Sistemas Operativos

Uso de semáforos

Curso 2024

Facultad de Ingeniería, UDELAR

Agenda

1. Acceso mutuo a una sección crítica
2. Secuencialización/señalización
3. Problema de los productores y consumidores
4. Problema de los lectores y escritores
5. Problema de los filósofos comensales

Acceso mutuo a una sección crítica

Acceso mutuo a una sección crítica

- Se tienen dos procesos que necesitan el acceso mutuo-excluido a una región crítica.

```
INIT(S, 1);

procedure Alicia()
   repeat
      P(S);
      acceder_A;
      V(S);
      otras tareas;
   until False
end procedure
```

```
procedure Bernardo()
   repeat
      P(S);
      acceder_B;
      V(S);
      otras tareas;
   until False
end procedure
```

Acceso mutuo a una sección crítica

- La inicialización del semáforo es la clave para garantizar el acceso mutuo-excluido a la región crítica.

```
INIT(S, 1);                    ▷ Inicializar el semáforo a 1

procedure Alicia()                  procedure Bernardo()
   repeat                              repeat
      P(S);                               P(S);
      acceder_A;                          acceder_B;
      V(S);                               V(S);
      otras tareas;                       otras tareas;
   until False                         until False
end procedure                       end procedure
```

Secuencialización/señalización

Secuencialización/señalización

- Dos procesos se ejecuten en una secuencia predeterminada (P1A – P2 – P1B).
- Se usan dos semáforos binarios inicializados en 0.
- P1 ejecuta P1A, realiza la operación V sobre S1 y luego la operación P sobre S2.
- P2 ejecuta la operación P sobre S1, solo inicia la ejecución cuando P1 lo habilita.
- Al finaliza de procesar, P2 realiza la operación V sobre S2.
- Este modo de uso se denomina señalización (signaling): permite que un proceso o hilo de ejecución le comunique a otro que algo ha sucedido.

Secuencialización/señalización

- La inicialización de los semáforos es clave para garantizar el acceso mutuo-excluido a la región crítica.
- El orden correcto de las operaciones P y V asegura la ejecución en la secuencia indicada.

```
INIT(S1, 0);   ▷ Inicializar el semáforo S1 a 0
INIT(S2, 0);   ▷ Inicializar el semáforo S2 a 0
```

```
procedure P1()
   P1_A();
   V(S1);
   P(S2);
   P1_B();
end procedure
```

```
procedure P2()
   P(S1);
   process()
   V(S2);
end procedure
```

Problema de los productores y consumidores

Problema de los productores y consumidores

- Se tiene un conjunto de procesos Productor que almacenan datos en un buffer y un conjunto de procesos Consumidor que sacan datos del buffer.
- Ambos procesos deben trabajar cooperativamente, en ejecución simultánea

Problema de los productores y consumidores: variante I

- Se tiene un conjunto de procesos Productor que almacenan datos en un buffer y un conjunto de procesos Consumidor que sacan datos del buffer.
- En su variante inicial, el problema se define con un buffer infinito.

Un consumidor solo puede sacar datos del buffer cuando el buffer no está vacío

Problema de los productores y consumidores: variante I

```
procedure productor()
   count = 0
   repeat
      producir();
      guardar();
      count++;
      if (count == 1) then
         despertar(consumidor);
      end if
   until False
end procedure
```

```
procedure consumidor()
   repeat
      if (count == 0) then
         esperar()
      end if
      tomar();
      count--;
      consumir();
   until False
end procedure
```

Qué problema(s) tiene esta solución?

Problema de los productores y consumidores: variante I

Problema 1

1. El consumidor consulta la variable count, nota que es cero y pasa a ejecutar el if para esperar.
2. Justo antes de invocar a la función esperar(), el consumidor es interrumpido y el productor comienza a trabajar.
3. El productor produce un dato, lo agrega al buffer e incrementa count. Como el buffer estaba vacío, intenta despertar al consumidor.
4. El consumidor aún no está durmiendo, la invocación a despertar() no tiene acción.
5. El consumidor resume su trabajo y espera por siempre, ya que el productor solo lo despierta si el valor de count es 1.

