

## LABORATORIO DE SISTEMAS OPERATIVOS

PERÍODO ACADÉMICO: 2025 – B

EQUIPO:

PROFESOR: Ing. Marcela Saavedra MSc.

TIPO DE INSTRUMENTO: Guía de Laboratorio

### TEMA: INTERBLOQUEOS O DEADLOCKS

#### ÍNDICE DE CONTENIDOS

1. OBJETIVOS .....	2
2. MARCO TEÓRICO.....	2
Detección de interbloqueo .....	2
3. PROCEDIMIENTO .....	3
3.1 Instalación de CPU-OS Simulator .....	3
3.2 Gráfico de asignación.....	3
3.3 Creación de un deadlock por medio de CPU-OS Simulator.....	3
3.4 Métodos de recuperación de una condición de bloqueo después de que se produzca..	11
3.5 Métodos de prevención de una condición de bloqueo .....	16
4. INFORME.....	24

#### ÍNDICE DE FIGURAS

Figura 1. Asignación de recursos a procesos .....	3
Figura 2. Ventana CPU Simulator.....	4
Figura 3. Compilador .....	4
Figura 4. Guardar los procesos .....	5
Figura 5. Compilación de procesos .....	5
Figura 6. Carga de procesos en memoria.....	6
Figura 7. Ventana de configuración .....	6
Figura 8. Procesos a estado listo .....	7
Figura 9. Uso de técnica RR.....	7
Figura 10. Simulación rápida .....	8
Figura 11. Ventana de recursos .....	8
Figura 12. Inicialización de simulación .....	9
Figura 13. Opción para mostrar estado de procesos .....	9
Figura 14. Liberación de recursos .....	11
Figura 15. Suspensión de procesos .....	14
Figura 16. Continuación de simulación.....	14
Figura 17. Técnicas de prevención de deadlocks .....	17

## 1. OBJETIVOS

- 1.1. Utilizar el simulador CPU Simulator para crear condiciones de bloqueo.
- 1.2. Implementar métodos para prevenir condiciones desbloqueo.

## 2. MARCO TEÓRICO

El interbloqueo es una situación que ocurre en el sistema operativo cuando cualquier proceso entra en un estado de espera porque otro proceso en espera está reteniendo el recurso demandado.

El interbloqueo es un problema común en el procesamiento múltiple donde varios procesos comparten un tipo específico de recurso mutuamente excluyente conocido como bloqueo suave o software.

### Detección de interbloqueo

El programador de recursos puede detectar una ocurrencia de interbloqueo. Un programador de recursos ayuda al sistema operativo a realizar un seguimiento de todos los recursos que se asignan a diferentes procesos. Por lo tanto, cuando se detecta un punto muerto, se puede resolver utilizando los métodos que se indican a continuación:

#### *Prevención de bloqueos:*

Es importante evitar un punto muerto antes de que pueda ocurrir. El sistema verifica cada transacción antes de que se ejecute para asegurarse de que no conduzca a las situaciones de punto muerto. De tal manera que, incluso un pequeño cambio que puede conducir a un *deadlock* en el futuro se permite que se ejecute.

Es un conjunto de métodos para garantizar que al menos una de las condiciones no pueda mantenerse.

#### *Ninguna acción preventiva:*

Sin preferencia: un recurso puede ser liberado solo voluntariamente por el proceso que lo retiene después de que ese proceso haya terminado su tarea.

### 3. PROCEDIMIENTO

#### 3.1 Instalación de CPU-OS Simulator

Descargar e instalar el simulador de:  
<https://teach-sim.com/downloads/>

#### 3.2 Gráfico de asignación

**Problema:** Se están ejecutando cuatro procesos. Se llaman P1 a P4. También hay cuatro recursos disponibles (sólo una instancia de cada uno). Se denominan R0 a R3. En algún momento de su existencia, cada proceso asigna un recurso diferente para su uso y lo mantiene para sí mismo para siempre. Posteriormente cada uno de los procesos solicita otro de los cuatro recursos.

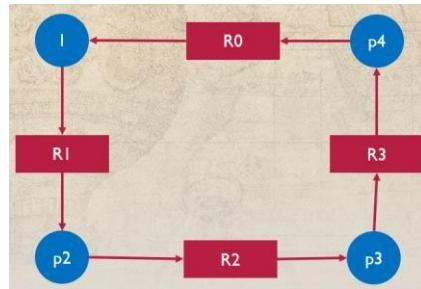


Figura 1. Asignación de recursos a procesos

#### 3.3 Creación de un deadlock por medio de CPU-OS Simulator.

##### Paso 1: Abrir CPU-OS simulator.

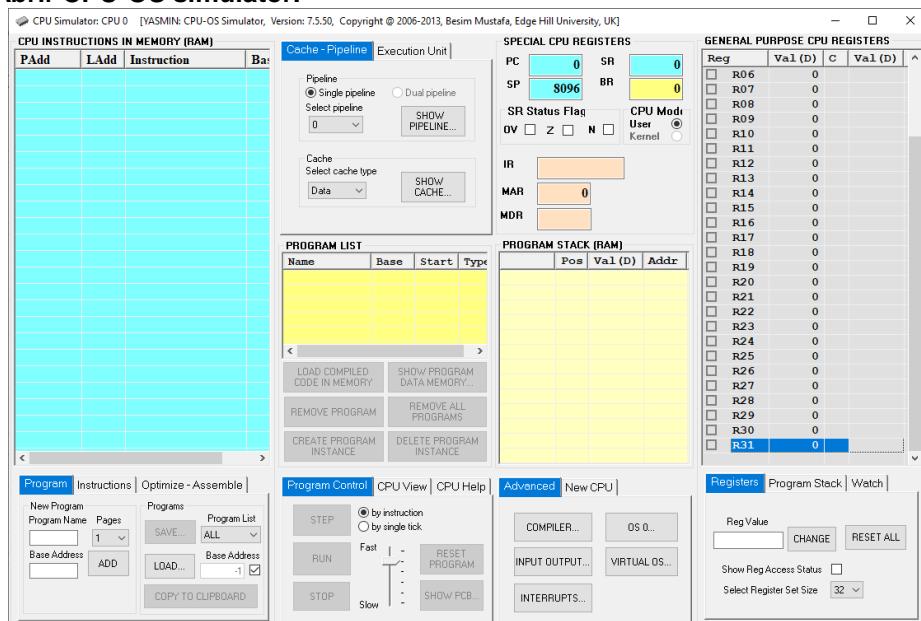


Figura 2. Ventana CPU Simulator

**Paso 2:** elegir la opción “compiler”, esto deberá abrir una ventana emergente.

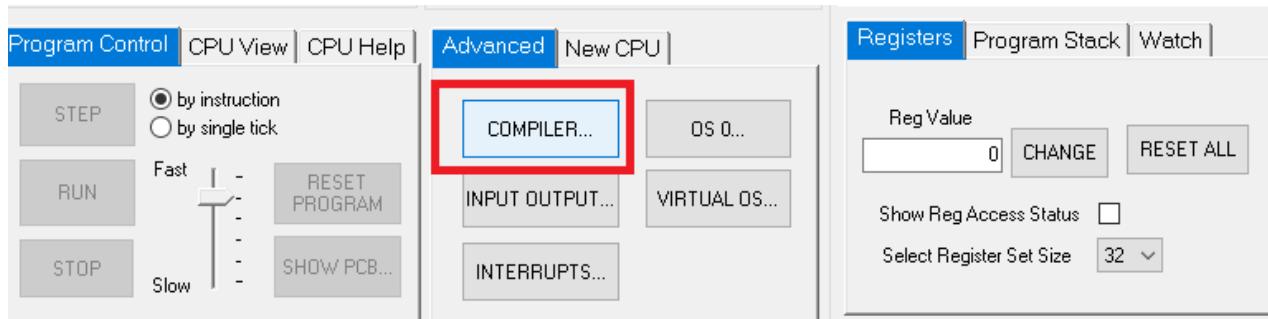


Figura 3. Compilador

**Paso 3.** Crear 4 procesos con recursos asignados y en espera de otro.

Colocar el código para cada proceso como el ejemplo de la figura en el Program Source y guardar como P.txt

