

# Examen de Programación Concurrente

septiembre 2004

Dpto. LSIIS. Unidad de Programación.

## Normas

Esta prueba consta de dos partes diferenciadas. La primera, cuyo enunciado estás leyendo en este momento, contiene la primera parte y se recogerá **una hora** después de haberse entregado. Después se entregará el enunciado de la segunda parte. La puntuación total del examen es de **10 puntos**.

Las soluciones aparecerán publicadas antes de la fecha de revisión y las calificaciones se darán a conocer el **21 de septiembre de 2004**. La revisión de este examen tendrá lugar el **23 de septiembre de 2004**.

## 1. Pastas de té

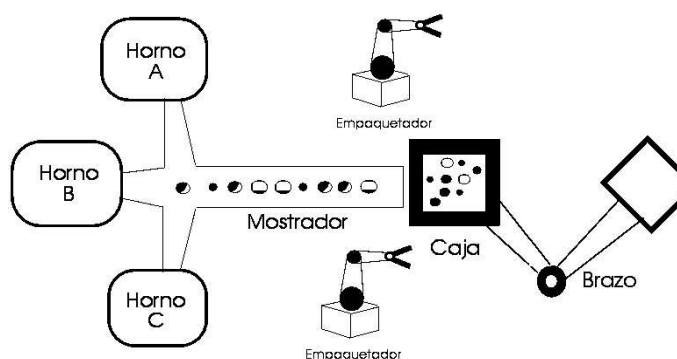
[3 puntos]

Una fábrica de pastas de té tiene tres hornos que producen tres tipos diferentes de pastas: A, B y C, con pesos diferentes: *pesoA*, *pesoB* y *pesoC*. Procedentes de los hornos, las pastas se van situando en un *mostrador* común.

Las pastas son empaquetadas en cajas. Uno o varios robots *Empaquetador* toman pastas del mostrador y las introducen en la caja (ver figura). Cada caja puede contener un número diferente de pastas siempre y cuando no se sobrepase un peso límite, denominado *PesoMaximo*. Por este motivo, antes de incluir una pasta en la caja, cada empaquetador debe asegurarse de que con su inclusión no se sobrepasa el peso máximo. Si no se sobrepasa el peso se incluye la pasta en la caja; en otro caso, un *Brazo* mecánico se encarga de retirar la caja que se estaba llenando y posteriormente la sustituye por una caja vacía.

Tened en cuenta de que se trata de llenar la caja lo más posible, lo cual puede ser conseguido por uno cualquiera de los robots que intentan depositar simultáneamente alguna pasta, de pesos que pueden variar. Se considera que no hay interferencia física entre robots que intentan soltar pastas al mismo tiempo en la caja.

Asumiremos que inicialmente hay una caja vacía junto al mostrador.



Para desarrollar el software de control se parte de paquetes Robot y Brazo – ya programados – que proporcionan, entre otras, los siguientes procedimientos de acceso a los dispositivos mecánicos (nos olvidamos del horno; no es asunto del software que debemos diseñar):

```
type Tipo_Empaquetador is Natural range 0..NumEmpaquetadores;
```

```
procedure Retirar.Caja;
```

- *Retirar.Caja* hace que el proceso que lo invoca
- quede bloqueado hasta que la caja que estaba siendo llenada es
- retirada por el brazo auxiliar.
- Requiere que haya una caja en la zona de relleno.

**procedure** Reponer\_Caja;

- *Reponer\_Caja* hace que el proceso que lo invoca
- *quede bloqueado* hasta que el brazo auxiliar coloque una caja
- *vacía en el área de relleno*.
- *No debe haber ninguna caja en la zona de relleno*.

**procedure** Tomar\_Pasta(Empaquetador: **in** Tipo\_Empaquetador;  
Peso: **out** TipoPeso);

- *Tomar\_Pasta(Empaq, Peso)* provoca que el proceso que lo invoca *quede*
- *bloqueado hasta que el el empaquetador con identificador Empaq*
- *toma la pasta más cercana del mostrador, registrando Peso*
- *el peso de la misma*.

