Real-Time Software

Real-Time Facilities

René Rydhof Hansen

14. september 2010

Topics

- Time
- Clocks, delays, and timeouts
- Specification of timing requirements
- Temporal scopes

Real-Time Systems

Characteristics of a RTS

Characteristics of a RTS

- Timing constraints
- Dependability requirements
- Concurrent control of separate components
- Facilities to interact with special purpose hardware

Characteristics of a RTS

- Timing constraints
- Dependability requirements
- Concurrent control of separate components
- Facilities to interact with special purpose hardware

Real-Time Facilities: Requirements

Interfacing with time

- Measuring durations
- Delaying processes until some future time
- Programming timeouts so non-occurrence of some event can be recognised

Representing timing requirements

- Specifying rates of execution
- Specifying deadlines

The Notion of Time

Fundamental Model

- Continuous
- Discrete
- Hybrid

Properties

- Transitive

$$\forall x, y, z\colon (x < y \land y < z) \implies x < z$$

- Linear

$$\forall x, y\colon x < y \lor y < x \lor x = y$$

- Irreflexive

$$\forall x\colon \neg(x < x)$$

- Dense

$$\forall x, y\colon x < y \implies \exists z\colon (x < z < y)$$

Access to a Clock

Time providers

- Direct access to time frame of environment through specialised hardware, e.g., UTC service provided by GPS
- Internal hardware (adequate approximation)
- Need a programming language abstraction, e.g., library functions, to access special hardware

Example (Clocks in Real-Time Java)

- `java.lang.System.currentTimeMillis`: returns the number of milliseconds since 01 JAN 1970 (GMT) and is used by `java.util.Date`
- Real-time Java adds real-time clocks with high reslution time types

Real-Time Java: Representing Time

Clock class

- General framework that allows definition of many (types of) clocks, e.g., execution time clocks
- Always one real-time clock sync'ed with external world: getRealTimeClock
- setResolution only supported on some platforms

HighResolutionTime

- HighResolutionTime is the "base" class for time in RT Java
- Subclassed for further use

AbsoluteTime and RelativeTime

- Subclasses of HighResolutionTime: one for absolute time and one for relative time (durations)

Real-Time Java: Clock class

```
public abstract class Clock
{
  public Clock();

  public static Clock getRealtimeClock();

  public abstract RelativeTime getResolution();

  public AbsoluteTime getTime();
  public abstract void getTime(AbsoluteTime time);

  public abstract
     void setResolution(RelativeTime resolution);

}
```

Real-Time Java Time Types: HighResolutionTime

```
public abstract class HighResolutionTime implements
        java.lang.Comparable
{
  public abstract AbsoluteTime absolute(Clock clock,
                              AbsoluteTime destination);

  ...

  public boolean equals(HighResolutionTime time);

  public final long getMilliseconds();
  public final int getNanoseconds();

  public void set(HighResolutionTime time);
  public void set(long millis);
  public void set(long millis, int nanos);
}
```

Real-Time Java Time Types: AbsoluteTime

```
public class AbsoluteTime extends HighResolutionTime
{
  // various constructor methods including
  public AbsoluteTime(AbsoluteTime T);
  public AbsoluteTime(long millis, int nanos);

  public AbsoluteTime absolute(Clock clock,
                               AbsoluteTime dest);

  public AbsoluteTime add(long millis, int nanos);
  public final AbsoluteTime add(RelativeTime time);

  ...

  public final RelativeTime subtract(AbsoluteTime time);
  public final AbsoluteTime subtract(RelativeTime time);

}
```

Real-Time Java Time Types: RelativeTime

```
public class RelativeTime extends HighResolutionTime
{
  // various constructor methods including
  public RelativeTime(long millis, int nanos);
  public RelativeTime(RelativeTime time);

  public AbsoluteTime absolute(Clock clock,
                               AbsoluteTime destination)

  public RelativeTime add(long millis, int nanos);
  public final RelativeTime add(RelativeTime time);

  public final RelativeTime subtract(RelativeTime time);

  ...
}
```

Real-Time Java: Measuring Time

Example (Measuring Time)

Real-Time Java: Measuring Time

Example (Measuring Time)

```
void foo()
{
  AbsoluteTime oldTime, newTime;
  RelativeTime interval;
  Clock clock = Clock.getRealtimeClock();

  oldTime = clock.getTime();

  // other computations

  newTime = clock.getTime();
  interval = newTime.subtract(oldTime);
}
```

Clocks in C and POSIX

ANSI C

- Standard library for interfacing to calendar time
- Defines a basic time type `time_t` and several routines for manipulating objects of type time

POSIX

- Requires at least one clock of minimum resolution 50Hz (20ms)

C/POSIX Real-Time Clocks

```
#define CLOCK_REALTIME ...; // clockid_t type

struct timespec {
  time_t tv_sec;   /* number of seconds */
  long   tv_nsec;  /* number of nanoseconds */
};
typedef ... clockid_t;

int clock_gettime(clockid_t clock_id, struct timespec *tp);
int clock_settime(clockid_t clock_id,
                  const struct timespec *tp);
int clock_getres(clockid_t clock_id, struct timespec *res);

int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);
int clock_getcpuclockid(pthread_t thread_id,
                        clockid_t *clock_id);

int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);
/* nanosleep return -1 if the sleep is interrupted by a */
/* signal. In this case, rmtp has the remaining sleep time */
```

Delaying a process

Why?

