CC41B : Sistemas Operativos
Control 2–Semestre Primavera'2002

Prof.: Luis Mateu.

Sin apuntes, 1 hora 30 minutos

Pregunta 1

Un programador ha implementado un par de procedimientos que sirven para sincronizar threads que corren en modo sistema, dentro de un núcleo *multi-threaded* para multiprocesadores. Varios threads pueden esperar hasta que ocurra un cierto evento invocando el procedimiento `kWait()`. Un único thread notifica la ocurrencia del evento invocando el procedimiento `kNotify()`. Si un thread invoca `kWait()` y el evento ya ocurrió, el thread continúa de inmediato (sin esperar). El programador implementó estos procedimientos usando *spin-locks*, de la siguiente manera:

```
int event_spinLock= CLOSED; /* spin-lock cerrado */
void kWait() {                 |  void kNotify() {
  spinLock(&event_spinLock);   |    /* abrir el spin-lock */
  spinUnlock(&event_spinLock); |    spinUnlock(&event_spinLock);
}                              |  }
```

- Parte a.- Discuta si esta solución es correcta desde un punto de vista funcional (es decir un thread nunca va a retornar de `kWait` antes de la invocación de `kNotify` y ningún thread se quedará esperando indefinidamente después de la invocación de `kNotify`).
- Parte b.- Critique esta solución desde un punto de vista de la eficiencia. Considere que la notificación del evento puede ocurrir segundos o minutos después de la invocación de `kWait`.
- Parte c.- Reprograme eficientemente ambos procedimientos (vea en el torpedo, al final de este enunciado, los procedimientos de bajo nivel disponibles en un núcleo multi-threaded).

Pregunta 2

Se tiene una versión minimal de nSystem en donde se han eliminado los semáforos, mensajes y cualquier otra herramienta de sincronización. En su reemplazo se le pide a Ud. programar un sistema de *pipes* para nSystem, el que permite enviar caracteres de una tarea a otra. Por simplicidad, asociado a cada pipe existe solo una tarea (el emisor) que puede enviar caracteres por ese pipe y solo una tarea (el receptor) que puede recibir caracteres de ese pipe. Los procedimientos son los siguientes:

- `nPipe nMakePipe(int tamano)`: Crea y entrega un pipe con un buffer interno de `tamano` caracteres.

- `void nWritePipe(nPipe pipe, char c)` : Envía el caracter `c` por `pipe`. Almacena el caracter `c` en el buffer interno y retorna de inmediato. Este procedimiento se bloquea si y solo si el buffer está lleno, en cuyo caso se espera hasta que el receptor extraiga un caracter con `nReadPipe`.
- `char nReadPipe(nPipe pipe)` : Recibe un caracter de `pipe`. Se extrae el caracter más antiguo en el buffer y se entrega como resultado de la llamada. Este procedimiento se bloquea si y solo si el buffer está vacío, en cuyo caso se espera hasta que el emisor deposite un caracter con `nWritePipe`.

El siguiente ejemplo muestra al emisor y al receptor en acción:

```
void Emisor(nPipe pipe) {
  for(;;) {
    char c= ...;
    nWritePipe(pipe, c);
  }
}
```

```
void Receptor(nPipe pipe) {
  for(;;) {
    char c= nReadPipe(pipe);
    ...
  }
}
```

Implemente este sistema de pipes para nSystem. Se recomienda representar un pipe mediante una estructura de datos con los siguientes campos: (i) un arreglo de caracteres para representar el buffer interno, (ii) el tamaño del buffer, (iii) la cantidad de caracteres almacenados en el buffer en un instante dado, (iv) la posición del caracter más antiguo del buffer, (v) la posición en donde almacenar el próximo caracter depositado, (vi) dos valores booleanos que indiquen si el emisor o el receptor se encuentran en espera, y (vii) dos identificadores de tareas para el emisor y el receptor.

Torpedo:

```
nucleo multi-threaded
---------------------
kTask currentTask();
Queue ready_queue;
int ready_queue_spinLock;

void kPutTask(Queue queue,
              kTask t);
void kPushTask(Queue queue,
               kTask t);
kTask kGetTask(Queue queue);
int kEmptyQueue(Queue queue);
Queue kMakeQueue;
void kResume();
```

```
nSystem
-------
nTask current_task;
Queue ready_queue;
void START_CRITICAL();
void END_CRITICAL();
void PutTask(Queue queue,
             nTask t);
void PushTask(Queue queue,
              nTask t);
nTask GetTask(Queue queue);
int EmptyQueue(Queue queue);
Queue MakeQueue();
void Resume();
```