Grupo ARCOS
Universidad Carlos III de Madrid

# Lección 3b

## procesos, periférico, *drivers* y servicios ampliados

Diseño de Sistemas Operativos
Grado en Ingeniería Informática

# Lecturas recomendadas

## Base


1. Carretero 2007:
   1. Cap.7

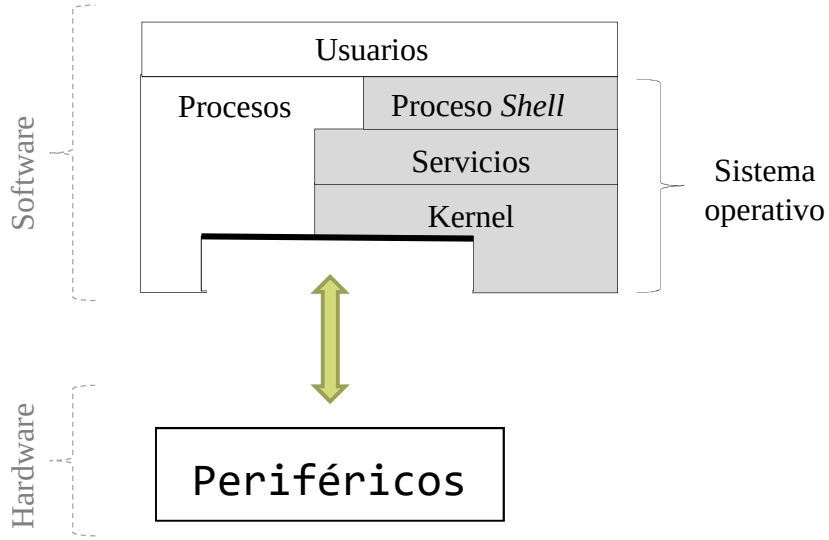
## Recomendada


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

# A recordar…

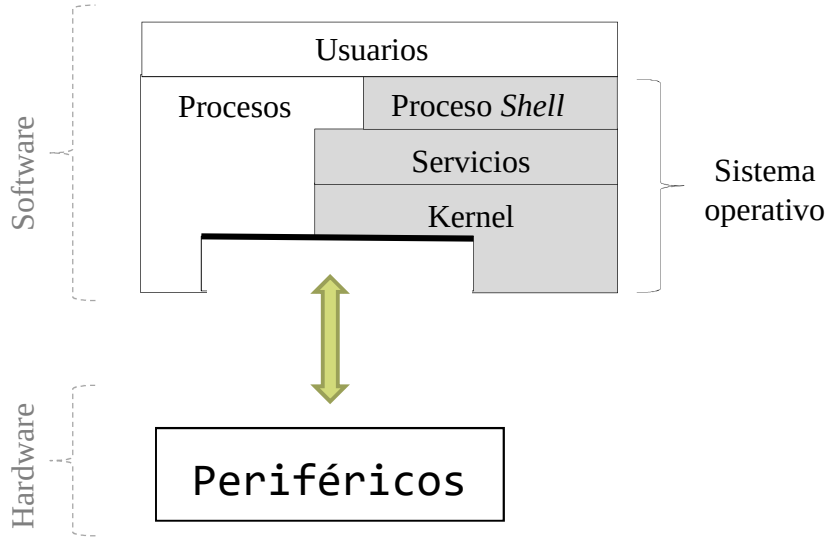
1. Estudiar la teoría asociada.
   - Estudiar el material asociado a la bibliografía: las transparencias solo no son suficiente.
   - Crear cuestiones con sus respuestas y justificación.
2. Repasar lo visto en clase.
   - Realizar el cuaderno de prácticas progresivamente.
3. Ejercitar las competencias.
   - Realizar las prácticas progresivamente.
   - Realizar todos los ejercicios posibles.

# Contenidos




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

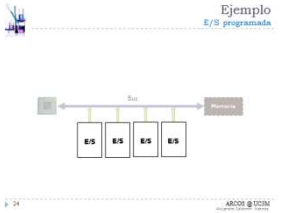
# Contenidos

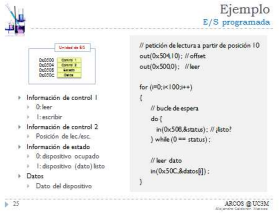


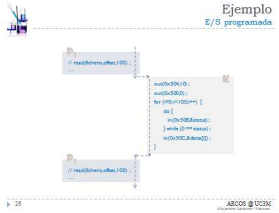

- **Introducción**
- C.C.V.
- Temporización y C.C.I.
- Planificación

# Impacto en el sistema operativo en el tratamiento de dispositivos

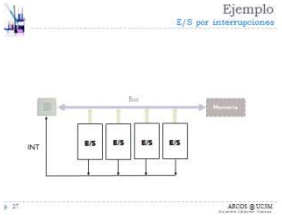
- **E/S programada o directa**

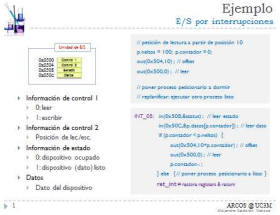


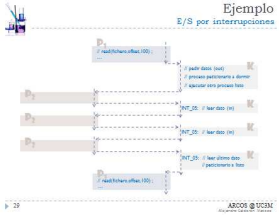



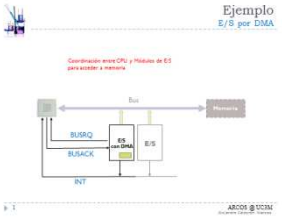

- **E/S por interrupciones**

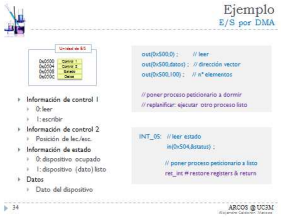


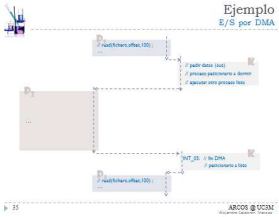



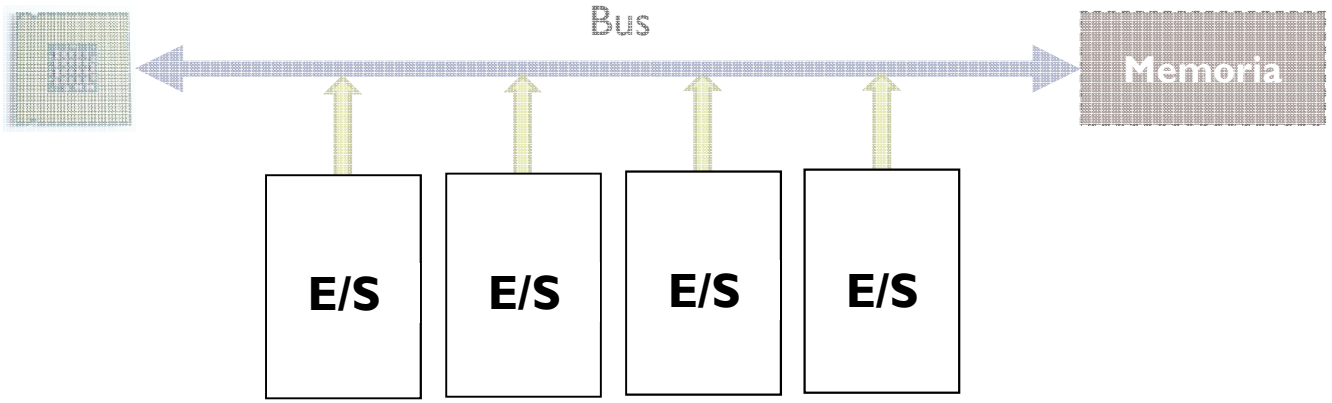

- **E/S por DMA**



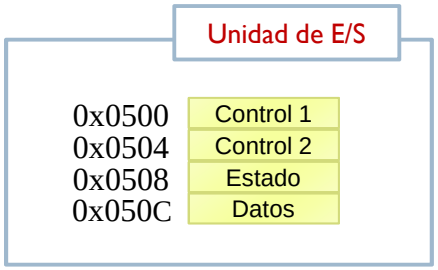



# Ejemplo
## E/S programada



# Ejemplo
## E/S programada




- Información de control 1
  - 0: leer
  - 1: escribir
- Información de control 2
  - Posición de lec./esc.
- Información de estado
  - 0: dispositivo ocupado
  - 1: dispositivo (dato) listo
- Datos
  - Dato del dispositivo

```
petición: for (i=0; i<100;i++)
{
      // pedir leer a partir de la posición 10
      out(0x504,10) ;
      out(0x500,0) ;

      // bucle de espera
      do {
         in(0x508,&status) ;  // ¿listo?
      } while (0 == status) ;

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

# Ejemplo
## E/S programada

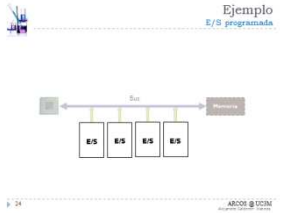
P1
```
// read(fichero,offset,100) ;
…
```

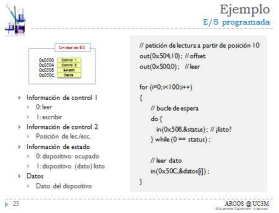
K
```
out(0x504,10) ;
out(0x500,0) ;
for (i=0; i<100;i++)  {
      do {
         in(0x508,&status) ;
      } while (0 == status) ;
      in(0x50C,&datos[i]) ;
}
```

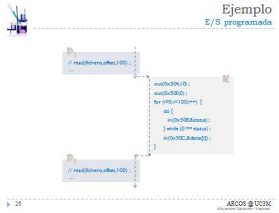
P1
```
// read(fichero,offset,100) ;
…
```

# Impacto en el sistema operativo en el tratamiento de dispositivos

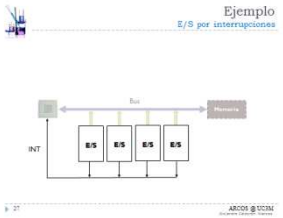
- **E/S programada o directa**

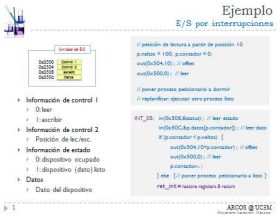


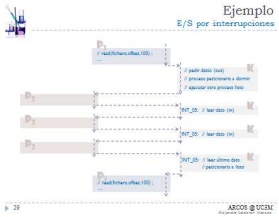




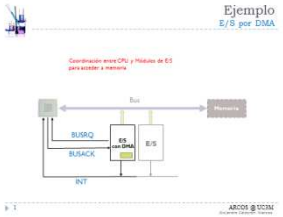

- **E/S por interrupciones**

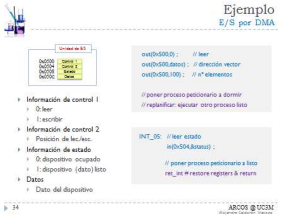


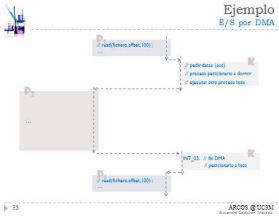




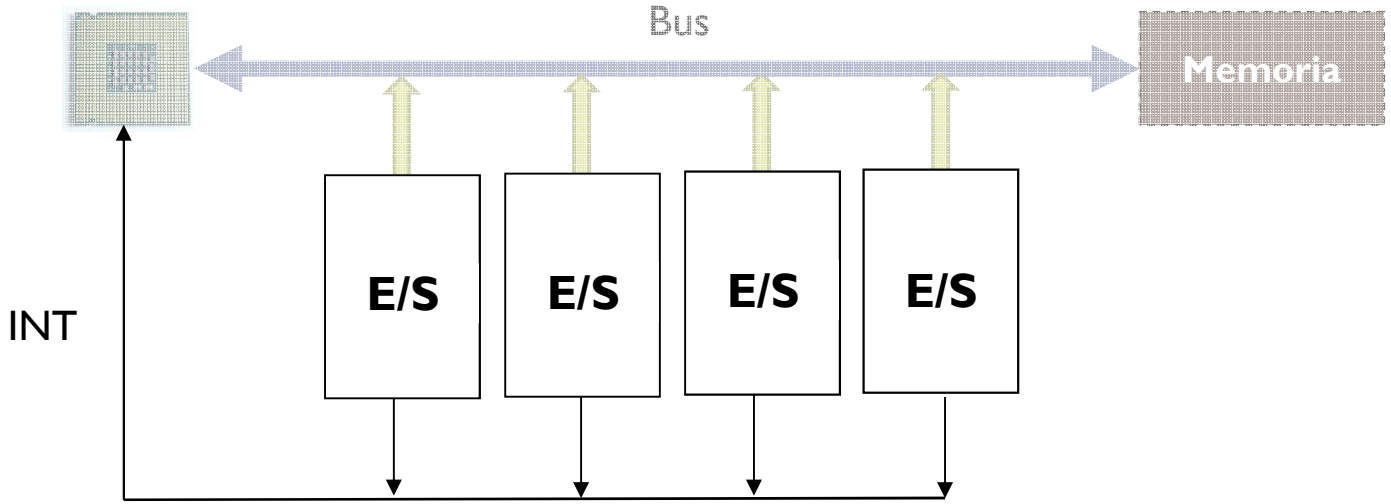

- **E/S por DMA**



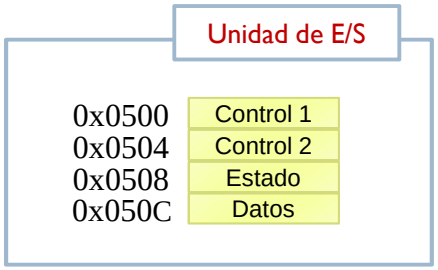




# Ejemplo
## E/S por interrupciones



# Ejemplo
## E/S por interrupciones

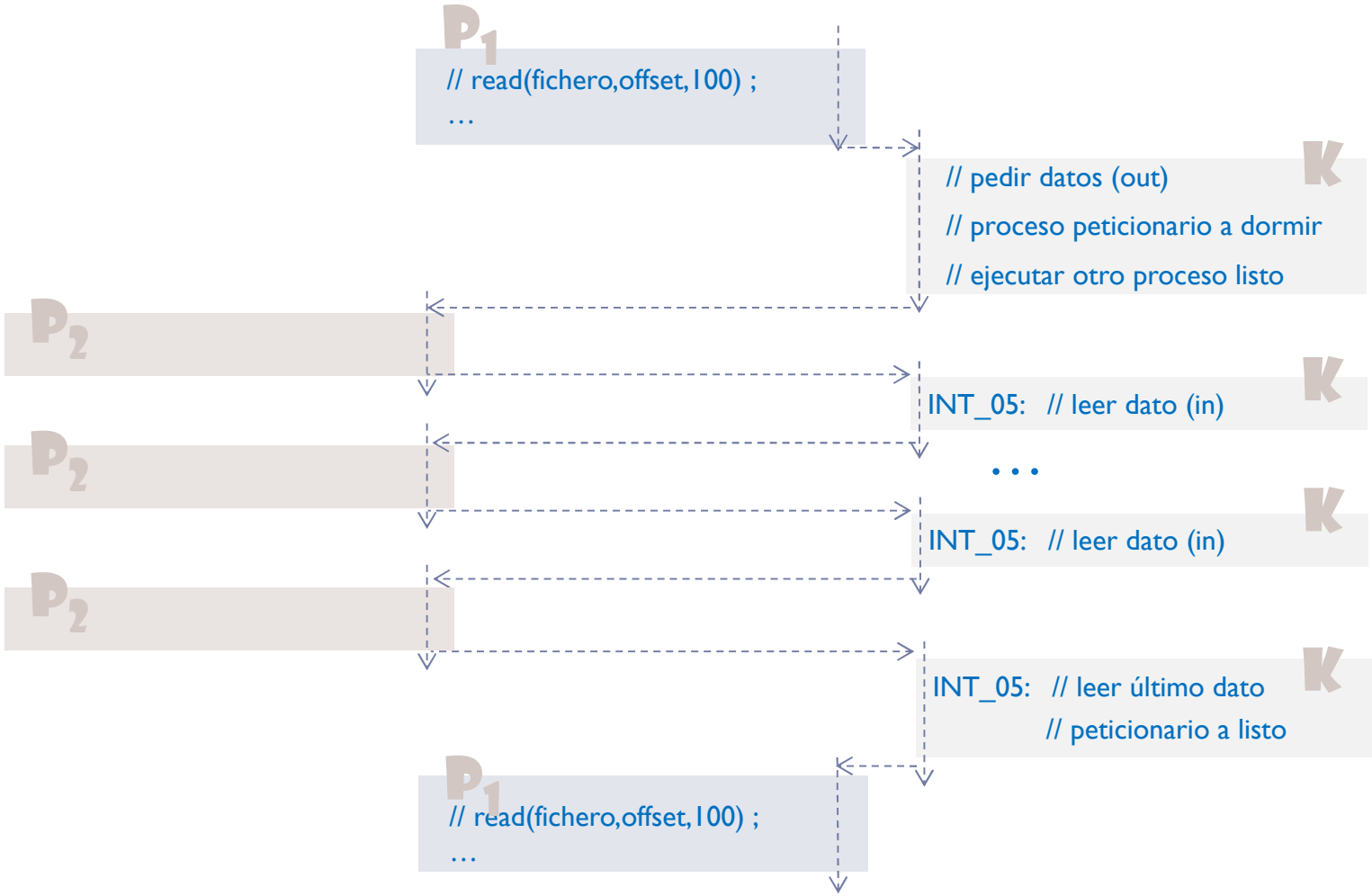



- Información de control 1
  - 0: leer
  - 1: escribir
- Información de control 2
  - Posición de lec./esc.
- Información de estado
  - 0: dispositivo ocupado
  - 1: dispositivo (dato) listo
- Datos
  - Dato del dispositivo