Relative and Absolute Delay

- Relative delay, e.g., “delay for 10ms”
- Absolute delay, e.g., “resume in 10ms”

Limitations

Delay operations (usually) only guarantees process is made runnable

- Granularity difference between clocks
- Disabled interrupts
- Process runnable but not executing

Delaying a process

Why?

- Useful (necessary!) to avoid busy-waits

Relative and Absolute Delay

- Relative delay, e.g., "delay for 10ms"
- Absolute delay, e.g., "resume in 10ms"

Limitations

Delay operations (usually) only guarantees process is made runnable

- Granularity difference between clocks
- Disabled interrupts
- Process runnable but not executing

Language Support for Delaying a Process

Example (Relative delay in Ada)

```
Start := Clock;
loop
  exit when (Clock - Start) > 10.0
end loop;
```

Why Ada?

- A *lot* of legacy code
- Still often used for critical embedded RTSs

Language Support for Delaying a Process

- POSIX: `sleep` and `nanosleep`
- Java: `sleep`
- Real-Time Java: high resolution `sleep`

Absolute Delays

Absolute delay

```
START := Clock;
FIRST_ACTION;
delay 10.0 - (Clock - START);
SECOND_ACTION;
```

Absolute delay

```
START := Clock;
FIRST_ACTION;
delay until START + 10.0;
SECOND_ACTION;
```

Absolute Delays

Absolute delay... WRONG!

```
START := Clock;
FIRST_ACTION;
delay 10.0 - (Clock - START);
SECOND_ACTION;
```

Absolute delay

```
START := Clock;
FIRST_ACTION;
delay until START + 10.0;
SECOND_ACTION;
```

- Both `delay` and `delay until` only guarantees minimum delays
- Real-Time Java: `sleep` can be relative or absolute
- POSIX requires use of an absolute timer and signals

Drift

Definition (Local Drift)

The time over-run associated with both relative and absolute delays

- Local drift can not be eliminated
- Cumulative drift arising from superimposed local drifts can be eliminated

Example (Regular activity)

```
task T;
task body T is
begin
  loop
    Action;
    delay 5.0; -- cannot delay less than 5 sec.
  end loop;
end T;
```

Handling Cumulative Drift: Periodic Activity

Example (Periodic activity)

```
task body T is
  Interval : constant Duration := 5.0;
  Next_Time : Time;
begin
  Next_Time := Clock + Interval;
  loop
    Action;
    delay until Next_Time;
    Next_Time := Next_Time + Interval;
  end loop;
end T;
```

- Will run on average every 5 seconds
- Local drift only
- If `Action` takes six seconds, the delay statement will have no effect

Timeouts

- Simplest time constraint
- Useful for specifying maximum wait time
  - Detect "non-occurrence" of event
  - Timely error recovery
- Useful for specifying maximum execution time
  - Result refinement (first compute fast but imprecise result, then use remaining time to refine result)

Timeouts in Real-Time Java

```
public class Timed
  extends AsynchronouslyInterruptedException
  implements java.io.Serializable
{
  public Timed(HighResolutionTime time)
    throws IllegalArgumentException;

  public boolean doInterruptible(Interruptible logic);

  public void resetTime(HighResolutionTime time);
}
```

- Uses ATC (asynchronous transfer of control): execution may be interrupted at any time (not only at sync time)

POSIX

- POSIX does not support ATC and, therefore, it is difficult to get the same effect as Ada and Real-Time Java
- POSIX does support timers

Temporal Scopes

Definition (from [BW])

"A collection of statements with an associated timing constraint"

Attributes of temporal scopes:

- Deadline: the time by which the execution of a TS must be finished
- Minimum delay: the minimum amount of time that must elapse before start of execution of a TS
- Maximum delay: the maximum amount of time that can elapse before the start of execution of a TS
- Maximum execution time: maximum amount of time a TS can be actively executing
- Maximum elapse time: maximum of time that can elapse from start to end of execution of a TS

Temporal scopes with combinations of these attributes are also possible

Temporal Scopes

Recurrence of temporal scopes

- Periodic
- Aperiodic (potentially zero minimal interarrival time)
- Sporadic (non-zero minimal interarrival time)

Deadlines

- Hard
- Soft
- Interactive ("only" performance issue)
- Firm

Scheduling

Algorithm to reduce non-determinism in a system to enable reasoning about feasibility of temporal scopes

Exercises

1. [BW] 9.1.
2. [BW] 9.2.
3. Discuss notions of time and (engineering) consequences.

Summary and Next Time

Summary:

- Time in a real-time programming language
  - Access to a clock
  - Delay
  - Timeouts
- Temporal scopes
  - Deadline, minimum delay, maximum delay, maximum execution time, maximum elapse time

Next time:

- Scheduling
- Response-Time Analysis (RTA)