Lección 3: Sincronización de procesos. El problema de la sección crítica

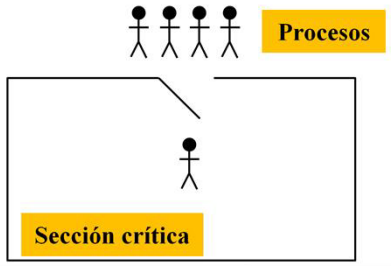
- Sincronización de procesos
- El problema de la sección crítica
- Primeros intentos de solución: propuesta de algoritmos y sus correspondientes modelos
- Algoritmos para la sección crítica:
  - Algoritmo de Dekker (2 procesos)
  - Algoritmo de la panadería
  - Algoritmo por turno de espera

Sincronización de procesos

- Objetivo de la sincronización: realización coordinada de tareas entre procesos
  - Una tarea antes que otra o a la vez
- Herramienta conceptual: instruccion ***await***
  - Muy general
  - Costosa de implementar
- Variedad de mecanismos de sincronización de procesos
  - semáforos, monitores, memoria compartida
  - Lenguajes de programación también ofrecen instrucciones de alto nivel para la sincronización (por ejemplo, "**`synchronized`**" de Java)

El problema de la sección crítica

- Enunciado del problema (Dijstra, 1976):
  - "n" procesos ejecutan repetidamente una sección no crítica, SNC, seguida de una **sección crítica**, SC
  - La sección crítica de un proceso es un secuencia de acciones que debe ser ejecutadas en **exclusión mutua** con las secciones críticas del resto de procesos.
    - Siempre que un proceso "entra" en la SC, "sale"



El problema de la sección crítica

- Esquema del algoritmo:

variables globales	
Process P	*Process Q*
variables locales	variables locales
loop forever	loop forever
SNC	SNC
Protocolo de entrada	Protocolo de entrada
SC	SC
Protocolo de salida	Protocolo de salida

El problema de la sección crítica

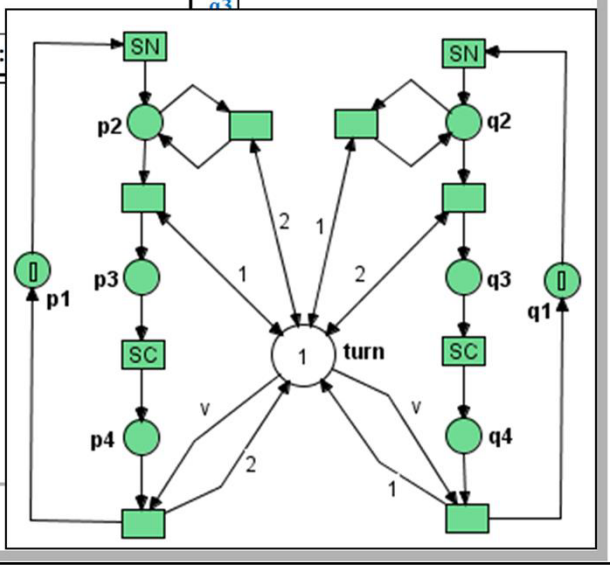
- **Objetivo**: Diseñar los protocolos de entrada y salida (mecanismos de sincronización) a la sección crítica de forma que se satisfagan los siguientes requisitos:
  - **Exclusión mutua**: como máximo un proceso puede estar ejecutando su sección crítica
  - **Ausencia de bloqueos**: si dos o más procesos tratan de acceder a su sección crítica, al menos uno lo logrará
  - **Ausencia de situaciones de inanición (individual)**: todo proceso que desee entrar en su SC, tarde o temprano entrará

Primer intento

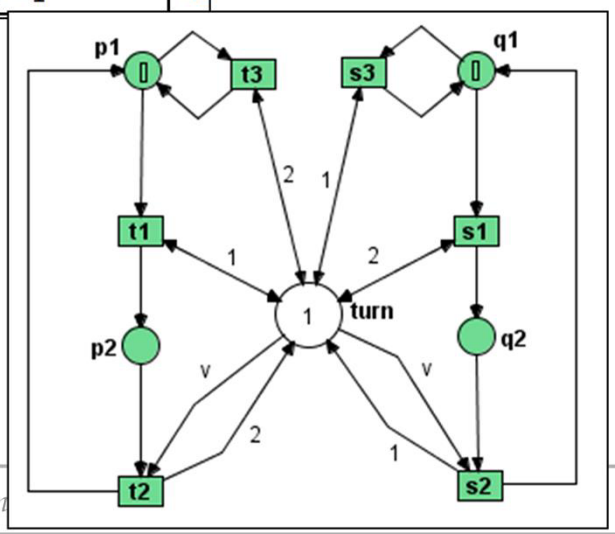
- Algoritmo:

