Programación Concurrente ATIC
Programación Concurrente (redictado)

Clase 5

Facultad de Informática

UNLP

Semáforos (continuación)

Alocación de Recursos y Scheduling

Problema: decidir cuándo se le puede dar a un proceso determinado acceso a un recurso. Implementar políticas de alocación de recursos generales controlando explícitamente cuál proceso toma un recurso si hay más de uno esperando.

Recurso: cualquier objeto, elemento, componente, dato, SC, por la que un proceso puede ser demorado esperando adquirirlo.

Definición del problema: procesos que compiten por el uso de unidades de un recurso compartido (cada unidad está ***libre*** o ***en uso***).

request (parámetros): ⟨await (request puede ser satisfecho) tomar unidades;⟩

release (parámetros): ⟨retornar unidades;⟩

- Puede usarse Passing the Baton:

request (parámetros): P(e);
if (request no puede ser satisfecho) DELAY;
tomar las unidades;
SIGNAL;

release (parámetros): P(e);
retornar unidades;
SIGNAL;

Alocación de Recursos y Scheduling

Alocación Shortest-Job-Next (SJN)

- Varios procesos que compiten por el uso de un recurso compartido *de **una sola unidad***.
- **request** (tiempo,id). Si el recurso está libre, es alocado inmediatamente al proceso ***id***; sino, el proceso ***id*** se demora.
- **release** ( ). Cuando el recurso es liberado, es alocado al proceso demorado (si lo hay) con el mínimo valor de ***tiempo***. Si dos o más procesos tienen el mismo valor de ***tiempo***, el recurso es alocado al que esperó más.
- SJN minimiza el tiempo promedio de ejecución, aunque no es ***fair*** (¿por qué?). Puede mejorarse con la técnica de ***aging*** (dando preferencia a un proceso que esperó mucho tiempo).
- Para el caso general de alocación de recursos (NO SJN):

  bool libre = true;
  request (tiempo,id): ⟨await (libre) libre = false;⟩
  release (): ⟨libre = true;⟩

Alocación de Recursos y Scheduling

Alocación Shortest-Job-Next (SJN)

- En SJN, un proceso que invoca a *request* debe demorarse hasta que el recurso esté libre y su pedido sea el próximo en ser atendido de acuerdo a la política. El parámetro tiempo entra en juego sólo si un pedido debe ser demorado.

```
request (tiempo, id):
        P(e);
        if (not libre) DELAY;
        libre = false;
        SIGNAL;

release ( ):
        P(e);
        libre = true;
        SIGNAL;
```

- En ***DELAY*** un proceso:
  - Inserta sus parámetros en un conjunto, cola o lista de espera (*pares*).
  - Libera la SC ejecutando V(e).
  - Se demora en un semáforo hasta que *request* puede ser satisfecho.
- En ***SIGNAL*** un proceso:
  - Cuando el recurso es liberado, si pares no está vacío, el recurso es asignado a un proceso de acuerdo a SJN.

- Cada proceso tiene una condición de demora distinta, dependiendo de su posición en ***pares***. El proceso ***id*** se demora sobre el semáforo ***b[id]***.

Alocación de Recursos y Scheduling

Alocación Shortest-Job-Next (SJN)

```
bool libre = true;  Pares = set of (int, int) = ∅;  sem e = 1, b[n] = ([n] 0);

request(tiempo,id):   P(e);
                      if (! Libre){ insertar (tiempo, id) en Pares; V(e); P(b[id]); }
                      libre = false;
                      V(e);

release( ):           P(e);
                      libre = true;
                      if (Pares ≠ ∅ ) { remover el primer par (tiempo,id) de Pares; V(b[id]); }
                      else V(e);
```

s es un ***semáforo privado*** si exactamente un proceso ejecuta operaciones ***P*** sobre ***s***. Resultan útiles para señalar procesos individuales. Los semáforos ***b[id]*** son de este tipo.