**procedure** Soltar\_Pasta(Empaquetador: **in** Tipo\_Empaquetador);

- *Soltar\_Pasta(Empaq)* provoca que el proceso que lo invoca *quede*
- *bloqueado hasta que el el empaquetador con identificador Empaq*
- *suelta la pasta que acaba de tomar en la caja del área de*
- *relleno*.
- *Es necesario que haya una caja en el área de relleno, que el*
- *robot empaquetador en cuestión tenga una pasta y que la inclusión*
- *de esa pasta en la caja no haga sobrepasar el peso máximo permitido*.
- *Físicamente, no hay interferencia entre dos robots que intentan*
- *depositar pastas simultáneamente en una misma caja*.

**Se pide:**

- Grafo de procesos/recursos que permita controlar el sistema arriba descrito.
- Código esquemático de las tareas propuestas.
- Especificación formal (CTADSOL) del recurso o recursos propuestos.

# Examen de Programación Concurrente

septiembre 2004

Dpto. LSIIS. Unidad de Programación.

## Normas

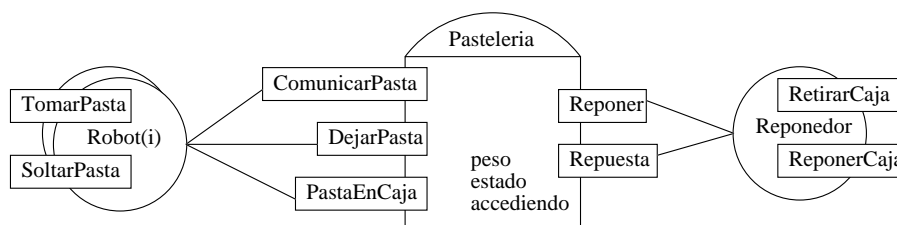
Disponéis de **dos horas** para responder a los 3 apartados ((a), (b) y (c)) de esta parte.

Cada uno de ellos se deberá entregar en un conjunto de hojas separadas.

## 2. Pastas de té – elaboración de código

[7 puntos]

A continuación se incluye un diseño propuesto: grafo de recursos y procesos, código de los procesos y especificación del recurso. Asumimos que el recurso está contenido en una variable P accesible por los procesos.



```
— Retirar y reponer una caja
loop
  P.Reponer;
  Retirar_Caja;
  Reponer_Caja;
  P.Repuesta;
end loop;
```

```
— Robot R-ésimo
loop
  Tomar_Pasta(R, Peso_Pasta);
  — Comprueba si este peso excede el máximo
  — e inicia la rutina de cambio de caja
  P.Comunicar_Pasta(Peso_Pasta);
  — Bloquea hasta que se pueda dejar esta pasta
  — y preincrementa el peso en la caja
  P.Dejar_Pasta(Peso_Pasta);
  Soltar_Pasta(R);
  P.Pasta_En_Caja;
end loop;
```

Se pide:

- Una implementación en Ada 95 de las operaciones **ComunicarPasta** y **DejarPasta** del C-TADSOL utilizando objetos protegidos que incluya la definición de tipos y variables de estado necesarias. [2.5 puntos]
- Una implementación en Ada 95 de las operaciones **ComunicarPasta** y **DejarPasta** del C-TADSOL utilizando *Rendez-Vous* (con paso de mensajes explícito si se considera necesario) que incluya la definición de tipos y variables de estado necesarias y el código de todas las operaciones. [2.5 puntos]
- Una variación del diseño en la que se considere que el brazo robot no puede calcular el peso de las pastas, sino que la plataforma en que está la caja tiene una balanza con una operación  
**PesoBalanza(Peso: out TipoPeso)**

que es **no bloqueante** y de ejecución **prácticamente inmediata**. El nuevo diseño debe cumplir en lo posible los mismos requisitos que el inicial. [2 puntos]

Recordad **entregar cada apartado en un juego de hojas separadas.**

#### C-TADSOL Pasteleria

##### OPERACIONES

ACCIÓN Reponer:  $TipoCaja[io]$   
 ACCIÓN Repuesta:  $TipoCaja[io]$   
 ACCIÓN ComunicarPasta:  $TipoCaja[io] \times TipoPeso[i]$   
 ACCIÓN DejarPasta:  $TipoCaja[io] \times TipoPeso[i]$   
 ACCIÓN PastaEnCaja:  $TipoCaja[io]$

##### SEMÁNTICA

###### DOMINIO:

TIPO:  $TipoCaja = (peso: TipoPeso \times estado: TipoEstado \times accediendo: \mathbb{N})$   
 $TipoEstado = preparada \mid a\_reponer \mid reponiendo$

