Administración de procesos: Bloqueos mutuos y políticas

Gunnar Wolf

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Índice

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Generalizando bloqueos mutuos

- Estudiamos ya varios casos de bloqueos mutuos al hablar de sincronización
- Pueden presentarse en varios otros entornos
  - De cómputo o de la vida real

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

El encuentro de dos trenes

Cuando dos trenes lleguen a un crucero, ambos deben detenerse por completo y no avanzar hasta que el otro se haya ido

Ley aprobada por el Estado de Kansas, principios del siglo XX

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

El cruce de un semáforo: ¿Qué queremos evitar?

Figura: Tránsito detenido en Nueva York

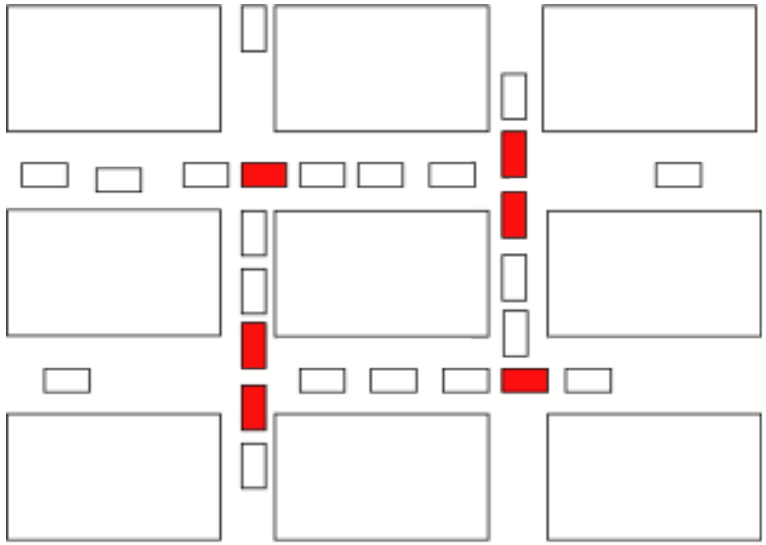

Figura: Bloqueo circular en el tránsito

Imágenes: Wikipedia: Gridlock

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

El cruce de un semáforo

- Cuando dos personas llegan a un crucero sin semáforo, ¿quién tiene el paso?

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

El cruce de un semáforo

- Cuando dos personas llegan a un crucero sin semáforo, ¿quién tiene el paso?
- Reglamento de tránsito: El conductor que viene *más por la derecha*
- ¿Y qué procede cuando cuatro conductores llegan *a la vez*?

El cruce de un semáforo

- Cuando dos personas llegan a un crucero sin semáforo, ¿quién tiene el paso?
- Reglamento de tránsito: El conductor que viene *más por la derecha*
- ¿Y qué procede cuando cuatro conductores llegan *a la vez*?
- Legalmente, los cuatro deben detenerse y nunca más avanzar
- Uno podría echarse en reversa, otro podría ignorar la ley y pasar de todos modos, ¡pero es porque los conductores humanos tienen iniciativa!

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

¿Cuándo se presenta un bloqueo mutuo?

Condiciones de Coffman

Exclusión mutua Los procesos reclaman acceso exclusivo de los recursos

Espera por Los procesos mantienen los recursos que ya les habían sido asignados mientras esperan recursos adicionales

No apropiatividad Los recursos no pueden ser extraídos de los procesos que los tienen hasta su completa utilización

Espera circular Existe una cadena circular de procesos en que cada uno mantiene a uno o más recursos que son requeridos por el siguiente en la cadena

Evaluando en base a las condiciones de Coffman

- Cada una de las condiciones presentadas son *necesarias, pero no suficientes* para que haya un bloqueo
- Pero pueden alertarnos hacia una situación de riesgo
- Cuando se presentan las cuatro, tenemos un bloqueo mutuo que sólo puede resolverse terminando a uno de los procesos involucrados
  - Pérdida de datos / estado

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Ejemplo clásico de bloqueo mutuo (1)

Asumimos: Un sistema con dos unidades de cinta (acceso secuencial, no-compartible)
Dos procesos, *A* y *B*, requieren de ambas unidades.

1. *A* solicita una unidad y se bloquea
2. *B* solicita una unidad y se bloquea
3. El sistema operativo otorga la unidad *1* a *A* y lo vuelve a poner en ejecución
4. *A* continúa procesando; termina su periodo de ejecución
5. El sistema operativo otorga la unidad *2* a *B* y lo vuelve a poner en ejecución

Ejemplo clásico de bloqueo mutuo (2)

6. *B* solicita otra unidad y se bloquea
7. El sistema operativo no tiene otra unidad por asignar. Mantiene a *B* bloqueado; otorga el control de vuelta a *A*
8. *A* solicita otra unidad y se bloquea
9. El sistema operativo no tiene otra unidad por asignar. Mantiene a *B* bloqueado; otorga el control de vuelta a otro proceso (o queda en espera)

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Esquematizando el ejemplo clásico

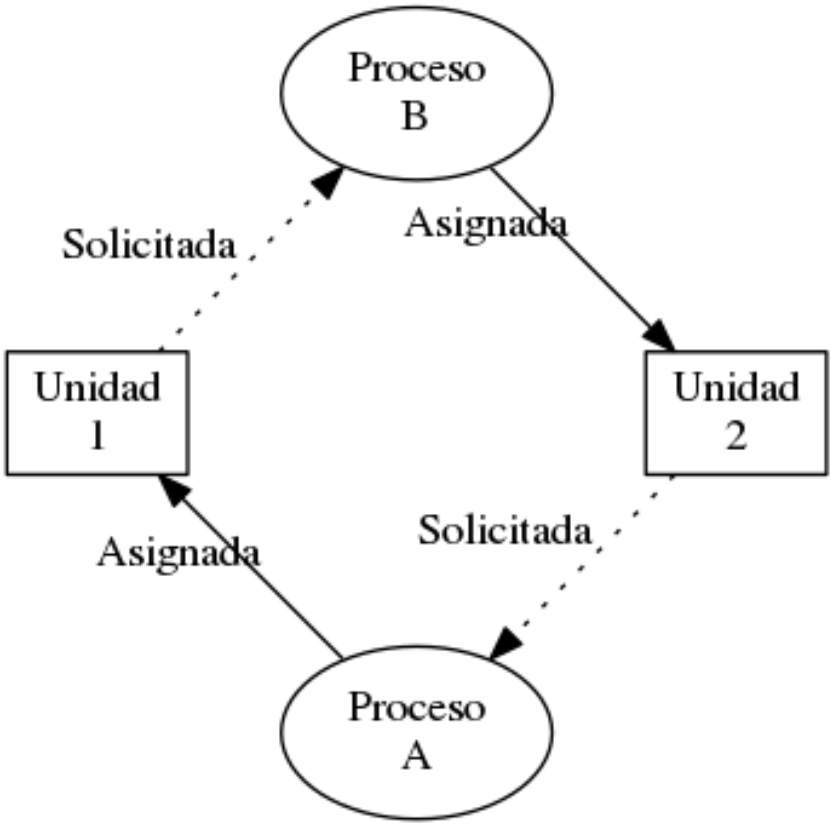

Figura: Esquema clásico de un bloqueo mutuo simple: Los procesos *A* y *B* esperan mutuamente para el acceso a las unidades *1* y *2*.

