Demostración de Propiedades de Algoritmo de Exclusión Mutua

Sistemas Concurrentes y Distribuidos

Ismael Sallami Moreno
Ingenería Informática + ADE
Universidad de Granada (UGR)

6 de diciembre de 2024

Índice

1 Enunciado

Supongamos un conjunto de n procesos distribuidos y conectados en anillo. El algoritmo de exclusión mutua se basa en las siguientes reglas:

- Tenemos los procesos $0, \ldots, n-1$. Cada proceso i tiene una variable $s[i]$, inicializada a 0, que puede tomar los valores 0 o 1.
- Para que un proceso i pueda entrar en la sección crítica (SC), deben cumplirse las siguientes condiciones:
  1. Si $i > 0$: $s[i] \neq s[i-1]$.
  2. Si $i = 0$: $s[0] = s[n-1]$.
- Al salir de la sección crítica, el proceso i ejecuta:
  1. Si $i > 0$: $s[i] := s[i-1]$, siempre que antes de la asignación $s[i] \neq s[i-1]$.
  2. Si $i = 0$: $s[0] := (s[0] + 1) \mod 2$.

Se pide:

1. Demostrar que el algoritmo permite únicamente a un proceso acceder a la sección crítica en cualquier configuración.
2. Demostrar que cada proceso $p[i]$ conseguirá entrar en la sección crítica infinitamente a menudo.

2 Demostración

2.1. Parte I: Demostrar que el algoritmo garantiza exclusión mutua

La exclusión mutua significa que, en cualquier configuración del sistema, como máximo un proceso puede estar en su sección crítica (SC) en un momento dado. A continuación, se analiza cómo las reglas del algoritmo garantizan esta propiedad.

2.1.1. Condiciones para entrar en la sección crítica

- Para $i > 0$: $s[i] \neq s[i-1]$. Esto implica que un proceso i solo puede entrar a la SC si su variable $s[i]$ es distinta de la de su vecino inmediato anterior $s[i-1]$. Por lo tanto:
  - Si un proceso i está en la SC, esto obliga a que $s[i] \neq s[i-1]$.
  - Su vecino inmediato anterior $i-1$, que también depende de $s[i-1]$, no podrá entrar a la SC al mismo tiempo porque para $i-1$ entrar se requeriría $s[i-1] \neq s[i-2]$, lo que sería inconsistente con la condición de i.

- Para $i = 0$: $s[0] = s[n-1]$. El proceso 0 solo puede entrar si su variable $s[0]$ coincide con la del último proceso $s[n-1]$. Esto asegura la sincronización entre el principio y el final del anillo. Si 0 está en la SC, ningún otro proceso puede cumplir las condiciones necesarias, porque las relaciones $s[i] \neq s[i-1]$ o $s[0] = s[n-1]$ no se mantendrían simultáneamente.

Por lo tanto, estas condiciones aseguran que, como máximo, un proceso puede satisfacer las reglas de entrada a la SC en cualquier momento.

2.1.2. Condiciones al salir de la sección crítica

Cuando un proceso i termina de ejecutar su sección crítica, realiza las siguientes actualizaciones:

- Para $i > 0$: $s[i] := s[i-1]$, solo si $s[i] \neq s[i-1]$. Al asignar $s[i]$ el valor de $s[i-1]$, el proceso i "cede el paso" a su vecino. Esto significa que, después de que i sale de la SC:
  - Su estado $s[i]$ se iguala al de $s[i-1]$, deshabilitando inmediatamente su propia entrada a la SC.
  - Esto permite que otros procesos cumplan las condiciones necesarias para acceder a la SC.
- Para $i = 0$: $s[0] := (s[0] + 1) \mod 2$. Al incrementar cíclicamente $s[0]$, el proceso 0 cambia el estado global del anillo, lo que obliga a que las condiciones de entrada se actualicen para otros procesos.

Estas actualizaciones aseguran que:

1. Un proceso no permanezca en la SC indefinidamente.
2. Los estados del sistema se actualicen de forma consistente para permitir que otros procesos entren en la SC.

2.1.3. Exclusión mutua en el anillo

Las condiciones de entrada y salida interactúan de la siguiente manera para garantizar exclusión mutua:

1. **Vecinos no pueden entrar simultáneamente:** La condición $s[i] \neq s[i-1]$ asegura que dos procesos consecutivos en el anillo no pueden entrar a la SC al mismo tiempo. Por ejemplo, si el proceso i entra en la SC, entonces $s[i] \neq s[i-1]$. Esto implica que el vecino $i-1$ no puede cumplir la condición para entrar en la SC al mismo tiempo.
2. **Inicio y fin del anillo sincronizados:** Para $i = 0$, la condición $s[0] = s[n-1]$ asegura que el proceso 0 solo puede entrar en la SC cuando está sincronizado con el estado del último proceso $n-1$. Esto evita conflictos en los extremos del anillo.
3. **Imposibilidad de más de un acceso:** Supongamos que dos procesos distintos, i y j, intentan entrar en la SC simultáneamente.

