AUX 5 – CC41B

Prof.: Luis Mateu Aux.: Juan Manuel Barrios

5 de Octubre 2007

Pregunta 1 (nPipe)
Implemente en nSystem la siguiente API para un pipe concurrente (nPipe):

`nPipe nMakePipe(int buffer_size)` Crea un nPipe con un buffer interno de tamaño buffer_size.

`char nGetPipe(nPipe pipe)` Recupera un carácter desde el buffer. En caso que no hayan caracteres disponibles se debe bloquear hasta recibir uno.

`void nPutPipe(nPipe pipe, char c)` Agrega al buffer un carácter. En caso que hayan tareas esperando, debe despertar a alguna. En caso que el buffer esté lleno debe esperar hasta que haya un espacio disponible.

`void nClosePipe(nPipe pipe)` Destruye y libera la memoria del nPipe.

Como simplificación, no es necesario que el pipe maneje un estado de abierto/cerrado. El programa cliente esperará por productores y consumidores antes de invocar a `nClosePipe`. Utilice los procedimientos de bajo nivel disponibles en 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();
```

Pregunta 2 (P1, Control 2, primavera 2002)
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 puede 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;
void kWait() {
  spinLock(&event_spinLock);
  spinUnlock(&event_spinLock);
}
```

```
void kNotify() {
  /* abrir el spin-lock */
  spinUnlock(&event_spinLock);
}
```

Parte a.- Discuta si esta solución es correcta desde un pinto 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 usando los procedimientos de bajo nivel disponibles en un núcleo multi-threaded:

```
Queue kMakeQueue();
void kPutProc(Queue queue, kProc p);
void kResume();
void kPushProc(Queue queue, kProc p);
Queue ready_queue;
kProc kGetProc(Queue queue);
int ready_queue_spinLock;
int kEmptyQueue(Queue queue);
kProc currentProc();
```