CC4302 Sistemas Operativos

Examen – Semestre Primavera 2013 – Prof.: Luis Mateu

Pregunta 1

En un parque de diversiones la atracción principal es el tagada. Los turistas son representados por múltiples threads que ejecutan el código de más abajo a la izquierda. Cuando un turista desea divertirse en el tagada invoca **`subirAlTagada`** con su nombre. El tagada funciona llamando a **`lanzarTagada`** que recibe un arreglo de nombres de turistas y su tamaño. Este procedimiento recibe un máximo de 10 turistas y tarda unos 3 minutos, tiempo durante el cual los turistas mencionados se divierten en el tagada.

La implementación actual de **subirAlTagada** aparece abajo a la derecha y sólo atiende a un turista a la vez, lo cual es ineficiente.

```
void turista(char *nombre) {
  ...
  subirAlTagada(nombre);
  ...
}
```

```
monitor ctrl;
void subirAlTagada(char *nombre) {
  char *grupo[10];
  grupo[0]= nombre;
  nEnter(ctrl);
  lanzarTagada(grupo, 1);
  nExit(ctrl);
}
```

Reprograme el procedimiento **subirAlTagada** de modo que el tagada se use eficientemente de acuerdo a la siguiente estrategia. El tagada está detenido si no hay turistas. Cuando llega un turista, Ud. lanza el tagada de inmediato con el turista que llegó. Al terminar puede ocurrir que (i) no hay turistas, en cuyo caso el tagada permanece detenido, o bien (ii) hay turistas esperando subirse al tagada. Para (ii) Ud. debe subir tantos turistas como pueda hasta un máximo de 10 y relanzar el tagada. Los demás turistas si los hay permanecen en espera. La atención de los turistas debe realizarse por orden de llegada. La sincronización debe lograrse usando un monitor de nSystem. El procedimiento **`lanzarTagada`** es dado.

Hint: Use un thread adicional que se encargue de operar el tagada. Además utilice una cola global de tipo `FifoQueue` para almacenar requerimientos de uso del tagada.

Pregunta 2

Parte a.- Se desea agregar a nSystem un sistema de casillas de mensajes *asíncronos*, llamadas nBox. La API para este sistema es la siguiente:

- `nBox nMakeBox( )`: Crea una casilla que puede almacenar un número ilimitado de mensajes.
- `void nPost(nBox box, void *msg)`: Deposita el mensaje `msg` en la casilla `box`. El mensaje es asíncrono, es decir `nPost` encola `msg` en `box` y retorna de inmediato. Si hay tareas esperando este mensaje, la tarea que invocó `nGetMsg` primero lo extrae y se desbloquea, el resto sigue esperando.
- `void* nGetMsg(nBox box)`: Extrae y entrega un mensaje de la casilla `box`. Si no hay mensajes encolados, la tarea se bloquea hasta poder extraer un mensaje.

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.

Parte b.- Considere una máquina con 8 cores físicos sin un núcleo de sistema operativo, y por lo tanto no hay *scheduler* de procesos y tampoco threads. Los *spin-locks* son la única herramienta de sincronización disponible. Los 8 cores ejecutan código en paralelo y comparten la memoria. Programe la misma API de la *parte a* para esta máquina. Dado que no hay una cola de procesos *ready*, para esperar no le queda otra que hacer *busy-waiting*.

Pregunta 3

i. ¿Qué herramienta de sincronización usaría para garantizar la exclusión mutua al acceder a la cola de procesos "ready" y por qué? Considere 2 sistemas: un núcleo clásico de Unix en una máquina con un solo core y núcleo moderno en una máquina multi-core.

ii. ¿En qué tipo de sistema el shell de comandos es más eficiente? ¿En un sistema que usa segmentación o en uno basado en paginamiento? Explique por qué.

iii. ¿Qué estrategia de paginamiento se comportará mejor? ¿La estrategia del reloj o la del *working set*? Explique considerando las siguientes situaciones: 1) Un solo proceso, hay memoria suficiente; 2) Un solo proceso, hay penuria de memoria; 3) Muchos procesos, hay memoria suficiente; 4) Muchos procesos, hay penuria de memoria.

iv. Considere que en un instante dado un disco se encuentra leyendo el bloque 305 y anteriormente los procesos P1, P2, P3 y P4 pidieron leer los bloques 207, 420, 348 y 958, respectivamente. En qué orden se harán las lecturas de P1 a P4 cuando la estrategia es 1) shortest seek first, 2) método del ascensor. Además explique si habrá una diferencia en el tiempo total de lectura si se está usando (a) un disco tradicional o (b) un SSD (*solid state drive*).

v. Programe la función *copy_to_user* que Ud. usó en la tarea 3 para copiar un trozo de memoria del núcleo en un trozo de memoria del area del usuario. Tenga presente que: se trata de un núcleo moderno, el núcleo debe ser seguro, un *segmentation fault* se traduce en un *system panic*. Ud. tiene acceso al descriptor del proceso en donde puede encontrar la información adicional que necesite. El encabezado de la función es:

```
int copy_to_user(char *to, char *from, ssize_t count);
```

La función entrega 0 si tuvo éxito o -1 en caso de error.