- Si $i > 0$ y $j > 0$, ambos procesos requerirían que $s[i] \neq s[i-1]$ y $s[j] \neq s[j-1]$, lo que es imposible porque sus estados están interdependientes y los valores $s[k]$ forman un único conjunto de valores en el anillo.
- Si $i = 0$, entonces $s[0] = s[n-1]$ debe cumplirse, pero esto es inconsistente con las condiciones $s[1] \neq s[0]$ necesarias para que otros procesos entren simultáneamente.

2.1.4. El diseño del algoritmo asegura que:

- Cada proceso depende exclusivamente de su propio estado $s[i]$ y el de su vecino inmediato ($s[i-1]$ o $s[n-1]$ para $i = 0$).
- Las condiciones de entrada y salida están sincronizadas para evitar conflictos.
- En cualquier configuración, solo un proceso puede cumplir las condiciones de entrada, lo que garantiza **exclusión mutua**.

Por lo que queda demostrado que el algoritmo asegura la Exclusión Mutua.

2.2. Parte II: Demostrar que cada proceso entra a la sección crítica infinitamente a menudo

El objetivo es demostrar que el algoritmo garantiza que ningún proceso queda bloqueado indefinidamente y que cada proceso puede entrar en su sección crítica (SC) un número infinito de veces.

2.2.1. Progreso del sistema

El progreso significa que, en cualquier configuración válida, siempre habrá al menos un proceso que cumpla las condiciones necesarias para entrar en la SC. Esto se garantiza por las siguientes razones:

1. **Dependencia local de las condiciones de entrada:** Cada proceso i puede decidir entrar en la SC únicamente basándose en el valor de $s[i]$ y $s[i-1]$ (o $s[n-1]$ si $i = 0$). Por lo tanto:
   - La decisión de un proceso i no depende de los valores de procesos distantes, lo que permite que las decisiones se tomen de forma local y eficiente, es decir, depende de sus vecinos inmediatos.
   - Esto implica que el estado global del anillo no bloquea el progreso de ningún proceso en particular.
2. **Actualización consistente de los valores de $s[i]$:** Cuando un proceso i sale de la SC, actualiza su variable $s[i]$ de acuerdo con el protocolo:
   - Si $i > 0$: $s[i] := s[i-1]$, lo que sincroniza el estado del proceso con su vecino anterior y permite que $i+1$ cumpla las condiciones para entrar.

- Si $i = 0$: $s[0] := (s[0] + 1) \mod 2$, lo que altera el estado global del anillo, asegurando que las condiciones para otros procesos cambien eventualmente.

Estas actualizaciones garantizan que los valores de $s[i]$ evolucionen de manera predecible, permitiendo que los procesos se turnen para entrar en la SC.

3. **Anillo cerrado y cíclico:** El diseño del anillo asegura que todos los procesos están conectados en una estructura cíclica. Si un proceso i no puede entrar en la SC debido a su vecino, ese vecino eventualmente actualizará su estado y permitirá que i entre. Por ejemplo, si $s[i] = s[i-1]$, el vecino $i-1$ debe entrar en la SC antes de que $s[i-1]$ cambie, lo que desbloqueará i.

En conjunto, estas propiedades garantizan que siempre haya al menos un proceso con las condiciones necesarias para entrar en la SC, asegurando progreso continuo.

2.2.2. Acceso infinito a la sección crítica

Ningún proceso puede quedar bloqueado indefinidamente, y cada proceso i podrá entrar en la SC un número infinito de veces.

Esto podemos garantizarlo en base a las siguientes razones:

1. **Evolución de las condiciones de entrada:** Las actualizaciones de $s[i]$ al salir de la SC aseguran que las condiciones necesarias para que otros procesos entren se modifiquen periódicamente. Por ejemplo:

   - Cuando un proceso i actualiza $s[i] := s[i-1]$, su vecino $i+1$ tiene una mayor probabilidad de cumplir $s[i+1] \neq s[i]$ en el futuro.
   - De manera similar, cuando $i = 0$, la actualización $s[0] := (s[0] + 1) \mod 2$ modifica cíclicamente las condiciones del anillo.

