

Ejercicios

Procesos e hilos, y planificación

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

Ejercicio

enunciado (1/2)

Una máquina monoprocesador tiene instalado un sistema operativo con núcleo no expulsivo que usa un algoritmo de planificación de procesos basado en *round-robin*.

Se quiere implementar dos llamadas al sistema (con un pequeño parecido a wait y signal) que permitan a los procesos disponer de una primitiva de sincronización: la “puerta”.

Una “puerta” tiene el siguiente funcionamiento:

- ▶ Con la llamada al sistema **esperarEnPuerta** bloquea al proceso que la invoca.
- ▶ La llamada al sistema **abrirPuerta** desbloquea a todos los procesos que desde la última llamada a *esperarEnPuerta* (o desde el arranque del ordenador la primera vez) y hasta el momento de llamar a *abrirPuerta* han estado esperando en la puerta.

Ejercicio

enunciado (2/2)

Se pide:

- ▶ Indique las estructuras de datos necesarias (incluyendo los estados de los procesos) para implementar las nuevas primitivas de sincronización. Tenga en cuenta que el sistema solo debe tener una puerta en todo el sistema.
- ▶ Implementar en pseudocódigo las llamadas al sistema descritas:
 - a) Llamada al sistema ***void esperarEnPuerta (void)***: Bloquea al proceso llamante de la forma anteriormente descrita.
 - b) Llamada al sistema ***void abrirPuerta (void)***: Desbloquea a todos los procesos que estén bloqueados de la forma descrita anteriormente.

Ejercicio

solución

1. Planteamiento inicial

1. Estado inicial del sistema
2. Estudio de qué hay que modificar

2. Responder a las preguntas

3. Revisar las respuestas

Ejercicio

solución

1. Planteamiento inicial

1. Estado inicial del sistema
2. Estudio de qué hay que modificar

2. Responder a las preguntas

3. Revisar las respuestas

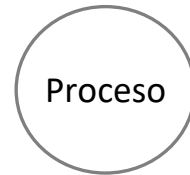
Ejercicio solución

Realicemos un diagrama con el estado inicial del sistema, con los elementos más relevantes para el problema

U

K

Ejercicio solución



En espacio de usuario (U) tenemos los procesos que hacen llamadas al sistema a través de `system_lib` o provocan excepciones, lo que provoca la ejecución del núcleo (K)

system_lib

U

K

Ejercicio solución

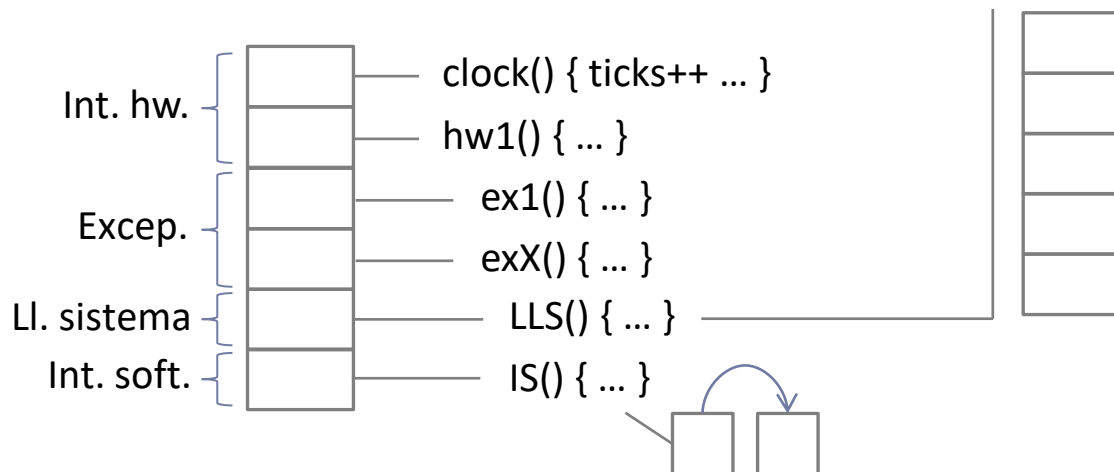
En el tema 2 se introducía el funcionamiento interno del núcleo del sistema operativo: interrupciones software, llamadas al sistema, excepciones e interrupciones hardware

