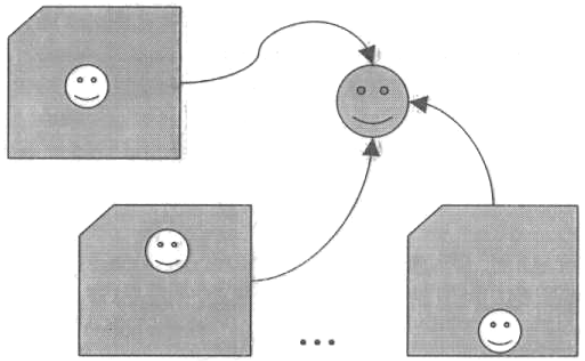
O problema da secção crítica

Surge quando temos n processos a competir pelo uso de dados partilhados



Neste caso, cada processo tem um segmento de código, chamados SECÇÃO CRITICA, no qual se efectua o acesso aos dados partilhados

O PROBLEMA:

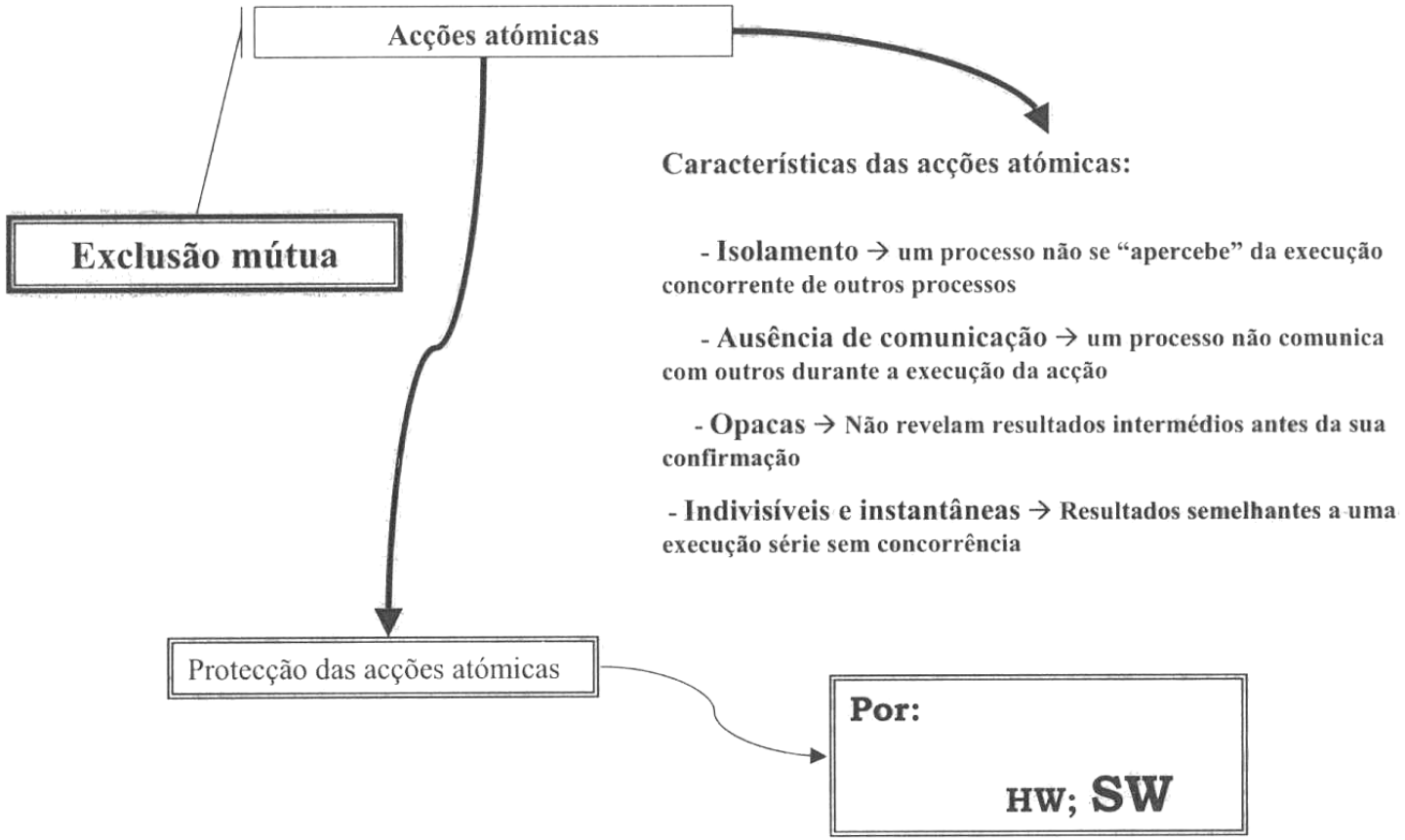
Assegurar que quando um processo está em execução na sua secção crítica, mais nenhum possa estar também em execução na sua secção crítica

Requisitos necessários para as soluções deste problema:

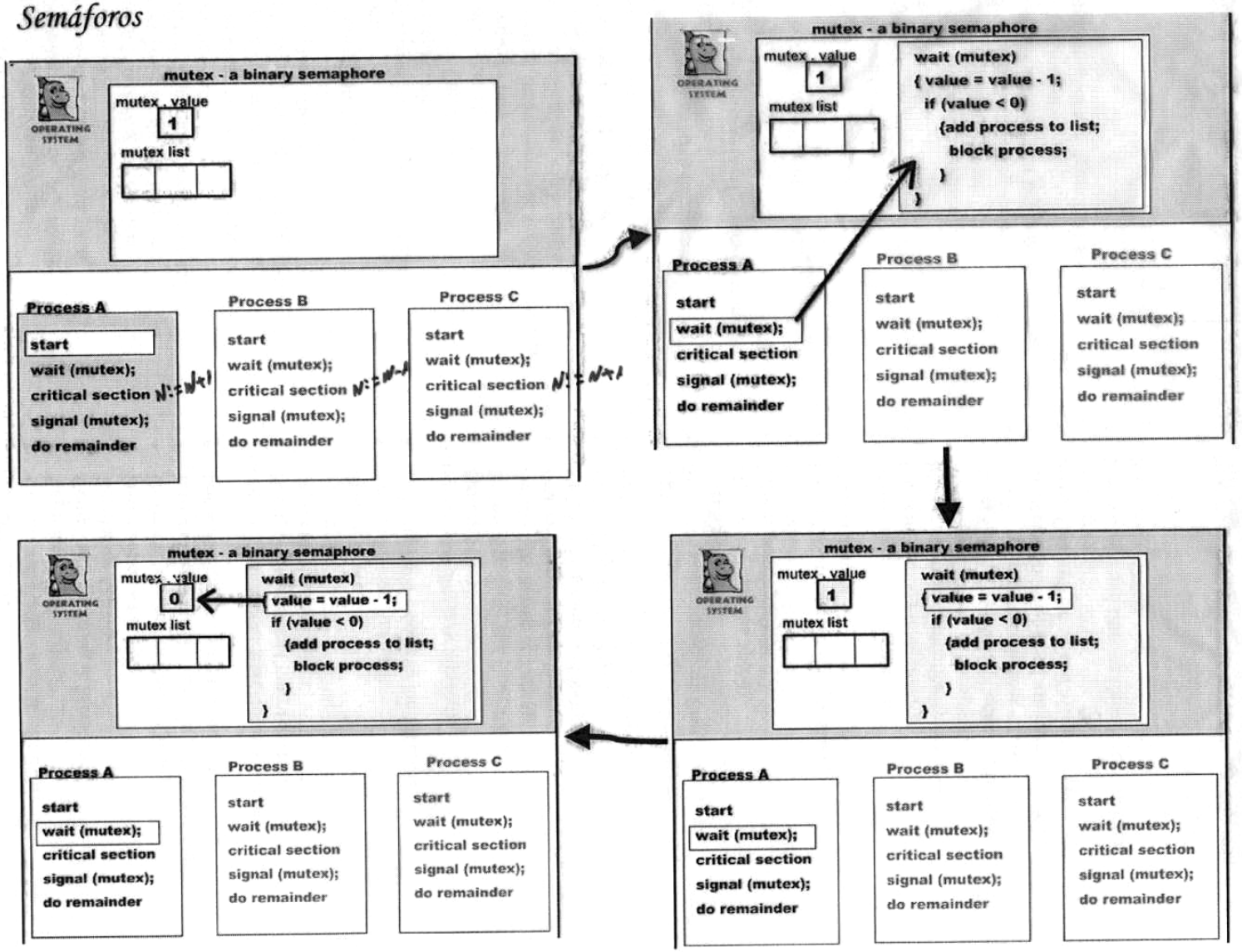
- **Exclusão mútua**
- **Progresso**
- **Espera limitada**

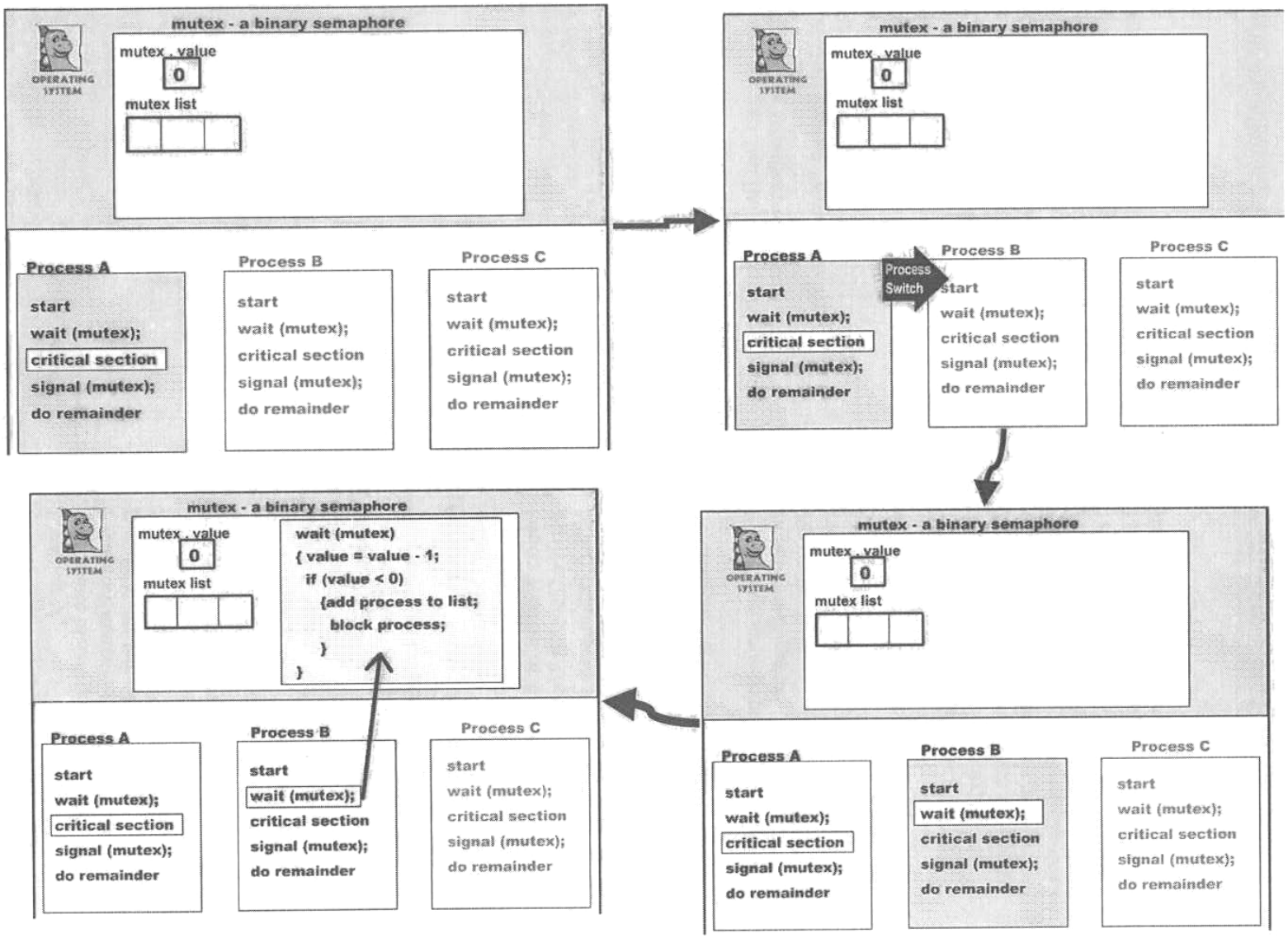
Solução:

