CC41B : Sistemas Operativos
Control 1–Semestre Primavera'2001

Prof.: Luis Mateu.

Sin apuntes, 1 hora 30 minutos

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 de qué canal desea recibir el mensaje. Por simplicidad un mensaje es siempre un número entero. En el siguiente ejemplo, el proceso receptor debe desplegar 0 1 2 3 4 :

```
    Canal c= nuevoCanal(); /* compartido por ambos procesos */
/* Emisor: */          |  /* Receptor: */
int i;                 |  int j;
for (i=0; i<5; i++)    |  for (j=0; j<5; j++)
  enviar(c, i);        |    printf("%d ", recibir(c));
```

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

```
typedef struct {                        | void enviar(Canal c, int msg) {
  Lock envLock, recLock;                |   c->msg= msg;
  int msg;                              |   Unlock(c->envLock);
}                                       |   Lock(c->recLock);
  *Canal;                               | }
                                        |
Canal nuevoCanal() {                    | int recibir(Canal c) {
  Canal c= (Canal)malloc(sizeof(*c));   |   Unlock(c->recLock);
  c->envLock= makeLock();               |   Lock(c->envLock);
  c->recLock= makeLock();               |   return c->msg;
  lock(c->envLock); /* parte ocupado */ | }
  lock(c->recLock); /* parte ocupado */ |
  return c;                             |
}                                       |
```

- Parte a.- Haga un diagrama de tareas que muestre que esta implementación es incorrecta cuando varios procesos envían mensajes a un mismo canal.
- Parte b.- Haga un diagrama de tareas que muestre que esta implementación podría no desplegar 0 1 2 3 4 en el ejemplo de más arriba.
- Parte c.- Corrija esta implementación de modo que funcione correctamente para el caso de un solo proceso que envía y un solo proceso que recibe. La solución debe basarse en locks solamente.

Obs. : En este sistema, malloc y printf sí admiten llamadas concurrentes.

Pregunta 2

N tareas necesitan compartir un recurso único bajo nSystem. Cada tarea se indentifica con un número entero entre 0 y N-1. Para evitar que dos o más tareas usen simultáneamente el recurso, cada tarea solicita previamente el recurso, suministrando su identificación. La misma tarea informa cuando desocupa el recurso.

Cada tarea tiene la siguiente forma:

```
void tarea(int id) { /* id es la identificacion */
  for(;;) {
    solicitarRecurso(id);
    usarRecurso();
    devolverRecurso();
    hacerOtrasCosas();
  }
}
```

Implemente los procedimientos `solicitarRecurso` y `devolverRecurso` empleando la siguiente política de asignación. Supongamos que la tarea k tiene asignado el recurso y existen otras tareas que lo han solicitado y aún no lo obtienen. Cuando k devuelve el recurso, se debe respetar el siguiente orden de prioridad para elegir la siguiente tarea que obtendrá el recurso: k+1, k+2, ..., N-1, 0, 1, 2, k-1, descartando por supuesto aquellas tareas que no han solicitado el recurso.

Por ejemplo, si la tarea 5 devuelve el recurso, y éste ha sido solicitado por las tareas 2, 4, 9 y 11, el recurso se asignará a la tarea 9, sin importar en qué instante se hayan hecho las solicitudes. Las tareas 2, 4 y 11 deben esperar. Si a continuación el recurso es solicitado por la tarea 10 y posteriormente liberado (por 9 evidentemente) entonces el recurso será asignado a 10. Más tarde, de no haber otras tareas que soliciten el recurso, éste deberá ser asignado en el siguiente orden: 11, 2, 4.

Restricción: Ud debe usar las tareas y mensajes de nSystem para implementar estos procedimientos. *No puede* usar locks, semáforos o monitores.