Ejemplos de sincronización 7

Sistemas operativos y distribuidos

Gustavo Distel
gd@cs.uns.edu.ar

DCIC - UNS

Ejemplos de sincronización

Problemas clásicos

- Los problemas de sincronización que se detallan a continuación son ejemplos de una gran variedad de problemas de control de concurrencia.
- Estos problemas se utilizan para probar cada nuevo esquema de sincronización propuesto.
- Se utilizarán semáforos para la sincronización, ya que esa es la forma tradicional de presentar tales soluciones.

Ejemplos de sincronización

Problemas clásicos: el problema de los productores y consumidores (*The Bounded-Buffer Problem*)

- Se usa comúnmente para ilustrar el poder de las primitivas de sincronización.
- Los procesos de productor y consumidor comparten las siguientes estructuras de datos:

```
int n;
semaphore mutex = 1;
semaphore empty = n;
semaphore full = 0;
```

- **n**: número de *slots* en el *buffer*.
- **mutex**: semáforo binario que brinda exclusión mutua para la región crítica.
- `empty:` número de *slots* disponibles.
- `full`: número de elementos que existen en el buffer.

Ejemplos de sincronización

Problemas clásicos: el problema de los productores y consumidores

- Solución para un productor y un consumidor:

Productor

```
while (true) {
    item =  produce_item();
    wait(empty);
    insert_item(item);
    signal(full);
}
```

Consumidor

```
while (true) {
    wait(full);
    item = remove_item();
    signal(empty);
    consume_item(item);
}
```

Ejemplos de sincronización

Problemas clásicos: el problema de los productores y consumidores

- Solución para múltiples productores y consumidores:

Productor

```
while (true) {

    item =  produce_item();
    wait(empty);
    wait(mutex);
    insert_item(item);
    signal(mutex);
    signal(full);
}
```

Consumidor

```
while (true) {

    wait(full);
    wait(mutex);
    item = remove_item();
    signal(mutex);
    signal(empty);
    consume_item(item);

}
```

Ejemplos de sincronización

Problemas clásicos: el problema de los lectores-escritores (*The Readers–Writers Problem*)

- Considerar una base de datos compartida entre varios procesos concurrentes.
- Algunos de estos procesos leen la base de datos (lectores), mientras que otros la actualizan (escritores).
- No ocurre ningún problema si dos lectores acceden a los datos compartidos simultáneamente.
- Sin embargo, si un escritor y algún otro proceso (ya sea lector o escritor) acceden simultáneamente, podrían producirse errores.
- Para garantizar que no surjan estas dificultades se requiere que los escritores tengan acceso exclusivo a la base de datos mientras la escriben.
- El problema de los lectores y escritores tiene distintas variaciones y todas ellas involucran prioridades.
- Por ejemplo: supongamos que en una clase hay un pizarrón, donde solo una persona puede escribir a la vez, pero al mismo tiempo muchos lectores pueden leer el pizarrón.

Ejemplos de sincronización

Problemas clásicos: el problema de los lectores-escritores

- El primer problema de lectores y escritores
  - Requiere que ningún lector espere a menos que un escritor ya haya obtenido permiso para usar el objeto compartido.
  - Es decir, ningún lector debe esperar a que otros lectores terminen simplemente porque un escritor está esperando.
- El segundo problema de lectores-escritores
  - Requiere que, una vez que un escritor esté listo, ese escritor realice su escritura lo antes posible.
  - Si un escritor está esperando acceder al objeto ningún lector nuevo puede comenzar a leer.
- Una solución a cualquiera de ambos problemas puede resultar en inanición.
- En el primer caso, los escritores pueden caer en inanición mientras que en el segundo caso son los lectores.
- Por esta razón, se han propuesto otras variantes; como **el tercer problema de lectores-escritores**.

Ejemplos de sincronización

Problemas clásicos: el problema de los lectores-escritores

Escritor

```
while(true) {
    wait(mutex);
     // Escritura
    signal(mutex);
}
```

Lector

```
while(true) {
    wait(mutex)
    // Lectura
    signal(mutex)
}
```

- Solución `incorrecta o subóptima`:
  - Es posible que un hilo lector T_1 tenga el *lock*, y luego otro hilo lector T_2 solicite acceso.
  - Sería ineficiente que T_2 espere hasta que T_1 estuviera listo antes de comenzar su propia operación de lectura; en cambio,
  - Se debe permitir que T_2 lea el recurso junto con T_1 porque las lecturas no modifican los datos, es decir, las lecturas concurrentes son seguras.