```
petición:
// lectura a partir de posición 10
   p.neltos = 100;  p.contador = 0;
   out(0x504,10) ;  // offset
   out(0x500,0) ;   // leer
// cambio de contexto voluntario (C.C.V.)
```

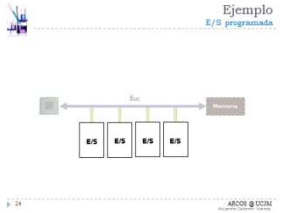
```
INT_05:  in(0x508,&status) ;   // leer estado
         in(0x50C,&p.datos[p.contador]) ;  // leer dato
         if (p.contador < p.neltos)  {
               out(0x504,10+p.contador) ;  // offset
               out(0x500,0) ;  // leer
               p.contador++ ;
        } else  { // poner proceso peticionario a listo }
        ret_int # restore registers & return
```

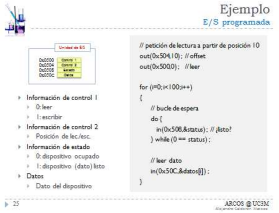
# Ejemplo
## E/S por interrupciones

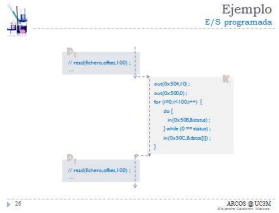


# Impacto en el sistema operativo en el tratamiento de dispositivos

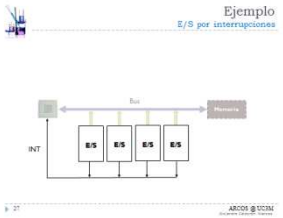
- **E/S programada o directa**

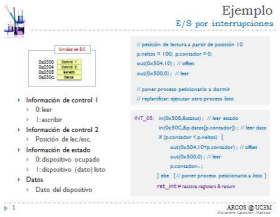


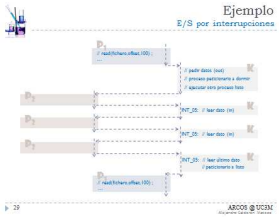




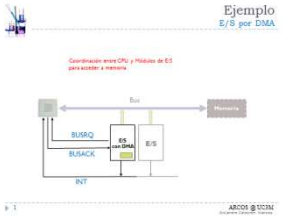

- **E/S por interrupciones**

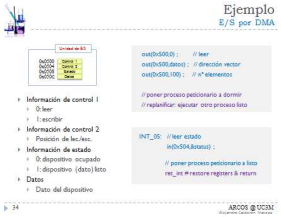


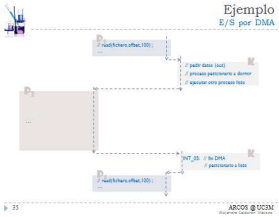





- **E/S por DMA**

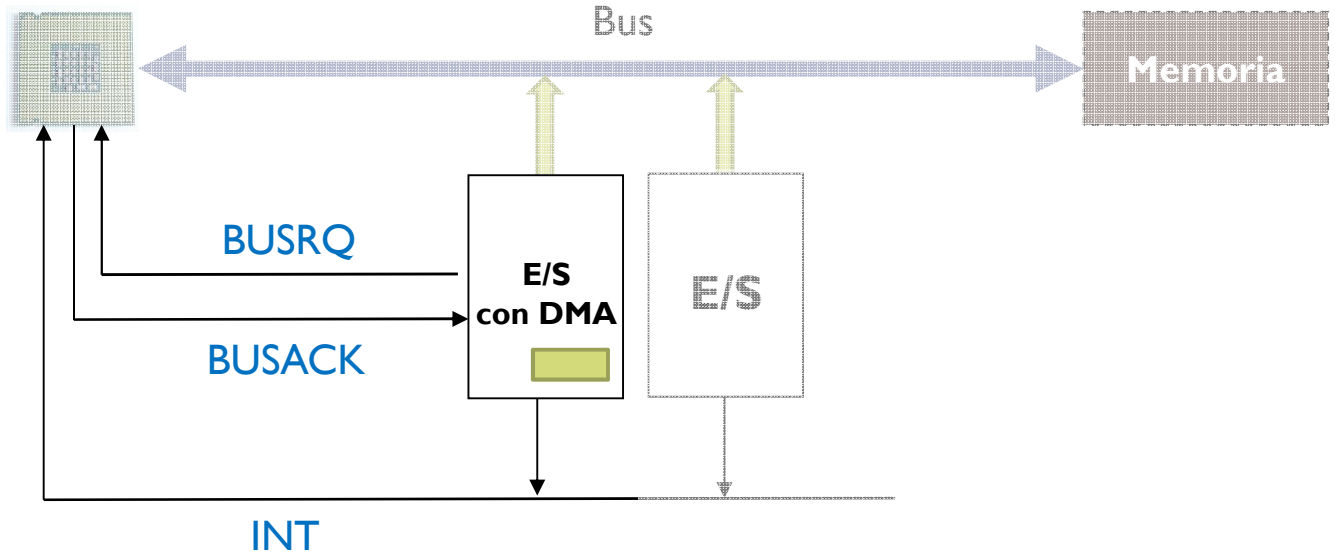






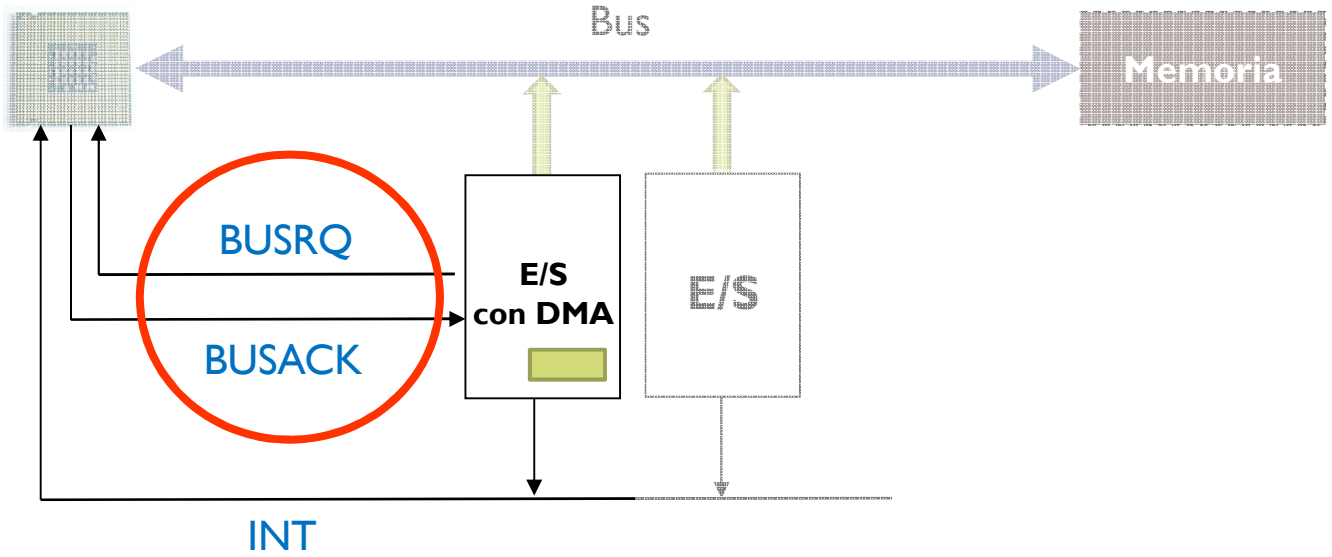
# Ejemplo
## E/S por DMA

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



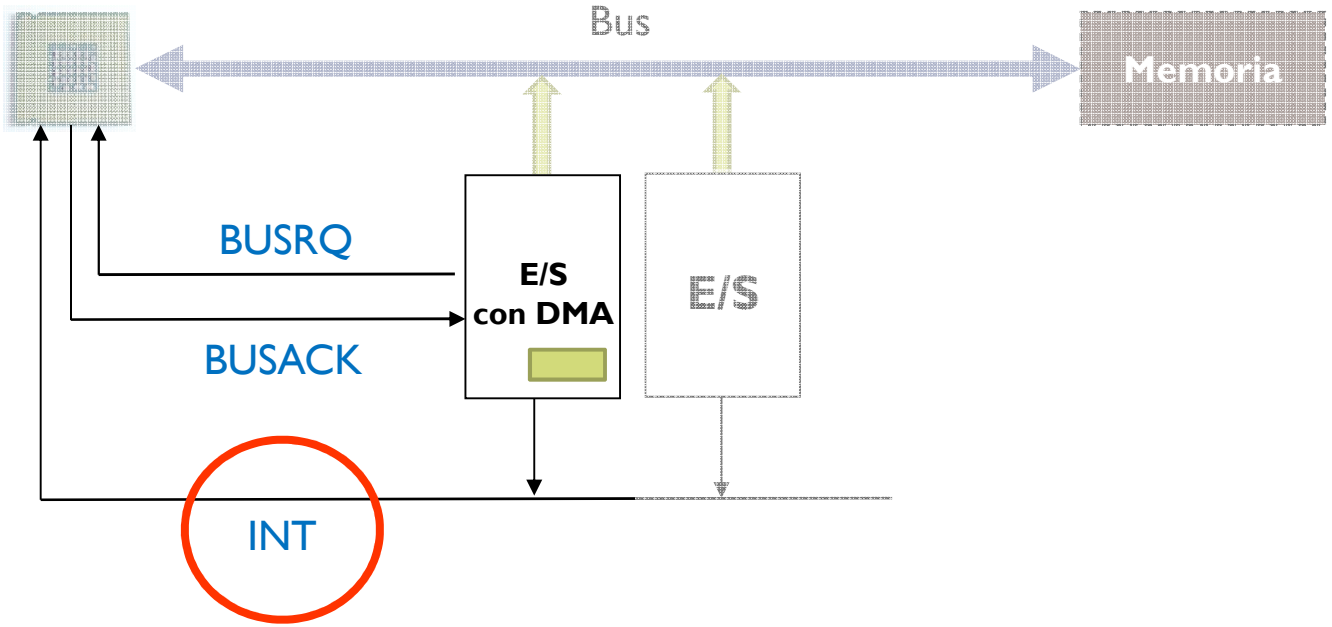
# Ejemplo
## E/S por DMA

Cada dato transferido a memoria supone:
- Pedir permiso para acceder a memoria (BUSRQ)
- Esperar el permiso (BUSACK)
- Transfiere a memoria
- Desactiva petición de permiso (BUSRQ)

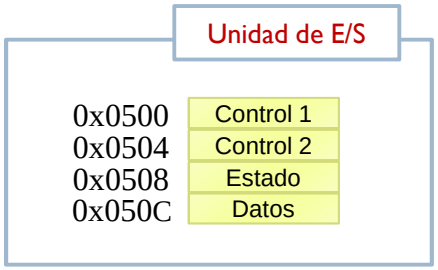


# Ejemplo
## E/S por DMA

Una vez transferido todos los datos:
- Generar una interrupción (INT) para avisar a la CPU



# Ejemplo
## E/S por DMA

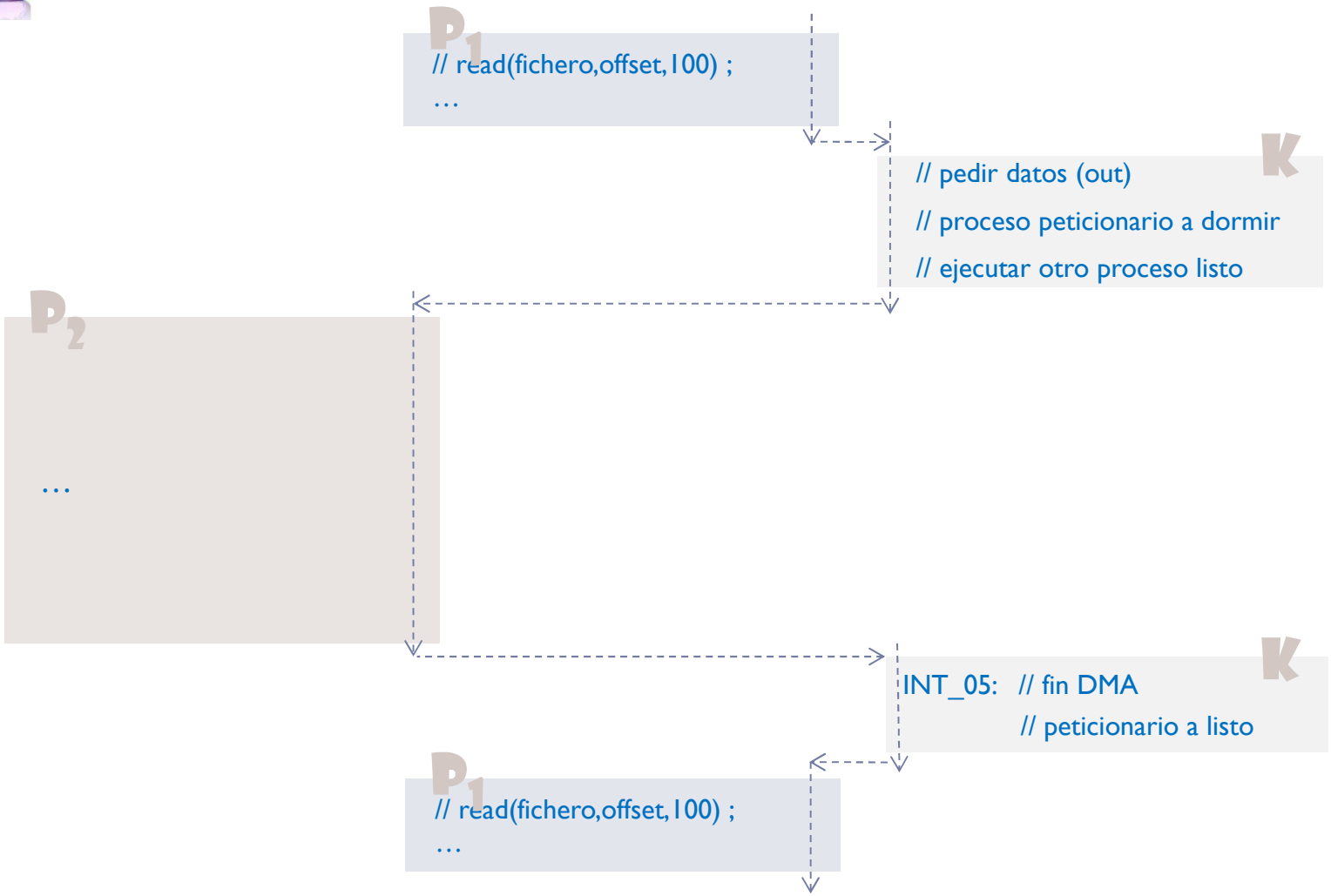



- Información de control 1
  - 0: leer
  - 1: escribir
- Información de control 2
  - Posición de lec./esc.
- Información de estado
  - 0: dispositivo ocupado
  - 1: dispositivo (dato) listo
- Datos
  - Dato del dispositivo

```
petición:
// Programar la petición del bloque
   out(0x500,0) ;        // leer
   out(0x500,datos) ;    // dirección vector
   out(0x500,100) ;      // nº elementos
   out(0x504,10) ;       // posición lec./esc.
// Cambio de contexto voluntario (C.C.V.)
```

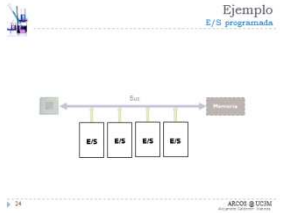
```
INT_05:   // leer estado y datos
          in(0x508,  &status) ;
          in(0x50C, &status) ;