El punto de vista del sistema operativo

- El rol del sistema operativo va más allá de lo presentado en las láminas anteriores (*Exclusión mutua*)
- No podemos asumir que los procesos cooperarán entre sí
  - Ni siquiera que sabrán por anticipado de la existencia mutua
- Un rol primario del sistema operativo es gestionar los recursos del equipo

Políticas de prevención o resolución de bloqueos mutuos

Si el sistema *establece políticas* respecto a la asignación de recursos, puede evitar casos como el presentado.
Las políticas pueden verse en un contínuo entre:

Liberales Buscan a otorgar los recursos lo antes posible cuando son solicitados

Conservadoras Controlan más el proceso de asignación de recursos

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Espectro liberal-conservador de políticas

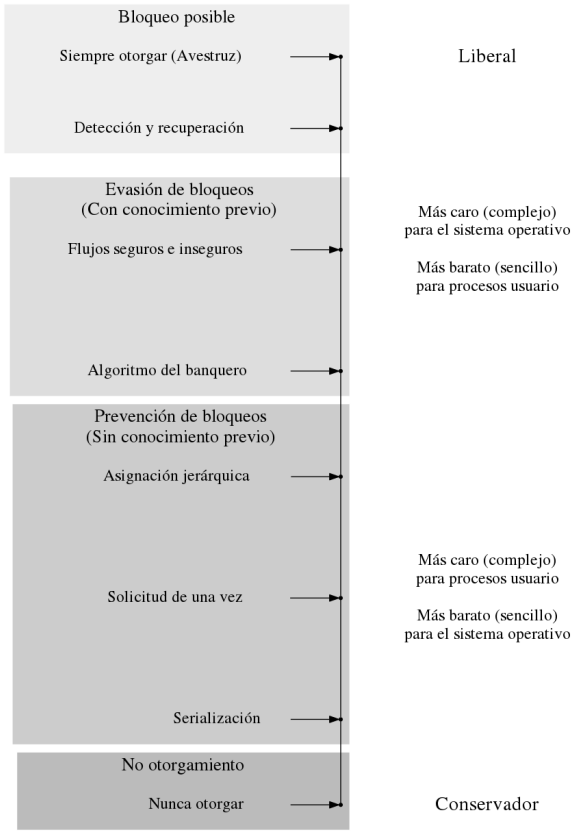

Figura: Espectro liberal—conservador de esquemas para evitar bloqueos

Categorías de estrategias ante bloqueos mutuos

Prevención Modela el comportamiento del sistema para *eliminar toda posibilidad* de un bloqueo.
Resulta en una utilización subóptima de recursos.

Evasión Impone condiciones menos estrictas. No puede evitar *todas las posibilidades* de un bloqueo; cuando éste se produce busca *evitar* sus consecuencias.

Detección y recuperación *Permite* que ocurran los bloqueos, pero busca *determinar si ha ocurrido* y actuar para eliminarlos.

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Índice

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Característica básica

Modela el comportamiento del sistema para *eliminar toda posibilidad* de un bloqueo.
Resulta en una utilización subóptima de recursos.

Serialización

- Previene caer en bloqueos negando que el sistema otorgue recursos a más de un proceso a la vez
- Los diferentes procesos pueden seguir ejecutando
  - Realizando cálculos
  - Empleando recursos *no rivales*
- Podría emplearse en un esquema tipo multiprogramación temprana (no interactiva)

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Serializando el ejemplo clásico de bloqueo mutuo (1)

1. *A* solicita una unidad y se bloquea
2. *B* solicita una unidad y se bloquea
3. El sistema operativo otorga la unidad *1* a *A* y lo vuelve a poner en ejecución
4. *A* continúa procesando; termina su periodo de ejecución
5. El sistema operativo mantiene bloqueado a *B*, dado que *A* tiene un recurso
6. *A* solicita otra unidad y se bloquea

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Serializando el ejemplo clásico de bloqueo mutuo (2)

7. El sistema operativo otorga la unidad *2* a *A* y lo vuelve a poner en ejecución
8. *A* libera la unidad *1*
9. *A* libera la unidad *2* (y con ello, el bloqueo de uso de recursos)
10. El sistema operativo otorga la unidad *1* a *B* y lo vuelve a poner en ejecución
11. *B* solicita otra unidad y se bloquea
12. El sistema operativo otorga la unidad *2* a *B* y lo vuelve a poner en ejecución
13. *B* libera la unidad *1*
14. *B* libera la unidad *2*

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Analizando a la serialización

- Previene toda posibilidad de bloqueo ante solicitud de recursos
- Pero se vuelve muy susceptible a la inanición
- Lleva a subutilización de los recursos
  - Con *n* procesos, puede haber *n-1* esperando a que uno libere los recursos.

Retención y espera o *Reserva* (*advance claim*)

- Política de prevención *menos conservadora*
- Todos los programas al iniciar su ejecución *declaran* qué recursos requerirán
- Apartados para uso exclusivo hasta que el proceso termina
- El sistema puede seguir concediendo solicitudes *que no rivalicen*
  - Si C y D requieren recursos *diferentes* de A y B, pueden ejecutarse en paralelo A, C y D
  - Posteriormente, B, C, D

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Desventajas de la *retención y espera*

- Recursos reservados por *toda la ejecución* del proceso
  - Incluso si la requieren por un tiempo muy limitado
- Percepción de *injusticia* por inanición
  - Tiempo de espera para el usuario que lanzó *B*
- Requiere que el programador sepa por anticipado los recursos que requerirá
  - Muchas veces es imposible

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Solicitud *de una vez* (*one-shot*)

- Otro mecanismo de prevención de bloqueos
- Un proceso sólo puede solicitar recursos *cuando no tiene ninguno*
  - Dos variantes: Todos de inicio, o soltar y readquirir
- Rompe la condición de *espera por*
- ... ¡Pero hay muchos recursos que deben *mantenerse bloqueados* por largo plazo!
- Cambia la lógica de programación al tener puntos definidos de adquisición y liberación

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Asignación de recursos *jerárquica*

- A cada categoría de recursos se le asigna una prioridad o *nivel jerárquico*
- Un proceso dado sólo puede solicitar recursos *de un nivel superior* a los que ya tiene asignados.
  - Para pedir dos dispositivos del mismo nivel, debe hacerse de forma *atómica*
  - Si tanto P_1 como P_2 requieren dos unidades de cinta, ambos deben pedirlas a ambas de una sola vez
  - El planificador elegirá cuál *gana*; ese tendrá ambos recursos
  - El otro esperará hasta que éste termine
  - No hay bloqueo
- Podríamos presentar a la solicitud *de una vez* como un caso degenerado de asignación jerárquica.