Ejemplos de sincronización

Problemas clásicos: el primer problema de los lectores-escritores

- Solución al primer problema de lectores y escritores; **“preferencia lectores”**.

```
semaphore resource = 1;
semaphore rmutex   = 1;
int read_count = 0;
```

- Variable `read_count`: cantidad de hilos que están leyendo el objeto actualmente.
- Semáforo `rmutex`: utilizado para la exclusión mutua de la variable `read_count`.
- Semáforo `resource`: exclusión mutua para escritores.
  - También lo utiliza el primer o el último lector que ingresa o sale de la sección crítica.
  - No se utiliza por lectores que entran o salen mientras otros lectores están en sus secciones críticas.

Ejemplos de sincronización

Problemas clásicos: el primer problema de los lectores-escritores

Escritor

```
while(true) {

    wait(resource);

    // Escritura

    signal(resource);

}
```

Lector

```
while(true) {

    wait(rmutex);
    read_count++;
    if (read_count == 1)
        wait(resource);

    signal(rmutex);

    // Lectura

    wait(rmutex);
    read_count--;
    if (read_count == 0)
        signal(resource);

    signal(rmutex);

}
```

Ejemplos de sincronización

Problemas clásicos: el segundo problema de los lectores-escritores

- Solución al segundo problema de lectores y escritores; **“preferencia escritores”**.

```
semaphore rmutex   = 1;
semaphore wmutex   = 1;
semaphore readTry  = 1;
semaphore resource = 1;
int read_count  = 0;
int write_count = 0;
```

- Variables `read_count/write_count`: cantidad de hilos que están leyendo/escribiendo el objeto actualmente.
- Semáforo `rmutex/wmutex`: para la exclusión mutua de la variable `read_count/write_count`.
- Semáforo `resource`: exclusión mutua para escritores y primer lector.
- Semáforo `readTry`: otorga la preferencia a escritores ya que cada lector debe bloquearlo y liberarlo; sin embargo, solo el primer escritor debe esperarlo y solo el último escritor lo libera.

Ejemplos de sincronización

Problemas clásicos: el segundo problema de los lectores-escritores

Escritor

```
while(true) {

  wait(wmutex);
  write_count++;
  if (write_count == 1)
    wait(readTry);
  signal(wmutex);
  wait(resource);

  // Escritura

  signal(resource);
  wait(wmutex);
  write_count--;
  if (write_count == 0)
    signal(readTry);
  signal(wmutex);

}
```

Lector

```
while(true) {

  wait(readTry);
  wait(rmutex);
  read_count++;
  if (read_count == 1)
    wait(resource);
  signal(rmutex);
  signal(readTry);

// Lectura

  wait(rmutex);
  read_count--;
  if (read_count == 0)
    signal(resource);
  signal(rmutex);

}
```

Ejemplos de sincronización

Problemas clásicos: el tercer problema de los lectores-escritores

- Solución al tercer problema de lectores y escritores; “solución justa para lectores y escritores”.
  - Se agrega la restricción de no permitir que ningún hilo caiga en inanición; es decir, la acción de obtener el *lock* de los datos compartidos siempre terminará en un período de tiempo acotado.

```
semaphore resource_Access = 1;
semaphore rmutex = 1;
semaphore service_Queue = 1;
int read_count = 0;
```

- Variable `read_count`: cantidad de lectores accediendo actualmente al recurso.
- Semáforo `resource_Access`: control de acceso lectura/escritura al recurso.
- Semáforo `rmutex`: para sincronizar cambios en la variable compartida **read_count**.
- Semáforo `service_Queue`: aplica justicia, conservando el orden de las solicitudes (`los` ***signals*** `deben ser` *FIFO*)

Ejemplos de sincronización

Problemas clásicos: el tercer problema de los lectores-escritores

- Construcción, primer paso: Justicia.

Escritor

```
while(true) {
    wait(service_Queue);
    signal(service_Queue);
}
```

Lector

```
while(true){
    wait(service_Queue);
    signal(service_Queue);
}
```

- Como primer paso se necesita crear una cola justa entre lectores y escritores para evitar la inanición (*starvation*).
- Para lograr eso, se utiliza el semáforo llamado `service_Queue` que dará el orden de llegada.
- Cualquier hilo que solicite acceso al recurso tomará este semáforo y lo liberara tan pronto como ese hilo obtenga acceso al recurso.

Ejemplos de sincronización

Problemas clásicos: el tercer problema de los lectores-escritores

- Construcción, segundo paso: código del escritor (el más sencillo de ambos).

