CC41B Sistemas Operativos
Control 2 – Semestre Otoño 2012
Prof.: Luis Mateu

Pregunta 1

Se desea agregar a nSystem *buffers* en forma nativa. Esta estructura se usará para sincronizar múltiples productores y consumidores. La API para manipular estos buffers es la siguiente:

Procedimiento	***Significado***
`nBuffer nMakeBuffer(int size);`	Crea y entrega un buffer de tamaño `size`
`void nPut(nBuffer b, void* p);`	Deposita el item `p` en el buffer `b`
`void* nGet(nBuffer b);`	Extrae un item del buffer `b`

Observe que en el buffer se depositan y extraen punteros opacos, es decir están declarados como `void*`. Esto significa que la aplicación puede depositar ahí punteros a objetos de cualquier tipo.

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 definir la estructura `nBuffer`.

Pregunta 2

Parte a.- (1 punto)

Discuta qué herramienta de sincronización usaría para garantizar la exclusión mutua al acceder a la cola de procesos *ready* en el núcleo de Unix. En su discusión considere monitores, semáforos y *spin-locks,* y además si se trata de un núcleo clásico o un núcleo moderno.

Parte b.- (1 punto)

Suponga que una implementación de Unix no verifica en la llamada a sistema *read* la validez del buffer pasado como argumento. Explique cómo usaría Ud. este defecto en su favor para lograr ejecutar su propio código en modo sistema (sin ser *root*).

Parte c.- (4 puntos)

Un multiprocesador dispone de 5 *cores* físicos. En él no existe un núcleo de sistema operativo y la única herramienta de sincronización disponible es el *spin-lock*. Cada *core* corre una función que corresponde a un filósofo, es decir alterna entre comer y pensar:

```
void filosofo(int i) {
  for (;;) {
    comer(i, (i+1)%5);
    pensar();
  }
}
```

Se le pide a Ud. modificar este código para resolver el problema de sincronización, es decir (i) evitar que 2 filósofos coman simultáneamente con el mismo palito y (ii) que no haya deadlock. Además se le pide maximizar las oportunidades de comer en paralelo y por lo tanto si un filósofo no puede comer porque le falta un solo palito, esto no debe impedir que el filósofo adyacente use el otro palito. Para respetar este requisito es inevitable que los filósofos de su solución sufran hambruna.

Dado que no hay núcleo de sistema operativo:

- No existe un scheduler de procesos: no puede llamar a `Resume`.
- no existe manejo de colas: no puede usar `PutTask`, `GetTask`
- no hay herramientas de sincronización avanzadas como monitores, semáforos o mensajes.

Ud. debe resolver el problema usando únicamente los *spin-locks*. Por lo tanto cuando un filósofo debe esperar a que ambos palitos estén libres, no tiene otra alternativa que hacer *busy-waiting*.