Clase 5- monitores

jueves, 12 de septiembre de 2024 13:08

Un monitor es la representación de un recurso compartido
Los monitores encapsulan las representaciones de recursos
Dan operaciones que son los medios para manipular los recursos. El proceso solo llama a funciones del monitor.

No hay que preocuparse por la exclusión mutua, es implícita. Solo puede haber un proceso dentro del monitor en simultáneo
Monitor es como secuencial. Lo van llamando y de a uno el monitor les da bola. Si lo sacan de la cpu y después vuelve va a seguir por la siguiente instrucción, no intercala cosas de procedimientos.

La única forma de liberar el monitor es o que termine el proceso o que el proceso se quiera ir

Interfaz: muestra nombre de los procedimientos y parámetros que necesita para que los procesos quieran usarlo sepan que necesita el monitor
Cuerpo: conjunto de variables que repreentan el estado del recurso y os procesos que implementan las operaciones de la interfaz.
NombreMonitor.opi (argumentos) para llamarlo

Un programa concrrente tiene **procesos activos y monitores pasivos**. Los procesos interactuan entre sí mediante la invocación de los procedures de un monitor

```
monitor NombreMonitor {
  declaraciones de variables permanentes;
  código de inicialización

  procedure op1 (par. formales1)
    {  cuerpo de op1
    }
  .....
  procedure opn (par. formalesn)
    {  cuerpo de opn
    }
}
```

Varibles permanentes: queda con el valor que le puso el último procedimiento que lo uso.
Código de inicialización: para inicializar variables complejas que son raris: ej inicializar matriz con ciertos valores.
Si un proceso tiene variables locales esas si son de la instancia, cada vez que se llame arranca con la variable de nuevo
Cuando quedan esperando para usar el monitor no se ejecutan siguiendo ning'n orden sino que lo hacen random

Como se comunican dos procesos: monitor con dos procesos. El proceso que quiere dar un dato se lo da al monitor y después el que lo quiere lo pide.
Como se maneja la exclusión mutua en monitores? Explícita pero hay que explicar por qué (pregunta de examen)

Sincronización por condición

Variables dentro del monitor
Se declaran con cond nombre y tiene una cola de procesos dormidos pq no tienen lo que necesitan para proseguir
Se usa como una cola fifo. El primer proceso dormido es el primero en salir
- wait(cv) --> el proceso se demora al final de la cola de cv y deja el acceso exclusivo al monitor.
- signal(cv) --> despierta al proceso que está al frente de la cola (si hay alguno) y lo saca de ella. El proceso despertado recién podrá ejecutar cuando reaquiera el acceso exclusivo al monitor. No se le da de una el acceso al monitor
- signal_all(cv) --> despierta todos los procesos demorados en cv, quedando vacía la cola asociada a cv.

En la materia usamos **signal and continue**, osea que el que se sigue ejecutando en el monitor es el que hizo el signal, no el que desperté. El despertado sale y compite por volver a entrar al monitor

Signal and wait el que llamó libera al monitor y el que lo usa es a quien desperté

WAIT	P
El proceso siempre se duerme	El proceso sólo se duerme si el semáforo es 0.

SIGNAL	V
Si hay procesos dormidos despierta al primero de ellos. En caso contrario no tiene efecto posterior.	Incrementa el semáforo para que un proceso dormido o que hará un P continue. No sigue ningún orden al despertarlos.

```
process A {
   int aux;
   while (true)
      { --Genera valor a enviar en aux
        Buffer.Enviar (aux);
        ……
      }
}
```

```
process B {
   int aux;
   while (true)
      { …….
        Buffer.Recibir (aux);
        --Trabaja con en vlor aux recibido
      }
}
```

```
monitor Buffer{
  int dato;
  bool hayDato = false;
  cond P, C;

  procedure Enviar (D: in int)
    {  if (hayDato) → wait (P);
       dato = D;
       hayDato = true;
       signal (C);
    }

  procedure Recibir (R: out int)
    {  if (not hayDato) → wait (C);
       R = dato;
       hayDato = false;
       signal (P);
    }
}
```

Esto sirve si hay solo 1 productor y un consumidor.

Simulado un semáforo

Con un if puede pasar que desperté a uno pero el turno se lo robó otro

```
monitor Semaforo
{  int s = 1;   cond pos;

   procedure  P ()
      { while (s == 0) wait(pos);
        s = s-1;
      };

   procedure V ()
      { s = s+1;
        signal(pos);
      };
};
```

En los semáforos no hay orden entre los que están dormidos o despiertos, en los monitores medio que sí pq yo despierto al siguiente que está dormido en la cola
Si pongo un signal all sería como en semáforos, pero es muy ineficiente porque estoy todo el tiempo durmiendo y despertando procesos.

Passing the conditions
Cuando quiero mantener cierto orden en la ejecución de las cosas. No puedo controlar quien de los que están esperando va a entrar, con esto me aseguro que si desperté a uno el proximo que ejecuta es el que desperté. Puede que entren algunos que no me importen pero se van a volver a dormir

Simulación de Semáforos

```
monitor Semaforo
{  int s = 1;   cond pos;

   procedure  P ()
      { if (s == 0) wait(pos)
        else s = s-1;
      };

   procedure V ()
     { if (empty(pos) ) s = s+1
       else signal(pos);
     };
};
```

Como resolver este problema al no contar con la sentencia *empty*.

```
monitor Semaforo
{  int s = 1, espera = 0;  cond pos;

   procedure  P ()
      { if (s == 0) { espera ++; wait(pos);}
        else  s = s-1;
      };

   procedure V ()
     { if (espera == 0 ) s = s+1
       else { espera --; signal(pos);}
     };
};
```

Re importante hacerlo antes del wait pq sino queda en 0

SJN (wait con prioridad)

Se realiza Passing the Condition, manejando el orden explícitamente por medio de una cola ordenada y variables condición privadas.

```
monitor Shortest_Job_Next
{ bool libre = true;
  cond turno[N];
  cola  espera;

  procedure request (int id, int tiempo)
     { if (libre) libre = false
       else { insertar_ordenado(espera, id, tiempo);
              wait (turno[id]);
            };
     };

  procedure release ()
    { if (empty(espera)) libre = true
      else   { sacar(espera, id);
               signal(turno[id]);
             };
     };
}
```