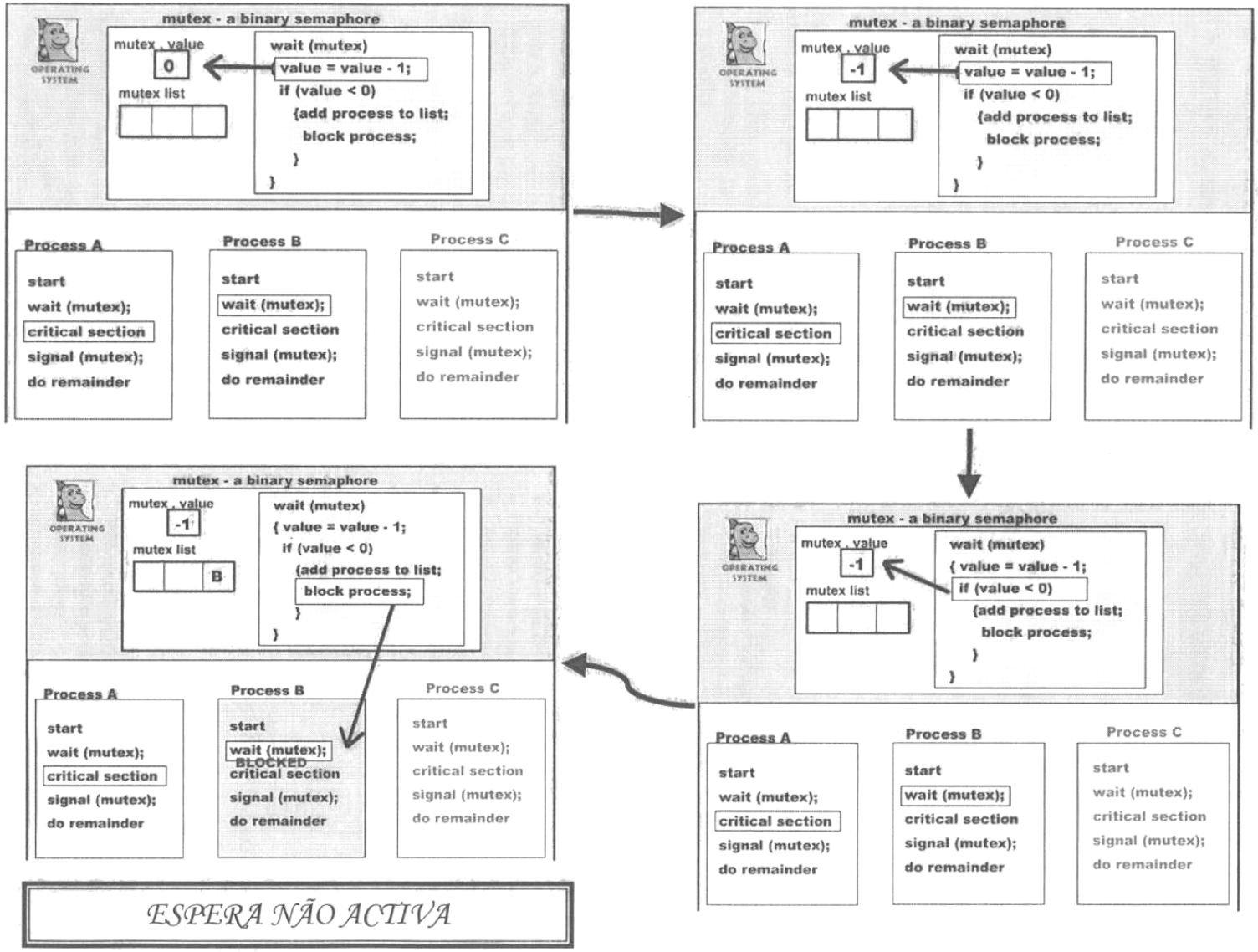
Implementar exclusão mútua por HW ou SW


Acções atómicas
Exclusão mútua
Características das acções atómicas:
- Isolamento → um processo não se "apercebe" da execução concorrente de outros processos
- Ausência de comunicação → um processo não comunica com outros durante a execução da acção
- Opacas → Não revelam resultados intermédios antes da sua confirmação
- Indivisíveis e instantâneas → Resultados semelhantes a uma execução série sem concorrência
Protecção das acções atómicas
Por:
HW; SW

