Sesión 18: Monitores – Traduciendo especificaciones de recurso

Julio Mariño
abril 2020

Concurrencia 2019/2020

Universidad Politécnica de Madrid

Grado en Ingeniería Informática
Grado en Matemáticas e Informática
2ble. grado en Ing. Informática y ADE

`http://babel.upm.es/teaching/concurrencia`

introducción/motivación

hacia un código de sincronización simple y trazable

- Nuestro objetivo es obtener soluciones a problemas de concurrencia que resultarían muy difíciles con semáforos.
- Pero ya hemos visto que las colas de monitores y *conditions* pueden interactuar de maneras no triviales.
- La solución que proponemos es usar los monitores de una manera restringida, con una traducción sencilla y trazable de las especificaciones de recurso.
- Más concretamente, tendremos que decidir: 1.- una estructura general para los métodos que implementan acciones de un recurso compartido, 2.- qué condition queues se necesitan para los bloqueos y 3.- una estrategia para efectuar los desbloqueos cuando hay varios hilos candidatos a ser desbloqueados.
- La estructura de los métodos será la que hemos usado en la clase anterior:

```
public tipo accion (...) {
  mutex.enter();
  if (! <CPRE>) { quecondition.await() };
  // si estamos aqui es que se cumple la CPRE
  <codigo que establece la POST>
  // podemos desbloquear a otro hilo?
  somecondition.signal();
  mutex.leave();
  // opcionalmente:
  return (<lo que sea>);
}
```

POLITÉCNICA

introducción/motivación

hacia un código de sincronización simple y trazable

- Nuestro objetivo es obtener soluciones a problemas de concurrencia que resultarían muy difíciles con semáforos.
- Pero ya hemos visto que las colas de monitores y *conditions* pueden interactuar de maneras no triviales.
- La solución que proponemos es usar los monitores de una manera restringida, con una traducción sencilla y trazable de las especificaciones de recurso.
- Más concretamente, tendremos que decidir: 1.- una estructura general para los métodos que implementan acciones de un recurso compartido, 2.- qué condition queues se necesitan para los bloqueos y 3.- una estrategia para efectuar los desbloqueos cuando hay varios hilos candidatos a ser desbloqueados.
- La estructura de los métodos será la que hemos usado en la clase anterior:

```
public tipo accion (...) {
  mutex.enter();
  if (! <CPRE>) { quecondition.await() };
  // si estamos aqui es que se cumple la CPRE
  <codigo que establece la POST>
  // podemos desbloquear a otro hilo?
  somecondition.signal();
  mutex.leave();
  // opcionalmente:
  return (<lo que sea>);
}
```

POLITÉCNICA

introducción/motivación

hacia un código de sincronización simple y trazable

- Nuestro objetivo es obtener soluciones a problemas de concurrencia que resultarían muy difíciles con semáforos.
- Pero ya hemos visto que las colas de monitores y *conditions* pueden interactuar de maneras no triviales.
- La solución que proponemos es usar los monitores de una manera restringida, con una traducción sencilla y trazable de las especificaciones de recurso.
- Más concretamente, tendremos que decidir: 1.- una estructura general para los métodos que implementan acciones de un recurso compartido, 2.- qué condition queues se necesitan para los bloqueos y 3.- una estrategia para efectuar los desbloqueos cuando hay varios hilos candidatos a ser desbloqueados.
- La estructura de los métodos será la que hemos usado en la clase anterior:

```
public tipo accion (...) {
  mutex.enter();
  if (! <CPRE>) { quecondition.await() };
  // si estamos aqui es que se cumple la CPRE
  <codigo que establece la POST>
  // podemos desbloquear a otro hilo?
  somecondition.signal();
  mutex.leave();
  // opcionalmente:
  return (<lo que sea>);
}
```

POLITÉCNICA

introducción/motivación

hacia un código de sincronización simple y trazable

- Nuestro objetivo es obtener soluciones a problemas de concurrencia que resultarían muy difíciles con semáforos.
- Pero ya hemos visto que las colas de monitores y *conditions* pueden interactuar de maneras no triviales.
- La solución que proponemos es usar los monitores de una manera restringida, con una traducción sencilla y trazable de las especificaciones de recurso.
- Más concretamente, tendremos que decidir: 1.- una estructura general para los métodos que implementan acciones de un recurso compartido, 2.- qué condition queues se necesitan para los bloqueos y 3.- una estrategia para efectuar los desbloqueos cuando hay varios hilos candidatos a ser desbloqueados.
- La estructura de los métodos será la que hemos usado en la clase anterior:

```
public tipo accion (...) {
  mutex.enter();
  if (! <CPRE>) { quecondition.await() };
  // si estamos aqui es que se cumple la CPRE
  <codigo que establece la POST>
  // podemos desbloquear a otro hilo?
  somecondition.signal();
  mutex.leave();
  // opcionalmente:
  return (<lo que sea>);
}
```