P1.txt	R0 → P1 → R1
P2.txt	R1 → P2 → R2
P3.txt	R2 → P3 → R3
P4.txt	R3 → P4 → R0

```
program DeadlockP4
resource(0, allocate)
wait(3)
resource(1, allocate)
for n = 1 to 20
next
end
```

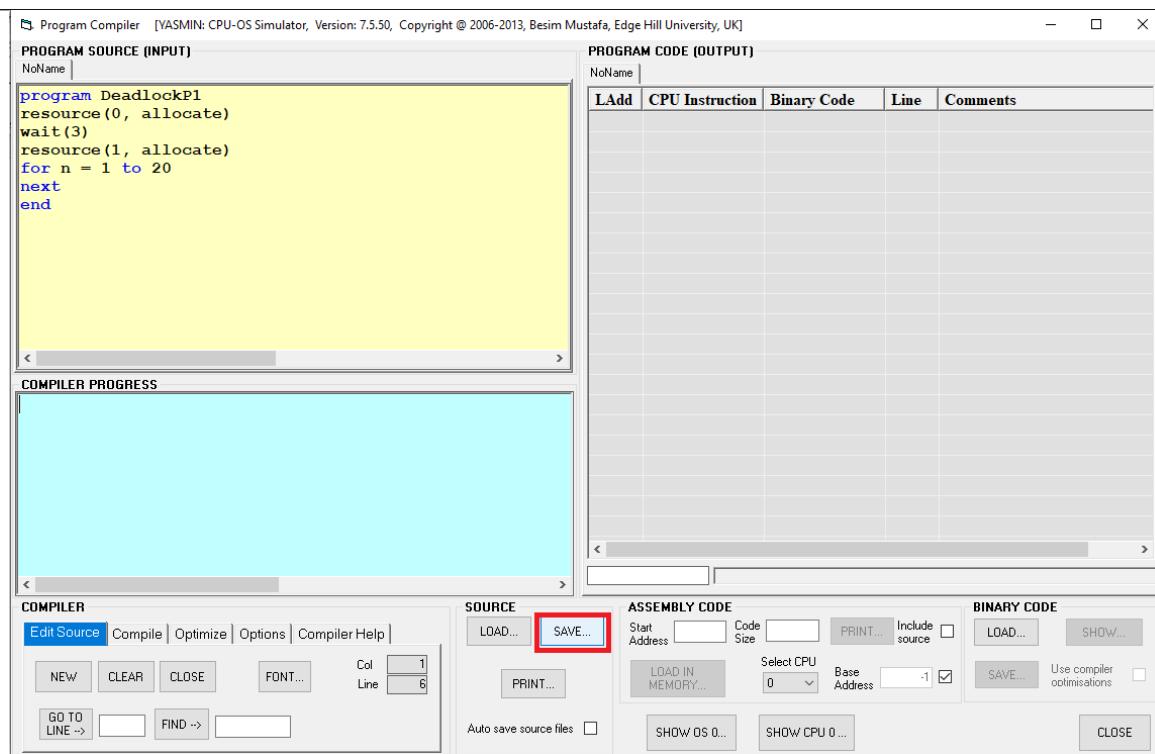


Figura 4. Guardar los procesos

#### Paso4. Compilar todos los programas creados

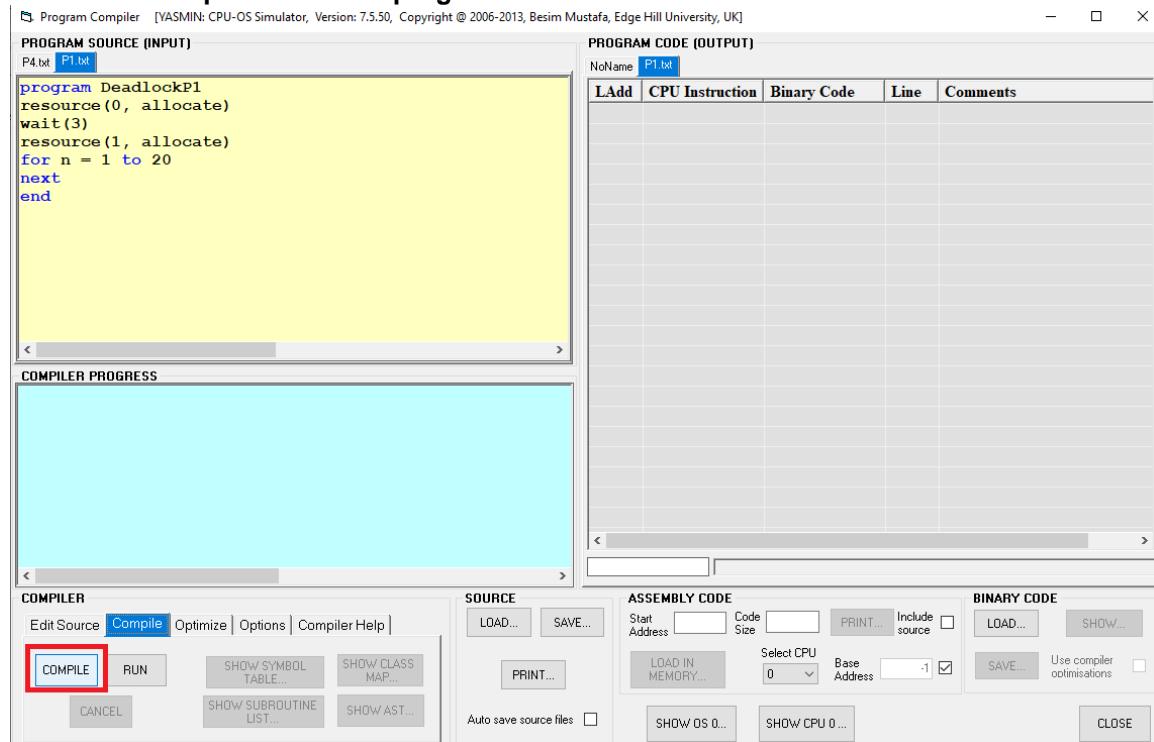


Figura 5. Compilación de procesos



**Paso 5: Cargar los programas a memoria.**

Escoger el programa y seleccionar la opción “load in memory”.

PROGRAM SOURCE (INPUT)  
P4.txt P1.txt P2.txt | P3.txt | P4.txt |  
**program** DeadlockP1  
resource(0, allocate)  
wait(3)  
resource(1, allocate)  
for n = 1 to 20  
next  
end

COMPILER PROGRESS  
1:....Found keyword NEXT [6]  
0:Found keyword END [7]  
  
Code generation completed...  
Displaying generated code  
Display completed...  
  