Asignación de recursos *jerárquica*

- Si P_1 y P_2 requieren una impresora y una unidad de cinta, hay un órden claro en que tienen que pedirse
  - No hay bloqueo
- Los recursos más escasos están más arriba en la jerarquía
  - Esto hace que lleguen menos solicitudes por ellos

... ¿Pero?

- Es un ordenamiento demasiado estricto para muchas situaciones del mundo real
- Lleva a los procesos a acaparar recursos de baja prioridad
- Conduce a inanición

Sorteando los mecanismos de prevención

- ¿Podemos *burlar* estos mecanismos?
  - P.ej. empleando procesos *representantes* (*proxy*)
- Un programador poco cuidadoso puede, sin llegar a *bloqueo por recursos*, llegar a un *bloqueo por procesos*
  - Pero una *buena práctica* sería descargar el uso de recursos rivales en un proceso que sepa cómo compartirlos *inteligentemente*

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Prevención de bloqueos: Resumiendo

- Mecanismos muy *conservadores* pero 100 % efectivos (si nos limitamos a lo *declarado*...)
- Parecerían poco acordes a un entorno multiusuario/multitarea como la mayoría de los actuales
- Sin embargo, empleados para ciertos subsistemas, con *esquemas de mediación*
  - Impresión
  - Audio

Índice

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Característica básica

Impone condiciones menos estrictas. No puede evitar *todas las posibilidades* de un bloqueo; cuando éste se produce busca *evitar* sus consecuencias.

Enfoque 1: Flujos seguros e inseguros

- Realizado por el planificador
- Requiere saber *por anticipado* qué procesos utilizarán qué recursos
  - Un poco menos detallado que la prevención (número de recursos por categoría)
- Saber *cuándo* se va a usar cada recurso
- Análisis de la interacción, marcando *áreas de riesgo*

Flujos seguros e inseguros

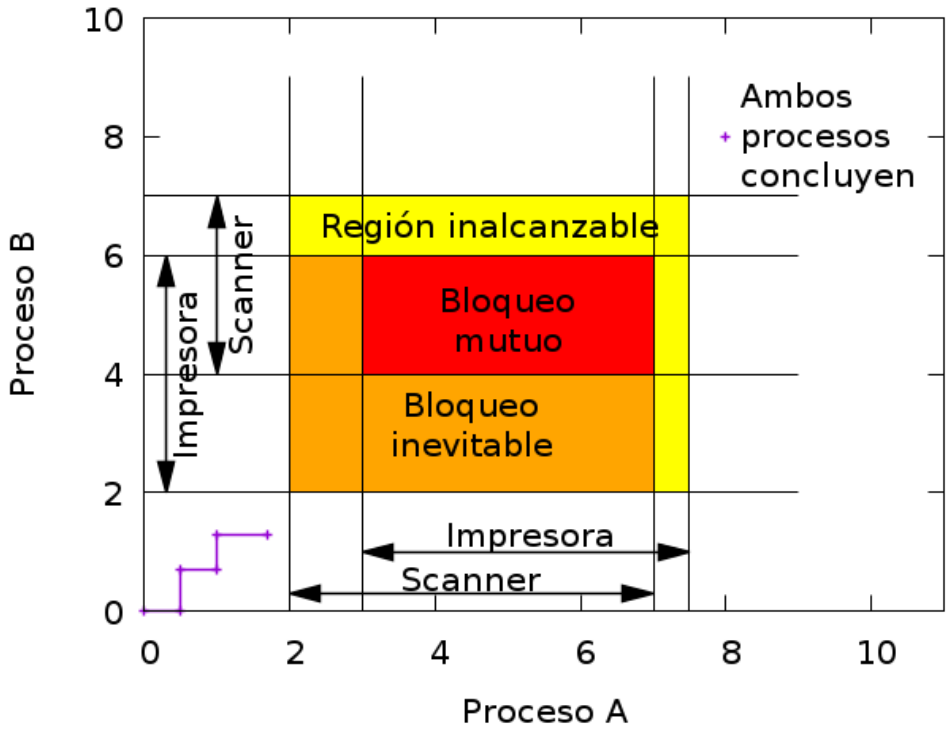

Figura: Evasión de bloqueos: Los procesos A (horizontal) y B (vertical) requieren del acceso exclusivo a dos recursos rivales, exponiéndose a bloqueo mutuo.

Áreas de riesgo

- Mientras avancemos por el *área segura* no hay riesgo de bloqueos
- Sistemas uniprocesador, sólo avance vertical/horizontal
  - Sistemas multiprocesador, avance diagonal
- Áreas de riesgo: Cuando *alguno de los recursos rivales* es otorgado
- El bloqueo mutuo se produce cuando $3 \leq t_A \leq 7$, y $4 \leq t_B \leq 6$
- Pero, *sabiendo esto*, el SO debe mantener a $t_B < 2$ siempre que $2 \leq t_A \leq 3$
  - O a $t_A < 2$ siempre que $2 \leq t_B \leq 4$.

Analizando esta estrategia

- Muy dificil de implementar en un sistema de propósito general
  - Requiere análisis estático previo del código
  - O requisitos de programación diferentes a los expuestos por los sistemas en uso generalizado
- Puede especificarse dentro de un *marco de desarrollo*: Asignación de recursos por *subrutina*

Enfoque 2: *Algoritmo del banquero*

- Edsger Dijkstra, para el sistema operativo THE, 1965-1968
- El sistema procede cuidando de la *liquidez* para siempre poder satisfacer los *préstamos* (recursos) de sus clientes
- Permite que el *conjunto de recursos* solicitados por los procesos sean mayores a los disponibles

Requisitos para el *Algoritmo del banquero*

- Debe ejecutarse cada vez que un proceso solicita recursos
- Todo proceso debe declarar su *reclamo máximo* (*claim*) de recursos al iniciar su ejecución
  - Si el reclamo en cualquier categoría es superior al máximo existente, el sistema niega la ejecución

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Estructuras a emplear: Relativas a procesos

Matrices (llaves por categoría y por proceso)

Reclamado Número de instancias de este recurso que han sido reclamadas

Asignado Número de instancias de este recurso actualmente asignadas a procesos en ejecución

Solicitado Número de instancias de este recurso actualmente pendientes de asignar (solicitudes hechas y no cumplidas)

Estructuras a emplear: Categorías de recursos

Listas (llave por categoría)

Disponibles Número total de instancias de este recurso disponibles al sistema

Libres Número de instancias de este recurso que no están actualmente asignadas a ningún proceso

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Estados

Estado Matrices de recursos disponibles, reclamos máximos y asignación de recursos a los procesos en un momento dado

Estado seguro Un estado en el cual todos los procesos pueden ejecutar hasta el final sin encontrar un bloqueo mutuo.

Estado inseguro Todo estado que no garantice que todos los procesos puedan ejecutar hasta el final sin encontrar un bloqueo mutuo.