POLITÉCNICA

introducción/motivación

hacia un código de sincronización simple y trazable

- Nuestro objetivo es obtener soluciones a problemas de concurrencia que resultarían muy difíciles con semáforos.
- Pero ya hemos visto que las colas de monitores y *conditions* pueden interactuar de maneras no triviales.
- La solución que proponemos es usar los monitores de una manera restringida, con una traducción sencilla y trazable de las especificaciones de recurso.
- Más concretamente, tendremos que decidir: 1.- una estructura general para los métodos que implementan acciones de un recurso compartido, 2.- qué condition queues se necesitan para los bloqueos y 3.- una estrategia para efectuar los desbloqueos cuando hay varios hilos candidatos a ser desbloqueados.
- La estructura de los métodos será la que hemos usado en la clase anterior:

```
public tipo accion (...) {
  mutex.enter();
  if (! <CPRE>) { quecondition.await() };
  // si estamos aqui es que se cumple la CPRE
  <codigo que establece la POST>
  // podemos desbloquear a otro hilo?
  somecondition.signal();
  mutex.leave();
  // opcionalmente:
  return (<lo que sea>);
}
```

POLITÉCNICA

generando el segmento de bloqueos

- Para la parte que va entre el mutex.enter() y el código de la acción propiamente dicha nos interesa en primer lugar que sólo se ejecute como mucho un await(). Por simplicidad de código y porque hemos visto que los bucles de comprobación de CPREs pueden afectar a la eficiencia cuando hay muchos hilos.
- Por tanto, la idea aquí es evaluar la CPRE y si esta no se cumple escoger una condition sobre la cual hacer await(). Después de este código supondremos que la CPRE se cumple y se puede proceder al código de la acción.
- Análisis de las CPREs para decidir las conditions a usar en cada método. Tendremos varios casos posibles:
  - CPRE a Cierto No se necesita condition.
  - CPRE distinta de Cierto pero no depende de parámetros de la acción Una condition.
  - CPRE distinta de Cierto y depende de uno o varios parámetros de la acción
    - Los parámetros pueden tener un rango finito de valores Tendremos una condition para cada posible valor de los parámetros.
    - Los parámetros pueden tener un rango infinito de valores Tendremos una condition para cada valor que aparezca en la invocación de llenos.
- El objetivo de este análisis es poder determinar, dada una condition, la CPRE que provocó que se bloquearan los procesos encolados en ella.

POLITÉCNICA

generando el segmento de bloqueos

- Para la parte que va entre el mutex.enter() y el código de la acción propiamente dicha nos interesa en primer lugar que sólo se ejecute como mucho un await(). Por simplicidad de código y porque hemos visto que los bucles de comprobación de CPREs pueden afectar a la eficiencia cuando hay muchos hilos.
- Por tanto, la idea aquí es evaluar la CPRE y si esta no se cumple escoger una condition sobre la cual hacer await(). Después de este código supondremos que la CPRE se cumple y se puede proceder al código de la acción.
- Análisis de las CPREs para decidir las conditions a usar en cada método. Tendremos varios casos posibles:
  - CPRE a Cierto No se necesita condition.
  - CPRE distinta de Cierto pero no depende de parámetros de la acción Una condition.
  - CPRE distinta de Cierto y depende de uno o varios parámetros de la acción
    - Los parámetros pueden tomar un número "pequeño" de valores Indexación de parámetros.
    - Los parámetros pueden tomar un número "grande" de valores o variar en tiempo de ejecución Indexación de clientes.
- El objetivo de este análisis es poder determinar, dada una condition, la CPRE que provocó que se bloquearan los procesos encolados en ella.