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

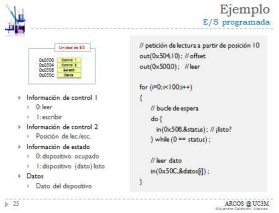
# Ejemplo
## E/S por DMA

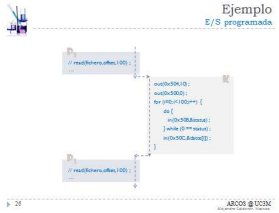


# Aprovechar mejor los tiempos de espera

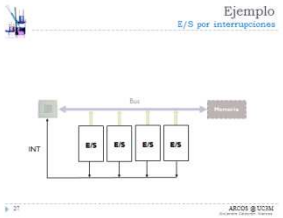
- **E/S programada o directa**

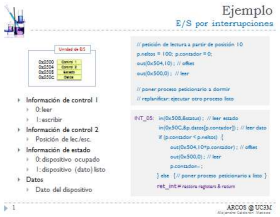


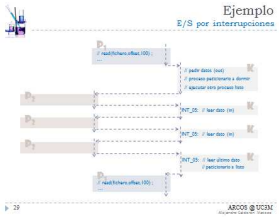



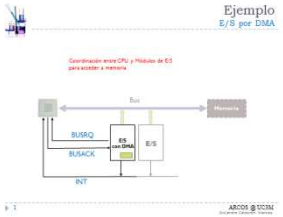

- **E/S por interrupciones**

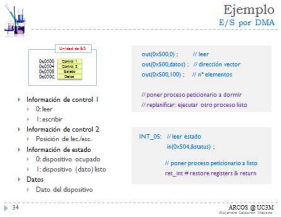


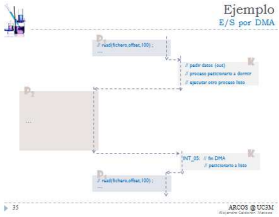




- **E/S por DMA**

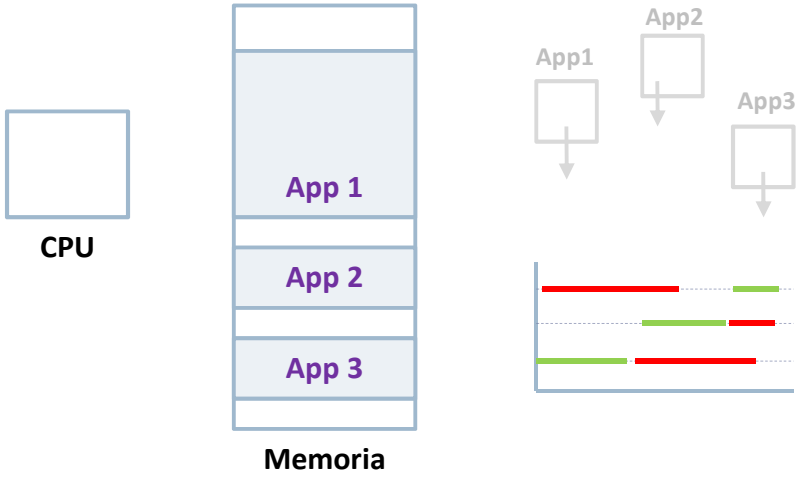


# Modelo ofrecido

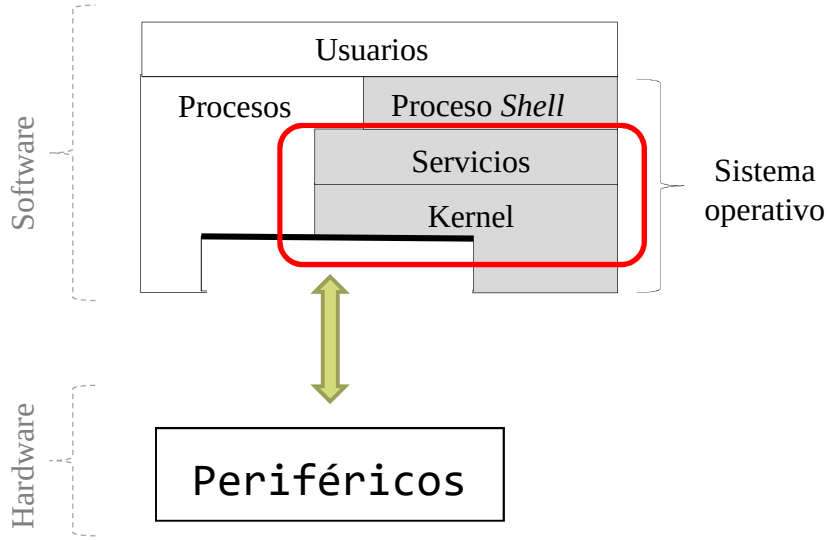
- recursos
- **multiprogramación**
  - protección/compartición
  - jerarquía de procesos
- multitarea
- multiproceso




## 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

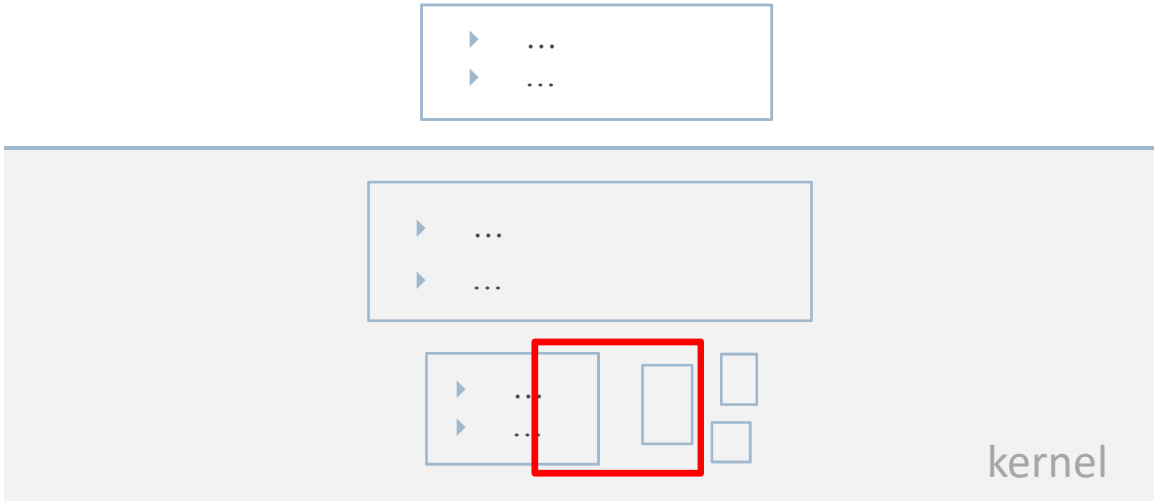
# Contenidos



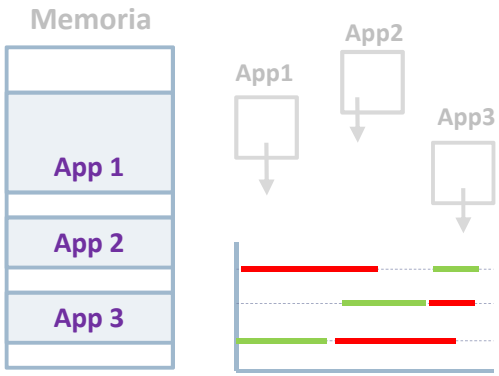

- Introducción
- **C.C.V.**
- Temporización y C.C.I.

# Multiprogramación (datos y funciones)

| Requisitos | Información (en estructuras de datos) | Funciones (internas, servicio y API) |
|---|---|---|
| Multiprogramación | • Estado de ejecución<br>• Contexto: registros de CPU…<br>• Lista de procesos | • Int. hw/sw de dispositivos<br>• Planificador<br>• Crear/Destruir/Planificar proceso |



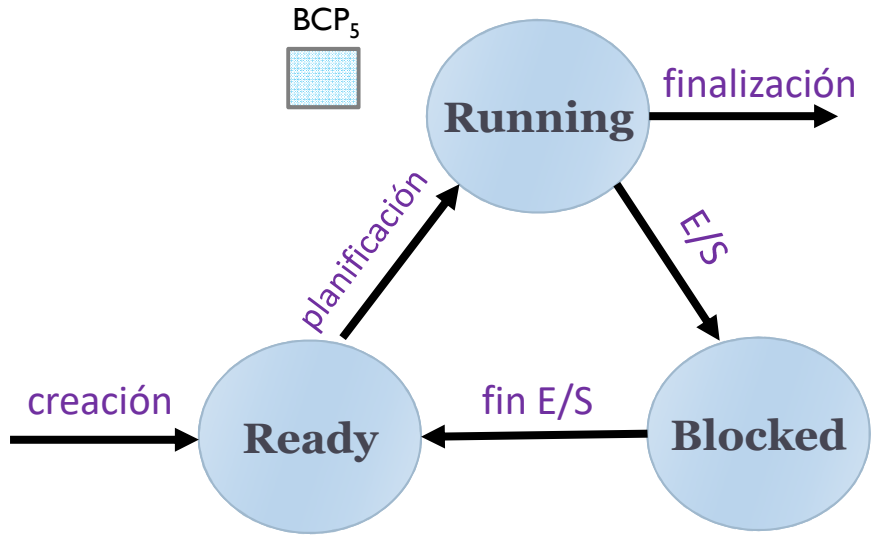
# Multiprogramación

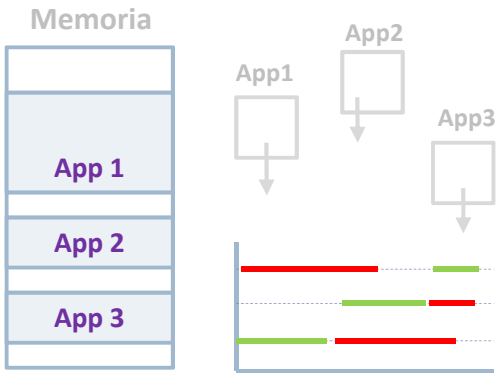



- Tener varias aplicaciones en memoria
- Si una aplicación se bloquea por E/S, entonces se ejecuta otra (hasta que quede bloqueada)
  - Cambio de contexto voluntario (C.C.V.)

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

- Estado
- Lista/Cola
- Contexto

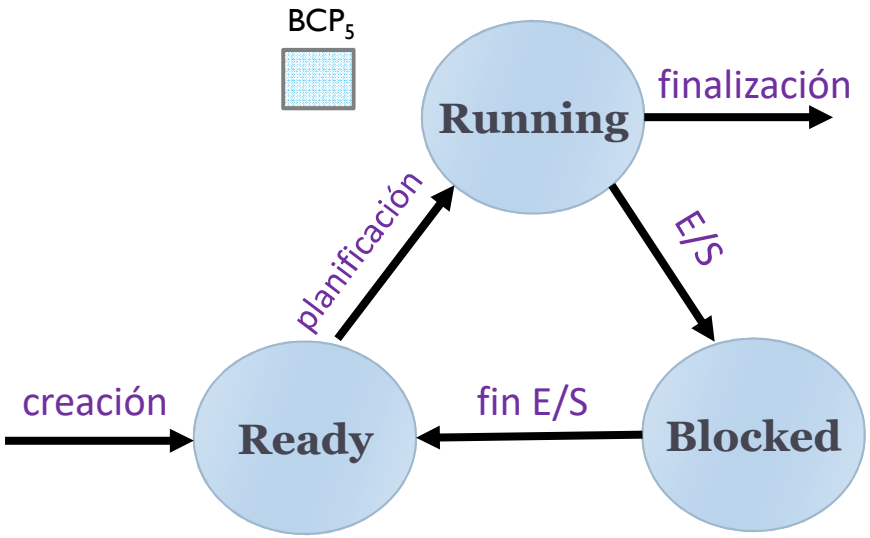



- Tener varias aplicaciones en memoria
- Si una aplicación se bloquea por E/S, entonces se ejecuta otra (hasta que quede bloqueada)
  - Cambio de contexto voluntario (C.C.V.)

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

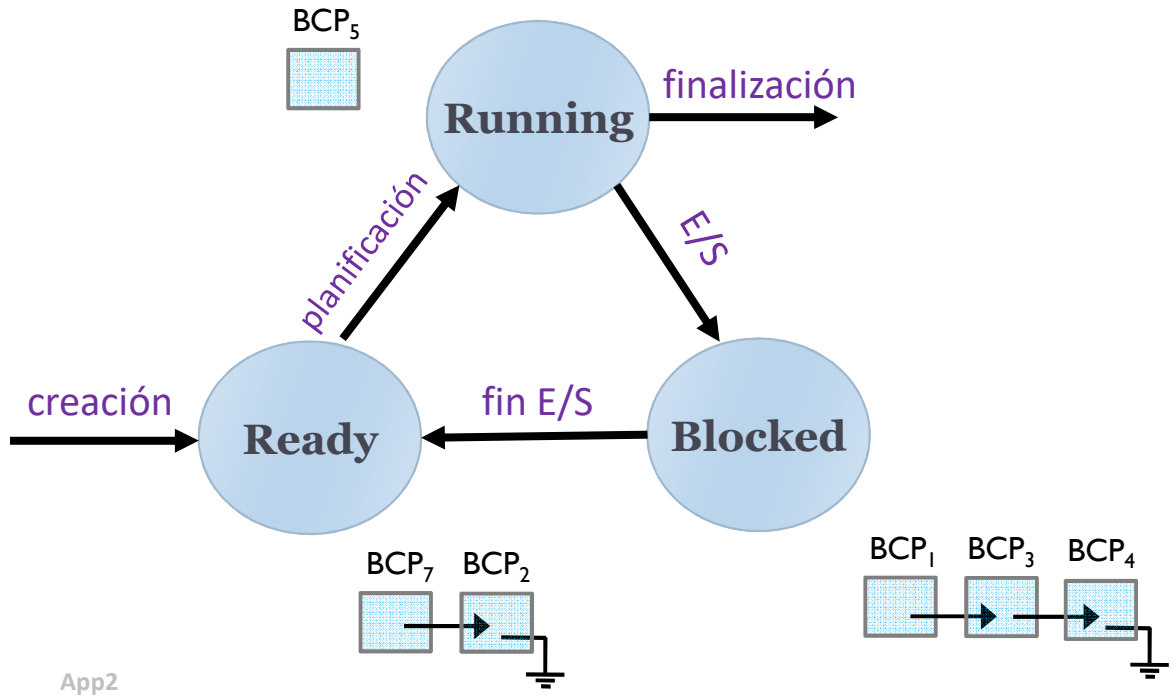
- Estado
- Lista/Cola
- Contexto

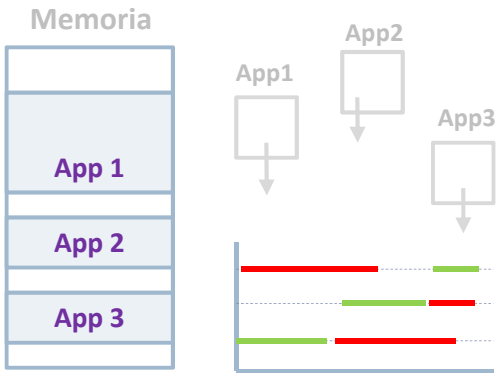



- En ejecución: con CPU asignada
- Listo para ejecutar: no procesador disponible para él
- Bloqueado: esperando un evento
- Suspendido y listo: expulsado pero listo para ejecutar
- Suspendido y bloqueado: expulsado y esperando evento

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

- Estado
- **Lista/Cola**
- Contexto

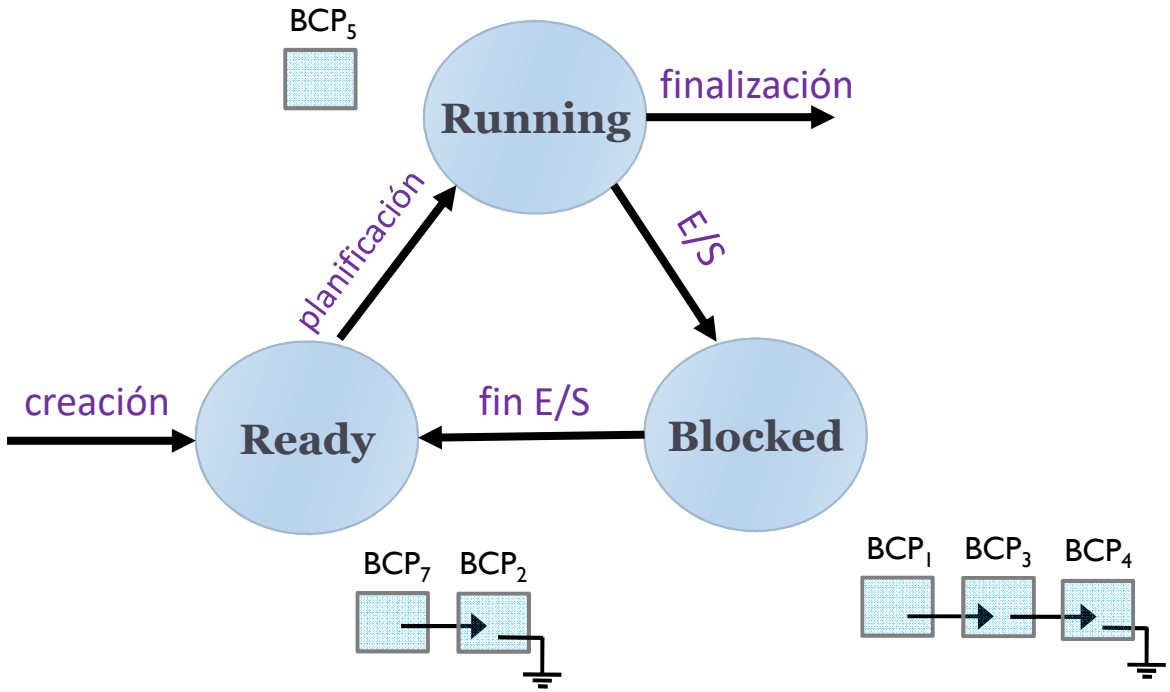



- Tener varias aplicaciones en memoria
- Si una aplicación se bloquea por E/S, entonces se ejecuta otra (hasta que quede bloqueada)
  - Cambio de contexto voluntario (C.C.V.)