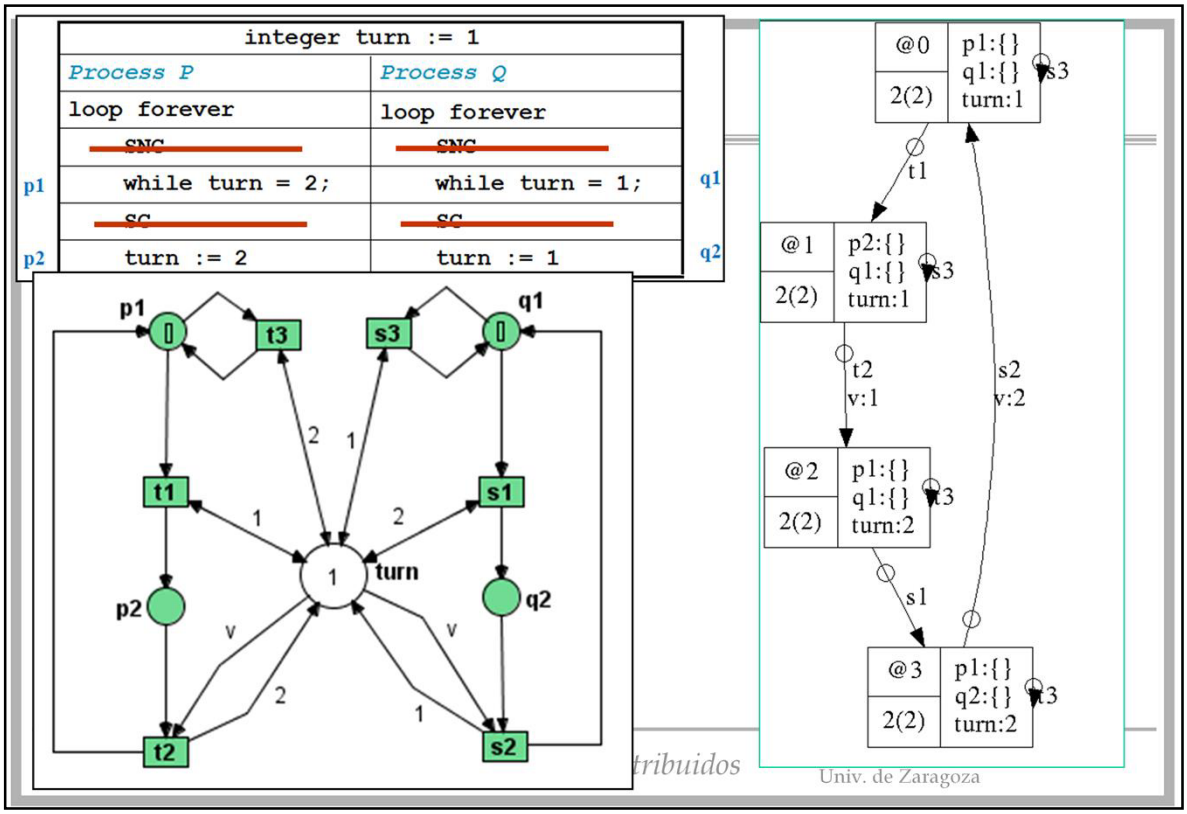
	integer turn := 1		
	Process P	*Process Q*	
	loop forever	loop forever	
p1	SNC	SNC	q1
p2	while turn = 2;	while turn = 1;	q2
p3	SC	SC	q3
p4	turn := 2	turn := 1	q4

	integer turn := 1	
	Process P	*Process Q*
	loop forever	loop forever
p1	SNC	SNC
p2	while turn = 2;	while turn = 1;
p3	SC	SC
p4	turn := 2	turn :



	integer turn := 1	
	Process P	*Process Q*
	loop forever	loop forever
	~~SNC~~	~~SNC~~
p1	while turn = 2;	while turn = 1; q1
	~~SC~~	~~SC~~
p2	turn := 2	turn := 1 q2




integer turn := 1
Process P
Process Q
loop forever
loop forever
SNC
SNC
p1
while turn = 2;
while turn = 1;
q1
SC
SC
p2
turn := 2
turn := 1
q2
p1
t3
s3
q1
t1
s1
turn
p2
q2
t2
s2
@0
p1:{}
q1:{}
turn:1
2(2)
s3
t1
@1
p2:{}
q1:{}
turn:1
2(2)
s3
t2
v:1
s2
v:2
@2
p1:{}
q1:{}
turn:2
2(2)
s3
s1
@3
p1:{}
q2:{}
turn:2
2(2)
s3

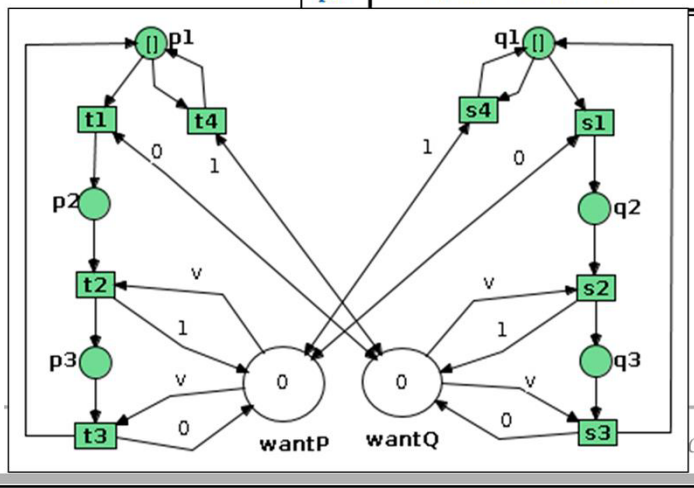
Segundo intento

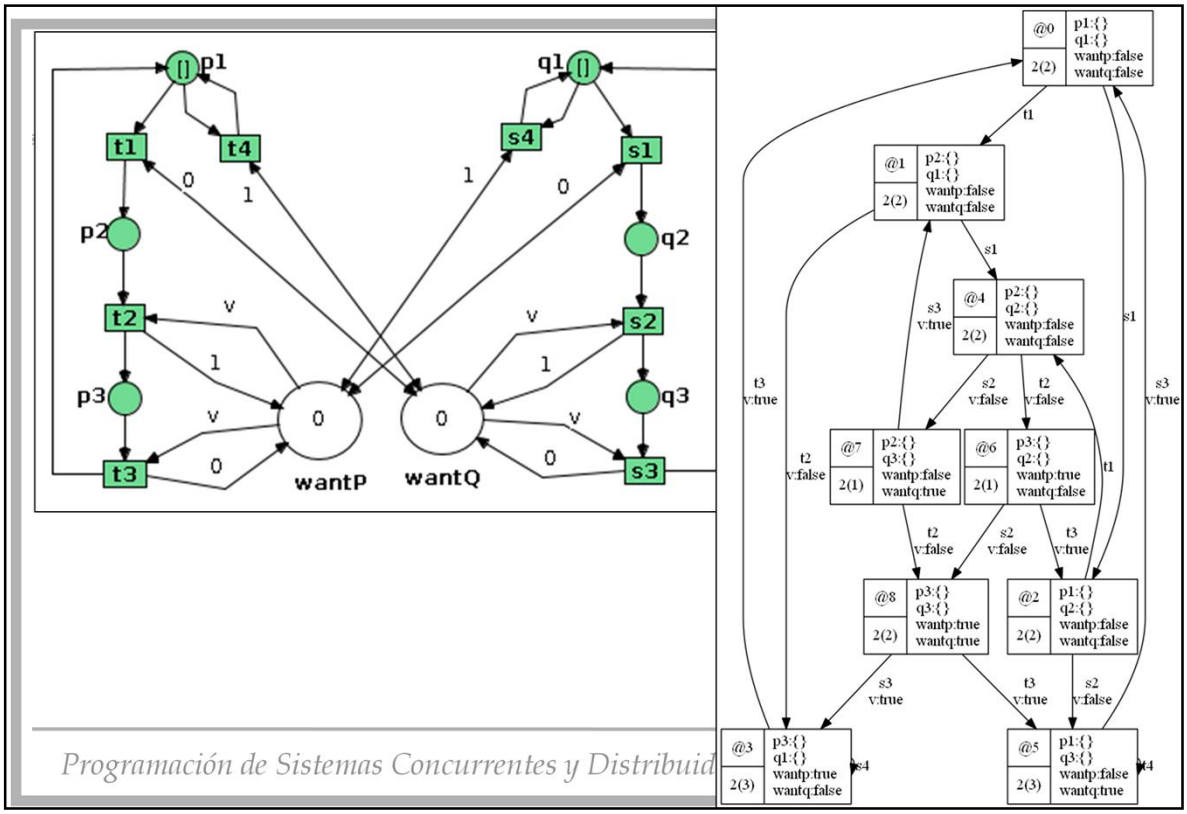
- Cada proceso controla con una variable de sincronización propia el acceso a la SC
- Algoritmo:

	boolean wantP := false, wantQ := false		
	Process P	*Process Q*	
	loop forever	loop forever	
	SNC	SNC	
p1	while wantQ = true	while wantP = true	q1
p2	wantP := true	wantQ := true	q2
	SC	SC	
p3	wantP := false	wantQ := false	q3

Modelo e his

	boolean wantP := false, wantQ := false		
	Process P	*Process Q*	
	loop forever	loop forever	
	SNC	SNC	
p1	while wantQ = true	while wantP = true	q1
p2	wantP := true	wantQ := true	q2
	SC	SC	
p3	wantP := false	wantQ := false	q3



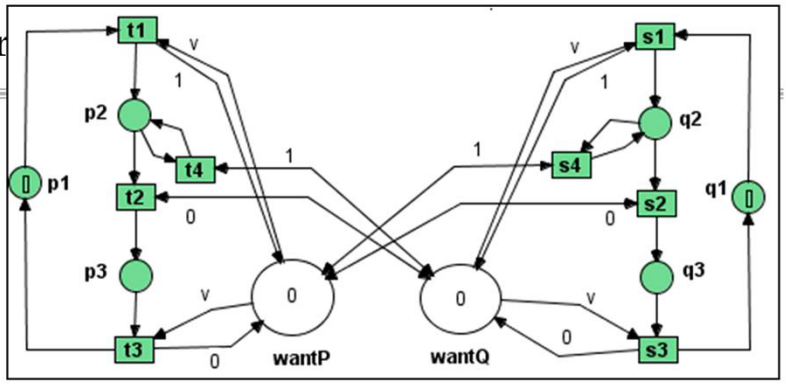

p1
q1
t1
t4
s4
s1
p2
q2
t2
s2
p3
q3
t3
s3
wantP
wantQ
@0
p1:{}
q1:{}
wantp:false
wantq:false
2(2)
@1
p2:{}
q1:{}
wantp:false
wantq:false
2(2)
@4
p2:{}
q2:{}
wantp:false
wantq:false
2(2)
@7
p2:{}
q3:{}
wantp:false
wantq:true
2(1)
@6
p3:{}
q2:{}
wantp:true
wantq:false
2(1)
@8
p3:{}
q3:{}
wantp:true
wantq:true
2(2)
@2
p1:{}
q2:{}
wantp:false
wantq:false
2(2)
@3
p3:{}
q1:{}
wantp:true
wantq:false
2(3)
@5
p1:{}
q3:{}
wantp:false
wantq:true
2(3)

Tercer intento

- Instrucción de sincronización forma parte de la SC
- Algoritmo:

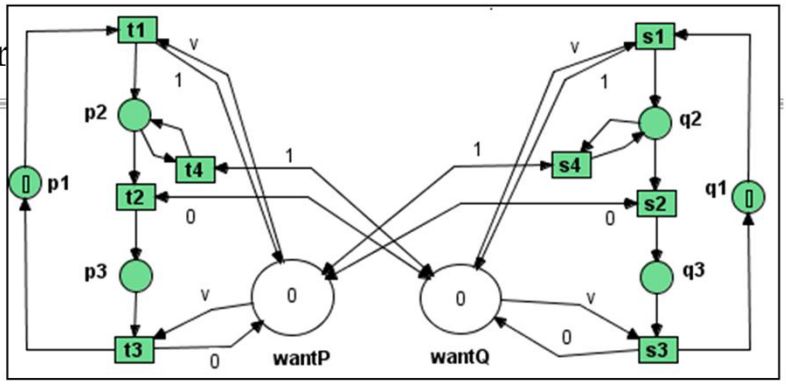
	boolean wantP := false, wantQ := false		
	Process P	*Process Q*	
	loop forever	loop forever	
	SNC	SNC	
p1	wantP := true	wantQ := true	q1
p2	while wantQ = true	while wantP = true	q2
	SC	SC	
p3	wantP := false	wantQ := false	q3