POLITÉCNICA

generando el segmento de bloqueos

- Para la parte que va entre el mutex.enter() y el código de la acción propiamente dicha nos interesa en primer lugar que sólo se ejecute como mucho un await(). Por simplicidad de código y porque hemos visto que los bucles de comprobación de CPREs pueden afectar a la eficiencia cuando hay muchos hilos.
- Por tanto, la idea aquí es evaluar la CPRE y si esta no se cumple escoger una condition sobre la cual hacer await(). Después de este código supondremos que la CPRE se cumple y se puede proceder al código de la acción.
- Análisis de las CPREs para decidir las conditions a usar en cada método. Tendremos varios casos posibles:
  - CPRE a Cierto No se necesita condition.
  - CPRE distinta de Cierto pero no depende de parámetros de la acción Una condition.
  - CPRE distinta de Cierto y depende de uno o varios parámetros de la acción
    - Los parámetros pueden tomar un número "pequeño" de valores Indexación de parámetros.
    - Los parámetros pueden tomar un número "grande" de valores o variar en tiempo de ejecución Indexación de clientes.
- El objetivo de este análisis es poder determinar, dada una condition, la CPRE que provocó que se bloquearan los procesos encolados en ella.

POLITÉCNICA

generando el segmento de bloqueos

- Para la parte que va entre el mutex.enter() y el código de la acción propiamente dicha nos interesa en primer lugar que sólo se ejecute como mucho un await(). Por simplicidad de código y porque hemos visto que los bucles de comprobación de CPREs pueden afectar a la eficiencia cuando hay muchos hilos.
- Por tanto, la idea aquí es evaluar la CPRE y si esta no se cumple escoger una condition sobre la cual hacer await(). Después de este código supondremos que la CPRE se cumple y se puede proceder al código de la acción.
- Análisis de las CPREs para decidir las conditions a usar en cada método. Tendremos varios casos posibles:

  CPRE a Cierto No se necesita condition.

  CPRE distinta de Cierto pero no depende de parámetros de la acción Una condition.

  CPRE distinta de Cierto y depende de uno o varios parámetros de la acción

  - Los parámetros pueden tomar un número "pequeño" de valores Indexación de parámetros.
  - Los parámetros pueden tomar un número "grande" de valores o variar en tiempo de ejecución Indexación de clientes.

- El objetivo de este análisis es poder determinar, dada una condition, la CPRE que provocó que se bloquearan los procesos encolados en ella.

POLITÉCNICA

generando el segmento de bloqueos

- Para la parte que va entre el mutex.enter() y el código de la acción propiamente dicha nos interesa en primer lugar que sólo se ejecute como mucho un await(). Por simplicidad de código y porque hemos visto que los bucles de comprobación de CPREs pueden afectar a la eficiencia cuando hay muchos hilos.
- Por tanto, la idea aquí es evaluar la CPRE y si esta no se cumple escoger una condition sobre la cual hacer await(). Después de este código supondremos que la CPRE se cumple y se puede proceder al código de la acción.
- Análisis de las CPREs para decidir las conditions a usar en cada método. Tendremos varios casos posibles:

  CPRE a Cierto No se necesita condition.

  CPRE distinta de Cierto pero no depende de parámetros de la acción Una condition.

  CPRE distinta de Cierto y depende de uno o varios parámetros de la acción

  Los parámetros pueden tomar un número "pequeño" de valores Indexación de parámetros.

  Los parámetros pueden tomar un número "grande" de valores o variar en tiempo de ejecución Indexación de clientes.

- El objetivo de este análisis es poder determinar, dada una condition, la CPRE que provocó que se bloquearan los procesos encolados en ella.

POLITÉCNICA

generando el segmento de bloqueos

- Para la parte que va entre el mutex.enter() y el código de la acción propiamente dicha nos interesa en primer lugar que sólo se ejecute como mucho un await(). Por simplicidad de código y porque hemos visto que los bucles de comprobación de CPREs pueden afectar a la eficiencia cuando hay muchos hilos.
- Por tanto, la idea aquí es evaluar la CPRE y si esta no se cumple escoger una condition sobre la cual hacer await(). Después de este código supondremos que la CPRE se cumple y se puede proceder al código de la acción.
- Análisis de las CPREs para decidir las conditions a usar en cada método. Tendremos varios casos posibles:

  CPRE a Cierto No se necesita condition.

  CPRE distinta de Cierto pero no depende de parámetros de la acción Una condition.

  CPRE distinta de Cierto y depende de uno o varios parámetros de la acción

  Los parámetros pueden tomar un número "pequeño" de valores Indexación de parámetros.

  Los parámetros pueden tomar un número "grande" de valores o variar en tiempo de ejecución Indexación de clientes.