Escritor

```
while(true) {

    wait(service_Queue);
    wait(resource_Access);
    signal(service_Queue);

    // Escritura
    signal(resource_Access);

}
```

Lector

```
while(true){

    wait(service_Queue);

    signal(service_Queue);

}
```

- El escritor debe tener acceso exclusivo al recurso.
- El escritor solicitará el semáforo `resource_Access` antes de modificar el recurso.

Ejemplos de sincronización

Problemas clásicos: el tercer problema de los lectores-escritores

- Construcción, tercer paso: final.
- Múltiples lectores acceden simultáneamente al recurso.
- El primer lector que tenga acceso al recurso lo bloqueara, así ningún escritor acceda al mismo tiempo.
- El último lector liberará el recurso.

Escritor

```
while(true) {
    wait(service_Queue);
    wait(resource_Access);
    signal(service_Queue);
    // Escritura
    signal(resource_Access);
}
```

Lector

```
while(true){

    wait(service_Queue);
    wait(rmutex);
    if (read_count == 0)
        wait(resource_Access);
    read_count++;
    signal(service_Queue);
    signal(rmutex);

    // Lectura

    wait(rmutex);
    read_count--;
    if (read_count == 0)
        signal(resource_Access);
    signal(rmutex);
}
```

Ejemplos de sincronización

Problemas clásicos: el problema de los lectores-escritores

- En algunos sistemas los problema de lectores y escritores y sus soluciones se han generalizado para proporcionar *locks* de lector y escritor.
- La adquisición de un *lock* de lector-escritor requiere especificar su modo: acceso de lectura o escritura.
- Cuando un proceso solo desea leer datos compartidos, solicita el *lock* en modo lectura.
- Un proceso que desee modificar los datos compartidos debe solicitar el *lock* en modo escritura.
- Está permitido que múltiples procesos adquieran simultáneamente un *lock* lector-escritor en modo de lectura, pero solo un proceso puede adquirir el bloqueo para escribir, ya que se requiere acceso exclusivo para los escritores.

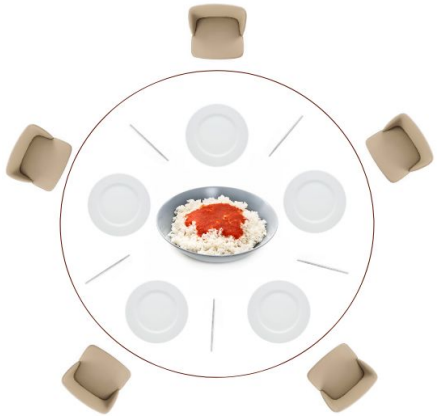
Ejemplos de sincronización

Problemas clásicos: el problema de los lectores-escritores

- Los *locks* de lector-escritor son útiles en las siguientes situaciones:
  - Aplicaciones en las cuales es fácil identificar qué procesos solo leen datos compartidos y cuales solo escriben datos compartidos.
  - En aplicaciones que tienen más lectores que escritores.
    - Esto se debe a que los *locks* de lector-escritor generalmente requieren más sobrecarga que los semáforos o *locks* de exclusión mutua.
    - La mayor concurrencia de permitir múltiples lectores compensa la sobrecarga involucrada en la configuración de *locks* lector-escritor.

Ejemplos de sincronización

Problemas clásicos: El problema de la cena de los filósofos (*The Dining-Philosophers Problem*)



- Consideremos cinco filósofos que pasan sus vidas `pensando` y `comiendo`.
- Los filósofos comparten una mesa circular con cinco sillas, cada una de las cuales pertenece a un filósofo. En el centro hay un tazón de arroz, y alrededor se colocan con cinco palillos.
- De vez en cuando, un filósofo tiene `hambre` y trata de recoger los dos palillos que están más cerca (los palillos que se encuentran entre él y sus vecinos izquierdo y derecho).
- Un filósofo puede recoger solo un palillo a la vez. Obviamente, no puede recoger un palillo que ya está en la mano de un vecino.
- Cuando un filósofo está `hambriento` y tiene dos palillos, come sin soltarlos. Cuando termina de comer, deja los palillos y comienza a `pensar` de nuevo.
- Esta es una representación simple de la necesidad de asignar varios recursos de una manera libre de *deadlocks* y libre de *starvation*.

Ejemplos de sincronización

Problemas clásicos: El problema de la cena de los filósofos

- Una solución simple es representar cada palillo con un semáforo.
- Un filósofo intenta agarrar un palillo ejecutando una operación `wait()` y suelta sus palillos ejecutando la operación `signal()`.