Problema de los productores y consumidores: variante I

Problema 2

Problemas de acceso compartido a la variable count por parte de productores y consumidores

Problema de los productores y consumidores: variante I

Solución: usar semáforos

N: cantidad de elementos en el buffer

```
INIT(N, 0);    ▷ Se asume que el buffer inicia vacío

procedure productor()
   repeat
      producir();
      guardar();
      V(N);
   until False
end procedure
```

```
procedure consumidor()
   repeat
      P(N);
      tomar();
      consumir();
   until False
end procedure
```

Problema de los productores y consumidores: variante II

- Se tiene un conjunto de procesos Productor que almacenan datos en un buffer y un conjunto de procesos Consumidor que sacan datos del buffer.
- El problema se define con un buffer infinito.

Un consumidor solo puede sacar datos del buffer cuando el buffer no está vacío

El acceso al buffer debe realizarse en exclusión mutua.

Problema de los productores y consumidores: variante II

S: semáforo para exclusión mutua

N: cantidad de elementos en el buffer

```
INIT(S, 1);                                  ▷ Exclusión mutua
INIT(N, 0);    ▷ Se asume que el buffer inicia vacío
```

```
procedure productor()
   repeat
      producir();
      P(S);
      guardar();
      V(S);
      V(N);
   until False
end procedure
```

```
procedure consumidor()
   repeat
      P(N);
      P(S);
      tomar();
      V(S);
      consumir();
   until False
end procedure
```

Problema de los productores y consumidores: variante III

- Se tiene un conjunto de procesos Productor que almacenan datos en un buffer y un conjunto de procesos Consumidor que sacan datos del buffer.
- Versión más realista del problema: el buffer es finito.

Un productor solo puede almacenar datos en el buffer cuando el buffer no está lleno

Un consumidor solo puede sacar datos del buffer cuando el buffer no está vacío

El acceso al buffer debe realizarse en exclusión mutua.

Problema de los productores y consumidores: variante III

S: semáforo para exclusión mutua, N: cantidad de elementos en el buffer
E: espacios vacíos en el buffer (N + E = TAM_BUFFER)

```
INIT(S, 1);                              ▷ Exclusión mutua
INIT(N, 0);    ▷ Se asume que el buffer inicia vacío
INIT(E, TAM_BUFFER);          ▷ Se asume que el buffer
inicia vacío
```

```
procedure productor()
   repeat
      producir();
      P(E);
      P(S);
      guardar();
      V(S);
      V(N);
   until False
end procedure
```

```
procedure consumidor()
   repeat
      P(N);
      P(S);
      tomar();
      V(S);
      V(E);
      consumir();
   until False
end procedure
```

Problema de los lectores y escritores

Problema de los lectores y escritores

- Se tiene un conjunto de procesos Lectores que acceden en modo lectura a un recurso (e. g. una base de datos)
- Se tiene un conjunto de procesos Escritores que acceden en modo escritura al mismo recurso.

Varios lectores pueden acceder al mismo tiempo al recurso, pero el acceso de los escritores debe ser de a uno por vez.

Comportamiento asimétrico de lectores y escritores.

Problema de los lectores y escritores: variante I

Motivación

- El acceso de lectura y escritura a una variable no es una operación atómica
- Ejemplo: sumar uno a una variable
  1. MOV (de memoria a registro)
  2. INC (para sumar uno)
  3. MOV (de registro a memoria)
- Si se ejecutan varias sumas en forma concurrente, pueden dar resultados equivocados
- La lectura de la variable consiste solamente en un MOV y puede hacerse en paralelo

Problema de los lectores y escritores: variante I

```
INIT(E,1);  ▷ Mutex acceso
INIT(mL,1);         ▷ Mutex
#lectores
cantLect = 0;
```

```
procedure Escritor()
   repeat
      P(E);
      escribir();
      V(E);
   until False
end procedure
```

```
procedure Lector()
   repeat
      P(mL);
      cantLect := cantLect + 1;
      if cantLect = 1 then
         P(E);    ▷ Acceso al recurso
      end if
      V(mL);
      leer();
      P(mL);
      cantLect := cantLect - 1;
      if cantLect = 0 then
         V(E);
      end if
      V(mL);
   until False
end procedure
```

Problema de los lectores y escritores: variante I

- En la variante I del problema de los lectores y escritores, los lectores tienen prioridad sobre los escritores.
- Se puede generar posposición indefinida de escritores si hay muchos lectores accediendo al recurso.
- Con la solución implementada, los escritores no tienen cómo acceder al recurso hasta que el último lector lo libere (no hay más lectores accediendo al recurso).