Proceso

system_lib

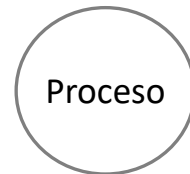
U

K



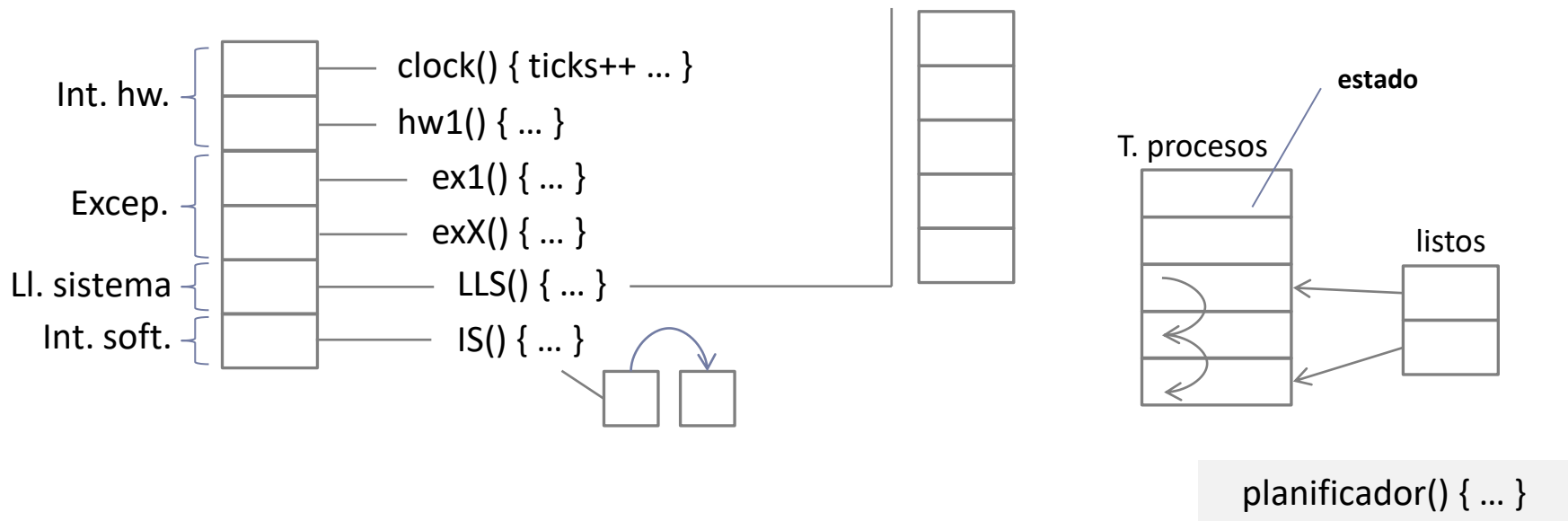
Ejercicio solución

En el tema 3 se introducía las estructuras y funciones internas para la gestión de procesos, como la tabla de procesos, la cola de listos para ejecutar, el planificador, etc.



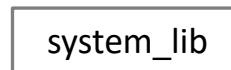
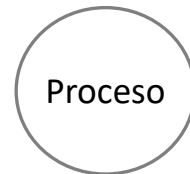
system_lib