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Lógica del algoritmo del banquero

Cada vez que un proceso solicita recursos, se calcula cuál sería el estado resultante de *otorgar* dicha solicitud, y se otorga siempre que:

- No haya reclamo por más recursos que los disponibles
- Ningún proceso solicite (o tenga asignados) recursos por encima de su reclamo
- La suma de los recursos *asignados* por cada categoría no sea mayor a la cantidad de recursos *disponibles* en el sistema para dicha categoría

Definiendo estados *seguros*

Un estado *es seguro* cuando hay una secuencia de procesos (denominada *secuencia segura*) tal que:

1. Un proceso *j* puede necesariamente terminar su ejecución
   - Incluso si solicitara *todos* los recursos que reservó en su reclamo
   - Siempre debe haber suficientes recursos libres para satisfacerlo
2. Un segundo proceso *k* de la secuencia puede terminar:
   - Si *j* termina y libera todos los recursos que tiene
   - Siempre que sumado a los recursos disponibles ahora, con aquellos que liberaría *j*, hay suficientes recursos libres
3. El *i*-ésimo proceso puede terminar si todos los procesos anteriores terminan y liberan sus recursos.

Peor caso con el algoritmo del banquero

En el peor de los casos, esta secuencia segura nos llevaría a bloquear todas las solicitudes excepto las del proceso único en el órden presentado.

Ejemplo

Asumiendo que tenemos *sólo una clase de recursos* y nos quedan dos instancias libres:

Proceso	Asignado	Reclamando
A	4	6
B	4	11
C	2	7

1. A puede terminar porque sólo requiere dos instancias adicionales
2. Terminado A, C puede recibir las 5 restantes que requiere
3. Terminados A y C, B puede satisfacer las 7
4. La secuencia A-C-B es segura.

Ejemplo 2

Sólo una clase de recursos, 2 instancias libres

Proceso	Asignado	Reclamado
A	4	6
B	4	11
C	2	9

1. A puede satisfacer su demanda
2. Pero una vez terminado A, no podemos asegurar las necesidades ni de B ni de C
3. Este es un *estado inseguro*, por lo cual el algoritmo del banquero *no debe permitir llegar a él*.

Ejemplo 3: Paso por paso

¿Hasta dónde nos dejaría *avanzar* el algoritmo del banquero?

- Estado inicial:
  - 8 recursos disponibles
  - Reclamos: P_1: 5, P_2: 5, P_3: 4
- t_1: P_1 con 2, P_2 con 1, P_3 con 1
- t_2: P_2 solicita 1
- t_3: P_3 solicita 1
- t_4: P_1 solicita 1

