Sistemas en Tiempo Real

Grado en Ingeniería Electrónica

UNIDAD 7: Acciones atómicas, procesos concurrentes y fiabilidad

Capítulo 7 de la edición Inglés (2009)

Capítulo 10 de la edición en Español (2003)

Tutoría

1. Introducción
2. Acciones atómicas
3. Acciones atómicas en lenguajes concurrentes
4. Notificación asíncrona
5. Notificación asíncrona en lenguajes concurrentes

Introducción

Hemos visto la interacción de tareas en relación con tres tipos de comportamiento:

- Independientes
- Cooperativas
- Competitivas

Las tareas **independientes** no se comunican con las demás, por lo que pueden realizar procedimientos de recuperación de errores de forma aislada del resto del sistema (Unidades 2 y 3).

Sin embargo, cuando el error se produce en tareas **cooperativas**, todas las tareas involucradas deben realizar la **recuperación del error**.

Las tareas **competitivas**, aunque esencialmente independientes, se comunican y se sincronizan para obtener recursos. Un error en una de ellas no debería afectar al resto. Pero, si el error ocurre mientras se le está **asignando un recurso**, podría hacerlo. Esto también puede pasar en las tareas **cooperativas**. Este aspecto de la asignación de recursos se comentará en la unidad 8.

Introducción

Procedimientos de recuperación de errores.

Existen dos tipos de recuperación:

- **Hacia adelante,** se continúa con la ejecución del sistema haciendo correcciones selectivas de su estado.
- **Hacia atrás,** en caso de error el sistema se restaura a un estado seguro previo y se ejecuta desde ese punto.

  Ventajas:

  - No es necesario localizar el error.
  - Permite el tratamiento de errores no previstos.

  Inconvenientes:

  - No se puede recuperar de errores del entorno.
  - Efecto dominó, se retorna a puntos de recuperación debido a posibles datos incorrectos en la comunicación entre varias tareas.
  - El error puede volver a ocurrir, debe ser eliminado en dos etapas: localización y reparación.

Acciones atómicas

La interacción entre varias tareas puede tener que hacerse en varias comunicaciones, y el estado del sistema debe ser consistente en todo momento. Cuando hay tareas concurrentes es fácil que los grupos de tareas interfieran entre sí.

En este caso, lo que se necesita es que cada grupo de tareas ejecute su actividad conjunta como una acción indivisible o atómica.

Propiedades de una **acción atómica**:

- Las tareas que la realizan no saben de la existencia de otra tarea activa, y ninguna otra tarea activa tiene constancia de las actividades de las tareas durante el tiempo que en el que están realizando la acción.
- Las tareas que la realizan solo se comunican entre sí, y no con otras tareas, mientras la están realizando.
- Las tareas que la realizan no pueden detectar ningún cambio de estado salvo aquellos realizados por ellas mismas, y no revelan sus cambios de estado hasta que la acción se haya completado.
- En lo que respecta a otras tareas, pueden ser consideradas indivisibles e instantáneas, de forma que los efectos sobre el sistema sean como si estuvieran entrelazadas y no en concurrencia.

Las acciones atómicas se pueden descomponer: acciones atómicas anidadas.

Acciones atómicas

Requisitos de las acciones atómicas

- **Frontera (límites bien definidos)**: cada acción atómica debe tener un comienzo, un fin y una demarcación.
- **Indivisibilidad (aislamiento)**: no debe permitir el intercambio de información entre las tareas activas de dentro de la acción y las de fuera (salvo los gestores de recursos).
- **Anidamiento**: pueden anidarse, mientras no se solapen con otras acciones atómicas
- **Concurrencia**: debería ser posible ejecutar concurrentemente distintas acciones atómicas.

Como el propósito de las acciones atómicas es formar la base para el confinamiento de los daños, deben permitir la programación de **procedimientos de recuperación de errores**.

A las acciones atómicas se les puede añadir la recuperación de errores **hacia adelante con manejadores de excepciones**. Si una tarea genera una excepción, todas las tareas activas en la acción deben manejar la excepción.