U
K

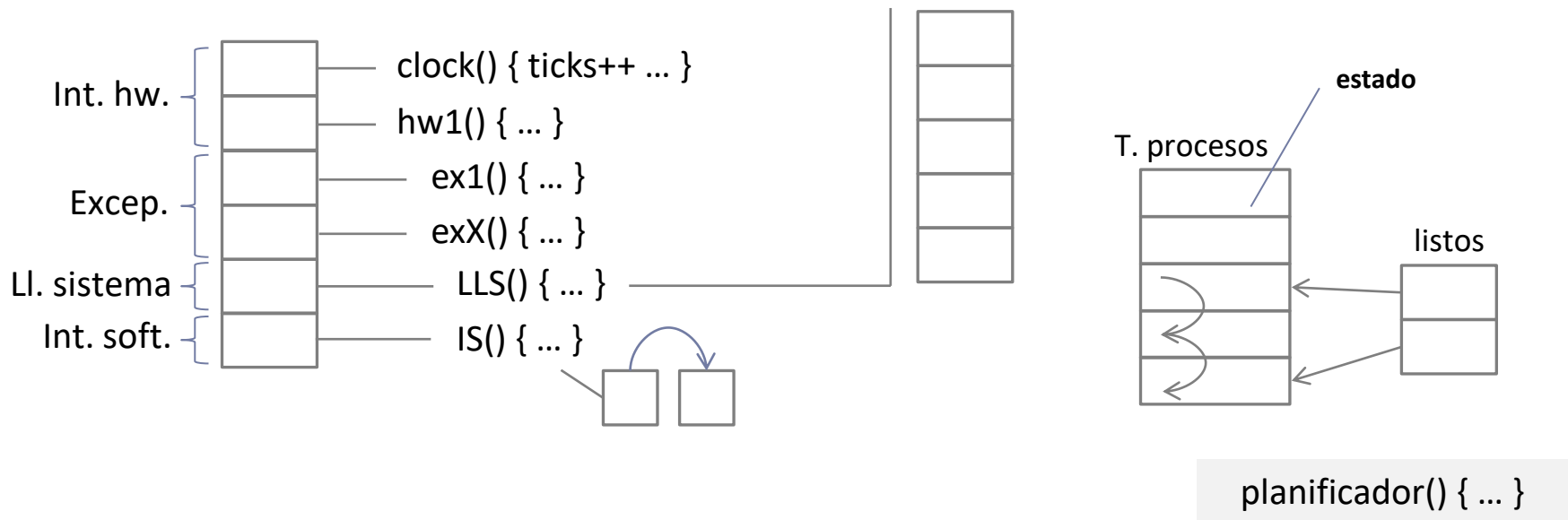


Ejercicio solución

Tenemos en este diagrama el estado inicial del sistema, con los elementos más relevantes para el problema



U
K



Ejercicio

solución

1. Planteamiento inicial

1. Estado inicial del sistema

2. Estudio de qué hay que modificar

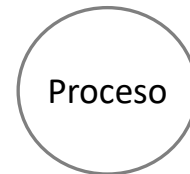
2. Responder a las preguntas

3. Revisar las respuestas

Ejercicio solución

Se parte del sistema inicial, al que hay que añadir dos primitivas:

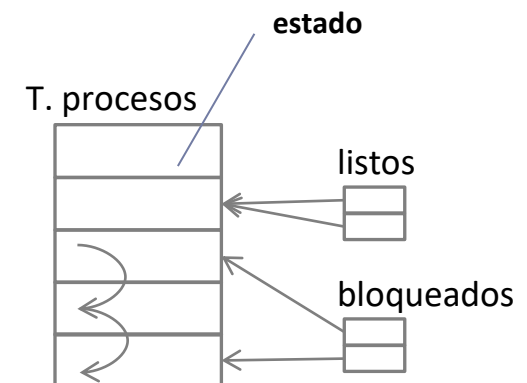
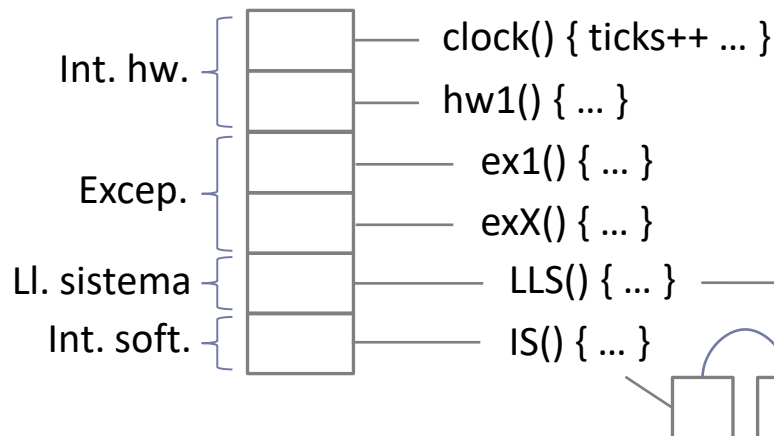
- `esperarEnPuerta()`
- `abrirPuerta()`



