Communications & Synchronization

Side effects

Operations have side effects which are visible …

- Addsem & conditional variables
- Synchronized methods
- Private locks
- Synchronization methods
- ... operations on the main memory will usually not be atomic

... yet perhaps for the actual case.

... yet perhaps the processor supplies atomic operations for the actual case.

... yet perhaps this scheduler is aware of the shared data.

... yet perhaps they are.
Towards synchronization

Distributed synchronization

Conditional Critical Regions

Basic idea:
- Critical regions are sets of statements where data is shared between processes, and which are guaranteed to be executed atomically.
- Critical regions are typically defined using condition variables or monitors.
- Critical regions are useful for coordinating access to shared resources, ensuring that only one process can access the resource at a time.
- Critical regions allow for fine-grained control of access to shared resources, making them well-suited for scalable systems.
- Critical regions are commonly used in concurrent and distributed systems, as they help to prevent data races and ensure data consistency.

Monitors

Basic idea:
- Monitors provide a way to control access to shared resources in a concurrent system.
- A monitor is a data structure that manages a set of shared variables and provides a synchronized interface to these variables.
- Monitors are typically used to control access to shared data, ensuring that only one process can access the data at a time.
- Monitors can be used to implement critical regions, as they provide a way to synchronize access to shared resources.
- Monitors are often used in operating systems to control access to shared resources, such as files and memory regions.

Semaphore thread

Basic idea:
- Semaphores are used to control access to shared resources in a concurrent system.
- Semaphores provide a way to synchronize access to shared resources, allowing multiple processes to access the resource in a controlled manner.
- Semaphores are typically implemented using monitors or other synchronization mechanisms.
- Semaphores are often used to implement critical regions, as they provide a way to control access to shared resources.
- Semaphores are commonly used in operating systems to control access to shared resources, such as files and memory regions.
Communication & Synchronization

Monitors with condition synchronization

Suggestions to overcome the multiple-tasks-in-monitor-problem:

• A signal is allowed only after the last action of a process has been the monitor.
• A signal operation has the side effect of blocking the calling process.
• Monitor, Modula-1, POSIX.

- Add operation which locks a process
  - while locked
    - protected data
    
• A signal operation which unlocks a process does not block the calling
  - but the unlocked process must have access to the monitor.

Centralized synchronization

Monitors in POSIX ('C')

Types and constants

- A signal provides a way for threads to inform each other
  - semaphore type in Modula-1
- The count of the highest active signal is incremented or decremented
- A signal operation which increments a process does not block the calling
  - but the unlocked process must have access to the monitor.

Communication & Synchronization

Monitors in Visual Basic

Monitors in Java

Java provides two mechanisms to construct a monitor-like structure:

• Synchronized methods and code blocks
  - all methods and code blocks which are using the synchronized
  - tag are mutually exclusive with respect to the addressed class.
• Notification methods
  - wait, notify, and notifyAll() can be used only in synchronized regions and are called only from threads
  - which are waiting in the same synchronized object.
Considerations:

2. Notification methods:

- `notify`: This method is used to wake up a single thread that was waiting on the monitor.
- `notifyAll`: This method is used to wake up all the threads that were waiting on the monitor.
- In order to implement a monitor all methods can break a Java monitor and enter at any time. It is impossible to analyse a Java monitor locally, since lock accesses can exist all over the system.

Synchronization by protected objects

- Information about the methods and the data they access is given in the monitor class.
- There are no explicit conditional variables associated with the monitor or data. The parent class might need to be adapted to the unjustified locked calling monitor reduces the usefulness of the calling monitor.

Methods to design and analyse expandible synchronized systems exist, yet they are complex and not offered in any concurrent programming language.

Centralized synchronization

Monitors in Java

Synchronization by protected objects

Conditions with monitors:

- Mutual exclusion is solved elegantly and safely.
- Conditional synchronization is on the level of semaphores still or all object-oriented programming and synchronization (Matsuoka & Yonezawa '93)
- A mixture of low-level and high-level synchronization constructs.

Mutual exclusion is solved elegantly and safely.

Conditional synchronization is on the level of semaphores still or all object-oriented programming and synchronization (Matsuoka & Yonezawa '93)

A mixture of low-level and high-level synchronization constructs.
Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)

Synchronization by protected objects

(Controlled synchronization: entries & barriers)
Conditional

Monitors

• Mutual exclusion in scopes
• Full implementation of the Dijkstra-/Hoare concept
• With locks to protect it
• Condition variables related to
• Monitor methods

Ada

• High-level synchronization support for tasks in large-scale programs
• Full range of resources including the CPU, main memory, etc.
• Task-based synchronization for very special cases

Ada has still no synchronization mechanisms in the usual sense (2010).

Synchronization in large scale concurrency

High Performance Computing (HPC) requires solutions for hundreds of thousands of processors.