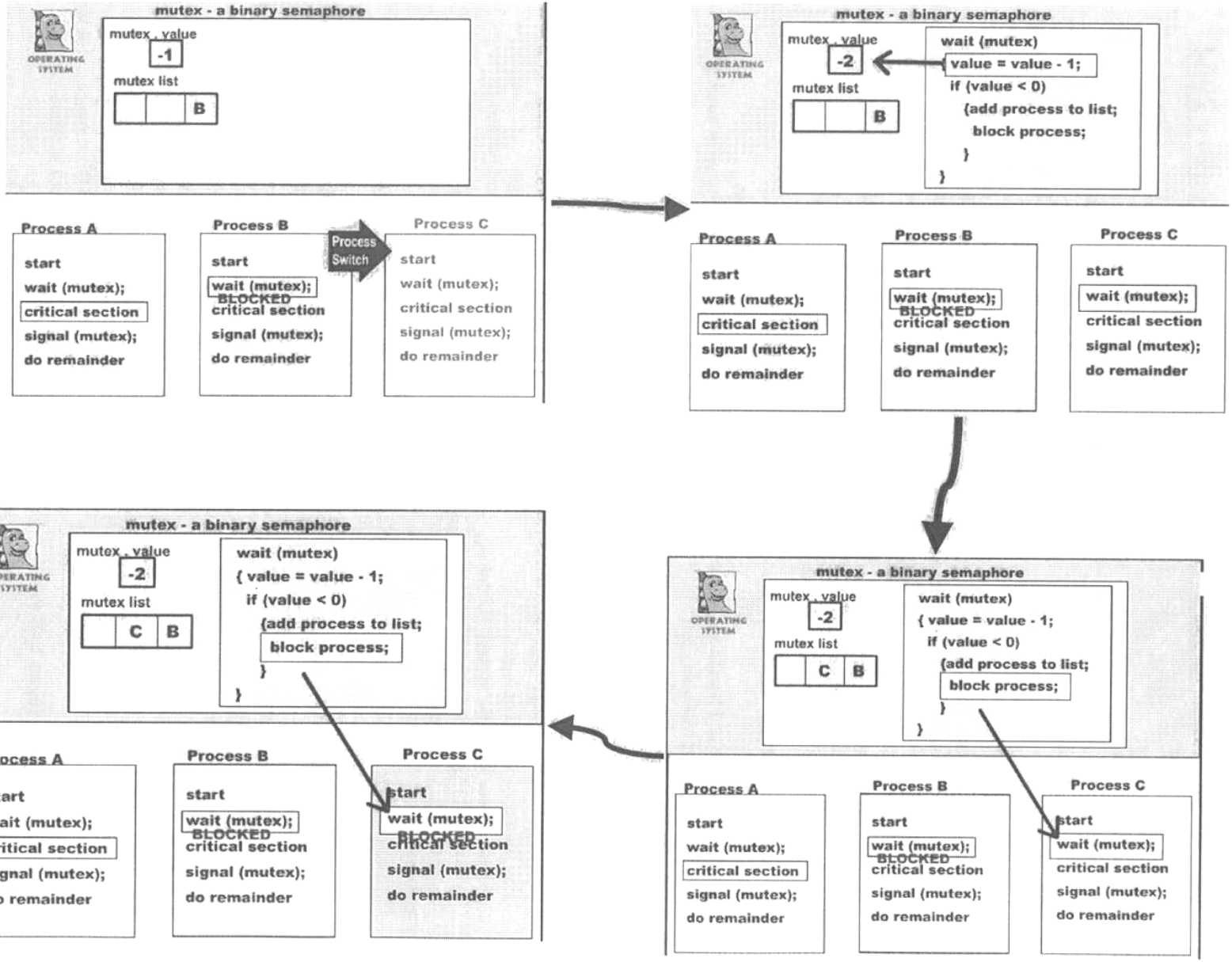
Semáforos

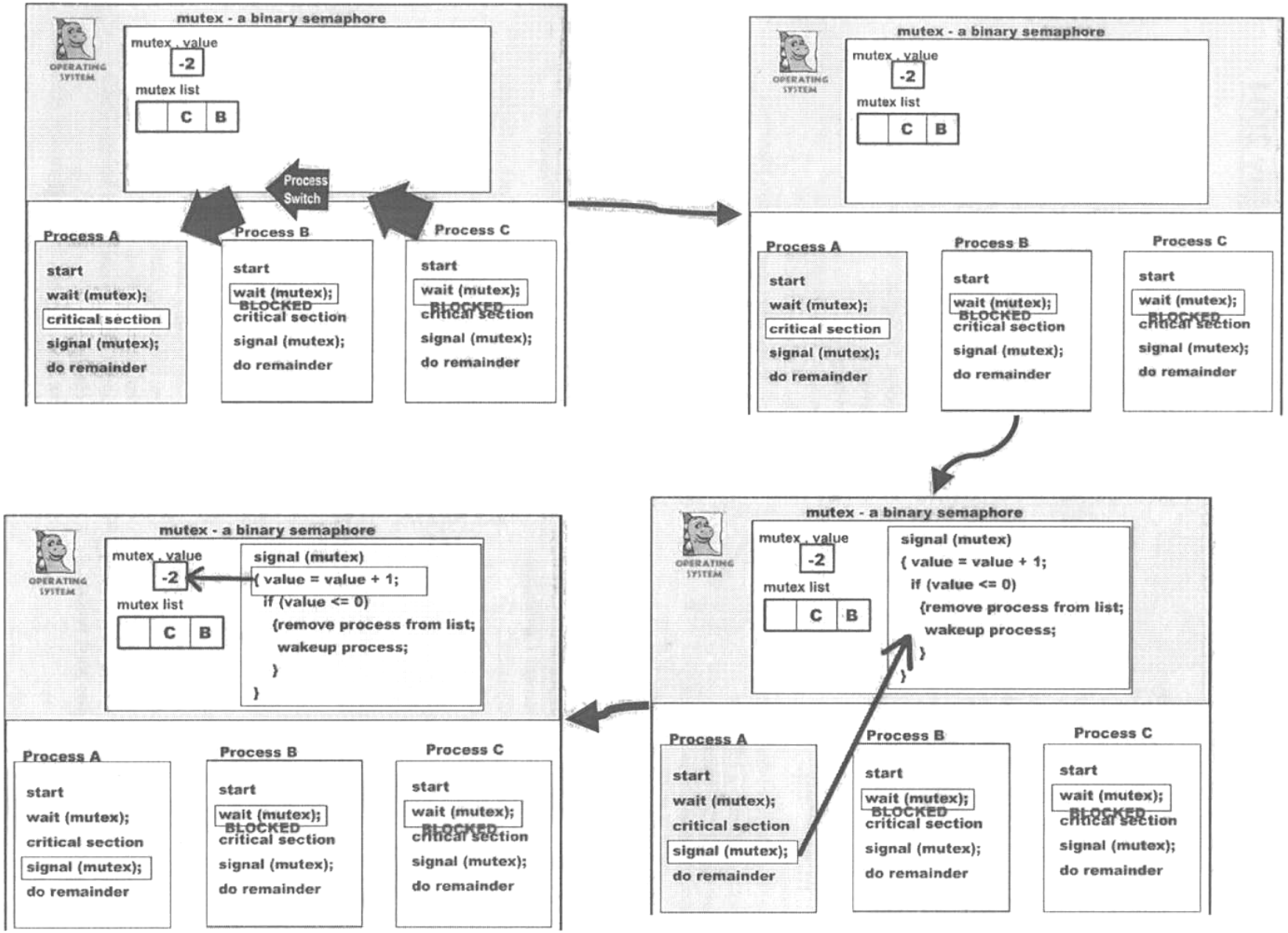

mutex - a binary semaphore
mutex . value
1
mutex list
wait (mutex)
{ value = value - 1;
if (value < 0)
{add process to list;
block process;
}
}
OPERATING SYSTEM
Process A
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process B
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process C
start
wait (mutex);
critical section
signal (mutex);
do remainder
0

