Synchronization

Synchronization in real-time systems

Synchronization: the run-time overhead?

- Is the potential overhead justified for single-data-structures?

```plaintext
int i, j;

i++;  // in one thread
j++;  // in another thread
```

- Are those operations atomic?
- Do we really need to introduce full-fledged synchronization methods here?

Word-access atomicity:

Assuming that any access to a word in the system is an atomic operation.

E.g. assigning two values (not wider than the size of word) to a memory cell simultaneously:

```plaintext
Task 1: x := 0;
Task 2: x := 5;
```

will result in either x = 5 or x = 0 — and no other value is ever observable.

Synchronization by flags

Assuming further that there is a shared memory area between two processes:

- A set of processes agree on a word-level atomic variable operating as a flag to indicate synchronization conditions.

```plaintext
var Flag : boolean := false;
```

```plaintext
i++;  // in one thread
j++;  // in another thread
```

- Depending on the hardware and the compiler, it might be atomic, it might be not!
- Handling a bit vector on a 32- or 64-bit controller will be atomic, but perhaps it is not.
- Any manipulations on the main-memory will not be atomic, but perhaps it is a register.
- Static data to a load/store operation cycle, the operation will be atomic. But perhaps the processor supplies atomic operations for the actual cases.

Assuming that all ‘perhapses’ are applying: how to expand this code?

Synchronization by semaphores

Assuming further that there is a shared memory area between two processes:

- A set of processes agree on a word-level atomic variable operating as a flag to indicate synchronization conditions.

```plaintext
var sync : semaphore := 0;
```

```plaintext
if S > 0 then S := S - 1;
```

- An atomic operation: e.g. assigning two values (not wider than the size of word) to a memory cell simultaneously:

```plaintext
if i > 0 then i := i - 1;
```

- Many effects stemming from asynchronous memory accesses are interpreted as (hardware) ‘perhapses’, since they are access and effect usually only some parts of the data.

- On an assembly level: synchronization by employing knowledge about the atomicity of CPU-operations and interrupt-structures is possible and done elsewhere.

- In anything higher than assembly level on small, predictable processors:

```plaintext
Flag := true;
```

- Measures for synchronization are required!

Synchronization in real-time systems

Some synchronization terms:

- Condition synchronization:
  - synchronization with an event given by another task.

- Critical sections:
  - code fragments which contain access to shared resources and need to be executed without interference with other critical sections, sharing the same resources.

- Mutual exclusion:
  - protection against asynchronous access to critical sections.

- Atomic operations:
  - an operation which is guaranteed by the underlying system (e.g. hardware).

- There must be a set of atomic operations to start with!

Synchronization

Word-access atomicity:

Assuming that any access to a word in the system is an atomic operation.

E.g. assigning two values (not wider than the size of word) to a memory cell simultaneously:

```plaintext
Task 1: x := 0;
Task 2: x := 5;
```

will result in either x = 5 or x = 0 — and no other value is ever observable.

Synchronization by flags

Assuming further that there is a shared memory area between two processes:

- A set of processes agree on a word-level atomic variable operating as a flag to indicate synchronization conditions.

```plaintext
var Flag : boolean := false;
```

```plaintext
i++;  // in one thread
j++;  // in another thread
```

- Depending on the hardware and the compiler, it might be atomic, it might be not!
- Handling a bit vector on a 32- or 64-bit controller will be atomic, but perhaps it is not.
- Any manipulations on the main-memory will not be atomic, but perhaps it is a register.
- Static data to a load/store operation cycle, the operation will be atomic. But perhaps the processor supplies atomic operations for the actual cases.

Assuming that all ‘perhapses’ are applying: how to expand this code?

Synchronization by semaphores

Assuming further that there is a shared memory area between two processes:

- A set of processes agree on a word-level atomic variable operating as a flag to indicate synchronization conditions.

```plaintext
var sync : semaphore := 0;
```

```plaintext
if S > 0 then S := S - 1;
```

- An atomic operation: e.g. assigning two values (not wider than the size of word) to a memory cell simultaneously:

```plaintext
if i > 0 then i := i - 1;
```

- Many effects stemming from asynchronous memory accesses are interpreted as (hardware) ‘perhapses’, since they are access and effect usually only some parts of the data.

- On an assembly level: synchronization by employing knowledge about the atomicity of CPU-operations and interrupt-structures is possible and done elsewhere.

- In anything higher than assembly level on small, predictable processors:

```plaintext
Flag := true;
```

- Measures for synchronization are required!

Synchronization in real-time systems

Some synchronization terms:

- Condition synchronization:
  - synchronization with an event given by another task.