\*\*\* NOTE: Click on numbers in brackets to highlight c

PROGRAM CODE (OUTPUT)  
NoName P1.txt P2.txt | P3.txt | P4.txt |  

LAdd	CPU Instruction	Binary Code	Line	Comments
****	CODE:			
****	MAIN PROG...			
0000	PSH #0	0E000000	0002	Push the value of 0 on top of stack
0004	SWI 12	2E02000C	0002	OS system call: Resource allocation
0008	POP R01	0F0101	0002	Pop the value from top of stack
0011	MOV #3, R04	000000030104	0003	Copy the value of 3 to _STempR
0017	PSH R04	0E0104	0003	Wait time
0020	SWI 6	2E020006	0003	OS system call: Wait
0024	PSH #1	0E000001	0004	Push the value of 1 on top of stack
0028	SWI 12	2E02000C	0004	OS system call: Resource allocation
0032	POP R02	0F0102	0004	Pop the value from top of stack
0035	MOV #1, R06	000000010106	0005	Copy the value of 1 to _STempR
0041	MOV R06, R04	0001060104	0005	Copy the value of _STempR to R04
0046	MOV #20, R06	0000000140106	0005	Copy the value of 20 to _STempR
0052	CMP R06, R04	3001060104	0005	Compare N with _STempReq
0057	JGT 71	20020047	0005	Jump to code at address 71 if greater than
0061	ADD #1, R04	110000010104	0006	Add: N = N + 1
0067	JMP 52	1D020034	0006	Jump to code at address 52
0071	HLT	2F	0007	Stop simulation

COMPILER  
Edit Source **Compile** Optimize | Options | Compiler Help |  
COMPILE RUN SHOW SYMBOL TABLE... SHOW CLASS MAP... SOURCE LOAD... SAVE... PRINT... ASSEMBLY CODE Start Address 0 Code Size 72 PRINT... Include source  LOAD... SHOW...  
SHOW SUBROUTINE LIST... SHOW AST... PRINT... Auto save source files  LOAD IN MEMORY... Select CPU 0 Base Address -1  SHOW OS 0... SHOW CPU 0...  
CANCEL SHOW 0... SHOW CPU 0... CLOSE

Figura 6. Carga de procesos en memoria

**Paso 6: Seleccionar la opción “OS Simulator”.**

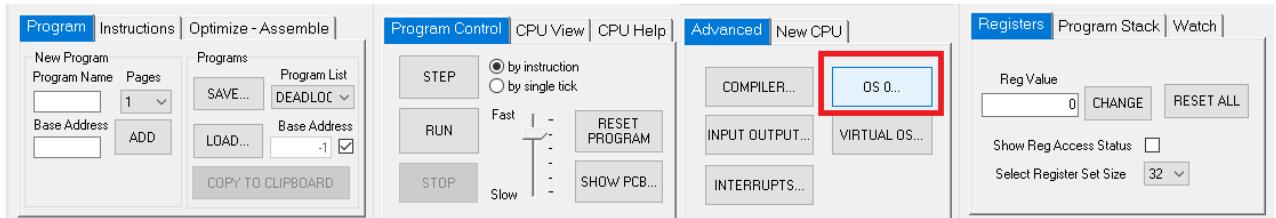


Figura 7. Ventana de configuración

**Paso 7: Crear una instancia de cada programa.**

Doble clic sobre cada nombre del programa en “Program list”, cuando se cree la instancia de los programas aparecerán en “READY PROCESSES (Ready Queue)”.

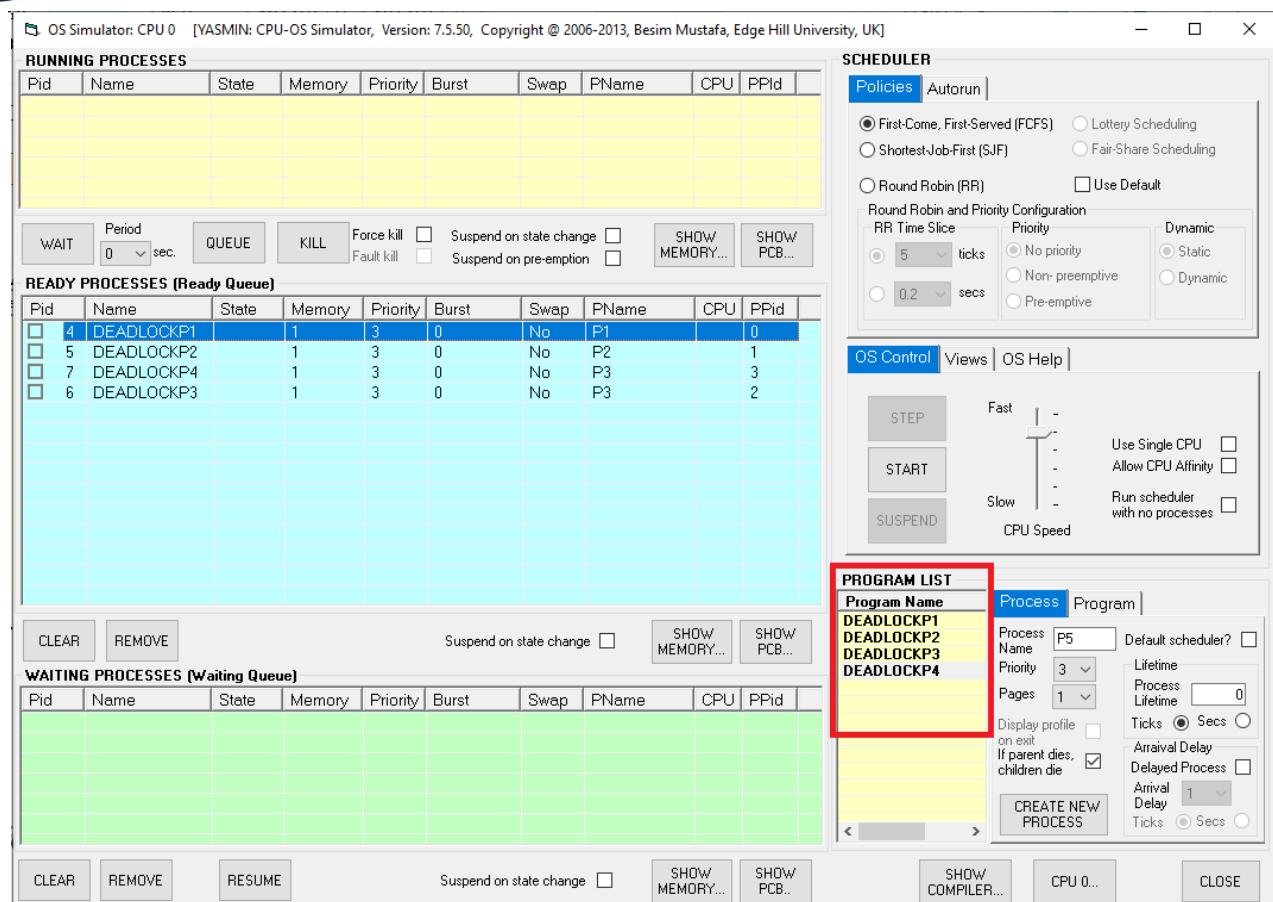


Figura 8. Procesos a estado listo

### Paso 8: Seleccionar la técnica Round Robin en SCHEDULER → Policies.

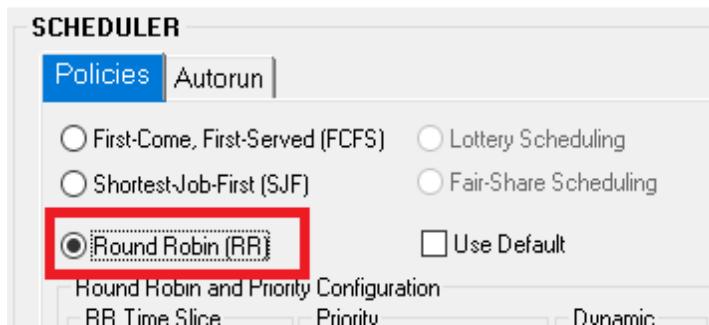


Figura 9. Uso de técnica RR

### Paso 9: Colocar “CPU speed” a su máxima velocidad en OS Control.

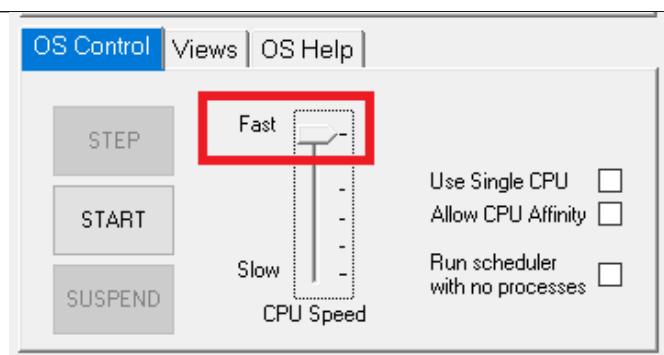


Figura 10. Simulación rápida

**Paso 10:** En la pestaña “Views” seleccionar “View resources”.  
En la ventana emergente marcar Stay on top.



Figura 11. Ventana de recursos

**Paso 11:** Dentro de la ventana “OS control” seleccionar “Start” para comenzar la simulación y observar cómo cambian los estados de los recursos.