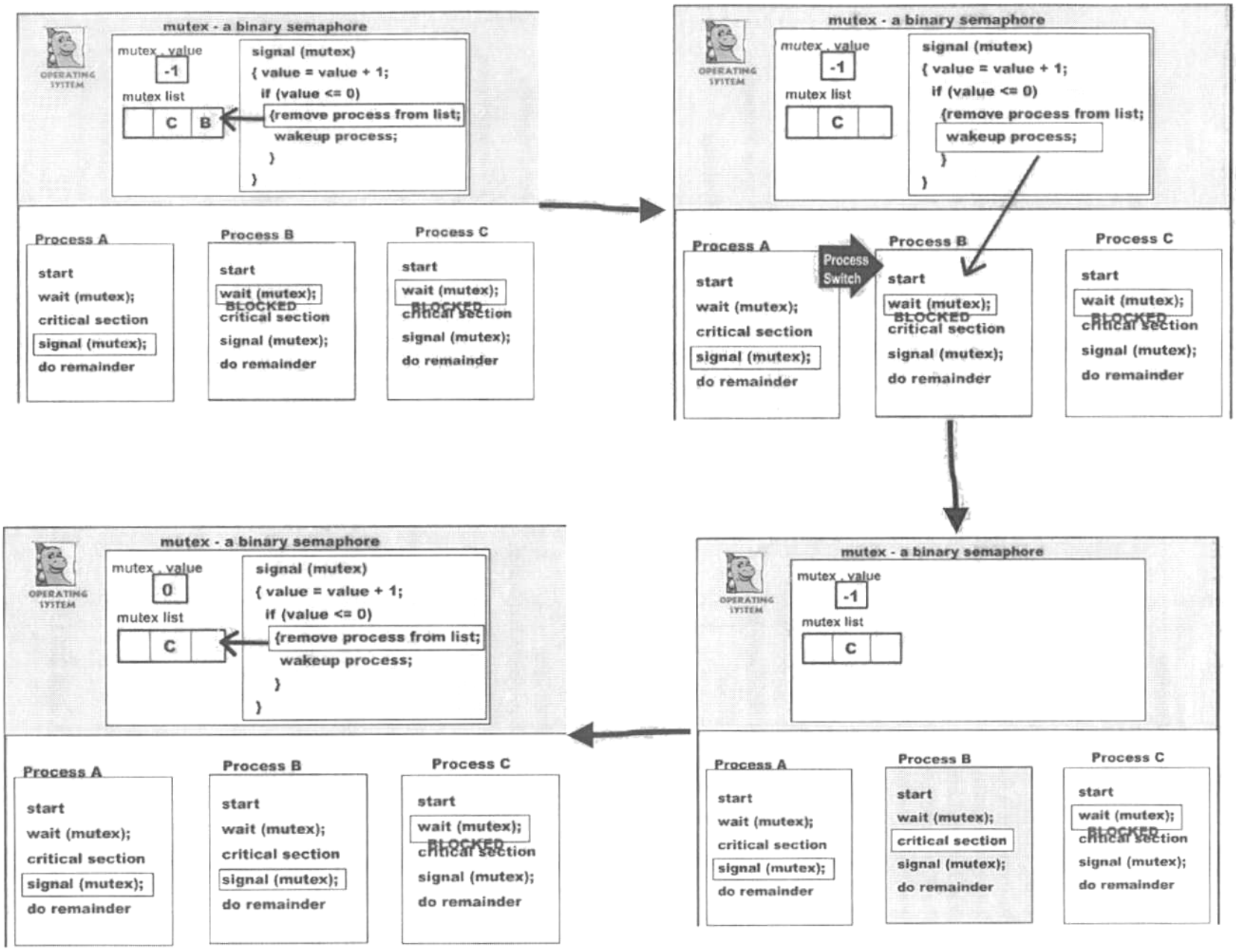

mutex - a binary semaphore
mutex . value
0
mutex list
OPERATING SYSTEM
Process A
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process B
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process C
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process Switch
wait (mutex)
{ value = value - 1;
if (value < 0)
{add process to list;
block process;
}
}