system_lib

U

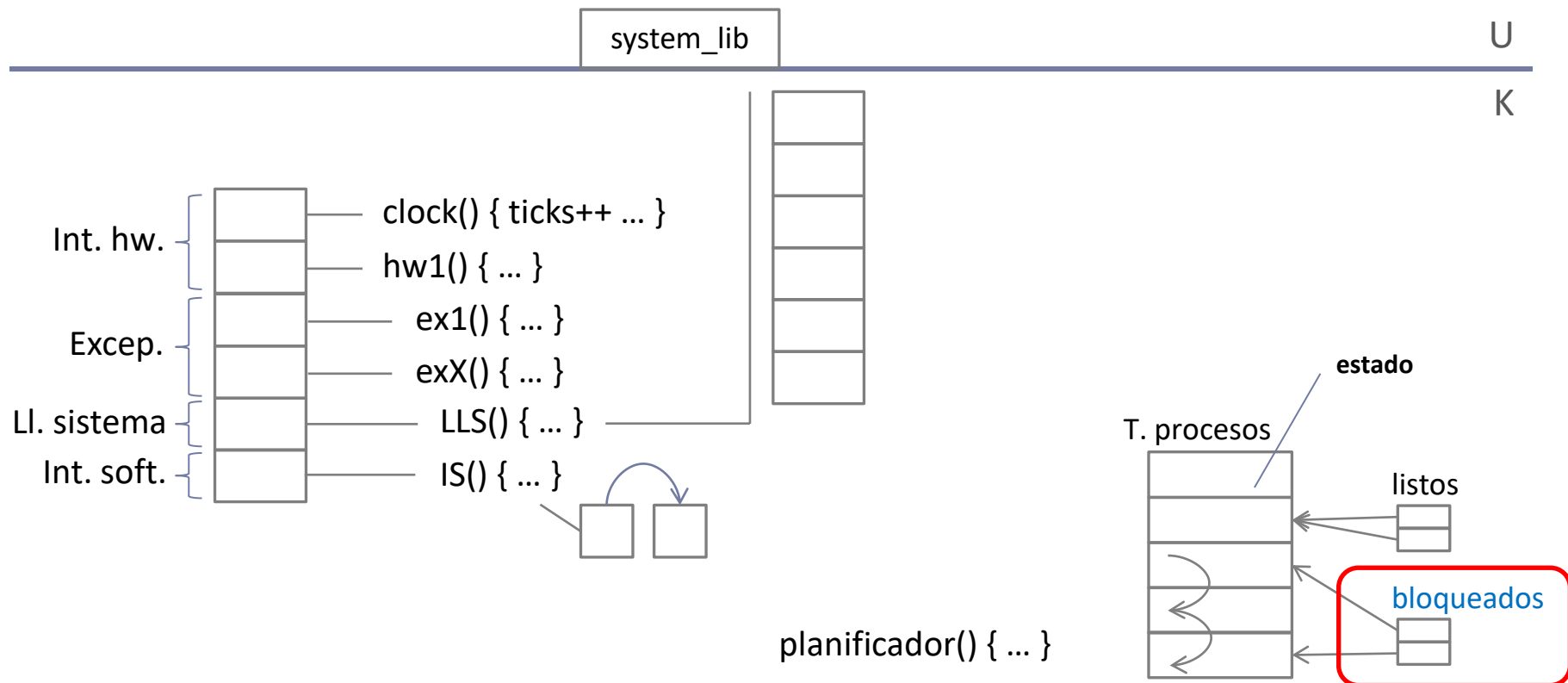
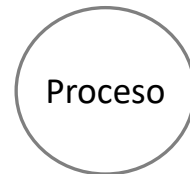
K



Ejercicio solución

Las estructuras de datos para el bloqueo de un proceso son:

- Estado en el BCP: incluir bloqueado.
- Lista de bloqueados por espera en puerta.



Ejercicio solución

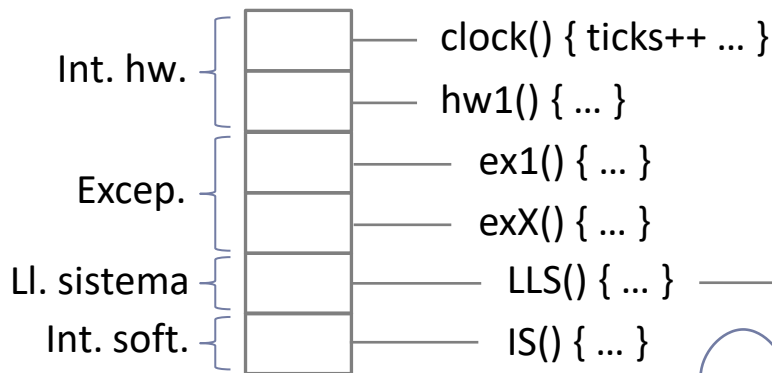
Las parte en el núcleo de las llamadas al sistema bloquean un proceso y los despiertan.