Alocación de Recursos y Scheduling

Alocación Shortest-Job-Next (SJN)

```
bool libre = true;
Pares = set of (int, int) = ∅;
sem e = 1, b[n] = ([n] 0);

Process Cliente[id: 0..N-1]::
{  int tiempo = …;
   …
   //request
   P(e);
   if (! Libre){ insertar (tiempo, id) en Pares; V(e); P(b[id]); }
   libre = false;
   V(e);
   //Usa el recurso compartido
   //reléase
   P(e);
   libre = true;
   if (Pares ≠ ∅ ) { remover el primer par (tiempo,id) de Pares; V(b[id]); }
   else V(e);
   …
}
```

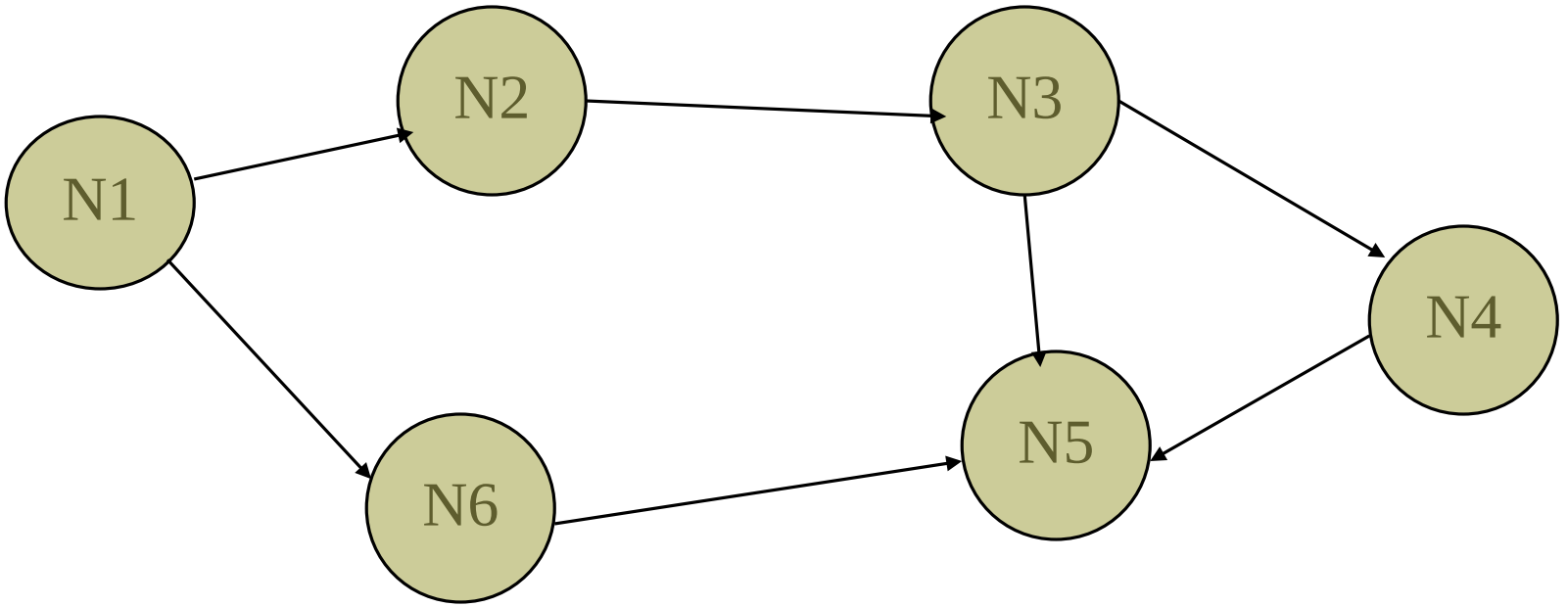
¿Que modificaciones deberían realizarse para generalizar la solución a recursos de múltiple unidad?

Casos Problema

Grafo de precedencia

Analizar el problema de modelizar N procesos que deben sincronizar, de acuerdo a lo especificado por un grafo de precedencia arbitrario (con semáforos).

Ejemplo: en este caso N4 hará P(N3) y V(N5), N1 hará V(N2) y V(N6).



Productores/Consumidores con broadcast

En el problema del buffer atómico, sea ***UN*** proceso productor y ***N*** procesos consumidores.

El Productor ***DEPOSITA*** y debe esperar que ***TODOS*** los consumidores consuman el mismo mensaje (*broadcast*).

