Una clasificación sencilla

- **Memoria Común (Programación Concurrente)**
- **Sistemas Distribuidos (Programación Distribuida)**
  - **Síncronos**
  - **Asíncronos**

Sistemas Paralelos (Programación Paralela)

Modelo básico de Concurrencia

Competencia. Cuando dos procesos compitan por el mismo recurso. Los recursos pueden ser un fichero, una variable en memoria o un canal de comunicación.

Comunicación. Dos procesos pueden necesitar comunicarse para pasarse información y así poder continuar con su ejecución.

Premisas del Modelo Básico

- Interleaving (entremezclado)
- Acciones Atómicas

Interleaving

Dos procesos

$$P: \ a \ \rightarrow \ b.$$
$$Q: \ r \ \rightarrow \ s \ \rightarrow \ t.$$

Una Ejecución entremezclada

$$a \ \rightarrow \ r \ \rightarrow \ s \ \rightarrow \ b \ \rightarrow \ t$$

Ventajas de este modelo

Sencillez. Para muchos problemas los condicionamientos de tiempo absoluto no son necesarios y solo suponen una complejidad añadida.

Generalidad. Los sistemas estan continuamente siendo actualizados con nuevos componentes más rápidos y con algoritmos más eficientes.

Adaptación. Incluso dentro de este modelo, el tiempo puede ser *simulado*.

Interleaving

Posibles ejecuciones en *interleaving* del programa compuesto por los procesos P y Q.

$$
\begin{array}{ccccccccc}
a & \rightarrow & b & \rightarrow & r & \rightarrow & s & \rightarrow & t \\
a & \rightarrow & r & \rightarrow & b & \rightarrow & s & \rightarrow & t \\
a & \rightarrow & r & \rightarrow & s & \rightarrow & b & \rightarrow & t \\
a & \rightarrow & r & \rightarrow & s & \rightarrow & t & \rightarrow & b \\
r & \rightarrow & a & \rightarrow & b & \rightarrow & s & \rightarrow & t \\
r & \rightarrow & a & \rightarrow & s & \rightarrow & b & \rightarrow & t \\
r & \rightarrow & a & \rightarrow & s & \rightarrow & t & \rightarrow & b \\
r & \rightarrow & s & \rightarrow & a & \rightarrow & b & \rightarrow & t \\
r & \rightarrow & s & \rightarrow & a & \rightarrow & t & \rightarrow & b \\
r & \rightarrow & s & \rightarrow & t & \rightarrow & a & \rightarrow & b
\end{array}
$$

Ordenación Temporal Relativa.

Ejecución Eventual.

Instrucciones Atómicas

Dos Procesos:

```
N: Integer := 0;

task body P1 is
begin
 N := N+1;
end P1;

task body P2 is
begin
 N := N+1;
end P2;
```

Posibles ejecuciones con `INC`

Proceso	Instrucción	Valor de `N`
`P1`	`INC N`	**1**
`P2`	`INC N`	**2**

Proceso	Instrucción	Valor de `N`
`P2`	`INC N`	**1**
`P1`	`INC N`	**2**

Instruciones Atómicas

Una ejecición con registros

Proceso	Instrucción	Reg. P1	Reg.de P2	N
P1	LOAD Reg, N	0	–	0
P2	LOAD Reg, N	0	0	0
P1	ADD Reg, 1	1	0	0
P2	ADD Reg, 1	1	1	0
P1	STORE Reg, N	1	1	1
P2	STORE Reg, N	1	1	1

Corrección de Programas Concurrentes

Seguridad. Las propiedades de seguridad ***(safety)*** son aquellas que ***siempre*** deben ser verificadas, sea cual sea el estado del programa.

- Exclusión Mútua.
- Ausencia de Deadlock (abrazo mortal).

Vivacidad. Las propiedades de vivacidad ***(liveness)*** son aquellas que deben verificarse en ***algún momento.***

- Inanición (starvation).
- Justicia (fairness).

Programación Concurrente en Memoria Compartida

Problema de la Exclusión Mútua.

1. **Dos o más procesos en exclusión mútua**

```
loop
  SeccionNoCritica;
  PreProtocolo;
  SeccionCritica;
  PostProtocolo;
end loop;
```

2. **Un proceso puede detenerse fuera de su sección crítica.**
3. **El programa debe carecer de deadlock.**
4. **No debe haber inanición *(starvation)*.**
5. **Dinamismo en ausencia de competencia.**

Instrucciones Usuales. Solución 1

```
Turn: Integer range 1..2 := 1;

task body P1 is
begin
  loop
    SeccionNoCritica_1;
    loop exit when Turn = 1; end loop;
    SeccionCritica_1;
    Turn := 2;
  end loop;
end P1;

task body P2 is
begin
  loop
    SeccionNoCritica_2;
    loop exit when Turn = 2; end loop;
    SeccionCritica_2;
    Turn := 1;
  end loop;
end P2;
```

