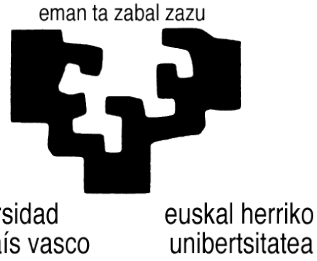
UPV / EHU



Programación Concurrente en Linux

Acceso coordinado a recursos compartidos

Alberto Lafuente, Dep. KAT/ATC de la UPV/EHU, bajo Licencia Creative Commons

UPV / EHU

Contenido

1. Recursos compartidos
2. Mecanismos de sincronización: espera por ocupado
3. Mecanismos de sincronización: espera por bloqueado

OCW OpenCourseWare

UPV / EHU

1. Recursos compartidos

OCW OpenCourseWare

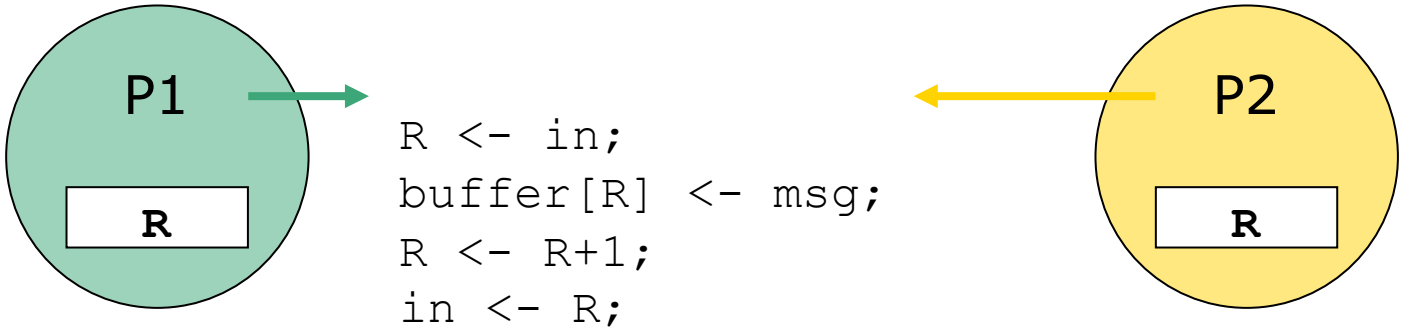
Recursos compartidos

- En un sistema las tareas (procesos o hilos) *compiten* por el uso a *recursos compartidos*.
- El acceso al recurso se realiza mediante la ejecución de un trozo de código.
- Ejemplo:
  - Recurso compartido: buffer fifo en memoria
  - Código máquina que ejecuta un hilo productor:

```
…
R <- in;
buffer[R] <- msg;
R <- R+1;
in <- R;
…
```

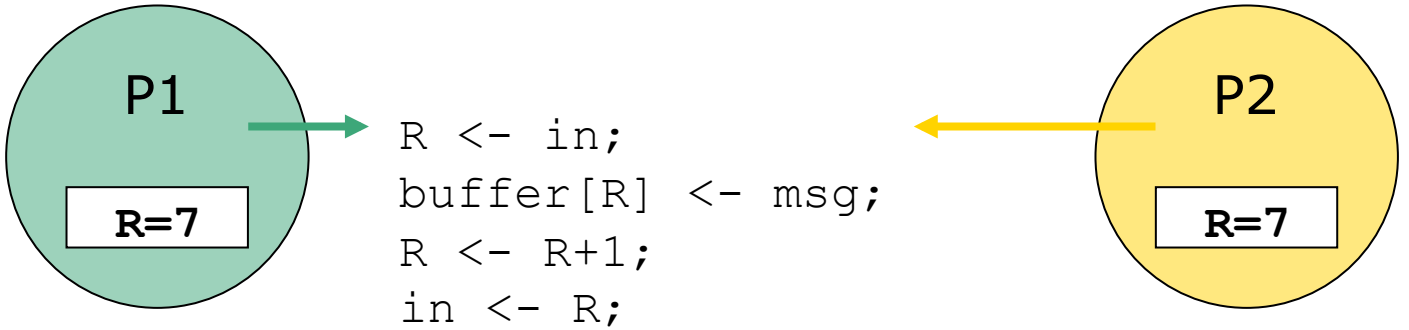
Recursos compartidos
Acceso concurrente

- Ejemplo de ejecución:
  - Dos hilos, P1 y P2, ejecutan concurrentemente el código de acceso al buffer compartido
    - Inicialmente, `in=7`



