Solution of a Problem in Concurrent Programming Control

Edsger Wybe Dijkstra
Technological University, Eindhoven, The Netherlands

Solución de un Problema de Control en Programación Concurrente

Traducción
Edgar Serna Montoya
Fundación Universitaria Luis Amigó
edgar.sernamo@amigo.edu.co

Es posible diseñar una serie de procesos secuenciales-cíclicos, principalmente independientes, con medios de comunicación limitados entre sí, de tal manera que en cualquier momento uno y sólo uno de ellos esté ocupado en la "sección crítica" de su ciclo.

Introducción
En este trabajo se describe una solución a un problema que, independientemente de su resolución y de acuerdo con mí conocimiento, es una cuestión que ha estado abierta por lo menos desde 1962. El documento consta de tres partes: el problema, la solución, y la prueba. Aunque en un primer momento el escenario del problema puede parecer un tanto académico, el autor confía en que cualquiera que esté familiarizado con los problemas lógicos que surgen en el acoplamiento del computador apreciará la importancia del hecho de que este problema se puede solucionar.

El problema
Para comenzar, considere N computadores, cada uno ocupado en un proceso que, para nuestros fines, puede ser considerado como cíclico. En cada uno de los ciclos ocurre una, así llamada, "sección crítica" y los computadores se deben programar de tal forma que en cualquier momento sólo uno de esos N procesos cíclicos esté en su sección crítica. Con el fin de efectuar esta exclusión mutua de la ejecución de la sección crítica los computadores pueden comunicarse entre sí a través de una reserva común. Escribir una palabra o leer una palabra de forma no destructiva desde esta reserva son operaciones inseparables; es decir, cuando dos o más computadores tratan de comunicarse simultáneamente —sea para leer o para escribir— con la misma ubicación común, estas comunicaciones se llevarán de un lugar a otro, pero en un orden desconocido.

La solución debe satisfacer los siguientes requerimientos:

a) La solución debe ser simétrica entre los N computadores; no se nos permite introducir una prioridad estática como un resultado.
b) No se puede asumir nada con respecto a la velocidad relativa de los N computadores; no podemos asumir que su velocidad es constante en el tiempo.
c) Si alguno de los computadores se detiene por fuera de su sección crítica, esto no debe originar un potencial bloqueo de los demás.
d) Si más de un computador está a punto de entrar en su sección crítica, les debe ser imposible, a esas velocidades, tomar una decisión para determinar cuál de ellos entrará en primero en su sección crítica por lo que se aplazará eternamente. En otras palabras, construcciones en las que todavía son posibles los bloqueos "después de usted", aunque improbables, no se deben considerar como soluciones válidas.

Desafiamos al lector a parar aquí por un momento y realizar él mismo un intento, ésta parece ser la única forma de hacerse a la idea de las difíciles consecuencias del hecho de que cada computador puede solicitar sólo un mensaje a la vez en un sólo sentido. Y sólo esto hará que el lector se dé cuenta hasta qué punto este problema está lejos de ser trivial.

La solución
La reserva común consiste de: "*Boolean array b, c[1:N]; integer k*".

El entero k debe satisfacer la lista $1<k<N$, y *c[i]* sólo la fijará el *i*-ésimo computador, y será inspeccionado por los demás. Se supone que todos los computadores se inician por fuera de sus secciones críticas y con todos los conjuntos de matrices booleanas mencionadas establecidos en Verdadero; el valor inicial de k es irrelevante.

El programa para el *i*-ésimo computador $(1<i<N)$ es:

```
"integer j;
Li0:    b[i] := false;
Li1:    if k ≠ i then
Li2:      begin c[i] := true;
Li3:      if b[k] then k := i;
            go to Lil
            end
        else
Li4:    begin c[i] := false;
```

```
    for j := 1 step 1 until N do
        if j ≠ i and not c[j] then go to Li1
        end;
    critical section;
    c[i] := true; b[i] := true;
    resto del ciclo en el que se permite parar;
    go to Li0"
```

La prueba

Empecemos por observar que la solución es segura en el sentido de que no puede haber dos computadores al mismo tiempo en su sección crítica. La única manera de entrar en su sección crítica es mediante la ejecución de la sentencia compuesta *Li*4 sin regresar a *Li*1; es decir, buscar todos los otros *c* verdaderos después de haber establecido su propio *c* en false.

La segunda parte de la prueba debe demostrar que no pueden ocurrir bloqueos infinitos "después de usted"; es decir, cuando ninguno de los computadores está en su sección crítica, del bucle de los computadores (es decir, regresando a *Li*1) se le permitirá por lo menos a uno entrar en su sección crítica a su debido momento (y por consiguiente exactamente a uno).

Si el computador k-ésimo no se encuentra en uno de los bucles, b[*k*] será verdadero y los otros bucles se encontrarán en $k \neq i$. Como resultado uno o más de ellos se encontrarán en el booleano *Li*3 con b[*k*] verdadero y por lo tanto uno o más decidirán asignarse "*k*: = *i*". Después de la primera asignación "*k*: = *i*", b[*k*] se convierte en falsa y ningún otro computador podrá decidir volverse a asignar un nuevo valor de *k*. Cuando se han realizado todas las asignaciones a *k*, *k* apuntará a uno de los computadores del bucle y por el momento no cambiará su valor, es decir, hasta que b[*k*] sea verdadero, a saber, hasta que el computador k-ésimo haya completado su sección crítica. Tan pronto como el valor de *k* no cambie más, el computador k-ésimo esperará hasta que todos los otros *c* sean verdaderas (a través de la sentencia compuesta *Li*4), pero sin duda esta situación se producirá, si no está ya presente, debido a que todos los otros bucles son forzados a establecer en verdadero su *c*, ya que encontró $k \neq i$. Y esto, creo, completa la prueba.

E. W. Dijkstra. "Solution of a Problem in Concurrent Programming Control". *Communications of the ACM*, Vol. 8, No. 9, p. 569. September, 1965. Ω