Concurrencia

Stallings 5ta ed. Capítulo 5.
Silberschatz 7ma ed. Capítulo 6.

Repaso clase anterior: Hilos

- Monohilo / Multihilo
- PCB / TCB.
- Estructuras de la imagen de un proceso en multihilo.
- Ventajas de los hilos.
- Estados / Planificación de Hilos.
- ULT / KLT.
- Arquitecturas de Kernel.

Introducción

- Multiprogramación / Multiprocesamiento.
- Competir por recursos.
- Compartir recursos.

Introducción

- Condición de carrera.
- Sección crítica.

Interacción entre procesos

- Comunicación entre procesos.
- Competencia de los procesos por los recursos.
- Cooperación de los procesos vía compartición.
- Cooperación de los procesos vía comunicación.

Sección Crítica

- Requisitos que deben cumplirse:
  - Exclusión Mutua.
  - Progreso.
  - Espera limitada.
  - Velocidad Relativa.

Sección Crítica

- Posibles Soluciones:
  - De Software.
  - De Hardware.
  - Provistas por el SO: Semáforos.
  - Provistas por los lenguajes de programación: Monitores.

ENTRADA

SECCIÓN CRÍTICA

SALIDA

Soluciones de Software

PRIMER INTENTO

int turno = 0;

Proceso 0:	Proceso 1:
while(turno!=0) /*nada*/;	while(turno!=1) /*nada*/;
../* SC */..	**../* SC */..**
turno = 1;	turno = 0;

Exclusión Mutua: SI

Progreso: NO (Alternancia)

Espera activa: SI

SEGUNDO INTENTO

int estado[] = {falso , falso};

Proceso 0:	Proceso 1:

while(estado[1])
/*nada*/;
estado[0] = true;
	while(estado[0])
	/*nada*/;
	estado[1] = true;

../* SC */..	**../* SC */..**
estado[0] = false;	estado[1] = false;

Exclusión Mutua: NO

Progreso: SI

Espera activa: SI

Soluciones de Software

SEGUNDO INTENTO

int estado[] = {falso , falso};

Proceso 0:	Proceso 1:

while(estado[1])
/*nada*/;
estado[0] = true;
 	
../* SC */..
 	
estado[0] = false;
	while(estado[0])
	/*nada*/;
	estado[1] = true;
	
	../* SC */..
	
	estado[1] = false;

Exclusión Mutua: NO

Progreso: SI

Espera activa: SI

TERCER INTENTO

int estado[] = {falso , falso};

Proceso 0:	Proceso 1:

estado[0] = true;
while(estado[1])
/*nada*/;
 	
../* SC */..
 	
estado[0] = false;
	estado[1] = true;
	while(estado[0])
	/*nada*/;
	
	../* SC */..
	
	estado[1] = false;

Exclusión Mutua: SI

Progreso: NO (Bloqueo)

Espera activa: SI

Soluciones de Software

TERCER INTENTO

int estado[] = {falso , falso};

Proceso 0:	Proceso 1:

estado[0] = true;
while(estado[1])
/*nada*/;
 	
../* SC */..
 	
estado[0] = false;
	estado[1] = true;
	while(estado[0])
	/*nada*/;
	
	../* SC */..
	
	estado[1] = false;

Exclusión Mutua: SI

Progreso: NO (Bloqueo)

Espera activa: SI

CUARTO INTENTO

int estado[] = {falso , falso};

Proceso 0:	Proceso 1:

estado[0] = true;
while(estado[1]) {
estado[0] = false;
sleep();
estado[0] = true;
}
 	
../* SC */..
 	
estado[0] = false;
	estado[1] = true;
	while(estado[0]) {
	estado[1] = false;
	sleep();
	estado[1] = true;
	}
	
	../* SC */..
	
	estado[1] = false;

Exclusión Mutua: SI

Progreso: NO (Livelock)

Espera activa: SI

Soluciones de Software

- Soluciones que cumplen con los requisitos de la Sección Crítica:
  - Algoritmo de Dekker.
  - Algoritmo de Peterson:

int estado[] = {falso , falso};

Proceso 0:	Proceso 1:

estado[0] = true;
turno=1;
 	
while(estado[1] && turno == 1);
 	
../* SC */..
 	
estado[0] = false;
	estado[1] = true;
	turno=0;
	
	while(estado[0] && turno == 0);
	
	../* SC */..
	
	estado[1] = false;

Soluciones de Hardware

ENTRADA

SECCIÓN CRÍTICA

SALIDA

- Deshabilitar interrupciones.