   Estas modificaciones aseguran que las condiciones necesarias para entrar en la SC no permanezcan estáticas, permitiendo que todos los procesos tengan una oportunidad justa de acceder.

2. **Estructura cíclica del anillo:** En un sistema de n procesos conectados en anillo:

   - Si el proceso i no puede entrar en la SC porque $s[i] = s[i-1]$, su vecino inmediato $i-1$ deberá entrar en la SC y actualizar $s[i-1]$ antes de que i pueda intentarlo de nuevo.
   - Este comportamiento garantiza que todos los procesos sean atendidos eventualmente, ya que no hay dependencia circular que pueda bloquear un proceso de manera indefinida.

3. **Periodicidad garantizada:** Debido a que $s[0]$ se actualiza cíclicamente como $(s[0] + 1) \mod 2$, las configuraciones del anillo eventualmente regresan a un estado similar al inicial. En cada ciclo completo, todos los procesos tienen la oportunidad de cumplir las condiciones necesarias para entrar en la SC. Este comportamiento asegura que cada proceso tendrá acceso a la SC un número infinito de veces.

2.2.3. Análisis combinando Progreso y Acceso Infinito a la Sección Crítica

1. **Progreso garantiza que el sistema no se detiene:** Siempre hay al menos un proceso que puede entrar en la SC. Cuando un proceso sale, actualiza las condiciones para permitir que otros procesos avancen.

2. **Justicia garantiza que nadie queda excluido:** Gracias a la estructura cíclica y las actualizaciones periódicas de $s[i]$, todos los procesos eventualmente tendrán una oportunidad de cumplir las condiciones para entrar en la SC.

3. **Interacción entre progreso y justicia:** Si un proceso i no puede entrar porque su vecino $i-1$ no ha actualizado $s[i-1]$, ese vecino eventualmente tendrá la oportunidad de entrar y actualizar $s[i-1]$. Esto evita bloqueos indefinidos y asegura que todos los procesos accedan a la SC un número infinito de veces.

Por lo tanto, se cumplen las propiedades de **progreso** y **acceso infinito a la sección crítica**, asegurando que ningún proceso quede bloqueado indefinidamente y que todos los procesos puedan acceder a la SC un número infinito de veces, por lo que esta propiedad queda demostrada.

2.3. Relación con los axiomas de la programación concurrente

El algoritmo cumple con los cinco axiomas fundamentales de la programación concurrente. A continuación, se describe cómo se satisfacen:

2.3.1. Atomicidad e Intercalación de Instrucciones

Este axioma establece que cada instrucción del programa concurrente se ejecuta de forma atómica y el resultado depende del orden en que las instrucciones se intercalan. En el contexto del algoritmo:

- **Atomicidad:** Las operaciones de lectura y escritura sobre las variables $s[i]$ son atómicas. Por ejemplo, al verificar la condición $s[i] \neq s[i-1]$ o al actualizar $s[i] := s[i-1]$, se garantiza que la operación se completa antes de que otro proceso acceda a las mismas variables.

- **Intercalación:** Los procesos avanzan en su ejecución intercalando las instrucciones, asegurando que sólo un proceso a la vez pueda acceder a la sección crítica. Esto se deriva de las condiciones de entrada ($s[i] \neq s[i-1]$ y $s[0] = s[n-1]$) y de las actualizaciones consistentes al salir de la sección crítica.

2.3.2. Consistencia de datos después del acceso concurrente

Este axioma requiere que el estado del sistema sea consistente después de cualquier acceso concurrente. En el algoritmo:

- La actualización de las variables $s[i]$ tras salir de la sección crítica garantiza que el sistema mantenga un estado consistente:

- Para $i > 0$: $s[i] := s[i-1]$ sincroniza el estado del proceso con su vecino anterior.
- Para $i = 0$: $s[0] := (s[0] + 1) \mod 2$ asegura que el estado global del anillo avance de manera predecible.

- Estas reglas evitan inconsistencias, como permitir que dos procesos cumplan simultáneamente las condiciones para entrar en la sección crítica.

2.3.3. Irrepetibilidad de las secuencias de instrucciones

Este axioma implica que una misma secuencia de instrucciones puede producir diferentes resultados en ejecuciones distintas debido a la concurrencia. En el algoritmo:

- Las decisiones para entrar en la sección crítica dependen del estado actual de las variables $s[i]$, que varía dinámicamente durante la ejecución del sistema.
- Por ejemplo, el proceso i puede cumplir $s[i] \neq s[i-1]$ en una configuración específica, pero no en otra, dependiendo de cómo los procesos vecinos actualicen sus valores $s[j]$.

