CC4302 Sistemas Operativos – Control 2 – Semestre Primavera 2014 – Prof.: Luis Mateu

Pregunta 1

Parte I.- (1 punto) Las variables globales son siempre una fuente de dataraces en los programas multi-thread. Explique cómo se resuelve el problema de los dataraces que podría producir las variables globales declaradas en un núcleo clásico de Unix. ¿Y en un núcleo moderno?

Parte II.- (5 puntos) Se desea agregar locks de lectura/escritura a nSystem. Estos locks son una solución para el problema de los lectores/escritores visto en clases. La API es la siguiente:

Procedimiento	***Significado***
`void nEnterRead(nRWLock* l);`	Solicitud para leer
`void nExitRead(nRWLock* l);`	Notificación de salida de lectura
`void nEnterWrite(nRWLock* l);`	Solicitud para escribir
`void nExitWrite(nRWLock* l);`	Notificación de salida de escritura

Implemente esta API, es decir la estructura *nRWLock* y los procedimientos *nEnterRead*, *nExitRead*, *nEnterWrite* y *nExitWrite*, como mecanismo de sincronización nativo de nSystem. Esto significa que Ud. debe implementar esta API usando los procedimientos de bajo nivel de nSystem (*START_CRITICAL*, *Resume*, *PutTask*, etc.). Ud. *no puede usar* otros mecanismos de sincronización ya disponibles en nSystem, como semáforos, monitores, mensajes, etc.

Requerimiento de simplicidad: su implementación debe dar prioridad a los escritores aún cuando esto signifique una eventual hambruna para los lectores.

Pregunta 2

Parte a.- (1 punto) Un programador plantea eliminar el busy-waiting de los spin-locks de la siguiente manera: cuando se pida un spin-lock que está cerrado, entonces suspender el thread que lo pide y retomar otro thread. ¿Cómo le respondería Ud.?

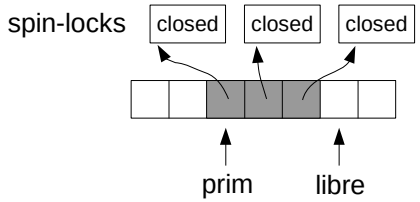
Parte b.- (5 puntos) Se tiene una máquina *octa-core* en la que no existe un núcleo de sistema operativo y por lo tanto la única herramienta de sincronización preexistente son los *spin-locks*. Se le pide programar un mutex que sirva para garantizar la exclusión mutua de los cores por períodos de unos pocos microsegundos. La API es la siguiente:

Procedimiento	***Significado***
`void InitMutex(Mutex *l);`	Inicializa un mutex
`void Lock(Mutex* l);`	Solicita obtener el mutex
`void Unlock(Mutex* l);`	Devuelve el mutex

Requerimientos: (i) el mutex debe ser eficiente en caso de alta contención (varios cores tratando de obtener el mutex) y (ii) las solicitudes para obtener el mutex se deben atender por orden de llegada.

Recuerde que dado que no hay núcleo de sistema operativo Ud. no dispone de monitores o semáforos y tampoco puede usar funciones como *Resume*, *PutTask*, etc. No existe *malloc* ni el tipo cola fifo. Por lo tanto, cuando un core debe esperar para obtener el mutex, no queda otra alternativa que esperar en un spin-lock.

Metodología: Use un arreglo circular de 7 punteros a spin-locks organizado como el buffer del problema del productor/consumidor.



La variable *prim* es el índice del elemento en el arreglo que contiene el puntero a un spin-lock que está cerrado, y que ha sido solicitado por el core que está en primer lugar en la fila para obtener el mutex. La variable *libre* es el índice del elemento en el arreglo que servirá para dejar esperando al próximo core que solicite el mismo mutex. El arreglo es circular: si se llega al índice 7, se vuelve a comenzar con el índice 0. No pueden haber más de 7 cores esperando por el mismo mutex. El octavo core es el que posee el mutex.

Use un spin-lock para garantizar la exclusión mutua durante el acceso al arreglo circular.

Parte c.- (Bonus de 1 punto) ¿Por qué los elementos del arreglo deben ser punteros a los spin-locks y no los spin-locks directamente? (Hint: en general los elementos adyacentes en un arreglo forman parte de la misma línea en la memoria cache.)