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Ejemplo 3: Paso por paso

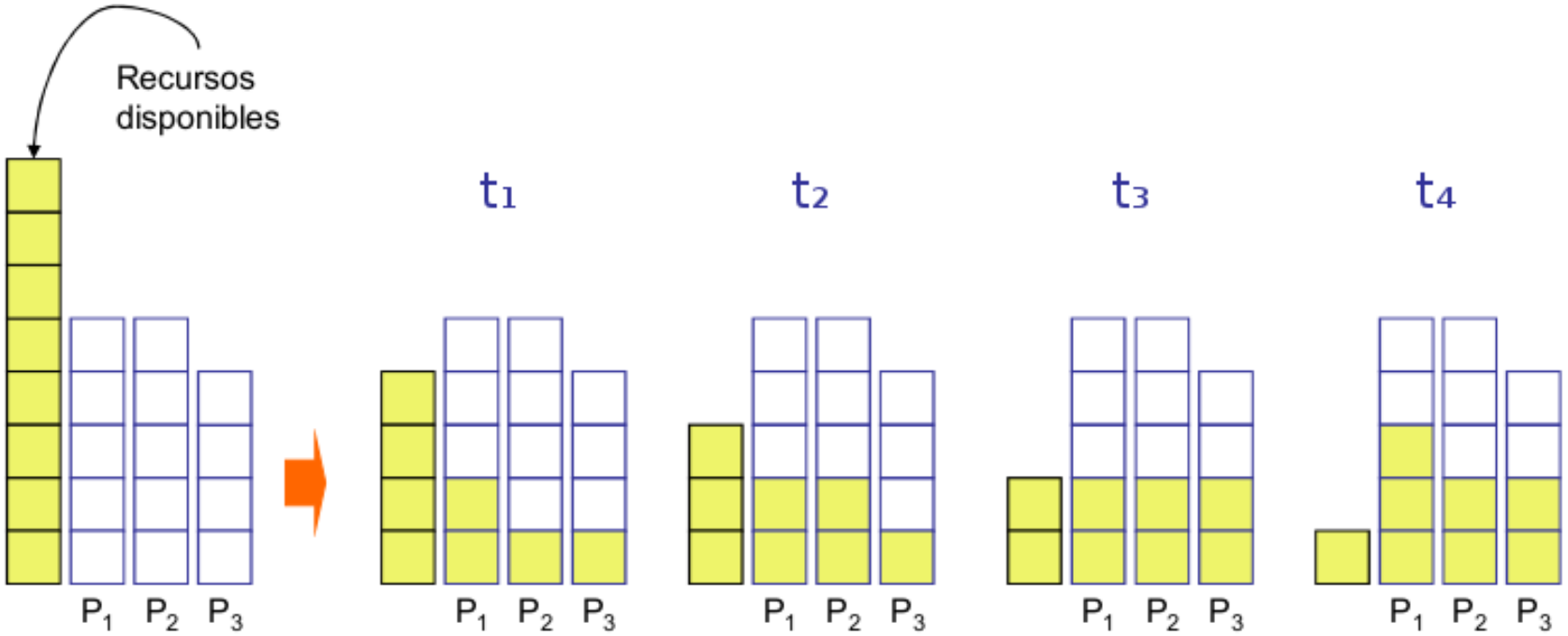

Imagen: Samuel Oporto Díaz

Ejemplo 3: Paso por paso

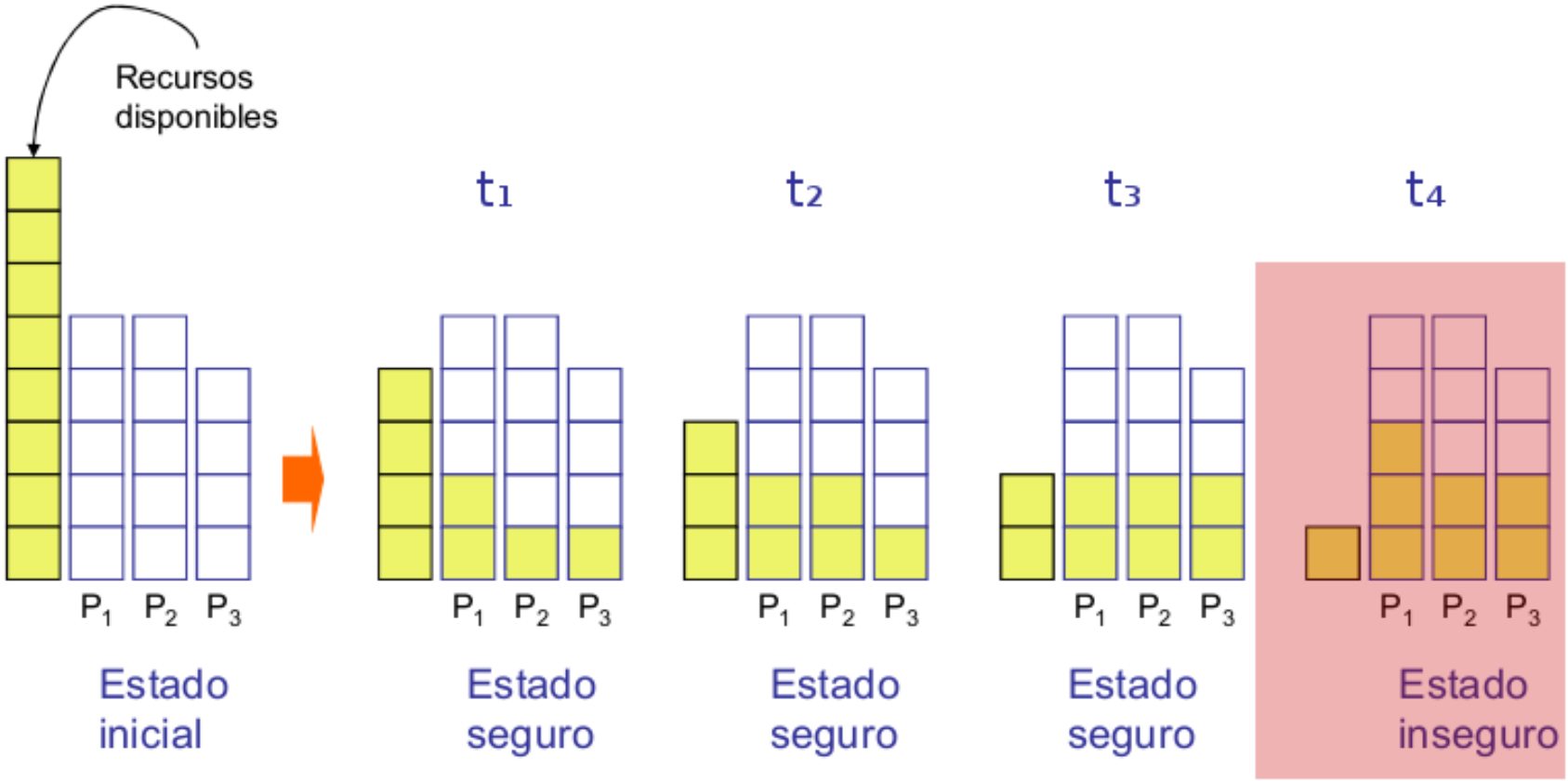

Imagen: Samuel Oporto Díaz

Ejemplo 3: Paso por paso

- El algoritmo nos permite avanzar por t_1, t_2, t_3
  - Uso *efectivo* de recursos del sistema: 14
  - *Sobrecompromiso* (*overcommitment*) ante los 8 recursos *reales*
- Al llegar a t_4, otorgar un recurso a P_1 nos ubica en un *estado inseguro*
  - Es imposible satisfacer el reclamo completo de P_1, P_2 o P_3
  - Desde t_3, *sólo* podemos otorgar recursos a P_3
  - P_1 y P_2 pueden seguir ejecutando *si no solicitan recursos*

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Implementación ejemplo (para una sóla categoría)

```
l = ['A', 'B', 'C', 'D', 'E']; # Todos los procesos del sistema
s = []; # Secuencia segura
while ! l.empty? do
  p = l.select {|id| reclamado[id] - asignado[id] <=
      libres}.first
  raise Exception, 'Estado inseguro' if p.nil?
  libres += asignado[p]
  l.delete(p)
  s.push(p)
end
puts "La secuencia segura encontrada es: " + s
```

Precisiones

- En el ejemplo 2, es posible que ni B ni C requirieran ya *todos* sus recursos reclamados
- Pero el estado es inseguro.
- El algoritmo del banquero, en el peor caso, puede tomar $O(n!)$
  - Típicamente toma $O(n^2)$
- Hay refinamientos a este algoritmo que reducen su costo de ejecución
  - Puede ser llamado con muy alta frecuencia

Precisiones

- Es un algoritmo *conservador*: Evita entrar en estados inseguros *a pesar de que no lleve con certeza a un bloqueo mutuo*
  - Pero es la política más liberal que evita los bloqueos *sin conocer órden y tiempo* de necesidad de recursos.
- Desventaja de *todos* los algoritmos basados en la evasión: Requieren saber por anticipado los reclamos máximos
  - No siempre es posible con el modelo actual de computación

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Índice

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Característica básica

Permite que ocurran los bloqueos, pero busca *determinar si ha ocurrido* y actuar para eliminarlos.
A diferencia de la *prevención* y la *evasión*, las cuatro condiciones de Coffman *pueden presentarse*.

Funcionamiento básico

- Se ejecuta como una tarea *periódica*
  - Busca *limitar* el impacto de los bloqueos *existentes* en el sistema
  - (Discutiremos al respecto más adelante...)
- Mantenemos una lista de recursos existentes, asignados y solicitados
- Eliminando lo *obviamente resuelto*, nos quedamos con lo *obviamente bloqueado*

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Representación con grafos dirigidos

- Procesos con cuadrados
- Recursos con círculos
  - Puede representarse una *clase de recursos* como un círculo grande, *conteniendo* círculos pequeños indicando *recursos idénticos*
  - Si un proceso puede solicitar *un recurso específico* (y no *cualquiera de una categoría*), no son idénticos
- Flecha de un recurso a un proceso: El recurso *está asignado* al proceso.
- Flecha de un proceso a un recurso: El proceso *está solicitando* al recurso.

Estrategia general

- Tenemos la representación completa de los recursos y procesos en el sistema
- Buscamos *reducir* la gráfica, retirando los elementos que no aportan *información relevante*:
  1. Los procesos que no estén solicitando ni tienen asignado un recurso
  2. Para los procesos restantes: Si todos los recursos que solicitan *pueden ser concedidos* (no están concedidos a otro), eliminamos al proceso y a todas sus flechas.
     - Repetimos cuantas veces sea posible
  3. Si no quedan procesos en el grafo, no hay interbloqueos y podemos continuar
  4. Los procesos "irreductibles" están en bloqueo mutuo.
     - Proceder según la política del sistema