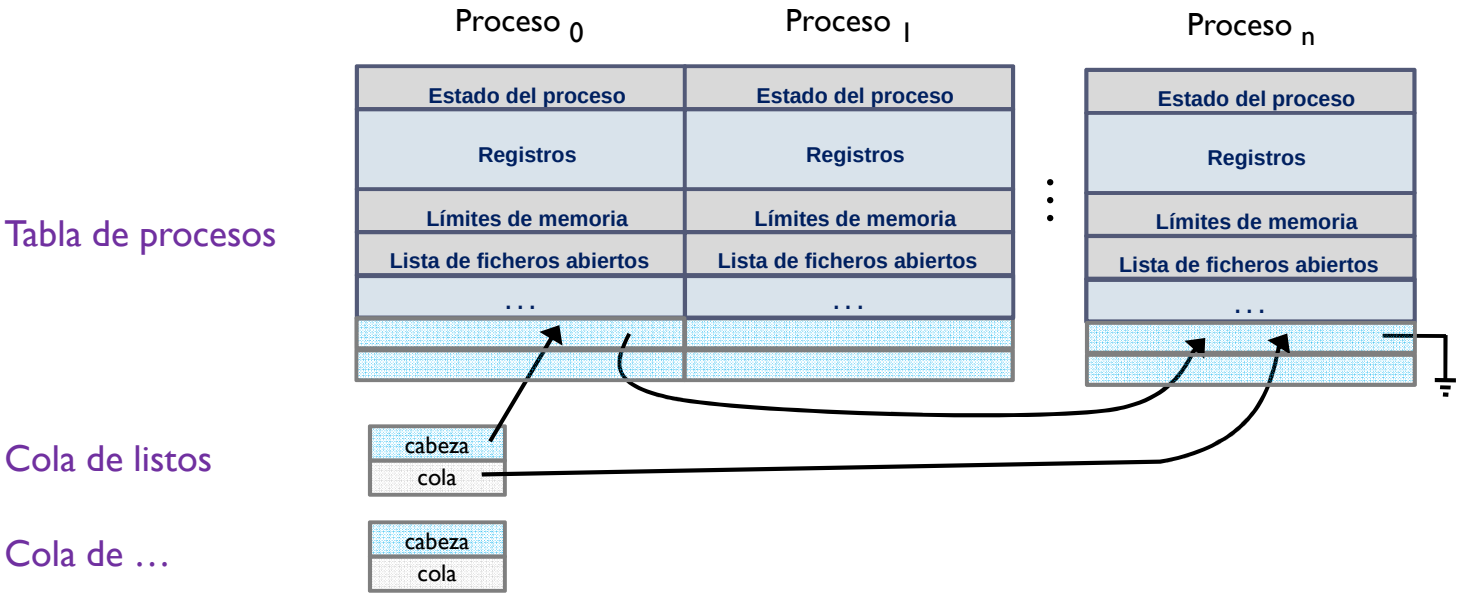
# Multiprogramación (datos)
## Lista/Colas de procesos (c.c.v.)

- 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)

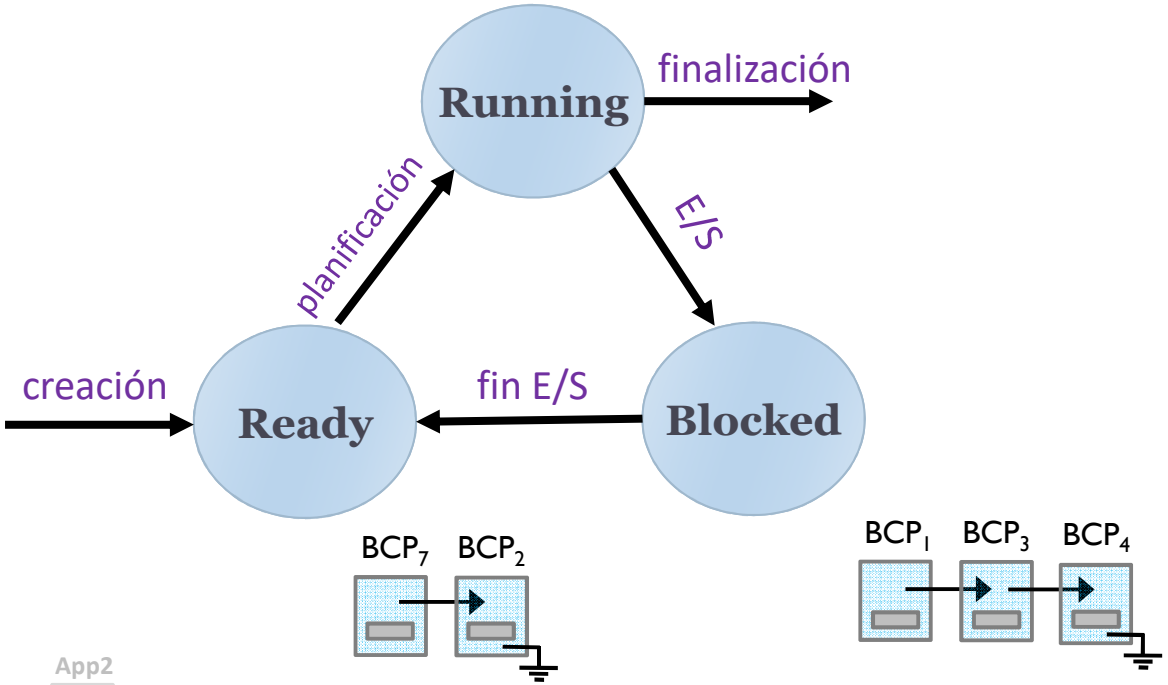
# Implementación de las colas de procesos

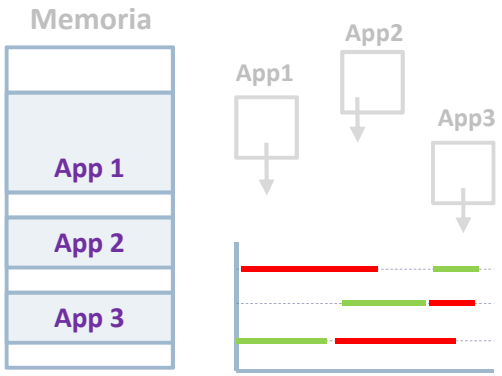



- Cola de listos: procesos esperando a ejecutar en CPU
- Cola de bloqueados por recurso: procesos a la espera de finalizar una petición bloqueante al recurso asociado
  - Un proceso solo puede estar en una cola (como mucho)

# Multiprogramación (datos)
## Contexto de un proceso

- Estado
- Lista/Cola
- **Contexto**

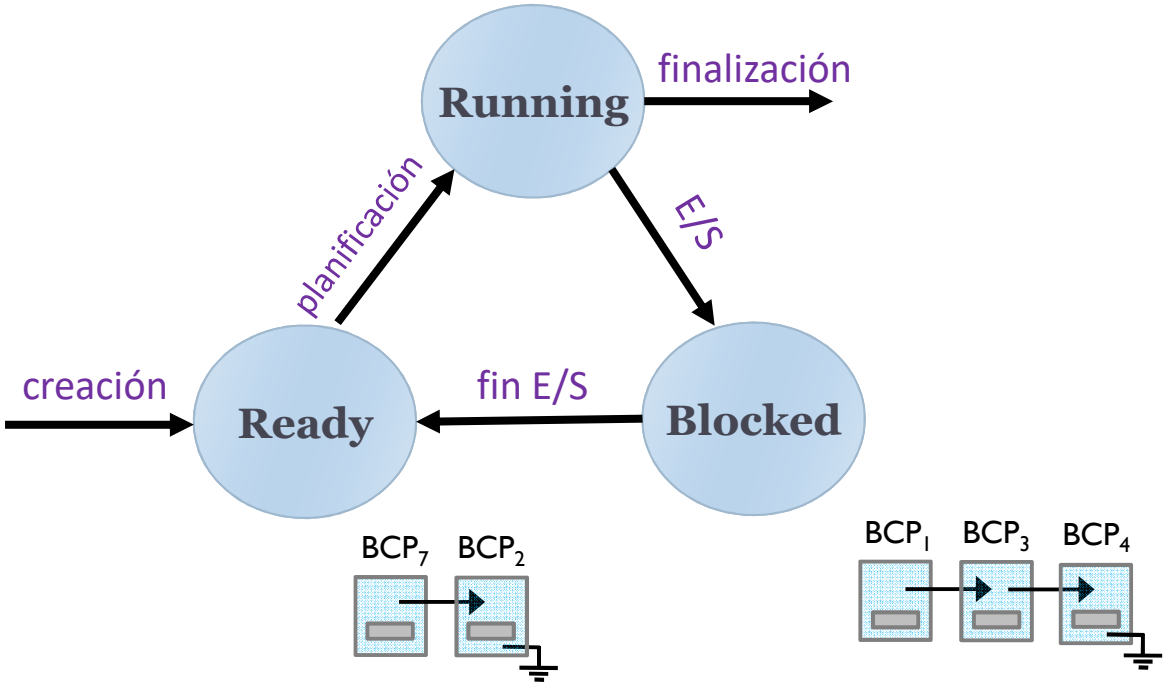



- Tener varias aplicaciones en memoria
- Si una aplicación se bloquea por E/S, entonces se ejecuta otra (hasta que quede bloqueada)
  - Cambio de contexto voluntario (C.C.V.)

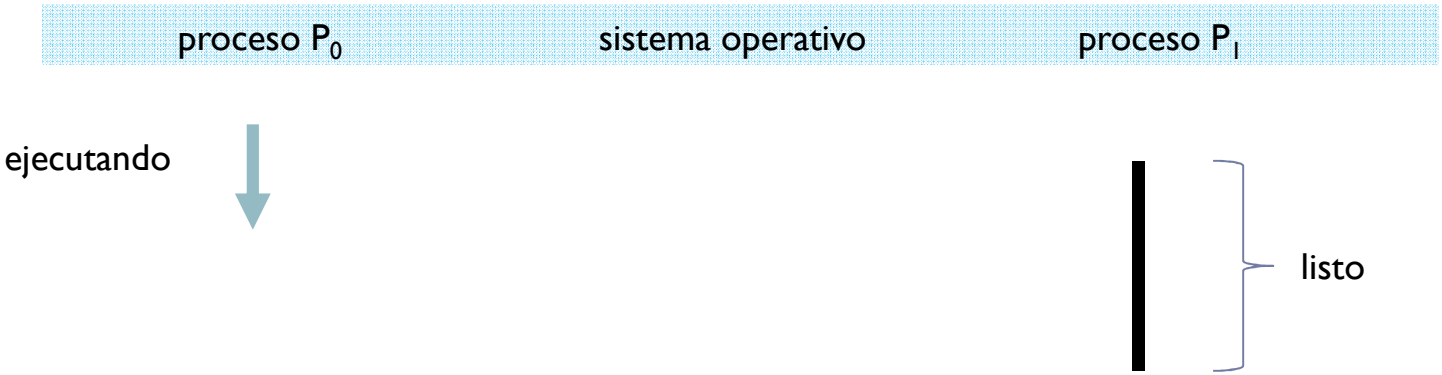
# Multiprogramación (datos)
## Contexto de un proceso

- Estado
- Lista/Cola
- **Contexto**

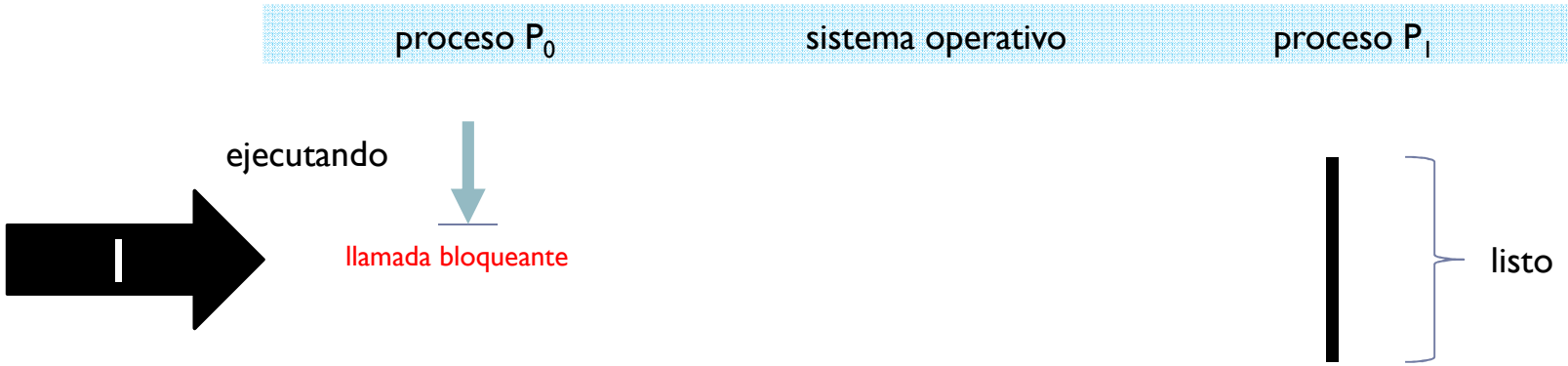



- Registros generales: PC, RE, etc.
- Registros específicos: Registros de coma flotante, etc.
- Referencias a recursos: puntero a código, datos, etc.

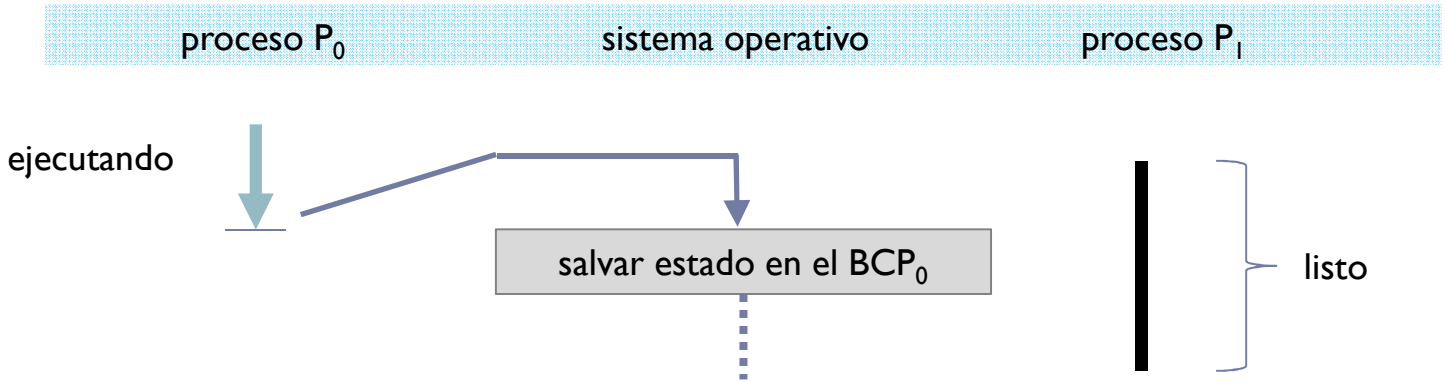
# Multiprogramación: ejemplo de ejecución



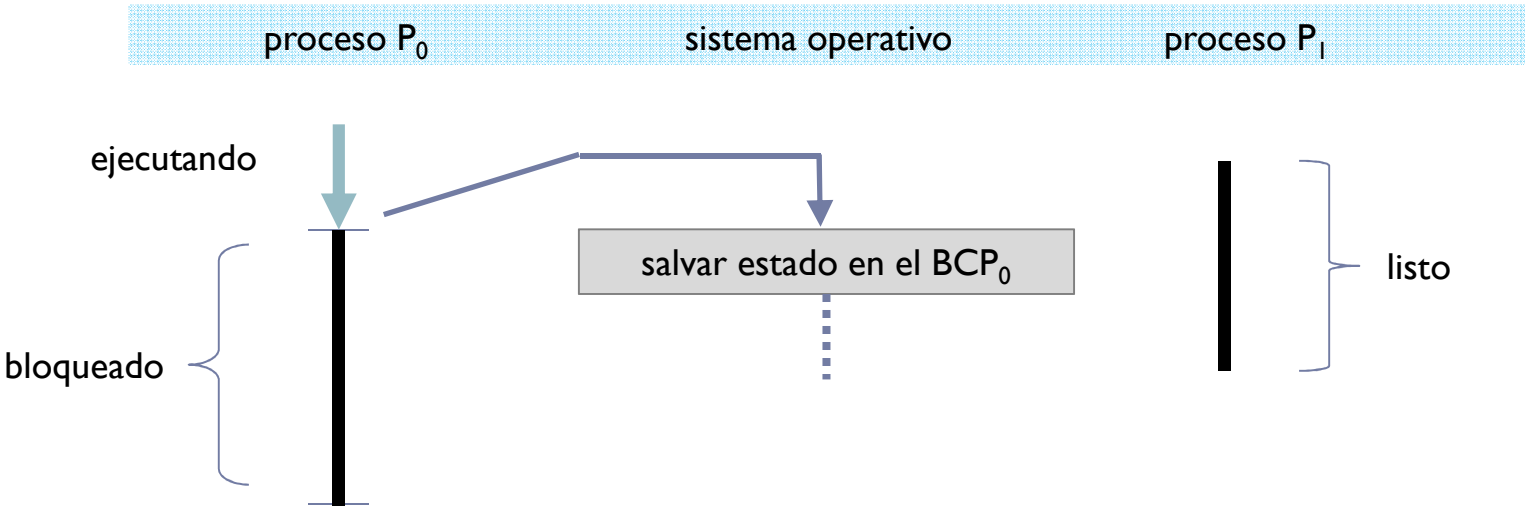
# Multiprogramación: ejemplo de ejecución