Problema de los lectores y escritores: variante II

- Se tiene un conjunto de procesos lectores que acceden en modo lectura a un recurso (e. g. una base de datos)
- Se tiene un conjunto de procesos escritores que acceden en modo escritura al mismo recurso.

Varios lectores pueden acceder al mismo tiempo al recurso, pero el acceso de los escritores debe ser de a uno por vez.

Comportamiento asimétrico de lectores y escritores.

Cuando hay lectores leyendo y llega un escritor, éste debe tener prioridad sobre los próximos lectores que lleguen.

Problema de los lectores y escritores: variante II

```
      INIT(E, 1);                                  ▷ Mutex buffer
      INIT(mL, 1);                               ▷ Mutex #lectores
      INIT(mE, 1);                             ▷ Mutex #escritores
      INIT(try, 1);                ▷ Bloqueo de nuevos lectores
      cantEsc = cantLect = 0;

procedure Escritor()
   repeat
      P(mE);                                       P(mE);
      cantEsc := cantEsc + 1;                      cantEsc := cantEsc - 1;
      if cantEsc = 1 then                          if cantEsc = 0 then
         P(try);                    ▷                 V(try);
bloquear lectores                                  end if
      end if                                       V(mE);
      V(mE);                                     until False
      P(E);                                    end procedure
      escribir();
      V(E);
```

Problema de los lectores y escritores: variante II

```
procedure Lector()
   repeat
      P(try);
      P(mL);
      cantLect++;
      if cantLect = 1 then
         P(E);
      end if
      V(mL);
      V(try);
      leer();
      P(mL);
      cantLect--;
      if cantLect = 0 then
         V(E);
      end if
      V(mL);
   until False
end procedure
```

Problema de los filósofos comensales

Problema de los filósofos comensales

- Varios filósofos viven sentados alrededor de una mesa circular ... pensando y comiendo.
- En la mesa se dispone de N platos y N tenedores ... pero para comer cada filósofo necesita utilizar dos tenedores
- Para comer, un filósofo toma primero un tenedor y luego el otro.

Figura 1: Caso de estudio: problema de los filósofos comensales, $N = 5$

Problema de los filósofos comensales: solución inicial

```
INIT(T[i], 1);                                  ▷ Mutex tenedor i
procedure filosofo_i()
   repeat
      P(T[i]);
      P(T[i+1 mod 5]);
      comer();
      V(T[i]);
      V(T[i+1 mod 5]);
      pensar();
   until False
end procedure
```

Problema:

- Deadlock si los cinco filósofos quieren comer a la vez.

Problema de los filósofos comensales: soluciones

Soluciones:

- Comer por turnos (token cíclico) ... ineficiente para N grande
- Comer en varios turnos (N/2 tokens) ... ¿cuándo cambiarlos? (se debe conocer con exactitud los tiempos medios de procesamiento y de uso del recurso)
- Cuando un filósofo quiere comer se pone en la cola de los dos tenedores que necesita ... genera deadlock. Debe resolverse el conflicto (e.g., espera de tiempo aleatorio).
- Solo permitir comer a cuatro filósofos a la vez.
- Un administrador que solo permita tomar los dos tenedores juntos.
- Que los tenedores solo se puedan tomar en un determinado orden.
- Incluir comunicaciones entre los filósofos o permitir asimetría

Problema de los filósofos comensales: solución

Restricción: sólo comen cuatro filósofos a la vez

```
INIT(N, 4);                                  ▷ Mutex comensales
INIT(T[i], 1);                               ▷ Mutex tenedor i
procedure filosofo_i()
   repeat
      P(N);
      P(T[i]);
      P(T[i+1 mod 5]);
      comer();
      V(T[i]);
      V(T[i+1 mod 5]);
      V(N);
      pensar();
   until False
end procedure
```