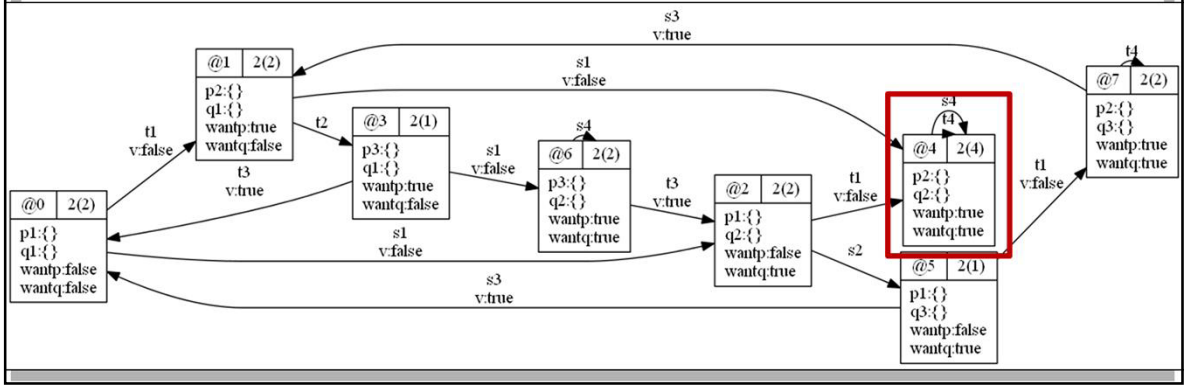
Modelo e histor



	boolean wantP := false, wantQ := false		
	Process P	*Process Q*	
	loop forever	loop forever	
	SNC	SNC	
p1	wantP := true	wantQ := true	q1
p2	while wantQ = true	while wantP = true	q2
	SC	SC	
p3	wantP := false	wantQ := false	q3

Modelo e histor



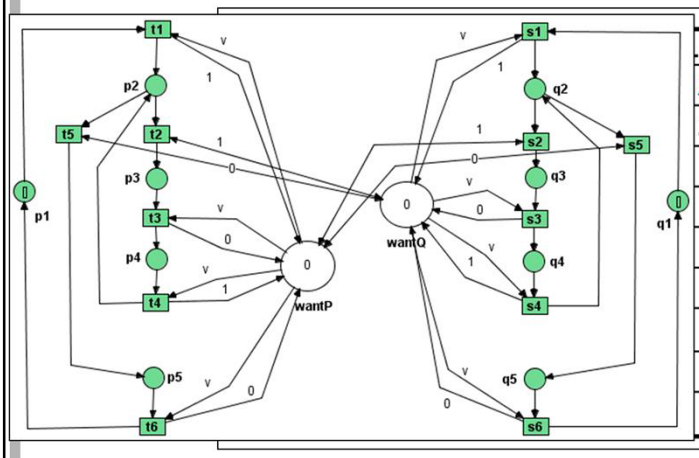


Cuarto intento

- Un proceso no debe empecinarse en entrar a su SC
- Algoritmo:

	boolean wantP := false, wantQ := false		
	Process P	*Process Q*	
	loop forever	loop forever	
	SNC	SNC	
p1	wantP := true	wantQ := true	q1
p2	while wantQ	while wantP	q2
p3	wantP := false	wantQ := false	q3
p4	wantP := true	wantQ := true	q4
	SC	SC	
p5	wantP := false	wantQ := false	q5

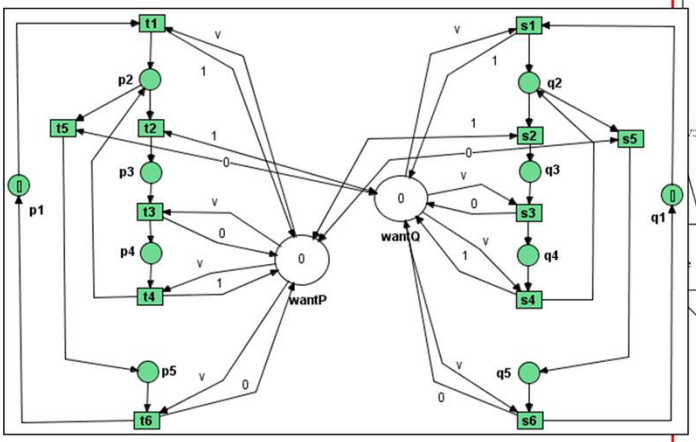
Modelo e historia de ejecución

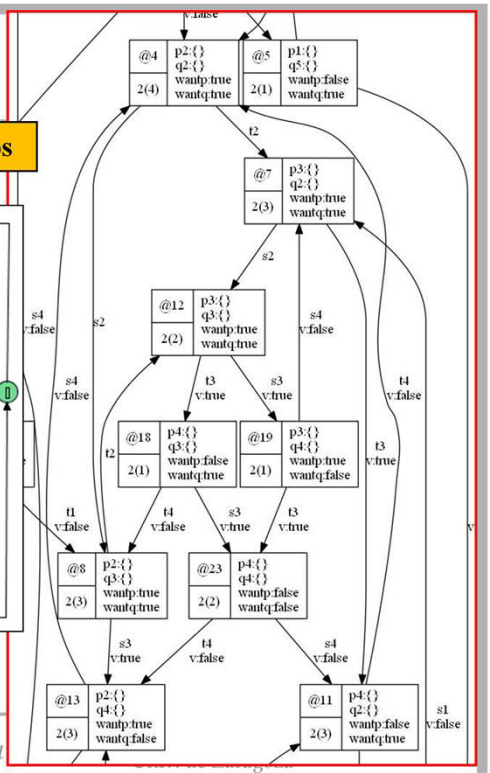



```
...se, wantQ := false
Process Q
loop forever
    SNC
    wantQ := true          q1
    while wantP            q2
        wantQ := false     q3
        wantQ := true      q4
    SC
    wantQ := false         q5
```

