Los mecanismos de sincronización están implementados en espera activa.

Desde un principio pensaron en mecanismos para implementar la inclusión mutua de forma fácil.

El primer mecanismo que se les ocurrió fue Disjkstra, que fueron los Semáforos.

El semáforo es un tipo abstracto de datos, es una clase de las cual podemos declarar las variables. Las cuales tienen un valor entero entero asociado, siempre mayor o igual que cero.

Solo se puede realizar dos operaciones:

En Java es la clase Semaphore

* Acquire()
* Release()

En otros lenguajes:

* Wait
* Signal

Cuando creamos el semáforo podemos crearlo con un valor inicial, mayor o igual a cero.

Una vez creado, podemos realizar dos acciones:

* Acquire() 🡪 if s > 0 then s = s-1 else dormir.proceso;
* Release() 🡪 if hay\_procesos\_bloqueados then despertar\_uno else s=s+1

Normalmente los semáforos estarán implementados con colas FIFO, pero no tiene porque.

Son operaciones atómicas.

While(true){

//Preprotocolo

Sec\_Crítica 🡪 mutex.acquire;

//Postprotocolo

No\_Critica 🡪 mutex.release;

}

Los semáforos que utilizamos para exclusión mutua, siempre tendrán de valor inicial 1.

Semaphore mutex = new Semaphore(1);

Cuando entra en la sección critica, el semáforo vale 0.

Cuando salgo de la sección critica, como el semáforo vale 0, despierto un proceso.

Si no hay procesos dormidos, volvemos a poner el semáforo a 1.

Semaphore mutex\_pantalla = new Semaphore(1);

M1, m2, m3;

M1.acquire

M2.acquire

M3.acquire

….

….

M3.release

M2.release

M1.release

Mutex\_red = new Semaphore(1);

Th1

….

…

Mutex\_pantalla.acquire;

Mutex\_red.acquire;

Sec\_Critica(accedo a pantalla y red)

Mutex\_red.release;

Mutex\_pantalla.release;

….

….