Proceso

system_lib

U

K



abrirPuerta

esperarEnPuerta

```
p=Primero(bloqueados);
while (p != NULL)
{
  p->estado=LISTO
  Eliminar(bloqueados, p)
  Insertar(listos, p)
  p=Primero(bloqueados)
}
Insertar(bloqueados, pA);
pA->estado=BLOQUEADO;
pAA=pA;
pA=planificador();
pA->estado=EJECUTANDO;
Activador(pAA, pA);
```

planificador() { ... }

estado

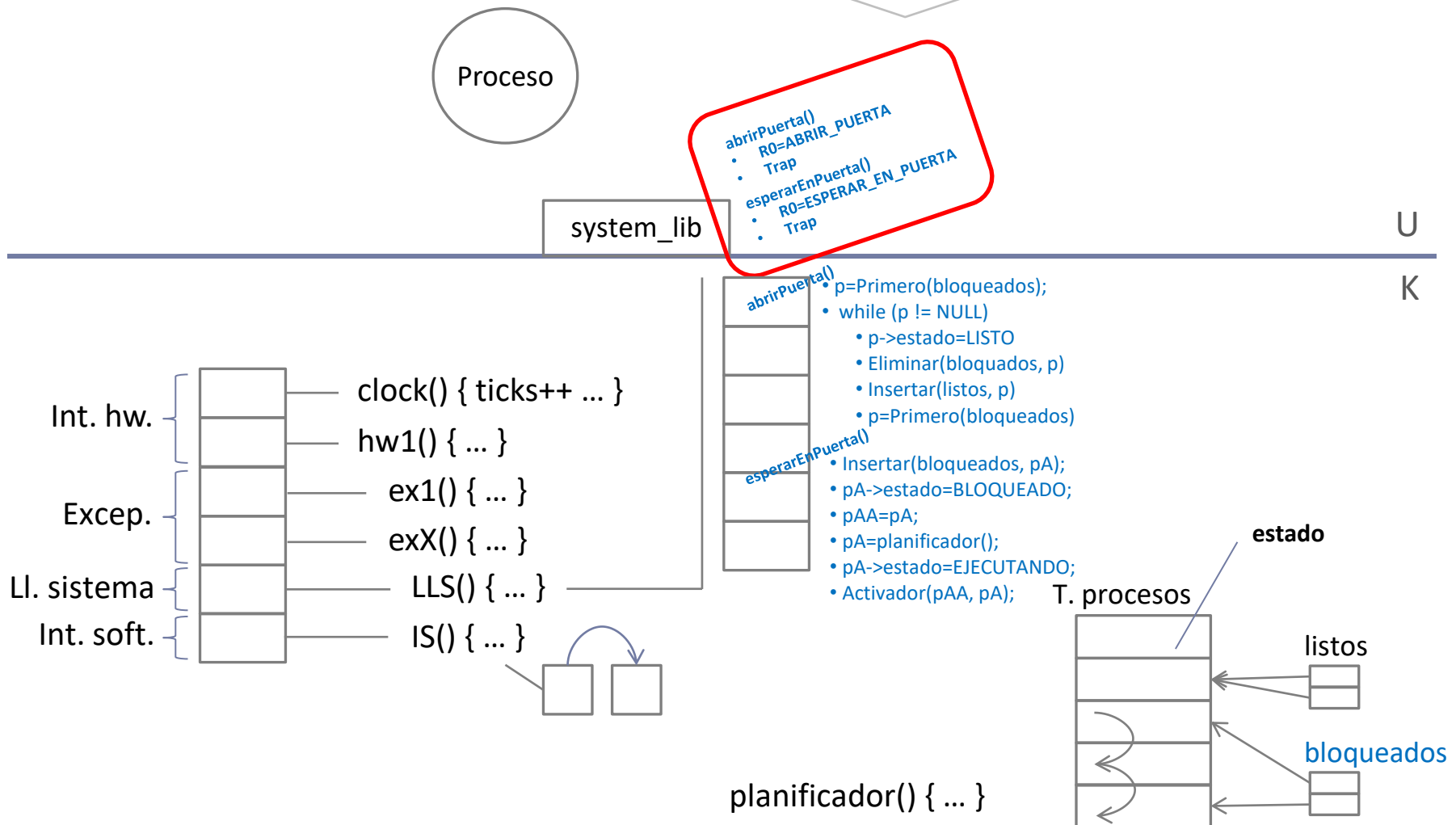
T. procesos

listos

bloqueados

Ejercicio solución

Las parte en el espacio de usuario invocan la contraparte del núcleo.



