# Problemas de sincronización – Barrera

Sistemas Operativos – 7 de septiembre de 2017

Matías Barbeito

## 1. Enunciado

Queremos hacer una Barrera de sincornización o Rendezvous (punto de encuentro). Cada $P_i$, $i \in [0 \dots N-1]$, tiene que ejecutar $a(i)$; $b(i)$, pero sólo puede ejecutar $b(i)$ si para todo $j \in [0 \dots N-1]$ ya $P(j)$ realizó $a(j)$.

Propiedad *BARRERA* a garantizar:

$$b(i) \text{ se ejecuta después de } \textbf{todos} \text{ los } a(j)$$

Es decir, queremos *poner una barrera* entre los $a$ y los $b$. Pero esto debe hacerse sin restringir de más: no hay que imponer ningún orden entre los $a(i)$ ni entre los $b(i)$.

## 2. Solución

```
1  atomic<int> cant = 0;  // Procs que terminaron a
2  semaphore barrera = 0; // Barrera baja
3
4  proc P(i) {
5    // R:
6    a(i);
7    // A: terminó a
8    // T:
9    if (cant.getAndInc() < N-1)
10     barrera.wait();
11   // C: no hay nada que hacer
12   // E:
13   barrera.signal();
14   // B: se puede hacer b
15   b(i);
16 }
```

## 3. Demostración de BARRERA

Queremos probar que para todo instante $k$ de la secuencia y para todo proceso $i \in [0 \dots N-1]$, si $i$ está en $B$ en $k$ (es decir que puede hacer $b(i)$) entonces para todo proceso $j$, existe un instante $k_j < k$ tal que $j$ está en $A$ en $k_j$ (terminó a(j)). Lo demostraremos por el absurdo. Supongamos que existe una ejecución en la que $i$ está en $B$ pero hay al menos un $j$ que aún está en $R$. Más formalmente, existe una secuencia $\tau_0 \rightarrow \tau_1 \dots$ y un $k$ tal que $\tau_k(i) = B$ y $\tau_k(j) = R$.

De lo anterior podemos deducir que existe un estado previo, llamémoslo $k' < k$, donde $\tau_{k'}(i) = T$. Para que luego $i$ llegue a $B$ debe haber sucedido alguna de las siguientes dos posibilidades:

1. o bien el resultado de *cant.getAndInc*() para $i$ fue $N - 1$
2. o bien otro proceso generó un signal que lo despertó

Analicemos el primer caso. Para que *cant.getAndInc*() de como resultado $N-1$ todos los procesos deben haber llegado al estado $T$, ya que es un *atomic* $<$ *int* $>$ que no puede sufrir condiciones de carrera, que comienza con valor 0 y que sólo se incrementa al entrar en el estado $T$. Por lo tanto es imposible que el proceso $j$ se encuentre en el estado $R$ si el proceso $i$ leyó $N - 1$ de *cant*. Absurdo.

Ahora veamos el segundo caso. Digamos entonces que un proceso $h$ en un instante $k''$ con $k' < k'' < k$ se encuentra en el estado $\tau_{k''(h)} = E$. Pero entonces $h$ debe haber estado previamente en un estado $T$ habiendo las mismas dos posibilidades que analizamos previamente. En resumen, para llegar a este caso también haría falta que todos los procesos hayan llegado al estado $T$ en un instante previo a $k$, pero $j$ se encuentra en estado $R$ en $k$. Absurdo.

El absurdo en ambos casos provino de suponer que un proceso $j$ podía estar aún en estado $R$ cuando otro proceso $i$ ya se encontraba en estado $B$. Por lo tanto si un proceso está en estado $B$ no puede haber ningún proceso en $R$. Es decir que vale **BARRERA**.

# 4. Demostración de G-PROG

Queremos probar que $\forall i.\Box OUT(i) \Rightarrow \forall i.\Box IN(i)$ con $OUT(i) \equiv CRIT(i) \Rightarrow \Diamond REM(i)$ y con $IN(i) \equiv TRY(i) \Rightarrow \Diamond CRIT(i)$.

Primero asumiremos que $a()$ termina para todo $i$. Luego, como hay $N$ procesos y asumimos que $a()$ termina para todos ellos, todos llegarán a $T$ y ejecutaran *cant.getAndInc*(). Por lo tanto existe un proceso $p$ tal que *cant.getAndInc*() $< N - 1$ es *false* y por $OUT(i)$ llegarán a $C$ y a $E$. Es decir que existe un instante $k$ tal que $\tau_k(p) = E$. Entonces, al ejecutarse *barrera.signal*() alguno de los restantes $N - 1$ procesos esperando por el semáforo *barrera* será despertado. En consecuencia existe un intante $k' > k$ y un proceso $p'$ tales que $\tau_{k'}(p') = E$ ya que el proceso $p'$ fue el que se despertó por el *barrera.signal*() generado por $p$. Se puede notar que luego ocurrirá lo mismo con un proceso $p''$ y así sucesivamente, ya que todos los procesos realizaran un *barrera.signal*(). Por lo tanto todos los procesos que alcancen el estado $T$ llegarán al $C$.

# 5. WAIT-FREE

Queremos ver si cumple $WAiT - FREE \equiv \forall i.\Box IN(i)$ con $IN(i) \equiv TRY(i) \Rightarrow \Diamond CRIT(i)$.

En este caso podemos notar que la propiedad no se cumple ya que si un proceso que se encuentra en $C$ muere o se cuelga los demás quedarán bloqueados en el semáforo *barrera* y nunca podrán llegar a la zona crítica.