CC4302 Sistemas Operativos – Control 2 – Semestre Otoño 2015 – Prof.: Luis Mateu

Pregunta 1

Parte a.- (2 puntos) Suponga que Ud. implementa un núcleo clásico de Unix para una máquina octacore.

i. Explique por qué no tiene sentido usar la variable global *current_process*. ¿Qué mecanismo usaría entonces para obtener el proceso en ejecución y cómo lo implementaría?

ii. Explique los 2 mecanismos que usaría para garantizar la exclusión mutua de la cola de procesos ready.

Parte b.- (4 puntos) Se desea agregar a nSystem un sistema de mensajes *asíncronos*. Esto significa que el emisor de un mensaje no se bloquea a la espera de una repuesta. Este sigue trabajando y puede emitir nuevos mensajes. Los mensajes se encolan en la tarea destinataria hasta que ésta los obtenga. Concretamente las operaciones que Ud. debe programar son:

- `void nPost(nTask t, void *msg)`: Emite el mensaje *msg* con la tarea *t* como destinataria. Este procedimiento encola *msg* en *t* y retorna de inmediato.
- `void* nGetMsg( )`: Lo invoca una tarea para recibir un mensaje pendiente extraído de su cola de mensajes. Los mensajes se entregan en orden FIFO. Si no hay mensajes encolados, la tarea se bloquea a la espera de que otra tarea invoque *nPost*.

Implemente 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. No olvide indicar las modificaciones que Ud. haga al descriptor de tareas de nSystem.

Nótese que no hay límite a la cantidad de mensajes que se pueden emitir. Es responsabilidad del usuario de este sistema de mensajes: (i) evitar agotar la memoria emitiendo mensajes indiscriminadamente sin que sean recibidos, y (ii) evitar liberar la memoria del mensaje antes de que el receptor termine de usar el mensaje.

Pregunta 2

Parte I. (1,5 puntos) Suponga que Ud. implementa un núcleo moderno de Unix para una máquina multicore. Las contrapartes de los procesos en el núcleo (los *peers*) necesitan consultar y modificar un diccionario, y leer o escribir en un disco. Discuta qué herramienta de sincronización usaría para garantizar la exclusión mutua (a) del diccionario, implementado mediante una tabla de hashing, y (b) del disco, considerando que cada acceso al disco toma unos 10 milisegundos. Elija para cada caso entre un *spin-lock* o un semáforo como el que se implementó en clases de cátedra para un núcleo moderno de una máquina multicore. Justifique sus 2 decisiones en términos de la eficiencia en tiempo de CPU y el mejor aprovechamiento del recurso CPU.

Parte II. (4,5 puntos) Considere 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*. Programe la siguiente API para el envío de mensajes asíncronos entre los distintos cores:

- `void post(int core, void *msg)`: Emite el mensaje *msg* con *core* como destinatario (un entero entre 0 y 7). Este procedimiento encola *msg* y retorna de inmediato.
- `void* getMsg( )`: Lo invoca un core para recibir un mensaje pendiente extraído de su cola de mensajes. Los mensajes se entregan en orden FIFO. Si no hay mensajes encolados, el core se bloquea a la espera de que otro core invoque *post*.

La funcionalidad pedida es equivalente al sistema de mensajes de la pregunta 1. Para programar su solución Ud. dispone de *spin-locks*, colas fifo (*fifoqueues*) y la función *coreId()* que entrega el número del core que la invoca. Use un arreglo de 8 colas fifo para almacenar los mensajes destinados a cada core.

Restricciones: Dado que no hay un núcleo de sistema operativo Ud. no dispone de monitores o semáforos y tampoco puede usar funciones como *Resume*, *PutTask*, etc. La única forma válida de esperar la llegada de un mensaje es utilizando un *spin-lock*. Otras formas de *busy-waiting* no están permitidas porque causarían mucha contención en el bus de datos.