Ejercicio solución

Repasamos el enunciado para comprobar que todos y cada uno de los requisitos del planteamiento se cumplen.

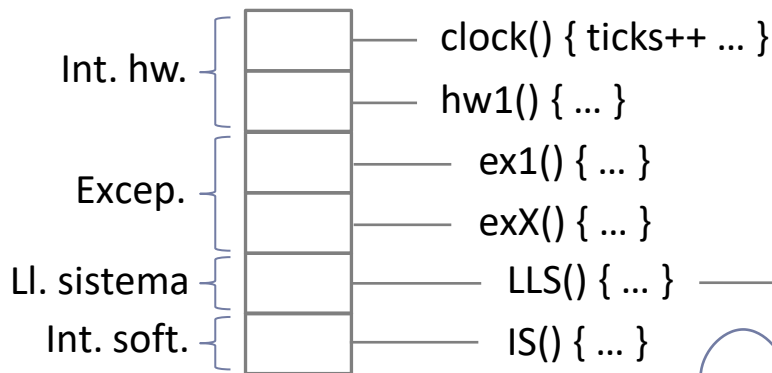
Proceso

system_lib

abrirPuerta()
• R0=ABRIR_PUERTA
• Trap
esperarEnPuerta()
• R0=ESPERAR_EN_PUERTA
• Trap

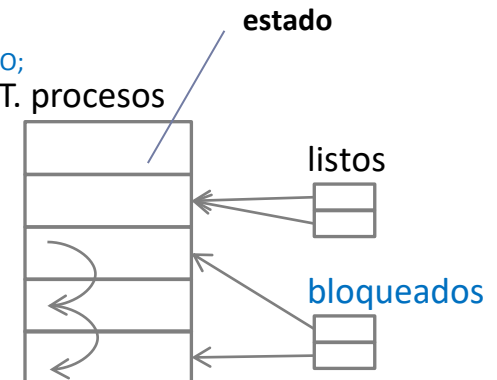
U

K



abrirPuerta()
• p=Primero(bloqueados);
• while (p != NULL)
• p->estado=LISTO
• Eliminar(bloqueados, p)
• Insertar(listos, p)
• p=Primero(bloqueados)
esperarEnPuerta()
• Insertar(bloqueados, pA);
• pA->estado=BLOQUEADO;
• pAA=pA;
• pA=planificador();
• pA->estado=EJECUTANDO;
• Activador(pAA, pA);

planificador() { ... }



Ejercicio

solución

1. Planteamiento inicial

1. Estado inicial del sistema
2. Estudio de qué hay que modificar

2. Responder a las preguntas

3. Revisar las respuestas

Ejercicio

solución

Mirando el planteamiento realizado, contestamos a las preguntas

Estructuras de datos:

- En el BCP:
 - Estado : LISTO, BLOQUEADO <- añadir constante
- En las listas de procesos:
 - Lista de procesos bloqueados por 'puerta'

Ejercicio

solución

Funciones:

Llamada_al_Sistema_krn_esperarEnPuerta (void):

- ▶ Insertar(puerta_bloqueados, ProcesoActual);
 - ▶ ProcesoActual->estado = BLOQUEADO;
 - ▶ ProcesoAnterior = ProcesoActual;
 - ▶ ProcesoActual = planificador();
 - ▶ ProcesoActual->estado = EJECUTANDO;
 - ▶ Activador(ProcesoAnterior, ProcesoActual); /* Aquí se bloquea el proceso */
- /* Aquí vuelve desde la llamada al Activador de otro proceso */

Ejercicio

solución

Funciones:

Llamada_al_Sistema_krn_abrirPuerta (void):

- ▶ Proc = ObtenerPrimerProceso (puerta_bloqueados);
- ▶ Mientras (Proc != NULO)
 - ▶ Proc->estado = LISTO;
 - /* recordar el orden: Eliminar+Insertar, no al revés */
 - ▶ Eliminar(puerta_bloqueados, ProcesoActual);
 - ▶ Insertar(Lista_LISTOS, ProcesoActual);
 - ▶ Proc = ObtenerPrimerProceso (puerta_bloqueados).