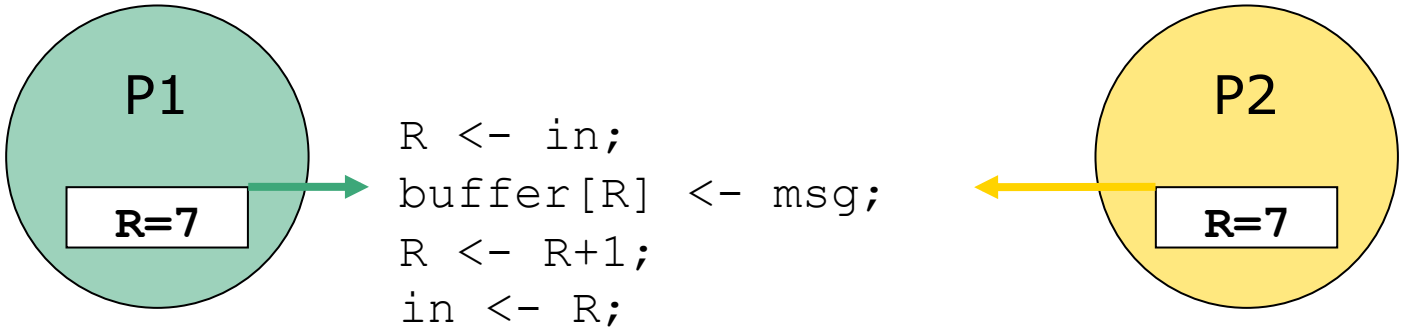
Recursos compartidos
Acceso concurrente

- Ejemplo de ejecución:
  - Dos hilos, P1 y P2, ejecutan concurrentemente el código de acceso al buffer compartido
    - Inicialmente, `in=7`



Recursos compartidos
Condición de carrera

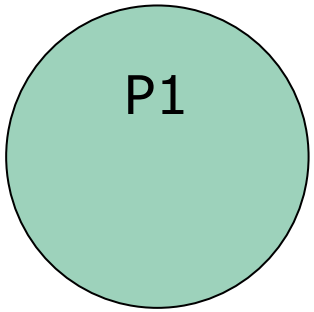
- *Condición de carrera*
  - Ambos hilos almacenan *msg* en la misma posición del buffer



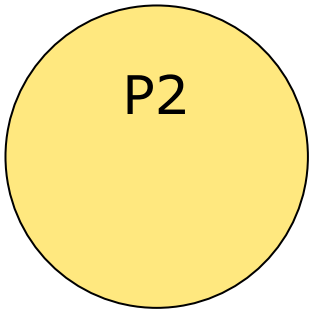
Secciones críticas

- En el acceso concurrente a recursos compartidos, las condiciones de carrera conducen a comportamientos incorrectos.
- ¿Cómo evitar condiciones de carrera?
  - El trozo de código que controla el acceso a recursos compartidos es una *sección crítica* de código.
  - Hay que proporcionar *acceso exclusivo* a las secciones críticas.

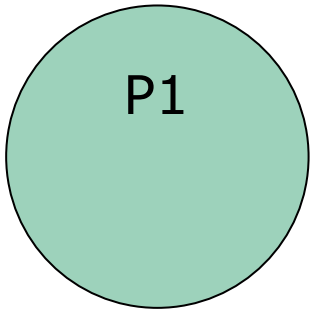
Acceso exclusivo a secciones críticas de código



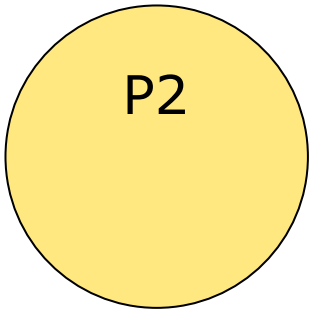

```
R <- in;
buffer[R] <- msg;
R <- R+1;
in <- R;
```



