Clase 4- semáforos

jueves, 5 de septiembre de 2024 13:13

Semáforo== es un tipo de dato abstracto con dos operaciones atómicas: p y v

Internamente el valor es un entero no negativo:

V → Señala la ocurrencia de un evento (incrementa). •
P → Se usa para demorar un proceso hasta que ocurra un evento (decrementa)

V no demora el proceso, p sí hasta que el semáforo esté con un valor mayor que 0 y recién ahí se decrementa

- **Declaraciones**

  sem s; → NO. Si o si se deben inicializar en la declaración
  sem mutex = 1;
  sem fork[5] = ([5] 1);

Tiene que ser iniciaaliado con algo mayor que 0

- **Semáforo general (o *counting semaphore*)**

  ***P(s)*:** ⟨ await (s > 0) s = s-1; ⟩
  ***V(s)*:** ⟨ s = s+1; ⟩

Busy waiting es más complejo e ineficiente
Los procesos que están esperando quedan demorados, no consumen procesador

Exclusión mutua

```
sem free= 1;
process SC[i=1 to n]
{  while (true)
      {  P(free);
         sección crítica;
         V(free);
         sección no crítica;
    }
}
```

Los semáforos que usamos pueden tomar valores mayores que 1 pero no deberíamos usarlo cuando lo usemos para exclusión mutua porque al hacerlo dejamos que más de un proc esté en su sección crítica a la vez

Barrera

Usamos un semáoforo para cada flad de sinconización. Cada proceso le dice a su semáforo que llegó con un V y lo limpia con un P

Cuando un proceso llega pone en v su semáforo y en p el del otro proceso. Así recién va a poer avanzar cuando el otro tb ponga su semáforo en v y el del otro en p

```
sem llega1=0, llega2=0;
process Worker1
{ .....
   V(llega1); P(llega2);
   .....
}
process Worker2
{ .....
   V(llega2); P(llega1);
   .....
}
```

Puedo hacer butterfly con esto de una manera medio volada

Semáforos binarios divididos:

Tengo que usar un recurso compartido tipo en sección crítica. La diferencia es que no cualquier proceso puede entrar. Hacen cosas distintas y se tiene que turnar quien va a usar el recurso compartido.

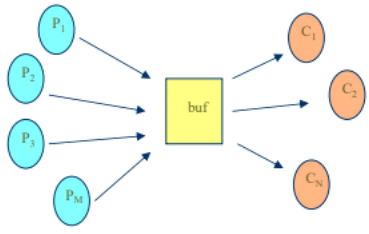
2 semáforos o más y la suma de todos tiene que dar entre 0 y 1

Semáforo Binario Dividido (Split Binary Semaphore). Los semáforos binarios b_1, ..., b_n forman un SBS en un programa si el siguiente es un invariante global:

$$SPLIT: 0 \leq b_1, + ... + b_n \leq 1$$

En general un proceso hace un p sobre un semáforo para esperar a que sea su turno y después un v sobre otro distinto para liberar el recurso

Ejemplo: buffer unitario compartido con múltiples productores y consumidores. Dos operaciones: ***depositar*** y ***retirar*** que deben alternarse.




```
typeT buf;  sem vacio = 1, lleno = 0;

process Productor [i = 1 to M]
{ while(true)
    { …
      producir mensaje datos
      P(vacio); buf = datos; V(lleno); #depositar
    }
}

process Consumidor[j = 1 to N]
{ while(true)
    { P(lleno); resultado = buf;  V(vacio); #retirar
      consumir mensaje resultado
      …
    }
}
```

vacio y ***lleno*** (juntos) forman un ***"semáforo binario dividido"***.

En ningún momento van a estar sumando más de 1 vacio y lleno

Contadores de recursos

Se medio solapa con señalización de eventos (barrera)
2 semáforos: uno cuenta cuantos lugares vacios tengo y uno cuantos lleno. El productor puede

generar hasta que se llene sin frenarse y el otro toma hasta que se vacia.
Van contando los lugares ocupados y vacios de un arreglo.

```
typeT buf[n];  int ocupado = 0, libre = 0;
sem  vacio = n, lleno = 0;
process Productor
{ while(true)
    { …
      producir mensaje datos
      P(vacio); buf[libre] = datos; libre = (libre+1) mod n;  V(lleno); #depositar
    }
}
process Consumidor
{ while(true)
    { P(lleno); resultado = buf[ocupado];  ocupado = (ocupado+1) mod n; V(vacio); #retirar
      consumir mensaje resultado
      …
    }
}
```

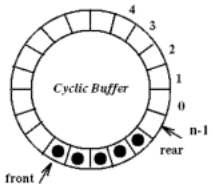
N es la cantidad de espacio que tiene el arreglo, a vacio lo inicializo con n porque está todo vacio.