Ejercicio

solución

Funciones:

Llamada_al_Sistema_usr_esperarEnPuerta (void):

- ▶ R0 = ESPERAR_EN_PUERTA;
- ▶ TRAP();

Llamada_al_Sistema_usr_abrirPuerta (void):

- ▶ R0 = ABRIR_PUERTA;
- ▶ TRAP();

Ejercicio

solución

1. Planteamiento inicial

1. Estado inicial del sistema
2. Estudio de qué hay que modificar

2. Responder a las preguntas





3. Revisar las respuestas

Fallos típicos



- 1) Contestar a la primera pregunta de un apartado únicamente.
- 2) Contestar a otra pregunta de la pedida.
- 3) Contestar a más de lo que se pide:
 - 1) Si está mal la parte extra, puede que se evalúe ...

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>Ejercicio 1</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 expansiva 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>DISEÑO DE SISTEMAS OPERATIVOS</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>Introducción a los mecanismos de cambio de contexto en procesos con Linux/Ubuntu</p>	<p>DISEÑO DE SISTEMAS OPERATIVOS</p> <p>GRADO EN INGENIERÍA INFORMÁTICA</p>  <p>UNIVERSIDAD CARLOS III DE MADRID</p> <p>Planificación de procesos</p> <p>David del Río Astorga</p>

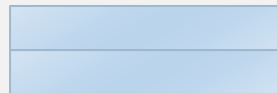
Colas/Listas de procesos

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)



...

Mecanismo

a. `atomic_t is_blocking_mode = ATOMIC_INIT(0);`
`DECLARE_WAIT_QUEUE_HEAD(dso_wq1);`

Estructuras de datos

- En el BCP:
 - Estado : LISTO, BLOQUEADO <- añadir constante
- En las listas de procesos:
 - Lista de procesos bloqueados por 'puerta'

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

Funciones.

Llamada_al_Sistema_krn_esperarEnPuerta (void):

- ▶ Insertar(puerta_bloqueados, ProcesoActual);
 - ▶ ProcesoActual->estado = BLOQUEADO;
 - ▶ ProcesoAnterior = ProcesoActual;
 - ▶ ProcesoActual = planificador();
 - ▶ ProcesoActual->estado = EJECUTANDO;
 - ▶ Activador(ProcesoAnterior, ProcesoActual); /* Aquí se bloquea el proceso */
- /* Aquí vuelve desde la llamada al Activador de otro proceso */

```
c. atomic_set(&is_blocking_mode, 1);  
wake_up_interruptible(&dso_wql);
```

Funciones.

Llamada_al_Sistema_krn_abrirPuerta (void):

- ▶ Proc = ObtenerPrimerProceso (puerta_bloqueados);
- ▶ Mientras (Proc != NULO)
 - ▶ Proc->estado = LISTO;
 - /* recordar el orden: Eliminar+Insertar, no al revés */
 - ▶ Eliminar(puerta_bloqueados, ProcesoActual);
 - ▶ Insertar(Lista_LISTOS, ProcesoActual);
 - ▶ Proc = ObtenerPrimerProceso (puerta_bloqueados).

Mecanismo

Funciones

Llamadas

▶ R0 = E

▶ TRAP()

```
static ssize_t dso_read ( struct file *file, char __user *buf, size_t count, loff_t *ppos )
{ ... }

static ssize_t dso_write ( struct file *file, const char __user *buf, size_t count, loff_t *ppos )
{ ... }

static const struct file_operations proc_ops = {
    .owner = THIS_MODULE,
    .read = dso_read,
    .write = dso_write,
};

static int dso_init ( void ) { proc_create("dso-barrier", 0, NULL, &proc_ops); return 0; }
static void dso_exit ( void ) { remove_proc_entry("dso-barrier", NULL); }

module_init(dso_init);
module_exit(dso_exit);
```

Llamada_al_Sistema_usr_abrirPuerta (void):

▶ R0 = ABRIR_PUERTA;

▶ TRAP();

Plan B: no usar nuevas llamadas al sistema, usar las del sistema de fichero y asociar a un fichero del directorio /proc

Ejercicios

Procesos e hilos, y planificación

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