# Communicating Sequential Processes

- C.A.R. Hoare

Radhey Shah



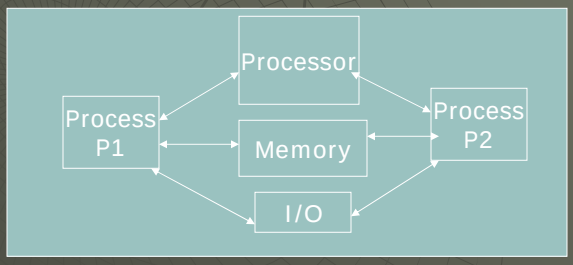
## Outline

- Introduction
- What is CSP?
- Syntax used for CSP
- Non-determinism
- Examples
- Producer-Consumer Problem
- Conclusion

# Introduction

- Traditional model
  - Monoprocessor
  - deterministic execution
  - processes communicate via

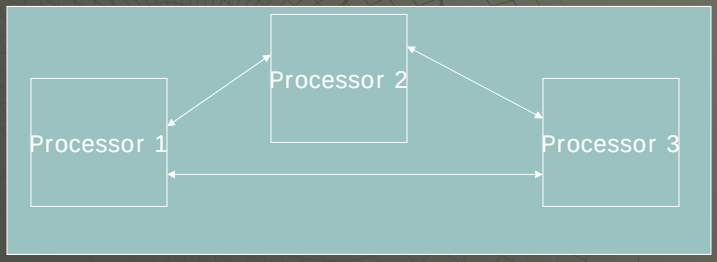
READ : Receive WRITE : SEND






# Introduction

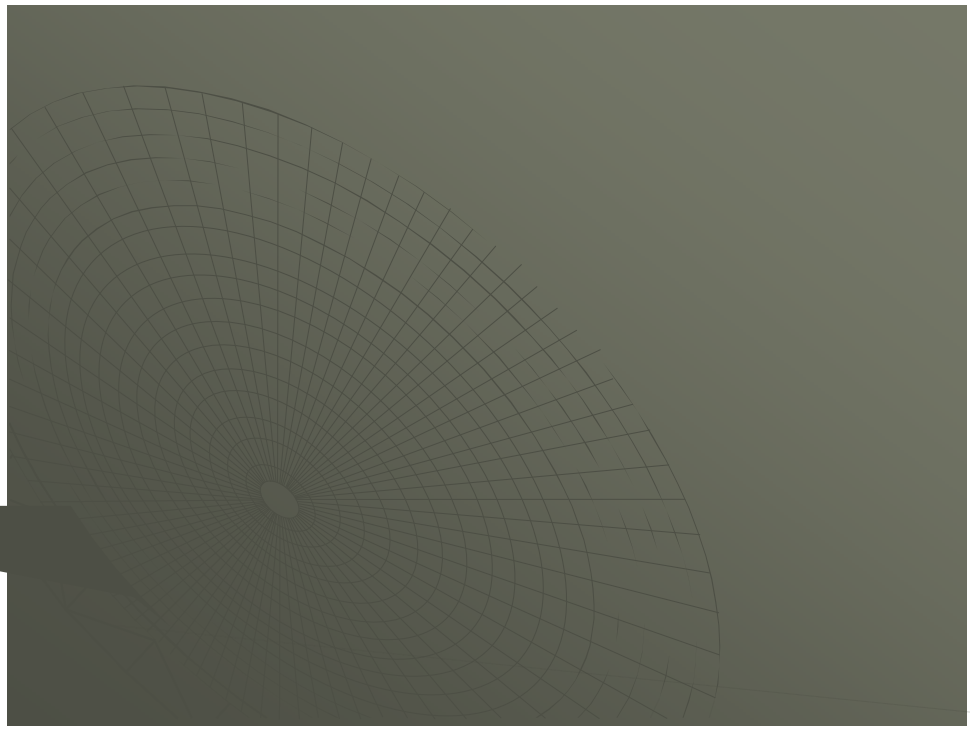
- Multi-processor machine:
  - collection of similar monoprocessors
  - each one has its own memory
  - powerful, reliable, economical



# What is CSP?

- Communicating Sequential Processes
- A programming language model to describe parallel programs for communication & synchronization between processes in multiprocessor system
- Program : a network of processes, which are connected using channels
- A channel is a point-to-point, uni-directional, synchronous unbuffered communications link
- occam supports the rules of CSP 10/11/200410/11/2004COP661