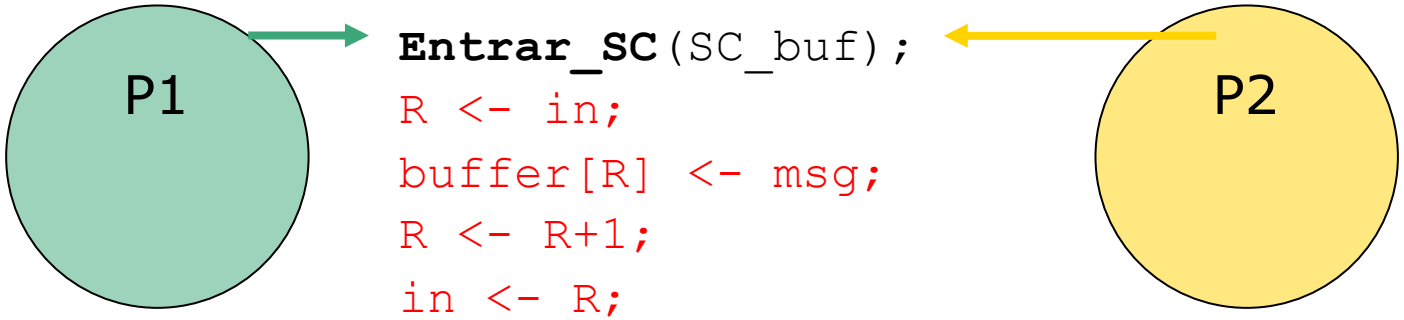
Acceso exclusivo a secciones críticas de código



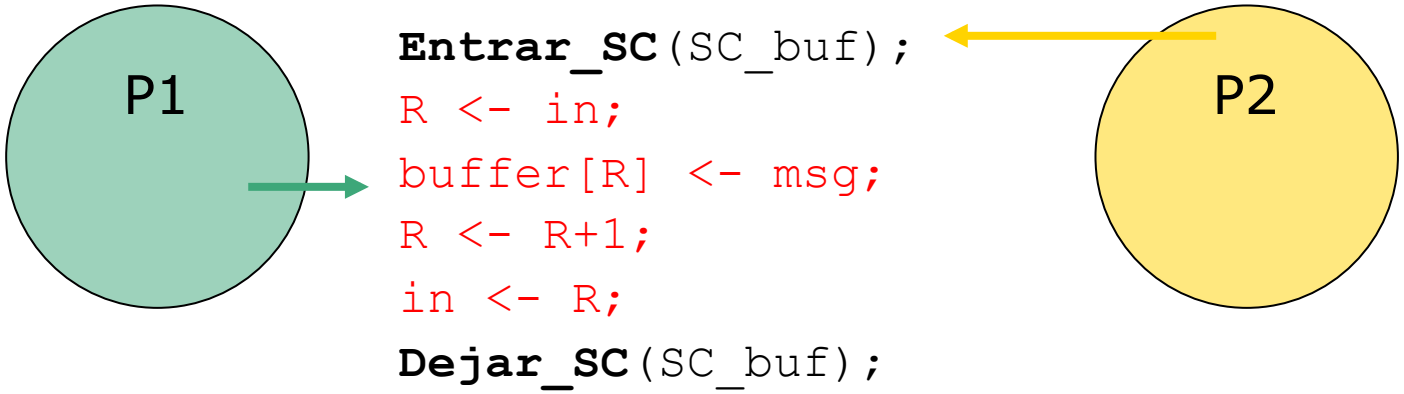

```
Entrar_SC(SC_buf);
R <- in;
buffer[R] <- msg;
R <- R+1;
in <- R;
Dejar_SC(SC_buf);
```



