Sistemas Operativos

Semáforos

Curso 2025

Facultad de Ingeniería, UDELAR

Agenda

Semáforos

1. Exclusión mutua y errores comunes
2. Semáforos binarios
3. Usos de semáforos
4. Acceso mutuo a una sección crítica
5. Secuencialización/señalización
6. Grafos de precedencias con semáforos
7. Problema de los productores y consumidores
8. Problema de los lectores y escritores
9. Problema de los filósofos comensales

Semáforos

Introducción

- Son una herramienta para sincronizar procesos.
- Propuesta originalmente por Dijkstra y su equipo en los principios de los años 1960.
- Se puede implementar con soporte del sistema operativo sin usar busy waiting.

Definición

- Un semáforo es un tipo de dato *integer* que se manipula mediante tres funciones:
  - Init(semáforo, valor)
  - wait(semáforo) o P(semáforo)
    - Prolaag -probeer te verlagen- (intentar reducir)
    - Proberen (intentar)
  - signal(semáforo) o V(semáforo)
    - Verhogen (aumentar)
- Luego de inicializado, toda modificación al dato *integer* se realiza únicamente mediante las funciones P o V.
- Ambas funciones (P y V) se ejecutan de forma atómica (indivisible, en una región crítica).

Implementación

```
s: integer;
(o s: semáforo;)

procedure INIT(s, 3)
   s := 3;
end procedure

procedure V(s)
   s := s + 1;
end procedure
```

```
procedure P(s)
   while s ≤ o do ;
   end while
   s := s - 1;
end procedure
```

Implementación

- No hay orden establecido para despertar los procesos que están esperando (cualquier implementación es válida)
- La implementación presentada usa busy waiting: implica un gran overhead, en especial al usar un único procesador.
- También llamados spinlock, tienen como ventaja que no necesitan cambios de contexto.
- Se pueden implementar con cambios de contexto con soporte del sistema operativo.

Implementación sin busy waiting

```
procedure P(s)
   if s > 0 then
      s := s - 1;
   else
      se bloquea el proceso
   end if
end procedure

procedure V(s)
   if hay proceso suspendido then
      se despierta un proceso    ▷ pasa a listo
   else
      s := s + 1;
   end if
end procedure
```

Implementación en código

```
typedef struct {
        int valor;
        struct process *lista;
} semaforo;
```

```
procedure V(S)
   S→valor := S→valor + 1;
   if  (S→valor ≤ 0)  then
      remover proc de S→lista;
      despertar(P);
   end if
end procedure
```

```
procedure P(S)
   S→valor := S→valor - 1;
   if  (S→valor < 0)  then
      agregar proc a S→lista;
      bloquear();
   end if
end procedure
```

Esta implementación permite valores negativos del semáforo.

Exclusión mutua y errores comunes

Utilidad: exclusión mutua

Resolver el problema de exclusión mutua para una sección donde pueden entrar hasta k procesos.

```
procedure Proc_1
   P(s);
   realiza tareas 1
   V(s);
end procedure
```

```
procedure Proc_N
   P(s);
   realiza tareas N
   V(s);
end procedure
```

```
procedure Main
   INIT(s, k);
   Cobegin
      proc_1;  … proc_N;
   Coend
end procedure
```

Errores comunes

Semáforo mal inicializado: genera deadlock.

```
procedure Proc_1
   P(s);
   realiza tareas 1
   V(s);
end procedure
```

```
procedure Proc_N
   P(s);
   realiza tareas N
   V(s);
end procedure
```

```
procedure Main
   INIT(s, 0);
   Cobegin
      proc_1;  …  proc_N;
   Coend
end procedure
```

Errores comunes

Invertir P() y V(): permite que más de k procesos ingresen a la región crítica

```
procedure Proc_1
   V(s);
   realiza tareas 1
   P(s);
end procedure
```

```
procedure Proc_N
   P(s);
   realiza tareas N
   V(s);
end procedure
```

```
procedure Main
   INIT(s, 2);
   Cobegin
      proc_1;  … proc_N;
   Coend
end procedure
```

Conclusiones

- Los semáforos son una herramienta muy poderosa para sincronizar procesos y resolver problemas de exclusión mutua.
- Es muy fácil cometer errores al utilizar semáforos y que el sistema quede en estados inconsistentes.

Semáforos binarios

Semáforos binarios

- Los semáforos binarios solo pueden valer 0 o 1.
- **V** es una operación binaria: V en un semáforo que vale 1 lo deja en 1.
- Los semáforos binarios son equivalentes a los semáforos de conteo.
    - Es posible implementar semáforos binarios con semáforos de conteo y viceversa.

