GUIA - MENSAJES

Sistemas Operativos – Primavera 2018

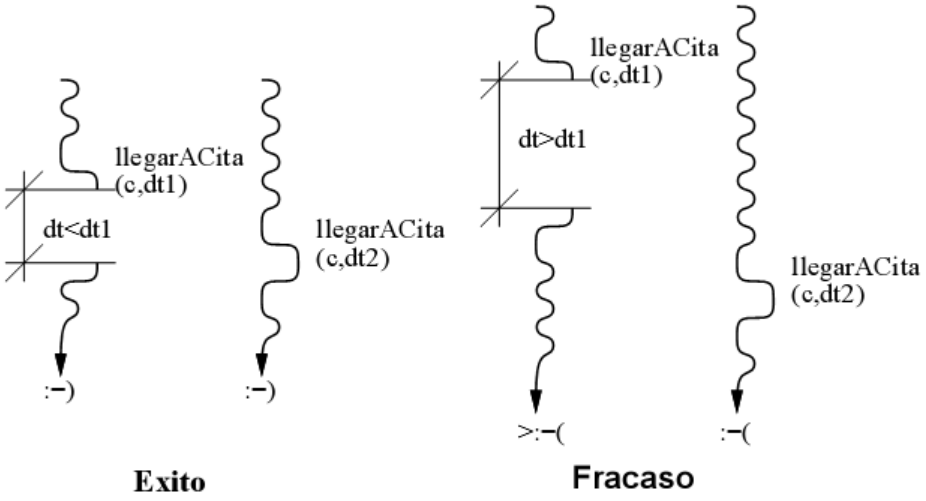
Profesores: Luis Gajardo

Objetivo: Solucionar problemas de sincronización de procesos mediante mecanismos de paso de mensajes.

1. Se desea implementar una nueva herramienta de sincronización entre tareas que sirve para realizar citas. Se han especificado dos operaciones:

   - `Cita programarCita()`
   - `int llegarACita(Cita cita, int espera)`

   La operación *`programarCita`* crea un objeto coordinador para la cita entre dos tareas. Sólo se conocerá cuáles son estas tareas a medida que lleguen a la cita invocando el procedimiento *`llegarACita`* con esa cita como argumento. La primera tarea que llega debe esperar a que llegue la segunda tarea. Como se muestra en el siguiente diagrama de avance de procesos, la cita es exitosa cuando la segunda tarea llega antes que transcurra el máximo tiempo de espera señalado como argumento en la primera tarea. En caso contrario la cita es un fracaso.

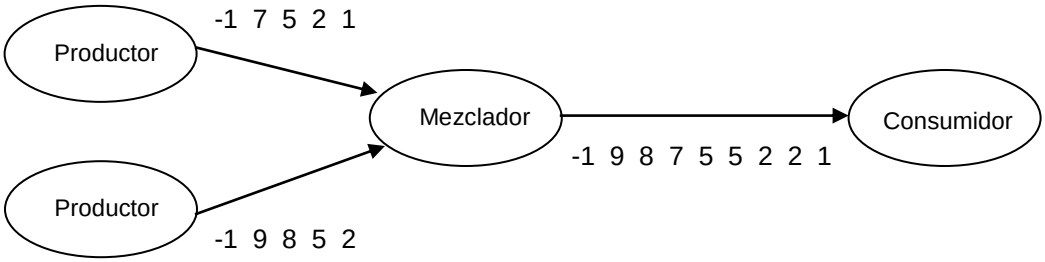


   Cuando la cita es exitosa, el llamado a *`llegarACita`* retorna EXITO1 en la tarea que llegó primero y EXITO2 en la tarea que llegó segunda. Entonces las dos tareas continúan su ejecución (y viven felices para siempre). Si la cita es un fracaso, transcurrido el máximo tiempo de espera, la tarea que llegó a la cita continúa su ejecución (indignada) y *`llegarACita`* retorna FRACASO. Tarde o temprano llegará a la cita la segunda tarea, la que continúa su ejecución recibiendo como valor de retorno ATRASADA.

   Implemente esta herramienta de sincronización (la representación de Cita y ambos procedimientos) utilizando las herramientas de sincronización de nSystem. El objeto coordinador (la cita) no es reutilizable para otra cita por lo que Ud. debe liberar cualquier recurso pedido al llegar la segunda tarea. En particular no se olvide de *`nWaitTask`*.

   Obs.: cualquier forma de busy-waiting es equivalente a una respuesta en blanco.

2. La tarea mezclador de la figura mezcla las dos secuencias de números ordenados provenientes de dos productores, generando una nueva secuencia ordenada que envía a un consumidor.



Un productor envía su secuencia en orden ascendente y terminándola con un –1. Cada número se envía por medio de un mensaje que resulta ser un entero(int). Cuando el mezclador recibe dos números provenientes de productores distintos, estos números no necesariamente están ordenados. El mezclador envía al consumidor una secuencia de números ordenados y terminada con un –1, cada número se envía en un mensaje.

El mezclador se crea mediante:

```
Mezcladador= nEmitTask(Mezcla, consumidor);
```

Programe el procedimiento Mezcla utilizando los mensajes de nSystem.