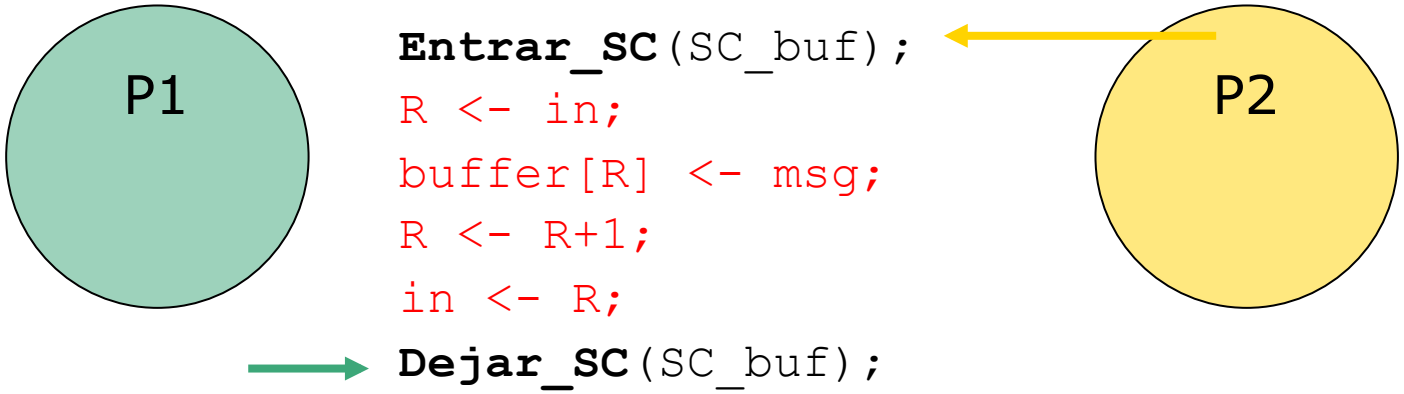
Acceso exclusivo a secciones críticas de código



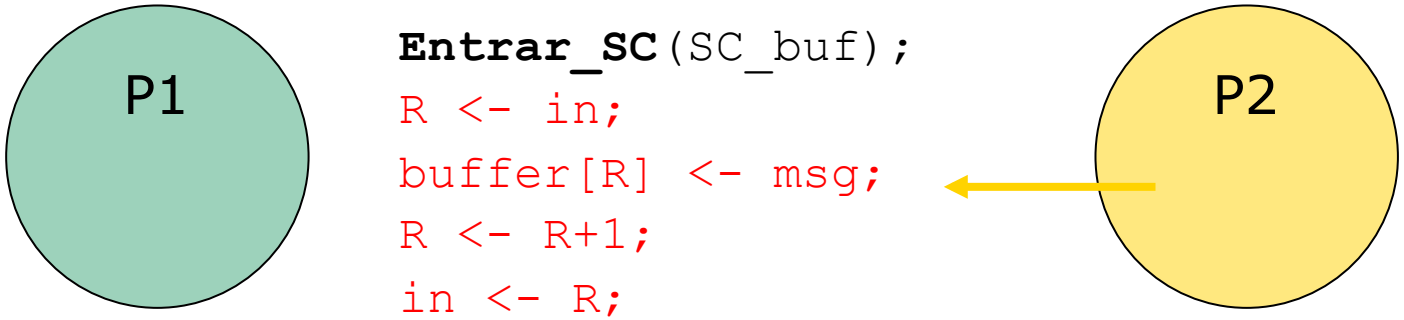
Acceso exclusivo a secciones críticas de código



Acceso exclusivo a secciones críticas de código



Acceso exclusivo a secciones críticas de código



Secciones críticas
Modelo de sección crítica

- Protocolo genérico de acceso a una sección crítica:

  Entrar_SC(la_SC) /* *Solicitud de ejecutar la_SC* */
  /* *código de la_SC* */
  Dejar_SC(la_SC) /* *Otro proceso puede ejecutar la_SC* */

- Un proceso que va a ejecutar la SC:
  1. Ejecuta Entrar_SC(). Si la SC está ocupada, el proceso espera.
  2. Ejecuta la SC.
  3. Ejecuta Dejar_SC(), permitiendo que entre uno de los procesos en espera.

Secciones críticas Propiedades

- El acceso a una sección crítica debe poseer las siguientes propiedades:
  1. *Exclusión mutua*. No puede haber más de un proceso simultáneamente en la SC.
  2. *No interbloqueo*. Ningún proceso fuera de la SC puede impedir que otro entre a la SC.
  3. *No inanición*. Un proceso no puede esperar por tiempo indefinido para entrar a la SC.
  4. *Independencia del hardware*. No se pueden hacer suposiciones acerca del número de procesadores o de la velocidad relativa de los procesos.
- Suposición: las instrucciones del Lenguaje Máquina son atómicas y se ejecutan secuencialmente

Secciones críticas
Mecanismos de sincronización

- ¿Cómo espera un proceso para acceder a una SC ocupada (paso 1 del protocolo)?
- Mecanismos de *sincronización* (o *coordinación*):
  a. El proceso ejecuta una espera activa (*espera por ocupado*).
     - En sistemas monoprocesador suele utilizarse inhibición de interrupciones.
  b. El proceso se bloquea (*espera por bloqueado*).