Instrucciones Usuales. Solución 1

A favor:

- **Exclusión mútua.**
- **Esta solución no puede bloquearse en deadlock.**
- **No hay inanición de ninguno de los procesos.**

En contra:

- **La solución no es correcta si no existe competencia.**
- **Ambos procesos se deben ejecutar estrictamente por turno.**

Instrucciones Usuales. Solución 2

```
C1,C2: Integer range 0..1 := 1;

task body P1 is
begin
  loop
    SeccionNoCritica_1;
    loop exit when C2 = 1; end loop;
    C1 = 0;
    SeccionCritica_1;
    C1 := 1;
  end loop;
end P1;

task body P2 is
begin
  loop
    SeccionNoCritica_2;
    loop exit when C1 = 1; end loop;
    C2 = 0;
    SeccionCritica_2;
    C2 := 1;
  end loop;
end P2;
```

Instrucciones Usuales. Solución 2

A favor:

- **Dinamismo ante falta de competencia.**

En contra:

- **No hay exclusión mútua.**
  1. `P1` **comprueba la igualdad** `C2=1` **y es cierta.**
  2. `P2` **comprueba la igualdad** `C1=1` **y es cierta.**
  3. `P1` **ejecuta** `C1:=0`**.**
  4. `P2` **ejecuta** `C2:=0`**.**
  5. `P1` **entra en su sección crítica.**
  6. `P2` **entra en su sección crítica.**
- **Inanición**
  1. `P1` **esta ejecutando su sección crítica.**
  2. `P2` **comprueba la igualdad** `C1=0` **y es falsa.**
  3. `P1` **sale de la sección crítica y ejecuta** `C1:=1`**.**
  4. `P1` **entra en su sección no crítica.**
  5. `P1` **entra en su sección crítica.**
  6. `P1` **comprueba la igualdad** `C2=1` **y es cierta.**
  7. `P1` **ejecuta** `C1:=0`**.**
  8. `P2` **comprueba la igualdad** `C1=0` **y es falsa.**
  9. `P1` **entra en su sección crítica.**
  10. `P2` **comprueba la igualdad** `C1=0` **y es falsa.**

Instrucciones Usuales. Solución 3

```
C1,C2: Integer range 0..1 := 1;

task body P1 is
begin
  loop
    SeccionNoCritica_1;
    C1 := 0;
    loop
      exit when C2 = 1;
      C1 := 1;
      C1 := 0;
    end loop;
    SeccionCritica_1;
    C1 := 1;
  end loop;
end P1;

task body P2 is
begin
  loop
    SeccionNoCritica_2;
    C2 := 0;
    loop
      exit when C1 = 1; ç
      C2 := 1;
      C2 := 0;
    end loop;
    SeccionCritica_2;
    C2 := 1;
  end loop;
end P2;
```

Instrucciones Usuales. Solución 3

A favor:

- **La exclusión mutua se garantiza.**

En contra:

- **Uno de los procesos puede sufrir inanición.**
- **Inanición global, livelock.**
  1. `P1` **esta ejecutando su sección no crítica.**
  2. `P2` **esta ejecutando su sección no crítica.**
  3. `P1` **ejecuta** `C1:=0`**.**
  4. `P2` **ejecuta** `C2:=0`**.**
  5. `P1` **comprueba la igualdad** `C2=0` **y la encuentra falsa.**
  6. `P2` **comprueba la igualdad** `C1=0` **y la encuentra falsa.**
  7. `P1` **ejecuta** `C1:=1`**.**
  8. `P2` **ejecuta** `C2:=1`**.**
  9. `P1` **ejecuta** `C1:=0`**.**
  10. `P2` **ejecuta** `C2:=0`**.**
  11. `P1` **comprueba la igualdad** `C2=0` **y la encuentra falsa.**
  12. `P2` **comprueba la igualdad** `C1=0` **y la encuentra falsa.**

Algoritmo de Dekker

```
C1,C2: Integer range 0..1 := 1;
Turn:  Integer range 1..2 := 1;

task body P1 is
begin
  loop
    SeccionNoCritica_1;
    C1 = 0;
    loop
      exit when C2 = 1;
      if Turn=2 then
        C1 := 1;
        loop exit when Turn=1; end loop;
        C1 := 0;
      end if;
    end loop;
    SeccionCritica_1;
    C1 := 1;
    Turn := 2;
  end loop;
end P1;
```

Algoritmo de Dekker

A favor:

- **Se respeta la exclusión mútua.**
- **No puede haber deadlock.**
- **No puede haber inanición.**
- **Dinámico si no hay competencia.**