

Grupo ARCOS  
Departamento de Informática  
Universidad Carlos III de Madrid

# Lección 3b

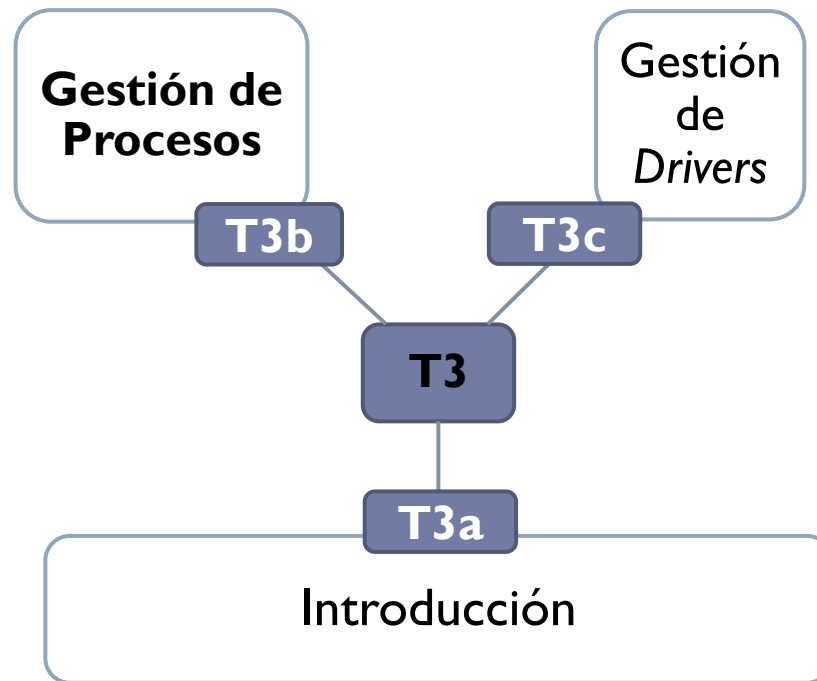
procesos, periféricos, *drivers* y servicios ampliados

Diseño de Sistemas Operativos  
Grado en Ingeniería Informática y Doble Grado I.I. y A.D.E.



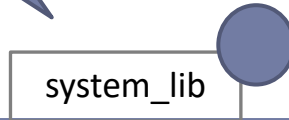
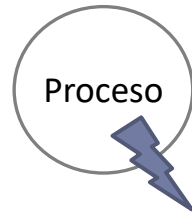
# Contexto...

---



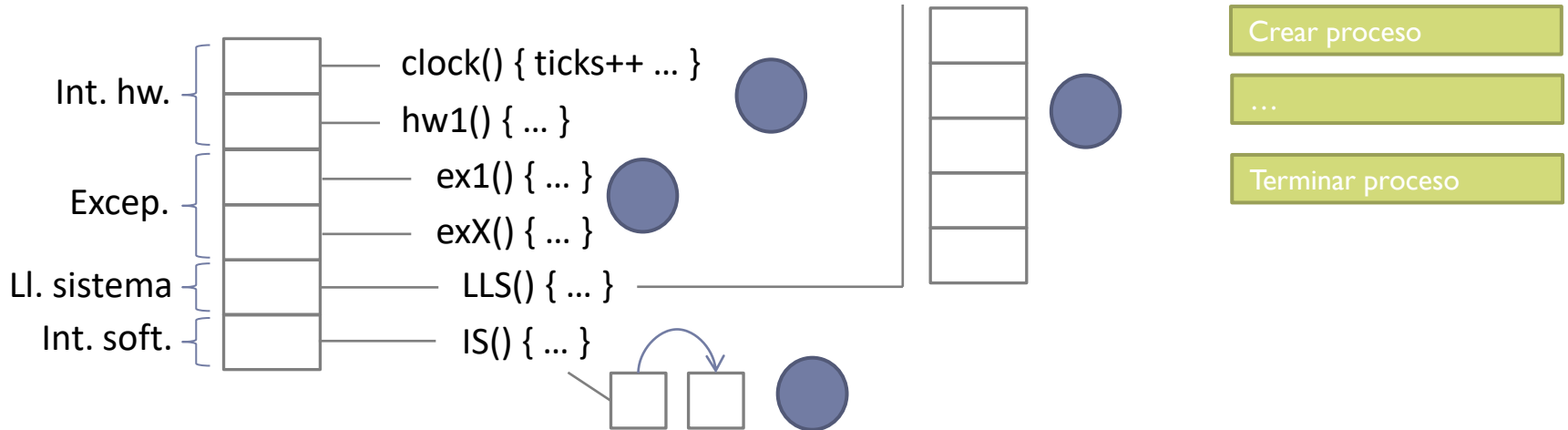
# Objetivos...

En el tema 3 se introducirá los aspectos del funcionamiento relativos a la **gestión de procesos...**



U

K



Periférico

# A recordar...

---

Antes de clase

Clase

Después de clase


Preparar los pre-requisitos.

Estudiar el material asociado a la **bibliografía**:  
las transparencias solo no son suficiente.  
Preguntar dudas (especialmente tras estudio).

Ejercitar las competencias:

- ▶ Realizar todos los **ejercicios**.
- ▶ Realizar los **cuadernos de prácticas** y las **prácticas** de forma progresiva.

# Ejercicios, cuadernos de prácticas y prácticas

	Ejercicios ✓	Cuadernos de prácticas ✓	Prácticas ✓
b	<p>© copyright all rights reserved</p> <p>Grado en Ingeniería Informática Diseño de Sistemas Operativos [3b] Planificación y procesos</p> <p>ARCOS</p> <p>Grupo: ____ NIA: _____ Nombre y apellidos: _____</p> <p><b>Ejercicio 1</b></p> <p>Considérese un sistema operativo que usa un algoritmo de planificación de procesos <i>round-robin</i> con una rodaja de 100 ms. Supóngase que se quiere compararlo con un algoritmo de planificación expulsiva por prioridades en el que cada proceso de usuario tenga una prioridad estática fijada en su creación. Dado el siguiente fragmento de programa, se pide analizar su comportamiento usando el planificador original y, a continuación, hacerlo con el nuevo modelo de planificación planteado. Para cada modelo de planificación, se deberá especificar la secuencia de ejecución de ambos procesos (se tendrán en cuenta sólo estos procesos) hasta que, o bien un proceso llame a la función P2 o bien el otro llame a P4.</p> <p>NOTA: La escritura en una tubería no bloquea al escritor a no ser que la tubería esté llena (situación que no se da en el ejemplo). Además, en este análisis se supondrá que a ninguno de los dos procesos se les termina el cuanto de ejecución.</p> <p>...</p>	<p><b>DISEÑO DE SISTEMAS OPERATIVOS</b></p> <p>GRADO EN INGENIERÍA INFORMÁTICA DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y ADMINISTRACIÓN DE EMPRESAS</p> <p>uc3m   Universidad Carlos III de Madrid</p>	<p>DISEÑO DE SISTEMAS OPERATIVOS</p> <p>GRADO EN INGENIERÍA INFORMÁTICA</p> 
c	<p>© copyright all rights reserved</p> <p>Grado en Ingeniería Informática Diseño de Sistemas Operativos [3c] Drivers y servicios ampliados</p> <p>ARCOS</p> <p>Grupo: ____ NIA: _____ Nombre y apellidos: _____</p> <p><b>Ejercicio 1</b></p> <p>Una compañía de accesorios informáticos ha creado un ratón y su correspondiente driver para sistema operativo básico (como el que está siendo presentado en pseudocódigo en la asignatura) cuya funcionalidad definida por su interfaz al usuario es:</p> <ul style="list-style-type: none"> <li>• Funciones <i>open/close</i>: Para establecer el acceso al ratón o liberarlo</li> <li>• Función <i>read</i>: Para obtener la posición actual del ratón.</li> </ul> <p>Se ha fabricado una nueva versión del ratón que permite configurar la precisión del mismo indicando la distancia entre posiciones consecutivas. Por tanto, resulta necesario modificar el driver del ratón para añadir dicha funcionalidad.</p> <p>Para realizar esto disponemos de la función:</p> <pre>ModificarPrecision (int valor);</pre>	<p><b>Introducción a un driver de teclado con Linux/Ubuntu</b></p>	<p>UNIVERSIDAD CARLOS III DE MADRID</p> <p><b>Planificación de procesos</b></p> <p>David DEL RÍO ASTORGA</p>

# Lecturas recomendadas

---

## Base



1. Carretero 2007:
  1. Cap.7

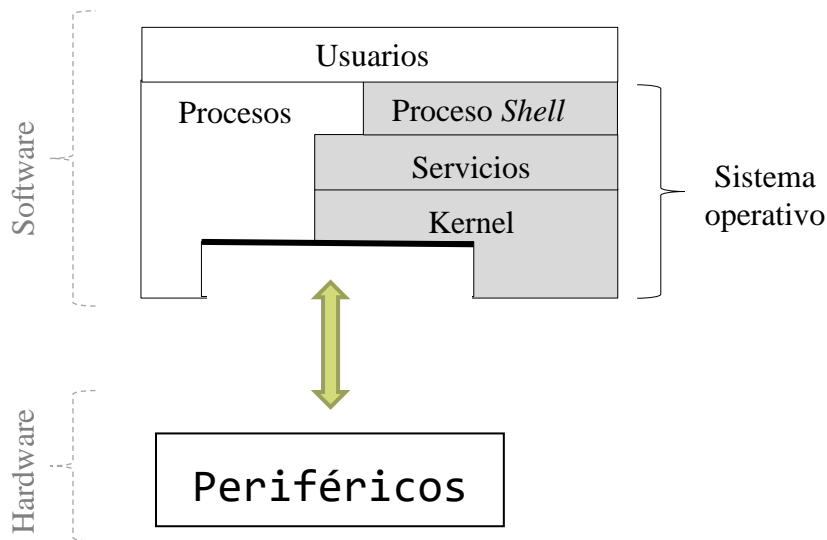
## Recomendada



1. Tanenbaum 2006(en):
  1. Cap.3
2. Stallings 2005(en):
  1. Parte tres
3. Silberschatz 2006:
  1. Cap. Sistemas de E/S

# Contenidos

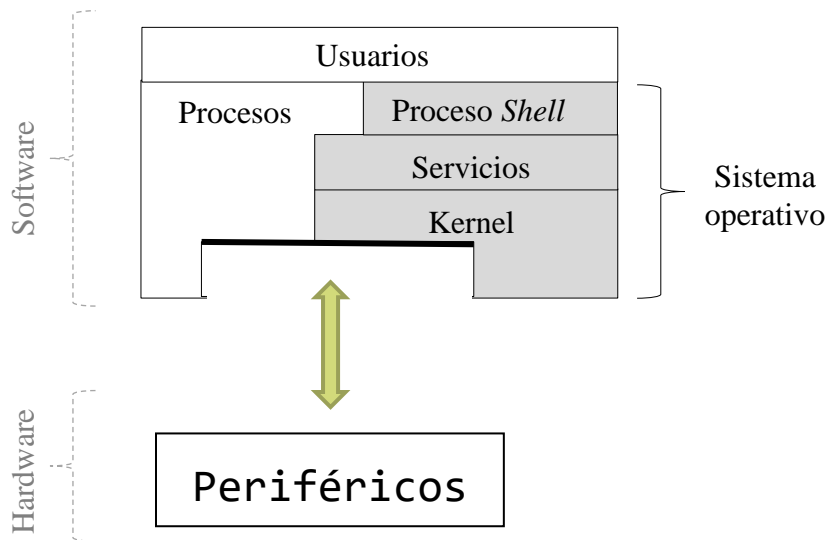
---



- ▶ Introducción
- ▶ C.C.V.
- ▶ Temporización y C.C.I.
- ▶ Planificación

# Contenidos

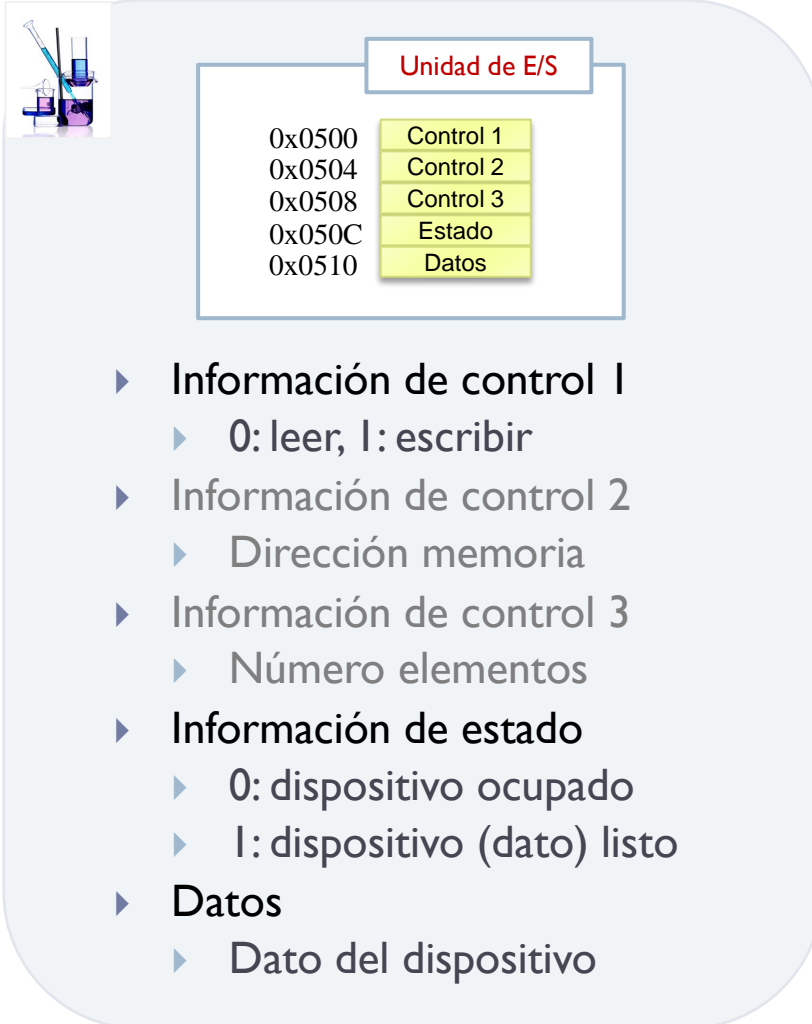
---



- ▶ **Introducción**
- ▶ **C.C.V.**
- ▶ **Temporización y C.C.I.**
- ▶ **Planificación**



