- Other systems do not have such stringent timing requirements or do not have a predictable worst case behavior; on-line analysis may be appropriate or, the only option available
- These systems must be able to tolerate schedulable objects not being schedulable and offer degraded services
- Furthermore, they must be able to handle deadlines being missed or situations where the assumed worst-case loading scenario has been violated

Basic Model

- The RTSJ provides a framework from within which on-line feasibility analysis of priority-based systems can be performed for single processor systems
- The specification also allows the real-time JVM to monitor the resources being used and to fire asynchronous event handlers if those resources go beyond that specified by the programmer
- The RTSJ introduces the notion of a **schedulable object** rather than considering just threads
- A schedulable object is any object which implements the `Schedulable` interface

Schedulable Objects Attributes I

- **`ReleaseParameters`**
  - the processing cost for each release
  - its deadline
  - if the object is periodic or sporadic then an interval is also given
  - event handlers can be specified for the situation where the deadline is missed or the processing resource consumed is greater than the cost specified
  - There is no requirement to monitor the processing time consumed by a schedulable object

Schedulable Objects Attributes II

- **`SchedulingParameters`**
  - an empty class
  - subclasses allow the priority of the object to be specified and, potentially, its importance to the overall functioning of the application
  - although the RTSJ specifies a minimum range of real-time priorities (28), it makes no statement on the importance parameter

Schedulable Objects Attributes III

- `MemoryParameters`
  - the maximum amount of memory used by the object in an associated memory area
  - the maximum amount of memory used in immortal memory
  - a maximum allocation rate of heap memory.

The Schedulable Interface

- Three groups of methods
  - Methods which communicate with the scheduler and result in the scheduler adding/removing the schedulable object from the list of objects it manages, or changing the parameters associated with it
    - the scheduler performs a feasibility test on the objects it manages
  - Methods which get/set the parameters associated with the schedulable object
    - If the parameter object set is different from the one currently associated with the schedulable object, the previous value is lost and the new one will be used in any future feasibility analysis
  - Methods which get/set the scheduler

Schedulable Interface

```
public interface Schedulable extends Runnable {
  public boolean addIfFeasible();
  public boolean addToFeasibility();
  public boolean removeFromFeasibility();
  public boolean setIfFeasible(ReleaseParameters r,
                 MemoryParameters mem);
  public boolean setReleaseParametersIfFeasible(
                 ReleaseParameters r);
  public boolean setSchedulingParametersIfFeasible(
                 SchedulingParameters s);
  public MemoryParameters getMemoryParameters();
  public void setMemoryParameters(MemoryParameters mem);
  public ReleaseParameters getReleaseParameters();
  public void setReleaseParameters(ReleaseParameters r);
  public SchedulingParameters getSchedulingParameters();
  public void setSchedulingParameters(SchedulingParameters s)
  public Scheduler getScheduler();
  public void setScheduler(Scheduler s);
```

Scheduler

```
public abstract class Scheduler  {
  protected Scheduler();
  public abstract boolean setIfFeasible(Schedulable s,
         ReleaseParameters r, MemoryParameters m);
  public abstract boolean setIfFeasible(
         Schedulable s, ReleaseParameters r,
         MemoryParameters m, ProcessingGroupParameters g);
  public abstract void fireSchedulable(Schedulable s);
  public static Scheduler getDefaultScheduler();
  public abstract String getPolicyName();
  public static void setDefaultScheduler(Scheduler s);
  protected abstract boolean addToFeasibility(Schedulable s)
  public abstract boolean isFeasible();
  protected abstract boolean removeFromFeasibility(
         Schedulable s);
}
```

The Priority Scheduler: Policy

- Orders the execution of schedulable objects on a single processor according to a priority
- Supports a real-time priority range of at least 28 unique priorities (the larger the value, the higher the priority)
- Allows the programmer to define the priorities (say according to the relative deadline of the schedulable object)
- Allows priorities may be changed at run time
- Supports priority inheritance or priority ceiling emulation inheritance for synchronized objects (covered later)

The Priority Scheduler: Mechanism

- Supports pre-emptive priority-based dispatching of schedulable objects — the processing resource is always given to the highest priority runnable schedulable object
- Does not defined where in the run queue (associated with the priority level), a pre-empted object is placed; however, a particular implementation is required to document its approach
- Places a blocked schedulable object which becomes runnable, or has its priority changed at the back of the run queue associated with its (new) priority
- Places a thread which performs a yield operation at the back of the run queue associated with its (new) priority

The Priority Scheduler: Feasibility Analysis

- Requires no particular analysis to be supported
- Default analysis assumes an adequately fast machine

The PriorityScheduler Class

```
public class PriorityScheduler extends Scheduler {
  public static final int MAX_PRIORITY,MIN_PRIORITY;
  protected PriorityScheduler();
  public void fireSchedulable(Schedulable s);
  public int getMaxPriority();
  public static int getMaxPriority(Thread thread);
  public int getMinPriority();
  public static int getMinPriority(Thread thread);
  public int getNormPriority();
  public static int getNormPriority(Thread thread);
```

