CC4302 Sistemas Operativos

Control 1 – Semestre Otoño 2014 – Prof.: Luis Mateu

Pregunta 1

Un *left/right lock* es un mutex que se puede otorgar completo o por mitades. Solo un thread puede obtener el mutex completo llamando a *fullLock*. Posteriormente, ese thread devuelve el mutex con *fullUnlock*. Hasta 2 threads pueden obtener cada uno una mitad distinta del mutex por medio de *halfLock*, que entrega LEFT si se otorgó la mitad izquierda o RIGHT si fue la mitad derecha. Más tarde cada thread devuelve su respectiva mitad con *halfUnlock*, especificando cuál fue la mitad que se le había otorgado. La siguiente implementación es incorrecta e ineficiente, pero funciona casi siempre:

```
enum { LEFT= 0, RIGHT= 1 };
int busy[2]= { FALSE, FALSE }; /* ambas mitades libres */
nSem m; /* = nMakeSem(1); parte con 1 ticket */
```

```
int halfLock() {
  if (!busy[LEFT] &&
      !busy[RIGHT])
    nWaitSem(m);
  int l= -1;
  while (l==-1) {
    if (!busy[LEFT])
      l= LEFT;
    else if (!busy[RIGHT])
      l= RIGHT;
  } /* ¡busy waiting! */
  busy[l]= TRUE;
  return l;
}
```

```
void halfUnlock(int l) {
  busy[l]= FALSE;
  if (!busy[LEFT] &&
      !busy[RIGHT])
    nSignalSem(m);
}
void fullLock() {
  nWaitSem(m);
}
void fullUnlock() {
  nSignalSem(m);
}
```

a) (1.5 puntos) Haga un diagrama de threads que muestre que estando el mutex completamente libre, puede ocurrir que 2 threads invoquen concurrentemente *halfLock* pero solo el primero obtenga una mitad. El segundo espera hasta que el primero invoque *halfUnlock*.

b) (1.5 puntos) Haga un diagrama de threads que muestre que se puede otorgar erróneamente medio mutex y el mutex completo simultáneamente. *Hint:* habiendo un thread obtenido una mitad del mutex con *halfLock*, cuando éste invoca *halfUnlock* concurrentemente con un 2do. thread que invoca *halfLock*, el semáforo *m* puede quedar con un 1 ticket. Así un 3er. thread puede obtener el mutex completo con *fullLock*, mientras que el 2do. thread ya posee una mitad.

c) (3 puntos) Reeimplemente correctamente y sin *busy-waiting* las funciones *halfLock* y *halfUnlock*, ***sin modificar las funciones fullLock y fullUnlock***. Para ello preserve la idea de la solución de más arriba, manteniendo el semáforo *m*, pero agregando 2 nuevos semáforos de nSystem. El primer semáforo para lograr la exclusión mutua dentro de *halfLock* y dentro de *halfUnlock*. El segundo semáforo mantiene hasta 2 tickets, uno por cada mitad libre del mutex. No se preopupe por la hambruna.

Pregunta 2

La empresa SoTraCon se dedica al transporte de contenedores. Los clientes de SoTraCon son tareas de nSystem que invocan la función *transportar* para solicitar el transporte del contenedor *cont* desde la ciudad *orig* hasta la ciudad *dest*. La implementación actual de esta función usa un semáforo para coordinar el uso del único camión que posee la empresa:

```
nSem mutex; /* = nMakeSem(1); */
Camion *c;  /* el unico camion de SoTraCon */
Ciudad *ubic= &Santiago; /* su ubicacion actual */
void transportar(Contenedor *cont,
                 Ciudad *orig, Ciudad *dest) {
  nWaitSem(mutex);
  conducir(c, ubic, orig);
  cargar(c, cont);
  conducir(c, orig, dest);
  descargar(c, cont);
  ubic= dest; /* se actualiza la ubicacion del camion */
  nSignalSem(mutex);
}
```

Las funciones *conducir*, *cargar* y *descargar* son dadas y toman mucho tiempo. Observe que *transportar* espera cuando el único camión de la empresa está siendo ocupado por otra llamada concurrente de *transportar*. Antes de cargar el camión con el contenedor hay que conducir el camión desde su ubicación actual a la ciudad de origen del contenedor. La función solo retorna una vez que el contenedor se transportó y descargó en su ciudad de destino.

SoTraCon acaba de aumentar su flota a 8 camiones lo que le permite transportar hasta 8 contenedores en paralelo. Se le pide a Ud. *reprogramar la función transportar* para que se usen eficientemente estos 8 camiones. Ud. dispone de:

```
#define P 8
Camion *camiones[P];
double distancia(Ciudad *orig, Ciudad *dest);
```

Inicialmente todos los camiones están en Santiago. Agregue otras variables globales y programe una función de inicialización. Por simplicidad los transportes pendientes pueden ser atendidos en cualquier orden.

Restricciones:

- Un camión puede ser usado para un solo transporte a la vez.
- Un camión no puede permanecer ocioso si existe un transporte pendiente.
- Si en el momento de invocar *transportar* hay varios camiones ociosos, por razones de eficiencia Ud. debe elegir el camión cuya ubicación actual sea la más cercana a la ciudad de origen del contenedor. Use la función *distancia* para elegir el camión más cercano.
- Resuelva el problema de sincronización usando los monitores de nSystem.