UPV / EHU

2. Mecanismos de sincronización:
espera por ocupado

OCW OpenCourseWare

Mecanismos de sincronización
Espera por ocupado

- En monoprocesadores: *inhibición de interrupciones*

```
s= inhibir()        /* Entrar_SC() */
   /* Sección crítica */
desinhibir(s)       /* Dejar_SC() */
```

UPV / EHU

Mecanismos de sincronización
Espera por ocupado

- En multiprocesadores: *cerrojos de espera activa*

```
lock(cerrojo_A)          /* Entrar_SC() */
   /* Sección crítica */
unlock(cerrojo_A)        /* Dejar_SC() */
```

- Implementación de lock()
  - Por Sw: muy restrictivo (algoritmo de Decker) o ineficiente (Lamport).
  - Instrucciones del Lenguaje Máquina que proporcionan consulta y modificación atómica sobre el cerrojo (tipo *test_and_set*).

Mecanismos de sincronización
Espera por ocupado

- Los mecanismos de espera por ocupado dependen del soporte (no cumplen la condición 4):
  - La inhibición de interrupciones sólo es válida para monoprocesadores.
  - La espera activa en monoprocesadores depende de la política de planificación:
    - Con prioridades estáticas y expulsión, si una SC está ocupada por un proceso de prioridad baja, un proceso de prioridad alta que llegue a la SC ejecutaría espera activa indefinidamente e impediría al de prioridad baja liberarla (fenómeno de *inversión de prioridad*).

UPV / EHU

3. Mecanismos de sincronización: espera por bloqueado

OCW OpenCourseWare

Mecanismos de sincronización
Espera por bloqueado

UPV / EHU

- El proceso que accede a una SC ocupada se bloquea.
  - Implica un cambio de contexto, luego sólo es rentable si la SC es de largo plazo.
  - El proceso bloqueado no consume CPU.
- El proceso que abandona la SC provoca que algún proceso en espera se desbloquee.

Mecanismos de sincronización
Espera por bloqueado: semáforos

- Un semáforo es una abstracción que implementa espera por bloqueado proporcionando orden FIFO.
- Un semáforo lleva asociadas
  - Una cuenta
  - Una cola
- Primitivas sobre semáforos:
  - **bajar**(sem)
    - Si la cuenta de sem es positiva, se decrementa y el proceso continúa.
    - Si la cuenta vale 0, el proceso se bloquea en la cola de sem.
  - **subir**(sem)
    - Si la cola de sem no está vacía, se desbloquea al primer proceso de la cola
    - Si la cola está vacía, se incrementa la cuenta de sem.
  - Primitiva para la inicialización de la cuenta.

Mecanismos de sincronización
Espera por bloqueado: semáforos

- La inicialización del semáforo determina su utilización.
- Para controlar el acceso a una sección crítica se usa un semáforo inicializado a 1:

```
bajar(sem);
  /* Sección crítica */
subir(sem);
```

- Si el semáforo se inicializa a *n*, permite gestionar el uso simultáneo de *n* recursos.
- Si se inicializa a 0, puede utilizarse como evento sobre el que esperar una notificación.

Mecanismos de sincronización
Ejemplo: gestión de un buffer circular

```
struct semaf huecos, items;
tipo_cerrojo mutex;
ini_semaforo(huecos, N);
ini_semaforo(items, 0);

consumidor()
{
    struct elemento elem;
    while (1) {
        bajar(items);
        lock(mutex);
        retirar(&elem);
        unlock(mutex);
        subir(huecos);
        consumir_elemento(elem);
    }
}
```

Mecanismos de sincronización
Ejemplo (cont)

UPV / EHU

```
productor()
{
    struct elemento elem;
    while (1) {
            producir_elemento(&elem);
            bajar(huecos);
            lock(mutex);
            insertar(elem);
            unlock(mutex);
            subir(items);
    }
}
```

OCW OpenCourseWare