- El objetivo de este análisis es poder determinar, dada una condition, la CPRE que provocó que se bloquearan los procesos encolados en ella.

POLITÉCNICA

generando el segmento de bloqueos

- Para la parte que va entre el mutex.enter() y el código de la acción propiamente dicha nos interesa en primer lugar que sólo se ejecute como mucho un await(). Por simplicidad de código y porque hemos visto que los bucles de comprobación de CPREs pueden afectar a la eficiencia cuando hay muchos hilos.
- Por tanto, la idea aquí es evaluar la CPRE y si esta no se cumple escoger una condition sobre la cual hacer await(). Después de este código supondremos que la CPRE se cumple y se puede proceder al código de la acción.
- Análisis de las CPREs para decidir las conditions a usar en cada método. Tendremos varios casos posibles:

  CPRE a Cierto No se necesita condition.

  CPRE distinta de Cierto pero no depende de parámetros de la acción Una condition.

  CPRE distinta de Cierto y depende de uno o varios parámetros de la acción

  - Los parámetros pueden tomar un número "pequeño" de valores Indexación de parámetros.
  - Los parámetros pueden tomar un número "grande" de valores o variar en tiempo de ejecución Indexación de clientes.

- El objetivo de este análisis es poder determinar, dada una condition, la CPRE que provocó que se bloquearan los procesos encolados en ella.

POLITÉCNICA

generando el segmento de bloqueos

- Para la parte que va entre el mutex.enter() y el código de la acción propiamente dicha nos interesa en primer lugar que sólo se ejecute como mucho un await(). Por simplicidad de código y porque hemos visto que los bucles de comprobación de CPREs pueden afectar a la eficiencia cuando hay muchos hilos.
- Por tanto, la idea aquí es evaluar la CPRE y si esta no se cumple escoger una condition sobre la cual hacer await(). Después de este código supondremos que la CPRE se cumple y se puede proceder al código de la acción.
- Análisis de las CPREs para decidir las conditions a usar en cada método. Tendremos varios casos posibles:

  CPRE a Cierto No se necesita condition.

  CPRE distinta de Cierto pero no depende de parámetros de la acción Una condition.

  CPRE distinta de Cierto y depende de uno o varios parámetros de la acción

  - Los parámetros pueden tomar un número "pequeño" de valores Indexación de parámetros.
  - Los parámetros pueden tomar un número "grande" de valores o variar en tiempo de ejecución Indexación de clientes.

- El objetivo de este análisis es poder determinar, dada una condition, la CPRE que provocó que se bloquearan los procesos encolados en ella.

POLITÉCNICA

generando el segmento de bloqueos

- Para la parte que va entre el mutex.enter() y el código de la acción propiamente dicha nos interesa en primer lugar que sólo se ejecute como mucho un await(). Por simplicidad de código y porque hemos visto que los bucles de comprobación de CPREs pueden afectar a la eficiencia cuando hay muchos hilos.
- Por tanto, la idea aquí es evaluar la CPRE y si esta no se cumple escoger una condition sobre la cual hacer await(). Después de este código supondremos que la CPRE se cumple y se puede proceder al código de la acción.
- Análisis de las CPREs para decidir las conditions a usar en cada método. Tendremos varios casos posibles:

  CPRE a Cierto No se necesita condition.

  CPRE distinta de Cierto pero no depende de parámetros de la acción Una condition.

  CPRE distinta de Cierto y depende de uno o varios parámetros de la acción

  - Los parámetros pueden tomar un número "pequeño" de valores Indexación de parámetros.
  - Los parámetros pueden tomar un número "grande" de valores o variar en tiempo de ejecución Indexación de clientes.

- El objetivo de este análisis es poder determinar, dada una condition, la CPRE que provocó que se bloquearan los procesos encolados en ella.

POLITÉCNICA

generando el segmento de desbloqueos

- Para la parte del código que se ejecuta después de la acción propiamente dicha, además de simplicidad nos interesa que se cumplan las dos propiedades siguientes:

  Seguridad Si se desbloquea un hilo mediante un `signal()` es porque garantizamos que se cumple su CPRE en ese momento.

  Viveza Si hay uno o más hilos bloqueados cuya CPRE se cumple, es obligatorio desbloquear a uno de ellos.

- La propiedad de seguridad es mucho más fácil de asegurar si como máximo ejecutamos un `signal()`:

POLITÉCNICA

generando el segmento de desbloqueos

- Para la parte del código que se ejecuta después de la acción propiamente dicha, además de simplicidad nos interesa que se cumplan las dos propiedades siguientes:

  Seguridad Si se desbloquea un hilo mediante un signal () es porque garantizamos que se cumple su CPRE en ese momento.