Modelo e historia de ejecución

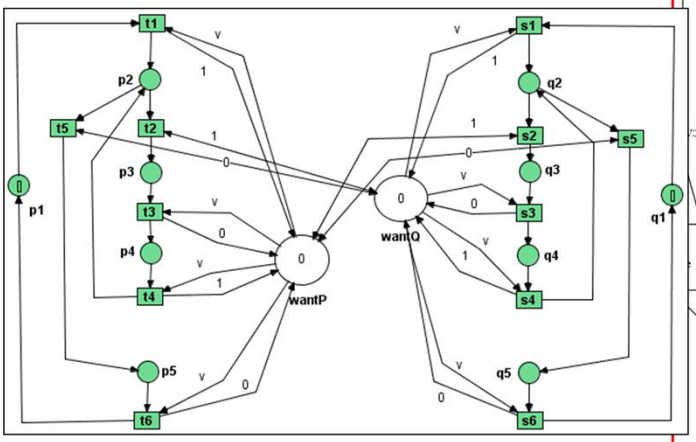
24 estados

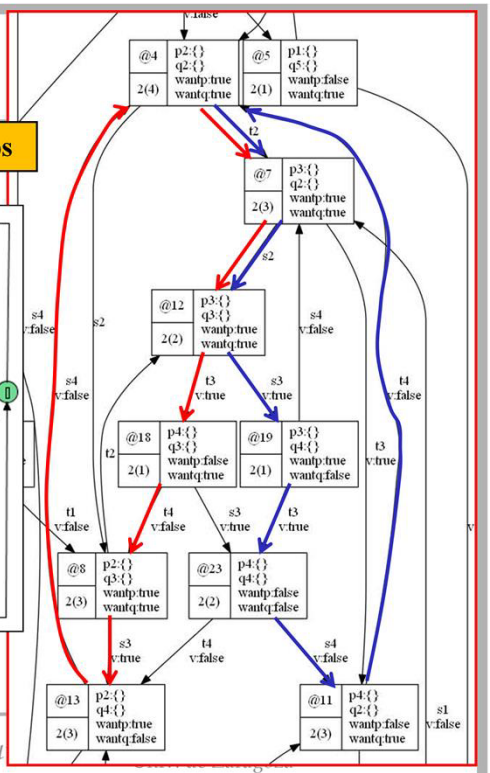




Modelo e historia de ejecución

24 estados





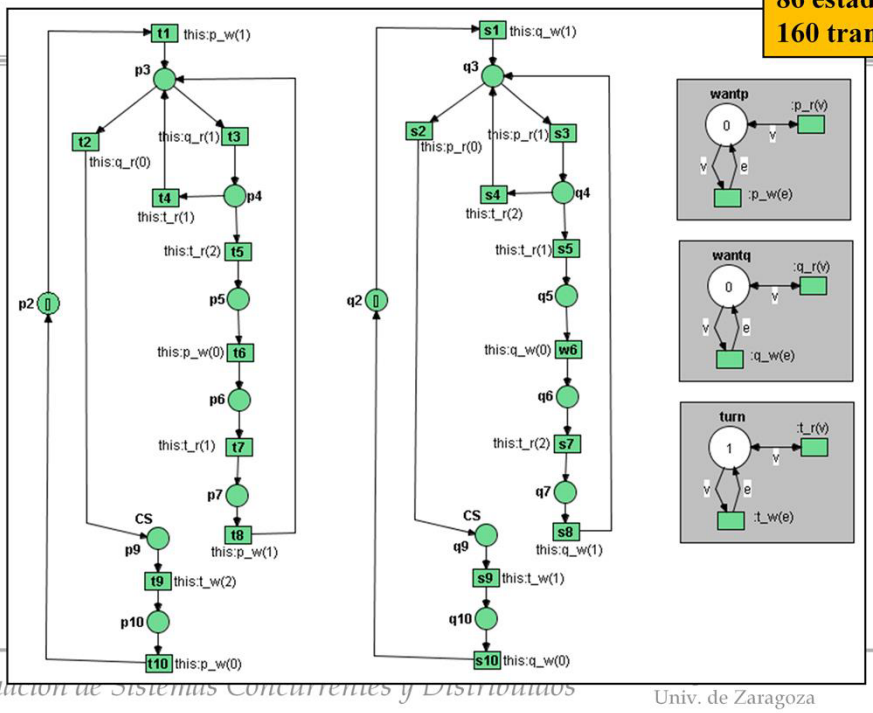
Solución al problema: Algoritmo de Dekker

- Algoritmo:

	boolean wantP := false, wantQ := false integer turn := 1		
	Process P	*Process Q*	
	loop forever	loop forever	
	SNC	SNC	
p2	wantP := true	wantQ := true	q2
p3	while wantQ	while wantP	q3
p4	if turn = 2	if turn = 1	q4
p5	wantP := false	wantQ := false	q5
p6	await turn = 1	await turn = 2	q6
p7	wantP := true	wantQ := true	q7
	SC	SC	
p9	turn := 2	turn := 1	q9
p10	wantP := false	wantQ := false	q10

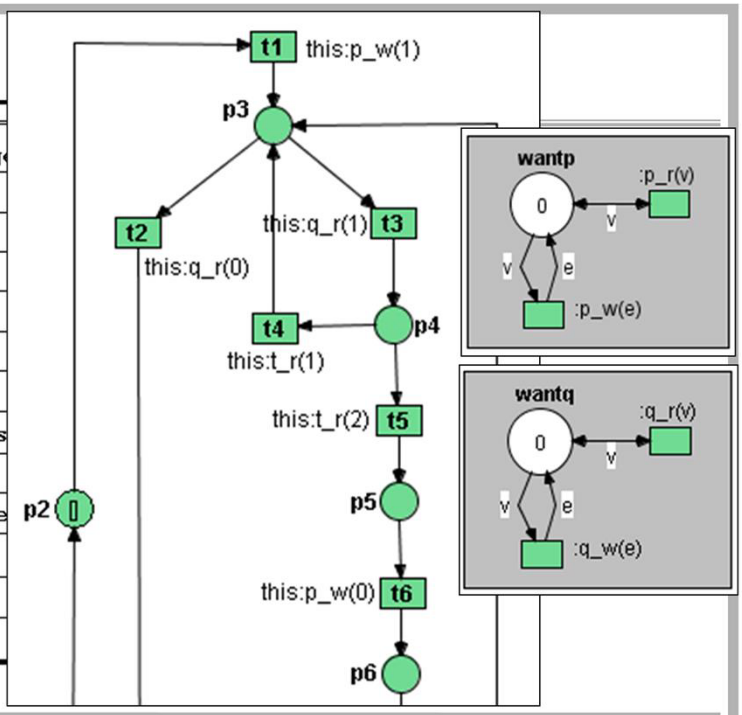
Modelo del sistema

86 estados
160 trans.



