

Sistemas Operativos

Semáforos

Curso 2021

Facultad de Ingeniería, UDELAR

Agenda

1. Semáforos de conteo
2. Utilidades y problemas
3. Semáforos binarios
4. Extendiendo el poder de otras herramientas

Semáforos de conteo

- Son una herramienta para sincronizar procesos.
- Propuesta originalmente por Dijkstra y su equipo en los principios de los años 60's.
- Se puede implementar con soporte del SO sin usar busy waiting.

Definición

- Un **semáforo** es un tipo de dato *integer* que se manipula mediante tres funciones:
 - INIT
 - wait o P
 - Prolaag -probeer te verlagen- (intentar reducir)
 - Proberen (intentar)
 - signal o V
 - 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 indivisible (región crítica).

Implementación

```
s: integer;
```

```
procedure INIT(s, 3)
```

```
    s = 3;
```

```
end procedure
```

```
procedure V(s)
```

```
    s = s + 1;
```

```
end procedure
```

```
procedure P(s)
```

```
    while s <= 0 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 vista usa busy waiting.
- Esto implica un importante overhead, en especial en casos de un único procesador.
- También llamados **spinlock**, tienen como ventaja que no necesitan cambios de contexto.
- Se pueden implementar con cambios de contextos con soporte del SO.

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 value;  
    struct process *list;  
} semáforo
```

```
procedure V(S)  
    S->value = S->value +  
    1;  
    if (S->value <= 0)  
    then  
        remove P from  
        S->list;  
        wakeup(P);  
    end if  
end procedure
```

```
procedure P(S)  
    S->value = S->value -  
    1;  
    if (S->value < 0)  
    then  
        add process to  
        S->list;  
        block();  
    end if  
end procedure
```

Esta implementación permite valores negativos.

Utilidades y problemas

Resolver el problema de mutuo-exclusión para una sección donde pueden entrar hasta 2 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, 2);
    Cobegin
        proc_1; ... proc_N;
    Coend
end procedure
```

Errores comunes

Semáforo mal inicializado → 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 \rightarrow entran más de 2

```
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
```

- Los **semáforos** son una herramienta muy poderosa para sincronizar (procesos).
- Es muy fácil cometer errores al utilizar esta herramienta y quedar en estados inconsistentes.

Semáforos binarios

Semáforos binarios

- Los **semáforos binarios** solo pueden valer 0 o 1.
- Hacer **V** en un semáforo que vale 1 lo deja en 1.
- Se puede demostrar que son equivalentes a los semáforos de conteo.
 - Implementando semáforos binarios con semáforos de conteo y viceversa.

Semáforos de conteo → binarios

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

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

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

Semáforos binarios → de conteo

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

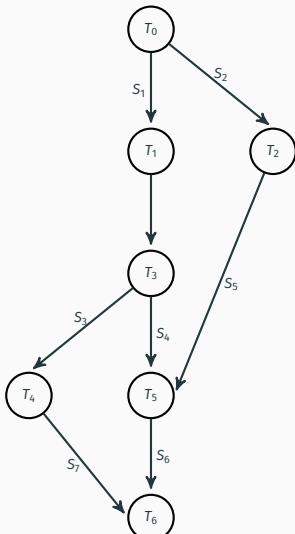
```
procedure Vcont()
  P(mutex);
  c = c + 1;
  if c <= 0 then
    V(wait);
  else
    V(mutex);
  end if
end procedure
```

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

Extendiendo el poder de otras herramientas

Cobegin-coend con semáforos

Dado el grafo de la clase 1, implementar las sincronizaciones con semáforos.



Cobegin-coend con semáforos (binarios)

```
procedure grafo
  INIT(s1, 0); INIT(s2, 0); INIT(s3, 0); INIT(s4, 0);
INIT(s5, 0);
  INIT(s6, 0); INIT(s7, 0);
  Cobegin
    Begin
       $T_0$ ; V(s1); V(s2);
    End
    Begin
      P(s1);  $T_1$ ;  $T_3$ ; V(s3); V(s4);
    End
    Begin
      P(s2);  $T_2$ ; V(s5);
    End
    Begin
      P(s3);  $T_4$ ; V(s7);
    End
    Begin
      P(s4); P(s5);  $T_5$ ; V(s6);
    End
    Begin
      P(s6); P(s7);  $T_6$ ;
```

Con un solo semáforo de conteo

```
procedure grafo
  INIT(s, 0);
   $T_0$ ;
  Cobegin
    Begin
       $T_1$ ;  $T_3$ ; V(s);  $T_4$ ;
    End
    Begin
       $T_2$ ; V(s);
    End
    Begin
      P(s); P(s);  $T_5$ ;
    End
  Coend
   $T_6$ ;
end procedure
```

Con un solo semáforo binario

```
procedure grafo
  INIT(s, 0);
   $T_0$ ;
  Cobegin
    Begin
       $T_1$ ;  $T_3$ ;
      Cobegin
         $T_4$ ;
        Begin
           $P(s)$ ;  $T_5$ ;
        End
      Coend
    End
    Begin
       $T_2$ ;  $V(s)$ ;
    End
  Coend
```