# Formas de trabajar con un módulo de E/S



▶ **E/S programada o directa**

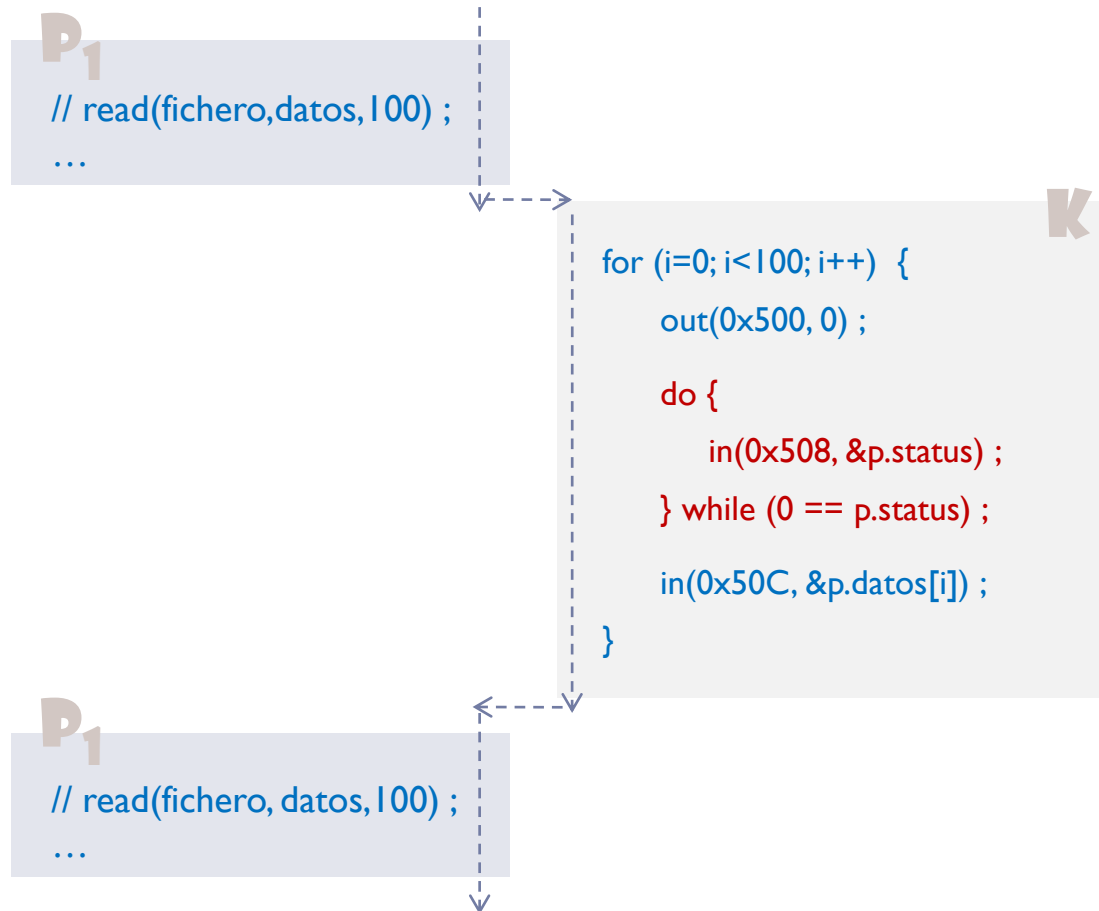
▶ **E/S por interrupciones**

▶ **E/S por DMA**



# Ejemplo

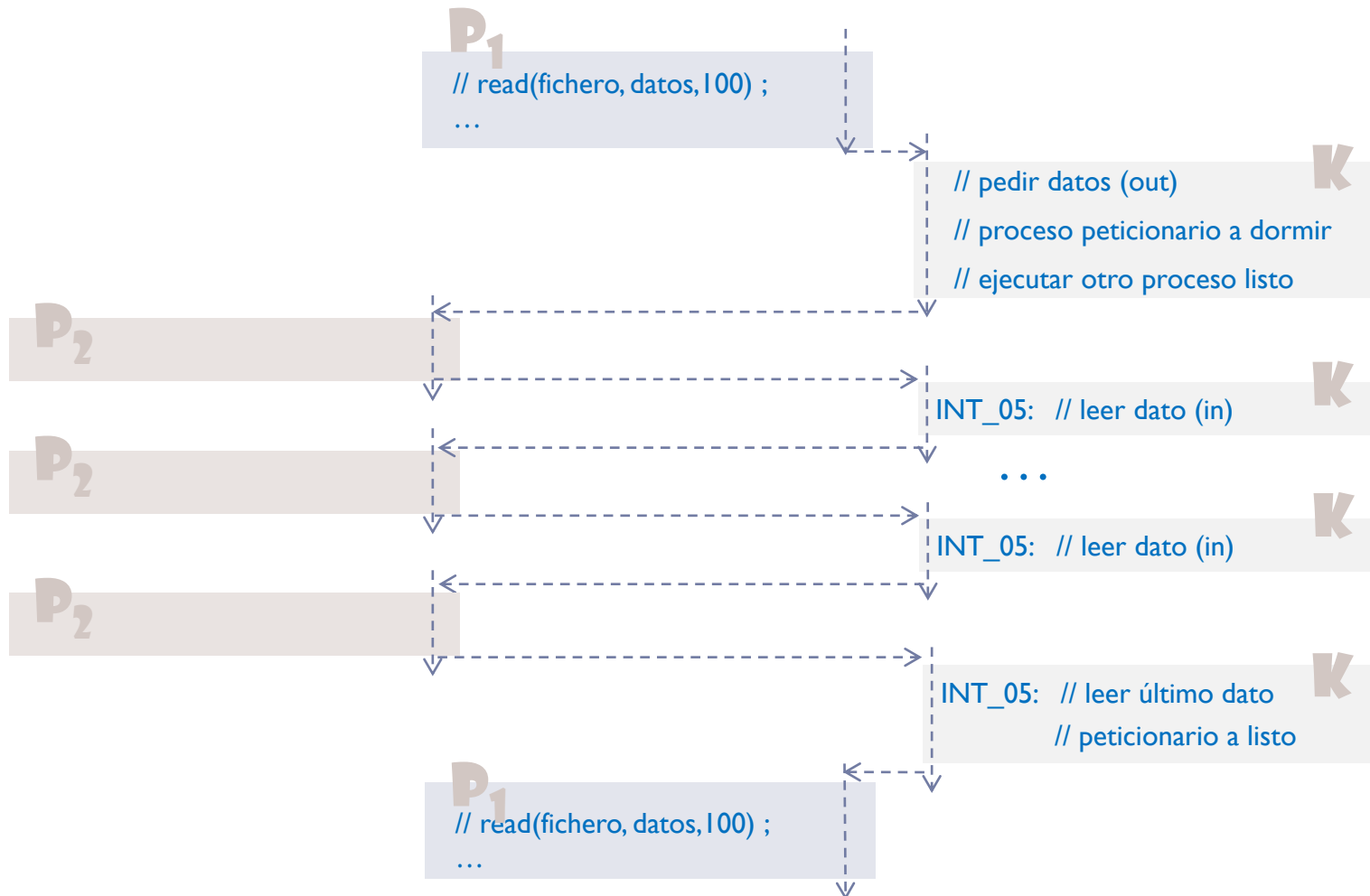
## E/S programada





# Ejemplo

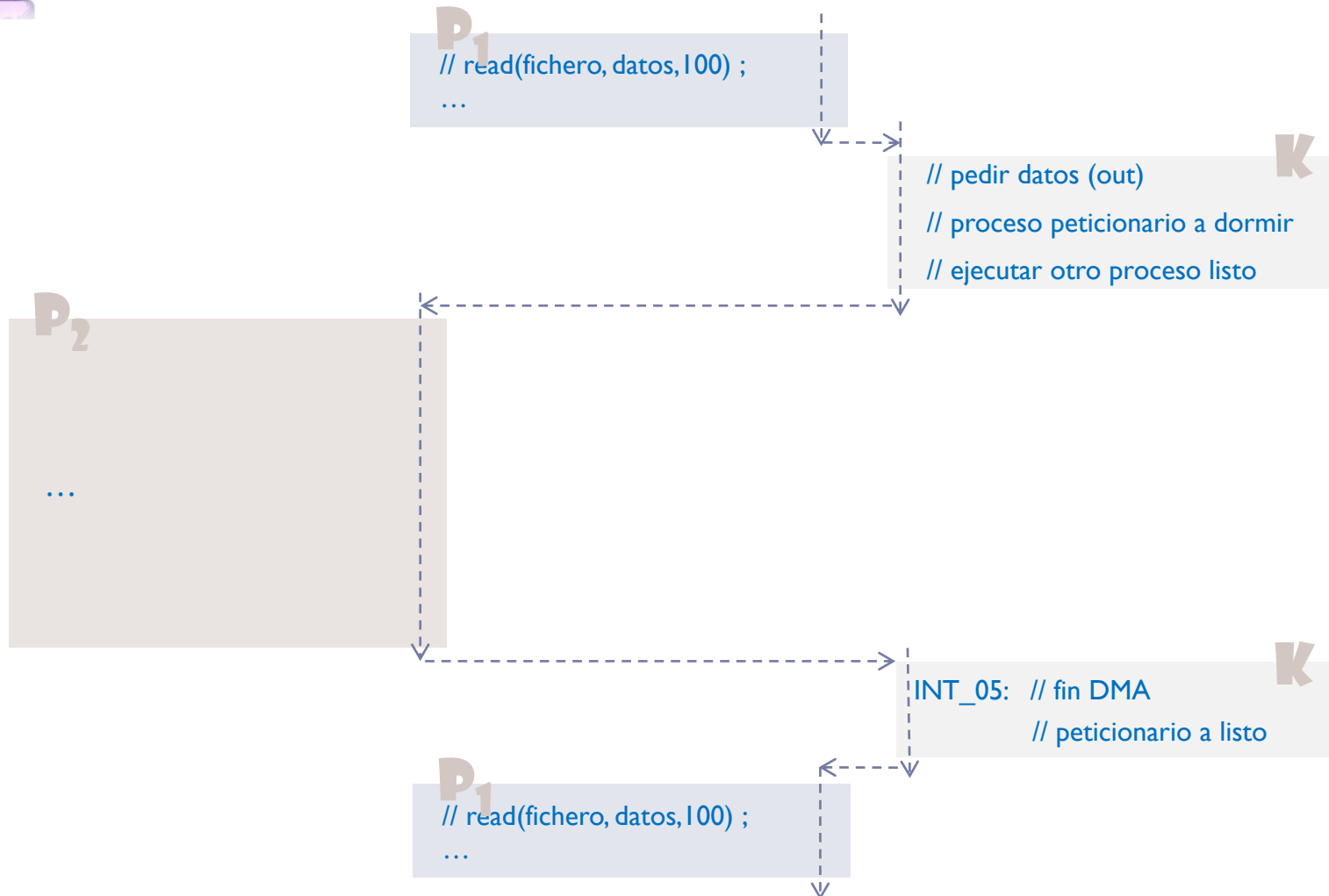
## E/S por interrupciones





# Ejemplo

## E/S por DMA



# Aprovechar mejor los tiempos de espera

## ▶ E/S programada o directa

**Ejemplo**  
E/S programada

24 ARCOS @ UC3M

**Ejemplo**  
E/S programada

```

if petición de lectura a partir de posición 10
out(0x0410); // write
out(0x000); // read
for (i=0; i<100; i++)
{
// bucle de espera
do {
in(0x000); // data?
}while (!in_ready);
// leer datos
in(0x000); // data
}
    
```

- Información de control 1
  - 0 leer
- Información de control 2
  - Posición de la lectura
- Información de estado
  - 0 dispositivo ocupado
  - 1 dispositivo (data) listo
- Datos
  - Dato del dispositivo

25 ARCOS @ UC3M

**Ejemplo**  
E/S programada

26 ARCOS @ UC3M

## ▶ E/S por interrupciones

**Ejemplo**  
E/S por interrupciones

27 ARCOS @ UC3M

**Ejemplo**  
E/S por interrupciones

```

if petición de lectura a partir de posición 10
// primer I/O: posición 10
out(0x0410); // write
out(0x000); // read
// primer proceso periférico a dormir
// registrar que se va a procesar los datos
int_00 = in(0x000); // leer estado
in(0x000); // leer dato
// segundo I/O: posición 11
// primer I/O: posición 11
out(0x0411); // write
out(0x000); // read
// segundo proceso periférico a dormir
// registrar que se va a procesar los datos
int_01 = in(0x000); // leer estado
in(0x000); // leer dato
// tercer I/O: posición 12
// tercer proceso periférico a dormir
// registrar que se va a procesar los datos
int_02 = in(0x000); // leer estado
in(0x000); // leer dato
    
```

- Información de control 1
  - 0 leer
  - 1 escribir
- Información de control 2
  - Posición de la lectura
- Información de estado
  - 0 dispositivo ocupado
  - 1 dispositivo (data) listo
- Datos
  - Dato del dispositivo

28 ARCOS @ UC3M

**Ejemplo**  
E/S por interrupciones

29 ARCOS @ UC3M

## ▶ E/S por DMA

**Ejemplo**  
E/S por DMA

Coordinación entre CPU y Módulos de E/S para escribir a memoria

30 ARCOS @ UC3M

**Ejemplo**  
E/S por DMA

```

out(0x000); // leer
out(0x000); // dirección memoria
out(0x000); // nº elementos
// primer proceso periférico a dormir
// registrar que se va a procesar los datos
int_00 = in(0x000); // leer estado
in(0x000); // leer dato
// primer proceso periférico a dormir
// registrar que se va a procesar los datos
int_01 = in(0x000); // leer estado
in(0x000); // leer dato
// segundo proceso periférico a dormir
// registrar que se va a procesar los datos
int_02 = in(0x000); // leer estado
in(0x000); // leer dato
    
```

- Información de control 1
  - 0 leer
  - 1 escribir
- Información de control 2
  - Posición de la lectura
- Información de estado
  - 0 dispositivo ocupado
  - 1 dispositivo (data) listo
- Datos
  - Dato del dispositivo

31 ARCOS @ UC3M

**Ejemplo**  
E/S por DMA

32 ARCOS @ UC3M



# Ejemplo

## E/S programada, por interrupciones y por DMA

### petición:

```
for (i=0; i<100; i++)
{
    // leer siguiente
    out(0x500, 0);

    // bucle de espera
    do {
        in(0x508, &p.status);
    } while (0 == p.status);

    // leer dato
    in(0x50C, &(p.datos[i]));
}
```

### petición:

```
p.contador = 0;
p.neltos = 100;
out(0x500, 0);
// C.C.V.
```

### INT\_05:

```
in(0x508, &(p.status));
in(0x50C, &(p.datos[p.contador]));
if ( (p.contador < p.neltos) &&
    (p.status == OK)
    {
        p.contador++;
        out(0x500, 0); // leer
    }
else { // proceso peticionario a listo }
ret_int # restore registers & return
```

### petición:

```
out(0x500, 0);
out(0x504, p.datos);
out(0x508, 100);
// C.C.V.
```

### INT\_05:

```
// leer estado y datos
in(0x50C, &status);

if (p.status...

// proceso peticionario a listo
ret_int # restore registers & return
```



# Ejemplo

## E/S programada, por interrupciones y por DMA

### petición:

```
for (i=0; i<100; i++)
{
    // leer siguiente
    out(0x500, 0);

    // bucle de espera
    do {
        in(0x508, &p.status);
    } while (0 == p.status);

    // leer dato
    in(0x50C, &(p.datos[i]));
}
```

### petición:

```
p.contador = 0;
p.neltos = 100;
out(0x500, 0);
// C.C.V.
```

### INT\_05:

```
in(0x508, &(p.status));
in(0x50C, &(p.datos[p.contador]));
if ( (p.contador < p.neltos) &&
      (p.status == OK)
    )
{
    p.contador++;
    out(0x500, 0); // leer
}
else { // proceso peticionario a listo }
ret_int # restore registers & return
```

### petición:

```
out(0x500, 0);
out(0x504, p.datos);
out(0x508, 100);
// C.C.V.
```

### INT\_05:

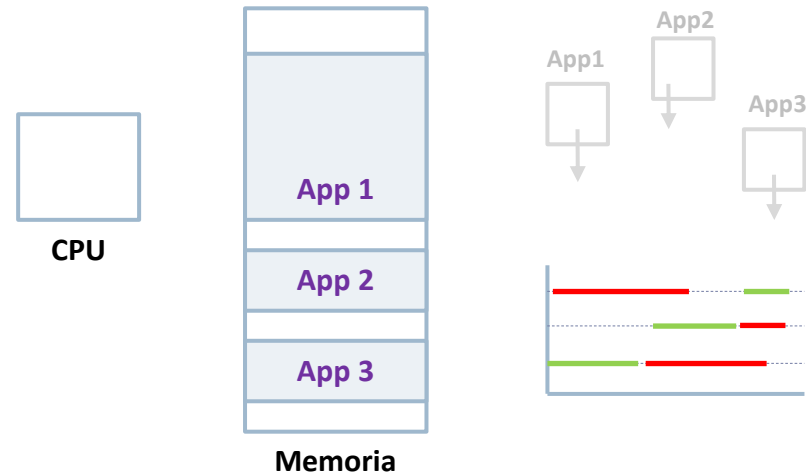
```
// leer estado y datos
in(0x50C, &status);

if (p.status...

// proceso peticionario a listo
ret_int # restore registers & return
```

# Modelo ofrecido

- recursos
- **multiprogramación**
  - protección/compartición
  - jerarquía de procesos
- multitarea
- multiproceso



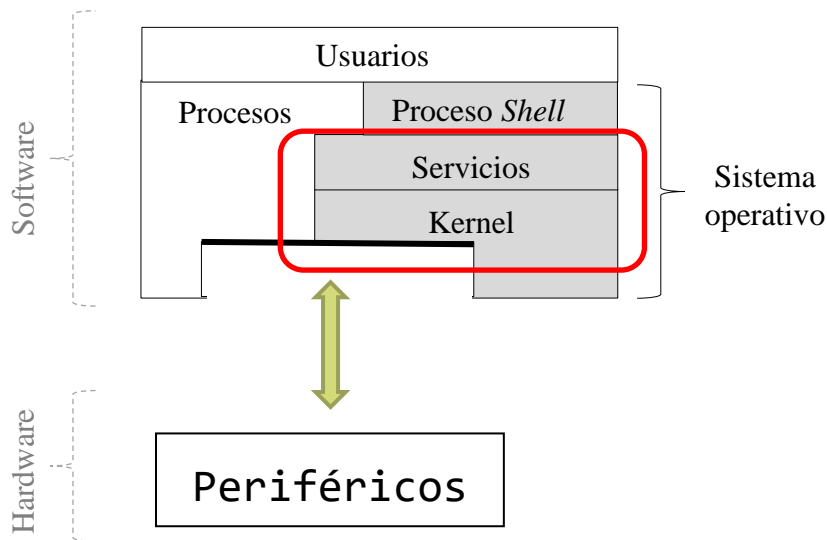
## ▶ Multiprogramación

- ▶ Tener varias aplicaciones en memoria
- ▶ Si una aplicación se bloquea por E/S, entonces se ejecuta mientras otra hasta que quede bloqueada
  - ▶ Cambio de contexto voluntario (C.C.V.)
- ▶ Eficiencia en el uso del procesador
- ▶ Grado de multiprogramación = número de aplicaciones en RAM



# Contenidos

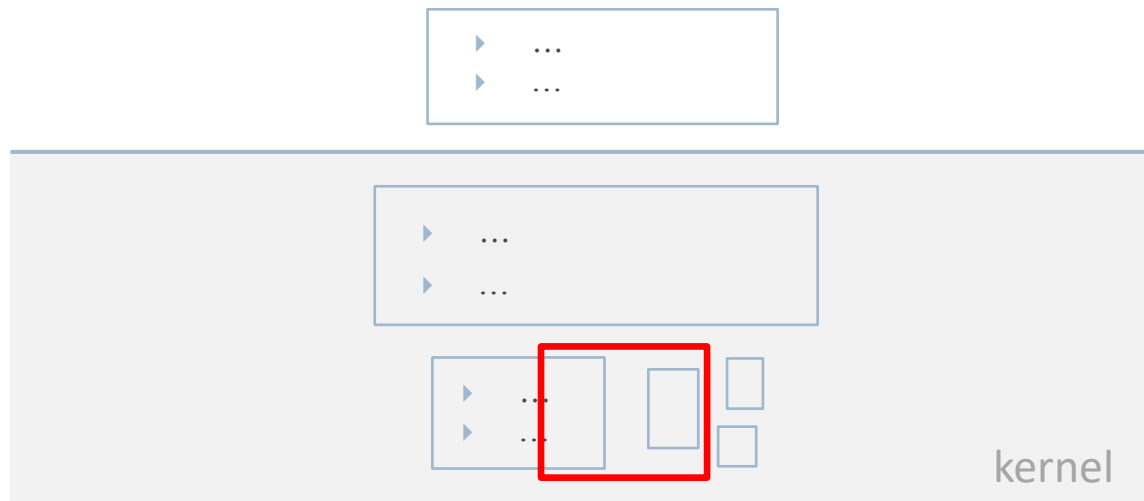
---



- ▶ Introducción
- ▶ **C.C.V.**
- ▶ Temporización y C.C.I.
- ▶ Planificación

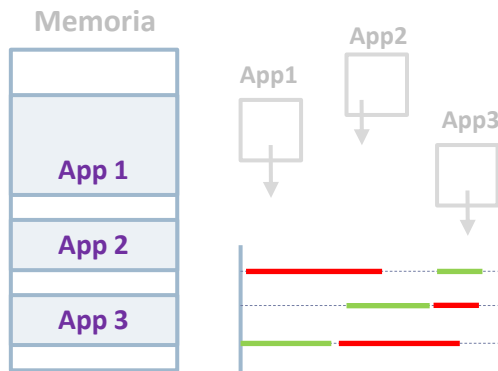
# Multiprogramación (datos y funciones)

Requisitos	Información (en estructuras de datos)	Funciones (internas, servicio y API)
Multiprogramación	<ul style="list-style-type: none"><li>• Estado de ejecución</li><li>• Contexto: registros de CPU...</li><li>• Lista de procesos</li></ul>	<ul style="list-style-type: none"><li>• Int. hw/sw de dispositivos</li><li>• Planificador</li><li>• Crear/Destruir/Planificar proceso</li></ul>



# Multiprogramación

---

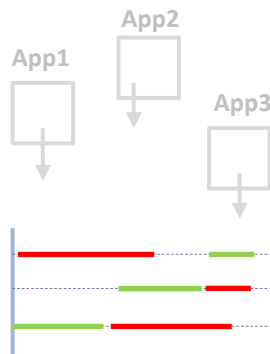
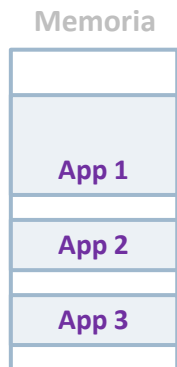
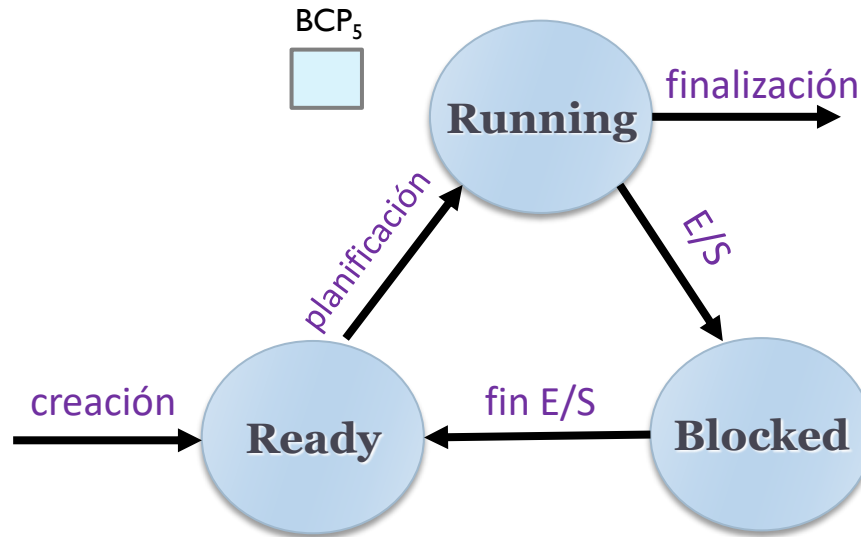


- ▶ Tener varias aplicaciones en memoria
- ▶ Si una aplicación se bloquea por E/S, entonces se ejecuta otra (hasta que quede bloqueada)
  - ▶ Cambio de contexto voluntario (C.C.V.)

# Multiprogramación (datos)

## Estados de un proceso (c.c.v.)

- Estado
- Lista/Cola
- Contexto

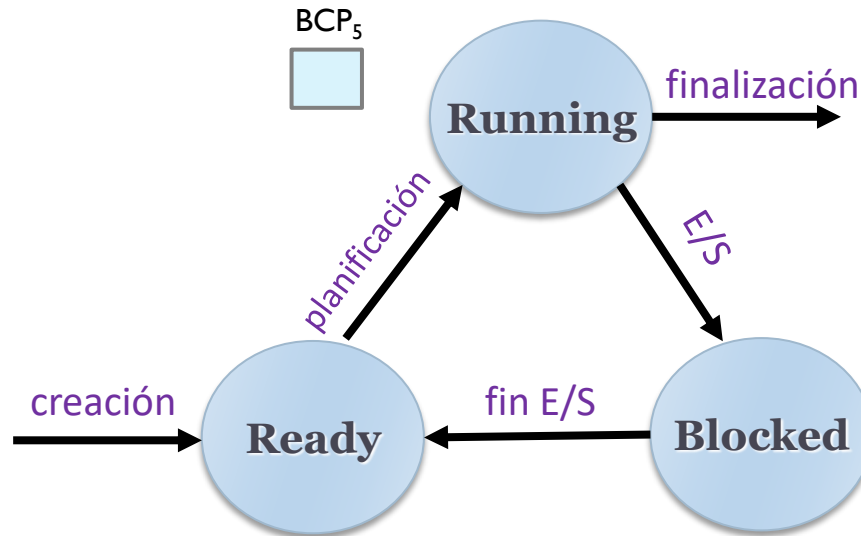


- ▶ Tener varias aplicaciones en memoria
- ▶ Si una aplicación se bloquea por E/S, entonces se ejecuta otra (hasta que quede bloqueada)
  - ▶ Cambio de contexto voluntario (C.C.V.)