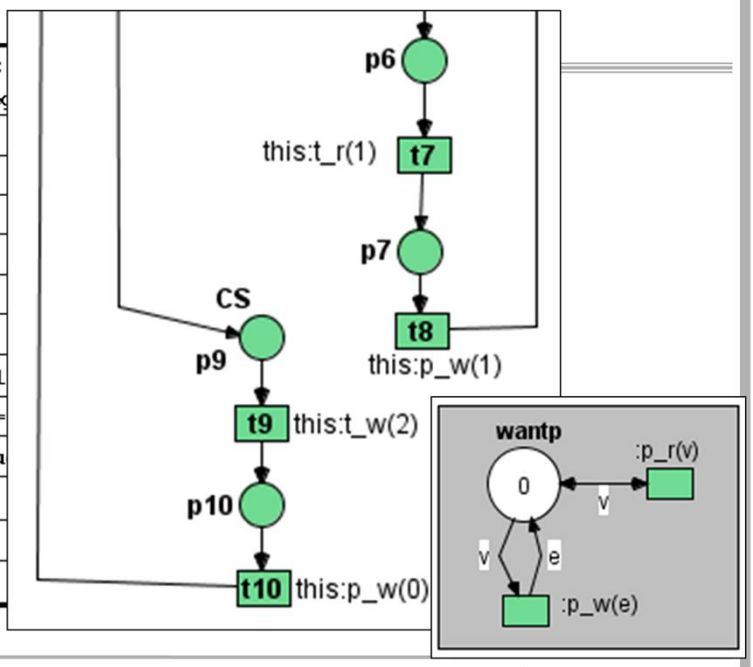
Modelo del sistema

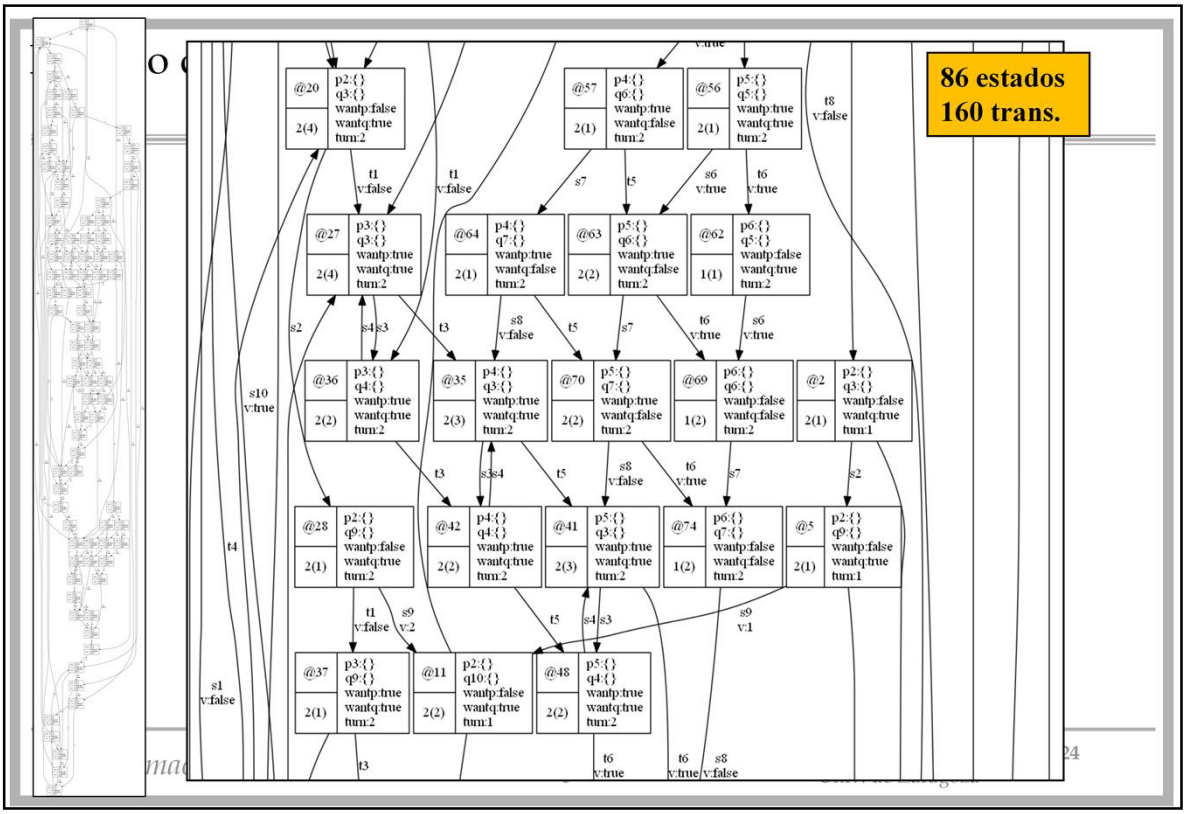
```
          boolean wantP :=
                        integ
Process P
loop forever
    SNC
p2  wantP := true
p3  while wantQ
p4      if turn = 2
p5          wantP := fals
p6          await turn =
p7          wantP := true
    SC
p9  turn := 2
p10 wantP := false
```



Modelo del sistema

```
          boolean wantP :
                        inte
Process P
loop forever
    SNC
p2  wantP := true
p3  while wantQ
p4      if turn = 2
p5          wantP := fal
p6          await turn =
p7          wantP := tru
    SC
p9  turn := 2
p10 wantP := false
```




86 estados
160 trans.
@20 p2:{} q3:{} wantp:false wantq:true turn:2 2(4)
@57 p4:{} q6:{} wantp:true wantq:false turn:2 2(1)
@56 p5:{} q5:{} wantp:true wantq:true turn:2 2(1)
@27 p3:{} q3:{} wantp:true wantq:true turn:2 2(4)
@64 p4:{} q7:{} wantp:true wantq:false turn:2 2(1)
@63 p5:{} q6:{} wantp:true wantq:false turn:2 2(2)
@62 p6:{} q5:{} wantp:false wantq:true turn:2 1(1)
@36 p3:{} q4:{} wantp:true wantq:true turn:2 2(2)
@35 p4:{} q3:{} wantp:true wantq:true turn:2 2(3)
@70 p5:{} q7:{} wantp:true wantq:false turn:2 2(2)
@69 p6:{} q6:{} wantp:false wantq:false turn:2 1(2)
@2 p2:{} q3:{} wantp:false wantq:true turn:1 2(1)
@28 p2:{} q9:{} wantp:false wantq:true turn:2 2(1)
@42 p4:{} q4:{} wantp:true wantq:true turn:2 2(2)
@41 p5:{} q3:{} wantp:true wantq:true turn:2 2(3)
@74 p6:{} q7:{} wantp:false wantq:false turn:2 1(2)
@5 p2:{} q9:{} wantp:false wantq:true turn:1 2(1)
@37 p3:{} q9:{} wantp:true wantq:true turn:2 2(1)
@11 p2:{} q10:{} wantp:false wantq:true turn:1 2(2)
@48 p5:{} q4:{} wantp:true wantq:true turn:2 2(2)
t1 v:false
t1 v:false
s7
t5
s6 v:true
t6 v:true
t8 v:false
s2
s4 s3
t3
s8 v:false
t5
s7
t6 v:true
s6 v:true
s10 v:true
t3
s3 s4
t5
s8 v:false
t6 v:true
s7
s2
t4
t1 v:false
s9 v:2
t5
s4 s3
s9 v:1
s1 v:false
t3
t6 v:true
t6 v:true
s8 v:false