Esta irrepetibilidad asegura que el comportamiento del sistema no sea rígido, adaptándose a las condiciones cambiantes del entorno concurrente.

2.3.4. Independencia de la velocidad de proceso

Este axioma establece que el sistema debe funcionar correctamente independientemente de las velocidades relativas de los procesos. En el algoritmo:

- La independencia de la velocidad se garantiza porque las condiciones de entrada a la sección crítica dependen únicamente de los estados locales ($s[i]$ y $s[i-1]$) y no de la velocidad con la que otros procesos se ejecutan.
- Incluso si un proceso avanza más rápido que los demás, no puede bloquear indefinidamente a otros procesos, ya que:
  - El diseño en anillo asegura que cada proceso eventualmente actualice su estado.
  - Las actualizaciones periódicas de $s[0]$ mediante $(s[0]+1) \mod 2$ permiten que las condiciones cambien para todos los procesos.

2.3.5. Hipótesis de progreso finito

Este axioma implica que cada proceso debe avanzar y alcanzar su objetivo (por ejemplo, entrar en la sección crítica) en un tiempo finito. En el algoritmo:

- **Progreso finito** se garantiza porque:
  - Las actualizaciones de $s[i]$ al salir de la sección crítica permiten que los procesos desbloqueen a sus vecinos.

    - La estructura cíclica del anillo asegura que las condiciones necesarias para entrar en la sección crítica evolucionen periódicamente, evitando bloqueos indefinidos.

- Cada proceso tiene garantizado el acceso infinito a la sección crítica gracias a la combinación de las propiedades de progreso (ningún proceso bloquea indefinidamente a otro) y justicia (todos los procesos tienen oportunidades equitativas de acceder a la sección crítica).

Podemos concluir con que el algoritmo cumple con los cinco axiomas de la programación concurrente.

2.4. Propiedades de vivacidad y seguridad

El algoritmo cumple con las dos propiedades fundamentales de los sistemas concurrentes: **vivacidad** y **seguridad**.

2.4.1. Vivacidad

La vivacidad asegura que el sistema no se bloquea indefinidamente y que cada proceso tiene garantizado el progreso. En este caso:

- Se garantiza que siempre hay al menos un proceso que puede entrar en la sección crítica debido a las condiciones dinámicas de entrada:
    - Para $i > 0$: $s[i] \neq s[i-1]$ asegura que un proceso solo depende de su vecino inmediato.
    - Para $i = 0$: $s[0] = s[n-1]$ garantiza que el anillo evoluciona cíclicamente.
- Las actualizaciones de $s[i]$ y la estructura en anillo aseguran que cada proceso tenga la oportunidad de entrar en la sección crítica después de que otros procesos completen la suya.
- Por lo tanto, no hay bloqueos ni inanición, cumpliéndose la propiedad de vivacidad.

2.4.2. Seguridad

La seguridad garantiza que nunca se violan las reglas del sistema, como permitir que más de un proceso acceda simultáneamente a la sección crítica. En este caso:

- Las condiciones de entrada a la sección crítica ($s[i] \neq s[i-1]$ para $i > 0$, y $s[0] = s[n-1]$ para $i = 0$) aseguran que solo un proceso pueda entrar en la sección crítica a la vez:
    - Si dos procesos intentaran entrar simultáneamente, las condiciones de entrada impedirían que ambos lo hagan, ya que al menos uno de ellos no cumpliría $s[i] \neq s[i-1]$.
- Las actualizaciones consistentes de $s[i]$ tras salir de la sección crítica evitan que otros procesos accedan simultáneamente, manteniendo la exclusión mutua.

- Por lo tanto, se cumple la propiedad de seguridad, garantizando que el acceso a la sección crítica sea exclusivo.

Podemos concluir que el algoritmo cumple con las propiedades de vivacidad y de seguridad.

3 Bibliografía

Referencias

[1] Diapositivas de la asignatura *Sistemas Concurrentes y Distribuidos*, en la Universidad de Granada.

[2] L. Lamport, *Time, Clocks, and the Ordering of Events in a Distributed System*, Communications of the ACM, vol. 21, no. 7, pp. 558-565, 1978.

[3] M. Ben-Ari, *Principles of Concurrent and Distributed Programming*, 2nd Edition, Addison-Wesley, 2006.

4 Anotaciones

Cabe destacar que la extensión de esta práctica se debe a la intención de utilizar la mayor cantidad posible de conceptos del temario, así como de proporcionar una demostración y explicación detallada de las propiedades de un algoritmo de exclusión mutua.