# Multiprogramación (datos)

## Estados de un proceso (c.c.v.)

- Estado
- Lista/Cola
- Contexto

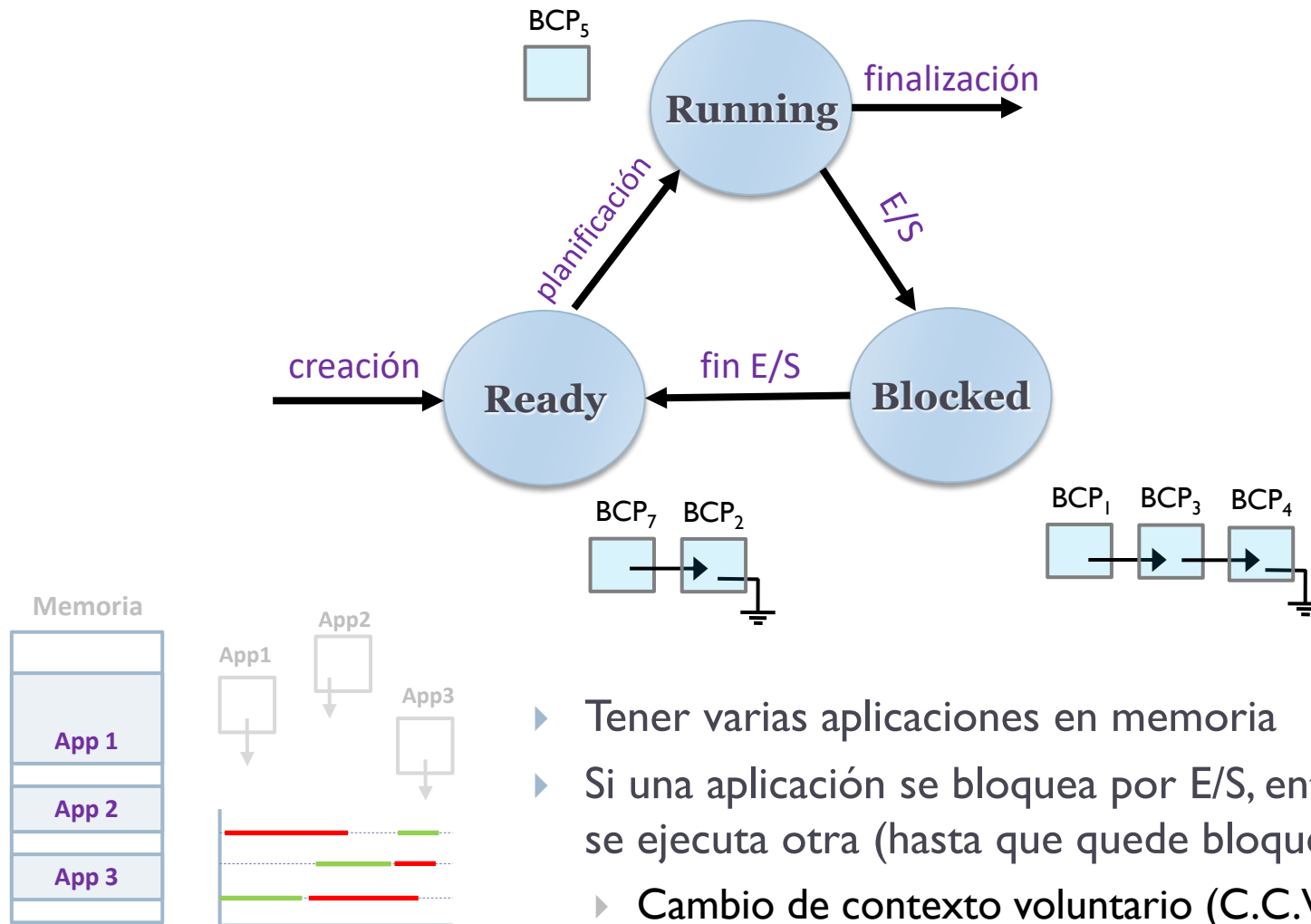


- ▶ **En ejecución:** con CPU asignada
- ▶ **Listo para ejecutar:** no procesador disponible para él
- ▶ **Bloqueado:** esperando un evento
- ▶ **Suspendido y listo:** expulsado pero listo para ejecutar
- ▶ **Suspendido y bloqueado:** expulsado y esperando evento

# Multiprogramación (datos)

## Lista/Colas de procesos (c.c.v.)

- Estado
- Lista/Cola
- Contexto

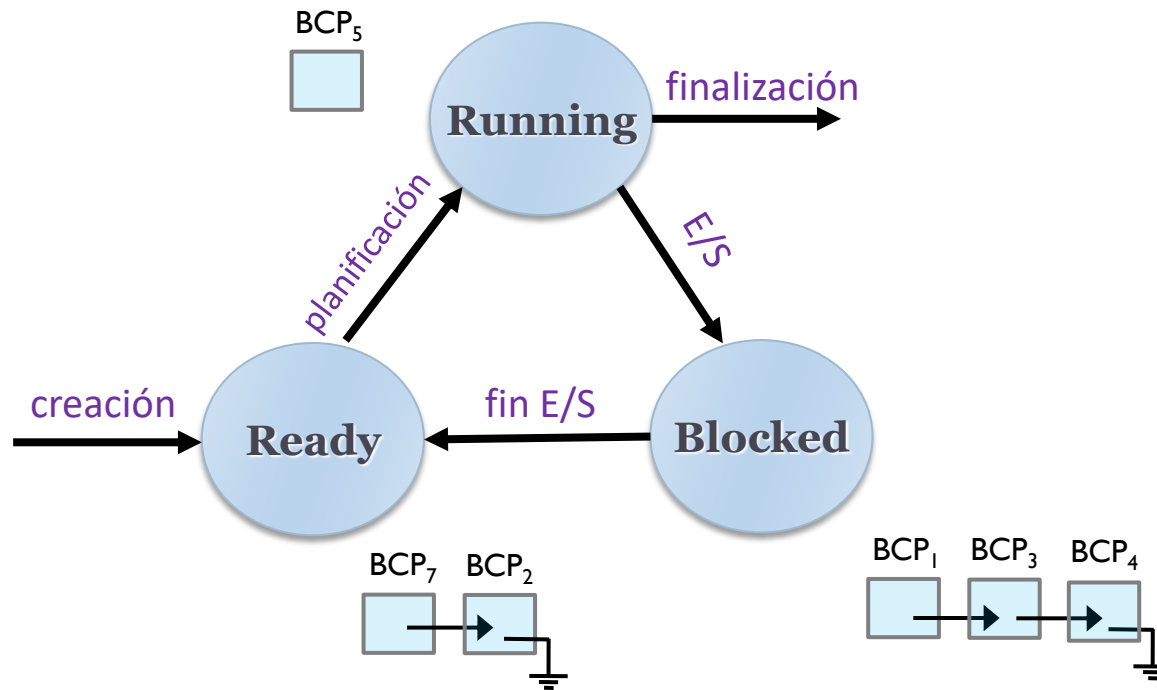


- ▶ Tener varias aplicaciones en memoria
- ▶ Si una aplicación se bloquea por E/S, entonces se ejecuta otra (hasta que quede bloqueada)
  - ▶ Cambio de contexto voluntario (C.C.V.)

# Multiprogramación (datos)

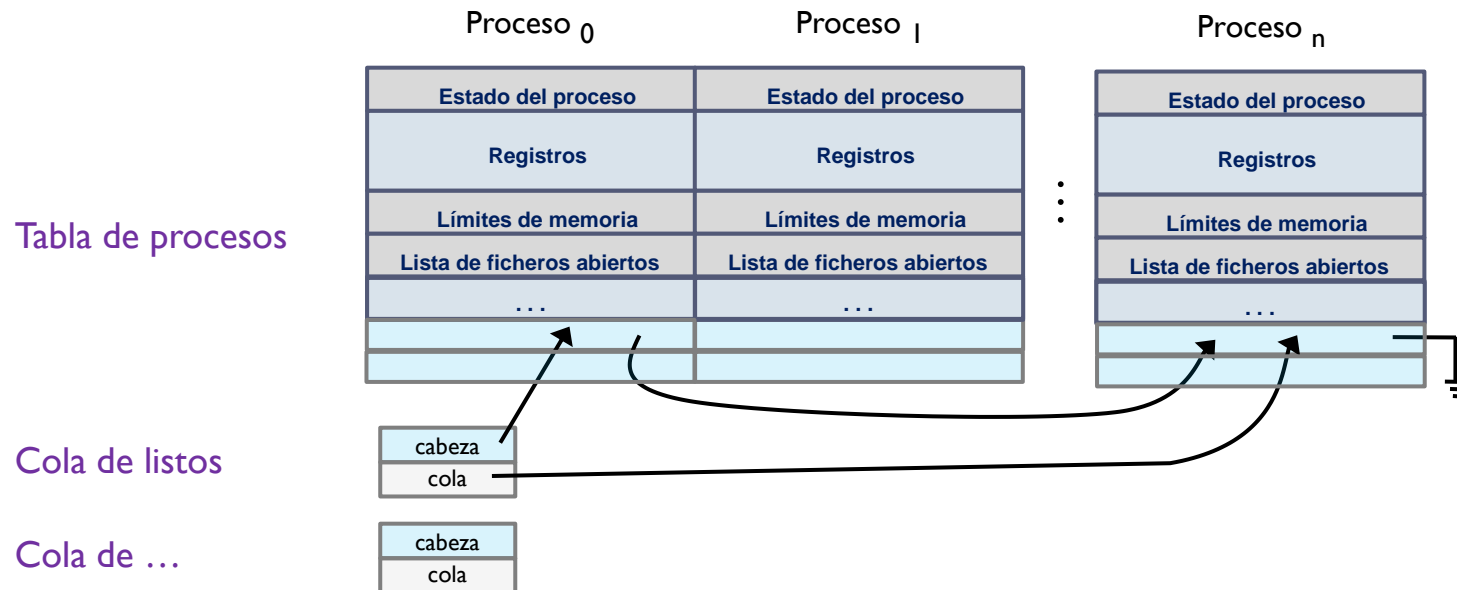
## Lista/Colas de procesos (c.c.v.)

- Estado
- Lista/Cola
- Contexto



- ▶ Cola de listos: procesos esperando a ejecutar en CPU
- ▶ Cola de bloqueados por recurso: procesos a la espera de finalizar una petición bloqueante al recurso asociado
- Un proceso solo puede estar en una cola (como mucho)

# Implementación de las colas de procesos



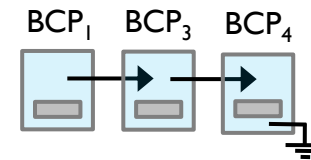
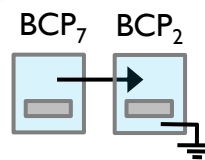
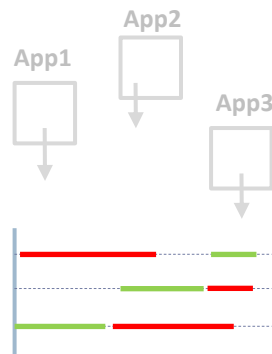
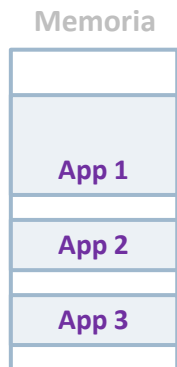
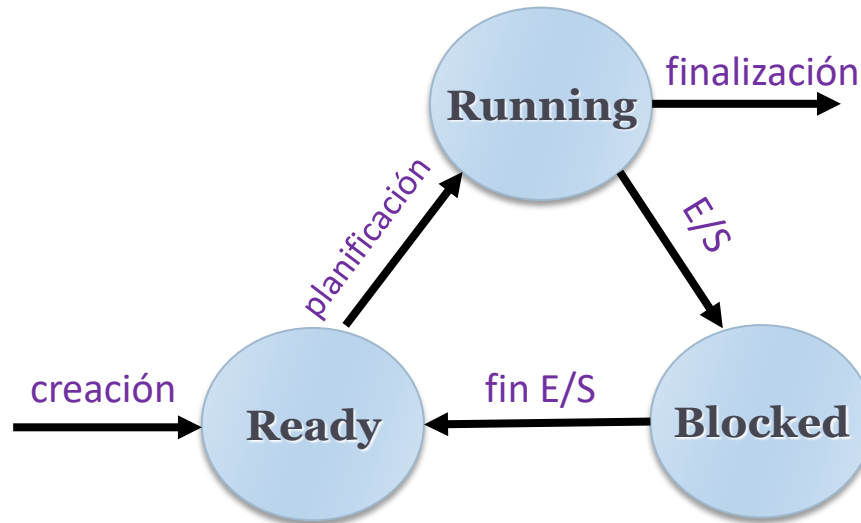
- ▶ Cola de listos: procesos esperando a ejecutar en CPU
- ▶ Cola de bloqueados por recurso: procesos a la espera de finalizar una petición bloqueante al recurso asociado
- Un proceso solo puede estar en una cola (como mucho)



# Multiprogramación (datos)

## Contexto de un proceso

- Estado
- Lista/Cola
- Contexto

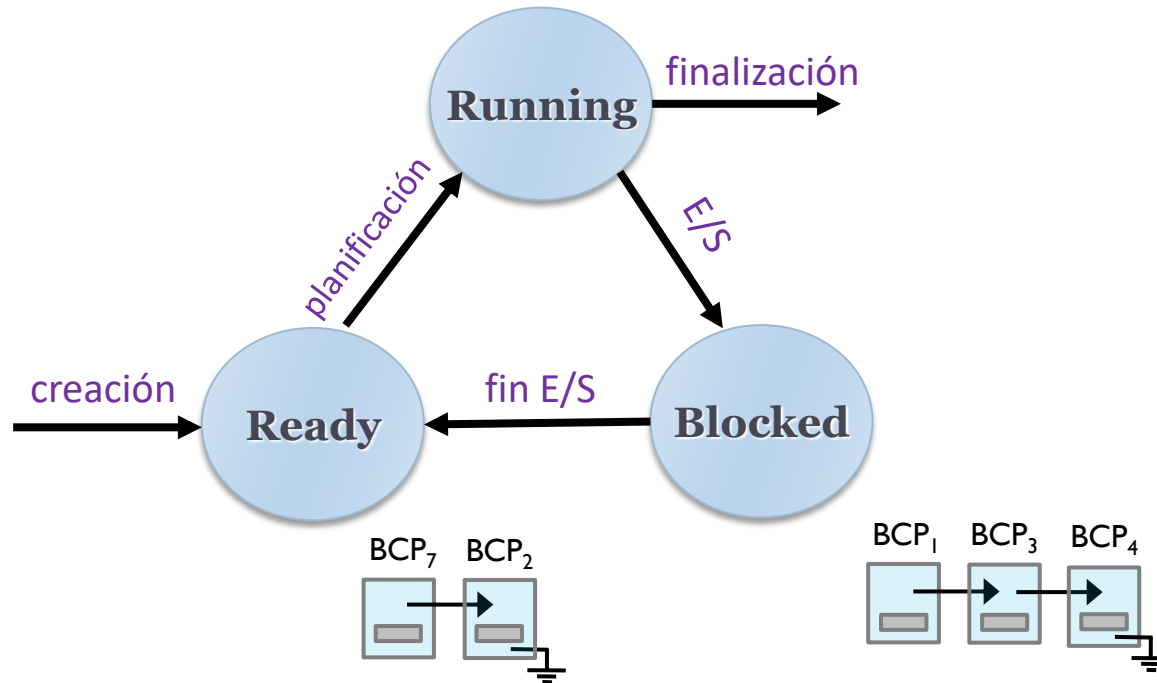


- ▶ Tener varias aplicaciones en memoria
- ▶ Si una aplicación se bloquea por E/S, entonces se ejecuta otra (hasta que quede bloqueada)
  - ▶ Cambio de contexto voluntario (C.C.V.)

# Multiprogramación (datos)

## Contexto de un proceso

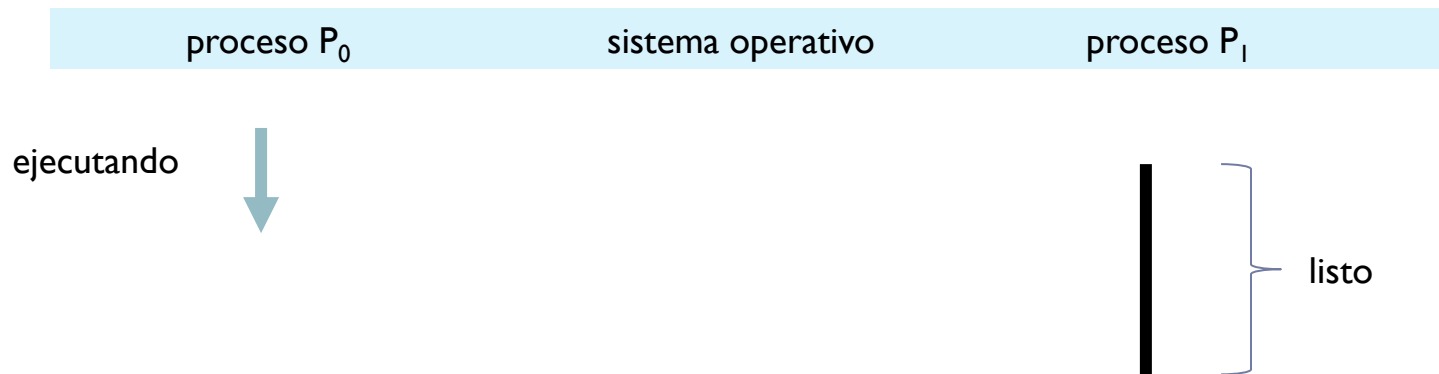
- Estado
- Lista/Cola
- Contexto



- ▶ Registros generales: PC, RE, etc.
- ▶ Registros específicos: Registros de coma flotante, etc.
- ▶ Referencias a recursos: puntero a código, datos, etc.

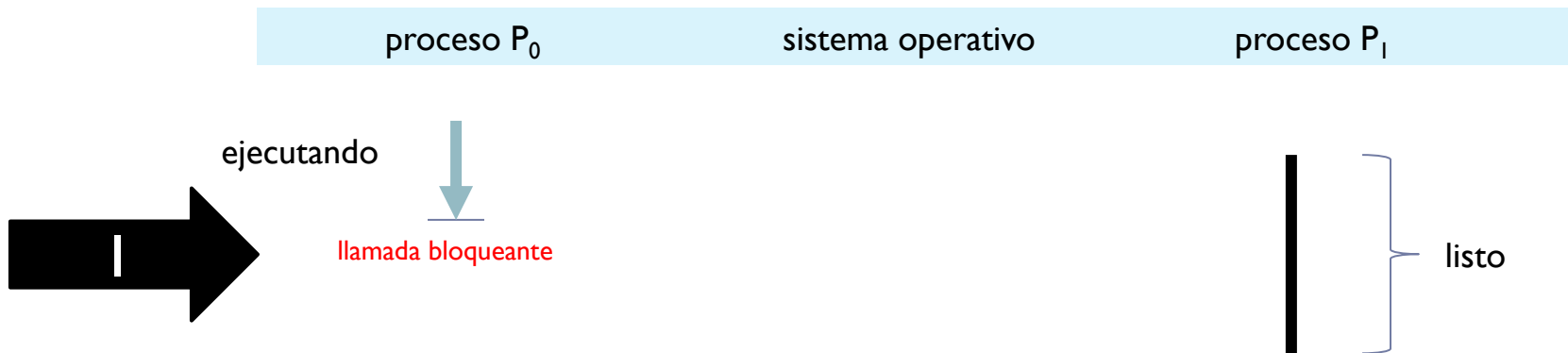
# Multiprogramación: ejemplo de ejecución