Datos compartidos

```
semaphore chopstick[5] = 1;
```

filósofo i

```
while (true) {
    wait(chopstick[i]);
    wait(chopstick[(i+1) % 5]);
    . . .
    /* eat for a while */
    . . .
    signal(chopstick[i]);
    signal(chopstick[(i+1) % 5]);
    . . .
    /* think for awhile */
    . . .
}
```

- Aunque esta solución garantiza que no haya dos vecinos comiendo simultáneamente, podría causar *deadlock*:
  - Si los cinco filósofos tienen hambre al mismo tiempo y cada uno agarra su palillo izquierdo todos los elementos del palillo serán `= 0`.
  - Cuando cada filósofo intente agarrar su palillo derecho, esperará por siempre.

Ejemplos de sincronización

Problemas clásicos: El problema de la cena de los filósofos

- Solución Tanenbaum:

```
#define N        5           /* número de filósofos */
#define LEFT     (i+N-1) % N/* número del i-ésimo vecino izquierdo */
#define RIGHT    (i+1) % N   /* número del i-ésimo vecino derecho */
#define THINKING 0           /* filósofo está pensando */
#define HUNGRY   1           /* filósofo está tratando de obtener palillos*/
#define EATING   2           /* filósofo está comiendo */

typedef int semaphore;       /* los semáforos son un tipo especial de int */
int state[N];                /* arreglo para realizar un seguimiento del estado
de todos */
semaphore mutex = 1;         /* exclusión mutua para regiones críticas */
semaphore s[N];              /* un semáforo por filósofo */
```

Ejemplos de sincronización

Problemas clásicos: El problema de la cena de los filósofos

```
void philosopher(int i){

    while (TRUE) {
        think();
        take_forks(i);
        eat();
        put_forks(i);
    }
}
```

```
void take_forks(int i){

    wait(&mutex);
    state[i] = HUNGRY;
    test(i);
    signal(&mutex);
    wait(&s[i]);
}
```

```
void put_forks(i){

    wait(&mutex);
    state[i] = THINKING;
    test(LEFT);
    test(RIGHT);
    signal(&mutex);
}
```

```
void test(i){ /* i: philosopher number, from 0 to N-1 */

    if(state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING){
        state[i] = EATING;
        signal(&s[i]);
  }
}
```

Para acceder a `state` o invocar a `test()`, un hilo debe mantener el `mutex`. Por lo tanto, la operación de chequear y actualizar el arreglo es atómica.

Ejemplos de sincronización

Problemas clásicos: El problema del barbero dormilón (*The barbershop problem*)

- Una barbería consta de una sala de espera con `n` sillas y la silla de barbero. Si no hay clientes que atender, el barbero se va a dormir. Si un cliente ingresa a la barbería y todas las sillas están ocupadas, abandona la tienda. Si el barbero está ocupado, pero hay sillas disponibles, entonces el cliente se sienta en una de las sillas libres. Si el barbero está dormido, el cliente lo despierta.
- Para hacer el problema un poco más concreto, se agrega la siguiente información:
  - Los hilos clientes deben invocar una función llamada `cortarseElPelo()`.
  - Si llega un hilo de cliente cuando la tienda está llena, puede invocar un ***`exit`***, que no retorna.
  - El hilo del peluquero debe invocar `cortarPelo()`.
  - Cuando el peluquero invoca `cortarPelo()`, debe haber exactamente un hilo que invoque `cortarseElPelo()` al mismo tiempo.

** El problema original fue propuesto por Dijkstra. Una variación del mismo aparece en Silberschatz and Galvin's Operating Systems Concepts. A su vez para esta clase se extrajo de The Little Book of Semaphores de Allen B. Downey versión 2.2.1.*

Ejemplos de sincronización

Problemas clásicos: El problema del barbero dormilón

```
int n = 4
int customers = 0
semaphore mutex = 1
semaphore customer = 0
semaphore barber = 0
semaphore customerDone = 0
semaphore barberDone = 0
```

- `n`: total de clientes que pueden estar en la tienda; tres en la sala de espera y uno en la silla
- `customers`: cuenta el número de clientes en la tienda, está protegido por `mutex`.
- El peluquero espera en `customer` hasta que un cliente ingresa a la tienda, luego el cliente espera en `barber` hasta que el peluquero le indica que tome asiento.
- Después del corte de pelo, el cliente señala a `customerDone` y espera en `barberDone`.

