CC4302 Sistemas Operativos

Examen – Semestre Otoño 2013 – Prof.: Luis Mateu

Pregunta 1

Una fábrica de automóviles tiene 8 sectores para una cadena de ensamble de 8 etapas. La siguiente función se encarga de ensamblar secuencialmente n automóviles en esta fábrica:

```
typedef void (*Etapa)(Auto *a);
Etapa etapas[8]= {etapa0, etapa1, ..., etapa7}; // ref. A
void ensamblar(Auto *autos, int n) {
  int i, k;
  for (i=0; i<n; i++) {
    Auto *a= &autos[i];
    for (k= 0; k<8; k++) {
      avanzar(a, k);     // B
      (*etapas[k])(a);   // C
    }
    estacionar(a);       // D
  }
}
```

Un auto *a* se ensambla invocando sucesivamente las funciones *etapa0*, *etapa1*, etc., las que son dadas (ver referencias A y C). Para realizar la etapa *k*, se necesita que el esqueleto del auto se encuentre en el sector *k*. Para hacer avanzar el esqueleto de *a* del sector *k-1* a *k* se debe invocar *avanzar(a, k)* (ver B). El caso *k=0* corresponde a la entrada del chasis de *a* a la fábrica. Cuando un auto está en el sector 7 ya ensamblado, su salida se logra invocando *estacionar(a)* (ver D).

Reescriba la función *ensamblar* de modo que las 8 etapas se realicen en paralelo usando 8 threads. La etapa 0 del auto *i* se debe realizar al mismo tiempo que la etapa 1 del auto *i-1* y que la etapa 2 del auto *i-2*, etc. Las funciones más costosas en términos de tiempo son *etapa0*, *etapa1*, etc., seguido de *avanzar* y *estacionar*.

Restricciones adicionales: Las etapas se invocan en orden y secuencialmente. Es decir que solo puede realizar la etapa *k* del auto *a* una vez que la etapa *k-1* de *a* concluyó. En cada sector cabe un solo auto de modo que Ud. no puede avanzar el auto *i* al sector *k*, si no hizo avanzar primero el auto *i-1* al sector *k+1*. Use los threads de nSystem. Para la sincronización use un solo monitor.

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*.

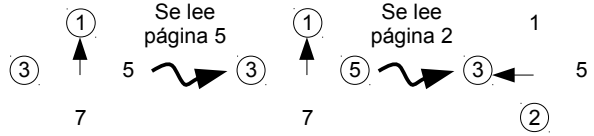
Implemente esta API usando los procedimientos de bajo nivel de nSystem (START_CRITICAL, Resume, PutTask, etc.). Ud. *no puede usar* otros mecanismos de sincronización ya disponibles en nSystem, como semáforos, monitores, mensajes, etc. Suponga que los threads no hacen trampas: por ejemplo nunca devuelven un medio candado o candado completo que no fue solicitado y otorgado previamente.

Pregunta 3

a) Explique si tiene o no sentido usar la estrategia del *working set* en un sistema operativo que ofrece solo procesos livianos. ¿Cuál otra estrategia sería más eficaz y por qué?

b) Compare las estrategias de scheduling de disco *first come first served*, *shortest seek first* y método del ascensor cuando (i) hay un solo proceso intensivo en E/S, (ii) cuando hay 2 procesos intensivos en E/S, y (iii) varios procesos intensivos en E/S. En su comparación considere como criterios el tiempo promedio de acceso al disco y si puede ocurrir hambruna o no.

c) La siguiente figura muestra los estados sucesivos de la memoria al realizar 2 accesos en un sistema de memoria virtual que usa la estrategia del reloj para el reemplazo de páginas.



Las páginas que tienen el bit de referencia en 1 aparecen encerradas en una circunferencia. La flecha indica la posición del puntero de la estrategia del reloj. Siguiendo el mismo esquema de la figura, muestre todos los estados por los que pasa la memoria al realizar los siguientes accesos:

5 7 2 4 2 3

d) ¿Cuantos *spin-locks* se necesitan en un núcleo clásico de Unix para una máquina multi-core? ¿Y si es mono-core?