INICIAL(c):  $c = (0, preparada, 0)$

CPRE:  $c = (-, a\_reponer, 0)$

**Reponer(c)**

POST:  $c^{sal} = (c^{ent}.peso, reponiendo, 0)$

CPRE: verdad

**Repuesta(c)**

POST:  $c^{sal} = (0, preparada, 0)$

CPRE:  $c.estado \neq reponiendo$

**ComunicarPasta(c, p)**

POST:  $c^{ent} = (pin, -, a) \wedge (pin + p > PesoMaximo \rightarrow c^{sal} = (pin, a\_reponer, a)) \wedge$   
 $(pin + p \leq PesoMaximo \rightarrow c^{sal} = (pin, preparada, a))$

CPRE:  $c.peso + p \leq PesoMaximo \wedge c.estado \neq reponiendo$

**DejarPasta(c, p)**

POST:  $c^{ent} = (pin, e, a) \wedge c^{sal} = (pin + p, e, a + 1)$

CPRE: verdad

**PastaEnCaja(c)**

POST:  $c^{ent} = (pin, e, a) \wedge c^{sal} = (pin, e, a - 1)$

## Notas sobre el funcionamiento del recurso

No es imprescindible que leáis estas notas; sin embargo pueden aclarar el funcionamiento del recurso para ayudar a realizar el apartado (c).

La reposición pasa por tres fases: *preparada* (se pueden dejar pastas), *a\_reponer* (se debe reponer una caja, porque hay un robot que puede sobrepasar su capacidad) y *reponiendo* (se ha iniciado la secuencia de reposición). El estado *a\_reponer* no es irreversible, pues sólo señala que un brazo robot necesita una caja nueva, pero puede haber otro brazo que lleve una pasta de peso inferior al necesario para exceder el peso máximo de la caja. Si la *señal* de este segundo brazo llega a tiempo puede eliminar la petición de recambio y dejar una pasta que aún cabe en la caja. El estado *reponiendo* sí es irreversible, pues indica que el brazo reponedor ha recibido la necesidad de reposición e iniciará el proceso de cambio de caja.

Para que haya un buen comportamiento en caso de varios robots, el peso de caja se guarda en el recurso, lo que garantiza su acceso atómico. La CPRE de **DejarPasta** asegura que:

- Ningún robot puede exceder el peso máximo ( $c.peso + p \leq PesoMaximo$ ), pues se comprueba justo antes de intentar dejar la pasta y su POST incrementa (atómicamente) dicho peso antes

de que este sea efectivo en la caja, con lo que el recurso contiene un avance del futuro de la caja y las decisiones tomadas según este valor son seguras.

- Ningún robot dejará una pasta mientras se está reponiendo la caja (*c.estado*  $\neq$  *reponiendo*).
- Pero un robot cuya pasta no causa exceso de peso puede dejar la mercancía aunque haya otro robot bloqueado por su pasta mientras no se haya iniciado la secuencia de reposición.

En este último caso, el robot que ha *adelantado* a los demás elimina la señal de reposición. Será este mismo (u otro, si hay más no bloqueados) el que la reactive, si es necesario, en el momento en que, tras dejar su pasta, adquiera otra del horno e intente depositarla en la caja. Esto intenta maximizar el peso final de la caja. De no hacerlo así, una solución extrema, correcta, pero evidentemente indeseable, es poner siempre una sola pasta y reponer la caja.

Una solución que maximiza el peso en lo posible es esperar a **todos** los robots en una barrera, calcular qué combinación de pastas de entre las disponibles daría un mejor resultado, elegir la mejor de las pastas, depositar **sólo** esa pasta repetir el ciclo. Sin embargo esta solución tiene una concurrencia muy baja, ya que tiende a que en cada momento sólo un robot acceda a la caja y al mostrador.

El conteo de número de brazos robot accediendo a la caja es necesario para implementar una exclusión parcial (como en los problemas de *Lectores y Escritores*) que impida que se inicie una reposición de la caja mientras un brazo robot accede a la misma. Ello podría suceder si mientras se deposita (físicamente) una pasta, otro robot excede el peso máximo y pide un cambio de caja que se inicia mientras el anterior está aún dejando la pasta. Por otra parte, se permite que varios brazos robots dejen simultáneamente pastas en la caja.