Ejemplos de sincronización

Problemas clásicos: El problema del barbero dormilón

Cliente

```
while (true){
    wait(mutex);
        if (customers == n)
            signal(mutex);
            exit();
        customers += 1;
    signal(mutex);

    signal(customer);
    wait(barber);

    cortarseElPelo();

    signal(customerDone);
    wait(barberDone);

    wait(mutex);
        customers -= 1;
    signal(mutex);
}
```

Barbero

```
wait(customer);
signal(barber);

cortarPelo();

wait(customerDone);
signal(barberDone);
```

Ejemplos de sincronización

Problemas clásicos: El problema de Papá Noel (*The Santa Claus problem*)

- Papá Noel duerme en su tienda en el Polo Norte y solo puede ser despertado por dos razones:
  - 1. Los nueve renos han regresado de sus vacaciones en el Pacífico Sur o,
  - 2. Alguno/s de los elfos tiene dificultades en la fabricación de juguetes.
- Para permitir que Papá Noel pueda dormir un poco, los elfos sólo pueden despertarlo cuando al menos tres de ellos tienen problemas. Cuando los tres elfos han resuelto sus problemas con Papá Noel, cualquier otro elfo que desee visitarlo debe esperar a que esos elfos regresen.
- Si Papá Noel se despierta y encuentra a los tres elfos esperando en la puerta de su tienda junto al último reno que ha vuelto del trópico, Papá Noel decide que los elfos pueden esperar hasta después de Navidad, porque es más importante preparar su trineo (se asume que los renos no quieren irse del trópico y, por lo tanto, permanecen allí hasta el último momento posible).
- El último reno en regresar informa su llegada a Papá Noel mientras que los otros renos esperan en una cabaña antes de ser enganchados al trineo.

Ejemplos de sincronización

Problemas clásicos: El problema de Papá Noel

- Algunas restricciones adicionales son:
  - Después de que llega el noveno reno y por cada uno de los renos, Papá Noel debe invocar a `prepararTrineo()`, y luego los nueve renos deben invocar `obtenerRienda()`.
  - Después de que llega el tercer elfo, Papá Noel debe invocar `ayudarElfos()`; al mismo tiempo, los tres elfos deben invocar `obtenerAyuda()`.
  - Los tres elfos deben invocar `obtenerAyuda()` antes de que ingresen otros elfos.
  - Las funciones mencionadas son genéricas e indican una acción.
- Papá Noel ejecuta un bucle para poder ayudar a muchos grupos de elfos. Podemos asumir que hay exactamente 9 renos, pero puede haber una cantidad indefinida de elfos. Hay un hilo separado para Papá Noel, cada elfo y cada reno.

** Este problema es de William Stallings's Operating Systems, pero el se lo atribuye a John Trono de St. Michael's College en Vermont. A su vez para esta clase se extrajo de The Little Book of Semaphores de Allen B. Downey versión 2.2.1.*

Ejemplos de sincronización

Problemas clásicos: El problema de Papá Noel

```
int elfos = 0
int reno = 0
semaphore papaSem = 0
semaphore renoSem = 0
semaphore elfTex = 1
semaphore mutex = 1
```

- `elfos` y `reno`: contadores protegidos por `mutex`.
- Papá Noel espera en `papaSem` hasta que un elfo o un reno lo señalice.
- Los renos esperan a `renoSem` hasta que Papá Noel los señalice.
- Los elfos usan `elfTex` para evitar que entren elfos adicionales mientras se ayuda a tres elfos.

Ejemplos de sincronización

Problemas clásicos: El problema de Papá Noel

Papá Noel

```
while(true){
  wait(papaSem);
  wait(mutex);
    if (reno >= 9)
      prepararTrineo();
      signal(renoSem,9);
      reno -= 9;
    else (if elfos == 3)
      ayudarElfos();
  signal(mutex);
}
```

Reno

```
while(true){
  wait(mutex);
    reno += 1;
    if (reno == 9)
      signal(papaSem);
  signal(mutex);

  wait(renoSem);
  obtenerRienda();
}
```

Elfos

```
while(true){
  wait(elfTex);
  wait(mutex);
    elfos += 1;
    if (elfos == 3)
      signal(papaSem);
    else
      signal(elfTex);
  signal(mutex);

  obtenerAyuda();

  wait(mutex);
    elfos -= 1;
    if (elfos == 0)
      signal(elfTex);
  signal(mutex);
}
```

Sincronización dentro del *Kernel*

- Sincronización en Windows
- Sincronización en Linux

Sincronización *POSIX*

- *POSIX Mutex Locks*
- *POSIX Semaphores*
  - *POSIX Named Semaphores*
  - *POSIX Unnamed Semaphores*
- *POSIX Condition Variables*