Implementación con busy waiting

```
procedure P(s)
   while not TestAndReset(s) do
   end while
end procedure

procedure V(s)
   s := True;
end procedure

function TestAndReset(var)          ▷ clase pasada
   ret := var;
   var := False;
   return ret;
end function
```

Semáforos binarios con semáforos de conteo

```
procedure Init(val)
   INIT(mutex, 1);
   INIT(wait, 0);
   free := val;
   espera := 0;
end procedure
```

```
procedure V_bin()
   P(mutex);
   if espera > 0 then
      espera := espera - 1;
      V(wait);
   else
      free := True;
      V(mutex);
   end if
end procedure
```

```
procedure P_bin()
   P(mutex);
   if not free then
      espera := espera + 1;
      V(mutex);
      P(wait);
   else
      free := False;
   end if
   V(mutex);
end procedure
```

Semáforos de conteo con semáforos binarios

```
procedure Init(k)
   INIT(mutex, 1);
   INIT(wait, 0);
   c := k;
end procedure
```

```
procedure V_cont()
   P(mutex);
   c := c + 1;
   if c ≤ 0 then
      V(wait);
   else
      V(mutex);
   end if
end procedure
```

```
procedure P_cont()
   P(mutex);
   c := c - 1;
   if c < 0 then
      V(mutex);
      P(wait);
   end if
   V(mutex);
end procedure
```

Usos de semáforos

Acceso mutuo a una sección crítica

Acceso mutuo a una sección crítica

- Sean 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/señalizacion

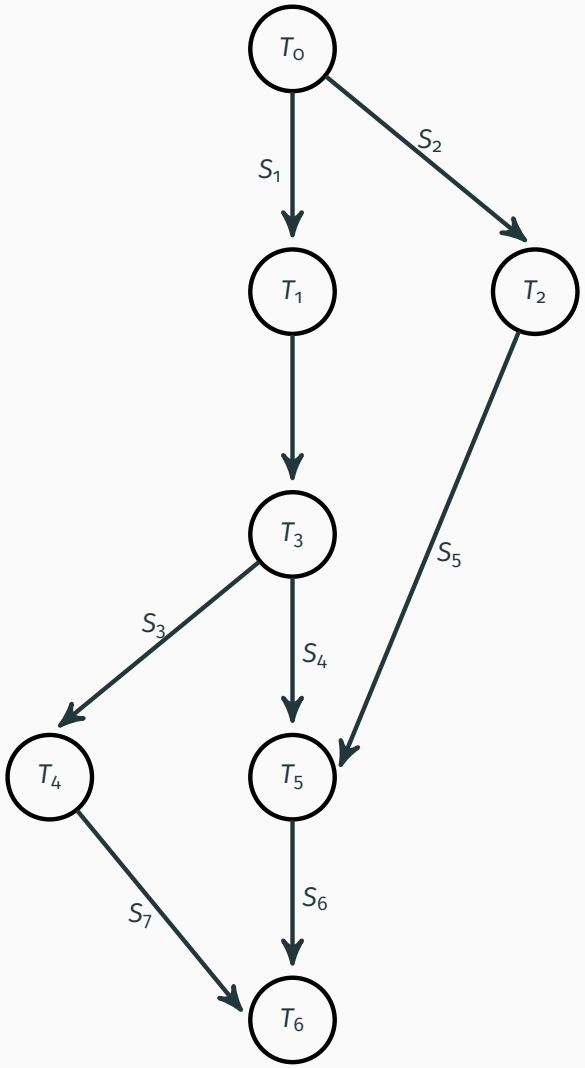
- La inicialización de los semáforos es clave para garantizar el acceso mutuo-excluido a la región crítica.
- El orden correcto de las operaciones P y V asegura la ejecución en la secuencia indicada.

```
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
```

Grafos de precedencias con semáforos

Grafos de precedencia con semáforos

Implementar las sincronizaciones con semáforos.



Solución con semáforos binarios

```
procedure grafo
   s1, s2, s3, s4, s5, s6, s7: semaforo;
   INIT(s1,0); INIT(s2,0); INIT(s3,0); INIT(s4,0);
   INIT(s5,0); INIT(s6,0); INIT(s7,0);
   Cobegin
      Begin T0; V(s1); V(s2); End
      Begin P(s1); T1; T3; V(s3); V(s4); End
      Begin P(s2); T2; V(s5); End
      Begin P(s3); T4; V(s7); End
      Begin P(s4); P(s5); T5; V(s6); End
      Begin P(s6); P(s7); T6; End
   Coend
end procedure
```

Solución con un único semáforo de conteo

```
procedure grafo
   s: semaforo;
   INIT(s, 0);
   T0;
   Cobegin
      Begin
         T1; T3; V(s); T4;
      End
      Begin
         T2; V(s);
      End
      Begin
         P(s); P(s); T5;
      End
   Coend
   T6;
end procedure
```

Solución con un único semáforo binario

```
procedure grafo
   s: semaforo;
   INIT(s, 0);
   T0;
   Cobegin
      Begin
         T1; T3;
         Cobegin
            T4;
            Begin P(s); T5; End
         Coend
      End
      Begin T2; V(s); End
   Coend
   T6;
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 cooperativametne, 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);                     ▷ 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 = 0; cantLect = 0;
```

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

Problema de los lectores y escritores: variante II

```
procedure Lector()
   repeat
      P(try);
      P(mL);

cantLect := cantLect + 1;
      if cantLect = 1 then
         P(E);
      end if
      V(mL);
      V(try);
      leer();
      P(mL);
```

```
cantLect := cantLect - 1;
      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
```