---

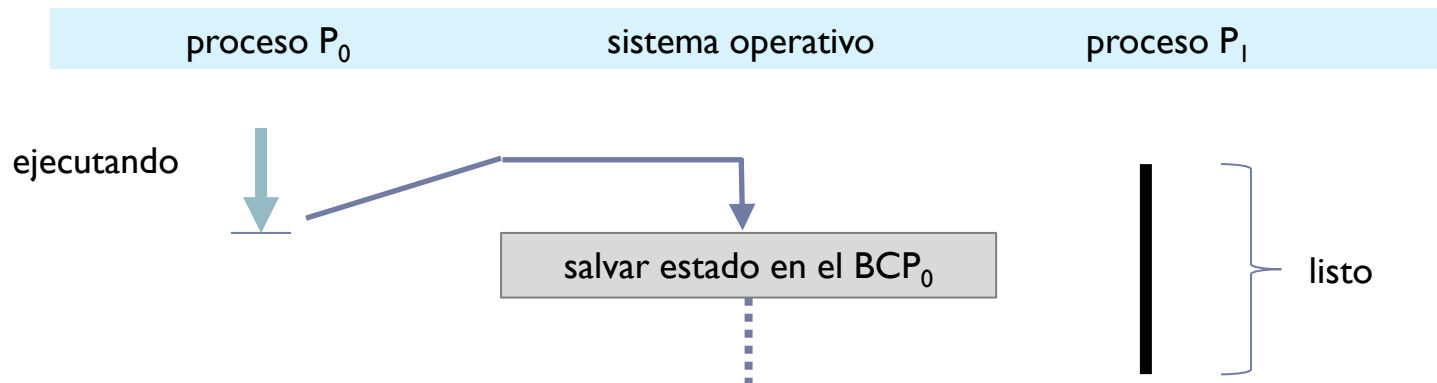


# Multiprogramación: ejemplo de ejecución

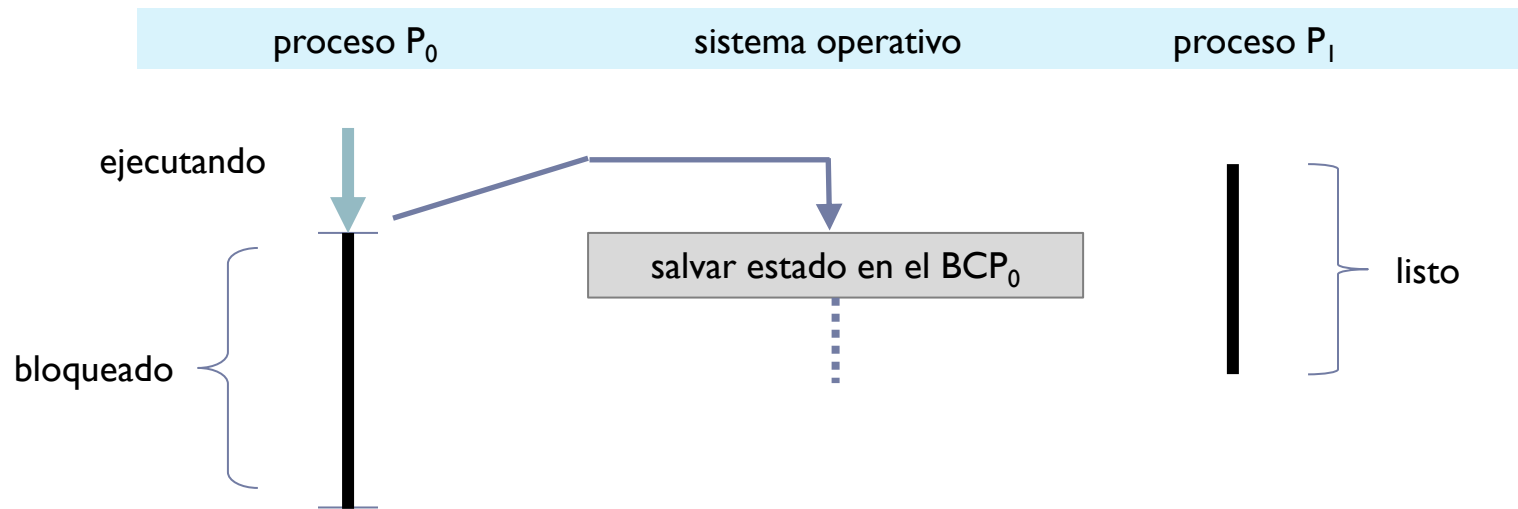
---



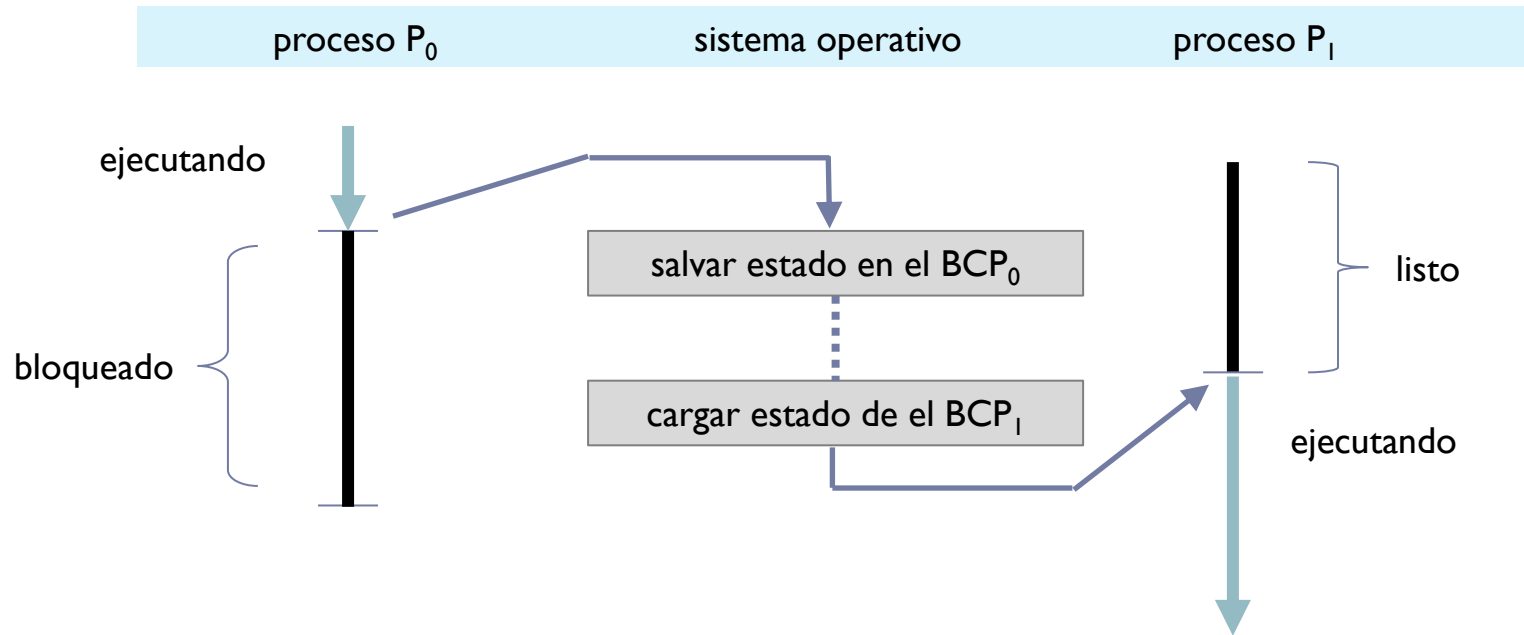
# Multiprogramación: ejemplo de ejecución



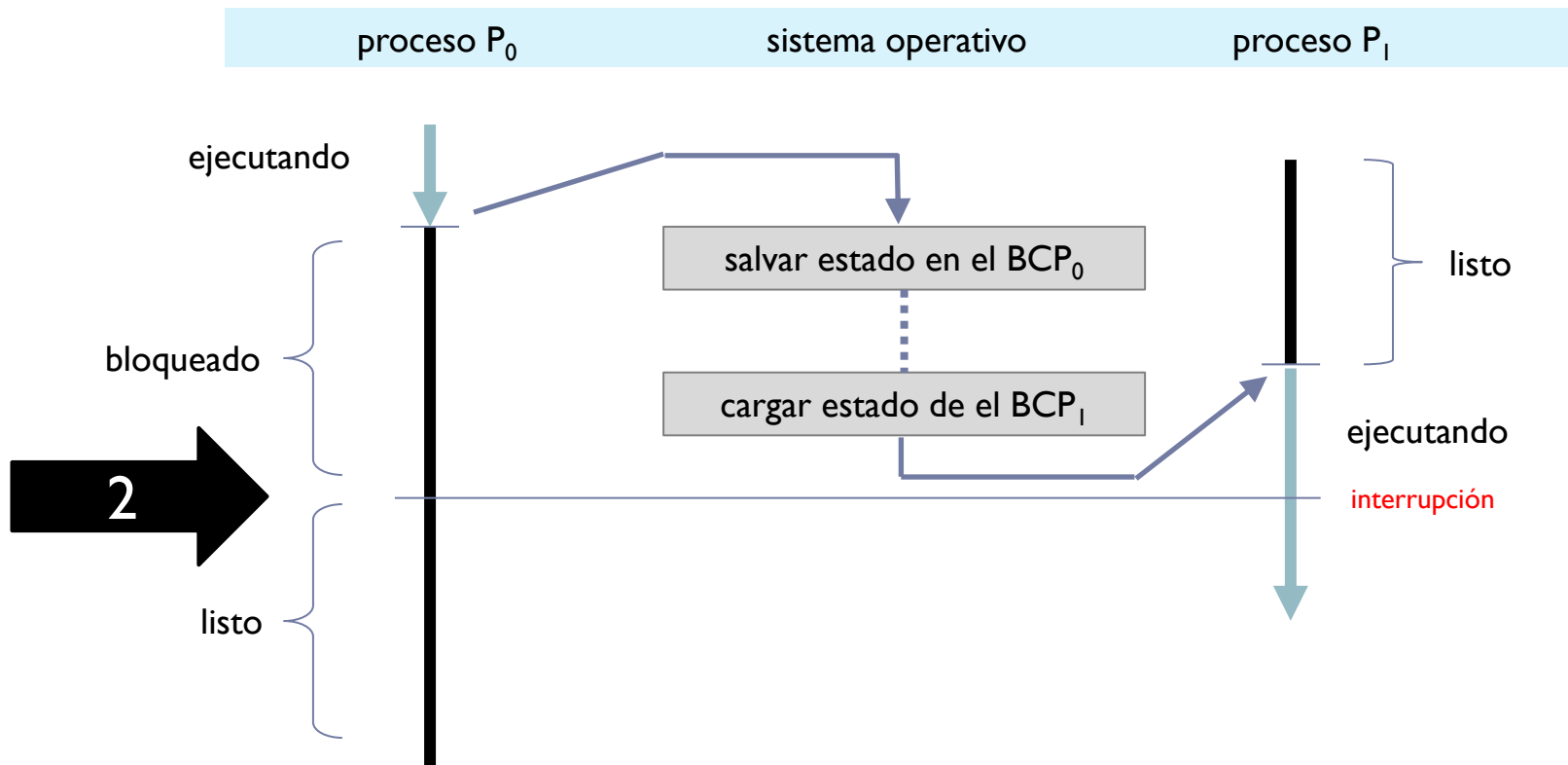
# Multiprogramación: ejemplo de ejecución



# Multiprogramación: ejemplo de ejecución

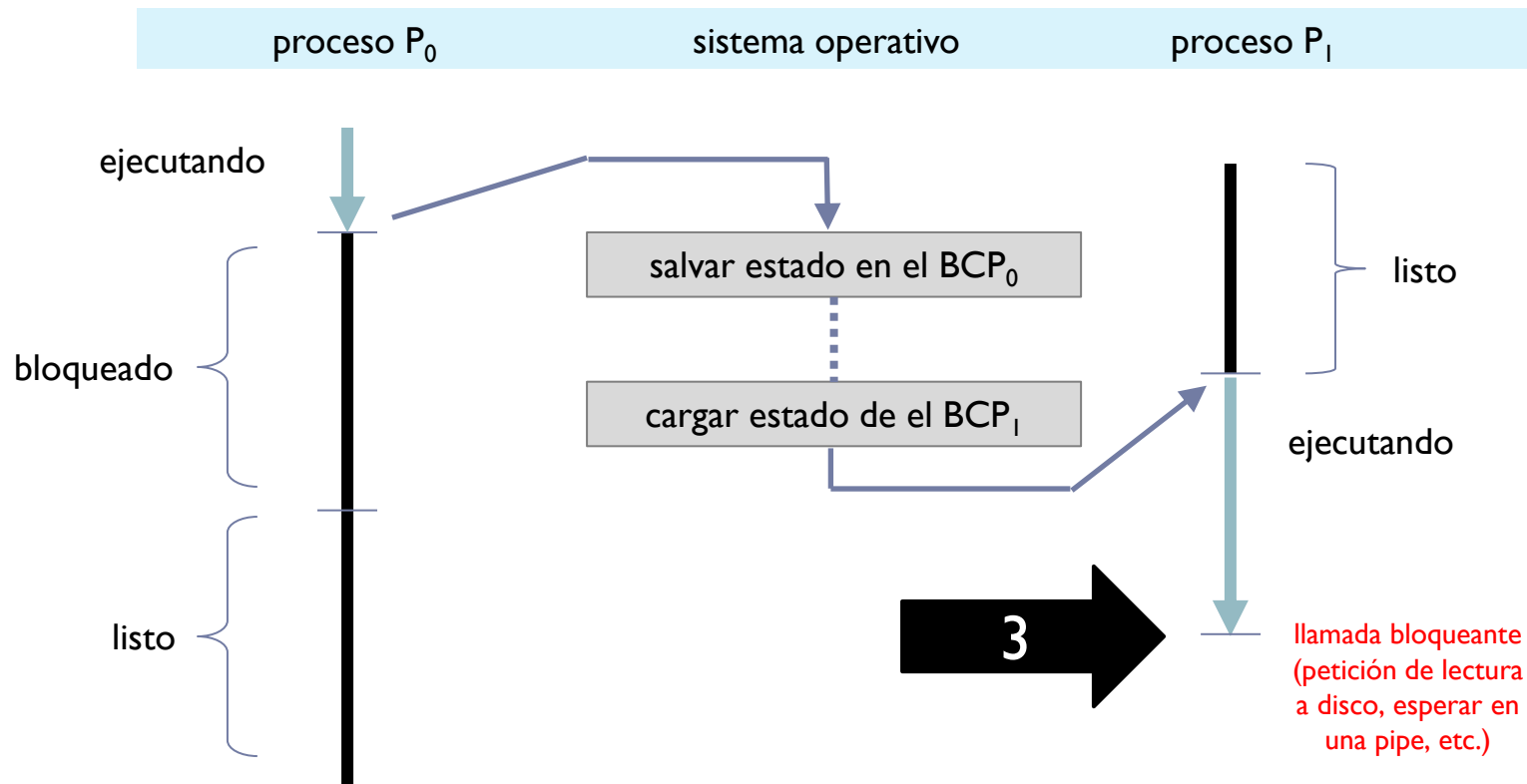


# Multiprogramación: ejemplo de ejecución

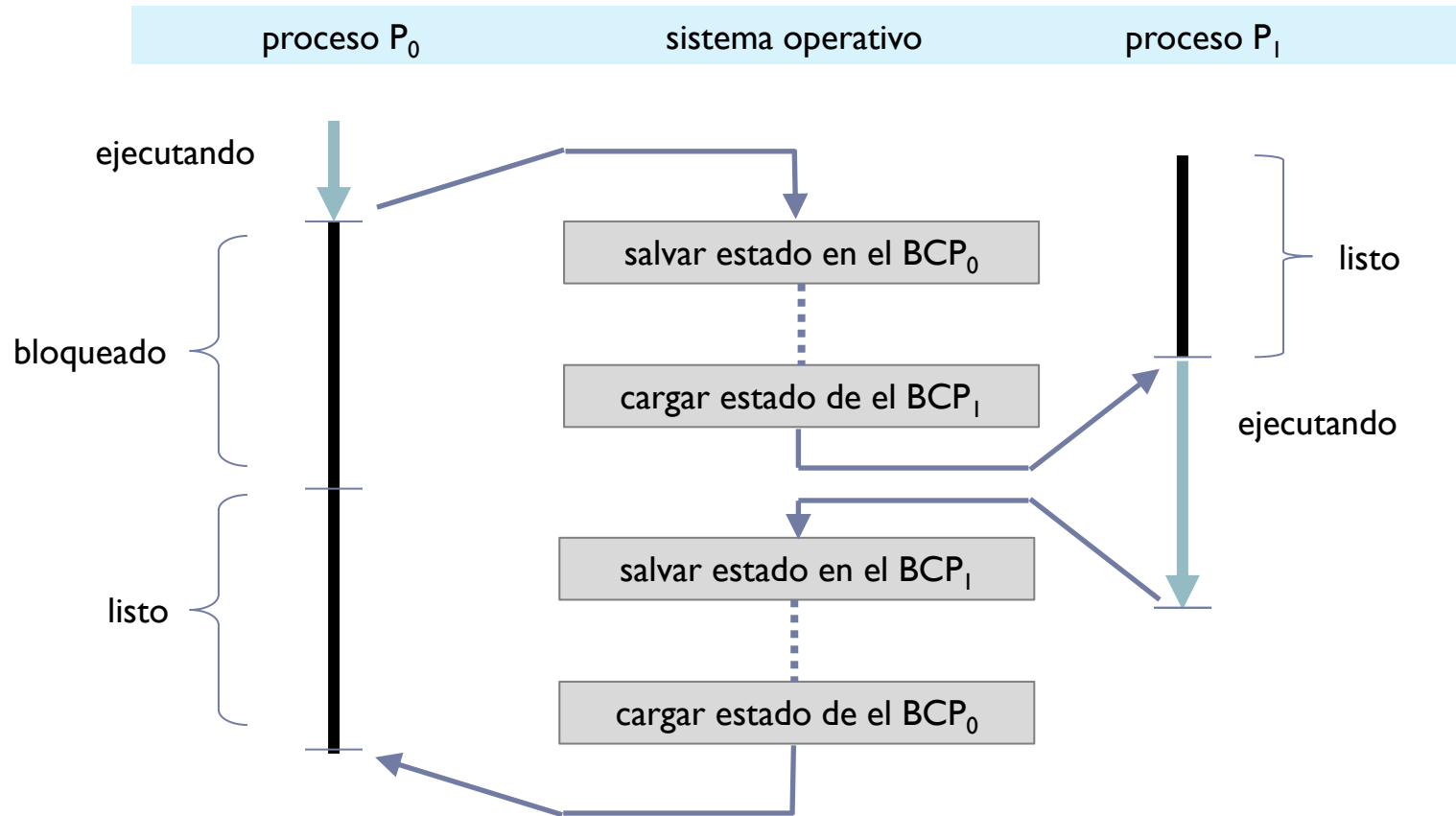




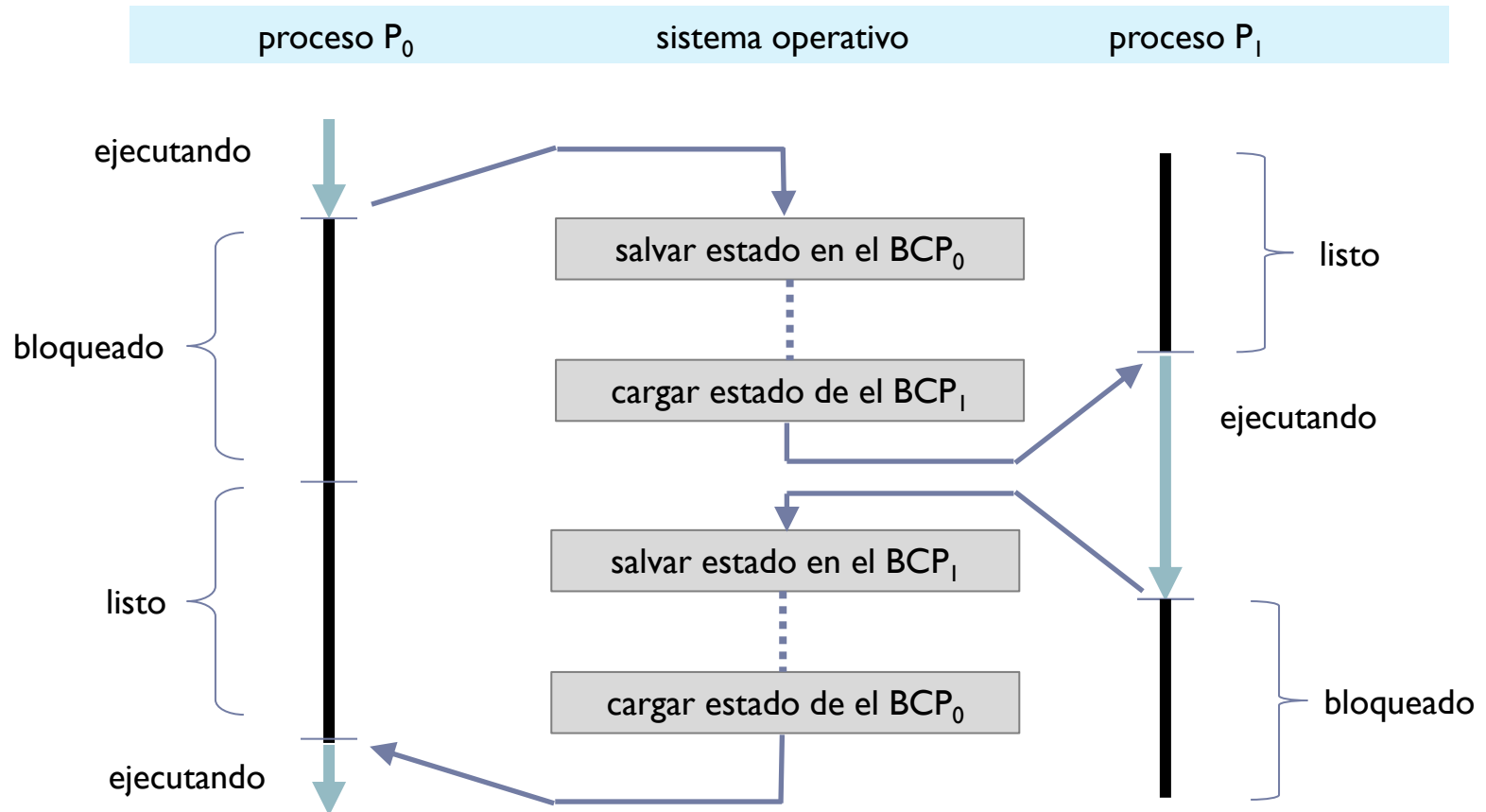
# Multiprogramación: ejemplo de ejecución



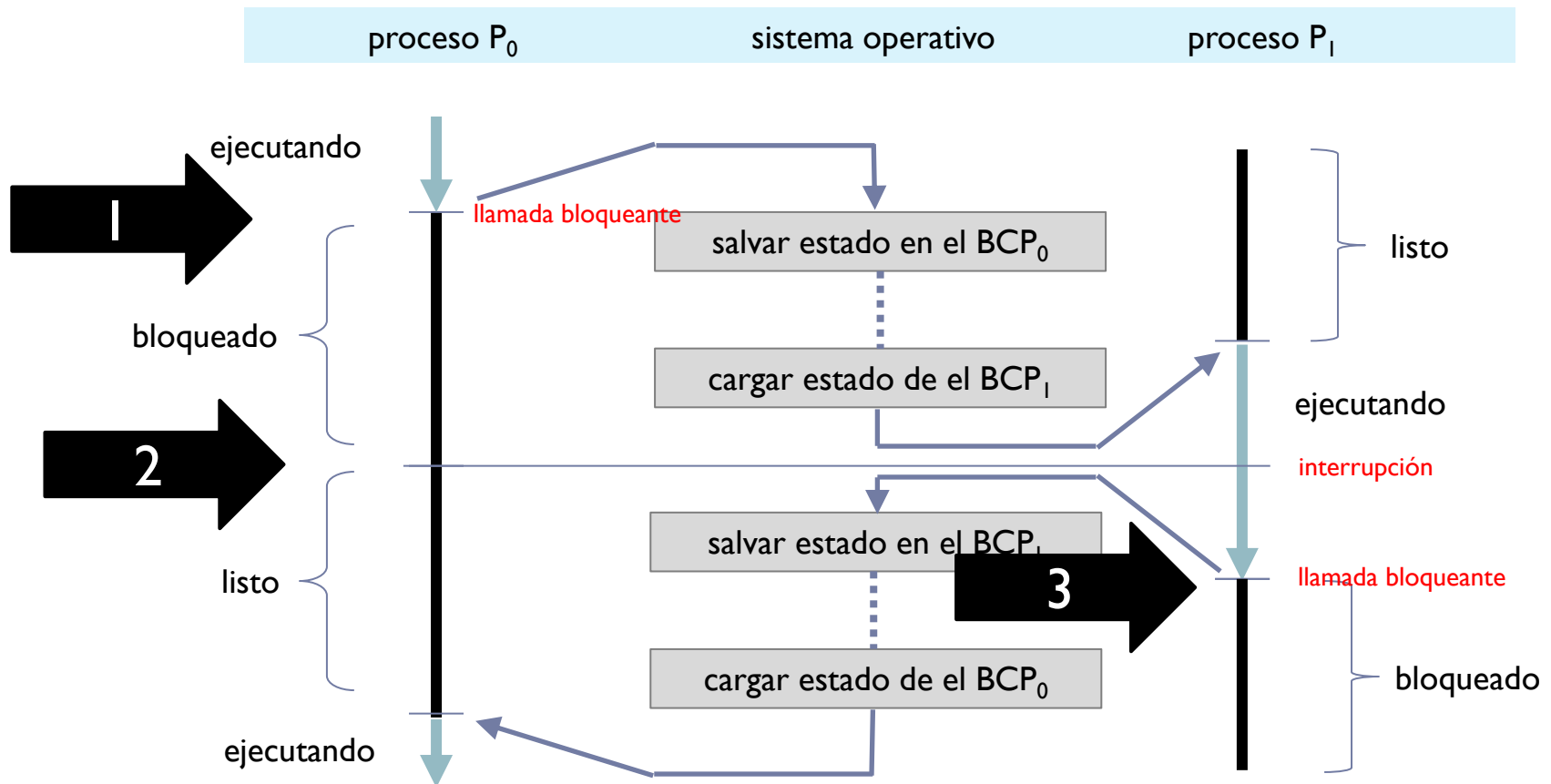
# Multiprogramación: ejemplo de ejecución