- Instrucciones especiales de procesador.
  - Test and Set
  - Exchange

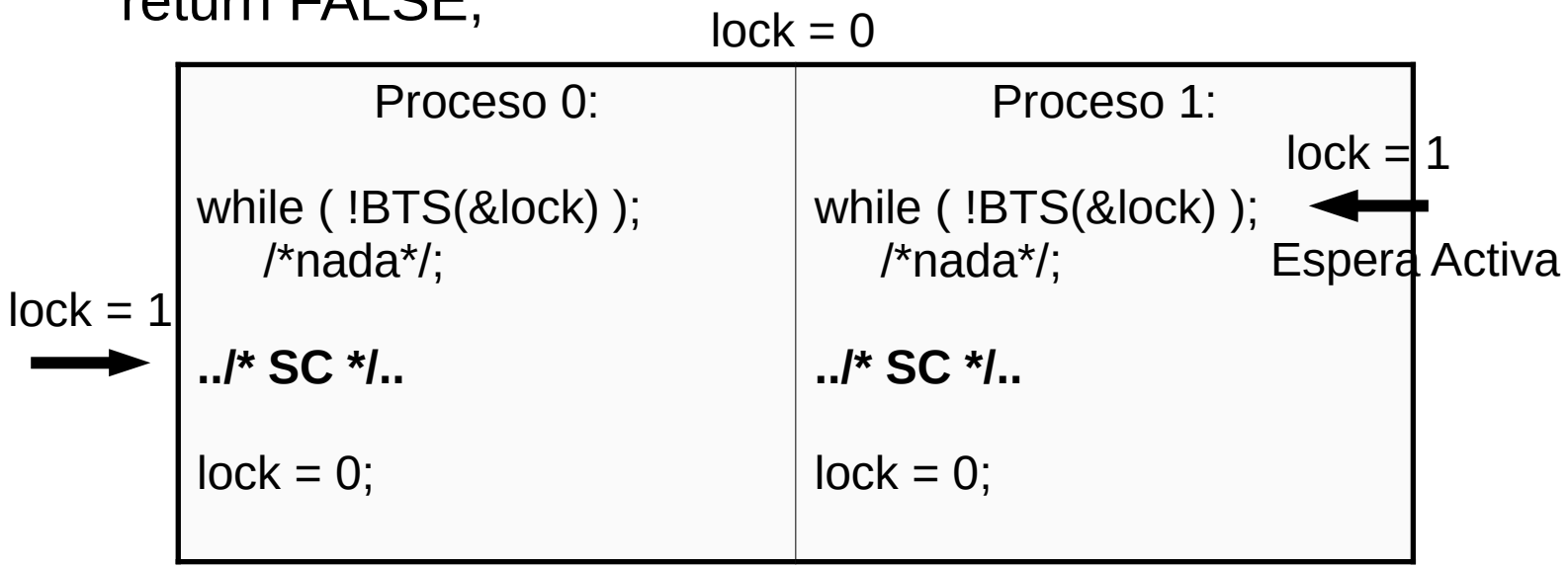
Soluciones de Hardware

```
BTS(*lock) {    //Test_and_set
     if (*lock == 0) {
          *lock = 1;
          return TRUE;
     }
     else
          return FALSE;
}
```

ENTRADA

SECCIÓN CRÍTICA

SALIDA



Semáforos

- Permite Exclusión Mutua entre varios procesos.
- Permite Sincronizar (u Ordenar) varios procesos.
- Permite Controlar acceso a recursos.
- Son utilizados mediante wait(s) y signal(s).
- Más simple de utilizar.

Semáforos

- Estructura:
  - Un valor entero.
  - Una lista de procesos bloqueado.

- Funciones sobre semáforos:
  - iniciar/finalizar un semáforo.
  - wait(sem) decrementa en uno el valor del semáforo.
  - signal(sem) incrementa en uno el valor del semáforo.

Semáforos

wait (s) {	signal (s) {
s->valor--;	s->valor++;
if (s->valor < 0);	if (s->valor <= 0);
bloquar(pid, s->lista);	pid = despertar(s->lista);
}	}

Semáforos

ENTRADA

SECCIÓN CRÍTICA

SALIDA

- Utilidad:
  - Exclusión Mutua:

s->valor = 1

Proceso 0:	Proceso 1:
wait(s);	wait(s);
../* SC */..	**../* SC */..**
signal(s);	signal(s);

Semáforos

- Utilidad:
  - Sincronizar:

s->valor = 1 / q->valor = 0

Proceso 0:	Proceso 1:
wait(s);	wait(q);
../* SC */..	**../* SC */..**
signal(q);	signal(s);

Semáforos

- Utilidad:
  - Acceso a recursos (N instancias):

s->valor = N

Proceso 0:	Proceso 1:
wait(s);	wait(s);
../* SC */..	**../* SC */..**
signal(s);	signal(s);

Semáforos

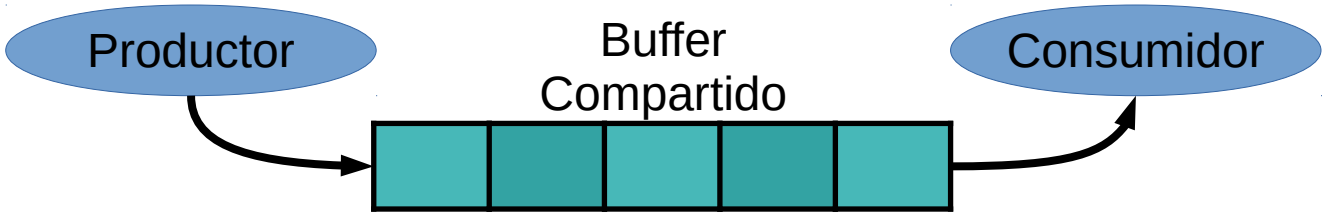
- Tipos de Semáforos:
  - General o Contador.
  - Binario (0 / 1).
    - Mutex (0 / 1).
- Valores de inicio de un semáforo: 0 o positivos.
- Valor del semáforo.