Puede haber más de una tarea activa en una acción atómica que genere excepciones diferentes al mismo tiempo (**excepciones concurrentes**) y podría ser complicado decidir qué manejador de excepciones utilizar. Una solución para esto consiste en utilizar **árboles de excepciones**. La excepción que se utiliza para identificar el manejador es la que ocupa la raíz del subárbol más pequeño que contiene todas las excepciones lanzadas.

Acciones atómicas

Transacción atómica

Tiene todas las propiedades de una **acción atómica**, más la característica de que su ejecución puede tener éxito o fallar.

Si falla una **acción atómica**, los componentes del sistema manipulados por ella podrían terminar en un estado inconsistente.

Una **transacción atómica** garantiza que, ante un fallo, los componentes se devuelven al estado que tenían antes de iniciarse la transacción (**recuperación hacia atrás**), por lo que se conocen también como **acciones recuperables**, aunque no permiten la escritura de procedimientos de recuperación alternativos.

Propiedades de una **transacción atómica**:

- **Atomicidad de fallo**: se debe completar con éxito o no tener efecto.
- **Atomicidad de sincronización**: es indivisible, no se puede observar su ejecución parcial.

Acciones atómicas

Conversaciones

Las acciones atómicas se pueden utilizar de manera que, si ocurre un error dentro de la acción atómica, las tareas involucradas pueden ser retrotraídas al comienzo de la acción (**recuperación hacia atrás)** y se pueden ejecutar algoritmos alternativos.

Funcionamiento de una conversación:

- Al entrar en la conversación se guarda el estado de la tarea y el conjunto de puntos de entrada forma la secuencia de recuperación.
- En el interior de la conversación, sólo se permite la comunicación con otras tareas activas en la conversación y con gestores generales de recursos.
- Para poder abandonar la conversación, todos los procesos activos en la conversación deben haber pasado el test de aceptación. En este caso, se finaliza la conversación y se desechan los puntos de recuperación.
- Si una tarea falla el test de aceptación, todas las tareas recuperarán el estado guardado y ejecutarán los módulos alternativos.
- Las conversaciones pueden anidarse, pero sólo si el anidamiento es estricto.
- Si fallan todas las alternativas de la conversación, la recuperación se realizará en un nivel superior.

Acciones atómicas en lenguajes concurrentes

- **C Real Time/POSIX** utiliza **variables de condición (semáforos)**.
- **Ada** usa **guardas** en lugar de semáforos. El controlador se implementa como un **tipo protegido**. El **objeto protegido** sincroniza la acción, para que solo las tareas involucradas estén activas y se sincronicen a la salida. La última tarea debe cerrar la guarda.
- La aproximación de **Java** es similar a la de Ada, y se puede extender fácilmente utilizando herencia:
  - Se define una interfaz para las acciones atómicas. Esta, puede ser extendida para proporcionar diversos controladores de acción que implementan varios modelos. Las aplicaciones eligen el controlador apropiado sin tener que cambiar el código.
  - Cada clase implementa los protocolos de sincronización de entrada y salida.
  - Es necesario contar las tareas cuando salen de la acción, para saber cuándo inicializar las estructuras de datos para la siguiente acción.

Notificación asíncrona

Permite la interacción entre las tareas y la recuperación de errores en las acciones atómicas de manera que una tarea puede llamar a otra sin tener que esperar (waiting o polling).

Casos en que no es adecuada la espera:

Recuperación de errores: Cuando hay grupos de tareas que realizan acciones atómicas, la detección de un error en una tarea necesita de la participación del resto para la recuperación.

Cambios de modo: Los sistemas de tiempo real normalmente tienen distintos modos de operación. Hay casos en que los cambios de modo no se pueden planificar. Cuando esto sucede, las tareas deben ser informadas de forma rápida y segura del cambio y deben proceder a un conjunto de acciones diferentes.

Planificación utilizando computaciones parciales/imprecisas: Hay algoritmos en los que la precisión del resultado depende del tiempo asignado al cálculo. En tiempo de ejecución, a cada algoritmo se le asigna una cierta cantidad de tiempo tras el cual el proceso debe finalizar.