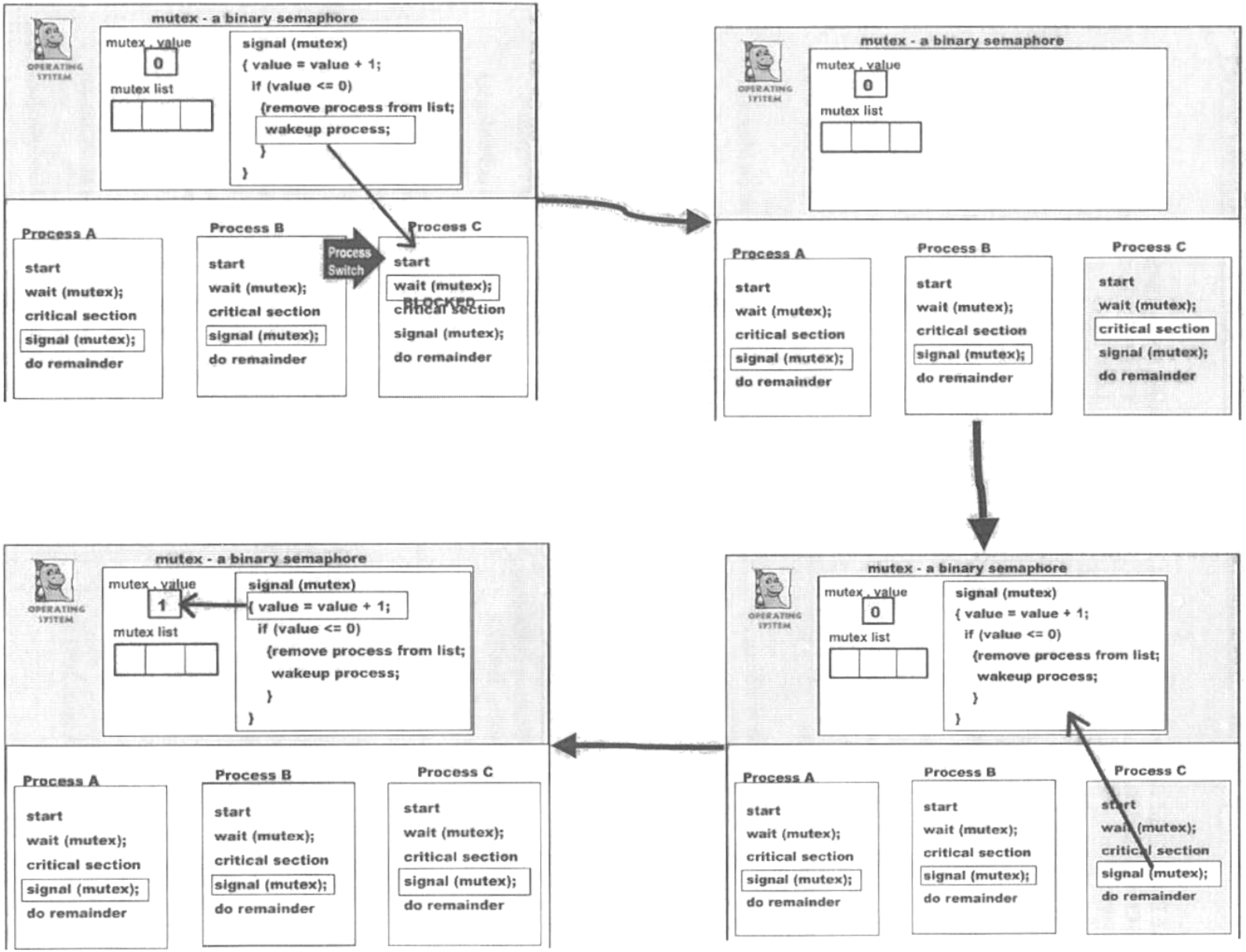


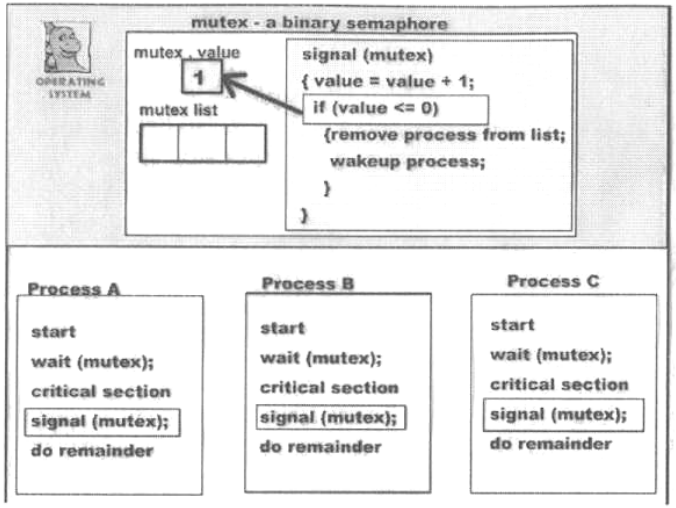
ESPERA NÃO ACTIVA


mutex - a binary semaphore
mutex . value
-1
mutex list
B
OPERATING SYSTEM
Process A
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process B
start
wait (mutex);
BLOCKED
critical section
signal (mutex);
do remainder
Process Switch
Process C
start
wait (mutex);
critical section
signal (mutex);
do remainder
mutex - a binary semaphore
mutex . value
-2
mutex list
B
wait (mutex)
value = value - 1;
if (value < 0)
{add process to list;
block process;
}
}
Process A
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process B
start
wait (mutex);
BLOCKED
critical section
signal (mutex);
do remainder
Process C
start
wait (mutex);
critical section
signal (mutex);
do remainder
mutex - a binary semaphore
mutex . value
-2
mutex list
C
B
wait (mutex)
{ value = value - 1;
if (value < 0)
{add process to list;
block process;
}
}
Process A
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process B
start
wait (mutex);
BLOCKED
critical section
signal (mutex);
do remainder
Process C
start
wait (mutex);
BLOCKED
critical section
signal (mutex);
do remainder
mutex - a binary semaphore
mutex . value
-2
mutex list
C
B
wait (mutex)
{ value = value - 1;
if (value < 0)
{add process to list;
block process;
}
}
Process A
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process B
start
wait (mutex);
BLOCKED
critical section
signal (mutex);
do remainder
Process C
start
wait (mutex);
BLOCKED
critical section
signal (mutex);
do remainder

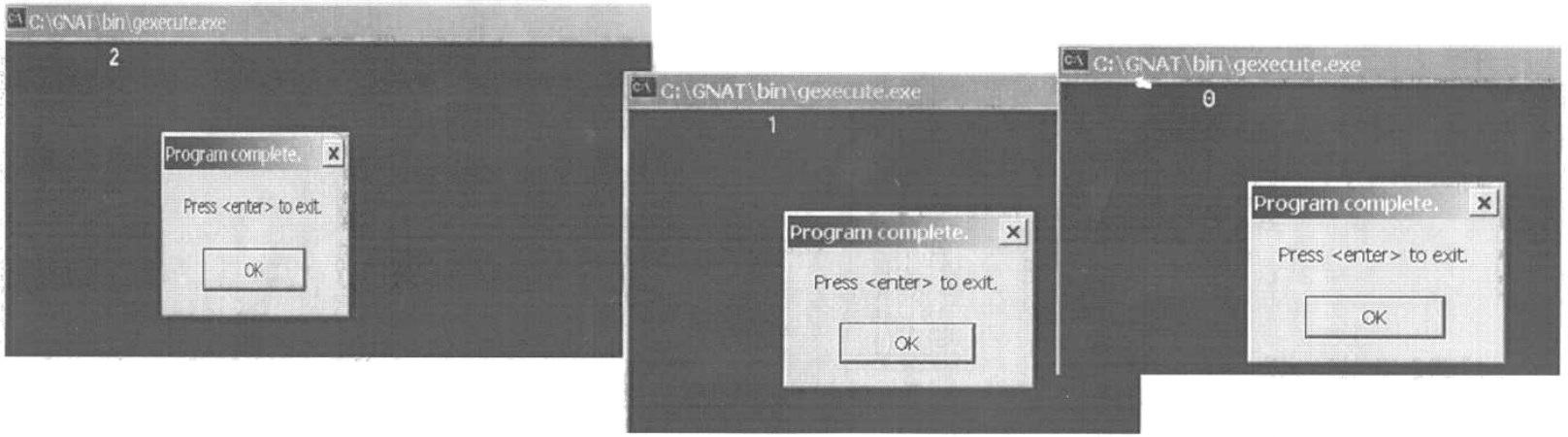

mutex - a binary semaphore
mutex . value
-2
mutex list
C
B
Process Switch
Process A
Process B
Process C
start
wait (mutex);
critical section
signal (mutex);
do remainder
BLOCKED
signal (mutex)
{ value = value + 1;
if (value <= 0)
{remove process from list;
wakeup process;
}
}
OPERATING SYSTEM


