CC41B Sistemas Operativos
Control 1 – Semestre Otoño 2012
Prof.: Luis Mateu

Pregunta 1

Se ha especificado un sistema de mensajes alternativo al de nSystem. El sistema se diferencia en que los mensajes se envían indirectamente a través de un canal y no a un proceso. El receptor indica por qué canal espera recibir el mensaje. Por simplicidad un mensaje es siempre un número entero. Los mensajes no necesitan ser respondidos pero *el emisor debe esperar hasta que su mensaje sea recibido*. En el siguiente ejemplo, la tarea receptora debe desplegar 0 1 2 3 4:

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

Emisor	Receptor

`int i;`
`for (i= 0; i<5; i++)`
`  enviar(c, i);`
	`int j;`
	`for (j= 0; j<5; k++)`
	`  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= (Canal*)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 cuando varias tareas envían mensajes a un mismo canal.

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

(c) Corrija esta implementación de modo que funcione correctamente para el caso de varias tareas que envían y varias que reciben por el mismo canal. La solución debe basarse en semáforos únicamente. No olvide que el receptor debe esperar hasta que su mensaje sea recibido.

Pregunta 2

Se necesita buscar un dato `d` en un conjunto de `NLIB` libros. La información no se almacena en un orden específico por lo que en el peor caso habría que consultar todos los libros. Para acelerar la búsqueda se dispone de `NEMP` empleados que pueden trabajar simultáneamente. La siguiente implementación sirve como referencia de la funcionalidad pedida, pero ocupa un solo empleado:

```
Empleado *empls[];
Libro *libros[];

int buscar(Dato *d) {
  int i;
  for (i= 0; i< NLIB; i++) {
    /* Solo ocupa el primer empleado en empls */
    if (consultar(empls[0], libros[i], d)) {
      return TRUE; /* dato encontrado! */
    }
  }
  return FALSE; /* dato no encontrado */
}
```

El procedimiento `consultar(e, l, d)` ordena al empleado `e` que busque `d` en el libro `l` y sólo retorna una vez que el empleado tiene la respuesta, lo cual puede tomar poco o mucho tiempo. El procedimiento `consultar` y las estructuras `Empleado` y `Libro` son dados.

Se le pide a Ud. reescribir el procedimiento `buscar` de manera que los `NEMP` empleados trabajen en paralelo para buscar `d`. El procedimiento `buscar` es invocado varias veces, pero desde un único *thread*. Esto significa que Ud. puede usar variables globales en su solución.

Restricciones:

- Un libro no puede ser consultado simultáneamente por 2 empleados.
- Un empleado puede consultar un solo libro a la vez.
- Ningún empleado podrá iniciar una consulta a partir del momento en que otro empleado encuentra el dato buscado. Sin embargo Ud. sí debe esperar que las consultas ya iniciadas terminen.
- Un empleado no debe permanecer ocioso si hay un libro que puede ser consultado (si esto no interfiere con el punto anterior).
- Resuelva el problema de sincronización usando los monitores de nSystem, asegurándose con `nWaitTask` que todas las tareas terminaron adecuadamente antes del retorno de `buscar`.