Interrupciones de usuario: En un entorno interactivo, el usuario puede querer finalizar el procesamiento y comenzarlo de nuevo.

Notificación asíncrona

Existen dos modelos básicos:

Reanudación (resumption)

Cada tarea indica las excepciones que maneja. En caso de que se lance una, se avisa y se cambia el flujo de control de la tarea al manejador. El manejador responde al evento asíncrono, y la tarea continúa su ejecución en el punto el que estaba.

POSIX y **Java Real Time** proporcionan soporte para este modelo.

Terminación (termination)

Cada tarea especifica un dominio de ejecución en el que está preparada para recibir una notificación asíncrona, que hará que el dominio sea finalizado (ATC: asynchronous transfer of control). Si una petición ATC ocurre fuera de este dominio es ignorada o encolada.

Una vez manejada la ATC, el control vuelve a la tarea interrumpida en una posición distinta a donde se encontraba al recibir la notificación.

Los lenguajes **Ada** y **Java Real Time** soportan ATC.

Notificación asíncrona en POSIX

Señales POSIX

Implementa el modelo de **reanudación**.

Existe un conjunto de señales predefinidas y otras que se definen en implementación. Cada señal tiene un manejador por defecto.

Las señales de Tiempo Real tienen información adicional para enviar al manejador desde la tarea en que se generó. Además, se guardan en una cola, y hay tres maneras de gestionarlas:

- **Bloquearla** y manejarla más tarde.
- **Manejarla** llamando a la función establecida para ello.
- **Ignorarla** en conjunto (la señal se pierde).

Notificación asíncrona en Ada

Select asíncrono (select-then-abort)

Ada implementa el modelo de **terminación** basado en la transferencia asíncrona de control (ATC).

La parte select empieza con un disparo (entry) o un retardo de disparo (delay). Si es un entry, se ejecuta la llamada. Si es un delay, se pone en cola y se ejecuta la parte abort.

Si la sentencia de disparo se completa antes que la parte abortable, esta se aborta y se ejecuta la secuencia de sentencias opcionales que siguen a la secuencia de disparo.

Si la parte abortable se completa antes que la invocación entry, se intenta cancelar la entry y, si esto tiene éxito, se finaliza la ejecución de la sentencia select asíncrona.

```
select
Trigger.Event:
-- secuencia opcional de sentencias que son
-- ejecutadas tras la recepción del evento
then abort
-- secuencia abortable de sentencias
end select;
```

