CC4302 Sistemas Operativos – Control 1 – Semestre Primavera 2017 – Prof.: Luis Mateu

Pregunta 1

Se ha especificado el siguiente sistema de canales de comunicación para las tareas de nSystem. Una tarea envía un mensaje por medio de un canal usando la función *enviar*. Otra tarea recibe el mensaje del canal usando la función *recibir*. Por simplicidad un mensaje es siempre un número entero. La tarea que envía *debe esperar hasta que su mensaje sea recibido*. En el siguiente ejemplo de uso la tarea receptora debe desplegar 0 1 2 3 4:

```
Canal *c= nuevoCanal(); // compartido por ambas tareas
```

Emisor	*Receptor*
`for (int i= 0; i<5; i++)` `  enviar(c, i);`	`for (int j= 0; j<5; j++)` `  nPrintf("%d ", recibir(c));`

La siguiente es una implementación incorrecta de este sistema de mensajes:

```
typedef struct {
  nSem envSem, recSem;
  int msg;
} Canal;

Canal *nuevoCanal() {
  Canal *c= nMalloc(
          sizeof(Canal));
  c->envSem= nMakeSem(0);
  c->recSem= nMakeSem(0);
  return c;
}
```

```
void enviar(Canal *c, int msg){
  c->msg= msg;
  nSignalSem(c->envSem);
  nWaitSem(c->recSem);
}

int recibir(Canal *c) {
  nSignalSem(c->recSem);
  nWaitSem(c->envSem);
  return c->msg;
}
```

(a) Haga un *diagrama de threads* que muestre que esta implementación es incorrecta porque se podría perder un mensaje cuando múltiples tareas envían mensajes en paralelo por un mismo canal (1 punto).

(b) Haga un diagrama de threads que muestre que esta implementación podría no desplegar 0 1 2 3 4 con el ejemplo de uso de más arriba (2 puntos).

(c) Corrija esta implementación manteniendo la idea, es decir no reprograme desde cero. La solución debe basarse en semáforos únicamente. Necesitará un semáforo adicional. No olvide que el emisor debe esperar hasta que su mensaje sea recibido (3 puntos).

Pregunta 2

Una empresa se dedica al transporte de contenedores. Sus clientes 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 único camión
Ciudad *ubic= &Santiago;     // su ubicación 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 ubicación del camión
  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 cargó en la ciudad de origen, se transportó y se descargó en su ciudad de destino.

La empresa de transportes 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.