CC4302 Sistemas Operativos

Control 2 – Semestre Primavera 2013

Prof.: Luis Mateu

Pregunta 1

Se desea agregar un sistema de *mutex* con prioridades a nSystem como herramienta de sincronización nativa. Concretamente la API de este sistema es:

- *nMutex nMakePriMutex( )*: Entrega un mutex con prioridades. En todo instante puede existir a lo más un solo propietario de este mutex.
- *void nLock(nMutex mtx, int pri)*: Solicita la propiedad de *mtx* con prioridad *pri* (un valor entero entre 0 y 3). Si *mtx* está libre, el solicitante adquiere su propiedad de inmediato y *nLock* retorna. Si actualmente otra tarea tiene la propiedad de *mtx*, el solicitante espera hasta adquirir en forma exclusiva *mtx* cuando otra tarea invoque *nUnlock*.
- *void nUnlock(nMutex mtx)*: Libera *mtx*. Si existen varias tareas esperando adquirir *mtx*, se entrega su propiedad a la tarea que lo haya solicitado con la prioridad más alta (3 es la más alta, 0 la menor).

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.

Pregunta 2

La siguiente es una implementación secuencial del juego de la vida.

```
int **newMat, **mat; /* la matriz de células: la nueva versión y la versión previa */
int n, k;            /* tamaño de la matriz y número de matrices a calcular */
void lifeGame() {
  int i, j, step;
  for (step= 0; step<k; step++) { /* ciclo externo: k iteraciones */
    for (i= 0; i<n; i++)
      for (j= 0; j<n; j++)
        newMat[i][j]= compute(mat, i, j); /* dado */
    swap(&mat, &newMat);                  /* dado */
  }
}
```

El programa parte con una Matriz M (*mat* en el programa) y calcula las matrices M_0, M_1, M_2, ..., M_{k-1}. Cada matriz es de n x n. El valor de cada elemento de la matriz M_{step} depende únicamente de M_{step-1}. Este hecho simplifica la paralelización de la función. El procedimiento *compute* es dado y su implementación no es relevante en esta pregunta.

Programe una versión paralela de la función *lifeGame* para una máquina *octa-core* en la que no existe un núcleo de sistema operativo y en donde la única herramienta de sincronización es el *spin-lock*. Concretamente la función que Ud. debe programar posee el siguiente encabezado:

```
void parLifeGame(int coreId);
```

Suponga que esta función se invoca una y solo una vez en todos los cores casi al mismo tiempo. El parámetro *coreId* es un número entre 0 y 7 que identifica el core en donde se hace la llamada de *parLifeGame*.

Notas:

- En su implementación necesitará usar variables globales adicionales inicializadas adecuadamente.
- Haga que cada core se encargue de calcular una submatriz de *newMat* de n/8 filas.
- En cada instante todos los cores deben estar calculando *newMat* para el mismo *step*.
- Si lo desea puede recurrir a las *fifoqueues* pero recuerde que si sus operaciones se realizan en paralelo pueden ocurrir dataraces, es decir no implementa ningún tipo de sincronización.
- En caso de tener que esperar no se puede retomar otro proceso porque no hay un núcleo de sistema operativo y por lo tanto *sí es válido* recurrir a *busy-waiting*.