mutex - a binary semaphore
mutex value
-1
mutex list
C
B
signal (mutex)
{ value = value + 1;
if (value <= 0)
{remove process from list;
wakeup process;
}
}
Process A
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process B
start
wait (mutex);
BLOCKED
critical section
signal (mutex);
do remainder
Process C
start
wait (mutex);
BLOCKED
critical section
signal (mutex);
do remainder
Process Switch
mutex - a binary semaphore
mutex value
-1
mutex list
C
signal (mutex)
{ value = value + 1;
if (value <= 0)
{remove process from list;
wakeup process;
}
}
Process A
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process B
start
wait (mutex);
BLOCKED
critical section
signal (mutex);
do remainder
Process C
start
wait (mutex);
BLOCKED
critical section
signal (mutex);
do remainder
mutex - a binary semaphore
mutex value
-1
mutex list
C
Process A
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process B
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process C
start
wait (mutex);
BLOCKED
critical section
signal (mutex);
do remainder
mutex - a binary semaphore
mutex value
0
mutex list
C
signal (mutex)
{ value = value + 1;
if (value <= 0)
{remove process from list;
wakeup process;
}
}
Process A
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process B
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process C
start
wait (mutex);
BLOCKED
critical section
signal (mutex);
do remainder


mutex - a binary semaphore
mutex . value
0
mutex list
signal (mutex)
{ value = value + 1;
if (value <= 0)
{remove process from list;
wakeup process;
}
}
Process A
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process B
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process C
start
wait (mutex);
BLOCKED
critical section
signal (mutex);
do remainder
Process Switch
mutex - a binary semaphore
mutex . value
0
mutex list
Process A
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process B
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process C
start
wait (mutex);
critical section
signal (mutex);
do remainder
mutex - a binary semaphore
mutex . value
0
mutex list
signal (mutex)
{ value = value + 1;
if (value <= 0)
{remove process from list;
wakeup process;
}
}
Process A
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process B
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process C
start
wait (mutex);
critical section
signal (mutex);
do remainder
mutex - a binary semaphore
mutex . value
1
mutex list
signal (mutex)
{ value = value + 1;
if (value <= 0)
{remove process from list;
wakeup process;
}
}
Process A
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process B
start
wait (mutex);
critical section
signal (mutex);
do remainder
Process C
start
wait (mutex);
critical section
signal (mutex);
do remainder



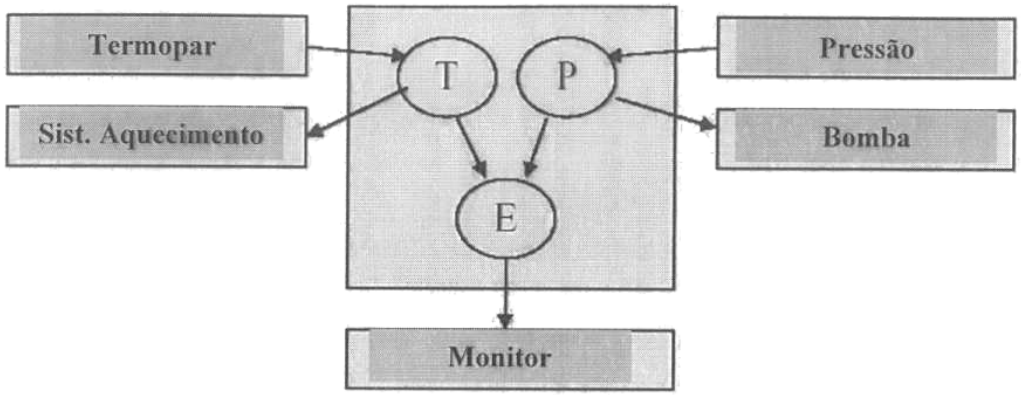

```
with Ada.Integer_Text_Io;
use Ada.Integer_Text_Io;
procedure Shared_Data is

   N : Integer := 1;

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

   task Decrement;
   task body Decrement is
   begin
      N := N - 1;
   end Decrement;

begin
   Put (N);
end Shared_Data;
```



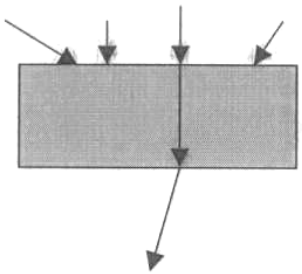
```
with Ada.Integer_Text_Io, Semaphores;
use Ada.Integer_Text_Io, Semaphores;
procedure Shared_Data is

   N : Integer := 1;
   S:Semaphore;

   task Increment;
   task body Increment is
   begin
      Wait(S);
      N := N + 1;
      Signal(S);
   end Increment;

   task Decrement;
   task body Decrement is
   begin
      Wait(S);
      N := N - 1;
      Signal(S);
   end Decrement;
begin
   Initial (S, 1);
   delay 2.0;
   Put (N);
end Shared_Data;
```