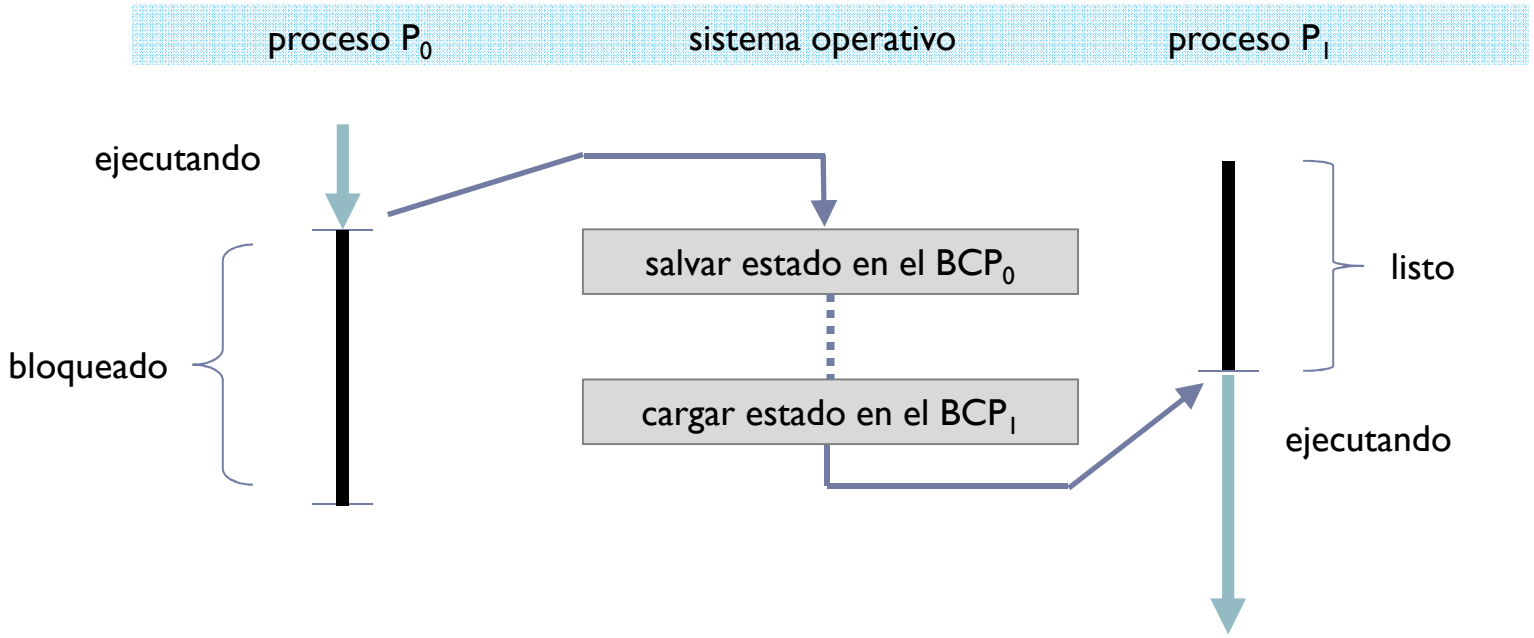
# Multiprogramación: ejemplo de ejecución



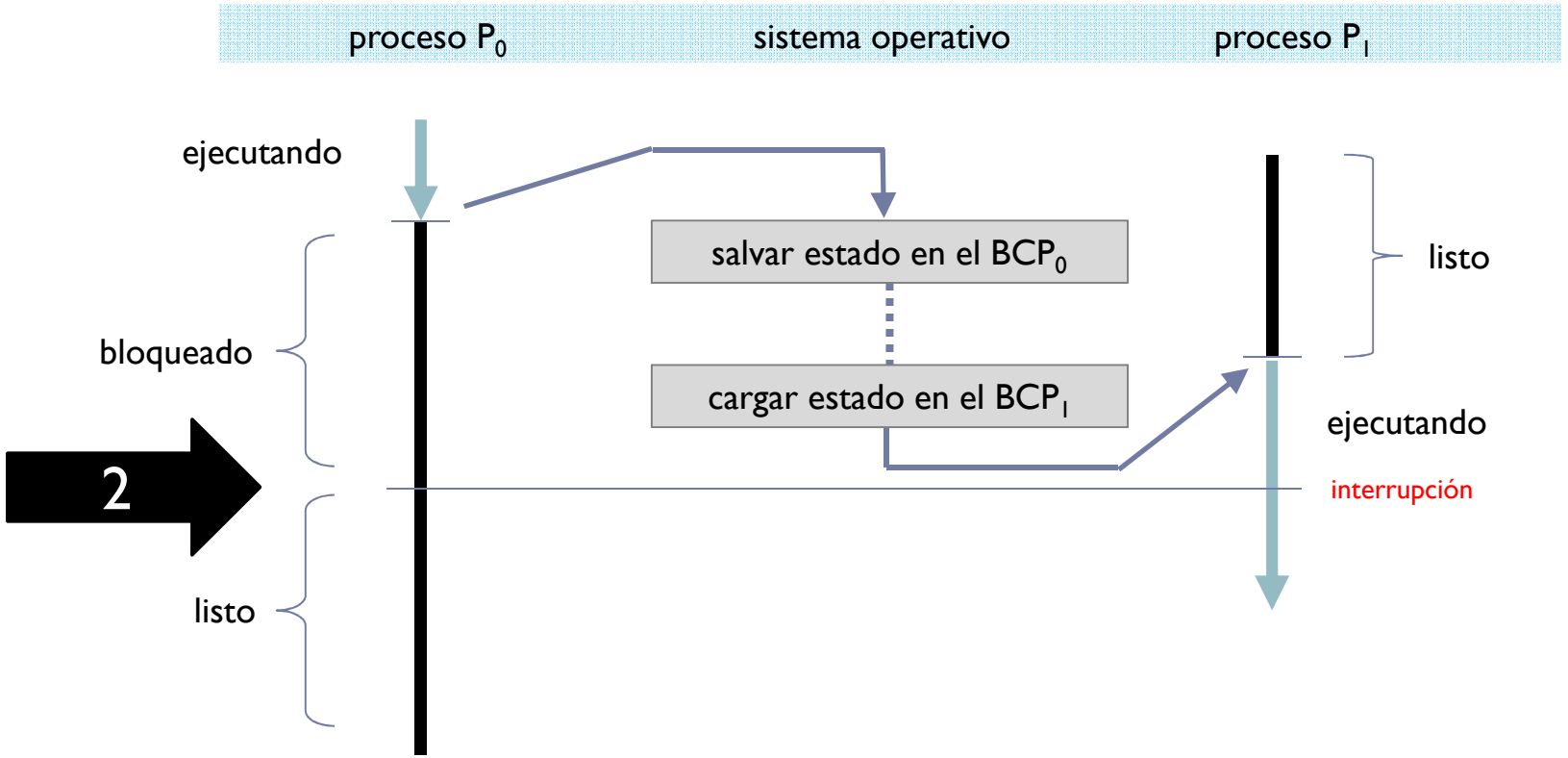
# Multiprogramación: ejemplo de ejecución



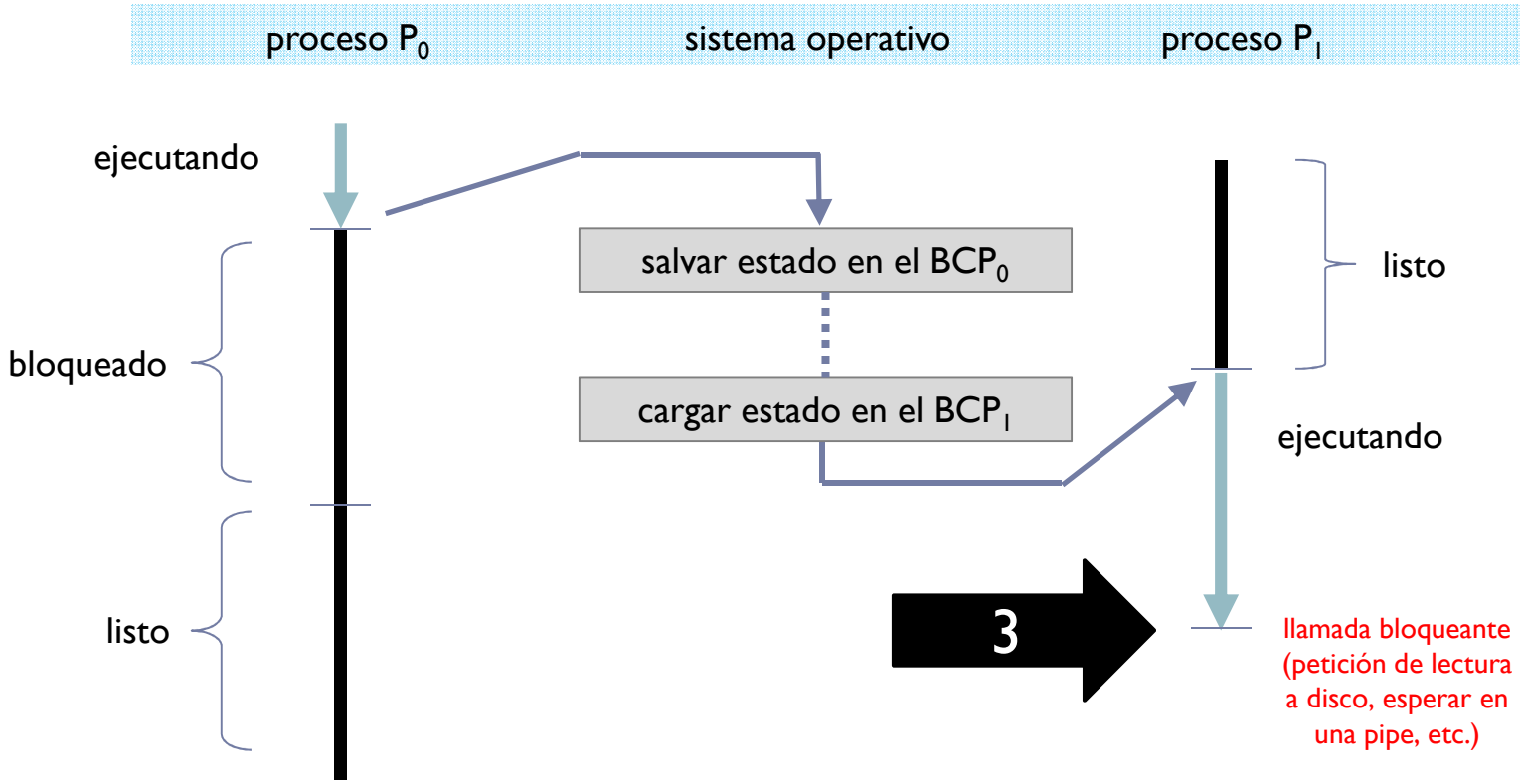
# Multiprogramación: ejemplo de ejecución



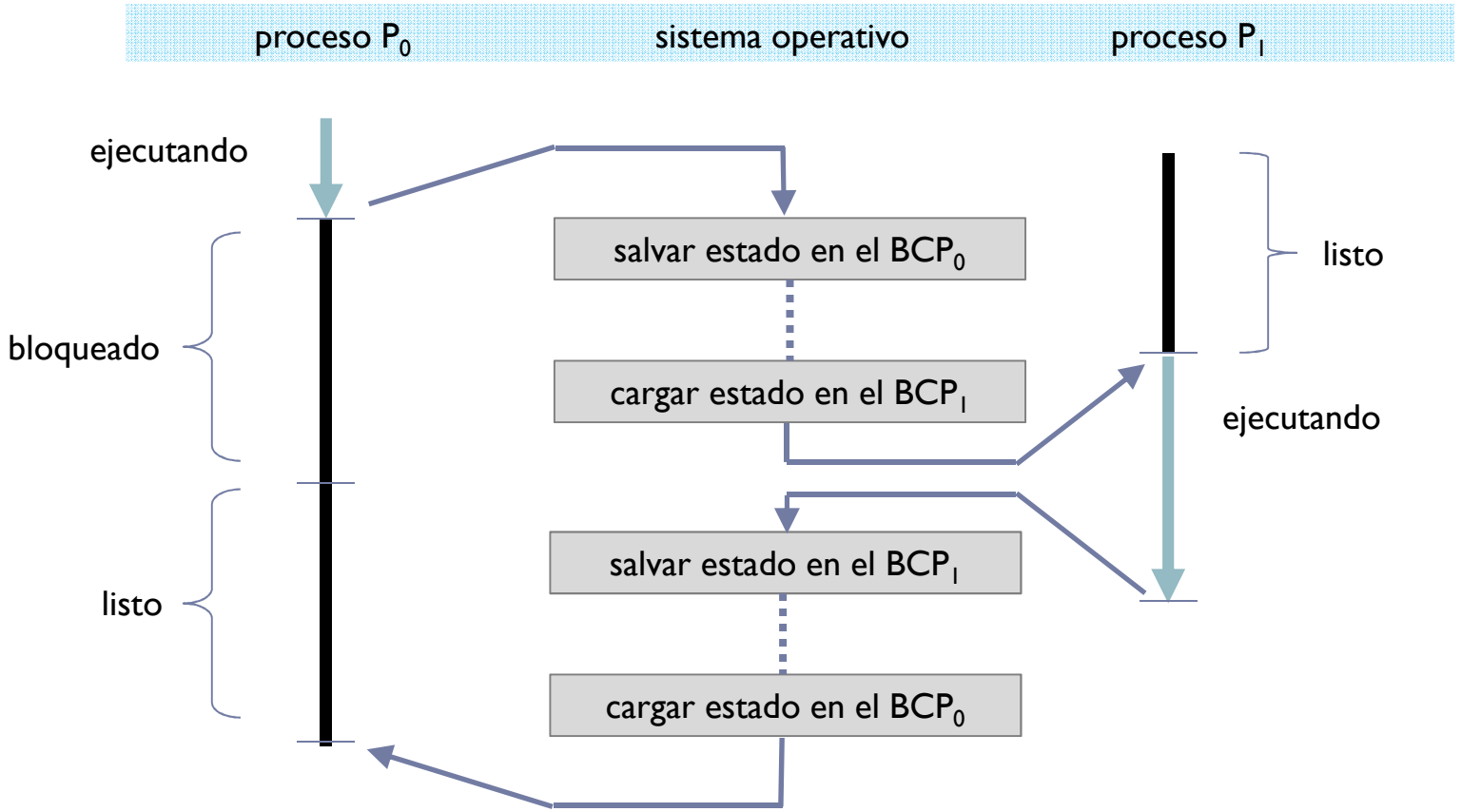
# Multiprogramación: ejemplo de ejecución



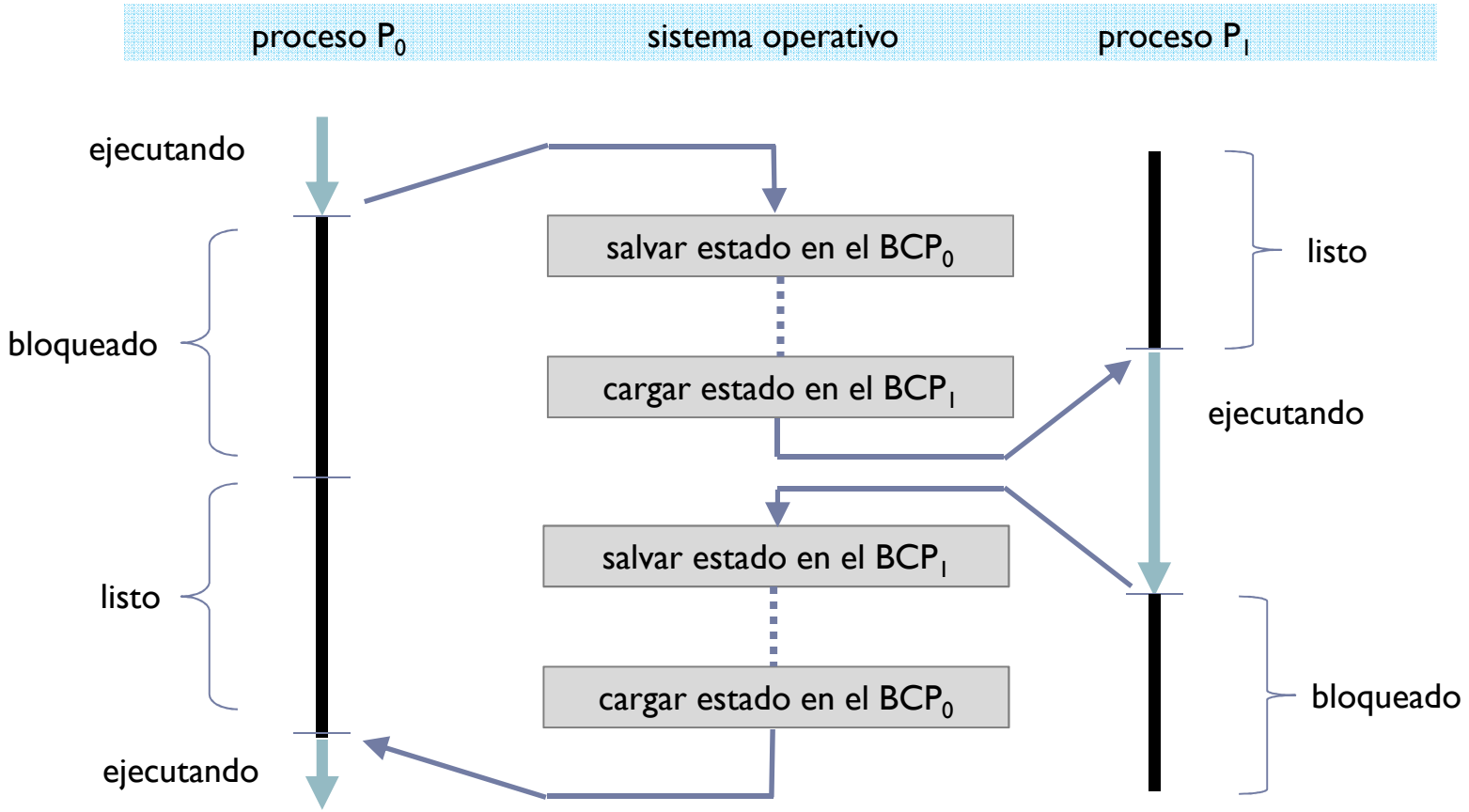
# Multiprogramación: ejemplo de ejecución