Notificación asíncrona en Java Real Time

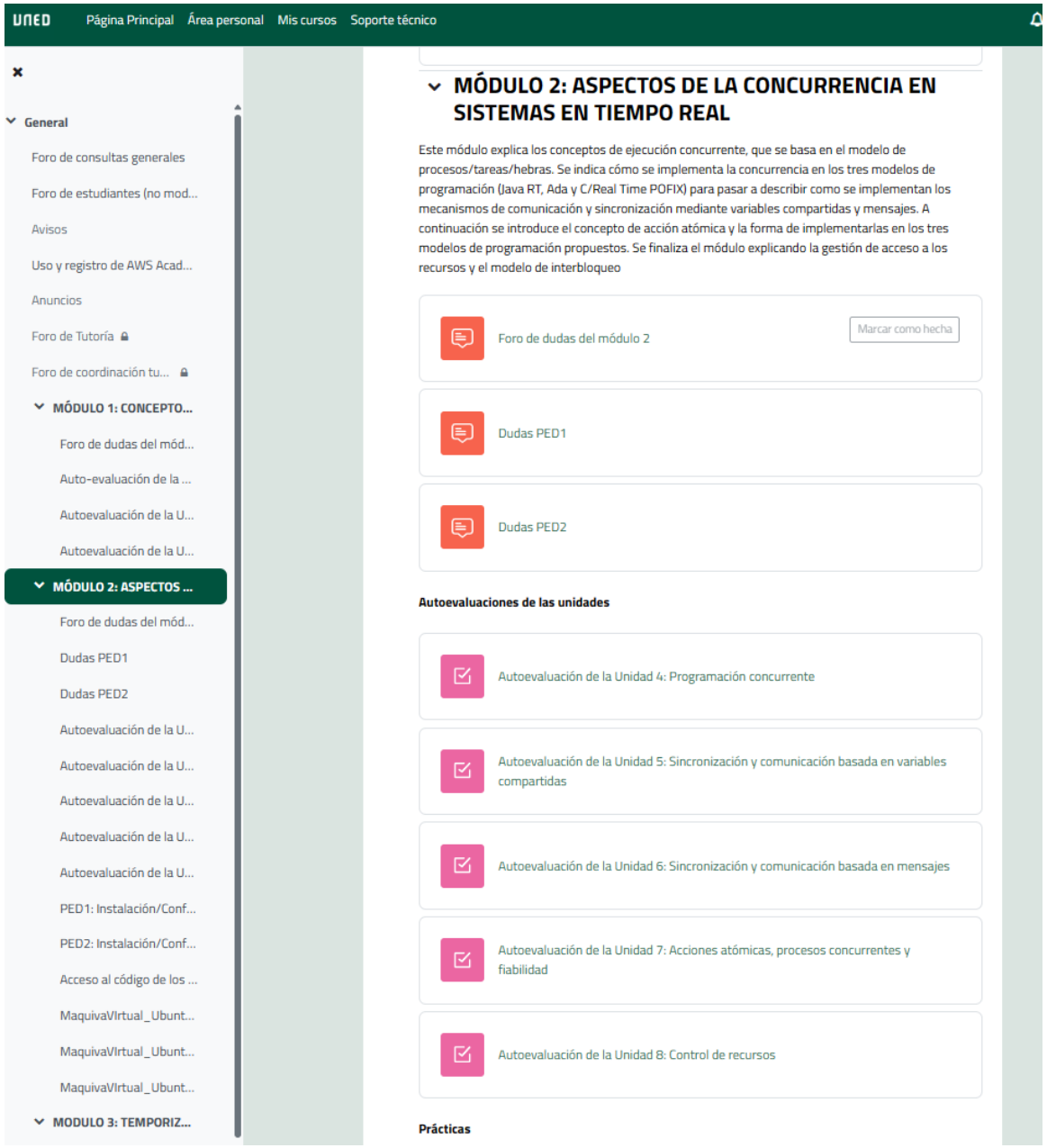
Java Real-Time soporta los modelos de reanudación y terminación.