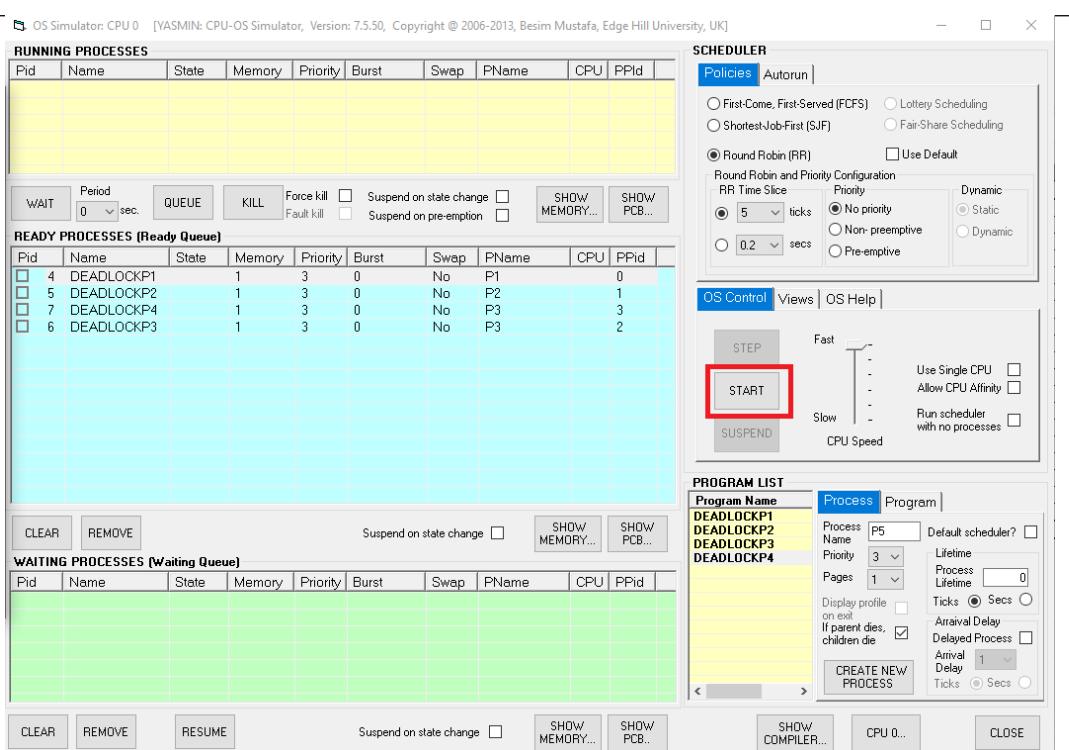


Figura 12. Inicialización de simulación

Mostrar el resultado que muestra la opción SHOW DEADLOCKED PROCESSES dentro de la vista de recursos.  
Explicar detalladamente lo observado.

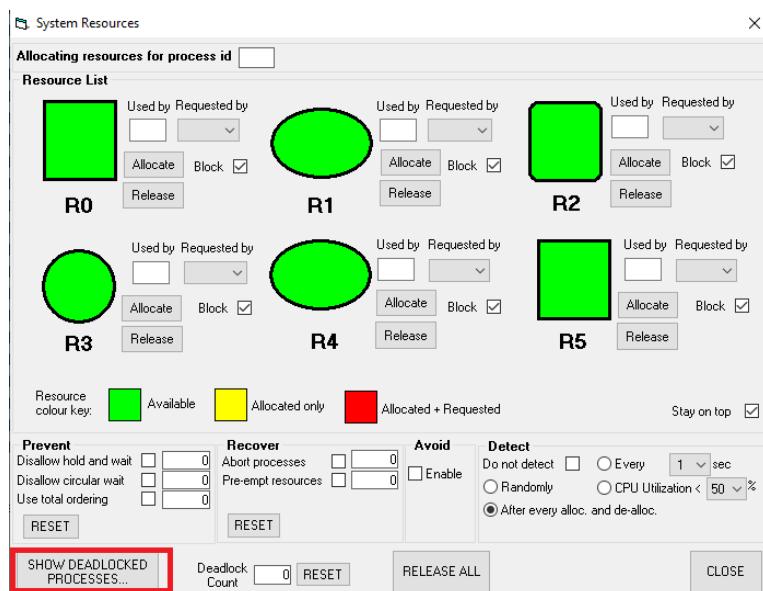
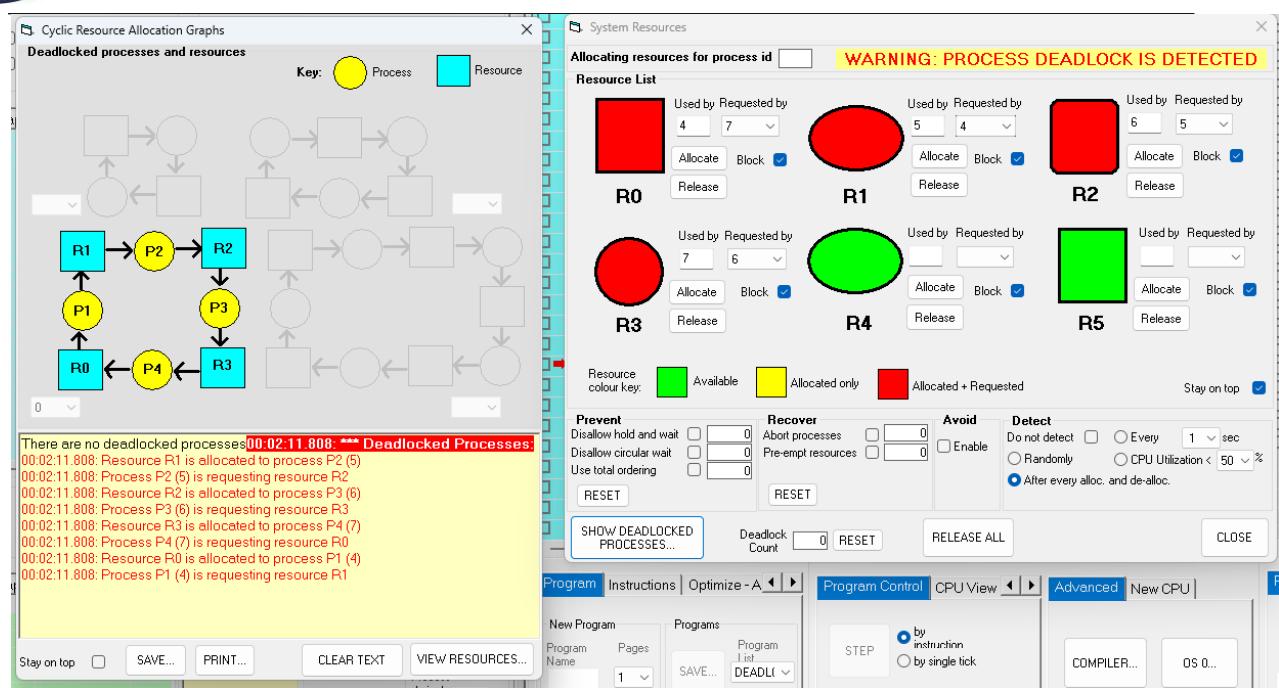