## Basic constructs

- A repetitive construct <while loop>
- An alternative construct <if..then..else>
- Normal sequential program composition<denoted by semicolon>



## CSP Notations

| Notation | meaning | Notation | meaning |
|---|---|---|---|
| ::= | is defined as | \| | or |
| < > | category names | { } | repetition |
| := | Assignment | \|\| | Parallel processes |
| ? | Input | ! | Output |
| □ | Mutual separator | → | then |
| [ ] | bracket program structure | * | repetitive command |
| ; | Separate sequence | | |

## Occam Syntax

- Sequential execution:

```
SEQ
    P1
    P2
    P3
```

- Parallel (Concurrent) execution

```
PAR
    P1
    P2
    P3
```



## Occam Syntax

- Input

```
Chan1 ? X
```

- Output

```
Chan2 ! [y FROM 0 FOR 100 ]
```

# Occam Syntax

- Selection:

```
ALT
    chan1?x
        P1
    (y>z) & chan3 ? R
        P2
ALT
  chan1?x
        P1
Clock ? AFTER (Now PLUS TenSeconds)
        P2
```



# Guarded commands

- A Boolean expression followed by a statement list. The statement list is executed only when the Boolean expression is true.
- <guarded command> ::= <guard> → <guarded list>
- <guard> ::= <Boolean expression>
- <guarded list> :: = <statement> { ; <statement> }
- <guarded command set> ::= <guarded command> { □<guarded command> }
- <alternative construct> :: =
  **if** <guarded command set> **fi**
- <repetitive construct> :: =
  **do** <guarded command set> **od**
- <statement> ::= <alternative construct> | <repetitive construct> | "other statements"

## Non-determinism

- guarded commands: Introduce & control non-determinism
- Constructs for which at least the activity evoked, but possibly even the final state, is not necessarily uniquely determined by the initial state.
- allows to map otherwise different programs on the same program text
- If..fi & do..od statements support non-determinacy



## Nondeterminacy

- if x>=y → m := x
  - y >=x → m := y

  fi
- q1,q2,q3,q4 := Q1,Q2,Q3,Q4;

  do q1>q2 → q1,q2:=q2,q1

  q2>q3 → q2,q3:=q3,q2

  q3>q4 → q3,q4:=q4,q3

  od

## Parallel commands

- <command> ::= <simple command>|<structured command>
- <parallel command> ::=

[<process> {||<process>} ]

- <process> ::=

<process label> <command list>

- Each process of a parallel command must be disjoint from every other process of command
- [ west :: DISASSEMBLE || X :: SQUASH]



## Assignment Command

## Input Command

- <input command> ::= <source>?<target variable>
- <source>::=<process name>

- cardreader?cardimage
- X?(x,y)
- console(i)?c
- X(i)?V()



## Output Command

- <output command> ::= <destination>!<expression>
- <destination>::=<process name>

- lineprinter!lineimage
- DIV!(3*a+b,13)
- console(j-1)!”A”
- sem!P()

## Alternative command

- <alternative command> ::= [<guarded command>{□<guarded command>} ]
- Specifies execution of exactly one command
- If all guards fail, alternative command fails
- [x>=y →m:=x □ y>=x → m:=y]



## Repetitive Command

- <repetitive command> ::=*<alternative command>
- Specifies as many iterations as possible of its constituent alternative command.
- When all guard fail, terminate with no effect
- i:=0;
  *[i<size;content(i)!=n→i:=i+1]

## Combination of commands

- *[X?V() → val:=val+1
  - val>0; Y?P() → val :=val-1]



## Co-routines

- COPY
  X:: *[c:character; west?c→east!c]
- SQUASH
  X::*[c:character;west?c→
  [c!=asterisk→east!c
  - c=asterisk→west?c
    [c!=asterisk→east!asterisk;east!c
  - c=asterisk →east!upward arrow
  ] ] ]

## Co-routines

- DISASSEMBLE

```
*[cardimage:(1..80)character;
  cardfile?cardimage→i:integer;i:=1;
  *[i<=80→X!cardimage(i); i:=i+1]
   X!space ]
```



## Co-routines

- ASSEMBLE

```
lineimage:(1..125)character;
i:integer;i:=1;
*[c:character; X?c
        lineimage(i):=c;
        [i<=124→i:=i+1
      □ i=125→lineprinter!lineimage;i:=1
        ] ];
[i=1→skip □ i>1→*[i<=125→lineimage(i):=space;
  i:=i+1]; lineprinter!lineimage ]
```