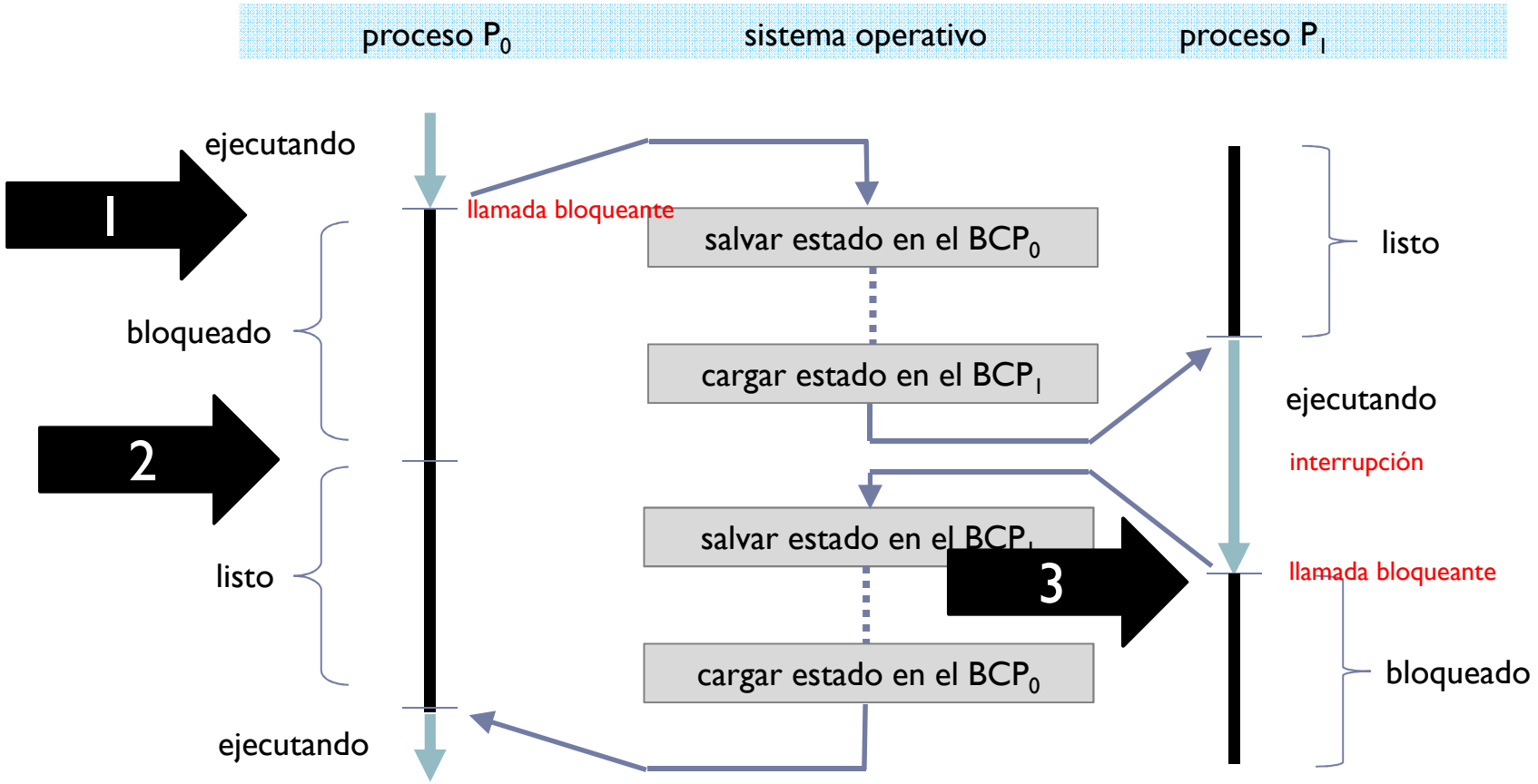
# Multiprogramación: ejemplo de ejecución



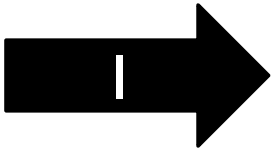
# Multiprogramación: ejemplo de ejecución



# Multiprogramación: ejemplo de ejecución



# Pseudocódigo de ejemplo (P0)

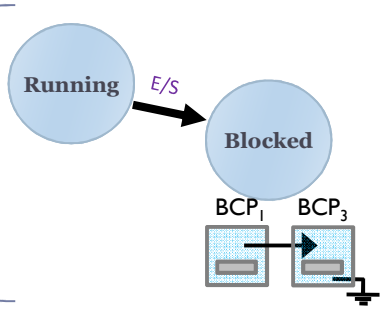


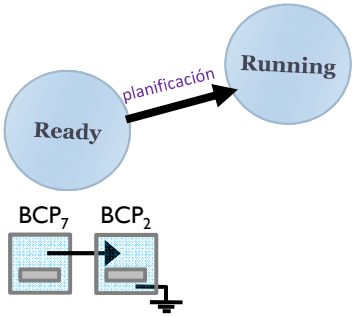
```
planificador()
• return extraer(CPU_Listo);
```

```
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) ;
```

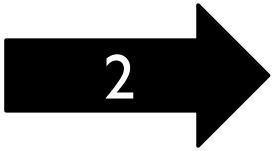
salvar estado en $BCP_0$

cargar estado en $BCP_I$



# Pseudocódigo de ejemplo (P1)

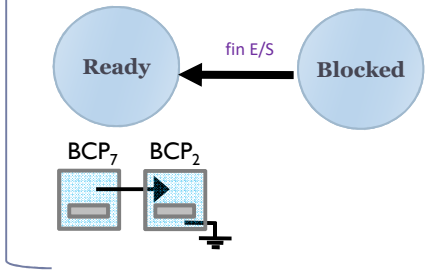



Teclado_Interrupción_Hardware ()

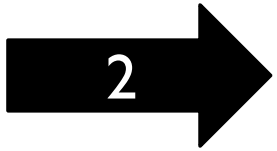
- T = in (TECLADO_HW_ID);
- proceso = insertar (T, Teclado_Teclas);
- Insertar (Teclado_interrupción_software);
- Activar_Interrupción_Software();

Teclado_Interrupción_Software ()

- proceso = primero (Teclado_Procesos);
- SI (proceso != NULL)
  - eliminar (Teclado_Procesos);
  - proceso->estado = LISTO;
  - insertar (CPU_Listos, proceso);
- return ok;



# Pseudocódigo de ejemplo (P1)

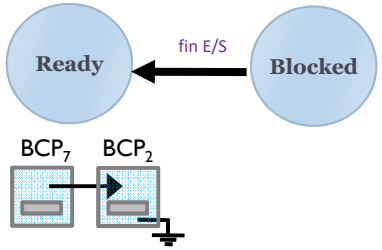



Teclado_Interrupción_Hardware ()

- T = in (TECLADO_HW_ID);
- proceso = insertar (T, Teclado_Teclas);
- Insertar (Teclado_interrupción_software);
- Activar_Interrupción_Software();

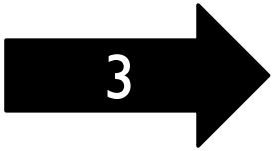
Teclado_Interrupción_Software ()

- 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()

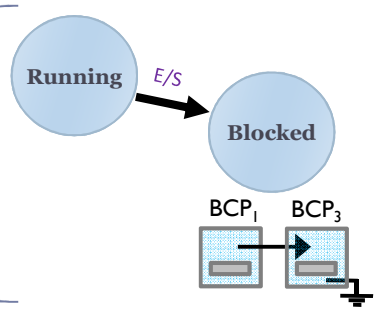
- return extraer(CPU_Listo);

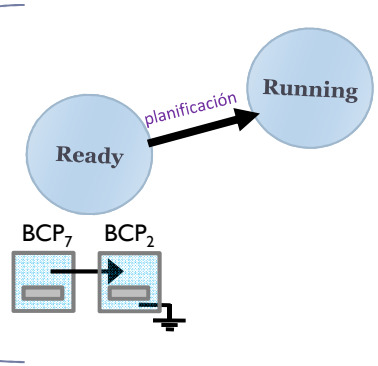
Teclado_LeerBloqueDisco()

- 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) ;

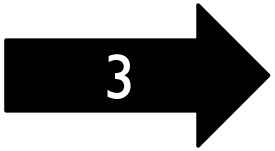
salvar estado en $BCP_I$

cargar estado en $BCP_0$



# Pseudocódigo de ejemplo (P0)

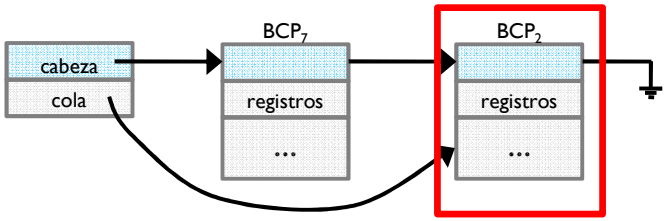



Teclado_LeerTecla()

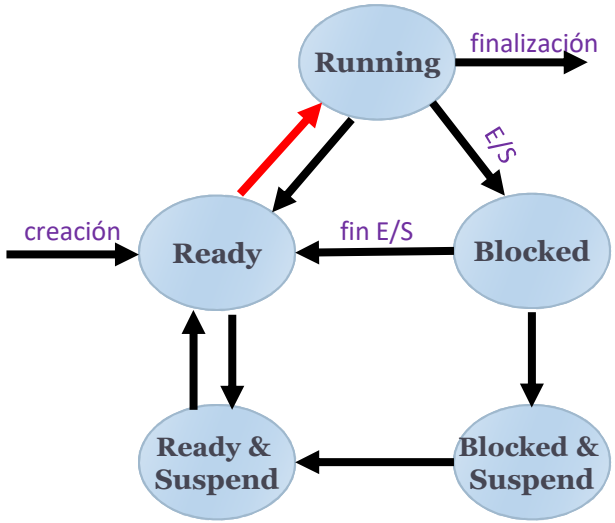
- Si (no hay tecla)
  - procesoActual->estado = BLOQUEADO;
  - Insertar(Teclado_Procesos, procesoActual);
  - proceso = procesoActual;
  - procesoActual = planificador();
  - procesoActual->estado = EJECUCION;
  - cambio_contexto( &(proceso->contexto), &(procesoActual->contexto));
- return extraer(Teclado_Teclas) ;

# Planificador y activador

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



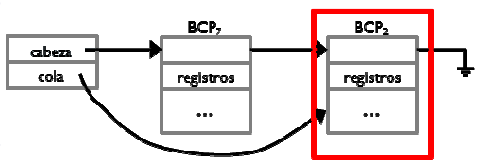

- Activador:
  Da control al proceso que el planificador ha seleccionado (cambio de contexto - restaurar)

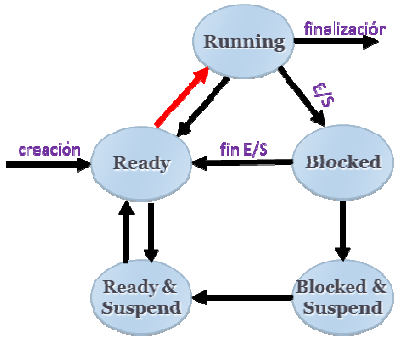


# Planificador y activador

**planificador**()

- return extraer(CPU_Listo);

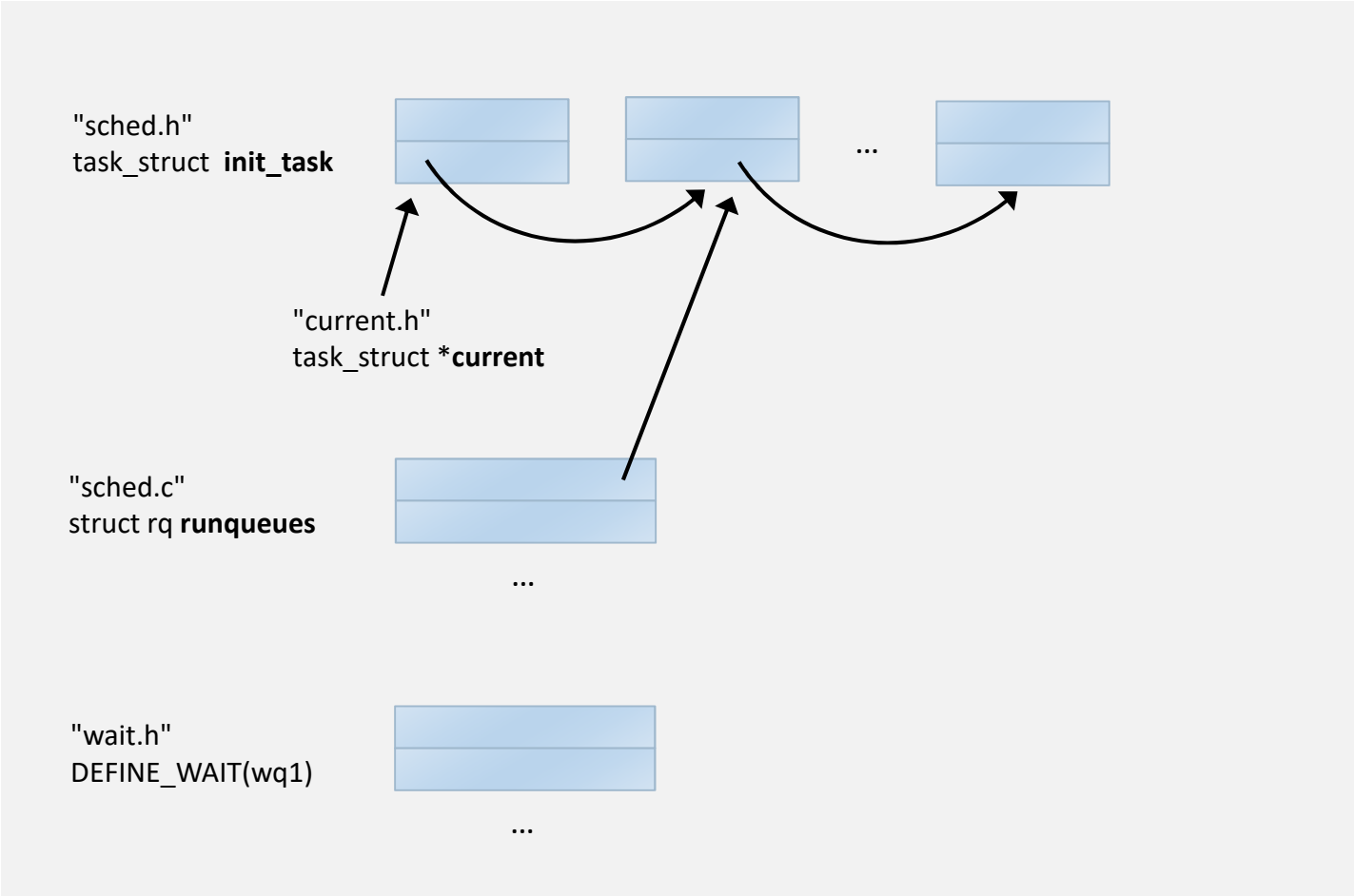



Teclado_LeerTecla()

- 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

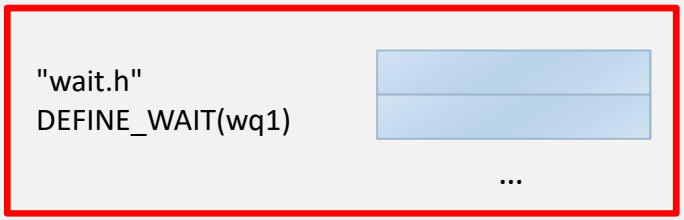
# Colas/Listas de procesos

## 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);



# 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);

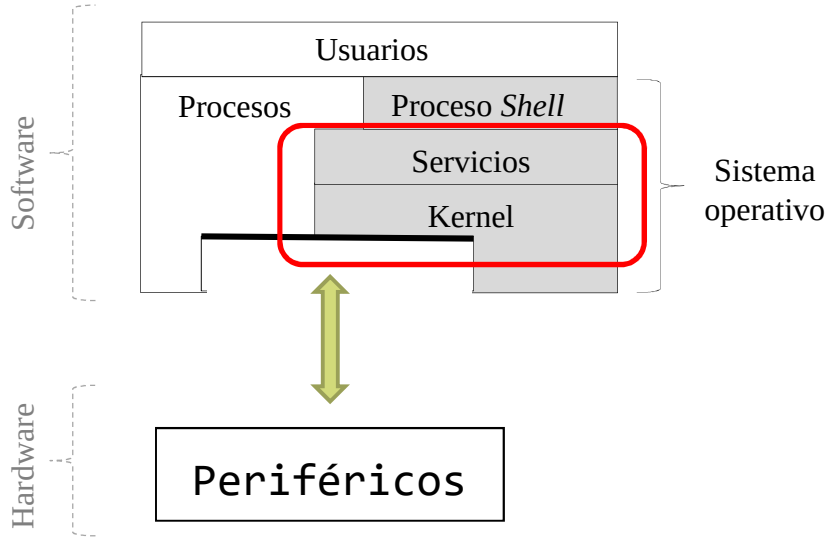
b. atomic_set(&is_blocking_mode, 0);
   wait_event_interruptible(dso_wq1,
   (atomic_read(&is_blocking_mode) == 1));

