CC41B : Sistemas Operativos
Control 1–Semestre Primavera'98

Prof.: Luis Mateu.

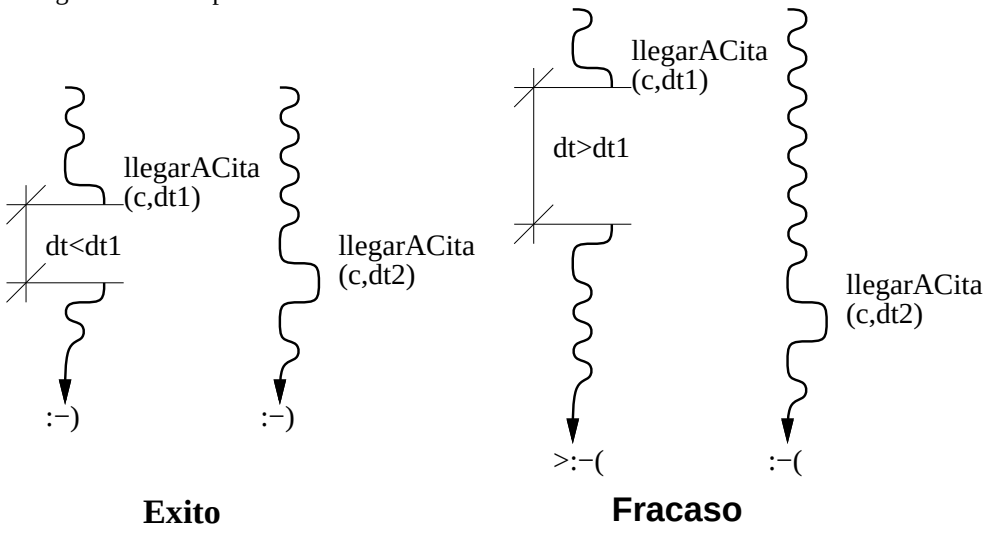
Sin apuntes, 1 hora 40 minutos

Pregunta 1 (40 %)

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.

Pregunta 2 (30 %)

Dos tarea t1 y t2 deben ejecutar primero Computo1() y Computo2() respectivamente. Sólo una vez que hayan finalizado ambos cómputos pueden ejecutar NuevoComputo1() y NuevoComputo2() respectivamente. Se ha diseñado la siguiente solución:

```
P() { /* la tarea 1 */
  nTask t;
  Computo1();
  if (nReceive(&t,0)==NULL)
    nSend(t2,"termine");
  else nReply(t,0);
  NuevoComputo1();
}
```

```
Q() { /* la tarea 2 */
  nTask t;
  Computo2();
  if (nReceive(&t,0)==NULL)
    nSend(t1,"termine");
  else nReply(t,0);
  NuevoComputo2();
}
```

Muestre por medio de un diagrama de avance de procesos que esta solución es incorrecta.

Pregunta 3 (30 %)

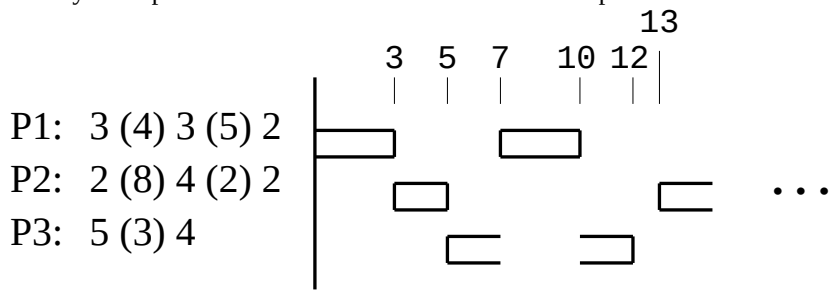
- Parte a.

  Se ha pensado en incorporar a nSystem una primitiva nKill que permita matar una tarea. Para implementarla, se propone guardar en el descriptor de cada tarea una lista enlazada con toda la memoria pedida por esa tarea. De esta forma nKill puede liberar la memoria pedida por esa tarea.

  Discuta la implementación propuesta.

- Parte b.

  El siguiente diagrama muestra el scheduling de 3 procesos. En tiempo 0 los 3 procesos están READY. La estrategia de scheduling es en base a prioridades fijas y distintas. Junto a cada proceso se indica la duración de las ráfagas de CPU y entre paréntesis la duración de los estados de espera.



  Responda: (i) Ordene los procesos de mejor a peor prioridad (ii) De qué scheduling se trata, premptive o non-preemptive y (iii) complete el diagrama. Justifique (i) y (ii).

Apéndice: API de nSystem

```
nTask nEmitTask(int (*proc)(), arg1, arg2, ... );
void nExitTask(int rc);
int nWaitTask(nTask task);
nTask nCurrentTask();
int nSend(nTask task, void *msg);
void *nReceive(nTask *ptask, int max_delay);
void nReply(nTask task, int rc);
void nSleep(int delay);
int nGetTime();
nSem nMakeSem(int count);
void nWaitSem(nSem sem);
```

```
void nSignalSem(nSem sem);
void Destroy(nSem sem);
```