

Sesión 15: Comunicación asíncrona (Linda)

Un sistema consta de tres procesos distribuidos que se coordinan a través de Linda. El proceso *ProdNum* genera y escribe un número aleatorio en el espacio de *tuplas* y espera a que algún proceso lector lo consuma. Este comportamiento lo repite indefinidamente. Por otro lado, los otros dos procesos, llamados *ConsPar* y *ConsImpar*, actúan como lectores de números pares e impares, respectivamente. Cada número producido lo leen una única vez, mostrándolo por su correspondiente pantalla.

Se pide completar el código de los tres procesos que forman el sistema, representando el estado del sistema y la información intercambiada con una única *tupla* linda. Es necesario explicar de forma breve y concisa el formato de este mensaje y la semántica de cada uno de sus campos.

Sesión 15: Comunicación asíncrona (Linda)

El sistema anterior constaba de un proceso escritor y dos lectores. Se pide programar una nueva versión del sistema en la que hay 10 procesos *ConsPar* y 10 procesos *ConsImpar*. En este caso, la solución debe garantizar que cada vez que se escriba un nuevo número par en el espacio de tuplas, éste sea leído una sola vez por todos los procesos *ConsPar* (similar comportamiento para los números impares y los procesos *ConsImpar*).

Se pide completar el código de los procesos involucrados, representando el estado del sistema y la información intercambiada con una única tupla *Linda*. Es necesario explicar de forma breve y concisa el formato de este mensaje y la semántica de cada uno de sus campos.

Sesión 15: Comunicación asíncrona (Linda)

Un sistema automático controla y gestiona la capacidad de un pantano. El sistema consta de 4 procesos: un proceso *controlador* y tres procesos *actuadores*. El controlador lee periódicamente un conjunto de sensores instalados en el pantano, determina cuál es el porcentaje de su capacidad en ese momento, y activa si es necesario alguna de las alertas de interés. Se han definido tres alertas y sus consiguientes acciones de respuesta:

- cuando el pantano se encuentra por debajo del 60% de su capacidad, se cierra la canalización que surte agua al regadío de la zona,
- cuando baja del 40%, se cierra la canalización para consumo humano,
- y por debajo del 20%, se abre la compuerta que permite que entre más agua procedente de sus afluentes.

Las condiciones de estas alertas no son mutuamente excluyentes. En el caso de que se cumpla más de una condición, se notificarán todas las alertas involucradas.

Sesión 15: Comunicación asíncrona (Linda)

Por otro lado, un proceso actuador es responsable de dar respuesta a un par alerta-acción. Todos los actuadores se comportan de la misma manera. Inicialmente están a la espera de que se active su alerta y, cuando esto sucede, ejecutan la correspondiente acción (por ejemplo, cerrar canalización o abrir compuerta). En el momento en el que la alerta deja de estar activada, ponen fin a la acción y vuelven a su estado de partida.

Además, se quiere contabilizar, con fines estadísticos, cuántas veces se ejecuta cada tipo de acción correctiva y mostrar periódicamente los resultados por pantalla. Las acciones se contabilizarán cuando estas se pongan en ejecución. El controlador será el responsable de imprimir las estadísticas.

Se pide programar un sistema distribuido basado en Linda que simule el comportamiento descrito. Un proceso *Init* será el encargado de gestionar la inicialización coordinada de los procesos del sistema. La solución debe contener una descripción detallada del protocolo de comunicación entre todos los procesos participantes, incluido el formato de las *tuplas*.

Sesión 15: Comunicación asíncrona (Linda)

PISTA: La versión más simple del protocolo consta de 2 tuplas.

A continuación se proporciona el código incompleto del procesos *Controlador* y del proceso *actuador* que gestiona la alerta de “capacidad por debajo del 60%”. El resto de actuadores tienen una estructura de código similar (la solución debe contener el código de los 3 actuadores).

```
Process Controlador
  integer capacidad // Valor de 0..100
  ... // Inicialización controlada
  while true
    capacidad := monitorizacion_sensores() // Capacidad actual?
    ... // Toma de decisiones en función de la lectura

    ... // Imprimir estadísticas

    espera() // Espera cierto tiempo hasta nueva lectura
  end
end

Process Actuador_60
  ... // Inicialización controlada
  while true
    ... // Alerta activada?
    cerrar_canalizacion()
    ... // Alerta desactivada?
    abrir_canalizacion()
  end
end
```

