Monitor SU para múltiples clientes y recursos

Sistemas Concurrentes y Distribuidos.

Ricardo Ruiz Fernández de Alba

08/11/2023

Proponemos el siguiente monitor para resolver el problema:

```
1  Monitor Recursos;
2  var
3      libre: array[1,...,m] of boolean;
4      peticion: array[1,...,n] of boolean;
5      I: integer; // Índice del último recurso liberado
6      no_peticion: condition;
7
8  procedure pedir(id_proceso: 1,...,n);
9  var k: integer; k := 1;
10 begin
11     peticion[id_proceso] := true;
12     while (not libre[k] and k < m ) do
13         k := k + 1;
14     end
15     if (not libre[k]) then
16         while(peticion[id_proceso]) do
17             no_peticion.signal() // Desbloqueo en cadena
18
19             {IM ^ L}
20             no_peticion.wait();
21             {C[id_proceso] ^ L}  // Se cumple la condición de
                   sincronización. Se comprueba si hay otras peticiones
                   prioritarias (menor índice de proceso)
22         end
23     else I := k;
24     end
25     libre[I] := false;
26 end
27
28 procedure devolver(id_recurso: 1,...,m);
29 var j: integer; j := 1;
30 begin
31     while (not peticion[j] and j < n) do
32         j := j + 1;
33     end
34     I := id_recurso;
35     if (petición[j]) then
36         {no_vacio(no_peticion) ^ L ^ C[j]}
37         peticion[j] := false;
38         no_peticion.signal();
39         {IM ^ L}
40     else libre[I] := true;
41     end
42 end
```

Código de inicialización

```
1  begin
2  {V}
3      for i := 1 to m do
4          libre[i] := true;
5      end
6      for i := 1 to n do
7          peticion[i] := false;
8      end
9  {IM ^ L}
10 end
```

Así pues, los procesos clientes P_i llamarán al procedimiento pedir con parámetro el identificador i del proceso.

Este procedimiento busca en el array de libres el primer recurso disponible. Si todos los recursos están ocupados, se termina con `k=m` y `libre[k$]=false`. En ese caso, se quqeda bloqueado en la cola de la variable condición no_peticion y con el valor `peticion[id_proceso] = true` hasta que haya un recurso disponible.

Cuando otro proceso llame al procedimiento devolver con parámetro el número de recurso, almacenará `I := id_recurso` como el último recurso liberado y se busca la primera petición pendiente, actualizando `peticion[j] := false` y señalando al proceso que tiene petición pendiente.

Como el monitor es de Señales Urgentes (SU), el proceso que espera en la cola se desbloquea, comprobando de nuevo si es el que cumple `C[id_proceso]` la condición lógica de sincronización. Así, desbloquea en cadena otros posibles procesos en la cola, y sale del while aquel con mayor prioridad (menor id de proceso). Finalmente, se escribiría `libre[I] := false`.

Si por el contrario hay procesos libres, `pedir(id_proceso)` actualiza `I := k` y `libre[I] := false`.

Igualmente, si no hay peticiones pendientes, `devolver(id_recurso)` actualiza `I := id_recurso` y `libre[I] := true`, dejando `id_recurso` libre para futuras peticiones de procesos.

Finalmente, se ha verificado formalmente que el invariante del monitor se cumple tras la inicialización, antes y después de cada procedimiento, y para la operacion `signal`, se cumple $\{no\_vacio(peticion) \land C[j] \land L\}$ como precondicion e $\{IM \land L\}$ como poscondición.

Y $\{IM \land L\}$ como precondición para la operación `wait` y $\{C[id\_proceso] \land L\}$ como poscondición.

Una ejecución del monitor para 4 procesos y 3 recursos sería:

```
process P1
begin
Recursos.pedir(1);
end

process P2
begin
Recursos.pedir(2);
end

Process P3
begin
Recursos.pedir(3);
end

cobegin
Recursos.pedir(4) | Recursos.pedir(1) | Recursos.devolver(1) | Recursos
    .devolver(2)
coend
```

Una posible interfoliación sería, que tras pedir 3 recursos, el array de libres quedaría ocupados, haciendo que P4 tenga dos procesos en la cola `no_peticion` hasta que se libere un recurso.

Una vez se ejecute el procedimiento devolver , el proceso 4 se desbloquearía, desbloqueando a su vez al proceso 1, que al tener menor índice de proceso, se ejecutaría primero. Por tanto, se devolvería el primer recurso que quedaría ocupado de nuevo. El proceso 4 se bloquearía de nuevo hasta que se libere el segundo recurso. Una vez este liberado, se ocuparía de nuevo terminando la ejecución del monitor.