Sistemas Operativos

SpinLocks

Rodrigo Urrea

Agenda Auxiliar

P1.

Distribución recurso compartido

P2.

Sistema de reuniones

Po.

Repaso

SpinLock

- Herramienta de sincronización primitiva.
- No dependen ni del Sistema Operativo ni del Scheduler.

Modo de uso:

- Creación:

  int lk = OPEN;
- Inicio sección crítica:

  spinLock(&lk);
- Fin sección crítica:

  spinUnlock(&lk);

Consistent Locking Behavior

Cuando 2 cores quieren acceder a una dirección d, uno de ellos modificandola, se debe dar que:

1. Core X obtiene mutex
2. Core X accede a d
3. Core X libera mutex
4. Core Y obtiene mutex
5. Core Y accede a d
6. Core Y libera mutex

Sanitize verifica que se cumpla con esta metodología

P1.

Distribución recurso compartido

Distribución recurso compartido (Tarea 6 2022-1)

Se dispone de un recurso único compartido por los múltiples cores de una máquina sin sistema operativo.

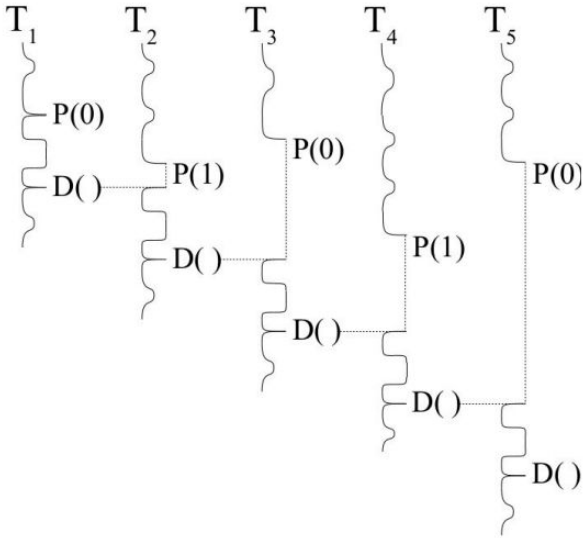
Los cores se agrupan en categoría 0 y categoría 1. Se dice que 0 es la categoría opuesta de 1, y 1 la categoría opuesta de 0. Un core de categoría *cat* solicita el recurso invocando la función ***pedir(cat)*** y devuelve el recurso llamando a la función ***devolver( )***, sin parámetros.

Se necesita programar las funciones ***pedir*** y ***devolver*** garantizando la exclusión mutua al acceder al recurso compartido. Se requiere una política de asignación alternada primero y luego por orden de llegada. Esto significa que cuando un core de categoría *cat* devuelve el recurso, si hay algún core en espera de la categoría opuesta a *cat*, el recurso se asigna inmediatamente al core de categoría opuesta que lleva más tiempo esperando.

Si no hay threads en espera de la categoría opuesta pero sí de la misma categoría *cat*, el recurso se asigna al core que lleva más tiempo en espera. Si no hay ningún core en espera, el recurso queda disponible y se asignará en el futuro al primer core que lo solicite, cualquiera sea su categoría.

Distribución recurso compartido (Tarea 6 2022-1)

El siguiente diagrama muestra un ejemplo de asignación del recurso. La invocación de pedir se abrevió como P(…) y la de devolver como D( ).



Distribución recurso compartido (Tarea 6 2022-1)

Programe las funciones ***pedir***, ***devolver***, ***iniciar*** y ***terminar*** con encabezados:

- void pedir(int cat);
- void devolver();
- void iniciar();
- void terminar();

Dado que la máquina no posee un sistema operativo, la única herramienta de sincronización disponible son los ***spin-locks***. Ud. sí dispone del tipo ***Queue***. No necesita saber que la máquina tiene 8 cores.

Dado que no hay sistema operativo, no puede usar funciones como *malloc*, *pthread_mutex_lock*, *sem_wait*, etc. Inicialice las variables globales en la función ***iniciar*** y libere los recursos solicitados en ***terminar*** (como las colas de cores en espera).

P2.