3. N tareas necesitan compartir un recurso único bajo nSystem. Cada tarea se identifica 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 aun 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.

4. En una bolsa de comercio trabajan operadores que venden o compran acciones de la empresa ACME. Los operadores son representados mediante tareas de nSystem. Cuando un operador desea vender acciones invoca el procedimiento:

```
nTask t= vendo(precio);
```

En donde precio es un entero que indica el precio de venta de la acción de ACME. Este procedimiento espera hasta que aparezca un operador que compre acciones a ese precio, en cuyo caso retorna el identificador de la tarea compradora.

Cuando un operador desea comprar acciones invoca el procedimiento:

```
nTask t= compro(precio);
```

En donde precio es el precio máximo que el comprador está dispuesto a pagar por la acción de ACME. Si en ese instante ningún operador vende a ese precio (o menos) se retorna NULL de inmediato sin quedarse en espera. Si hay vendedores por ese precio, se elige como contraparte al operador que venda más barato y se retorna su identificador. La contraparte también debe retornar.

Se le pide a Ud. implementar una solución de este problema usando las tareas y mensajes de nSystem. Para ello Ud. debe programar los procedimientos compro y vendo y además una tarea coordinadora que reciba mensajes de los operadores con la siguiente estructura:

```
typedef struct {
  int tipo; /* si es COMPRADOR o VENDEDOR */
  int precio; /* el precio especificado en la llamar
                 a compro o vendo */
  nTask yo; /* la tarea que realiza la llamada */
  nTask contraparte; /* la tarea con quien se hace la transaccion */
} Operacion;
```

Los campos yo y contraparte son asignados en la tarea coordinadora.

5. La federación de tenis de mesa de Chile organiza un torneo en donde todos los jugadores se enfrentan con todos. En el torneo participan 8 jugadores, numerados de 0 a 7 que se enfrentan en 7 rondas de 4 partidos. Para disminuir la duración de los torneos se requiere que los partidos se jueguen en paralelo, sujeto a que se dispone de 3 mesas identificadas como A, B, C.

Para hacer enfrentarse los jugadores *x* e *y*, en la mesa *m*. Usted dispone del procedimiento *enfrentar(x, y, m)*. Este procedimiento toma mucho tiempo y solo retorna cuando el partido finalizó. Algunos partidos pueden ser muy cortos (3 sets) mientras otros pueden ser muy extensos (5 sets muy disputados).

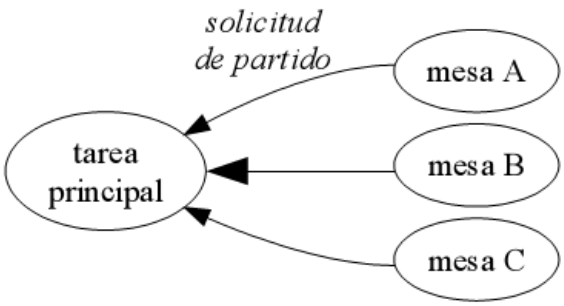
Se ha programado el siguiente procedimiento para realizar el torneo pedido:

```
struct { int x, y; } partidos[4*7]= /* 7 fechas, 4 partidos por ronda */
  { {0,4}, {1,5}, {2,6}, {3,7}, /* 0 enfrenta a 4, 1 a 5, etc. por la 1era. ronda, */
    {0,1}, {2,4}, {3,5}, {7,6}, /* 0 enfrenta a 1, 2 a 4, etc. por la 2da. ronda, */
    {0,2}, {3,1}, {7,4}, {6,5}, ... }; /* 3era. ronda ... y así hasta la 7ma. */
int prox; /* índice del próximo partido que se debe jugar */
```

```
void torneo() {
  prox= 0;
  nTask m1= nEmitTask(mesa, "A");
  nTask m2= nEmitTask(mesa, "B");
  nTask m3= nEmitTask(mesa, "C");
  nWaitTask(m1);
  nWaitTask(m2);
  nWaitTask(m3);
}
```

```
int mesa(char *m) {
  while (prox<4*7) {
    int x= partidos[prox].x;
    int y= partidos[prox].y;
    prox++;
    enfrentar(x, y, m);
  }
  return 0;
}
```

Resuelva el problema utilizando como única herramienta de sincronización los mensajes de nSystem. Utilice la organización de tareas que se indica en la figura de más abajo. Cada mesa es una tarea que solicita partidos a la tarea principal. Para ello la `mesa` envía un mensaje a la tarea principal, la que responde con el partido que se debe jugar. Entonces, la mesa realiza el enfrentamiento. Al finalizar, la mesa envía un nuevo mensaje a la tarea principal indicando qué partido se jugó. La tarea principal responderá este mensaje con el nuevo partido que se debe jugar a continuación (-1 si no quedan partidos para jugar), etc.



Restricción: si una mesa está disponible para jugar un nuevo partido, pero el próximo partido en el arreglo no se puede jugar porque uno de los jugadores todavía está jugando en otra mesa, Usted debe buscar algún otro partido que sí se pueda jugar y entregárselo a la `mesa`. Como en la pregunta anterior, `torneo` solo retorna cuando todos los partidos finalizaron.