The Parameter Classes

- Each schedulable objects has several associated parameters
- These parameters are tightly bound to the schedulable object and any changes to the parameters can have an immediate impact on the scheduling of the object or any feasibility analysis performed by its scheduler
- Each schedulable object can have only one set of parameters associated with it
- However, a particular parameter class can be associated with more than one schedulable object
- In this case, any changes to the parameter objects affects all the schedulable objects bound to that parameter

Release Parameters I

- Release parameters characterize
  - how often a schedulable object runs
  - the worst case processor time needed for each run
  - a relative deadline by which each run must have finished
- There is a close relationship between the actions that a schedulable object can perform and its release parameters
- E.g., the `RealtimeThread` class has a method called `waitForNextPeriod`; however a RT thread can only call this methods if it has `PeriodicParameters` associated with it
- This allows a thread to change its release characteristics and hence adapt its behavior

The ReleaseParameter Class

```
public class ReleaseParameters {

  protected ReleaseParameters();
  protected ReleaseParameters(RelativeTime cost,
                RelativeTime deadline,
                AsyncEventHandler overrunHandler,
                AsyncEventHandler missHandler);

  public RelativeTime getCost();
  public AsyncEventHandler getCostOverrunHandler();
  public RelativeTime getDeadline();
  public AsyncEventHandler getDeadlineMissHandler();
  public boolean setIfFeasible(RelativeTime cost,
                 RelativeTime deadline);
```

Release Parameters II

- The minimum information that a scheduler will need for feasibility analysis is the **cost** and **deadline**
- cost: a measure of how much CPU time the scheduler should assume that the scheduling object will require for each release
  - This is dependent on the processor on which it is being executed; consequently, any programmer-defined value will not be portable
- deadline: the time from a release that the scheduler object has to complete its execution
- `overrunHandler`: the asynchronous event handler that should be released if the schedulable object overruns
- `missHandler` is released if the schedulable object is still executing when its deadline arrives

Cost Enforcement

- If cost monitoring is supported:
  - the RTSJ requires that the priority scheduler gives a schedulable object a CPU budget of no more than its cost value on each release
  - if a schedulable objects overrun its cost budget, it is automatically descheduled immediately
  - it will not be re-scheduled until either its next release occurs (in which case its budget is replenished) or its cost value is increased

The PeriodicParameters Class

- For schedulable objects which are released on a regular basis
- The start time can be an `AbsoluteTime` or a `RelativeTime` and indicates when the schedulable should be first released
- For a real-time thread, the actual first release time is given by
  - if start is a RelativeTime value
    - Time of invocation of start method + start
  - if start is an AbsoluteTime value
    - Max of (Time of invocation of start method, start)
- A similar formula can be given for an async event handler
- The deadline for a schedulable with periodic parameters is measured from the time it is released not when it is started/fired

The PeriodicParameters Class

```
public class PeriodicParameters extends ReleaseParameters {
  public PeriodicParameters(
         HighResolutionTime start, RelativeTime period,
         RelativeTime cost, RelativeTime deadline,
         AsyncEventHandler overrunHandler,
         AsyncEventHandler missHandler);
  public RelativeTime getPeriod();
  public HighResolutionTime getStart();
  public void setPeriod(RelativeTime period);
  public void setStart(HighResolutionTime start);
  public boolean setIfFeasible(RelativeTime period,
          RelativeTime cost, RelativeTime deadline);
}
```

The AperiodicParameters Class I

```
class AperiodicParameters extends ReleaseParameters {
  public AperiodicParameters(
         RelativeTime cost, RelativeTime deadline,
         AsyncEventHandler overrunHandler,
         AsyncEventHandler missHandler);
  public boolean setIfFeasible(RelativeTime cost,
                               RelativeTime deadline);
```

The deadline of an aperiodic schedulable object can never be guaranteed, why? Hence, the deadline attribute should be interpreted as a target completion time rather than a strict deadline.

The AperiodicParameters Class II

- Schedulable object with aperiodic parameters are released at irregular intervals
- When an aperiodic schedulable object is released for the first time, it becomes runnable and is scheduled for execution.
- Before it completes its execution, it may be released again.
- It is necessary for an implementation to queue the release events to ensure that they are not lost
- The programmer is able to specify the length of this queue and the actions to be taken if the queue overflows

Alternative Schedulers

- Most RT OSs support fixed priority pre-emptive based scheduling with no online feasibility analysis
- As more computers become embedded, there is need for more flexible scheduling
- In general, there are three approaches to flexible scheduling:
  - Pluggable scheduler: the system provides a framework from within which different schedulers can be plugged in
  - Application-defined schedulers: the system notifies the application every time an event occurs which requires a scheduling decision; the application informs the system which thread should execute next
  - Implementation-defined schedulers: an implementation is allowed to define alternative schedulers; typically this would require the underlying system to be modified