- Critical sections:
  - code fragments which contain access to shared resources and need to be executed without interference with other critical sections, sharing the same resources.

- Mutual exclusion:
  - protection against asynchronous access to critical sections.

- Atomic operations:
  - an operation which is guaranteed by the underlying system (e.g. hardware).

- There must be a set of atomic operations to start with!
Synchronization

Semaphores in Ada95

package Ada.Synchronous.Task_Control is
  type Suspension_Object is limited private;
  function Current_State (S : Suspension_Object) return Boolean;
  procedure Set_False (S : in out Suspension_Object);
  procedure Take (S : in out Suspension_Object;
                  guard : in out Boolean; -- guarded by the semaphore
                  when : in out Boolean; -- guarded by the semaphore
                  new_value : out Integer;
                  old_value : in out Integer);
end Ada.Synchronous.Task_Control;

package Ada.Synchronous.Task_Control is
  type Suspension_Object is limited private;
  function Current_State (S : Suspension_Object) return Boolean;
  procedure Set_False (S : in out Suspension_Object);
  procedure Take (S : in out Suspension_Object;
                  guard : in out Boolean; -- guarded by the semaphore
                  when : in out Boolean; -- guarded by the semaphore
                  new_value : out Integer;
                  old_value : in out Integer);
end Ada.Synchronous.Task_Control;

Basic idea:
• All guarded regions are isolated from the rest of the code.
• All guards need to be re-evaluated, when another guarded region is entered.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
• All guarded regions are isolated from the rest of the code.
Considerations:
1. Synchronized methods and code blocks:
   - In order to implement monitors all methods in an object need to be synchronized.
   - In any other standard method a break for the monitor and enter any time.
   - Methods outside the monitor object can synchronize after object.
   - In impossible to analyze a monitor locally since lock accesses can exist all over the system.
   - Static data is shared between all objects of a class.
   - Access to static data need to be synchronized over the whole class.
   - Either in static synchronized blocks synchronized (this.getInstance) () or in static methods public synchronized static method ().

2. Notification methods:
   - wait (x) is implemented as thread that releases the monilex.
   - notify (x) is used for releasing the lock.
   - Methods that are associated with notification need to wait for the lock.
   - A wait notification thread is held in a queue, the wait1 (foreground) is waiting for the lock in order to release them in the lock (as possible to Java).
   - There are two explicit conditional variables:
     - on notified thread need to wait for the lock to be released and reevaluate its entity condition.

Methods outside the monitor object can synchronize at this object.
- Calls will keep all enclosing locks.
- All sub-classes will keep all enclosing locks.
- Calls will keep all enclosing locks.

Examples (tasks and creation):
- Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
  - Synchronized methods and code blocks:
Monitors in Real-time Java

protected type SharedData (Initial : Data_Item) is
  -- the protected variables are encapsulated.
  -- the guard syntax is re-used.
private
  Buffer_Size : constant Integer := 10;
begin
  entry Put (Item : in Data_Item)
  when Num < Buffer_Size
  -- the guard is re-used.
  ∼ protected procedures allow simultaneous access
  to different parts of the data.
protected
  type Buffer_T is array (Index) of Data_Item;
protected
  Index := mod Buffer_Size;
type Index is mod Buffer_Size;
subtype Count is Natural range 0 .. Buffer_Size;
end SharedData;
Synchronization by protected objects in Ada95

Protected entries are used like task entries.

```plaintext
Buffer : BoundBuffer;
```

Further refinements on task control:
- Entry families
  - A protected entity declaration can contain a discrete subtype selector, which can be evaluated by the task (rather parameters cannot be evaluated by barriers) and implements an array of protected entries.
- Response facility
  - A protected operation can use `response` to release tasks from other internal, external, or private entries. The current protected operation is finished and the lock on the object is released.
- Private entries
  - A protected entry can be only accessible from outside the protected object, but can be employed in destinations for `response` operations.

Synchronization by protected objects in Ada95

Guarded operations allow to protect critical regions.

```plaintext
package Node is
  type Node, Node_Type is new Integer, String, String;
  protected procedure SetNode (Node : in Node_Type; Node_Type : in Node_Type);
  protected entry Wait (Node : in Node_Type; Node_Type : in Node_Type);
private
  Current_Node : Node_Type := Node_Type;
and Node_Type := Node_Type;
end Node;
```

Synchronization by protected objects in Ada95

Shared memory based synchronization

**Criteria**
- Level of abstraction
- Control of distributed concepts
- Support for consistency and correctness obligations
- Error recovery
- Repeatability, re-execution by release some semantics of conditional signals (exit, signal)
- Portability
- Efficiency

Shared memory based synchronization

**Module-1, CHIL:**
- Full monitor implementation
- Directly using Ada99 Monitor concept.
- No messages, no less...
- All features are in context of mentioned apply.