Sistema de reuniones

Sistema de reuniones (C2 2018-1)

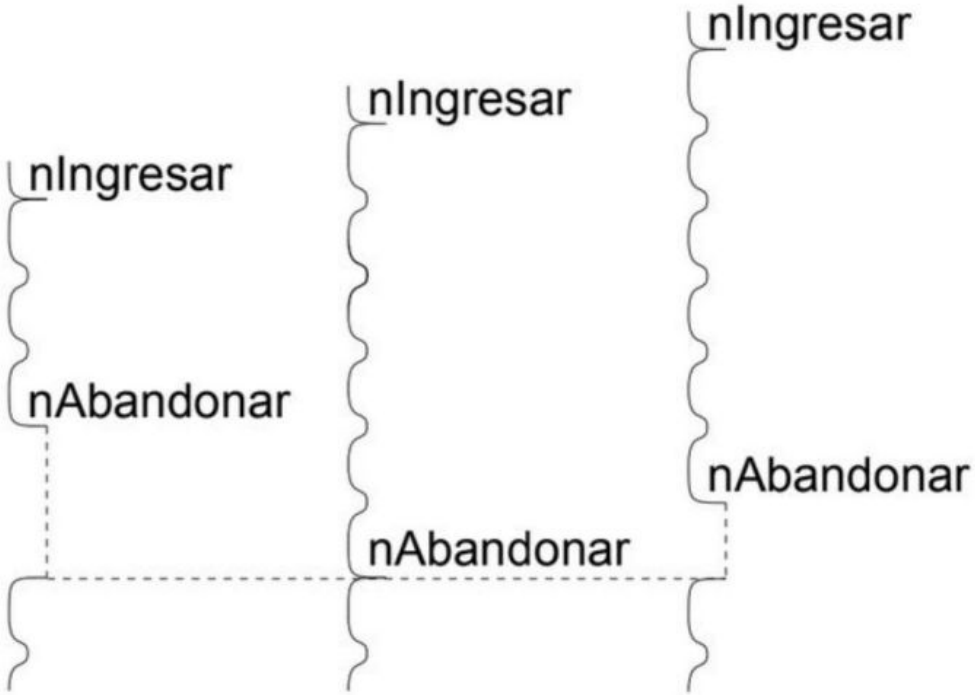
Considere una máquina con 8 cores físicos que comparten la memoria, sin un núcleo de sistema operativo y por lo tanto no hay scheduler de procesos.

Se requiere implementar un sistema de reuniones. Las funciones son las siguientes:

- ***void ingresar( )***: notifica el ingreso a la reunión.
- ***void abandonar( )***: solicita finalizar la reunión. Se bloquea en espera hasta que todos los cores que hayan ingresado hayan solicitado finalizar la reunión. Esto incluye aquellos cores que ingresen después de esta invocación de abandonar.

Para programar su solución Ud. dispone de ***spin-locks*** y la función ***coreId()*** que entrega el número del core que la invoca. Necesitará declarar variables globales.

Sistema de reuniones (C2 2018-1)



Agenda Auxiliar

P1.

Distribución recurso compartido

P2.

Sistema de reuniones

P-Extra.

Portero (Ex 2022-1)

Portero (Ex 2022-1)

El tipo Portero sirve para evitar que entren a una tienda más de ***max*** personas simultáneamente. Se le pide que defina el tipo ***Portero*** y programe las siguientes funciones:

- ***void iniPortero(Portero *p, int max)***: Inicializa el portero ***p*** que permite que ingresen simultáneamente hasta ***max*** threads.
- ***void entrar(Portero *p)***: Solicita al portero ***p*** ingresar a la tienda. Si ya hay ***max*** personas en el interior debe esperar.
- ***void salir(Portero *p)***: Notifica al portero ***p*** la salida de la tienda. Si hay threads en espera se hace pasar al thread que lleva más tiempo esperando.

Para programar lo pedido usted cuenta con una máquina multicore sin sistema operativo. Por lo tanto la única herramienta de sincronización disponible es el ***spin-lock***. Para hacer esperar a los cores use una cola (tipo ***Queue***) de spin-locks. Debe respetar el orden de llegada.

Sistemas Operativos

SpinLocks

Rodrigo Urrea