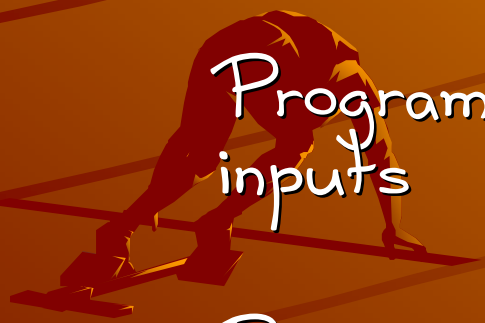
Ideas from 'Distributed Processes'

CS 451 Lecture 2003

Originally proposed for Real Time programs

- Properties of Real time environments:

  Things happen simultaneously and at fast speeds



  Program needs to respond to nondeterministic inputs

  Program never terminates but continues to work as long as the environment works

Process

- Local variables
- Common Procedures
- Initial Statement

- Initial statement executes until it terminates or waits for a condition to become true
- External requests are interleaved with initial statement
- Process continues to exist even if the initial statement terminates

Interleaving within a process

- Controlled not by system clock
- But by the program behavior
- A process switches from one operation to another when:
  - An operation terminates
  - or
  - When it waits for a condition in a guarded region

Procedures

- Specifying:

  Proc aprocedure (input params,
  output params)

  - Local variables
  - Procedure code

- Calling them:

  - Call P.aprocedure (....)

Nondeterminism

- Guarded Region
  - It can delay an operation
- Guarded Command
  - It cannot delay an operation
  - Makes an arbitrary choice from among many
    - (compare with alternative command in CSP)

Guarded Commands

- If B1:S1 | B2:S2| ... end

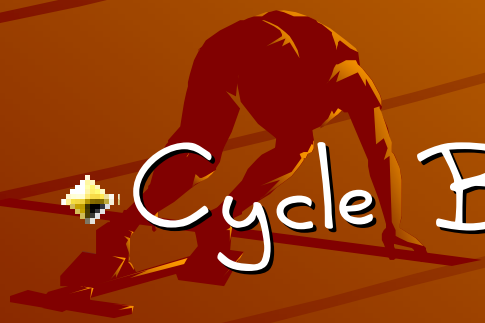
  If any one of boolean conditions B1,B2.. are true, select one of the true Bi's and execute Si.
  Else stop the program

- Do B1:S1 | B2:S2 | ... end

  While some of the Bi's are true..

Guarded Regions

- When B1:S1 | B2:S2| ... end
  Wait until one of them becomes true



- Cycle B1:S1 | B2:S2 | ... end

Endless repetition of when

Example 1: Implement monitor using DP constructs

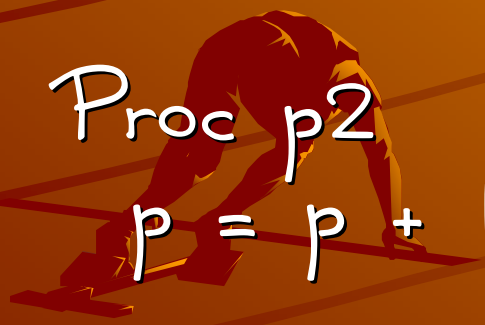
- Solution 1: using counts
- Solution 2: just a Boolean variable

Example 2

```
Process xy2
Int p
Proc p1
  when p > 0 :  p = p-1  end

Proc p2
  p = p + 1

P=0;
```



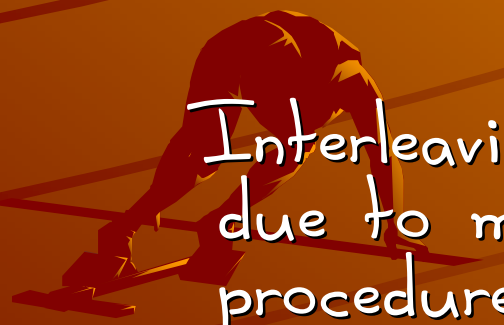
Use in other processes:
call xy2.p1
or
Call xy2.p2

What was Example 2?

A Semaphore implemented as a Distributed process

Discussions

CSP needs to identify sender explicitly (in a blocking input statement such as X?i) whereas in DP, servers don't identify their clients



Interleaving within a Distributed process is due to monitor type of switching between procedures/init statement and not because of context switches.

Discussions

One Distributed Process is not to be used to model parallel activities, as concurrency within a process follows monitor interleaving. It is to be used as a monitor.

Whereas, many distributed processes can execute in parallel with each other, i.e. they can be either located on different machines in a network, or if they are located on a single processor, they are interleaved due to context switching of the scheduler.

We noted that a semaphore can be implemented as a distributed process. Can you compare semaphores with distributed processes? The former is a synchronization construct, whereas the latter is a full programming language for distributed computing

Attempting parallel code within 1 distributed process

```
Process parallel
  proc producer
      cycle true: producer code
  proc consumer
      cycle true: consumer code
..end

Process dummy1 true:call parallel.producer
Process dummy2 true:call parallel.consumer
```

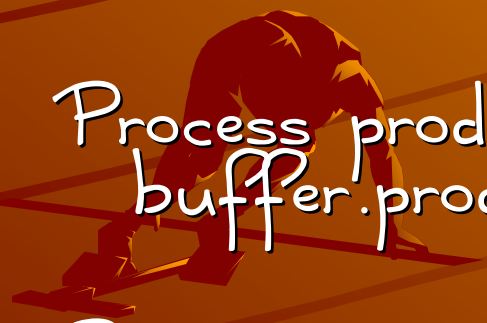
Correcting: writing parallel code with distributed process

```
Process buffer
  proc produce


  proc consume
..end

Process producer cycle true:.....; call
  buffer.produce(item);

Process consumer cycle true:...;call buffer.consume
```



Summary of Distributed process paradigm

- One process contains local variables, procedure entries, and init statement
- Procedure entries follow monitor type interleaving
- Procedure entries may block in guarded regions
- Init statement is a procedure entry that is started only once: as soon as a process is instantiated
- Distributed processes never stop. They are ready to serve incoming requests as long as the environment needs them
- Many process can run in parallel with each other on different processors, or concurrently on a single processor

Note:
There is a n(ot so) old book on operating systems, which explains the whole kernel as a set of monitors (It has a preface by either hansen or hoare. (chek out a copy in the library)

In fact, set of 'monitors' is indeed a fitting abstraction to explain most of what goes on in the kernel.

Readings

- Per Brinch Hansen, Distributed processes: A concurrent programming concept, CACM, November 1978