  Viveza Si hay uno o más hilos bloqueados cuya CPRE se cumple, es obligatorio desbloquear a uno de ellos.

- La propiedad de seguridad es mucho más fácil de asegurar si como máximo ejecutamos un signal () :

POLITÉCNICA

generando el segmento de desbloqueos

- Para la parte del código que se ejecuta después de la acción propiamente dicha, además de simplicidad nos interesa que se cumplan las dos propiedades siguientes:

  Seguridad Si se desbloquea un hilo mediante un signal () es porque garantizamos que se cumple su CPRE en ese momento.
  Viveza Si hay uno o más hilos bloqueados cuya CPRE se cumple, es obligatorio desbloquear a uno de ellos.

- La propiedad de seguridad es mucho más fácil de asegurar si como máximo ejecutamos un signal () :

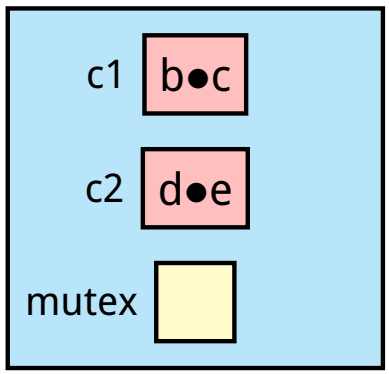
POLITÉCNICA

generando el segmento de desbloqueos

- Para la parte del código que se ejecuta después de la acción propiamente dicha, además de simplicidad nos interesa que se cumplan las dos propiedades siguientes:

  Seguridad Si se desbloquea un hilo mediante un signal() es porque garantizamos que se cumple su CPRE en ese momento.
  Viveza Si hay uno o más hilos bloqueados cuya CPRE se cumple, es obligatorio desbloquear a uno de ellos.

- La propiedad de seguridad es mucho más fácil de asegurar si como máximo ejecutamos un signal():



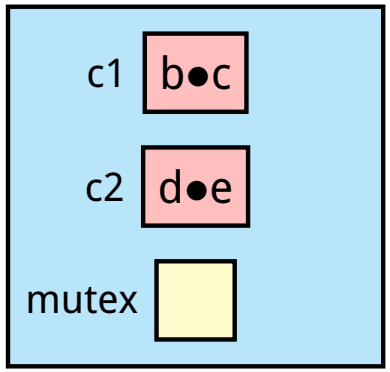
POLITÉCNICA

generando el segmento de desbloqueos

- Para la parte del código que se ejecuta después de la acción propiamente dicha, además de simplicidad nos interesa que se cumplan las dos propiedades siguientes:

  Seguridad Si se desbloquea un hilo mediante un signal () es porque garantizamos que se cumple su CPRE en ese momento.
  Viveza Si hay uno o más proceso bloqueados cuya CPRE se cumple, es obligatorio desbloquear a uno de ellos.

- La propiedad de seguridad es mucho más fácil de asegurar si como máximo ejecutamos un signal () :



a:c1.signal();

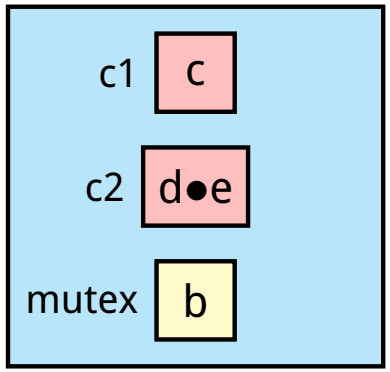
POLITÉCNICA

generando el segmento de desbloqueos

- Para la parte del código que se ejecuta después de la acción propiamente dicha, además de simplicidad nos interesa que se cumplan las dos propiedades siguientes:

  Seguridad Si se desbloquea un hilo mediante un signal () es porque garantizamos que se cumple su CPRE en ese momento.
  Viveza Si hay uno o más proceso bloqueados cuya CPRE se cumple, es obligatorio desbloquear a uno de ellos.