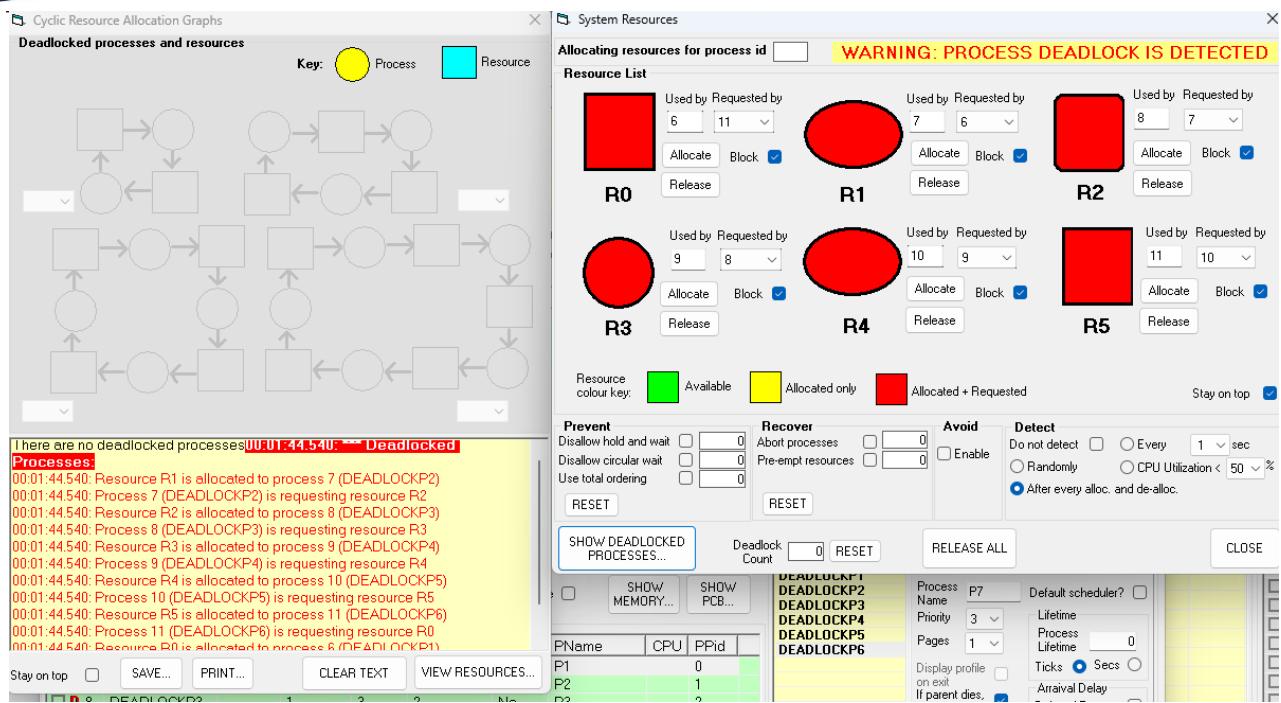
Figura 13. Opción para mostrar estado de procesos



En la imagen se puede notar que al momento de ejecutar los procesos, existe una espera circular, ya que existe un "bucle" en el que cada proceso espera un recurso "R" de otro.

**Repetir el procedimiento con 6 procesos y 6 recursos.**

- P1: R0 → R1
- P2: R1 → R2
- P3: R2 → R3
- P4: R3 → R4
- P5: R4 → R5
- P6: R5 → R0



En el caso de los 6 procesos, aunque haya una ausencia de la imagen de los interbloqueos, podemos notar en el texto la espera circular que se notaba en el anterior ejemplo, solo que en este caso, existen mas procesos y recursos, podemos notar que hay un bloqueo en los 5 recursos, ya que como se mencionó antes, existe un bucle de espera entre cada uno

### 3.4 Métodos de recuperación de una condición de bloqueo después de que se produzca

**Paso 1:** En la ventana de Recursos del Sistema liberar el recurso R0.

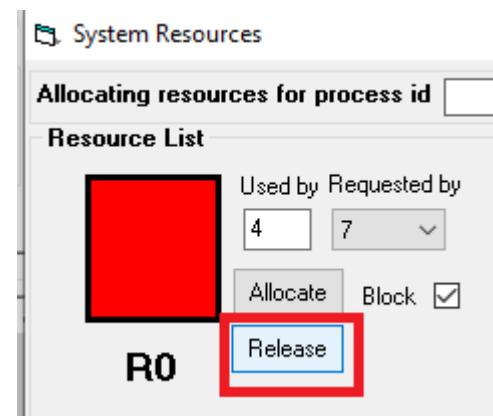


Figura 14. Liberación de recursos

**Paso 2:** Observar durante 5 minutos y documentar lo que ocurre.

Responder:

¿Se ha resuelto la condición de bloqueo? ¿Por qué?



System Resources

Allocating resources for process id [ ] **WARNING: PROCESS DEADLOCK IS DETECTED**

Resource List

Resource	Used by	Requested by	Allocation	Action
R0	6	11	<input type="checkbox"/> Allocate <input type="checkbox"/> Block <input type="checkbox"/> Release	
R1	7	6	<input type="checkbox"/> Allocate <input checked="" type="checkbox"/> Block <input type="checkbox"/> Release	
R2	8	7	<input type="checkbox"/> Allocate <input type="checkbox"/> Block <input type="checkbox"/> Release	
R3	9	8	<input type="checkbox"/> Allocate <input type="checkbox"/> Block <input type="checkbox"/> Release	
R4	10	9	<input type="checkbox"/> Allocate <input checked="" type="checkbox"/> Block <input type="checkbox"/> Release	
R5	11	10	<input type="checkbox"/> Allocate <input type="checkbox"/> Block <input type="checkbox"/> Release	

Resource colour key:  Available  Allocated only  Allocated + Requested

Stay on top

**Prevent**

Disallow hold and wait  0  
Disallow circular wait  0  
Use total ordering  0

**RESET**

**Recover**

Abort processes  0  
Pre-empt resources  0

**RESET**

**Avoid**

Enable

**Detect**

Do not detect   
Every  1 sec  
Randomly   
CPU Utilization <  50 %  
 After every alloc. and de-alloc.

**SHOW DEADLOCKED PROCESSES...** Deadlock Count  **RESET** **RELEASE ALL** **CLOSE**



Se puede notar que los procesos se van liberando, pasando de rojo a amarillo y después a blanco, en la otra pantalla podemos notar que todos los procesos se han removido. Primero empieza liberando el recurso 0, lo cual causa una reacción en cadena que rompe la espera circular y por ende quita el deadlock.

**Paso 3: Repetir la condición de bloqueo.**

**Paso 4: En la ventana del OS Simulator, seleccionar un proceso de la cola de espera, luego seleccionar SUSPENDER y luego REMOVER.**



The screenshot shows the YASMIN CPU-OS Simulator interface. The main window displays three tables: 'RUNNING PROCESSES', 'READY PROCESSES (Ready Queue)', and 'WAITING PROCESSES (Waiting Queue)'. The 'READY PROCESSES' table is highlighted in light blue. The 'WAITING PROCESSES' table has a red box around the 'REMOVE' button, with a red number '3' above it. The 'SCHEDULER' panel on the right shows various scheduling policies and configuration options. The 'OS Control' panel at the bottom includes buttons for 'STEP', 'STOP', and 'RESUME', with the 'RESUME' button highlighted by a red box and a red number '2' above it. The 'PROGRAM LIST' panel on the right shows a list of processes with their details.