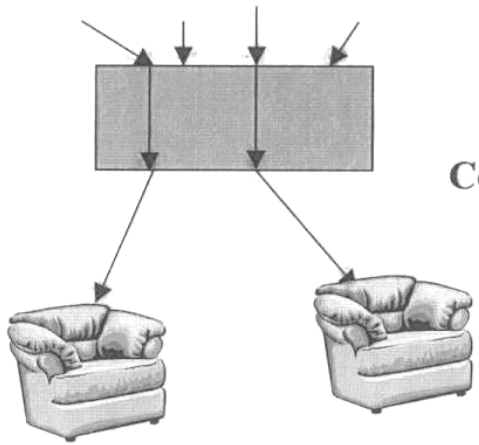
Outras utilizações para os semáforos

Sincronização



Exclusão Mútua

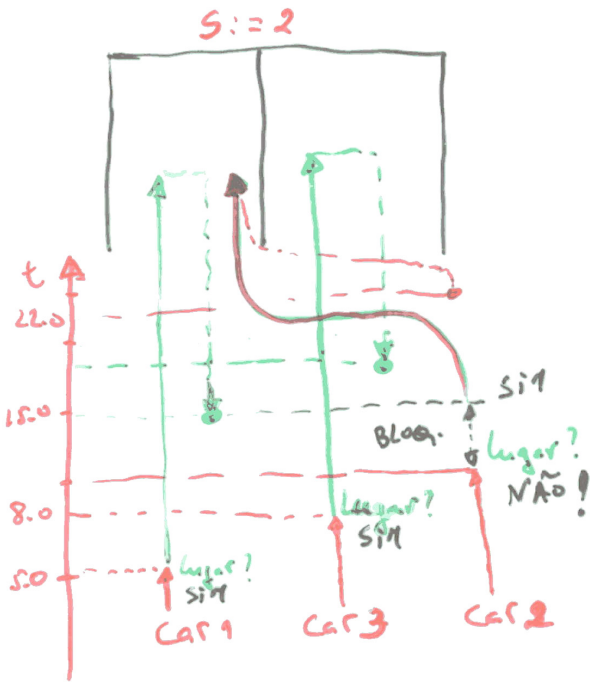




Controlo de recursos

Exclusão Mútua ⟶ **Caso anterior...**

Controlo recursos → Ex.



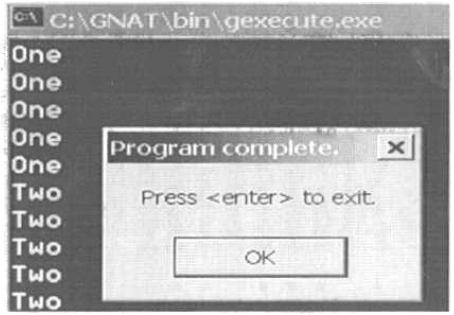

```
with TEXT_IO, SEMAPHORES;
use TEXT_IO, SEMAPHORES;
procedure EXAMPLE is

  SPACES : SEMAPHORE;                    --> Declaração do semáforo usado na alocação

  task type CAR (START, PARKED : DURATION);
  task body CAR (START, PARKED : DURATION) is
  begin
    delay (START);
    WAIT (SPACES);                       --> Chama a operação WAIT quando pretende usar o recurso.
    delay (PARKED);
    SIGNAL (SPACES);                     --> Chama a operação SIGNAL quando deixa de usar o recurso.
  end CAR;

  CAR_1 : CAR (5.0, 10.0);
  CAR_2 : CAR (10.0, 12.0);
  CAR_3 : CAR (8.0, 10.0);

begin
  INITIAL (SPACES, 2);                   --> Inicialização do semáforo com o número de recursos disponíveis
end EXAMPLE;
```

Sincronização → Ex.

```
with Text_Io, Semaphores;
use Text_Io, Semaphores;
procedure Example is

   Screen : Semaphore;
   task Two;
   task body Two is
   begin
      Wait (Screen);
      for I in 1..5 loop
         Put_Line ("Two");
      end loop;
      Signal (Screen);
   end Two;

   task One;
   task body One is
   begin
      Wait (Screen);
      for I in 1..5 loop
         Put_Line ("One");
      end loop;
      Signal (Screen);
   end One;
begin
   Initial (Screen, 1);
end Example;
```

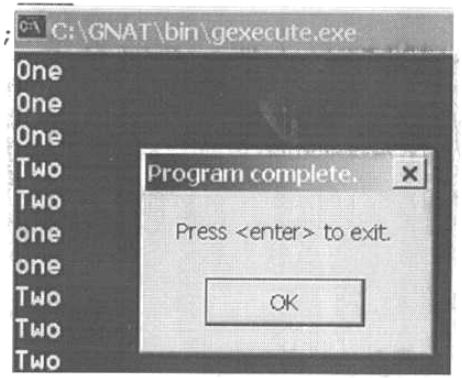
```
with Text_Io, Semaphores;
use Text_Io, Semaphores;
procedure Example is

   Sinc1: Semaphore;

   task Two;
   task body Two is
   begin
      Wait (Sinc1);
      for I in 1..5 loop
         Put_Line ("Two");
      end loop;
   end Two;

   task One;
   task body One is
   begin
      for I in 1..5 loop
         Put_Line ("One");
      end loop;
      Signal (Sinc1);
   end One;

begin
   Initial (Sinc1, 0);
end Example;
```

one
⋮
one
two
⋮
two



Sempre

```
with Text_Io, Semaphores;
use Text_Io, Semaphores;
procedure Example is

   Sinc1, Sinc2: Semaphore;

   task Two;
   task body Two is
   begin
      Wait (Sinc1);
      for I in 1..2 loop
         Put_Line ("Two");
      end loop;
      Signal (Sinc2);
      Wait (Sinc1);
      for I in 1..3 loop
         Put_Line ("Two");
      end loop;
   end Two;

   task One;
      task body One is
      begin
         for I in 1..3 loop
            Put_Line ("One");
         end loop;
         Signal (Sinc1);
         Wait (Sinc2);
         for I in 1..2 loop
            Put_Line ("one");
         end loop;
         Signal (Sinc1);
      end One;
begin
   Initial (Sinc1, 0);
   Initial (Sinc2,0);
end Example;
```



Os problemas dos semáforos

Signal (s); ... Signal (S) → Não existe exclusão mútua

Wait (s); ... Wait (S) → Deadlock

Signal (S); ... Wait (s) → Depende

CONFUSÃO DE PROPÓSITOS

PEQUENO SIGNIFICADO SEMÂNTICO

```
with Ada.Integer_Text_Io, Semaphores;
use Ada.Integer_Text_Io, Semaphores;
procedure Shared_Data is

   N : Integer := 1;
   S:Semaphore;

   task Increment;
   task body Increment is
   begin
      Wait(S);
      N := N + 1;
      Signal(S);
   end Increment;

   task Decrement;
   task body Decrement is
   begin
      Wait(S);
      N := N - 1;
      Signal(S);
   end Decrement;
begin
   Initial (S, 0);
   delay 2.0;
   Put (N);
end Shared_Data;
```