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)

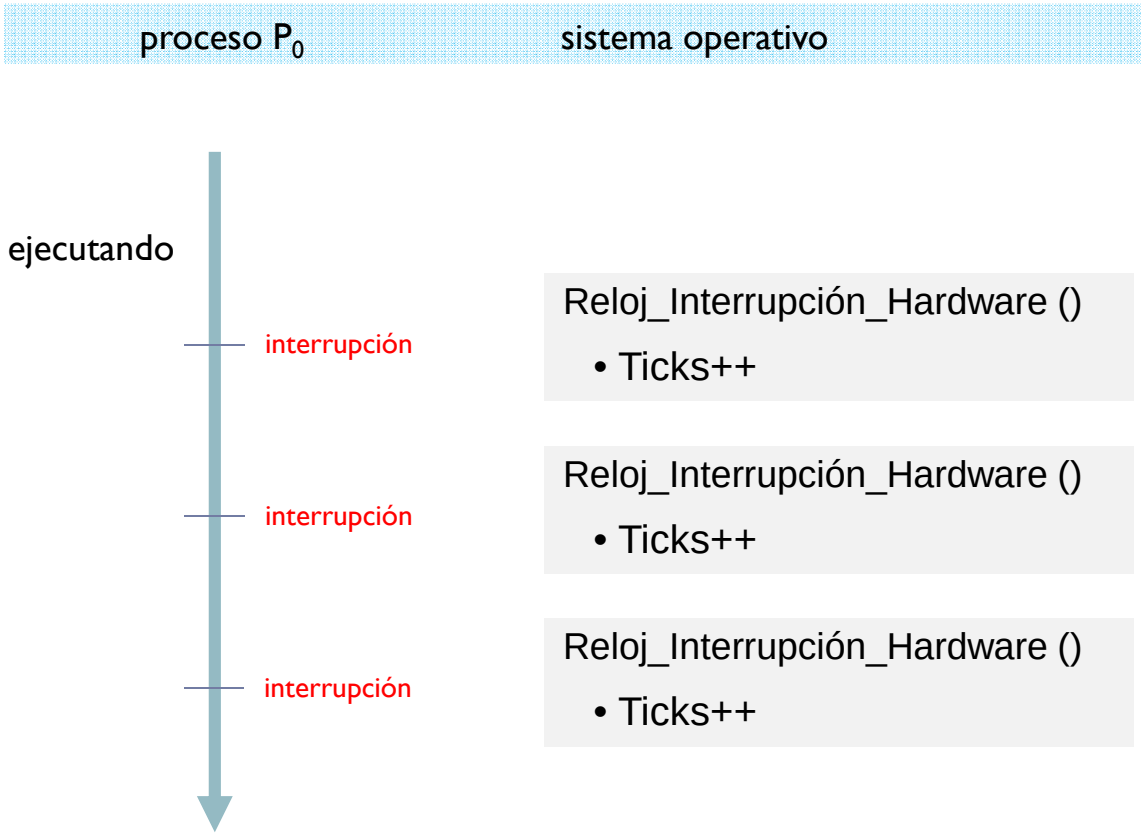
wake_up, wake_up_nr, wake_up_all, wake_up_interruptible,
wake_up_interruptible_nr, wake_up_interruptible_all,
wake_up_interruptible_sync, wake_up_locked(queue)