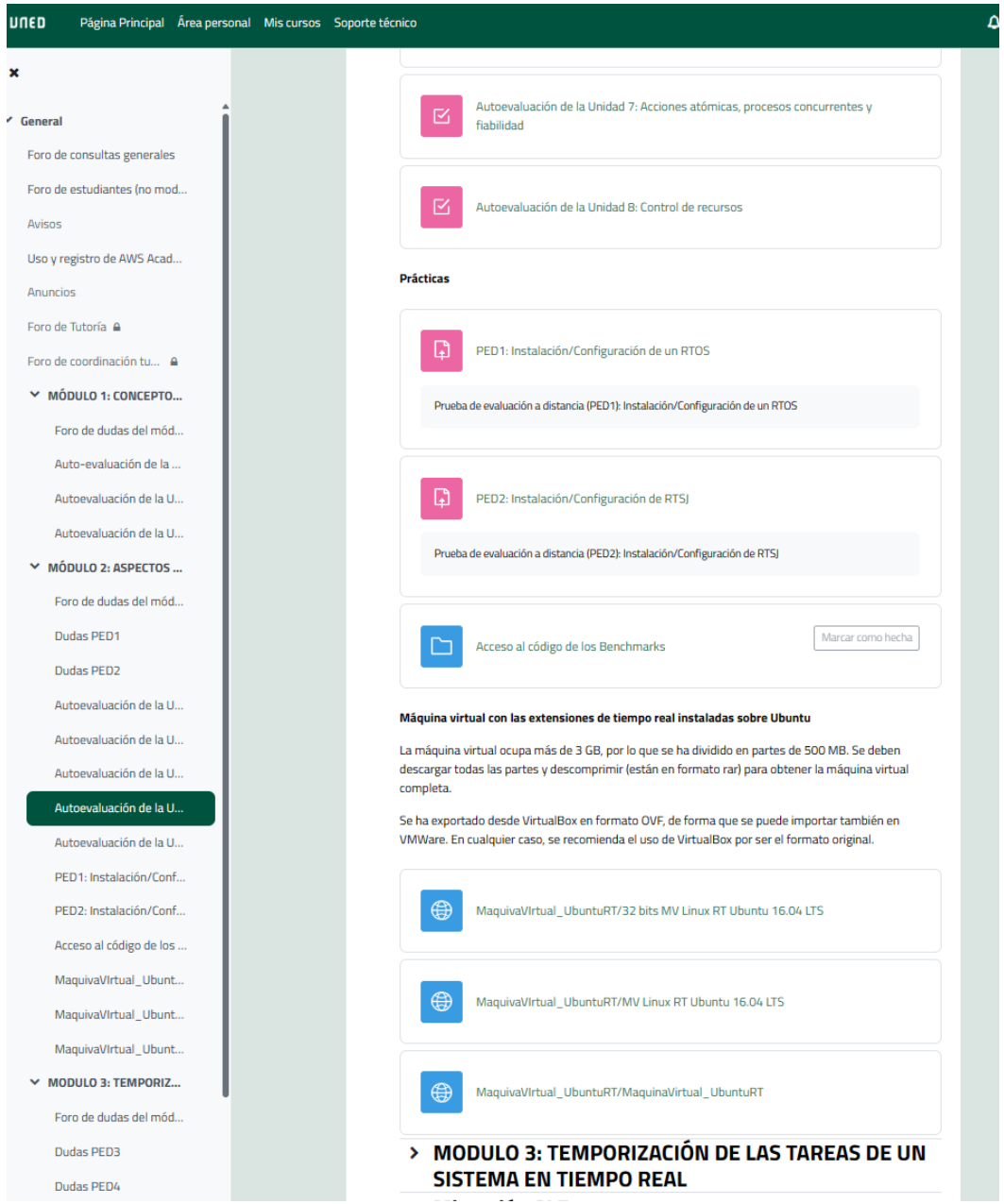
Los **eventos asíncronos** son el equivalente a las señales POSIX para implementar el modelo de **reanudación.**

En él, los manejadores de eventos son entidades programadas y cada tarea puede tener asociado más de un manejador.

El modelo de **terminación** se basa en la transferencia asíncrona de control (ATC). Es similar al de Ada, pero con las siguientes diferencias:

- Está integrado con el mecanismo de manejo de excepciones, mientras que el modelo Ada está integrado con los mecanismos de la sentencia *select* y del manejo de entradas.
- Requiere que cada método indique que está preparado para permitir la ocurrencia de una ATC. La ATC se difiere hasta que la tarea esté en ejecución en dicho método. Sin embargo, la opción por defecto de Ada es permitir la ATC y las respuestas diferidas deben controlarse explícitamente.