Solución con herramientas más potentes

- Imaginemos una instrucción tan potente como

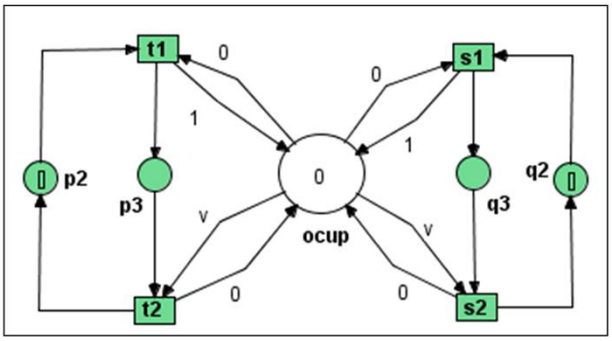
```
<await B
   S
>
```

- Podríamos hacer

	boolean ocup := false	
	Process P	*Process Q*
	loop forever	loop forever
p2	<await ocup=false	<await ocup=false	q2
	ocup := true>	ocup := true>	
	SC	SC	
p3	ocup := false	ocup := false	q3

Solución con herramientas más potentes

	boolean ocup := false		
	Process P	*Process Q*	
	loop forever	loop forever	
p2	<await ocup=false	<await ocup=false	q2
	ocup := true>	ocup := true>	
	SC	SC	
p3	ocup := false	ocup := false	q3



Solución con herramientas más potentes

`TS(comun,local):<local:=comun;comun:=true>`

- Muchos procesadores cuentan con instrucciones del tipo "test-and-set" atómico
- Y vale para n procesos

boolean ocup := false	
Process P	*Process Q*
boolean no_ent	boolean no_ent
loop forever	loop forever
SNC	SNC
TS(ocup,no_ent)	TS(ocup,no_ent)
while no_ent	while no_ent
TS(ocup,no_ent)	TS(ocup,no_ent)
SC	SC
ocup := false	ocup := false

Solución con herramientas más potentes

- Pero ¿existe eso?

```
▪ Tipo test&set
    •IBM370, M68040, VAX, [SPARC]
▪ Tipo compare & swap
    •IBM 370, Pentium
▪ Tipo swap-atomic
    •SPARC, MC88100
▪ Tipo load-locked/store-conditional
    •Alpha, MIPS (>= R4000)
▪ Tipo Fetch & add
    •CONVEX
```

Algoritmo de la Panadería (N procesos)

- Algoritmo:

```
integer array[1..N] number := (1..N,0)
Process P(i:1..N)
loop forever
    SNC
    number[i] := 1 + max(number)
    for all other processes j
        await (number[j] = 0) or (number[i] << number[j])
    SC
    number[i] := 0
```

(number[i] < number[j]) or
((number[i] = number[j]) and (i<j))

Algoritmo de turno de espera (N procesos)

- Algoritmo:

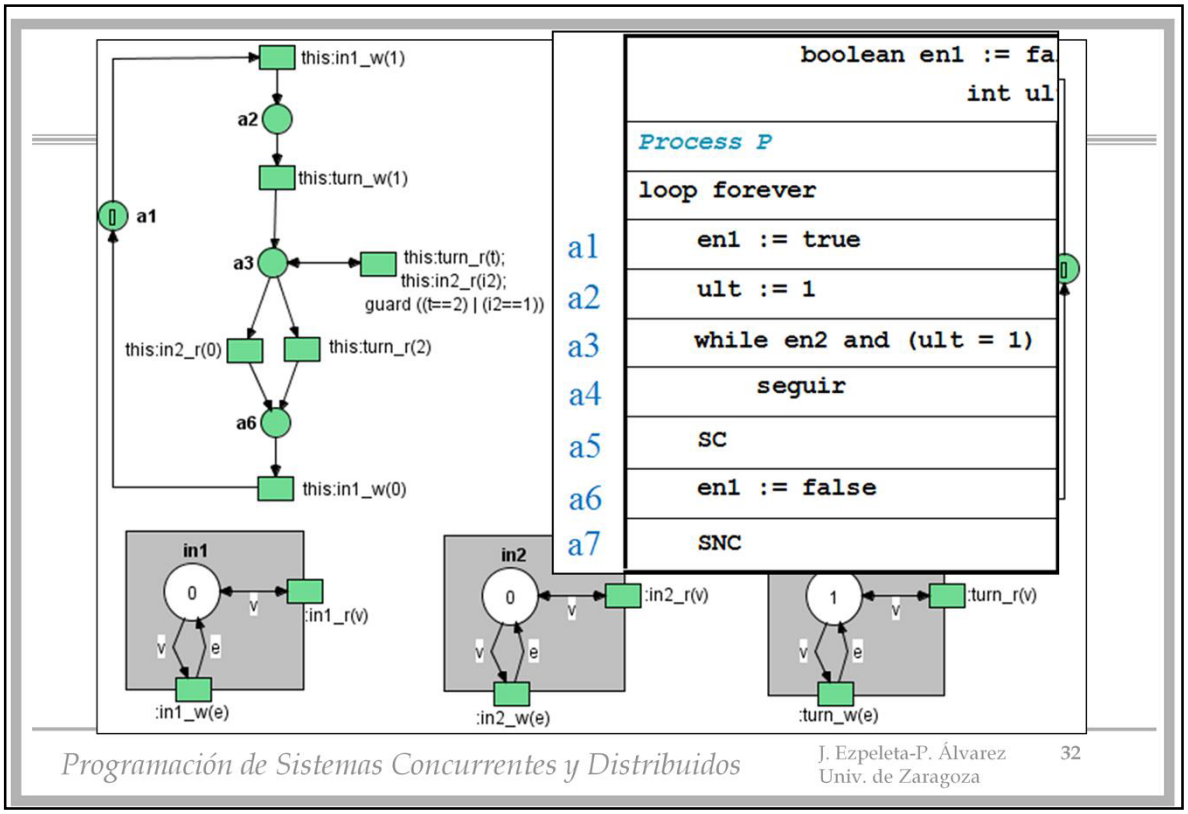
```
integer number := 0, next:= 0
integer array[1..N] turn := (1..N,0)

Process P(i:1..N)
loop forever
    turn[i] := assignTurn(number, 1)
    while turn[i] <> next
        follow
    SC
    next := next + 1
    SNC
```