Ejemplo 1

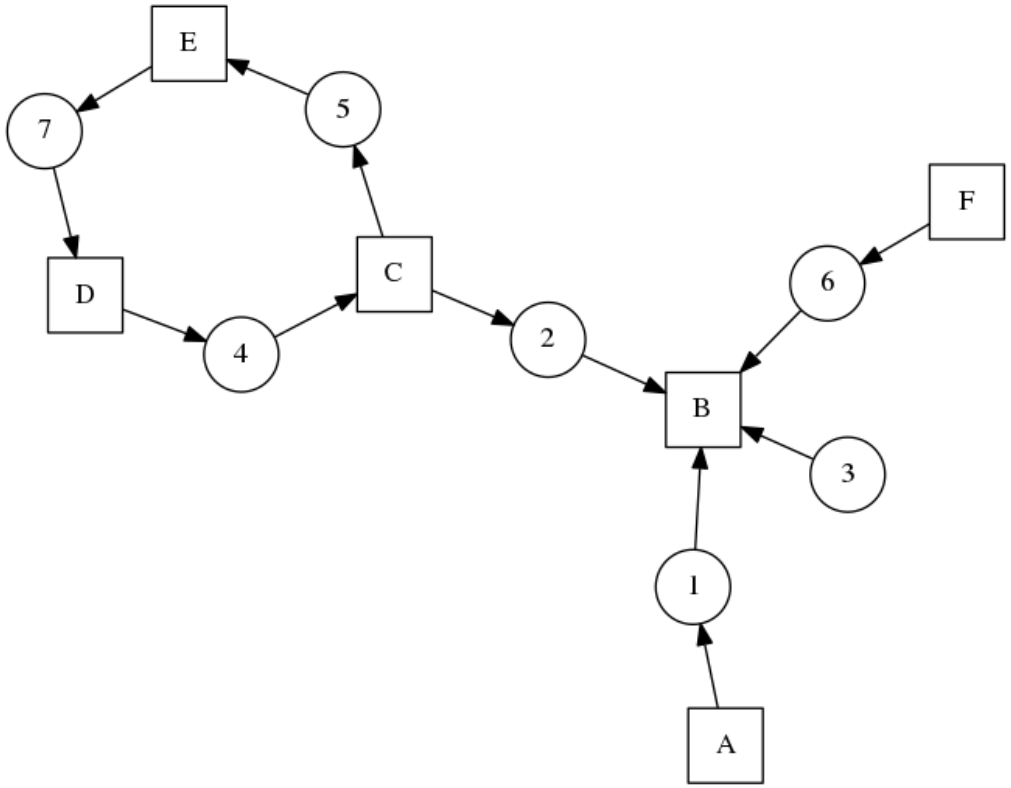

Figura: Detección de ciclos denotando bloqueos: Grafo de procesos y recursos en un momento dado

Resolviendo el ejemplo 1

1. Reducimos por *B*, dado que actualmente no está esperando a ningún recurso
2. Reducimos por *A* y *F*, dado que los recursos por los cuales están esperando quedarían libres en ausencia de *B*
3. Quedamos con un interbloqueo entre *C*, *D* y *E*, en torno a los recursos *4*, *5* y *7*.

Ejemplo 2

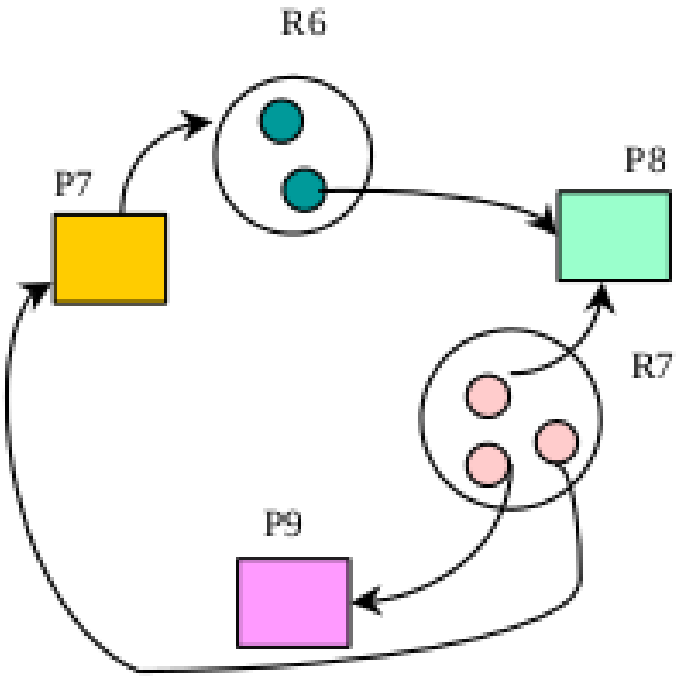

Figura: Ejemplo con categorías de recursos (Imágenes: Luis La Red)