• Basic software infrastructure needed.
• Need for scalable synchronization mechanisms.
• High degree of overlap of compute and I/O.
• Need for high-level synchronization mechanisms.

Synchronously, the synchronization mechanism using MPI.

Message Passing Interface (MPI) while implementing separate address spaces.

a) All of these address space models can be found and be implemented on a single computer, whereas a need for memory category synchronization in shared partitioned address spaces.

Synchronous message (sends imminent)

• Delay the sender process until

• Receiver becomes available

• Receive acknowledgment

Message-based synchronization

Message protocols

Asynchronous message (simulated by synchronous messages)

Introducing an intermediate process

• Intermediate needs to remove messages among processes
• Intermediate has to wait for messages to arrive

Asynchronous messages are added at the semantic level of synchronisation messaging. They are not directly available for implementation in a computer

Remote invocation (no results)

Short live form of remote invocation which does not need the results to be passed back

• Selectable lazy or actually synchronous at the lower level of the invocation.

Remote invocation (simulated by asynchronous messages)

• Sendable has synchronous messages
• Receivers are never actually synchronized

Current developments

Synchronous operation in X10

STL offers only atomic blocks in conditional and unconditional form.

• Uncondatonal atomic blocks are guaranteed to be non-blocking,
which means that they can be inserted and removed by the compiler using volatile blocks.

• Conditioned atomic blocks can be declared as a part of a conditional expression system.

• No need for volatile blocks

• Safe and correct

• Read temporaries and (first access) are now boxed (read not deleted)

• Writing temporary

• No need for volatile blocks

• Writing temporary

• No need for volatile blocks

• Reading temporary

Meanwhile, there are still enthusiastic — others (still) hope for a

• Stronger shared memory synchronization

• Asynchronous message

No buffer for short duration

• No buffer

• No buffer

• No buffer

In summary

Remote invocation

• Local case

• Interprocess communication

• Validate case

• Interprocess communication

• Validate case

In summary

Remote invocation

• Local case

• Interprocess communication

• Validate case

• Interprocess communication

• Validate case

In summary

Remote invocation

• Local case

• Interprocess communication

• Validate case

• Interprocess communication

• Validate case

In summary
Message-based synchronization

Synchronous vs. asynchronous communications

- Synchronous messages / remote invocations
- Asynchronous messages

Synchronous message passing in distributed systems requires hardware support.

One-to-all broadcast

• can be used by one writer and one reader process only
• and is synchronous:
  • can be employed then:
    - send SensorChannel (reading)
    - receive case SignalSensorChannel = (int) to consumertype;
  • as time consuming (monitors, buffered message passing, synchronous channels)

Communication & Synchronization

Message structure (Ada)

-- generate reading

procedure Read (Stream : in out Stream; buf : in out Buffer) return int;

conversion routines

-- not specified by the language

Java:

- no message passing system defined

Essential Occam2 keywords

- DATA TYPE RECORD
- ANY CHAN OF
- ALT PAR SEQ
- PRI

Communication & Synchronization

Communication is measured by means of a channel, which:

- can be read by one writer and one reader process only
- and is symmetrical

Communication is measured by means of a channel, which:

- can be used by one writer and one reader process only

Message-based synchronization in Occam2

Communication is measured by means of a channel, which:

- can be used by one writer and one reader process only

Message-based synchronization in Ada

Ada supports remote invocations (remote procedure invocation)

- nets to point-to-point
- set of protocol profiles supported

Hardware and the remote invocations are split into two architectures, on one hand the initial profile and on the other hand the communication components are taken care of by the communication components.

Synchronous

• Both tasks are synchronized at the beginning of the remote invocation on ‘begin’.

• The calling task (task A) has to wait until the remote node is completed: `end begin`
Message-based synchronization in Ada

Message-based synchronization

accept <entry_name> [(index)] <parameter_profile>;

------ waiting for synchronization
------ waiting for synchronization
------ waiting for synchronization
synchronized

Some things to consider for task-entries:

• In contrast to protected-object-entries, task-entry bodies can call other blocking operations.
• Accept statements can be nested (but need to be different).
• Accept statements can have a dedicated exception handler (for any other code-block).
• Parameters cannot be 'access' parameters, but can be access-types.
• 'count' on task-entries is defined, but is only accessible from inside the tasks which owns the entry.
• Entry characters can be 'private' or 'public'.
• Private names accessible to internal tasks are supported.

Summary

Shared memory based synchronization
• Flags, condition variables, semaphores, monitors, protected objects.
• Guard evaluation times, nested monitor calls, deadlocks, simultaneous reading, queue management.
• Synchronization and object orientation, blocking operations and re-queuing.

Message based synchronization
• Synchronization models
• Addressing methods
• Message structures
• Examples