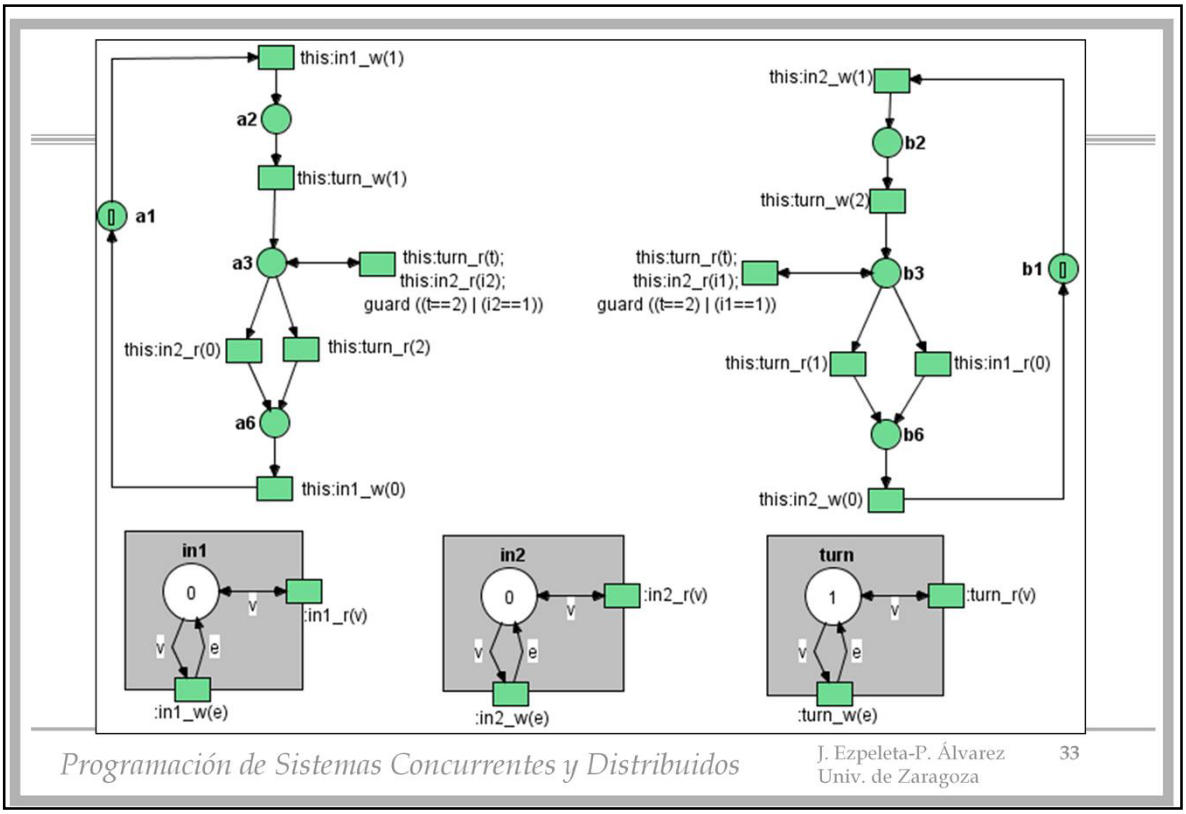
Cada proceso turno único

```
assignTurn (value, increment) {
    temp:= value
    value := value + increment
    return (temp) }
```

El algoritmo de Peterson

	boolean en1 := false, en2 := false int ult := 1	
	Process P	*Process Q*
	loop forever	loop forever
a1	en1 := true	en2 := true
a2	ult := 1	ult := 2
a3	while en2 and (ult = 1)	while en1 and (ult = 2)
a4	seguir	seguir
a5	SC	SC
a6	en1 := false	en2 := false
a7	SNC	SNC


this:in1_w(1)
a2
this:turn_w(1)
a1
a3
this:turn_r(t);
this:in2_r(i2);
guard ((t==2) | (i2==1))
this:in2_r(0)
this:turn_r(2)
a6
this:in1_w(0)
boolean en1 := fa
int ul
Process P
loop forever
a1 en1 := true
a2 ult := 1
a3 while en2 and (ult = 1)
a4 seguir
a5 SC
a6 en1 := false
a7 SNC
in1
0
v
:in1_r(v)
v
e
:in1_w(e)
in2
0
v
:in2_r(v)
v
e
:in2_w(e)
1
v
:turn_r(v)
v
e
:turn_w(e)


this:in1_w(1)
a2
this:turn_w(1)
a1
a3
this:turn_r(t);
this:in2_r(i2);
guard ((t==2) | (i2==1))
this:in2_r(0)
this:turn_r(2)
a6
this:in1_w(0)
this:in2_w(1)
b2
this:turn_w(2)
this:turn_r(t);
this:in2_r(i1);
guard ((t==2) | (i1==1))
b3
b1
this:turn_r(1)
this:in1_r(0)
b6
this:in2_w(0)
in1
0
v
:in1_r(v)
v
e
:in1_w(e)
in2
0
v
:in2_r(v)
v
e
:in2_w(e)
turn
1
v
:turn_r(v)
v
e
:turn_w(e)