Shared memory based synchronization

**Ada95**
- Complete synchronization support
- Low-level synchronization for very specific cases
- Predictable timing (e.g., schedule)
- Mutual exclusion and synchronization conditions are realized by the compiler or the runtime environment directly, rather than the programmer

Message-based synchronization

**Synchronization model**
- Asynchronous
- Synchronize
- Remote invocation

**Addressing (name space)**
- Direct communication
- Mailbox communication

**Message structure**
- Primitive
- Extends to basic types
- Restricted to untyped communication

Real-Time Java

- Mutual exclusion
- Synchronize
- Remote invocation
- Direct communication

**Message-based synchronization**
- Asynchronous messages
  - If there is a listener
  - Need the message directly
Synchronization

Message-based synchronization

Asynchronous messages
- If the receiver becomes available at a later stage
  - the message need to be buffered

Synchronous messages
- Delay the sender:
  - until the receiver got the message
  - two asynchronous messages required

Remote invocation
- Delay the sender:
  - until a receiver becomes available
  - a receiver executed an addressed routine

Asynchronous remote invocation
- Delay the sender:
  - until a receiver becomes available
  - a receiver executed an addressed routine

Addressing (name space)

Communication mediums
- Connections
  - one-to-one
  - one-to-many
  - many-to-one
  - many-to-many
- Functionality
  - buffer, queue, synchronization
  - broadcast
  - general server, synchronization
  - local server, synchronization
  - one-to-one
  - one-to-many
  - many-to-one
  - many-to-many

Message structure
- Machine-dependent representations need to be taken care of in a distributed environment.
- Communication system is often outside the typed language environment.

Most communication systems are handling streams (packets) of a basic element type only.
- Conversion routines for data structures other than the basic element type are supplied...
  - manually (POSIX)
  - automatically (Real-time CORBA)
  - automatic and are typed-persistent (Ada95)

Synchronous vs. asynchronous communications
- Synchronous communications are emulated by a combination of synchronous messages in some systems.
- Asynchronous communications can be emulated in synchronous message passing systems by introducing "buffer-under-the-coupling" sender and receiver as well as allowing (buffer) broadcasts.

Parallel communication:
- synchronous messages / remote invocations
- Real-time messages / remote invocations

Remote invocation
- P2
- P1

Synchronous messages
- a receiver becomes available
- a receiver got the message

Asynchronous messages / asynchronous remote invocations
- a receiver becomes available
- a receiver got the message

Delay the sender, until:
- a receiver becomes available
- a receiver executed an addressed routine
- a receiver got the message

If the receiver becomes available at a later stage:
- the message need to be buffered

Conversion routines for data structures other than the basic element type are supplied...
- manually (POSIX)
- automatically (Real-time CORBA)
- automatic and are typed-persistent (Ada95)
Entry families

```
case 2: r := 2; break;
case 3: r := 1;
```

`end <entry_name>;
```

Parameters cannot be direct involved tasks.

```
all

... --

Exceptions, which are not handled during the rendezvous phase

```

Accept statements can have a dedicated exception handler (like any other code-block).

```
fi
```

Helpful e.g. to synchronize more than two tasks.

```
rem. invoc.
```

Selective waiting

```
Dinclusive guarded commands:
if (true) then
  p1 := c1
else
  p1 := c2
fi;
```

The programme needs to design the alternatives as parallel options, all cases need to be covered and overlapping conditions need to lead to the same result.

Extremely important: C-switch

```
switch (c) {
  case 0:
    c := 0;
    break;
  case 1:
    c := 1;
    break;
  case 2:
    c := 2;
    break;
  default:
    c := 0;
```

Selection is non-deterministic.

```
```

the sequence of alternatives has a crucial role.
Synchronization

Selective waiting in Occam2

Basic forms of selective synchronization

Non-determinism in selective synchronizations

Conditional & timed entry-calls

Shared memory based synchronization

Message-based selective synchronization in Ada95

Selective & timed entry-calls

Guarded select-or-delay

Synchronization

Selective waiting in Occam2

Conditional & timed entry-calls

Summary

Basic forms of selective synchronization

Mixed select-or-delay

Non-determinism in selective synchronizations

Conditional & timed entry-calls

Selective acceptance

Synchronization

Selective & timed entry-calls

Shared memory based synchronization

Message based synchronization

Selective & timed entry-calls

Selective acceptance

Selective & timed entry-calls

Shared memory based synchronization

Message based synchronization