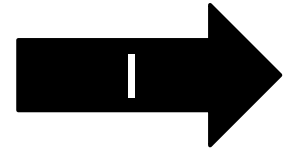
# Multiprogramación: ejemplo de ejecución



# Multiprogramación: ejemplo de ejecución



# Pseudocódigo de ejemplo (P0)



planificador()

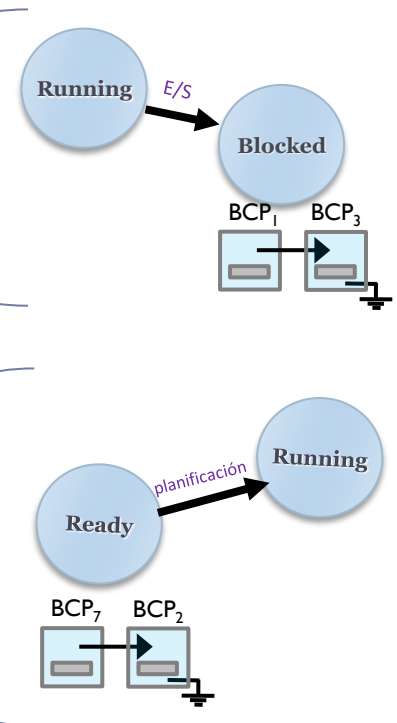
- return extraer(CPU\_Listos);

Teclado\_LeerTecla()

- Si (esVacio(Teclado\_Teclas))
  - procesoActual->estado = BLOQUEADO;
  - Insertar(Teclado\_Procesos, procesoActual);
  - proceso = procesoActual;
- procesoActual = planificador();
- procesoActual->estado = EJECUCION;
- cambio\_contexto( &(proceso->contexto),  
&(procesoActual->contexto));
- return extraer(Teclado\_Teclas) ;

salvar estado en BCP<sub>0</sub>

cargar estado en BCP<sub>1</sub>



# Pseudocódigo de ejemplo (P1)

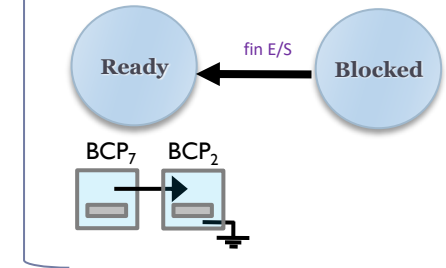
2

Teclado\_Interrupción\_Hardware ()

- T = in (TECLADO\_HW\_ID);
- proceso = **insertar** (T, Teclado\_Teclas);
- Insertar (Teclado\_interrupción\_software);
- Activar\_Interrupción\_Software();

Teclado\_Interrupción\_Software ()

- proceso = **primero** (Teclado\_Procesos);
- SI (proceso != NULL)
  - **eliminar** (Teclado\_Procesos);
  - proceso->estado = LISTO;
  - **insertar** (CPU\_Listos, proceso);
- return ok;



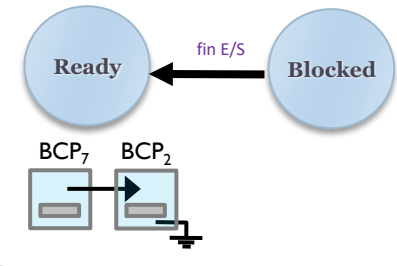
# Pseudocódigo de ejemplo (P1)

Teclado\_Interrupción\_Hardware ()

- T = in (TECLADO\_HW\_ID);
- proceso = **insertar** (T, Teclado\_Teclas);
- Insertar (Teclado\_interrupción\_software);
- Activar\_Interrupción\_Software();

Teclado\_Interrupción\_Software ()

- proceso = **primero** (Teclado\_Procesos);
- SI (proceso != NULL)
  - **eliminar** (Teclado\_Procesos);
  - proceso->estado = LISTO;
  - **insertar** (CPU\_Listos, proceso);
- return ok;



- Un proceso solo puede estar en una cola (como mucho):  
[correcto] eliminar + insertar  
[incorrecto] insertar + eliminar

# Pseudocódigo de ejemplo (P1)

planificador()

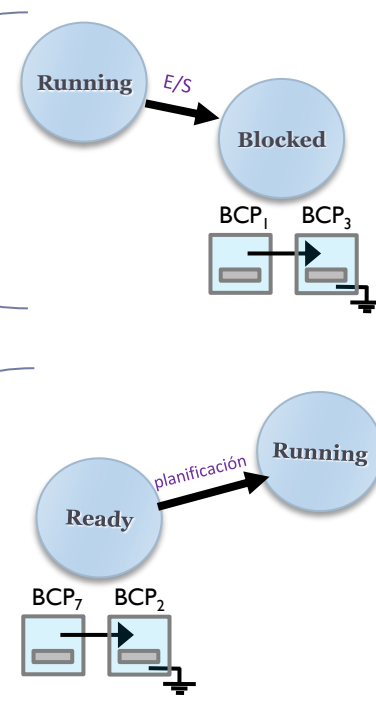
- return extraer(CPU\_Listos);

Disco\_LeerBloqueDisco()

- Si (no hay bloque en caché)
  - procesoActual->estado = BLOQUEADO;
  - Insertar(Disco\_Procesos, procesoActual);
  - proceso = procesoActual;
- procesoActual = planificador();
- procesoActual->estado = EJECUCION;
- cambio\_contexto( &(proceso->contexto), &(procesoActual->contexto));
- return extraer(Disco\_caché, bloque) ;

salvar estado en BCP<sub>1</sub>

cargar estado en BCP<sub>0</sub>





# Pseudocódigo de ejemplo (P0)

## Disco\_LeerBloqueDisco()

- Si (no hay bloque en caché)
  - procesoActual->estado = BLOQUEADO;
  - Insertar(Disco\_Procesos, procesoActual);
  - proceso = procesoActual;
- procesoActual = planificador();
- procesoActual->estado = EJECUCION;
- cambio\_contexto( &(proceso->contexto),  
&(procesoActual->contexto));
- return extraer(Disco\_caché, bloque) ;

## Teclado\_LeerTecla()

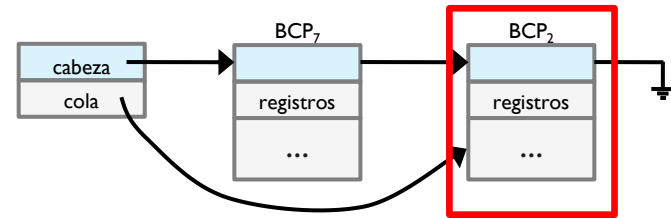
- Si (no hay tecla)
  - procesoActual->estado = BLOQUEADO;
  - Insertar(Teclado\_Procesos, procesoActual);
  - proceso = procesoActual;
- procesoActual = planificador();
- procesoActual->estado = EJECUCION;
- cambio\_contexto( &(proceso->contexto),  
&(procesoActual->contexto));
- return extraer(Teclado\_Teclas) ;



# Planificador y activador

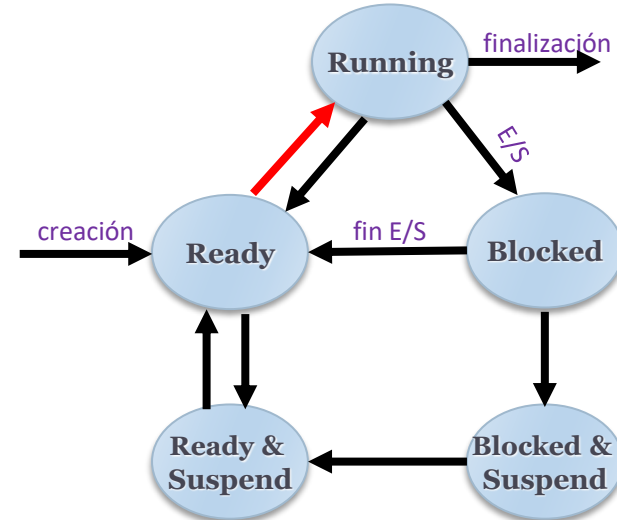
## ► Planificador:

Selecciona el proceso a ser ejecutado entre los que están listos para ejecutar



## ► Activador:

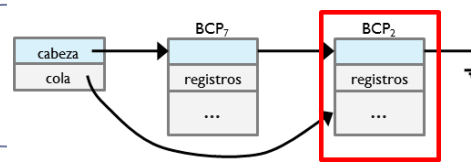
Da control al proceso que el planificador ha seleccionado (cambio de contexto - restaurar)



# Planificador y activador

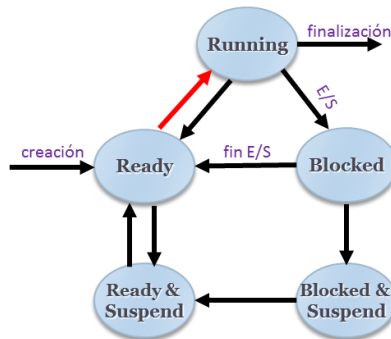
## planificador()

- return extraer(CPU\_Listos);



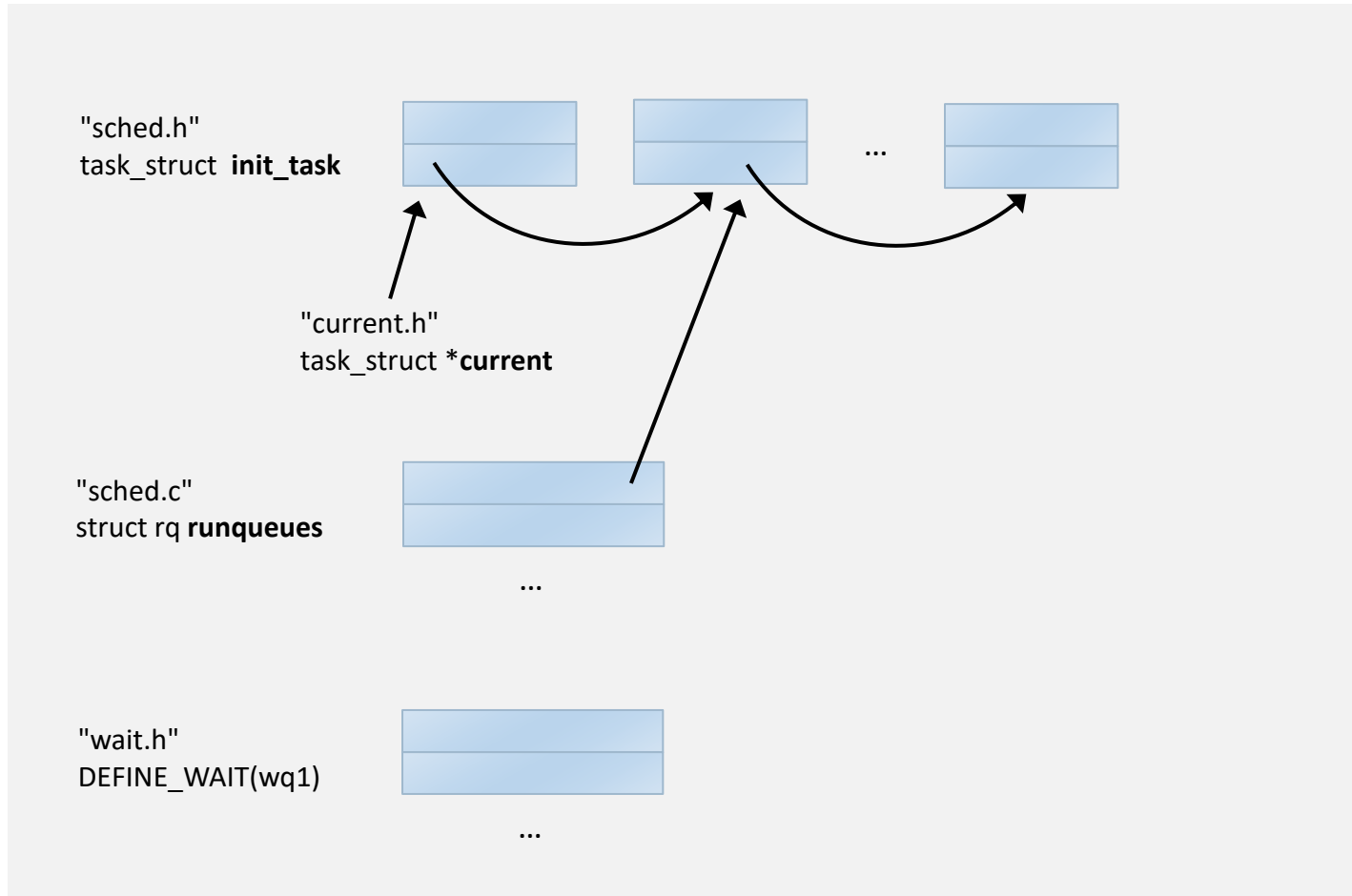
## Teclado\_LeerTecla()

- Si (no hay tecla)
    - procesoActual->estado = BLOQUEADO;
    - Insertar(Teclado\_Procesos, procesoActual);
    - proceso = procesoActual;
  - procesoActual = **planificador**();
  - procesoActual->estado = EJECUCION;
  - **activador** ( &(proceso->contexto), &(procesoActual->contexto));
- return extraer(Teclado\_Teclas) ;



# Colas/Listas de procesos

## Linux



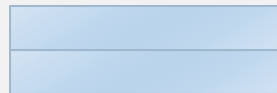
# Colas/Listas de procesos

## Linux



- a. `atomic_t is_blocking_mode = ATOMIC_INIT(0);  
DECLARE_WAIT_QUEUE_HEAD(dso_wq1);`
- b. `atomic_set(&is_blocking_mode, 0);  
wait_event_interruptible(dso_wq1,  
                          (atomic_read(&is_blocking_mode) == 1));`
- c. `atomic_set(&is_blocking_mode, 1);  
wake_up_interruptible(&dso_wq1);`

"wait.h"  
DEFINE\_WAIT(wq1)



...

# Colas/Listas d

## Linux

- DEFINE\_WAIT, DECLARE\_WAIT\_QUEUE\_HEAD(wq)
- wq->flags &= ~WQ\_FLAG\_EXCLUSIVE  
wq->flags |= WQ\_FLAG\_EXCLUSIVE

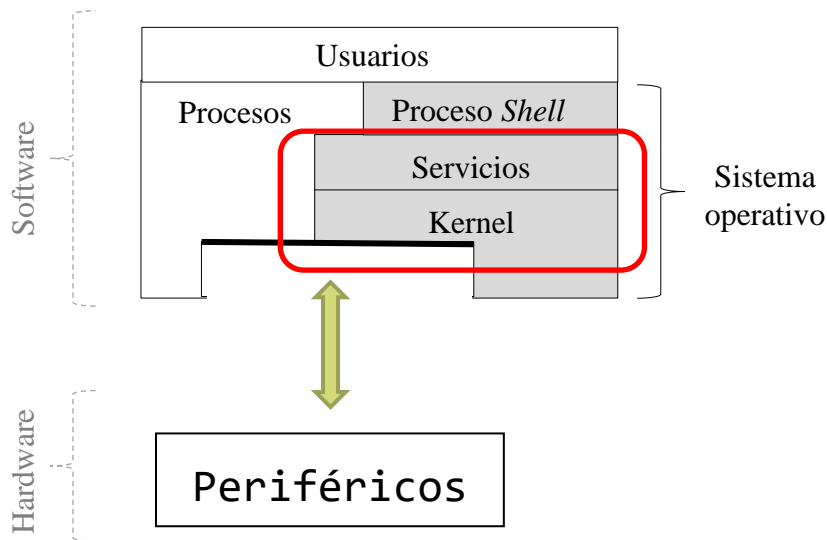
- a. atomic\_t is\_blocking\_mode = ATOMIC\_INIT(0);  
DECLARE\_WAIT\_QUEUE\_HEAD(dso\_wq1);
- b. atomic\_set(&is\_blocking\_mode, 0);  
wait\_event\_interruptible(dso\_wq1,  
(atomic\_read(&is\_blocking\_mode) == 1));
- c. atomic\_set(&is\_blocking\_mode, 1);  
wake\_up\_interruptible(&dso\_wq1);

wait\_event, wait\_event\_interruptible (wq, condition)  
wait\_event\_timeout,  
wait\_event\_interruptible\_timeout (wq, condition, timeout)

wake\_up, wake\_up\_nr, wake\_up\_all, wake\_up\_interruptible,  
wake\_up\_interruptible\_nr, wake\_up\_interruptible\_all,  
wake\_up\_interruptible\_sync, wake\_up\_locked(queue)

# Contenidos

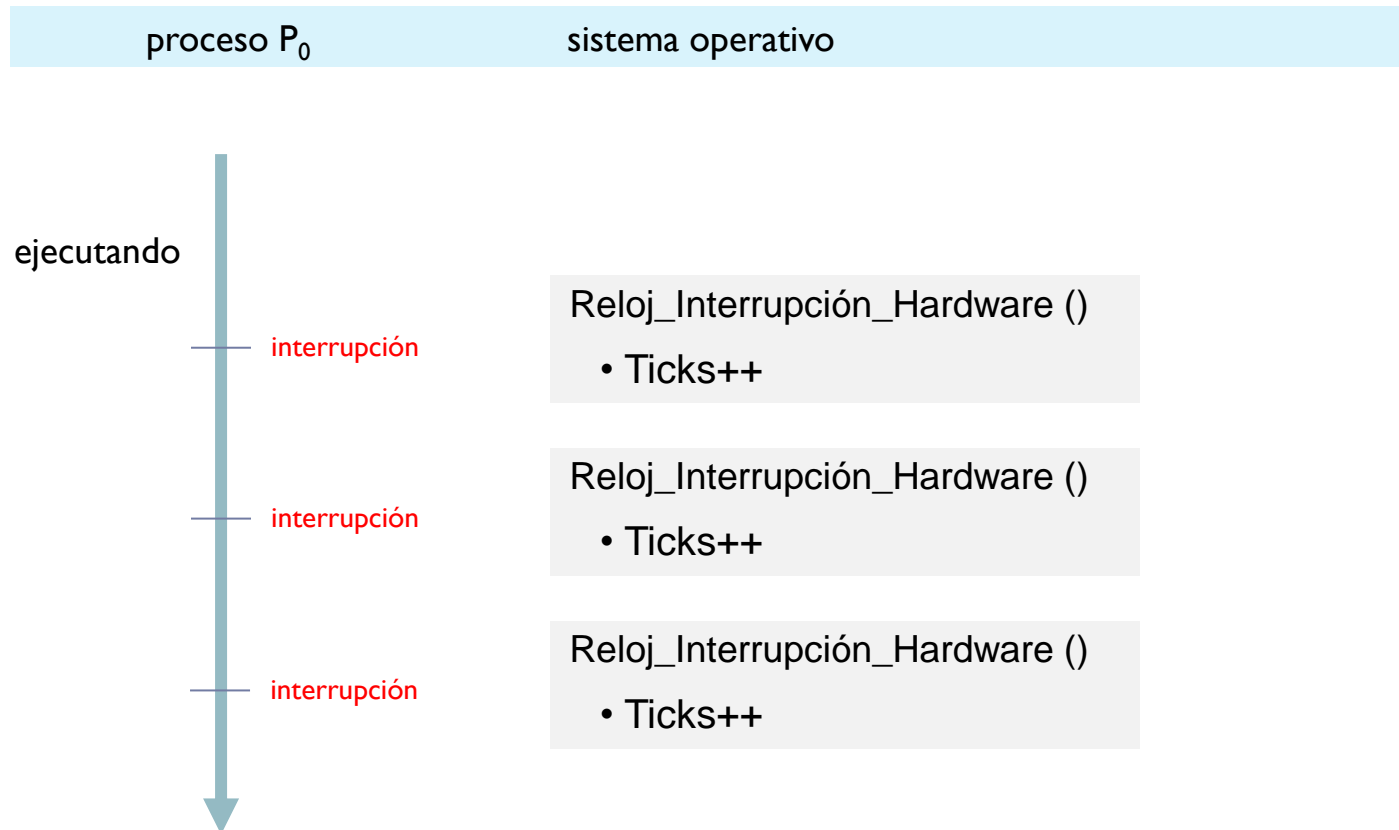
---



- ▶ Introducción
- ▶ C.C.V.
- ▶ **Temporización y C.C.I.**
- ▶ Planificación

# El reloj: tratamiento mínimo

---



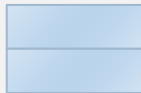


# Temporización

## Linux

- void `process_timeout` (unsigned long `__data`) {  
    **wake\_up\_process**((task\_t \*)`__data`);  
}
- timespec `t`;  
  unsigned long `expire`;  
  struct timer\_list `timer`;

"timer.h"  
timer\_list



...