Las ints son los lugares siguientes al que el productor tiene que dejar y el consumidor agarrar, es como un puntero.

Como uso un array pueden estar el consumidor y el generador poniendo y sacando del arreglo pq me asegura que no van a estar al mismo tiempo interactuando con la misma posición. En una cola/pila no podria (osea si pero hay que hacer más cositas)

Con varios productores y consumidores

```
typeT buf[n];  int ocupado = 0, libre = 0;
sem  vacio = n, lleno = 0;
sem mutexD = 1, mutexR = 1;
process Productor [i = 1..M]
{ while(true)
    { producir mensaje datos
      P(vacio);
      P(mutexD); buf[libre] = datos; libre = (libre+1) mod n; V(mutexD);
      V(lleno);
    }
}
process Consumidor [i = 1..N]
{ while(true)
    { P(lleno);
      P(mutexR); resultado = buf[ocupado];  ocupado = (ocupado+1) mod n; V(mutexR);
      V(vacio);
      consumir mensaje resultado
    }
}
```



Exclusión mutua selectiva: no todos comparten el mismo recurso compatido

Cuando tengo que usar procesos con cierto orden: passing de baton

Ej: usar cajero automático de a uno a la vez.
Respetar orden en el que entro a la sección critica con semáfotos

El proceso qe tiene el batón es quien puede usar la variable compartida que administra el cajero. Una vez que termina le pasa el baton a otro proceso. El que no, no puede.

Cada proceso tiene un semáforo. Voy prendiendo semáforos puntuales (a uno en partiular)
Contador para saber cuántos están esperando por entrar

```
int nr = 0, nw = 0;

process Lector [i = 1 to M]
{ while(true)
    { …
      ⟨ await (nw == 0) nr = nr + 1; ⟩
      lee la BD;
      ⟨ nr = nr – 1; ⟩
    }
}


process Escritor [j = 1 to N]
{ while(true)
    { …
      ⟨ await (nr==0 and nw==0) nw=nw+1; ⟩
      escribe la BD;
      ⟨ nw = nw – 1; ⟩
    }
}
```

```
int nr = 0, nw = 0, dr = 0, dw = 0;
sem e = 1, r = 0, w = 0;

process Lector [i = 1 to M]
{ while(true) {
        P(e);
        if (nw > 0){dr = dr+1; V(e); P(r); }
        nr = nr + 1;
        SIGNAL1 ;
        lee la BD;
        P(e); nr = nr – 1; SIGNAL2 ;
    }
}
process Escritor [j = 1 to N]
{ while(true) {
        P(e);
        if (nr > 0 or nw > 0) {dw = dw+1; V(e); P(w); }
        nw = nw + 1;
        SIGNAL3 ;
        escribe la BD;
        P(e); nw = nw – 1; SIGNAL4 ;
    }
}
```

??

? Revisar

Alcación de recursos y scheduling

Determinar cuándo se le puede dar a un proceso el acceso a un recurso
Esto sirve cuando son los mismos workers los que tienen que ver a quien le toca el recurso compartido (no un coordinador)

Problema: decidir cuándo se le puede dar a un proceso determinado acceso a un recurso.

Recurso: cualquier objeto, elemento, componente, dato, SC, por la que un proceso puede ser demorado esperando adquirirlo.

Definición del problema: procesos que compiten por el uso de unidades de un recurso compartido (cada unidad está ***libre*** o ***en uso***).

request (parámetros): ⟨await (request puede ser satisfecho) tomar unidades;⟩
release (parámetros): ⟨retornar unidades;⟩

Si no me importa el orden, uso como el problema de la sección crítica (cualquiera lo puede usar)
El tema es si me importa el orden

SJN(shortes job next)
En SJN, un proceso que invoca a request debe demorarse hasta que el recurso esté libre y su pedido sea el próximo en ser atendido de acuerdo a la política. El parámetro tiempo entra en juego sólo si un pedido debe ser demorado.

request (tiempo, id):
    P(e);
    if (not libre) DELAY;
    libre = false;
    SIGNAL;

release ( ):
    P(e);
    libre = true;
    SIGNAL;

- En ***DELAY*** un proceso:
  - Inserta sus parámetros en un conjunto, cola o lista de espera (*pares*).
  - Libera la SC ejecutando V(e).
  - Se demora en un semáforo hasta que *request* puede ser satisfecho.
- En ***SIGNAL*** un proceso:
  - Cuando el recurso es liberado, si pares no está vacío, el recurso es asignado a un proceso de acuerdo a SJN.

(delay)La cola está ordenada. Me quedo dormido en un semáforo privado
??
??
??
??
??
??
??
??
??