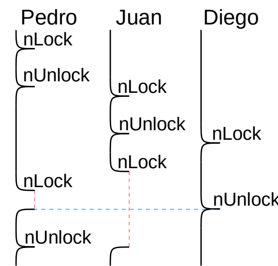
CC4302 Sistemas Operativos – Examen – Semestre Primavera 2021 – Prof.: Luis Mateu

Pregunta 1

Se necesita implementar de manera nativa en nSystem un nuevo mutex que satisface la siguiente propiedad: si al devolver el mutex hay varios threads en espera de ese mutex, el mutex se otorga al thread que lleva más tiempo sin usar el mutex. Estas son las operaciones que debe implementar:

void MutexInit( );
void nLock( );
void nUnlock( );



La figura de la izquierda es un ejemplo de uso. Observe que cuando el thread Diego libera el mutex, Pedro obtiene el mutex antes que Juan, a pesar de haberlo solicitado después que Juan. Esto es porque Pedro usó por última vez el mutex antes que Juan.

Implemente *nLock y nUnlock* 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. Por simplicidad, hay uno solo de estos mutex, por lo que debe usar variables globales para implementarlos. No necesita implementar algún tipo de condiciones asociadas a este mutex. Tampoco se especifica una prioridad para los threads que solicitan el mutex por primera vez. Inicialice las variables globales que requiera en la función *MutexInit*.

***Metodología obligatoria*:** En la implementación de nSystem incluida en los archivos adjuntos, use el estado predefinido *WAIT_MUTEX*. Para otorgar el mutex en el orden solicitado, defina una variable global con un número serial. Increméntelo cada vez que otorgue el mutex y copie su valor en el descriptor del thread vencedor en un campo predefinido *last_serial*. Al liberar el mutex, entre todos los threads en espera, el que lleva más tiempo sin usar el mutex es aquel thread cuyo descriptor almacena el número serial más pequeño. Este mecanismo es fácil de programar con las colas de prioridad que ya vienen implementadas en el archivo *nMutex.c*. Defina una variable global con una cola de prioridades. Cuando se solicite un mutex ocupado, haga esperar al thread en la cola de prioridad. Su prioridad es el campo *last_serial* del descriptor del thread solicitante.

Pregunta 2

Considere una máquina multi-core en la que no existe un núcleo de sistema operativo y por lo tanto no hay un scheduler de procesos. Implemente el mismo problema de la pregunta 1. Los nombres de las funciones que Ud. debe implementar son *MutexInit*, *lock* y *unlock*.

***Restricciones*:** La única forma válida de garantizar la exclusión mutua es usando un spin-lock. Para esperar hasta que se otorgue el mutex debe recurrir a un spin-lock inicialmente cerrado. Otras formas de *busy-waiting* no están permitidas.

Ayuda: Use la misma cola de prioridades de la pregunta 1 para registrar los cores en espera. La constante NCORES indica la cantidad de cores disponibles. La función *coreId( )* entrega un identificador entero (entre 0 y NCORES-1) del core que la invoca. Como no hay descriptor de procesos, use un arreglo de tamaño NCORES para almacenar el valor *last_serial* de cada core. Ud. puede probar su solución usando los mismos spin-locks de la tarea 5, incluidos en el archivo *test.c*, que viene en los archivos adjuntos.

Pregunta 3

A. Un usuario sostiene que usar procesos pesados para los comandos de Unix es ineficiente porque al lanzar los comandos con *fork*, hay que duplicar toda la memoria utilizada por el shell de comandos. Recomienda reimplementar los comandos de Unix usando threads. ¿Es así de ineficiente fork? ¿Y cuál sería la desventaja de implementar los comandos mediante threads?

B. El mismo usuario sostiene que Unix es lento porque al lanzar una aplicación hay que esperar a que otra aplicación termine para que se libere un core y así poder ejecutar la aplicación lanzada. ¿Qué le respondería Ud.?

C. Si tuviese que programar desde cero un núcleo de sistema operativo para máquinas mono-core con poca memoria, ¿elegiría un núcleo clásico o un núcleo moderno? Explique la razón. ¿Por qué cambiaría su decisión si se tratase de máquinas multi-core?

D. Un programador propone hackear un sistema Unix de la siguiente manera. El sabe en qué dirección de la memoria está el vector de interrupciones. Entonces él propone cambiar la función que ejecuta las llamadas al sistema por su propia función. Explique si puede o no lograr hackear la máquina.