Notar la diferencia entre una solución por memoria compartida y otra por pasaje de mensajes.

Versión más compleja: buffer con ***K*** lugares, ***UN*** productor y ***N*** consumidores.

El productor puede depositar hasta ***K*** mensajes, los ***N*** consumidores deben leer cada mensaje para que el lugar se libere y el orden de lectura de cada consumidor debe ser ***FIFO***.

Analizar el tema con semáforos.

Variantes del problema de los filósofos

- Si en lugar de administrar tenedores, cada filósofo tiene un estado (comiendo, pensando, con hambre) y debe consultar a sus dos filósofos laterales para saber si puede comer, tendremos una solución distribuida. ¿Se podría usar la técnica de “passing the baton” para resolverlo?.
- Otra alternativa es tener una solución de filósofos centralizada, en la que un scheduler administra por ejemplo los tenedores y los asigna con posiciones fijas o variables (notar la diferencia).
- En la solución que vimos de filósofos, para evitar deadlock utilizamos un código distinto para un filósofo (orden de toma de los tenedores). Otra alternativa es la de la elección al azar del primer tenedor a tratar de tomar... ¿Cómo?

Problema de los baños

- Un baño único para varones o mujeres (excluyente) sin límite de usuarios.
- Un baño único para varones o mujeres (excluyente) con un número máximo de ocupantes simultáneos (que puede ser diferente para varones y mujeres)
- Dos baños utilizables simultáneamente por un número máximo de varones (K1) y de mujeres (K2), con una restricción adicional respecto que el total de usuarios debe ser menor que K3 (K3 < K1 + K2).

Problema de la molécula de agua

- Existen procesos O (oxígeno) y H (hidrógeno) que en un determinado momento toman un estado “listo” y se buscan para combinarse formando una molécula de agua (HH0).
- Puede pensarse en un esquema Cliente/Servidor, donde el servidor recibe los pedidos y cuando tiene 2 H listos y 1 O listo concreta la molécula de agua y libera los procesos H y O.
- También puede pensarse como un esquema “passing the baton” que puede iniciarse por cualquiera de los dos H o por el O, pero tiene un orden determinado (analizar con cuidado).
- ¿Podría pensar la solución con un esquema de productores-consumidores? ¿Quiénes serían los productores y quienes los consumidores?

Puente de una vía

- Suponga un puente de una sola vía que es accedido por sus extremos (procesos Norte y procesos Sur).
- Cómo especificaría la exclusión mutua sobre el puente, de modo que circulen los vehículos (procesos) en una sola dirección.
- Es un caso típico donde es difícil asegurar fairness y no inanición. ¿Por qué? ¿Cuál podría ser un método para asegurar no inanición con un scheduler? ¿Qué ideas propone para tener fairness entre Norte y Sur ?
- Suponga que cruzar el puente requiere a lo sumo 3 minutos y Ud. quiere implementar una solución tipo “time sharing” entre los procesos Norte y Sur, habilitando alternativamente el puente 15 minutos en una y otra dirección. ¿Cómo lo esquematizaría?

Search – Insert – Delete sobre una lista con procesos concurrentes

- Una generalización de la Exclusión Mutua Selectiva entre clases de procesos visto con lectores-escritores es el problema de procesos que acceden a una lista enlazada para ***Buscar*** (search), ***Insertar*** al final o ***Borrar*** un elemento en cualquier posición.
- Los procesos que BORRAN se excluyen entre sí y además excluyen a los procesos que buscan e insertan. Sólo un proceso de borrado puede acceder a la lista.
- Los procesos de inserción se excluyen entre sí, pero pueden coexistir con los de búsqueda. A su vez, los de búsqueda NO se excluyen entre sí

Modificaciones a SJN

- Suponga que se quiere cambiar el esquema de alocación SJN por uno LJN (longest JOB next). Analice el código y comente los cambios. ¿Cuál de los dos esquemas le parece más fair? ¿Cuál generará una cola mayor de procesos pendientes?.

- En el esquema SJN, suponga que lo quiere cambiar por un esquema FIFO. Analice el código y comente los cambios. ¿Cuál de los dos esquemas le parece más eficiente? ¿Más Fair ?.