# 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;

# 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);
```

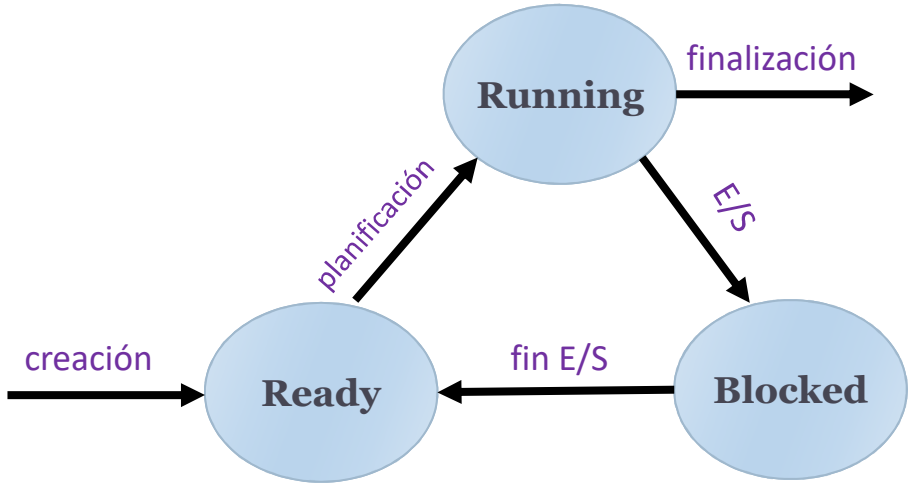
# Multitarea (datos y funciones)

| Requisitos | Información (en estructuras de datos) | Funciones (internas, servicio y API) |
|---|---|---|
| Recursos | • Zonas de memoria (código, datos y pila)<br>• Archivos abiertos<br>• Señales activas | • Diversas funciones internas<br>• Diversas funciones de servicio para memoria, ficheros, etc. |
| Multiprogramación | • Estado de ejecución<br>• Contexto: registros de CPU…<br>• Lista de procesos | • Int. hw/sw de dispositivos<br>• Planificador<br>• Crear/Destruir/Planificar proceso |
| o Protección / Compartición | • Paso de mensajes<br>  • Cola de mensajes de recepción<br>• Memoria compartida<br>  • Zonas, locks y conditions | • Envío/Recepción mensaje y gestión de la cola de mensaje<br>• API concurrencia y gestión de estructuras de datos |
| o Jerarquía de procesos | • Relación de parentesco<br>• Conjuntos de procesos relacionados<br>• Procesos de una misma sesión | • Clonar/Cambiar imagen de proceso<br>• Asociar procesos e indicar proceso representante |
| Multitarea | • Quantum restante<br>• Prioridad | • Int. hw/sw de reloj<br>• Planificador<br>• Crear/Destruir/Planificar proceso |
| Multiproceso | • Afinidad | • Int. hw/sw de reloj<br>• Planificador<br>• Crear/Destruir/Planificar proceso |

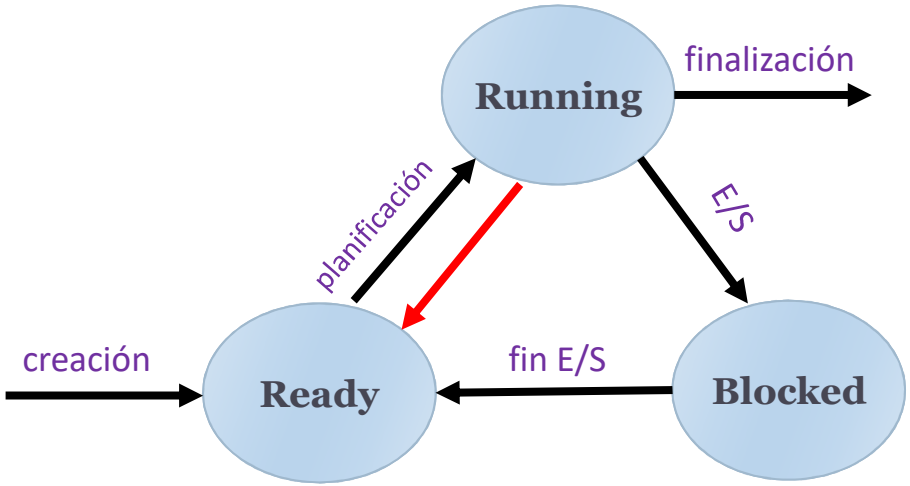
# Estados de un proceso

- Estado
- Lista/Cola
- Contexto

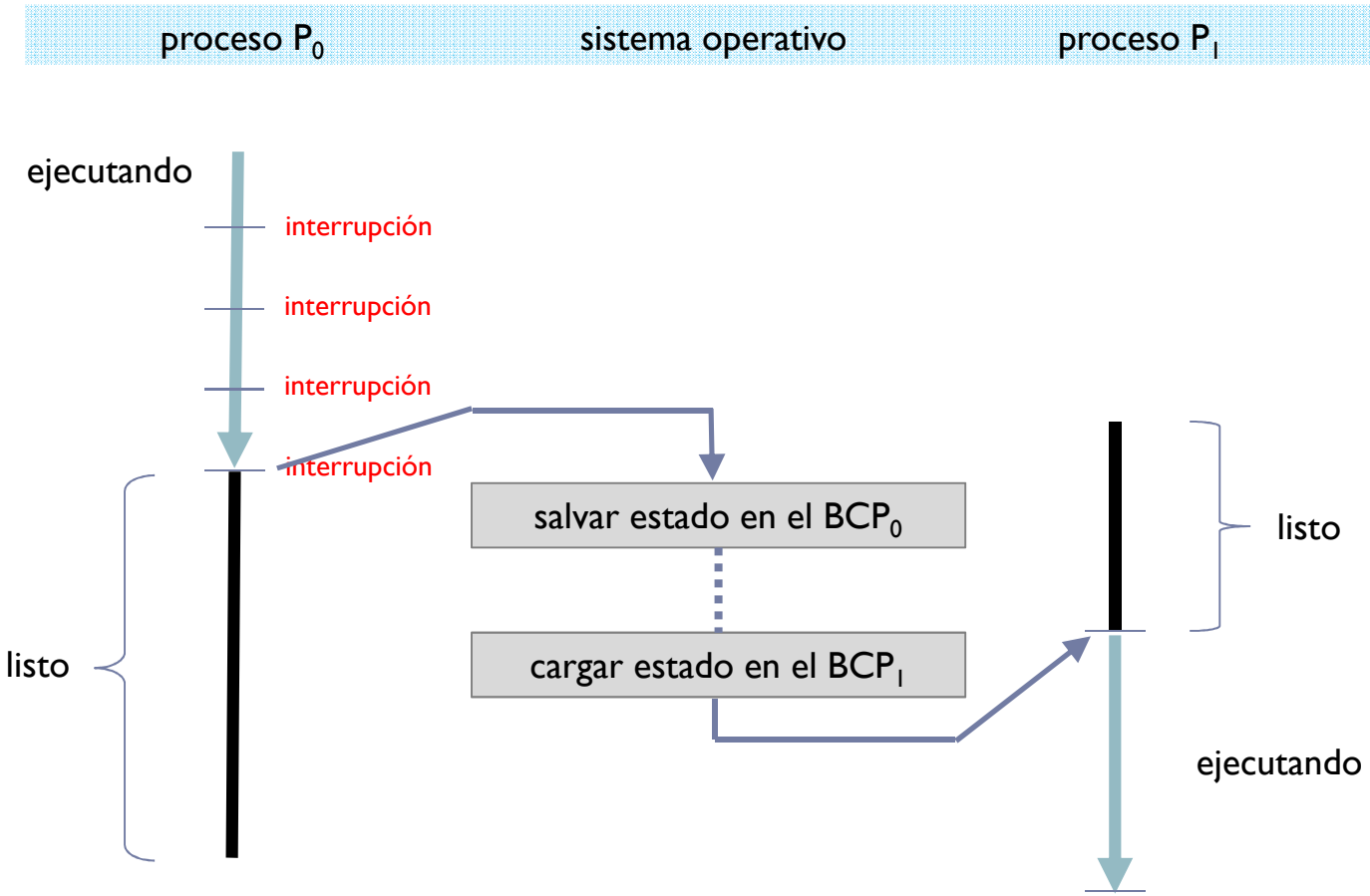
C.C.V.




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 ()

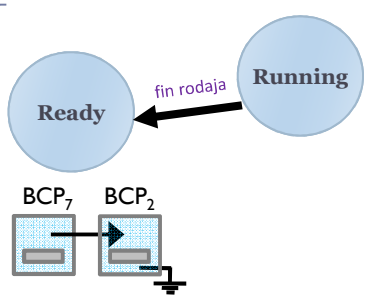
- pActual->rodaja = pActual->rodaja - 1;
- SI (pActual->rodaja == 0)
  - pActual->estado = LISTO;
  - insertar (CPU_Listos, pActual);
  - proceso = pActual;
  - pActual = planificador();
  - pActual->estado = EJECUCIÓN;
  - cambio_contexto(
    &(proceso->contexto),
    &(pActual->contexto));
- return ok;

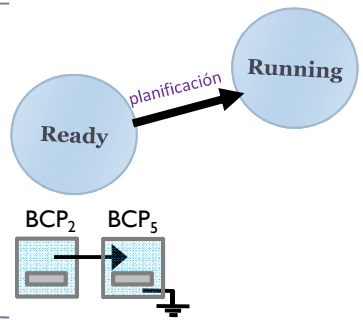
planificador()

- return extraer(CPU_Listo);

salvar estado en $BCP_0$

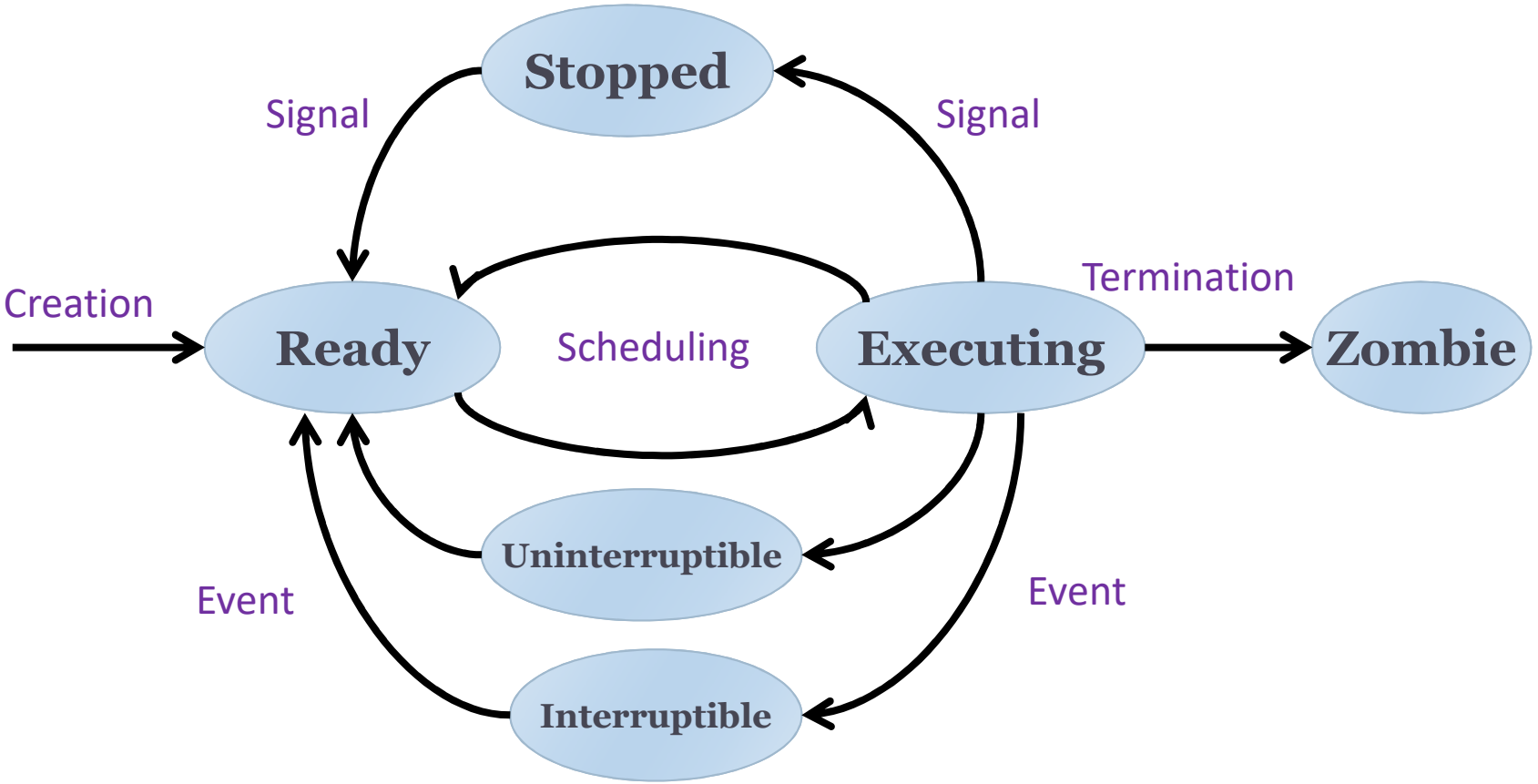
cargar estado en $BCP_I$



# Estados de un proceso

## Linux

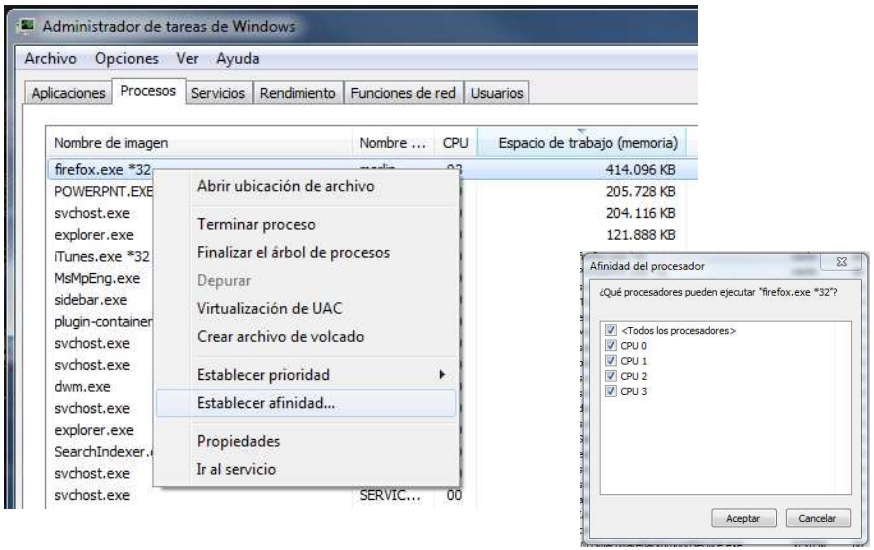


# Multiproceso

| Requisitos | Información (en estructuras de datos) | Funciones (internas, servicio y API) |
|---|---|---|
| Recursos | • Zonas de memoria (código, datos y pila)<br>• Archivos abiertos<br>• Señales activas | • Diversas funciones internas<br>• Diversas funciones de servicio para memoria, ficheros, etc. |
| Multiprogramación | • Estado de ejecución<br>• Contexto: registros de CPU…<br>• Lista de procesos | • Int. hw/sw de dispositivos<br>• Planificador<br>• Crear/Destruir/Planificar proceso |
| o Protección / Compartición | • Paso de mensajes<br>  • Cola de mensajes de recepción<br>• Memoria compartida<br>  • Zonas, locks y conditions | • Envío/Recepción mensaje y gestión de la cola de mensaje<br>• API concurrencia y gestión de estructuras de datos |
| o Jerarquía de procesos | • Relación de parentesco<br>• Conjuntos de procesos relacionados<br>• Procesos de una misma sesión | • Clonar/Cambiar imagen de proceso<br>• Asociar procesos e indicar proceso representante |
| Multitarea | • Quantum restante<br>• Prioridad | • Int. hw/sw de reloj<br>• Planificador<br>• Crear/Destruir/Planificar proceso |
| Multiproceso | • Afinidad | • Int. hw/sw de reloj<br>• Planificador<br>• Crear/Destruir/Planificar proceso |

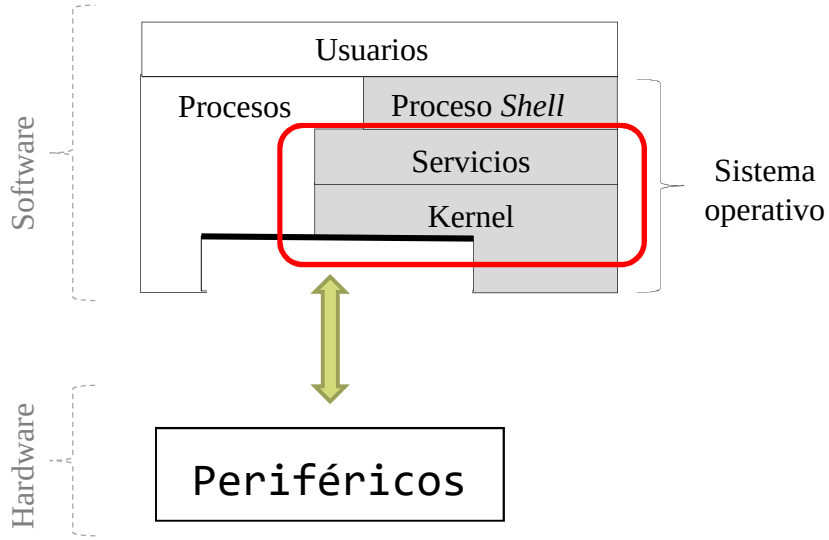
# Multiproceso

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



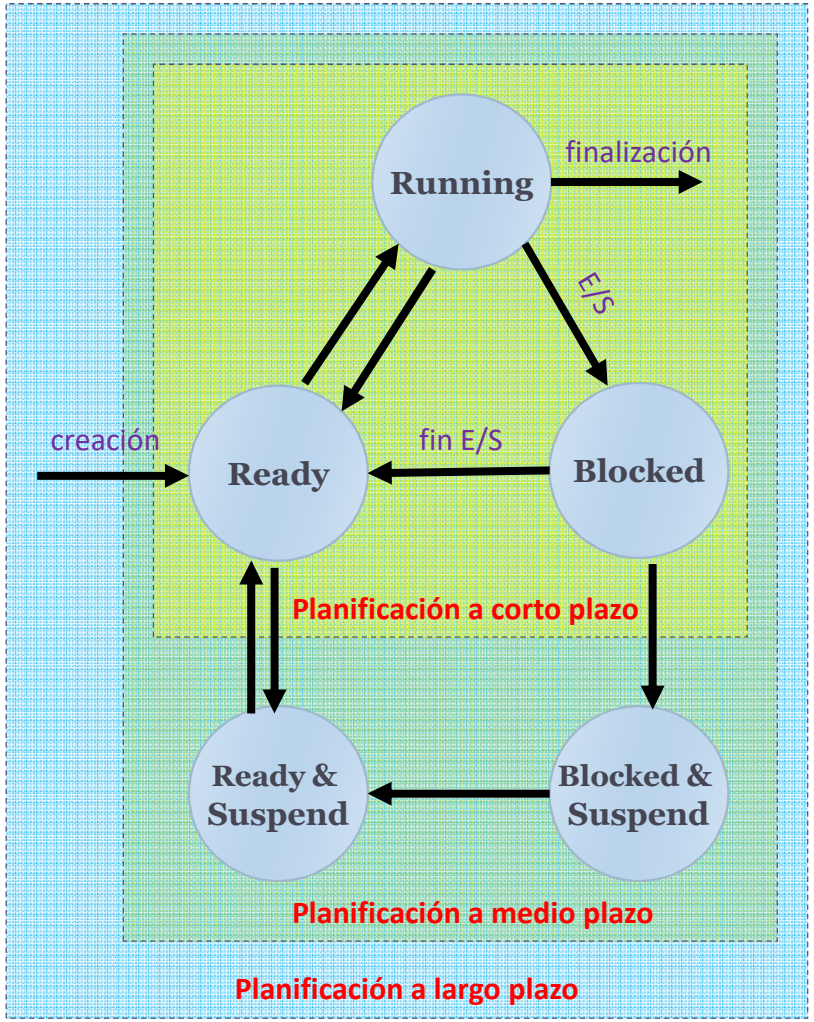

- Simetría:
  - Los procesos se ejecutan en la CPU que tienen unas capacidades específicas

# Contenidos




- Introducción
- C.C.V.
- Temporización y C.C.I.
- **Planificacion**

# 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)

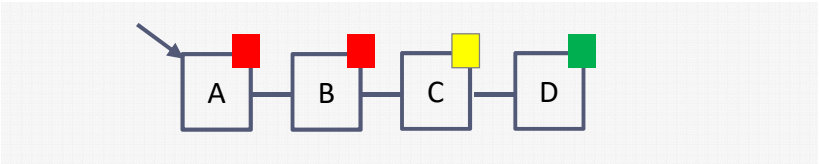
- Todos los sistemas:
  - Equitativo – ofrece a cada proceso una parte equitativa de la CPU
  - Expeditivo – cumplimiento de la política emprendida de reparto
  - Balanceado – mantener todas las partes del sistema ocupadas
- Sistemas *batch*:
  - Productividad – maximizar el número de trabajos por hora
  - 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 expectaciones 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)

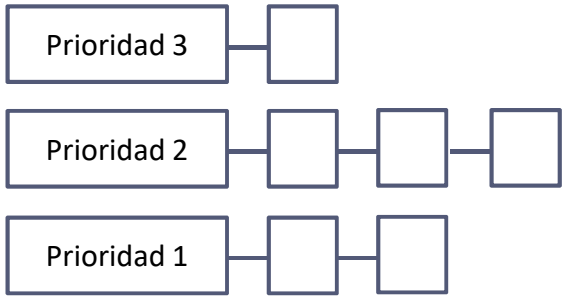

- Grande:
  - menos cambios de contexto
  - Peor reactividad

  - 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

http://www.phoronix.com/scan.php?page=article&item=linux_2637_video&num=1

# 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

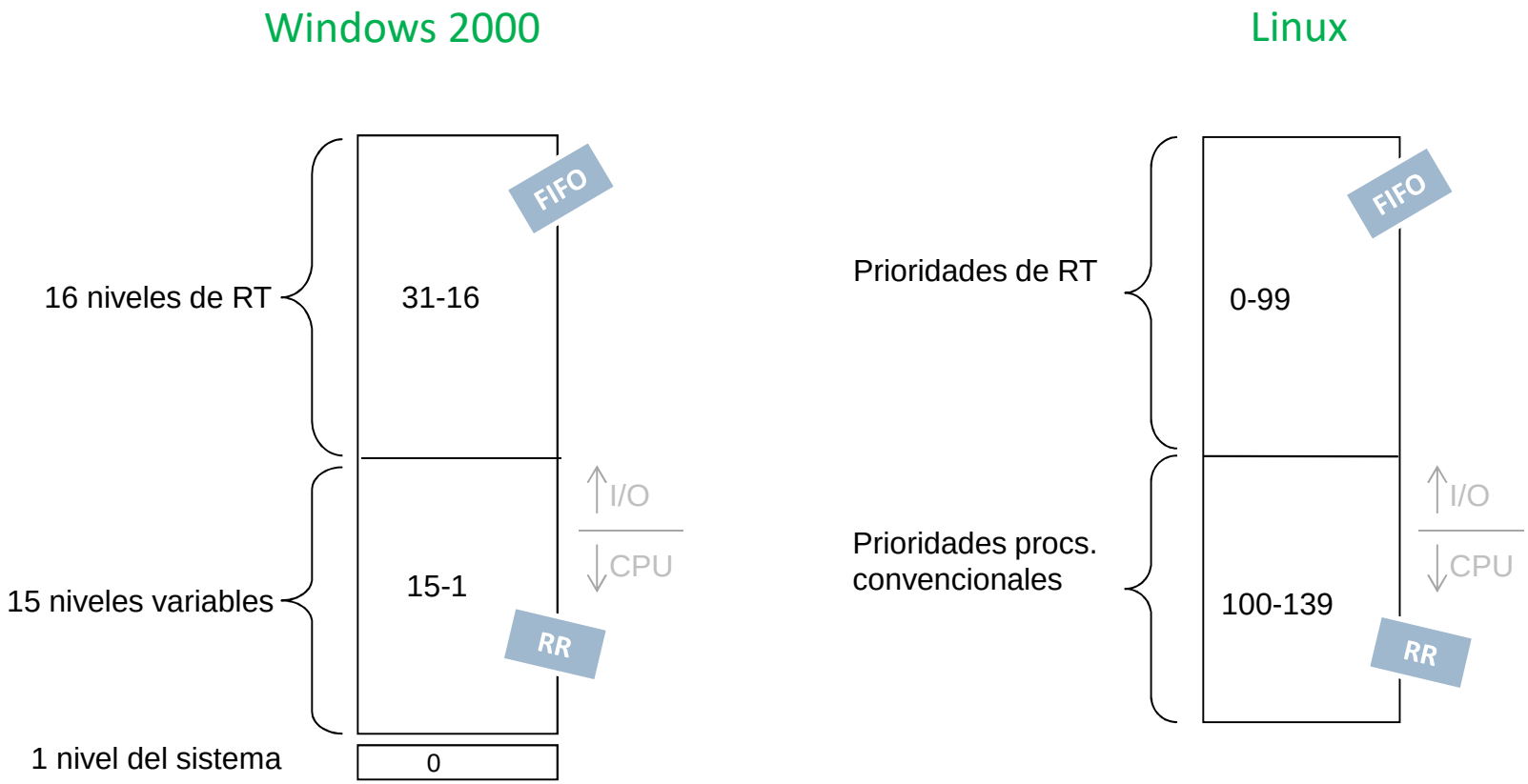
# Planificación de procesos
## **principales algoritmos** de planificación (3/3)

- 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*.

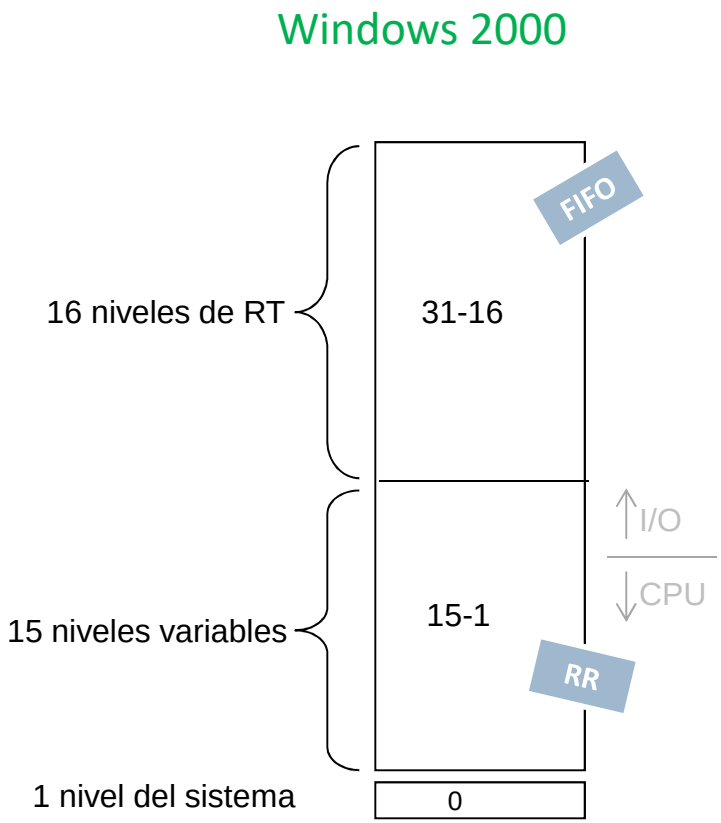
# Política vs mecanismo

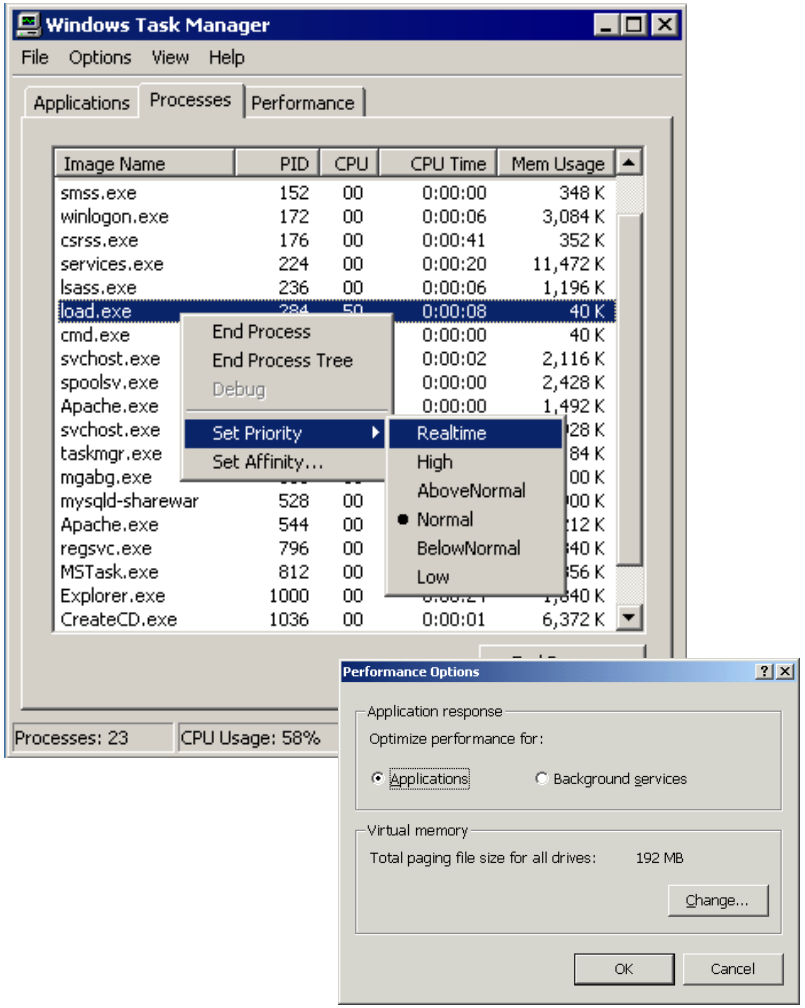
- Separación de lo qué se puede hacer
  de cómo se puede hacer
  - Normalmente, un proceso conoce cuál es el hilo más prioritario, el que más E/S necesitará, etc.
- Uso de algoritmos de planificación parametrizados
  - Mecanismo en el kernel
- Parámetros rellenados por los procesos de usuarios
  - Política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

# Planificación multipolítica
## Windows 2000 y Linux



# Planificación multipolítica
## Windows 2000



Grupo ARCOS
Universidad Carlos III de Madrid

# Lección 3b

## procesos, periférico, *drivers* y servicios ampliados

Diseño de Sistemas Operativos
Grado en Ingeniería Informática