Sesión 16: Comunicación asíncrona (Linda)

Un sistema distribuido contabiliza y gestiona las partidas de un juego en red. El número máximo de partidas que se pueden estar jugando simultáneamente es de 25. En cada partida se enfrentan dos jugadores cualesquiera de entre los registrados en el sistema. Antes de iniciar una partida es necesario crear una pareja de jugadores. Para ello, un primer jugador solicita iniciar una nueva partida y espera a que realice la misma acción un posible contrincante. Cuando lo hace, se ponen de acuerdo para comenzar a jugar. En un momento determinado, el jugador de menor identificador de la pareja decide terminar el juego, se lo comunica a su pareja, espera su confirmación y actualiza el estado del sistema. El sistema consta de 100 procesos *Jugador*, que participan de forma continuada en distintas partidas, y un proceso *Gestor*, que inicializa el sistema y muestra periódicamente, por la salida estándar, el estado del sistema (se limita a informar del número de parejas que están jugando).

Sesión 16: Comunicación asíncrona (Linda)

Se pide completar el código de los procesos involucrados, utilizando la notación presentada en clase, y resolviendo las cuestiones de comunicación y sincronización del sistema por medio de Linda.

Además, la solución debe satisfacer las siguientes restricciones: 1) No podrá estar basada en un proceso servidor que encapsule el estado del sistema y sea el responsable de gestionar la creación de las parejas, y 2) Deberá contener una descripción clara y concisa de la sintaxis y la semántica de las *tuplas* que se utilizan para coordinar los procesos involucrados.

Sesión 16: Comunicación asíncrona (Linda)

Se pide
notación
comuni
Adem
podrá
sistem
Deber
semáfor
involu

```
process Jugador(i::1..100)
  -- Declaración de variables locales
  loop forever
    DESCANSANDO() --implementado
    -- Buscar contrincante
    -- Sincronizar el inicio de la partida
    JUGAR() --implementado
    -- Sincronizar el final de la partida
    -- Finalizar de la partida (proceso con menor identificador)
  end loop
end process

process Gestor
  -- Declaración de variables locales
  -- Establecer el estado inicial del sistema
  loop forever
    -- Mostrar por salida estándar el número de parejas jugando
    ESPERANDO()
  end loop
end process
```

lo la
de

No
del
y 2)
y la
esos

Sesión 16: Comunicación asíncrona (Linda)

Un supermercado ha establecido las siguientes restricciones de acceso a sus clientes: el número máximo de clientes que puede haber en su interior es de 20 y no más de 5 clientes simultáneamente en cada pasillo. El supermercado tiene 3 pasillos. El comportamiento de un cliente es el siguiente: intenta entrar en la tienda y, cuando lo consigue, visita en cierto orden los pasillos para ir completando su compra hasta que ésta finaliza. Este comportamiento lo repite de manera continua, una vez tras otra.

Se pide programar un sistema distribuido compuesto por 100 procesos cliente que simule el comportamiento del supermercado. Las restricciones de acceso se programarán utilizando el modelo de coordinación Linda. La solución deberá contener una descripción clara y concisa del formato de las *tuplas* utilizadas y del protocolo definido para sincronizar la acción de los procesos.

Sesión 16: Comunicación asíncrona (Linda)

Además, se deberá completar el código base de los clientes que se proporciona como parte del enunciado. La estructura de repetición interna (*while*) representa el comportamiento del cliente una vez ha logrado entrar en la tienda. En cada repetición se decide cuál el siguiente pasillo a visitar (por medio de la función *siguientePasillo()*), se añaden a la cesta los productos a comprar (*cogerProductos()*), y se comprueba si la compra está completa (*seguirComprando()*). Debe asumirse que estas tres funciones están ya implementadas.

```
Process Cliente(i:1..100)::
  integer pasillo
  boolean compraIncompleta := true

  loop
    ... //protocolo de entrada a la tienda
    while (compraIncompleta)
      pasillo := siguientePasillo()
      ... //protocolo de entrada al pasillo
      cogerProductos()
      ... //protocolo de salida del pasillo
      compraIncompleta := seguirComprando()
    end
    ... //protocolo de salida de la tienda
  end
end
```