- La propiedad de seguridad es mucho más fácil de asegurar si como máximo ejecutamos un signal () :



a:c1.signal();

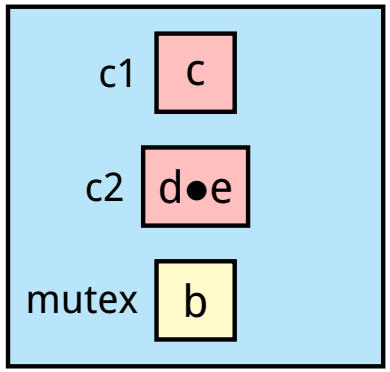
POLITÉCNICA

generando el segmento de desbloqueos

- Para la parte del código que se ejecuta después de la acción propiamente dicha, además de simplicidad nos interesa que se cumplan las dos propiedades siguientes:

  Seguridad Si se desbloquea un hilo mediante un signal () es porque garantizamos que se cumple su CPRE en ese momento.
  Viveza Si hay uno o más proceso bloqueados cuya CPRE se cumple, es obligatorio desbloquear a uno de ellos.

- La propiedad de seguridad es mucho más fácil de asegurar si como máximo ejecutamos un signal () :




```
a:c1.signal();
a:c2.signal();
```

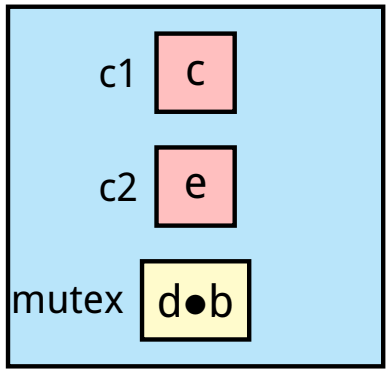
POLITÉCNICA

generando el segmento de desbloqueos

- Para la parte del código que se ejecuta después de la acción propiamente dicha, además de simplicidad nos interesa que se cumplan las dos propiedades siguientes:

  Seguridad Si se desbloquea un hilo mediante un signal () es porque garantizamos que se cumple su CPRE en ese momento.
  Viveza Si hay uno o más proceso bloqueados cuya CPRE se cumple, es obligatorio desbloquear a uno de ellos.

- La propiedad de seguridad es mucho más fácil de asegurar si como máximo ejecutamos un signal () :




```
a:c1.signal();
a:c2.signal();
```

POLITÉCNICA

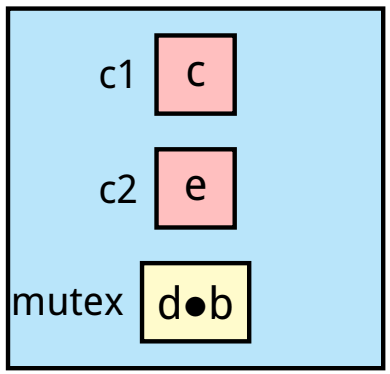
generando el segmento de desbloqueos

- Para la parte del código que se ejecuta después de la acción propiamente dicha, además de simplicidad nos interesa que se cumplan las dos propiedades siguientes:

  Seguridad Si se desbloquea un hilo mediante un signal() es porque garantizamos que se cumple su CPRE en ese momento.

  Viveza Si hay uno o más proceso bloqueados cuya CPRE se cumple, es obligatorio desbloquear a uno de ellos.

- La propiedad de seguridad es mucho más fácil de asegurar si como máximo ejecutamos un signal():



a:c1.signal();
a:c2.signal();

- Llamamos regla 0/1 a esta idea de que cada proceso ejecuta uno o ningún await() y uno o ningún signal() entre el mutex.enter() y el mutex.leave().
- Para la propiedad de viveza necesitamos asegurarnos de que el signal() va a ser efectivo, es decir que no se hace sobre una condition vacía. Para esto es útil el método waiting() de la clase Monitor.Cond.

POLITÉCNICA

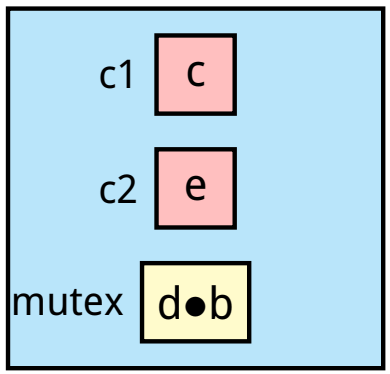
generando el segmento de desbloqueos

- Para la parte del código que se ejecuta después de la acción propiamente dicha, además de simplicidad nos interesa que se cumplan las dos propiedades siguientes:

  Seguridad Si se desbloquea un hilo mediante un signal() es porque garantizamos que se cumple su CPRE en ese momento.