Ejemplo 2

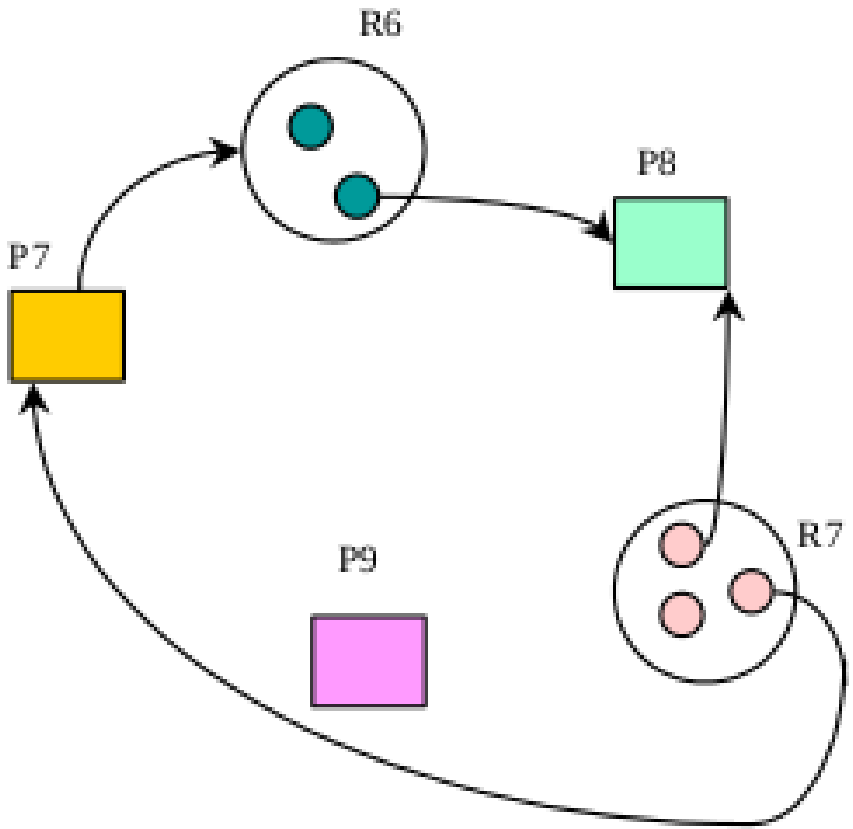

Figura: Resultado de reducir por P9

Ejemplo 2

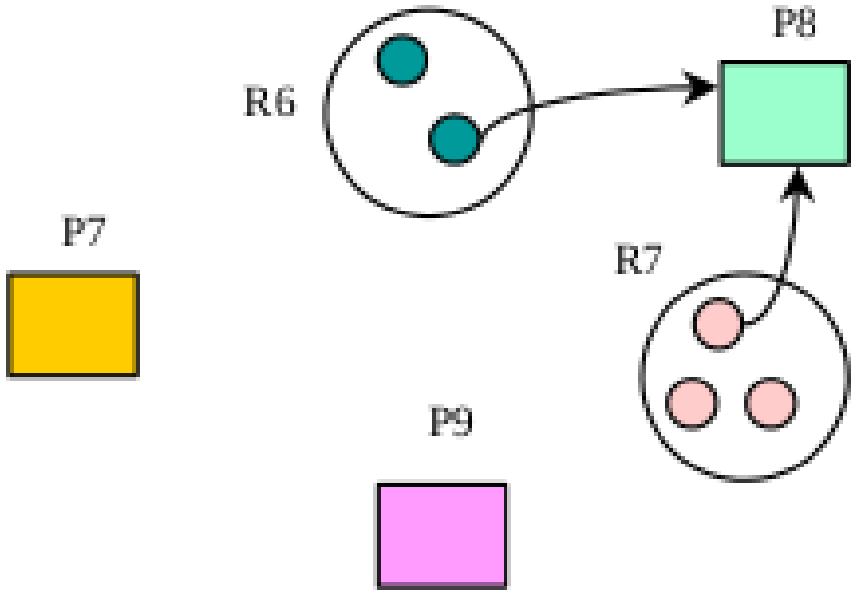

Figura: Resultado de reducir por P7

Ejemplo 2

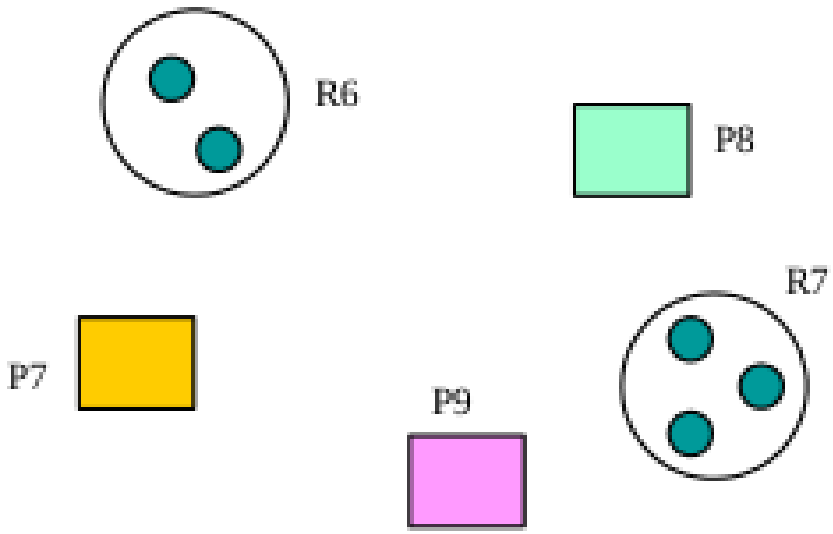

Figura: Resultado de reducir por P8

¡No todo ciclo es un bloqueo!

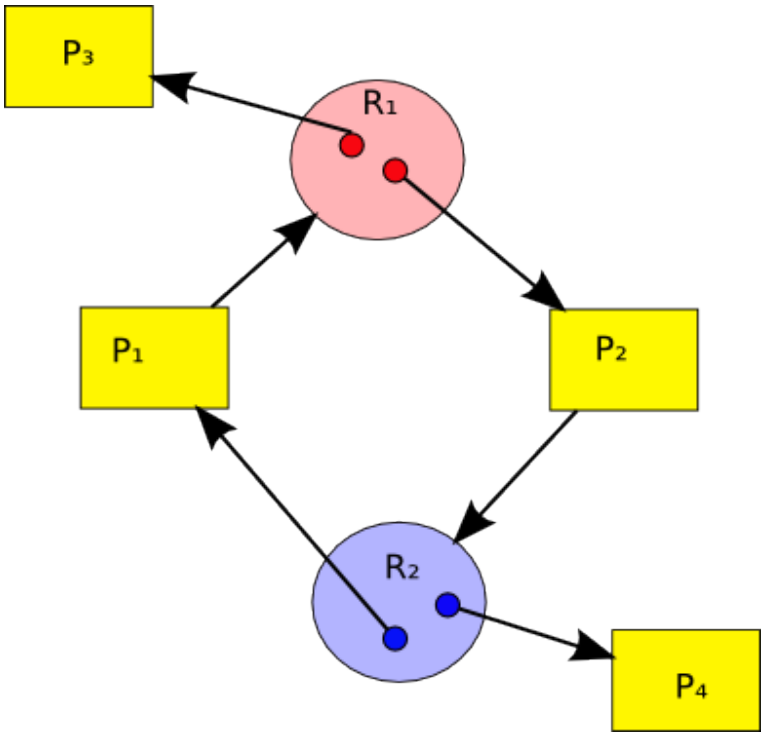

Figura: Al manejar *categorías* de recursos, no siempre un ciclo implicará un bloqueo

Bloqueado *vs.* bloqueando

- *Reducir un proceso* no implica que este *haya entregado* sus recursos
  - Sólo que *está en posibilidad de hacerlo*
  - Puede haber inanición por parte de cualquier proceso que espera
- Podría agregarse a este algoritmo una ponderación por tiempo excesivo de retención
  - Pero causaría muchos falsos positivos → ¡Muchos casos de procesos finalizados sin necesidad!

Ahora... Una vez detectado el bloqueo, ¿qué procede?

Mátelos, luego 'virigüa

- Terminar a *todos* los procesos bloqueados
- Técnica más sencilla y más justa...
  - Para cierta definición de justicia
- Maximiza la pérdida de información

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Retroceder al último *punto de control* (*checkpoint*)

- Sólo cuando el sistema implementa esta funcionalidad
- Sólo cuando *ninguno* de los procesos depende de factores externos
- Retroceder en el tiempo, ¿no llevaría a un nuevo bloqueo?

Retroceder al último *punto de control* (*checkpoint*)

- Sólo cuando el sistema implementa esta funcionalidad
- Sólo cuando *ninguno* de los procesos depende de factores externos
- Retroceder en el tiempo, ¿no llevaría a un nuevo bloqueo?
- Los bloqueos están ligados al órden *específico* de ejecución
- Probablemente otro orden específico se salve del bloqueo
  - Y si no... Se vuelve una vez más.
  - Tal vez, agregar un contador de *intentos* que cambie el planificador

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Terminación selectiva / limitada

- Terminar forzosamente, uno a uno (y no en bloque), a los procesos bloqueados
  - Terminar a uno y re-evaluar el estado
  - Los restantes pueden continuar operando
- ¿Cómo elijo al desafortunado? Varias estrategias...