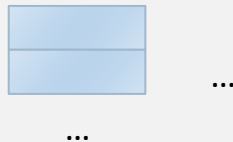
...

# Temporización

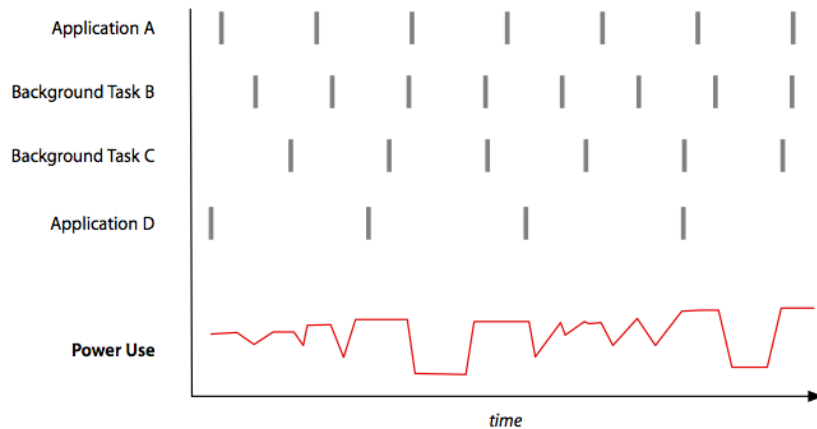
## Linux

- `expire = timespec_to_jiffies(&t) + 1 + jiffies;`  
`init_timer(&timer);`  
`timer.expires = expire;`  
`timer.data = (unsigned long) current;`  
`timer.function = process_timeout;`  
`add_timer(&timer);`  
`current->state = TASK_INTERRUPTIBLE;`  
`schedule(); /* ejecutar mientras otro proceso */`  
`del_singleshot_timer_sync(&timer);`

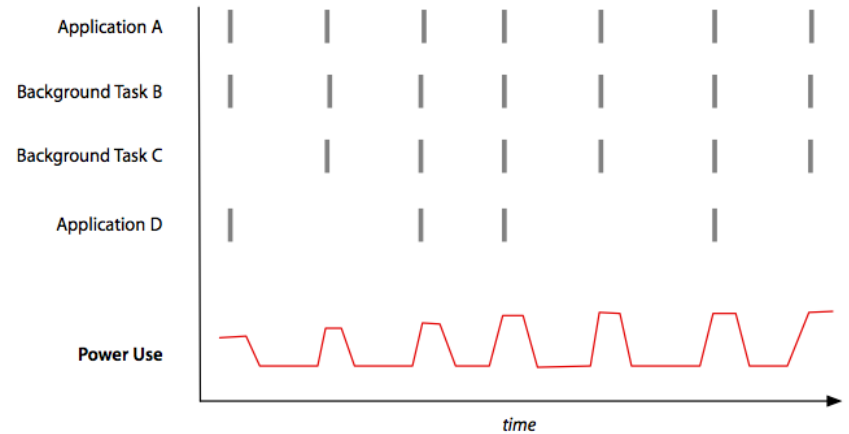
"timer.h"  
timer\_list



# Curiosidad: *Timer Coalescing* en MacOS 10.9.x



Typically, numerous applications and background processes use timers with different intervals.

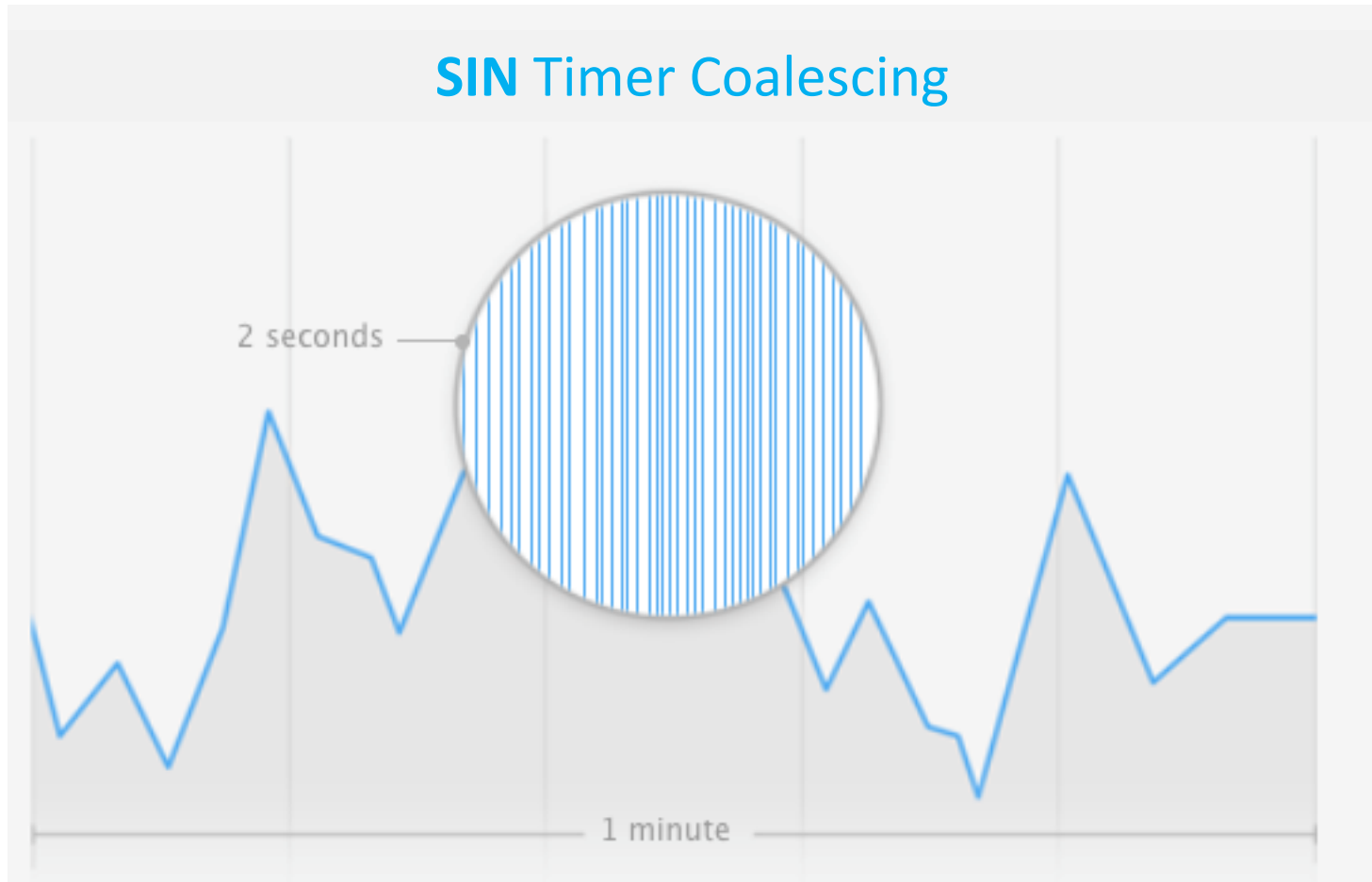


Timer Coalescing shifts timers of multiple applications to execute at the same time.

# Curiosidad:

## *Timer Coalescing* en MacOS 10.9.x

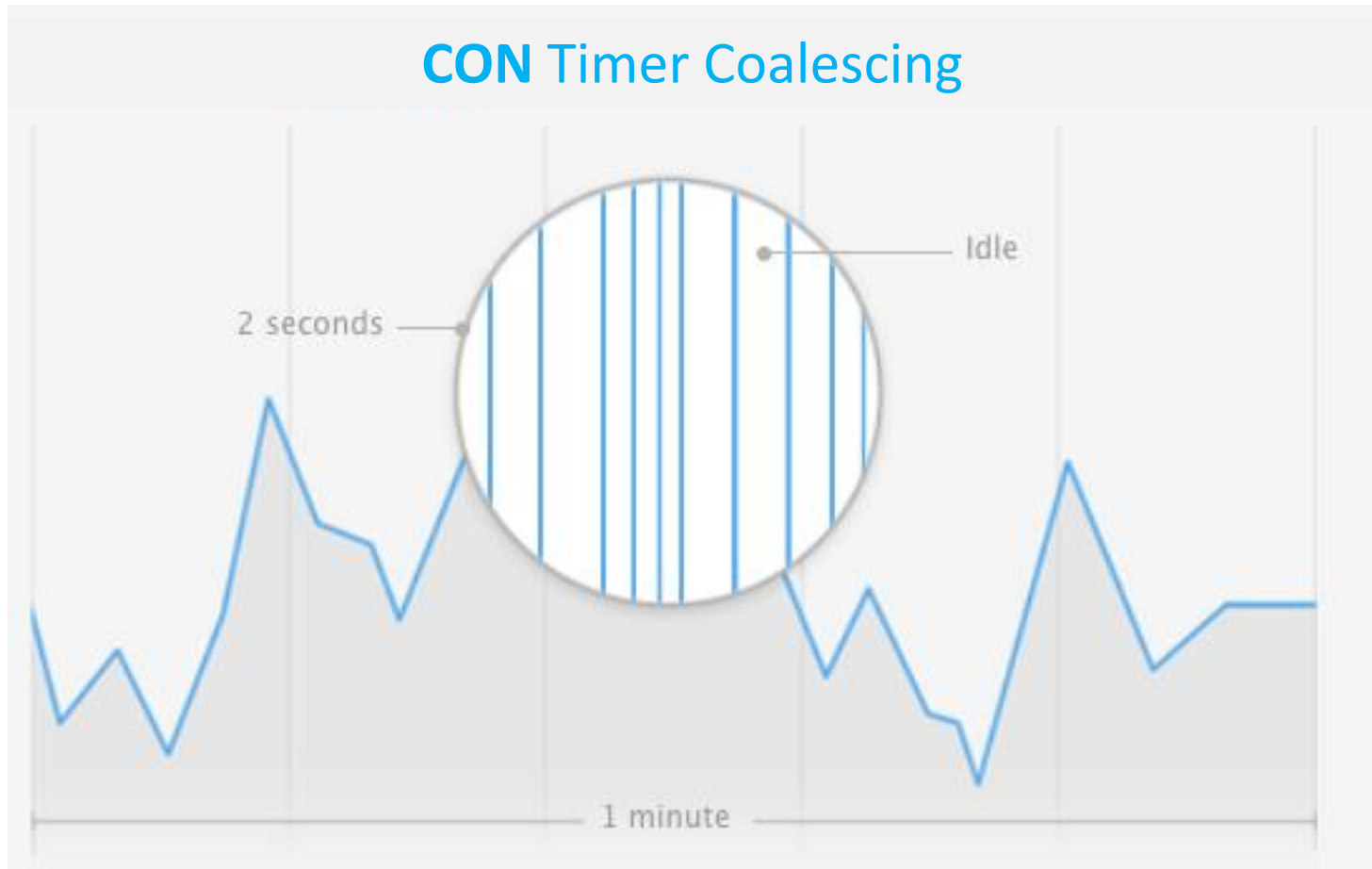
---



# Curiosidad:

## *Timer Coalescing* en MacOS 10.9.x

---



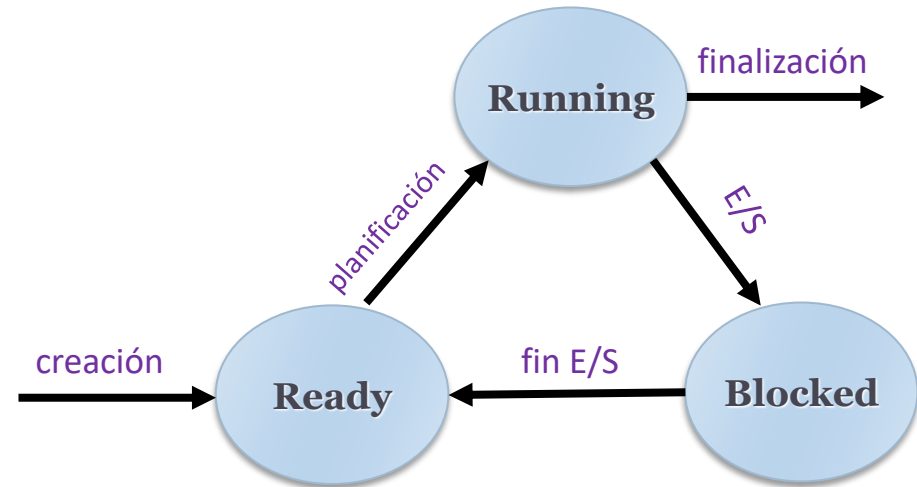
# Multitarea (datos y funciones)

Requisitos	Información (en estructuras de datos)	Funciones (internas, servicio y API)
Recursos	<ul style="list-style-type: none"> <li>Zonas de memoria (código, datos y pila)</li> <li>Archivos abiertos</li> <li>Señales activas</li> </ul>	<ul style="list-style-type: none"> <li>Diversas funciones internas</li> <li>Diversas funciones de servicio para memoria, ficheros, etc.</li> </ul>
Multiprogramación	<ul style="list-style-type: none"> <li>Estado de ejecución</li> <li>Contexto: registros de CPU...</li> <li>Lista de procesos</li> </ul>	<ul style="list-style-type: none"> <li>Int. hw/sw de dispositivos</li> <li>Planificador</li> <li>Crear/Destruir/Planificar proceso</li> </ul>
○ Protección / Compartición	<ul style="list-style-type: none"> <li>Paso de mensajes                             <ul style="list-style-type: none"> <li>Cola de mensajes de recepción</li> </ul> </li> <li>Memoria compartida                             <ul style="list-style-type: none"> <li>Zonas, locks y conditions</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>Envío/Recepción mensaje y gestión de la cola de mensaje</li> <li>API concurrencia y gestión de estructuras de datos</li> </ul>
○ Jerarquía de procesos	<ul style="list-style-type: none"> <li>Relación de parentesco</li> <li>Conjuntos de procesos relacionados</li> <li>Procesos de una misma sesión</li> </ul>	<ul style="list-style-type: none"> <li>Clonar/Cambiar imagen de proceso</li> <li>Asociar procesos e indicar proceso representante</li> </ul>
Multitarea	<ul style="list-style-type: none"> <li>Quantum restante</li> <li>Prioridad</li> </ul>	<ul style="list-style-type: none"> <li>Int. hw/sw de reloj</li> <li>Planificador</li> <li>Crear/Destruir/Planificar proceso</li> </ul>
Multiproceso	<ul style="list-style-type: none"> <li>Afinidad</li> </ul>	<ul style="list-style-type: none"> <li>Int. hw/sw de reloj</li> <li>Planificador</li> <li>Crear/Destruir/Planificar proceso</li> </ul>

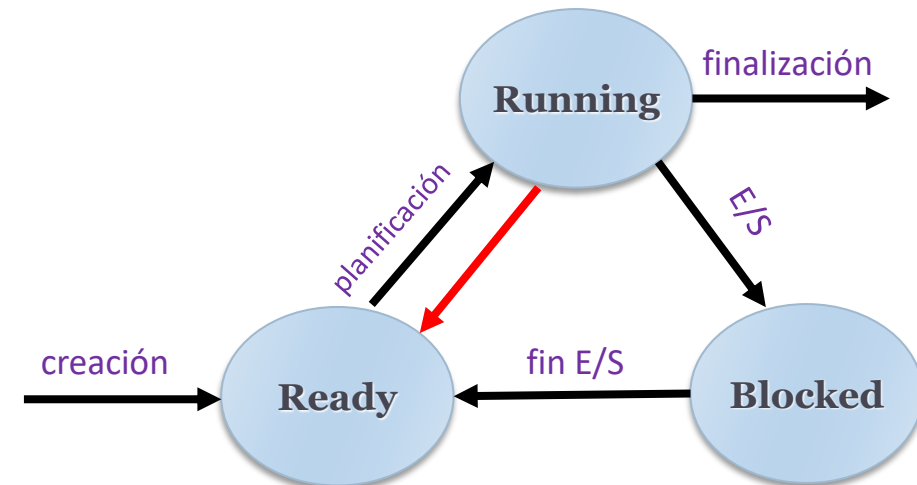
# Estados de un proceso

- Estado
- Lista/Cola
- Contexto

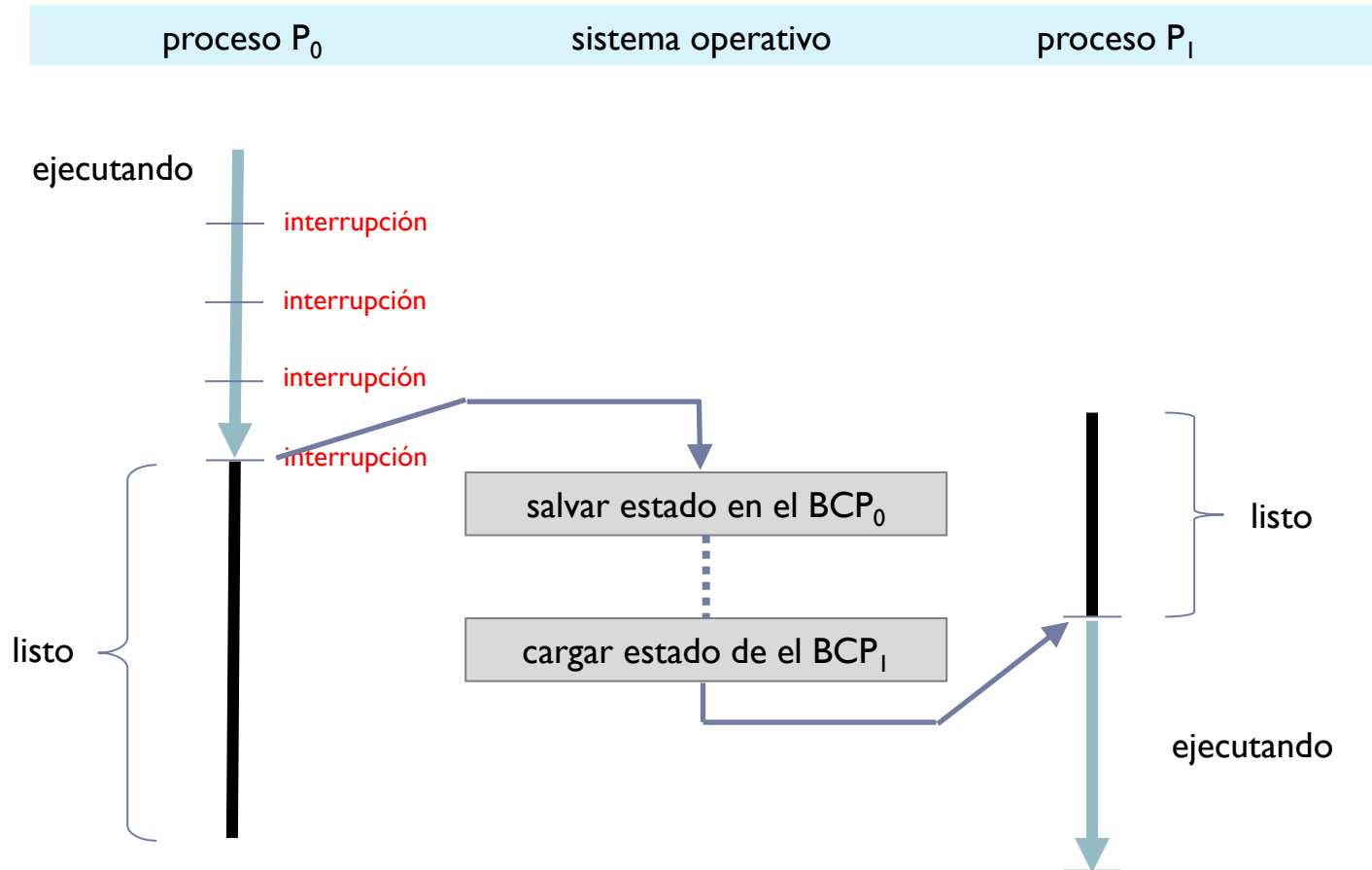
C.C.V.



C.C.V. + C.C.I.