  Viveza Si hay uno o más proceso bloqueados cuya CPRE se cumple, es obligatorio desbloquear a uno de ellos.

- La propiedad de seguridad es mucho más fácil de asegurar si como máximo ejecutamos un signal():



a:c1.signal();
a:c2.signal();

- Llamamos regla 0/1 a esta idea de que cada proceso ejecuta uno o ningún await() y uno o ningún signal() entre el mutex.enter() y el mutex.leave().
- Para la propiedad de viveza necesitamos asegurarnos de que el signal() va a ser efectivo, es decir que no se hace sobre una condition vacía. Para esto es útil el método waiting() de la clase Monitor.Cond.

POLITÉCNICA

ejemplo: lectores & escritores con monitores

la solución más simple posible

```
public class GestorLE_Mon {
    private int nLect;
    private boolean esc;
    private Monitor mutex = new Monitor();
    private Monitor.Cond c_espEsc  =
        mutex.newCond();
    private Monitor.Cond c_espLeer =
        mutex.newCond();

    public GestorLE_Mon() {
        nLect = 0;
        esc   = false;
    }
    public void iniciar_lectura() {
        mutex.enter();
        if(esc)
            c_espLeer.await();
        nLect++;
        desbloqueo_generico();
        mutex.leave();
    }
    public void terminar_lectura() {
        mutex.enter();
        nLect--;
        desbloqueo_generico();
        mutex_leave();
    }
```

```
    private void desbloqueo_generico() {
        if (nLect == 0 && !esc) {
            if (c_espEsc.waiting() > 0)
                c_espEsc.signal()
            else
                c_espLeer.signal();
        } else if (!esc) {
            c_espLeer.signal();
        }
    }
    public void iniciar_escritura() {
        mutex.enter();
        if (esc || nLect > 0)
            c_espEsc.await();
        esc = true;
        desbloqueo_generico();
        mutex.leave();
    }
    public void terminar_escritura() {
        mutex.enter();
        esc = false;
        desbloqueo_generico();
        mutex.leave();
} }
```

POLITÉCNICA

ejemplo: lectores & escritores con monitores

la solución más simple posible

```
public class GestorLE_Mon {
    private int nLect;
    private boolean esc;
    private Monitor mutex = new Monitor();
    private Monitor.Cond c_espEsc  =
        mutex.newCond();
    private Monitor.Cond c_espLeer =
        mutex.newCond();
    public GestorLE_Mon() {
        nLect = 0;
        esc   = false;
    }
    public void iniciar_lectura() {
        mutex.enter();
        if(esc)
            c_espLeer.await();
        nLect++;
        desbloqueo_generico();
        mutex.leave();
    }
    public void terminar_lectura() {
        mutex.enter();
        nLect--;
        desbloqueo_generico();
        mutex_leave();
    }
    private void desbloqueo_generico() {
        if (nLect == 0 && !esc) {
            if (c_espEsc.waiting() > 0)
                c_espEsc.signal()
            else
                c_espLeer.signal();
        } else if (!esc) {
            c_espLeer.signal();
        }
    }
    public void iniciar_escritura() {
        mutex.enter();
        if (esc || nLect > 0)
            c_espEsc.await();
        esc = true;
        desbloqueo_generico();
        mutex.leave();
    }
    public void terminar_escritura() {
        mutex.enter();
        esc = false;
        desbloqueo_generico();
        mutex.leave();
} }
```

POLITÉCNICA

ejemplo: lectores & escritores con monitores

la solución más simple posible

```
public class GestorLE_Mon {
    private int nLect;
    private boolean esc;
    private Monitor mutex = new Monitor();
    private Monitor.Cond c_espEsc  =
        mutex.newCond();
    private Monitor.Cond c_espLeer =
        mutex.newCond();
    public GestorLE_Mon() {
        nLect = 0;
        esc   = false;
    }
    public void iniciar_lectura() {
        mutex.enter();
        if(esc)
            c_espLeer.await();
        nLect++;
        desbloqueo_generico();
        mutex.leave();
    }
    public void terminar_lectura() {
        mutex.enter();
        nLect--;
        desbloqueo_generico();
        mutex_leave();
    }
```

```
    private void desbloqueo_generico() {
        if (nLect == 0 && !esc) {
            if (c_espEsc.waiting() > 0)
                c_espEsc.signal()
            else
                c_espLeer.signal();
        } else if (!esc) {
            c_espLeer.signal();
        }
    }
    public void iniciar_escritura() {
        mutex.enter();
        if (esc || nLect > 0)
            c_espEsc.await();
        esc = true;
        desbloqueo_generico();
        mutex.leave();
    }
    public void terminar_escritura() {
        mutex.enter();
        esc = false;
        desbloqueo_generico();
        mutex.leave();
} }
```

