

Mohamed Boudiaf
University
-M'sila-



Faculty of Mathematics
and Computer Science



Computer Science
Department

OPERATING SYSTEMS II

CHAPTER II PROCESS SYNCHRONIZATION

CRITICAL REGIONS & PATH EXPRESSIONS

3rd Year, Computer Science - SI

Dr. Hanene CHERAIT

2025-2026

SYNCHRONISATION

Events

Locks

Semaphores

Monitors

Critical Regions

Path Expressions

CRITICAL REGION

- ◉ The concept of a **conditional critical region** provides a structured notation for describing synchronization.
- ◉ Shared variables are placed into groups called **Resources**.
- ◉ Each shared variable belongs to **at most one resource** and can be accessed only within a region of the program of a process called a *conditional critical region*.
- ◉ The term **conditional** comes from the fact that synchronization is based on the evaluation of explicit conditions within the critical regions.

DECLARATION OF CRITICAL REGIONS

- ⊙ A resource **R** can be declared as follows:
 - **Resource R: X1, X2, ..., Xi;**
 - ⦿ X1, X2, ..., Xi form a group of variables that make up the shared resource.
- ⊙ A conditional critical region is declared as follows:
 - **Region R When condition Do instructions**
- ⊙ The execution of the instructions of a **CCR** can occur only if the condition is **true**.
- ⊙ The evaluation of the condition and the execution of the instructions in a **CCR** are **indivisible**.

EXECUTION BEHAVIOR

- ◉ Before a process can execute the instructions within the critical region, it must first test the value of the condition.
 - If the condition is **true**, it executes the instructions **without any interruption**.
 - Otherwise (if the region is being used by another process), it enters a **blocked state** until it is **awakened by another process**.

THE PRODUCER/CONSUMER PROBLEM

Program ProducersConsumers;

Const N = ...;

Type item = ...;

Var buff: **Array**[0..N-1] **of** item;

Counter: 0..N; **In**: 0..N-1; **Out**: 0..N-1;

Resource Buffer: buff; counter; **In**; **Out**;

Process Producer-I;

Var producedItem: item;

Begin

Repeat

Produce(producedItem);

Region Buffer **when** counter < N

Do

Begin

buff[In] := producedItem;

In := (In + 1) mod N;

Counter := Counter + 1;

End;

Until false;

End;

Process Consumer-J;

Var consumedItem: item;

Begin

Repeat

Region Buffer **when** counter > 0

Do

Begin

consumedItem := buff[Out];

Out := (Out + 1) mod N;

Counter := Counter - 1;

End;

Consume(consumedItem);

Until false;

End;

SYNCHRONISATION

Events

Locks

Semaphores

Monitors

Critical Regions

Path Expressions

PATH EXPRESSIONS

- ◎ Path expressions specify the **activation mode** of a module's procedures that are subject to concurrent access (for example, a Monitor).
- ◎ In this approach, the implementation of procedures is separated from execution constraints.
- ◎ In a language that supports path expressions (for example, **Path Pascal**), resources are managed by modules, such as monitors:
 - A module contains **permanent variables** representing the state of the resource.
 - The **procedures** that perform operations on the resource.
 - **Path expressions** are included in the headers of these modules.

DECLARATION OF PATH EXPRESSIONS

PATH <set> End;

<set> consists of the operations and the operators that define the path.

Operator	Designation
,	The comma denotes concurrent execution of operations.
;	The semicolon denotes sequential execution of operations.
N:(set)	The N and parentheses denote N concurrent executions of the set.
[set]	The brackets denote an unlimited number of concurrent executions of the set.

EXECUTION SCENARIO EXAMPLES

- ◎ **PATH deposit, withdraw END;**
 - Execute the two functions deposit and withdraw concurrently.
- ◎ **PATH deposit ; withdraw END;**
 - Deposit an item and then withdraw it sequentially.
- ◎ **PATH 3:(deposit , withdraw) END;**
 - Repeat the sequence of deposit and withdraw three times concurrently.
- ◎ **PATH [deposit], [withdraw] END;**
 - Deposit and withdraw items an unlimited number of times concurrently.

THE PRODUCER/CONSUMER PROBLEM

THE PRODUCER/CONSUMER PROBLEM

- ◉ **Definition of a module containing the following variables:**
 - Buffer
 - In
 - Out
 - Counter (counts the number of items in the buffer)
- ◉ **Procedures:**
 - deposit
 - withdraw
- ◉ **Path expression:**
 - **PATH N: (1: (deposit); 1: (withdraw)) END;**

It indicates that the number of activations of deposit followed by withdraw **never exceeds N**

It indicates that only **one activation of withdraw** is allowed at a time—**only one withdrawal at a time**. This ensures **mutual exclusion** for access to the buffer.

PATH **N**: (**1**: (deposit); **1**: (withdraw)) END;

It indicates that only **one activation of deposit** is allowed at a given time. This ensures **mutual exclusion** for access to the buffer.

It indicates that each activation of withdraw is **preceded by a completed activation of deposit**

Program ProducersConsumers;

Module LimitedBuffer;

PATH N: (1: (deposit); 1: (withdraw)) END;

Const N = ...;

Type item = ...;

Var

buffer: **Array**[0..N-1] of item;

Counter: 0..N;

In: 0..N-1;

Out: 0..N-1;

Procedure deposit(l: item);

Begin

End;

Procedure withdraw(var l: item);

Begin

End;

Begin

Counter := 0;

In := 0;

Out := 0;

End;

Process Producer-I;

Var producedItem: item;

Begin

Repeat

Produce(producedItem);

LimitedBuffer.deposit(producedItem);

Until false;

End;

Process Consumer-J;

Var consumedItem: item;

Begin

Repeat

LimitedBuffer.withdraw(consumedItem);

Consume(consumedItem);

Until false;

End;

THANK YOU FOR YOUR
ATTENTION



NEXT CHAPTER

INTER-PROCESS COMMUNICATION (IPC)