## Reformat

- [west::DISASSEMBLE || X::COPY ||east :: ASSEMBLE]
- [west::DISASSEMBLE || X::SQUASH || east:: ASSEMBLE]

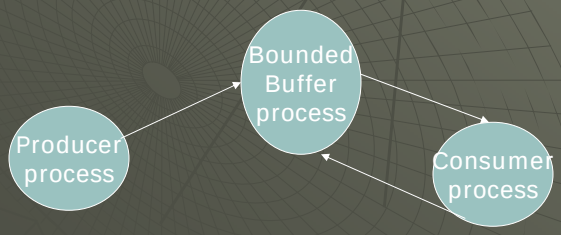


## Recursion- Factorial

```
[fac(i:1..limit)::
 *[n:integer;fac(i-1)?n→
        [n=0→fac(i-1)!]
  □ n>0→fac(i+1)!n-1;
   r:integer;fac(i+1)?r;
   fac(i-1)!(n*r)
 ] ]
|| fac(0) :: :USER ]
```

## Producer Consumer Problem

- Needs bounded buffer
- In CSP, channel is an un-buffered communication link
- Create a “buffering process”






## Producer-Consumer problem

- X : A bounded-buffer process
- Producer produces : X!p
- Consumer contains two commands:
  - X!more() ➔ Consumer is ready
  - X?p ➔ Consume

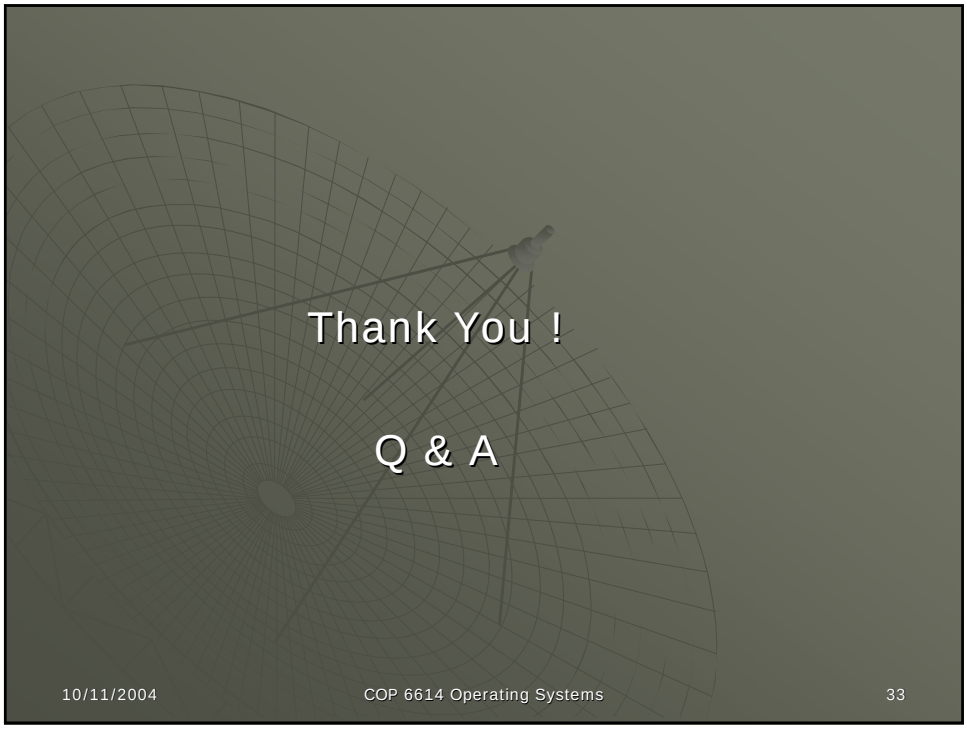
## Conclusion

- CSP gives a language structure to achieve concurrency & synchronization in multi-processor system
- input, output and concurrency constructs are as important as basic constructs
- Introduces distributed computing



## References


- Hoare, C.A.R., J. ”Communicating Sequential Processes”, Communications of the ACM, August 1978, pp. 666-677
- Dijkstra, E.W. “Guarded commands, nondeterminacy, and formal derivation of programs.” *Comm. A CM 18,* 8 (Aug. 1975), 453-457.
- http://frmb.org/occtutor.html
- http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html


Thank You !
Q & A
10/11/2004
COP 6614 Operating Systems
33