UNED
Página Principal
Área personal
Mis cursos
Soporte técnico
General
Foro de consultas generales
Foro de estudiantes (no mod...
Avisos
Uso y registro de AWS Acad...
Anuncios
Foro de Tutoría
Foro de coordinación tu...
MÓDULO 1: CONCEPTO...
Foro de dudas del mód...
Auto-evaluación de la ...
Autoevaluación de la U...
Autoevaluación de la U...
MÓDULO 2: ASPECTOS ...
Foro de dudas del mód...
Dudas PED1
Dudas PED2
Autoevaluación de la U...
Autoevaluación de la U...
Autoevaluación de la U...
Autoevaluación de la U...
Autoevaluación de la U...
PED1: Instalación/Conf...
PED2: Instalación/Conf...
Acceso al código de los ...
MaquivaVIrtual_Ubunt...
MaquivaVIrtual_Ubunt...
MaquivaVIrtual_Ubunt...
MODULO 3: TEMPORIZ...
MÓDULO 2: ASPECTOS DE LA CONCURRENCIA EN SISTEMAS EN TIEMPO REAL
Este módulo explica los conceptos de ejecución concurrente, que se basa en el modelo de procesos/tareas/hebras. Se indica cómo se implementa la concurrencia en los tres modelos de programación (Java RT, Ada y C/Real Time POFIX) para pasar a describir como se implementan los mecanismos de comunicación y sincronización mediante variables compartidas y mensajes. A continuación se introduce el concepto de acción atómica y la forma de implementarlas en los tres modelos de programación propuestos. Se finaliza el módulo explicando la gestión de acceso a los recursos y el modelo de interbloqueo
Foro de dudas del módulo 2
Marcar como hecha
Dudas PED1
Dudas PED2
Autoevaluaciones de las unidades
Autoevaluación de la Unidad 4: Programación concurrente
Autoevaluación de la Unidad 5: Sincronización y comunicación basada en variables compartidas
Autoevaluación de la Unidad 6: Sincronización y comunicación basada en mensajes
Autoevaluación de la Unidad 7: Acciones atómicas, procesos concurrentes y fiabilidad
Autoevaluación de la Unidad 8: Control de recursos
Prácticas


UNED
Página Principal
Área personal
Mis cursos
Soporte técnico
General
Foro de consultas generales
Foro de estudiantes (no mod...
Avisos
Uso y registro de AWS Acad...
Anuncios
Foro de Tutoría
Foro de coordinación tu...
MÓDULO 1: CONCEPTO...
Foro de dudas del mód...
Auto-evaluación de la ...
Autoevaluación de la U...
Autoevaluación de la U...
MÓDULO 2: ASPECTOS ...
Foro de dudas del mód...
Dudas PED1
Dudas PED2
Autoevaluación de la U...
Autoevaluación de la U...
Autoevaluación de la U...
Autoevaluación de la U...
Autoevaluación de la U...
PED1: Instalación/Conf...
PED2: Instalación/Conf...
Acceso al código de los ...
MaquivaVIrtual_Ubunt...
MaquivaVIrtual_Ubunt...
MaquivaVIrtual_Ubunt...
MODULO 3: TEMPORIZ...
Foro de dudas del mód...
Dudas PED3
Dudas PED4
Autoevaluación de la Unidad 7: Acciones atómicas, procesos concurrentes y fiabilidad
Autoevaluación de la Unidad 8: Control de recursos
Prácticas
PED1: Instalación/Configuración de un RTOS
Prueba de evaluación a distancia (PED1): Instalación/Configuración de un RTOS
PED2: Instalación/Configuración de RTSJ
Prueba de evaluación a distancia (PED2): Instalación/Configuración de RTSJ
Acceso al código de los Benchmarks
Marcar como hecha
Máquina virtual con las extensiones de tiempo real instaladas sobre Ubuntu
La máquina virtual ocupa más de 3 GB, por lo que se ha dividido en partes de 500 MB. Se deben descargar todas las partes y descomprimir (están en formato rar) para obtener la máquina virtual completa.
Se ha exportado desde VirtualBox en formato OVF, de forma que se puede importar también en VMWare. En cualquier caso, se recomienda el uso de VirtualBox por ser el formato original.
MaquivaVIrtual_UbuntuRT/32 bits MV Linux RT Ubuntu 16.04 LTS
MaquivaVIrtual_UbuntuRT/MV Linux RT Ubuntu 16.04 LTS
MaquivaVIrtual_UbuntuRT/MaquinaVirtual_UbuntuRT
MODULO 3: TEMPORIZACIÓN DE LAS TAREAS DE UN SISTEMA EN TIEMPO REAL

Sistemas en Tiempo Real

Grado en Ingeniería Electrónica

UNIDAD 7: Acciones atómicas, procesos concurrentes y fiabilidad

UNED

Nieves Pedreira Souto