Pid	Name	State	Memory	Priority	Burst	Swap	PName	CPU	PPid

Pid	Name	State	Memory	Priority	Burst	Swap	PName	CPU	PPid

Pid	Name	State	Memory	Priority	Burst	Swap	PName	CPU	PPid
<input checked="" type="checkbox"/> D 4	DEADLOCKP1	1	3	2	No	P1	0		
<input type="checkbox"/> D 6	DEADLOCKP3	1	3	2	No	P2	2		
<input type="checkbox"/> D 5	DEADLOCKP2	1	3	2	No	P2	1		
<input type="checkbox"/> D 7	DEADLOCKP4	1	3	2	No	P4	3		

Figura 15. Suspensión de procesos

**Paso 5: Seleccionar RESUME para continuar con el proceso.**

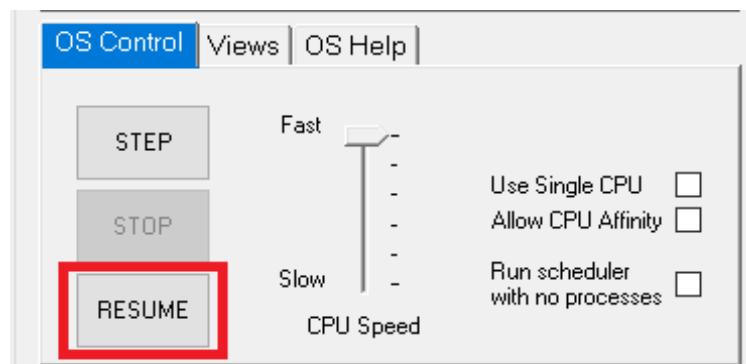


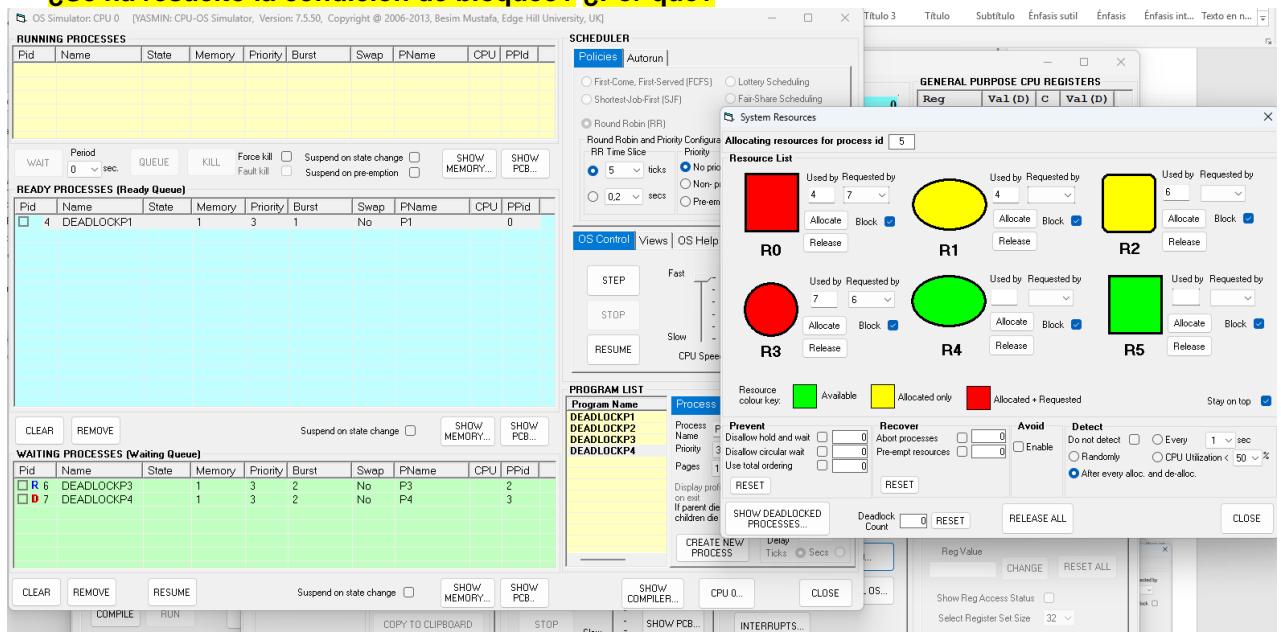
Figura 16. Continuación de simulación



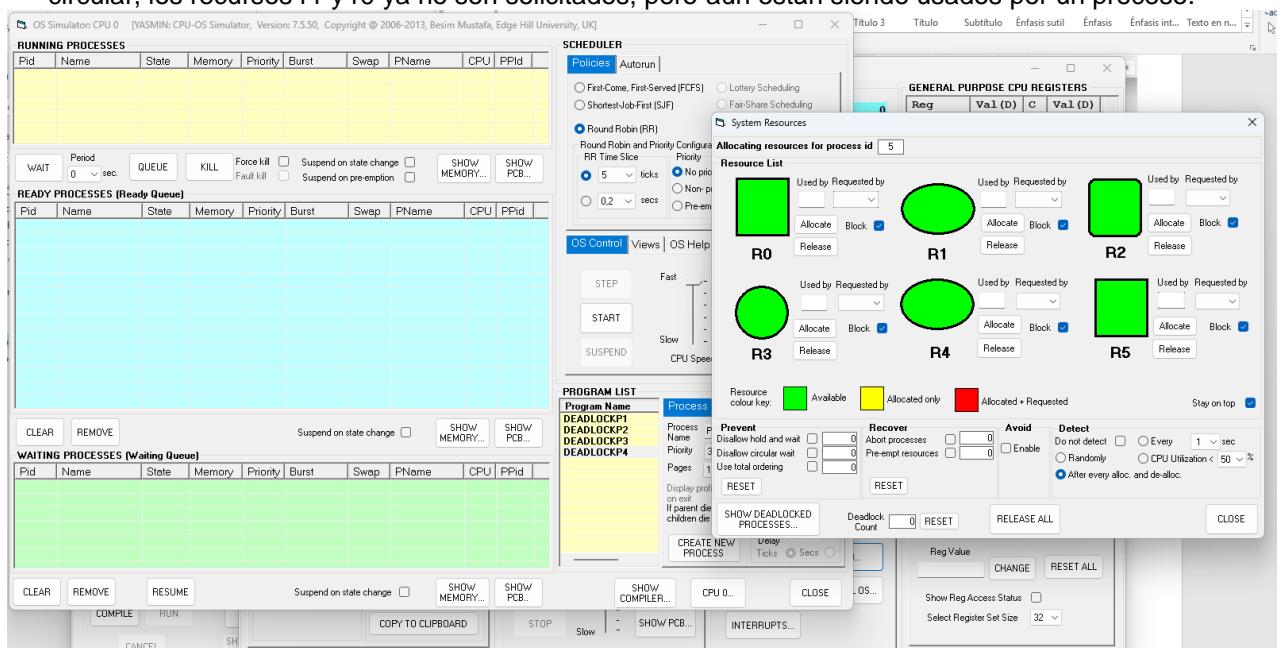
**Paso 6: Observar durante 5 minutos y documentar lo que ocurre.**

