CC41B Sistemas Operativos
Control 2 – Semestre Primavera 2009
Prof.: Luis Mateu

Pregunta 1

El siguiente código implementa un semáforo global en base a *spin-locks*. Este semáforo contiene 0 tickets inicialmente.

```
        int tickets= 0; /* contador de tickets */
    int mutex= OPEN, barrera= CLOSED; /* spin-locks */
```

```
void semSignal() {
  if (tickets==0)
    spinUnlock(&barrera);
  spinLock(&mutex);
  tickets++;
  spinUnlock(&mutex);
}
```

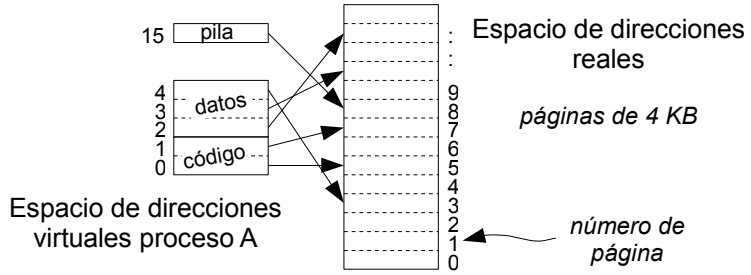
```
void semWait() {
  spinLock(&barrera);
  spinLock(&mutex);
  tickets--;
  spinUnlock(&mutex);
  if (tickets>0)
    spinUnlock(&barrera);
}
```

(a) Pruebe con un diagrama de threads que esta implementación es incorrecta mostrando una situación en que un thread espera con `semWait` indefinidamente en un semáforo que contiene un ticket.

(b) Reescriba el código de más arriba de modo que se obtenga una implementación de semáforos correcta. Haga mínimos cambios. Mantenga el espíritu de la solución usando solo *spin-locks*.

(c) Explique bajo qué condiciones esta implementación sería más eficiente que una implementación tradicional (como la de nSystem por ejemplo) y cuando sería "terriblemente" ineficiente.

(d) La siguiente figura muestra la asignación de páginas de un proceso A que se ejecuta en un sistema Unix.



i. Haga la tabla de páginas para el proceso A indicando en la tabla: número de página virtual, número de página real y atributos de validez y escritura (V y W).

ii. Suponga que el proceso A invoca **fork**. Haga las tablas de páginas para el hijo y el padre justo después del **fork** considerando que el sistema operativo implementa **fork** usando la técnica *copy-on-write*.

(e) Rehaga la figura de la parte (d) justo después del fork considerando un sistema que implementa segmentación. No haga las tablas de segmentos.

Pregunta 2

Se desea agregar a nSystem una nueva primitiva de sincronización: los *left/right locks*. Este es un tipo de candado (*lock*) que se puede otorgar por mitades. La API es la siguiente:

- `nLRLock nMakeLeftRightLock( )`: Construye y entrega un nuevo *left/right lock*.
- `int nHalfLock(nLRLock l)`: solicita un medio candado de `l`. Se bloquea hasta que esté disponible el lado izquierdo o el derecho. Entrega el lado otorgado: LEFT o RIGHT.
- `int nHalfUnlock(nLRLock l, int side)`: devuelve el medio candado `side` de `l` otorgado previamente por `nHalfLock`.
- `void nFullLock(nLRLock l)`: solicita el candado `l` completo. Se bloquea hasta que los lados izquierdo y derecho estén disponibles.
- `void nFullUnlock(nLRLock l)`: devuelve el candado `l` completo otorgado previamente por `nFullLock`.

Cuando un thread posee el candado completo, ningún otro thread puede obtener el mismo candado ni una de sus mitades hasta que se devuelva el candado. Cuando un thread posee una mitad del candado, un segundo thread puede obtener la otra mitad, pero ningún thread podrá obtener el candado completo hasta que ambas mitades estén libres. El otorgamiento del candado debe ser *fair* (sin hambruna) y eficiente, es decir se debe otorgar simultáneamente ambas mitades del candado, cuando esto no interfiere con el requisito de *fairness*.

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. Suponga que los threads no hacen trampas: nunca devuelven un medio candado o candado completo que no fue solicitado y otorgado previamente. Un thread que pide el candado completo, no lo devuelve por mitades. Un mismo thread no pide el candado completo por mitades.