# El reloj: tratamiento con c.c.v. + c.c.i.





# Pseudocódigo de ejemplo (P0)

## Reloj\_Interrupción\_Hardware ()

- Ticks++;
- Insertar (Reloj\_Planificar\_Rodaja);
- Activar\_Interrupción\_Software();

## Reloj\_Planificar\_Rodaja ()

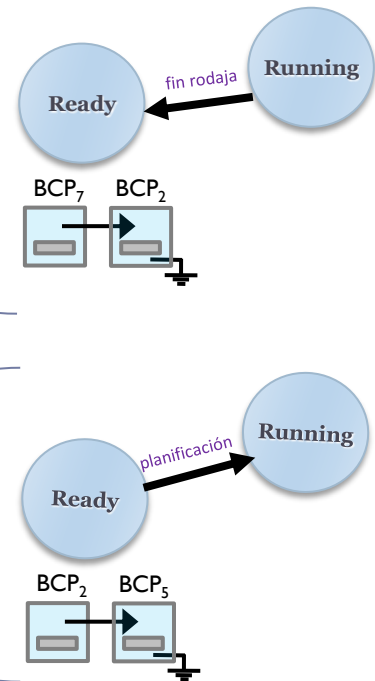
- pActual->rodaja = pActual->rodaja - 1;
- Si (pActual->rodaja == 0)
  - pActual->estado = LISTO;
  - pActual->rodaja = RODAJA;
  - insertar (CPU\_Listos, pActual);
  - proceso = pActual;
- pActual = planificador();
- pActual->estado = EJECUCIÓN;
- cambio\_contexto(
  - &(proceso->contexto),
  - &(pActual->contexto));
- return ok;

## planificador()

- return  
extraer(CPU\_Listos);

salvar estado en BCP<sub>0</sub>

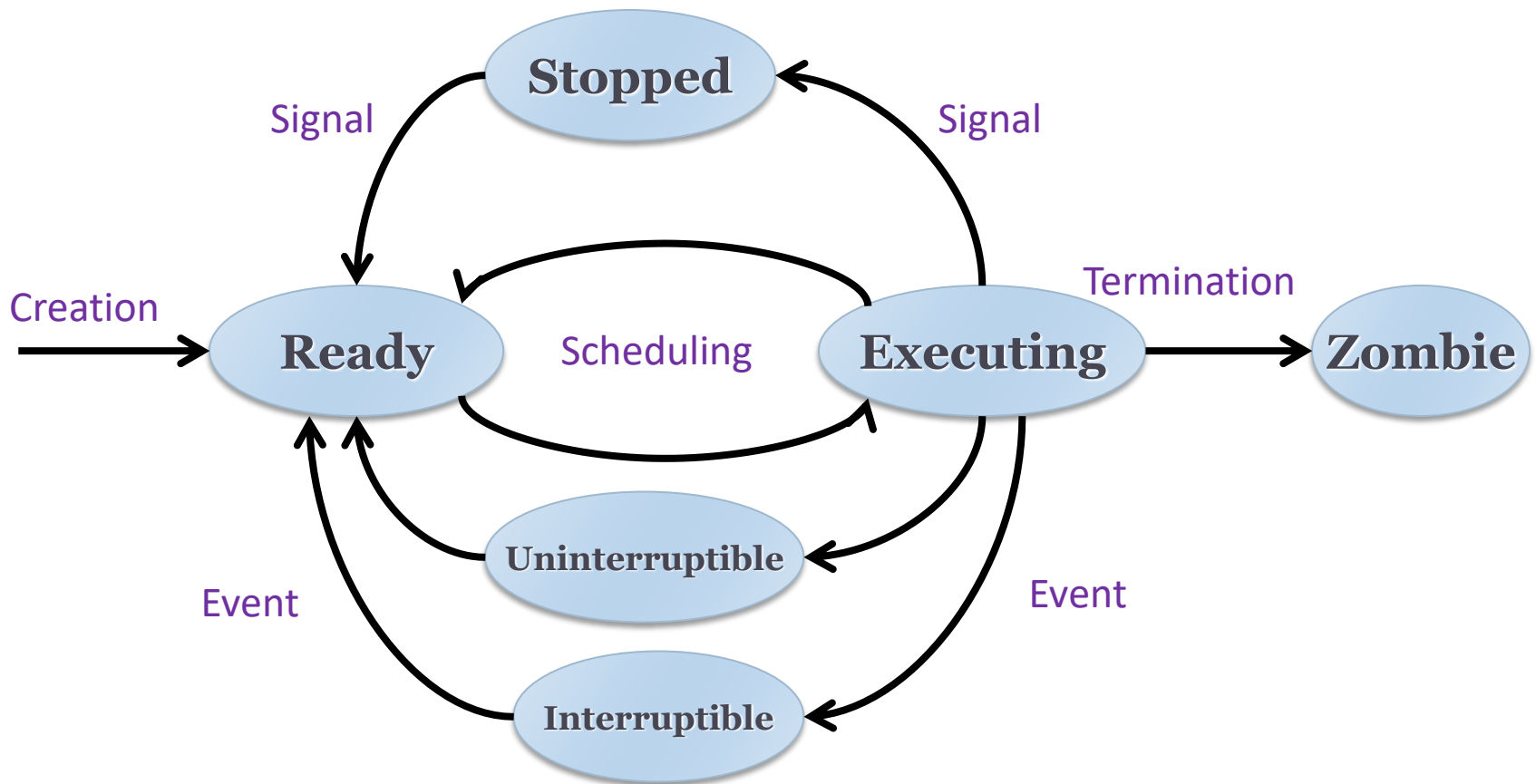
cargar estado en BCP<sub>1</sub>



# Estados de un proceso

## Linux

---



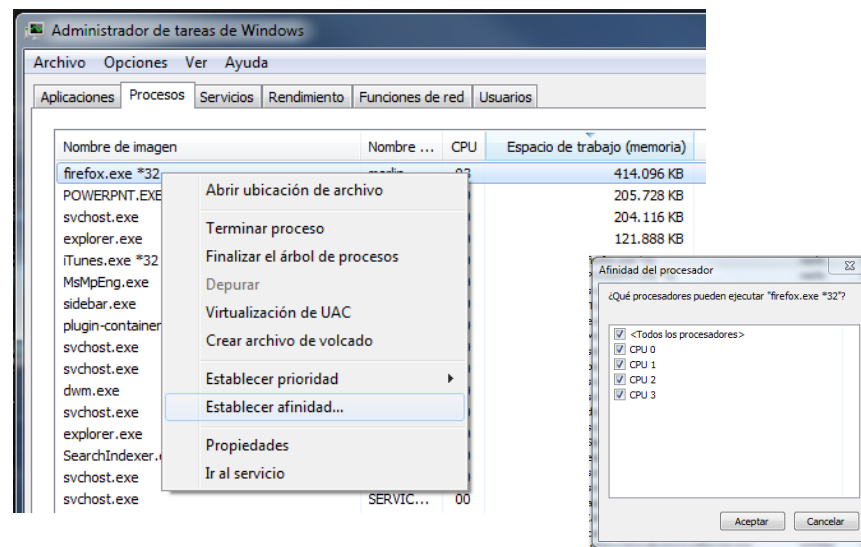
# Multiproceso

Requisitos	Información (en estructuras de datos)	Funciones (internas, servicio y API)
Recursos	<ul style="list-style-type: none"> <li>Zonas de memoria (código, datos y pila)</li> <li>Archivos abiertos</li> <li>Señales activas</li> </ul>	<ul style="list-style-type: none"> <li>Diversas funciones internas</li> <li>Diversas funciones de servicio para memoria, ficheros, etc.</li> </ul>
Multiprogramación	<ul style="list-style-type: none"> <li>Estado de ejecución</li> <li>Contexto: registros de CPU...</li> <li>Lista de procesos</li> </ul>	<ul style="list-style-type: none"> <li>Int. hw/sw de dispositivos</li> <li>Planificador</li> <li>Crear/Destruir/Planificar proceso</li> </ul>
○ Protección / Compartición	<ul style="list-style-type: none"> <li>Paso de mensajes                             <ul style="list-style-type: none"> <li>Cola de mensajes de recepción</li> </ul> </li> <li>Memoria compartida                             <ul style="list-style-type: none"> <li>Zonas, locks y conditions</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>Envío/Recepción mensaje y gestión de la cola de mensaje</li> <li>API concurrencia y gestión de estructuras de datos</li> </ul>
○ Jerarquía de procesos	<ul style="list-style-type: none"> <li>Relación de parentesco</li> <li>Conjuntos de procesos relacionados</li> <li>Procesos de una misma sesión</li> </ul>	<ul style="list-style-type: none"> <li>Clonar/Cambiar imagen de proceso</li> <li>Asociar procesos e indicar proceso representante</li> </ul>
Multitarea	<ul style="list-style-type: none"> <li>Quantum restante</li> <li>Prioridad</li> </ul>	<ul style="list-style-type: none"> <li>Int. hw/sw de reloj</li> <li>Planificador</li> <li>Crear/Destruir/Planificar proceso</li> </ul>
Multiproceso	<ul style="list-style-type: none"> <li>Afinidad</li> </ul>	<ul style="list-style-type: none"> <li>Int. hw/sw de reloj</li> <li>Planificador</li> <li>Crear/Destruir/Planificar proceso</li> </ul>

# Multiproceso

## ► Afinidad:

- Los procesos tienen ‘afinidad’ (*affinity*) a una CPU: «mejor volver a la misma CPU»

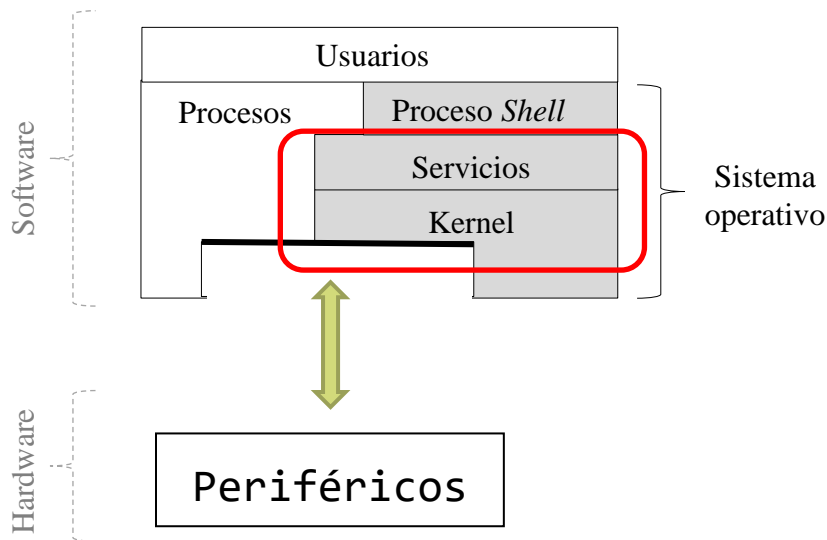


## ► Simetría:

- Los procesos se ejecutan en la CPU que tienen unas capacidades específicas

# Contenidos

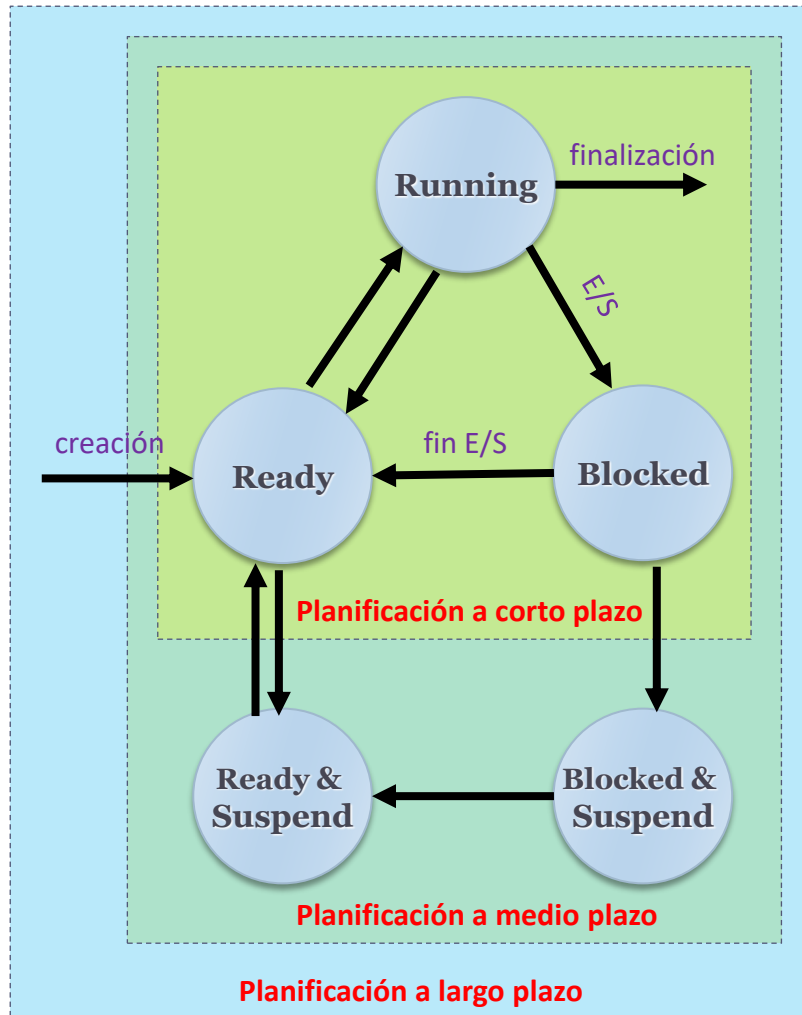
---



- ▶ **Introducción**
- ▶ **C.C.V.**
- ▶ **Temporización y C.C.I.**
- ▶ **Planificación**

# Planificación de procesos

## niveles de planificación



- ▶ **A largo plazo**
  - ▶ añadir procesos a ejecutar
  - ▶ Invocado con baja frecuencia
    - ▶ puede ser algo lento
- ▶ **A medio plazo**
  - ▶ añadir procesos a RAM
- ▶ **A corto plazo**
  - ▶ qué proceso tiene la UCP
  - ▶ Invocado frecuentemente
    - ▶ rápido

# Planificación de procesos

## objetivos de los algoritmos de planificación (según sistema)

---

### ▶ Todos los sistemas:

- ▶ **Equitativo** – ofrece a cada proceso una parte equitativa de la CPU
- ▶ **Expeditivo** – cumplimiento de la política emprendida de reparto
- ▶ **Balanceado** – mantener todas las partes del sistema ocupadas

### ▶ Sistemas *batch*:

- ▶ **Productividad** – maximizar el número de trabajos por hora
- ▶ **Tiempo de espera** – minimizar el tiempo entre emisión y terminación del trabajo
- ▶ **Uso de CPU** – mantener la CPU ocupada todo el tiempo

### ▶ Sistemas **Interactivos**:

- ▶ **Tiempo de respuesta** – responder a las peticiones lo más rápido posible
- ▶ **Ajustado** – satisfacer las expectativas de los usuarios

### ▶ Sistemas de **tiempo real**:

- ▶ **Cumplimiento de plazos** – evitar la pérdida de datos
- ▶ **Predecible** – evitar la degradación de calidad en sistemas multimedia

# Planificación de procesos

## características de los algoritmos de planificación (1/2)

---

### ▶ *Preemption:*

#### ▶ Sin expulsión:

- ▶ El proceso conserva la CPU mientras desee.
- ▶ Cambios de contexto voluntarios (C.C.V.)
- ▶ [v/i] Un proceso puede bloquear al resto pero solución fácil a la compartición de recursos
- ▶ Windows 3.1, Windows 95 (16 bits), NetWare, MacOS 9.x.

#### ▶ Con expulsión:

- ▶ Exige un reloj que interrumpe periódicamente:
  - cuando pasa el quantum de un proceso se cambia a otro
- ▶ (Se añade) Cambios de contexto involuntarios (C.C.I.)
- ▶ [v/i] Mejora la interactividad pero precisa de mecanismos para condiciones de carrera
- ▶ AmigaOS (1985), Windows NT-XP-Vista-7, Linux, BSD, MacOS X



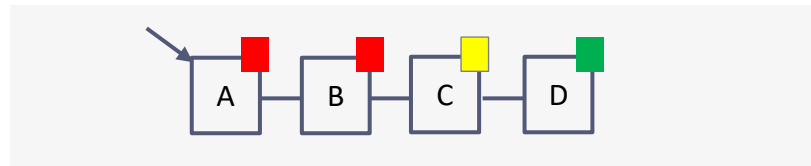
# Planificación de procesos

## características de los algoritmos de planificación (2/2)

---

### ▶ Clasificación de elementos en las colas:

#### ▶ Por prioridad



#### ▶ Por tipo

- ▶ CPU-bound (más 'rachas' –burst– de tiempo usando CPU)
- ▶ IO-bound (más 'rachas' de tiempo esperando E/S)

### ▶ CPU-aware:

#### ▶ Afinidad:

- ▶ Los procesos tienen 'afinidad' (*affinity*) a una CPU: «mejor volver a la misma CPU»

#### ▶ Simetría:

- ▶ Los procesos se ejecutan en la CPU que tienen unas capacidades específicas a dicha CPU

# Planificación de procesos

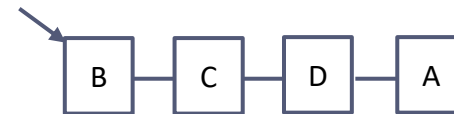
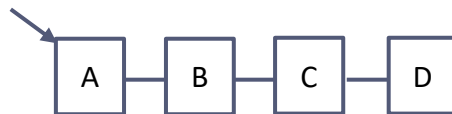
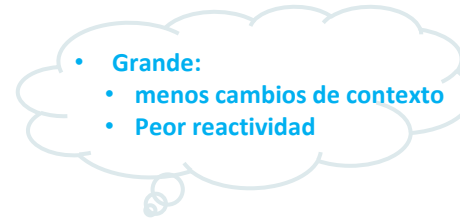
## principales algoritmos de planificación (1/3)

---

### ▶ Cíclico o *Round Robin*:

#### ▶ Asignación rotatoria del procesador

- ▶ Se asigna un tiempo máximo de procesador (rodaja o quantum)



#### ▶ Equitativo pero interactivo:

- ▶ Mejor **por UID** que por proceso

#### ▶ En Linux:

- Aparición en 11/2010 de un parche para el kernel que automáticamente crea un grupo de tareas **por TTY** para mejorar la interactividad en sistemas cargados.
- Son 224 líneas de código que modifican el planificador del kernel que en las primeras pruebas muestra que la latencia media cae a una 60 veces (1/60).

#### ▶ Uso en sistemas de tiempo compartido

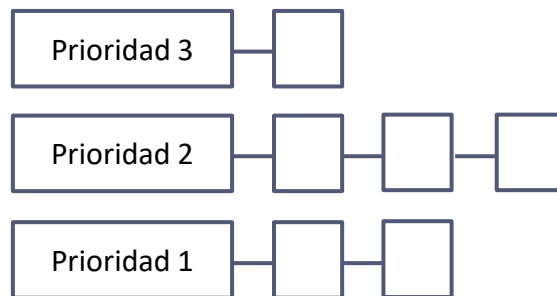
# Planificación de procesos

## principales algoritmos de planificación (2/3)

---

### ▶ Por prioridad:

- ▶ Asignación a procesos más prioritarios el procesador
  - ▶ Se puede combinar con cíclica. Ejemplo con tres clases de prioridad



- ▶ Características:
  - ▶ Uso de prioridades fijas: problema de inanición
  - ▶ No fijas: uso de algún algoritmo de envejecimiento
- ▶ Uso en sistemas de tiempo compartido con aspectos de tiempo real

# Planificación de procesos

## principales algoritmos de planificación (3/3)

---

### ▶ Primero el trabajo más corto:

- ▶ Dado un conjunto de trabajo del que se sabe la duración total de la ejecución de cada uno de ellos, se ordenan de la menor a la mayor duración.
- ▶ Características:
  - ▶ [v] Produce el menor tiempo de respuesta (medio)
  - ▶ [i] Penaliza los trabajos largos.
- ▶ Uso en sistemas *batch*.

### ▶ FIFO:

- ▶ Ejecución por el estricto orden de llegada.
- ▶ Características:
  - ▶ [v] Simple de implantar.
  - ▶ [i] Penaliza los trabajos prioritarios.
- ▶ Uso en sistemas *batch*.