**Responder:**

**¿Se ha resuelto la condición de bloqueo? ¿Por qué?**

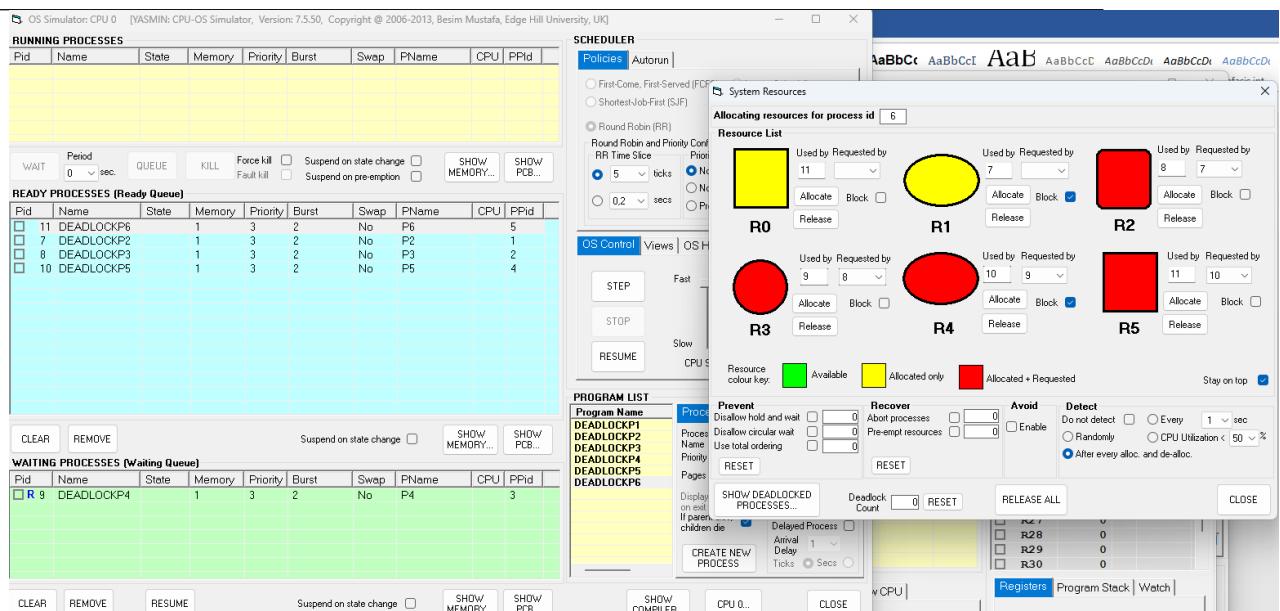


Al momento de remover y suspender el proceso se elimina el deadlock, ya que se rompió la espera circular, los recursos r1 y r0 ya no son solicitados, pero aún están siendo usados por un proceso.

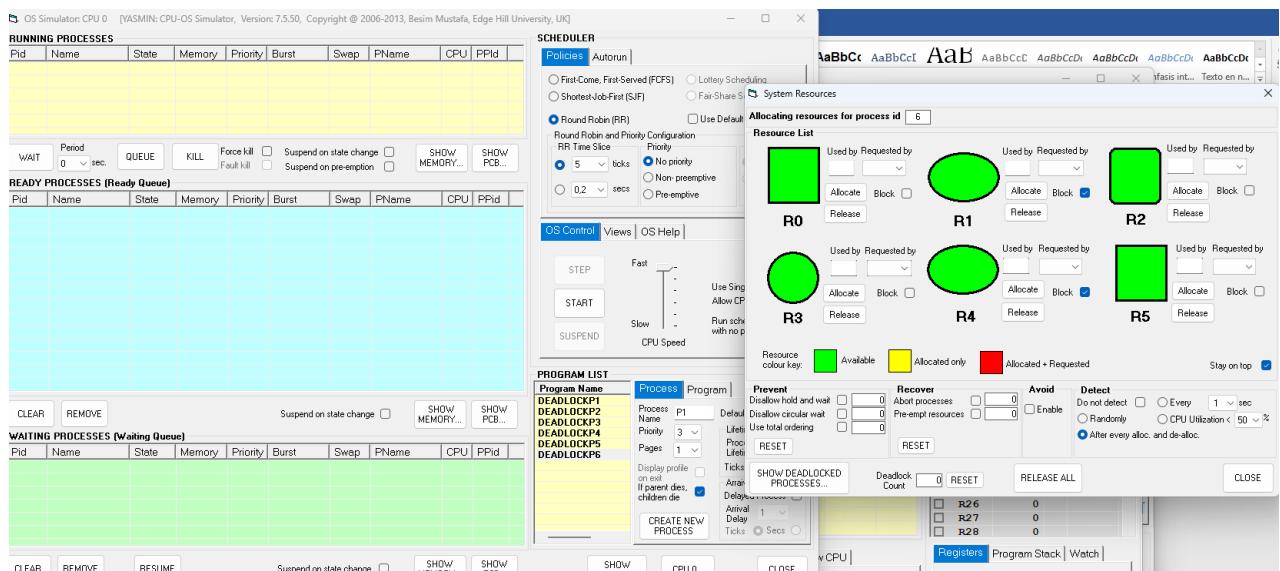


Al reanudar los procesos notamos que ocurre la misma reacción en cadena vista en el ejemplo con los 6 procesos y recursos

**Repetir el procedimiento con 6 procesos y 6 recursos.**



Al momento de remover y suspender el proceso se elimina el deadlock, ya que se rompió la espera circular, los recursos r1 y r0 ya no son solicitados, pero aún están siendo usados por un proceso.



Finalmente, cuando reanudamos el proceso notamos que se liberan totalmente todos los recursos, lo que quiere decir que se ha resuelto la condición de bloqueo, porque se ha eliminado un proceso que mantenía la espera circular, en este caso p1, eso causa que R0 deje de ser solicitado, entonces P6 ya no se queda esperando a dicho recurso, causando la misma reacción en cadena que el anterior ejercicio.

### 3.5 Métodos de prevención de una condición de bloqueo

**Paso 1: Repetir el proceso para condición de bloqueo de 4 y 6 procesos, pero antes de seleccionar START habilitar la casilla “Disallow hold and wait”.**



System Resources

Allocating resources for process id 4

Resource List

Resource	Used by	Requested by	Allocate	Block	Release
R0	[ ]	[ ]	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/> Block	<input type="button" value="Release"/>
R1	[ ]	[ ]	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/> Block	<input type="button" value="Release"/>
R2	[ ]	[ ]	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/> Block	<input type="button" value="Release"/>
R3	[ ]	[ ]	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/> Block	<input type="button" value="Release"/>
R4	[ ]	[ ]	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/> Block	<input type="button" value="Release"/>
R5	[ ]	[ ]	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/> Block	<input type="button" value="Release"/>

Resource colour key:  Available  Allocated only  Allocated + Requested

Stay on top

**Prevent**

Disallow hold and wait <input checked="" type="checkbox"/>	Abort processes <input type="button" value="0"/>
Disallow circular wait <input type="checkbox"/>	Pre-empt resources <input type="button" value="0"/>
Use total ordering <input type="checkbox"/>	<input type="checkbox"/> Enable

**RESET**

**Recover**

Abort processes <input type="button" value="0"/>
--

**Avoid**

<input type="checkbox"/> Enable
---------------------------------

**Detect**

Do not detect <input type="checkbox"/>	Every <input type="button" value="1 sec"/>
Randomly <input type="radio"/>	CPU Utilization < <input type="button" value="50 %"/>
<input type="radio"/> After every alloc. and de-alloc.	

**SHOW DEADLOCKED PROCESSES...**

Deadlock Count  **RESET**

**RELEASE ALL**

**CLOSE**

Figura 17. Técnicas de prevención de deadlocks

procesos:



Notamos el mismo desarrollo que con los 6 procesos, fue bastante más rápido y el contador queda en 6, que como establecimos anteriormente, cuenta los posibles casos evitados en los que se pudo causar un deadlock.