RTSJ and Alternative Schedulers

- The RTSJ adopts the implementation-defined schedulers approach
- Applications can determine dynamically whether the real-time JVM on which it is executing has a particular scheduler
- This is the least portable approach, as an application cannot rely of any particular implementation-defined scheduler being supported

Flexible Scheduling and Priority-based Systems

- Priority-based scheduling is very flexible if schedulable objects can change their priority (as they can in the RTSJ)
- Much can be done which is portable by allowing the application to implement its own scheduling policy on top of the `PriorityScheduler`

EDF Scheduler Approach I

- All objects which are to be scheduled by the `EdfScheduler` run at one of three priority levels: low, medium and high
- When schedulable objects are released, they are released at the high priority level; they immediately call the reschedule method to announce themselves to the `EdfScheduler`
- The `EdfScheduler` keeps track of the schedulable object with the closest absolute deadline; this object has its priority set to the medium level
- When a call is made to reschedule, the `EdfScheduler` compares the deadline of the calling object with that of its closest deadline; if the caller has a closer deadline, the object with the current closest deadline has its priority set to low and the caller has its priority set to medium

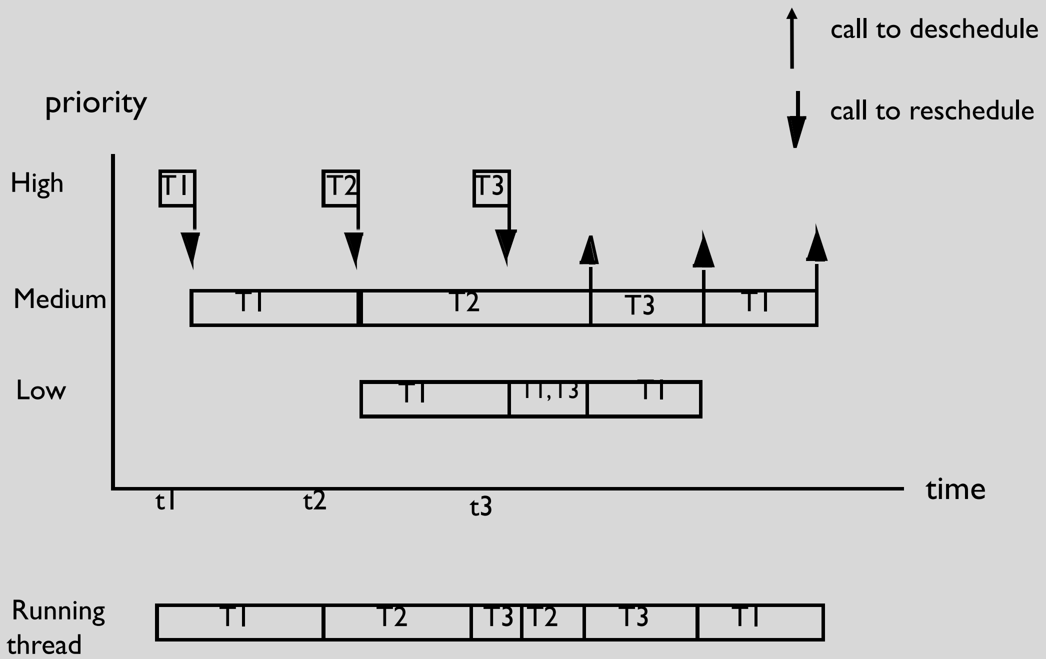
EDF Scheduler Approach II

- After the end of each release, a schedulable object calls deschedule
- The EdfScheduler sets caller's priority to high (ready for the next release), and then scans its list of schedulable objects to find the one with the closest deadline
- It sets the priority of this object to medium
- The constructor for the class gives the appropriate priority values for low, medium and high
- It also contains the maximum number of schedulable objects to be scheduled by the scheduler

EDF Example I

- Consider the execution of three real-time threads (T1,T2 and T3) which are released at times t1, t2, and t3 respectively (where t1 < t2 < t3)
- T2 has the closest deadline, followed by T3 and T1.

EDF Example II



EDF Example III

- With this approach, a thread with a longer deadline will execute in preference to a shorter deadline thread but only for a small limited time when it is released
- This can be ensured by encapsulating the calls to reschedule and deschedule

Summary

- Scheduling is the ordering of thread execution so that hardware and software resources are efficiently and predictably used
- The only scheduler mandated is a fixed priority scheduler (FPS)
- Scheduling policy: FPS requires
  - statically allocating schedulable objects to processors
  - ordering the execution of schedulable objects according to a priority
  - assigning priorities to schedulable objects at their creation time
  - priority inheritance when accessing resources.
- Scheduling mechanism: FPS requires preemptive priority-based dispatching of processes
- Feasibility analysis: RTSJ does not mandate any schedulability analysis technique