POLITÉCNICA

ejemplo: lectores & escritores con monitores

la solución más simple posible

```
public class GestorLE_Mon {
    private int nLect;
    private boolean esc;
    private Monitor mutex = new Monitor();
    private Monitor.Cond c_espEsc  =
        mutex.newCond();
    private Monitor.Cond c_espLeer =
        mutex.newCond();
    public GestorLE_Mon() {
        nLect = 0;
        esc   = false;
    }
    public void iniciar_lectura() {
        mutex.enter();
        if(esc)
            c_espLeer.await();
        nLect++;
        desbloqueo_generico();
        mutex.leave();
    }
    public void terminar_lectura() {
        mutex.enter();
        nLect--;
        desbloqueo_generico();
        mutex_leave();
    }
```

```
private void desbloqueo_generico() {
    if (nLect == 0 && !esc) {
        if (c_espEsc.waiting() > 0)
            c_espEsc.signal()
        else
            c_espLeer.signal();
    } else if (!esc) {
        c_espLeer.signal();
    }
}
public void iniciar_escritura() {
    mutex.enter();
    if (esc || nLect > 0)
        c_espEsc.await();
    esc = true;
    desbloqueo_generico();
    mutex.leave();
}
public void terminar_escritura() {
    mutex.enter();
    esc = false;
    desbloqueo_generico();
    mutex.leave();
} }
```

POLITÉCNICA

ejemplo: lectores & escritores con monitores

la solución más simple posible

```
public class GestorLE_Mon {
    private int nLect;
    private boolean esc;
    private Monitor mutex = new Monitor();
    private Monitor.Cond c_espEsc  =
        mutex.newCond();
    private Monitor.Cond c_espLeer =
        mutex.newCond();
    public GestorLE_Mon() {
        nLect = 0;
        esc   = false;
    }
    public void iniciar_lectura() {
        mutex.enter();
        if(esc)
            c_espLeer.await();
        nLect++;
        desbloqueo_generico();
        mutex.leave();
    }
    public void terminar_lectura() {
        mutex.enter();
        nLect--;
        desbloqueo_generico();
        mutex_leave();
    }
```

```
private void desbloqueo_generico() {
    if (nLect == 0 && !esc) {
        if (c_espEsc.waiting() > 0)
            c_espEsc.signal()
        else
            c_espLeer.signal();
    } else if (!esc) {
        c_espLeer.signal();
    }
}
public void iniciar_escritura() {
    mutex.enter();
    if (esc || nLect > 0)
        c_espEsc.await();
    esc = true;
    desbloqueo_generico();
    mutex.leave();
}
public void terminar_escritura() {
    mutex.enter();
    esc = false;
    desbloqueo_generico();
    mutex.leave();
} }
```

POLITÉCNICA

ejemplo: lectores & escritores con monitores

la solución más simple posible

```
public class GestorLE_Mon {
    private int nLect;
    private boolean esc;
    private Monitor mutex = new Monitor();
    private Monitor.Cond c_espEsc  =
        mutex.newCond();
    private Monitor.Cond c_espLeer =
        mutex.newCond();
    public GestorLE_Mon() {
        nLect = 0;
        esc   = false;
    }
    public void iniciar_lectura() {
        mutex.enter();
        if(esc)
            c_espLeer.await();
        nLect++;
        desbloqueo_generico();
        mutex.leave();
    }
    public void terminar_lectura() {
        mutex.enter();
        nLect--;
        desbloqueo_generico();
        mutex_leave();
    }
```

```
private void desbloqueo_generico() {
    if (nLect == 0 && !esc) {
        if (c_espEsc.waiting() > 0)
            c_espEsc.signal()
        else
            c_espLeer.signal();
    } else if (!esc) {
        c_espLeer.signal();
    }
}
public void iniciar_escritura() {
    mutex.enter();
    if (esc || nLect > 0)
        c_espEsc.await();
    esc = true;
    desbloqueo_generico();
    mutex.leave();
}
public void terminar_escritura() {
    mutex.enter();
    esc = false;
    desbloqueo_generico();
    mutex.leave();
} }
```

POLITÉCNICA