**6 procesos:**



System Resources

Allocating resources for process id 6

Resource List

Resource	Used by	Requested by	Allocate	Block	Release
R0	11		<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/> Block	<input type="button" value="Release"/>
R1	7	6	<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/> Block	<input type="button" value="Release"/>
R2	7		<input type="button" value="Allocate"/>	<input type="checkbox"/> Block	<input type="button" value="Release"/>
R3			<input type="button" value="Allocate"/>	<input type="checkbox"/> Block	<input type="button" value="Release"/>
R4			<input type="button" value="Allocate"/>	<input checked="" type="checkbox"/> Block	<input type="button" value="Release"/>
R5	10		<input type="button" value="Allocate"/>	<input type="checkbox"/> Block	<input type="button" value="Release"/>

Resource colour key: █ Available █ Allocated only █ Allocated + Requested Stay on top

**Prevent**

Disallow hold and wait  11  
Disallow circular wait  0  
Use total ordering  0

**RESET**

**Recover**

Abort processes  0  
Pre-empt resources  0

**RESET**

**Avoid**

Enable

**Detect**

Do not detect   
Every  1 sec  
Randomly   
CPU Utilization < 50 %  
After every alloc. and de-alloc.

**SHOW DEADLOCKED PROCESSES...** **Deadlock Count** 0 **RESET** **RELEASE ALL** **CLOSE**



System Resources

Allocating resources for process id 6

Resource List

Resource	Used by	Requested by
R0	[ ]	[ ]
R1	[ ]	[ ]
R2	[ ]	[ ]
R3	[ ]	[ ]
R4	[ ]	[ ]
R5	[ ]	[ ]

Allocate Block  Release

Resource colour key: Available (Green) Allocated only (Yellow) Allocated + Requested (Red)

Stay on top

Prevent

Disallow hold and wait  12

Disallow circular wait  0

Use total ordering  0

RESET

Recover

Abort processes  0

Pre-empt resources  0

RESET

Avoid

Enable

Detect

Do not detect  Every 1 sec

Randomly  CPU Utilization < 50 %

After every alloc. and de-alloc.

SHOW DEADLOCKED PROCESSES... Deadlock Count 0 RESET RELEASE ALL CLOSE

Notamos que, a diferencia de los otros procesos, no hubo un deadlock, y “Disallow hold and wait”, subió su contador a 12. Lo que pasa al activar dicha opción previene que un proceso retenga un recurso mientras solicita otro. Gracias a eso, cada proceso debe solicitar todos los recursos que va a necesitar desde un inicio, por eso en el proceso se nota que van cambiando de color, porque están utilizando dichos recursos y finalmente los liberan antes de solicitar uno nuevo, por todo lo dicho, no hay posibilidad de que se genere espera circular y por ende no hay deadlock.

Finalmente, el contador subió a 12 porque este corresponde al conteo de deadlocks que pudieron ocurrir y refleja que el sistema está reintentando asignaciones bajo las condiciones que impone activar disallow hold and wait.

**Paso 2: Repetir el proceso para condición de bloqueo de 4 y 6 procesos, pero antes de seleccionar START habilitar la casilla “Disallow circular wait”.**

4 procesos:



The screenshot shows the OS Simulator interface with several windows open. The main window displays process tables for Running, Ready, and Waiting queues. The Ready queue contains 6 processes labeled D4 through D9. The Waiting queue contains 5 processes labeled D10 through D14. The Resource List window shows 5 resources (R0-R5) and 6 processes (D1-D5) in a state where no process can release its held resources, leading to a deadlock. The General Purpose CPU Registers window displays register values for process D1, showing a warning about a detected deadlock.

Se vio un resultado bastante similar al de los 6 procesos, donde se causa el deadlock, pese a que se activa la opción que debería evitarlo.

### 6 procesos:

The screenshot shows the OS Simulator interface with several windows open. The main window displays process tables for Running, Ready, and Waiting queues. The Ready queue contains 6 processes labeled D6 through D11. The Waiting queue contains 6 processes labeled D12 through D17. The Resource List window shows 6 resources (R0-R5) and 6 processes (D1-D6) in a state where no process can release its held resources, leading to a deadlock. The General Purpose CPU Registers window displays register values for process D1, showing a warning about a detected deadlock.

La opción disallow circular wait, aunque se supone debe prevenir deadlocks, en esta práctica ha fallado porque los procesos ya han adquirido recursos antes de activar esta opción, permitiendo que un ciclo exista en el momento de la verificación. Además, el simulador puede implementar una versión no estricta que solo evita ciclos inmediatos, pero no detecta ni rompe ciclos formados durante condiciones de carrera o en escenarios con múltiples instancias de recursos, esto explica porque aún se genera el deadlock.



**Paso 3: Repetir el proceso para condición de bloqueo de 4 y 6 procesos, pero antes de seleccionar START habilitar la casilla “Use total ordering”.**  
**4 procesos:**



Podemos notar un comportamiento similar al ejemplo de los 6 procesos y recursos, con la diferencia en el contador de la opción, que como fue anteriormente establecido son los posibles casos en los que pudo haber un deadlock, este es de 3, debido a que son menos procesos y recursos los que estaban en ejecución

**6 procesos:**



System Resources

Allocating resources for process id

Resource List

Resource	Used by	Requested by
R0	6	
R1	7	
R2	8	
R3	9	
R4	10	
R5	11	10

Buttons: Allocate, Block, Release.

Resource colour key:  Available    Allocated only    Allocated + Requested

Stay on top

**Prevent**

Disallow hold and wait  12  
Disallow circular wait  0  
Use total ordering  5

**RESET**

**Recover**

Abort processes  0  
Pre-empt resources  0

**RESET**

**Avoid**

Enable

**Detect**

Do not detect   
Every  1 sec  
Randomly   
CPU Utilization <  50 %  
 After every alloc. and de-alloc.

**SHOW DEADLOCKED PROCESSES...**

Deadlock Count  **RESET**

**RELEASE ALL**

**CLOSE**



Notamos que al activar “Use total ordering”, todos los recursos son utilizados y solicitados (se nota por su cambio de color), pero nunca se causa un deadlock, mientras eso ocurre notamos que el contador de la opción sube a 5. Dicha opción establece un orden orden global para todos los recursos, cada proceso debe solicitar los recursos en dicho orden impuesto, esto causa que todos los recursos sean asignados y solicitados, sin producir un deadlock, ya que, elimina la posibilidad de espera circular, porque, un recurso de orden mayor, nunca podrá solicitar uno de orden menor, que otro proceso ya esté usando. El contador sube a 5, porque cuenta los posibles deadlocks evitados por esta regla que impone la opción activada.

## 4. INFORME

## 5. CONCLUSIONES Y RECOMENDACIONES

En conclusión, es fundamental conocer como funcionan los procesos dentro de un sistema operativo, conocer las razones por las que pueden entrar en un deadlock y saber cómo prevenirlos, este conocimiento lo obtuvimos con los incisos en los que se solicitaba activar opciones como “Disallow hold and wait”, remover un proceso, etc. Gracias a esto, tendremos los conocimientos necesarios para planificar cómo programar los procesos en futuros trabajos.

Gracias a la simulación con CPU-OS Simulator permitió visualizar de manera práctica cómo se generan, detectan y resuelven los interbloqueos, confirmando que la prevención mediante las opciones



## FACULTAD DE INGENIERÍA EN SISTEMAS

como “Disallow hold and wait” o “Use total ordering” es efectiva. Este ejercicio refuerza la importancia de diseñar sistemas con estrategias que equilibren seguridad y eficiencia, aplicando técnicas adecuadas según el contexto para evitar bloqueos que comprometan la disponibilidad del sistema.