Semáforos

- Implementación de semáforos
  - “s” es variable compartida.
  - Requiere Exclusión Mutua.
    - Soluciones de Software.
    - Soluciones de Hardware.

Semáforos

- Productor / Consumidor



s_buffer = 1
s_cant = 0
s_lugar = N

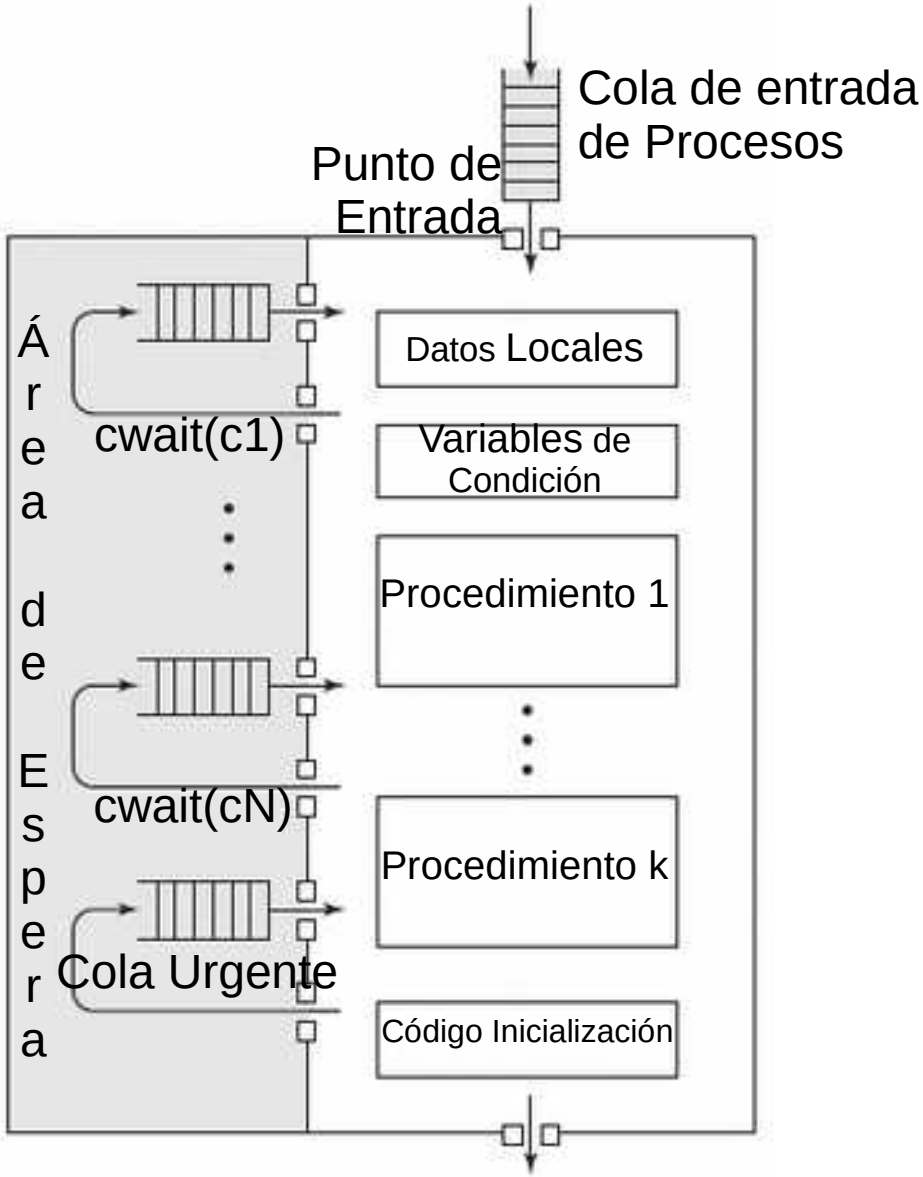
```
Productor()    {
    X = producir();
    wait(s_lugar);

    wait(s_buffer);
    agregar(X, buffer);
    signal(s_buffer);

    signal(s_cant);
}
```

```
Consumidor()    {
    wait(s_cant)

    wait(s_buffer);
    Y = extraer(buffer);
    signal(s_buffer);

    signal(s_lugar);
    consumir(Y);
}
```

Monitores

- Provistos por los (algunos) lenguajes de programación.
- Sólo un proceso o hilo puede estar utilizando el monitor en un determinado momento.

Monitores


Cola de entrada
de Procesos
Punto de
Entrada
Área de Espera
cwait(c1)
cwait(cN)
Cola Urgente
Datos Locales
Variables de
Condición
Procedimiento 1
Procedimiento k
Código Inicialización