Estrategias para elegir el proceso a terminar (1)

- Los procesos más sensibles para detener/relanzar: Los que demandan *garantías de tiempo real*. Evitar penalizarlos.
- Causar una menor pérdida de trabajo. El que haya consumido menor cantidad de procesador hasta el momento.
- Mayor tiempo restante estimado (si puedo estimar cuánto procesamiento *queda pendiente*)

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Estrategias para elegir el proceso a terminar (2)

- Menor *número* de recursos asignados hasta ahora (como criterio de *justicia*: ¿qué proceso está haciendo un uso *más juicioso* del sistema?)
- Prioridad más baja (cuando la hay)
- Procesos con naturaleza repetible sin pérdida de información (aunque sí de tiempo)
  - p.ej. es mejor interrupmir una compilación que la actualización de una BD

Periodicidad del chequeo

- Si buscamos bloqueos cada vez que se solicita un recurso, es una sobrecarga administrativa demasiado grande
- Con una periodicidad fija: Arriesga a que los procesos pasen más tiempo bloqueados ($\approx 0{,}5t$)
- Cuando el *nivel de uso* del CPU baje de cierto porcentaje
  - Indica que hay un nivel elevado de procesos en espera
  - ...Pero un sistema demasiado ocupado puede nunca *disparar* esta condición
- Una estrategia combinada

Índice

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Frustraciones...

- Les advertí que este tema me resultaba frustrante
- Hemos visto varios enfoques a cómo lidiar con la presencia de bloqueos
- Es un área viva y activa de investigación, y viene avanzando desde los primeros días del multiprocesamiento
- Sin embargo...

Recapitulando: Conservadores y liberales

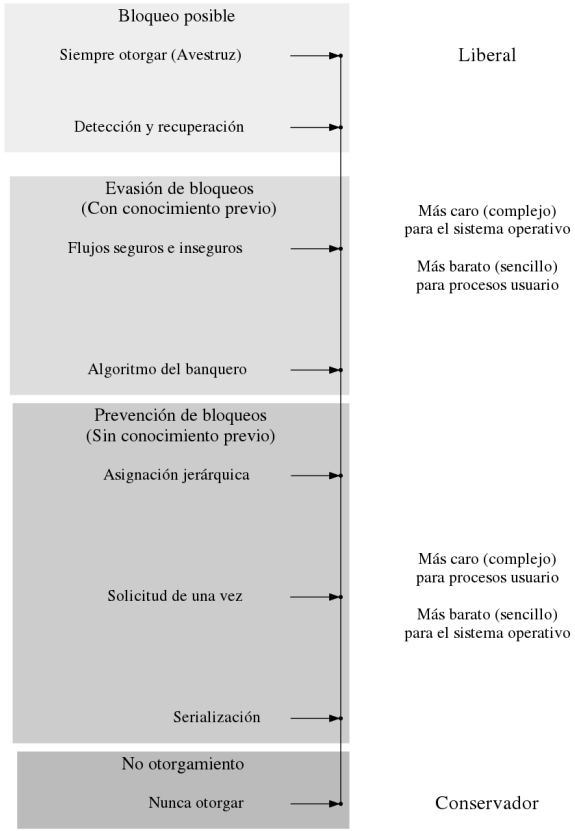

Figura: Espectro liberal—conservador de esquemas para evitar bloqueos

El Algoritmo del avestruz

- Mecanismo más frecuente utilizado por los sistemas operativos de propósito general
- Ignorar las situaciones de bloqueo
- Esconder la cabeza bajo tierra y esperar a que pasen
- Esperar que su ocurrencia sea suficientemente poco frecuente
- Esperar que el usuario detecte y resuelva el problema por sí sólo

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

El por qué del avestruz

- Las condiciones impuestas por las diversas estrategias son casi imposibles de alcanzar
- Conocimiento previo de requisitos insuficiente
- Bloqueos originados fuera de nuestra esfera de acción
  - Recursos externos
  - Procesos *representante* (*proxy*)
  - Estructuras a niveles superiores
- Modelo de computación tendiente al interactivo
  - En servidores: Programas (¡y personas!) dedicados al monitoreo

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

¿Qué es un recurso?

- Para que estos mecanismos funcionen, tenemos que definir qué es un recurso — A qué esperamos cubrir (y, por tanto, a qué no)
  - No sólo cintas, impresoras, tarjetas de audio...
  - ¡También segmentos de memoria, tiempo de procesamiento!
  - ¿Y las estructuras *lógicas* creadas por el SO? Archivos, semáforos, monitores, ...
  - ¿Y elementos que vienen de *fuera de nuestro sistema*? (cómputo distribuído, SOAP/servicios Web, etc.)

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Cuestión de compromisos

- Un sistema operativo de propósito general tiene que funcionar para muy distintas situaciones
  - Es muy difícil plantear esquemas aptos para toda necesidad
- Debe tomarse una decisión entre lo *correcto* y lo *conveniente*
  - Un SO no debería permitir que hubiera bloqueos
  - Pero la inconveniencia a usuarios y programadores sería demasiada

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Cuestión de programar *defensivamente*

El programador puede tomar ciertas medidas para evitar que *su programa* caiga en situaciones de bloqueo

- Solicitar recursos con llamadas *no bloqueantes*
- Temporizadores y manejo de errores
- Particularmente, *notificar al usuario* en caso de demoras o fallas repetidas

Aplicaciones de monitoreo en espacio de usuario

- El monitoreo también ha caído hacia el terreno de las aplicaciones
- Brinda mayor flexibilidad
  - Aunque menos poder
- Hay cientos de aplicaciones de monitoreo para todo tipo de situaciones
- Un monitor *inteligente*, adaptado para un *servicio* en particular, puede detectar mejor situaciones anómalas (y no sólo bloqueos)

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

Posiciones disciplinares: Matemáticos contra ingenieros

- ¿Dónde cabe mejor la computación? ¿Y dónde cabemos mejor *cada uno de nosotros*?
- La asignación de recursos puede verse como un problema formal, matemático
  - Los mecanismos descritos no son perfectos
  - Pero el problema *no ha demostrado ser intratable*
  - Un bloqueo es claramente un error
- Los matemáticos en nuestro *árbol genealógico académico* nos llaman a no ignorar el problema
  - A resolverlo sin importar la complejidad computacional

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM

El punto de vista del ingeniero

- La ingeniería es la ciencia *aplicada al mundo real*
- Debe, sí, evitar efectos nocivos
  - Pero también, calcula costos, probabilidades de impacto, umbrales de tolerancia...
- Un sistema típico *corre riesgo* de caer en un bloqueo con una probabilidad $p > 0$
  - Impacto: Perder *dos procesos* en un sistema
  - Otros factores de fallo: Hardware, otros subsistemas del SO, errores en cada uno de los programas...
- Prevenir el bloqueo conlleva complejidad en desarrollo o disminución en rendimiento

Resumiendo a las avestruces

- No podemos decir *este camino es correcto* o *este no*
- Depende de muchos factores
  - Ámbito del bloqueo
  - Función del sistema operativo en cuestión
  - Costo de implementación
- Hecho: Tenemos avestruces *ahí afuera*, al igual que tenemos sistemas críticos que requieren verificación formal a cada asignación
- ...Lo que no vale es que *nosotros* seamos las avestruces.

Gunnar Wolf • sistop@gwolf.org • http://sistop.gwolf.org
Sistemas Operativos (1554) • Facultad de Ingeniería UNAM