# Política vs mecanismo

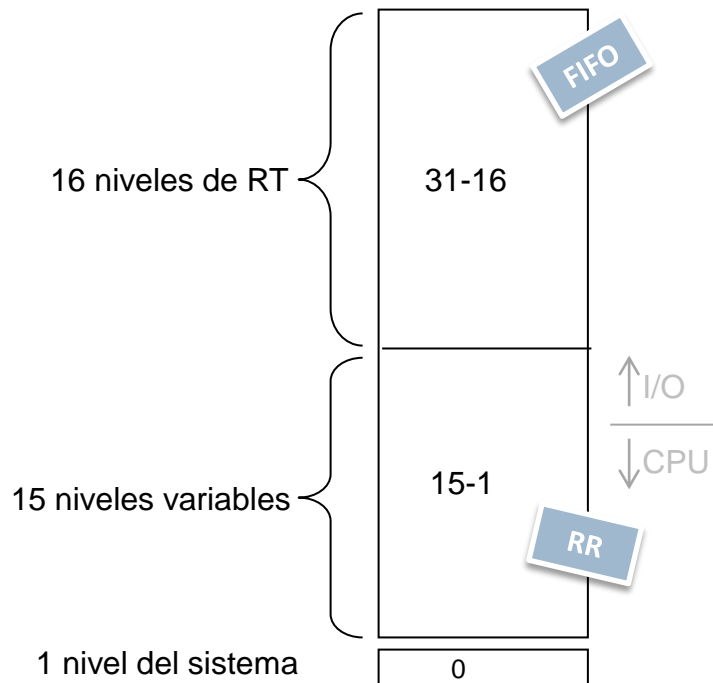
---

- ▶ Separación de lo qué se puede hacer de cómo se puede hacer
  - ▶ Normalmente, un proceso conoce cuál es el hilo más prioritario, el que más E/S necesitará, etc.
- ▶ Uso de algoritmos de planificación parametrizados
  - ▶ Mecanismo en el kernel
- ▶ Parámetros rellenos por los procesos de usuarios
  - ▶ Política establecida por los procesos de usuario

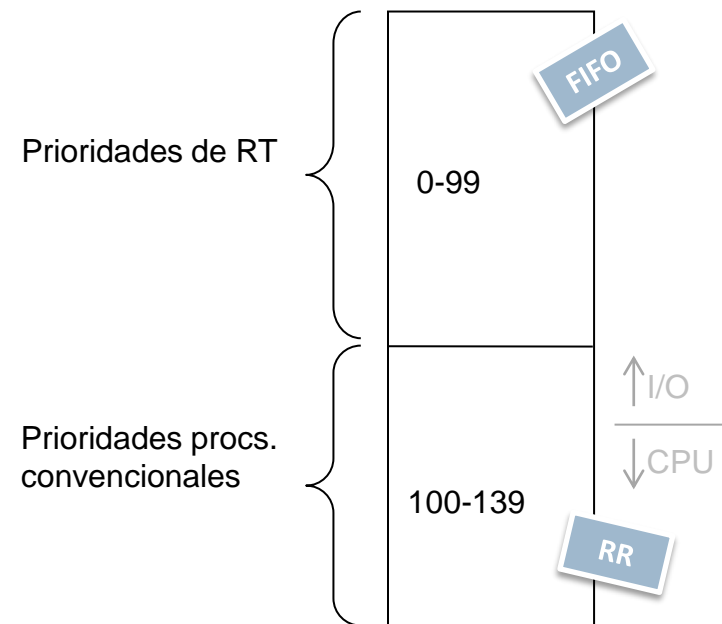
# Planificación multipolítica

## Windows 2000 y Linux

### Windows 2000



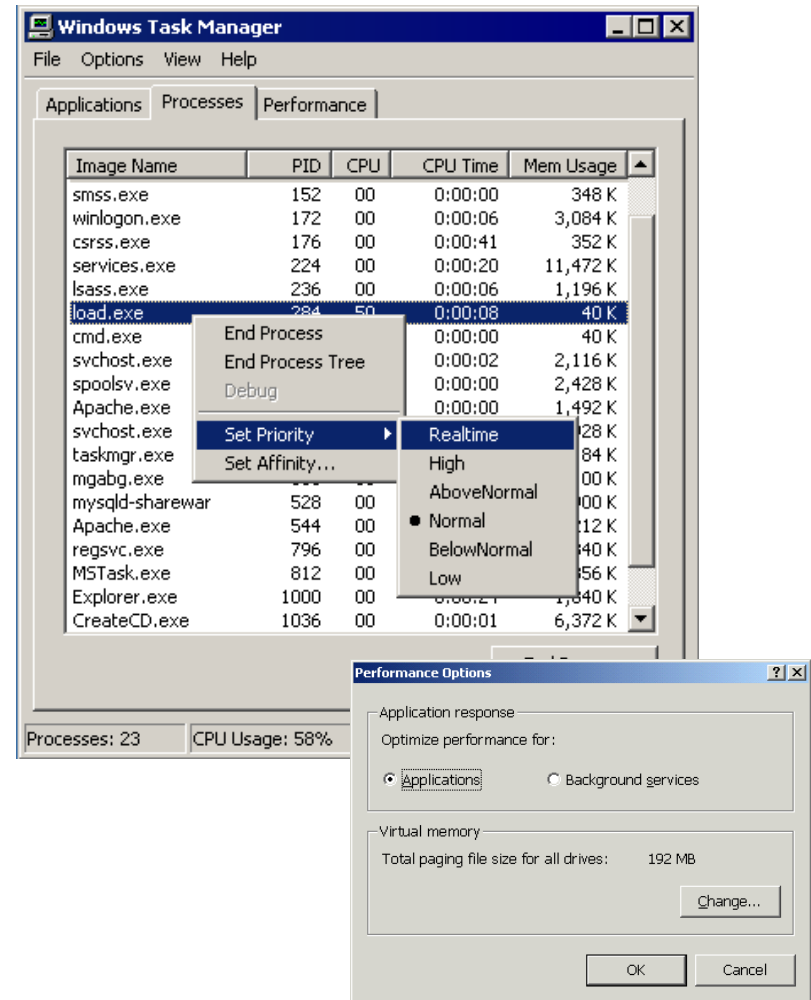
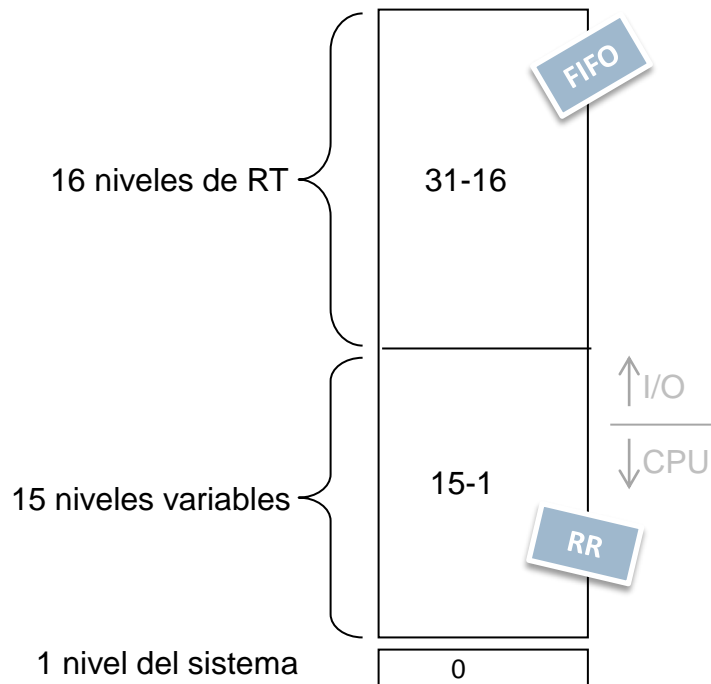
### Linux



# Planificación multipolítica

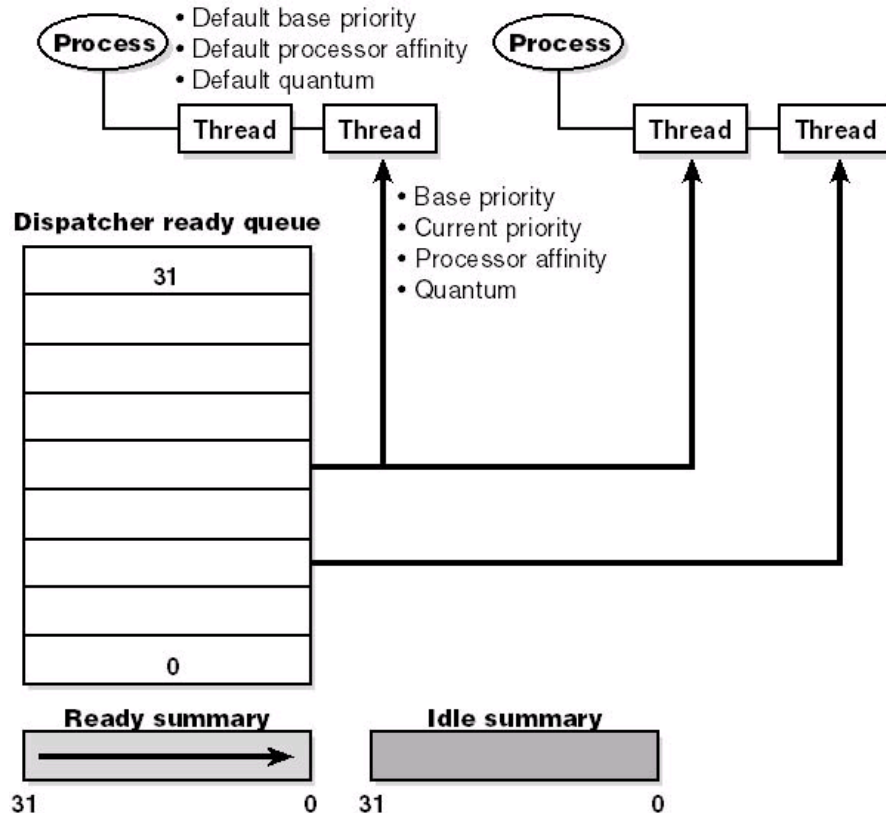
## Windows 2000

Windows 2000



# Planificación: estructuras de datos

## Windows 2000



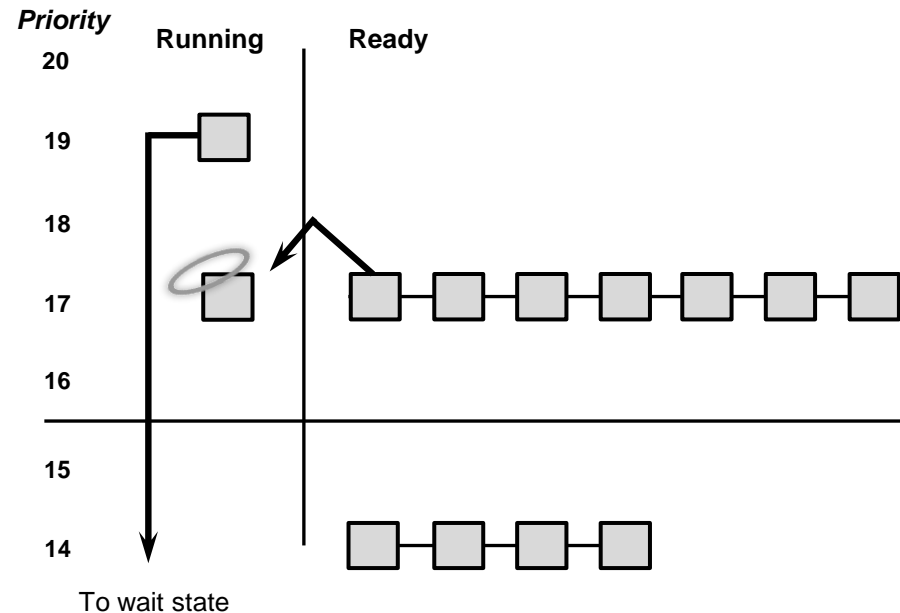
- ▶ *Dispatcher database:*
  - ▶ base de datos de hilos esperando para ejecutar y a qué proceso pertenecen
- ▶ *Dispatcher ready queue*
  - ▶ Una cola por nivel de prioridad
- ▶ *Ready summary*
  - ▶ Un bit por nivel
  - ▶ Si  $\text{bit}_i = 1 \rightarrow$  un hilo en ese nivel
  - ▶ Aumenta velocidad de búsqueda
- ▶ *Idle summary*
  - ▶ Un bit por procesador
  - ▶ Si  $\text{bit} = 1 \rightarrow$  procesador libre



# Planificación: escenarios (1 / 3)

## Windows 2000

- ▶ Cambio de contexto voluntario:
  - ▶ Entra en el estado de espera por algún objeto:
    - ▶ evento, mutex, semáforo, operación de E/S, etc.
  - ▶ Al terminar pasa al final de la cola de listos + *temporary priority boost*.
  - ▶ Rodaja de T: se mantiene

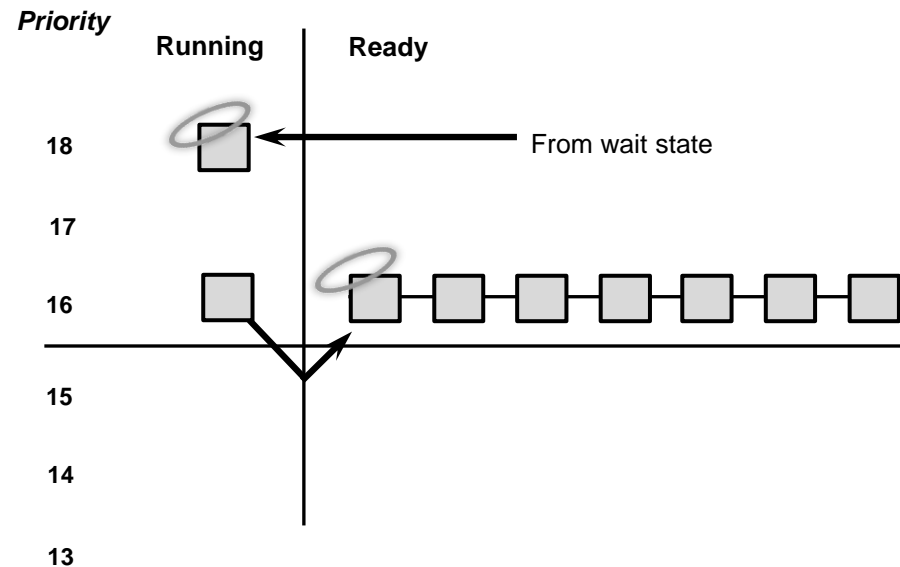


# Planificación: escenarios (2/3)

## Windows 2000

### ▶ Expulsión:

- ▶ Un hilo T de menor prioridad es expulsado cuando otro de mayor prioridad se vuelve listo para ejecutar
- ▶ T se pone a la cabeza de la cola de su prioridad
- ▶ Rodaja de T: si RT entonces se reinicia en caso contrario se mantiene

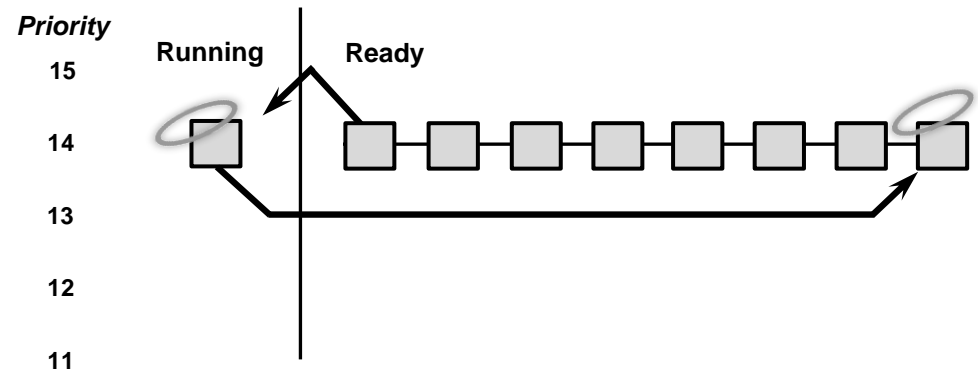


# Planificación: escenarios (3/3)

## Windows 2000

### ▶ Fin de rodaja:

- ▶ Un hilo T agota su rodaja de tiempo (quantum)
- ▶ Acciones del planificador:
  - ▶ Reducir la prioridad de T → otro hilo pasa a ejecutar
  - ▶ No reducir la prioridad → T pasa al último de la cola de su nivel (si vacía, vuelve de nuevo)
- ▶ Rodaja de T: se reinicia

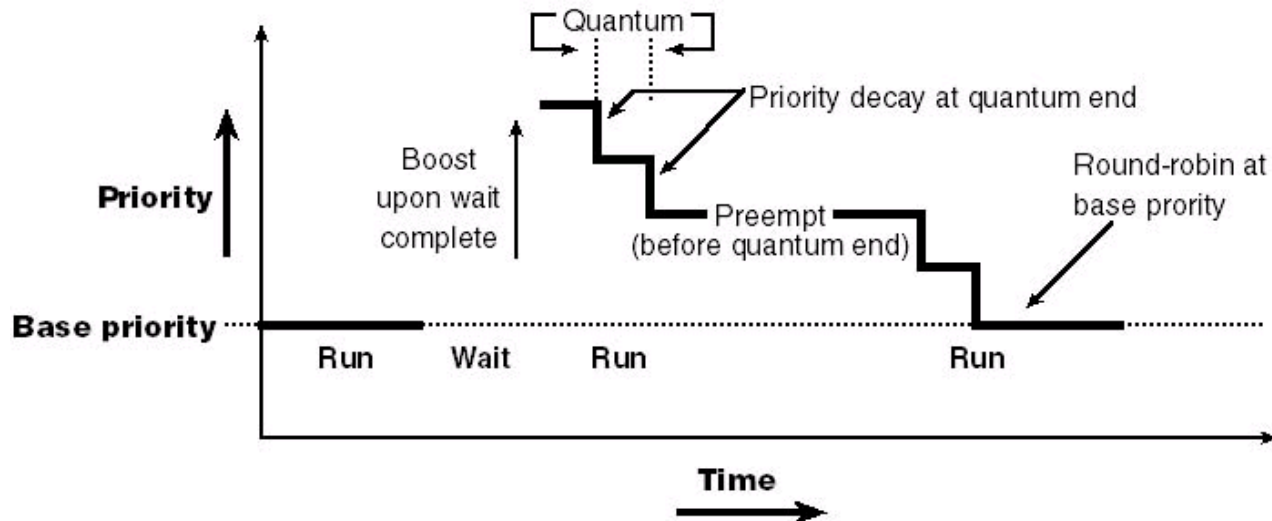


# Planificación: aumento de prioridad

## Windows 2000

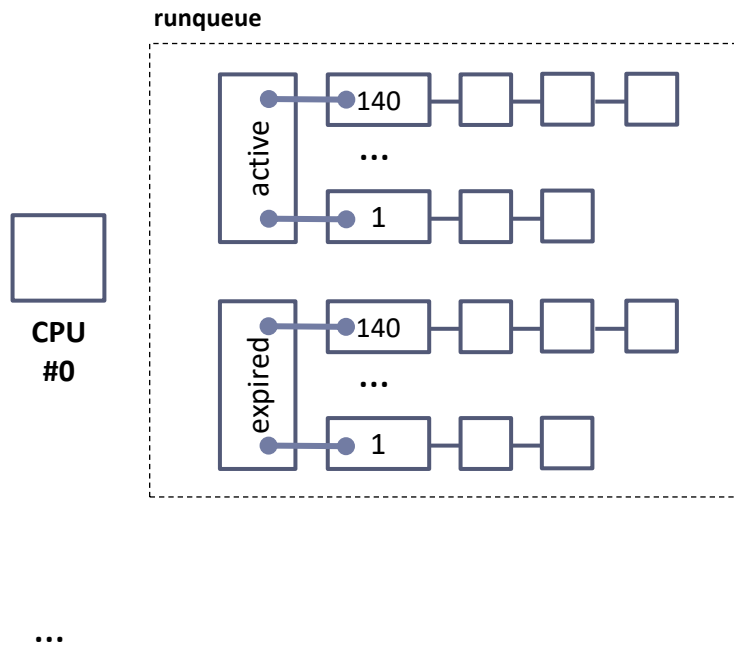
### ► *Priority boost:*

- Se aumenta la prioridad en ciertas ocasiones (solo en los niveles 0-15):
  - Cuando se completa una operación de E/S
  - Al salir del estado de una operación *wait*
  - Cuando el hilo lleva «mucho tiempo» en la cola de listo sin ejecutar:
    - El hilo de kernel *balance set manager* aumenta la prioridad por «envejecimiento»
    - Muestra 1 vez por segundo la cola de listos y si  $T.estado=READY$  más de 300 ticks (~3 ó 4 segundos) entonces  
 $T.prioridad = 15$   
 $T.rodaja = 2 * rodaja\_normal$



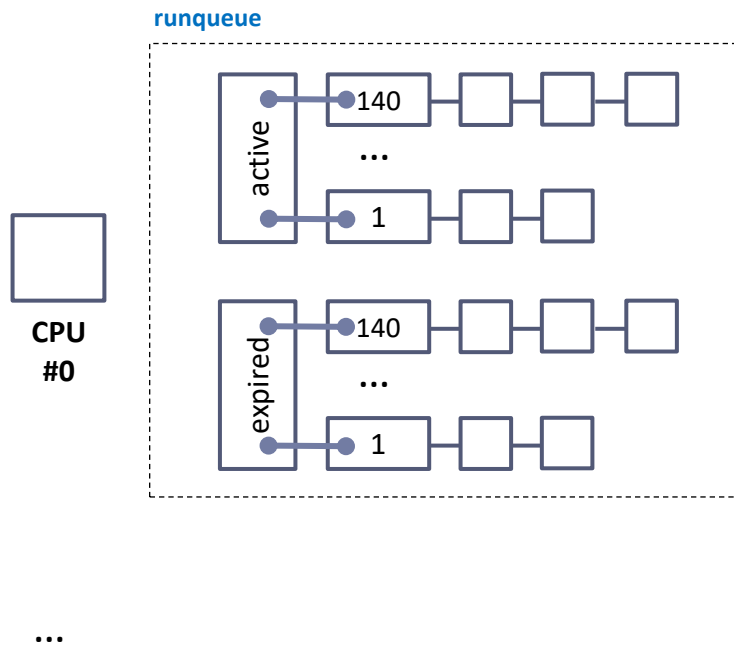
# Planificación: estructura de datos

## Linux



- ▶ Kernel/sched.c
- ▶ Cada procesador tiene su propio *runqueue*
- ▶ Cada *runqueue* tiene dos vectores de prioridad:
  - ▶ Activo y Expirado
- ▶ Cada vector de prioridad tiene 140 listas:
  - ▶ Una por nivel de prioridad
  - ▶ Incluye 100 niveles de tiempo real

# Planificación: gestión Linux



- ▶ El planificador elige los procesos de la lista de activos de acuerdo a su prioridad
- ▶ Cuando expira la rodaja de un proceso, lo mueve a la lista de Expirado
  - ▶ Se recalcula prioridad y rodaja
- ▶ Cuando la lista de activos está vacía, el planificador intercambia las listas de activo y expirados
- ▶ Si un proceso es suficientemente interactivo permanecerá en la lista de activos

Grupo ARCOS  
Departamento de Informática  
Universidad Carlos III de Madrid

# Lección 3b

procesos, periféricos, *drivers* y servicios ampliados

Diseño de Sistemas Operativos  
Grado en Ingeniería Informática y Doble Grado I.I. y A.D.E.

