Explicación Práctica de Semáforos

Ejercicios

EJERCICIO 2

Hay C chicos y hay una bolsa con caramelos **limitada a N caramelos**. Los chico de a UNO van sacando de a UN caramelo y lo comen. Los chicos deben llevar la cuenta de cuantos caramelos se han tomado de la bolsa.

Comenzamos modificando la solución del Ejercicio 1 para que no intenten sacar más caramelos si la bolsa quedó vacía (*cant = N*)

```
int cant = 0;
sem mutex = 1;

Process Chico[id: 0..C-1]
{  while (cant < N)
      {  P(mutex);
         -- tomar caramelo
         cant = cant + 1;
         V(mutex);
         -- comer caramelo
      }
}
```

Se podrán sacar más de N caramelos.

EJERCICIO 2

El chequeo de la condición que indica que se debe tomar otro caramelo se debe proteger en una SC que también incluya la modificación de esa condición → en este caso el chequeo y *cant = cant +1*

```
int cant = 0;
sem mutex = 1;

Process Chico[id: 0..C-1]
{ P(mutex);
  while (cant < N)
     { -- tomar caramelo
       cant = cant + 1;
       -- comer caramelo
     }
  V(mutex);
}
```

Un único chico tomará los N caramelos

```
int cant = 0;
sem mutex = 1;

Process Chico[id: 0..C-1]
{ P(mutex);
  while (cant < N)
     { -- tomar caramelo
       cant = cant + 1;
       V(mutex);
       -- comer caramelo
     }
}
```

Libera la SC y nunca más asegura la EM

```
int cant = 0;
sem mutex = 1;

Process Chico[id: 0..C-1]
{ P(mutex);
  while (cant < N)
     { -- tomar caramelo
       cant = cant + 1;
       V(mutex);
       -- comer caramelo
       P(mutex);
     }
}
```

Sólo un proceso termina, el resto se bloquea.

EJERCICIO 2

Al salir del *while* se debe liberar la SC para que otro proceso pueda acceder a ella y darse cuenta de que debe terminar su procesamiento.

```
int cant = 0;
sem mutex = 1;

Process Chico[id: 0..C-1]
{ P(mutex);
  while (cant < N)
      {  -- tomar caramelo
         cant = cant + 1;
         V(mutex);
         -- comer caramelo
         P(mutex);
      }
  V(mutex);
}
```

EJERCICIO 5

En una empresa de genética hay N clientes que envían secuencias de ADN para que sean analizadas y esperan los resultados para poder continuar. Para resolver estos análisis la empresa cuenta con 2 servidores que van alternando su uso para no exigirlos de más (en todo momento uno está trabajando y el otro descansando); cada 5 horas cambia en servidor con el que se trabaja. El servidor que está trabajando, toma un pedido (de a uno de acuerdo al orden de llegada de los mismos), lo resuelve y devuelve el resultado al cliente correspondiente. Cuando terminan las 5 horas se intercambian los servidores que atienden los pedidos. Si al terminar las 5 horas el servidor se encuentre atendiendo un pedido, lo termina y luego se intercambian los servidores.

Nos basamos en la solución del ejercicio 4 para empezar. Los clientes no deberán modificarse, a ellos no le importa quien lo atiende. Hay que modificar el servidor y agregar un proceso *reloj* para que cuente las 5 horas de cada servidor.

EJERCICIO 5

¿Cómo resolvemos el reloj?

```
sem mutex = 1,   pedidos = 0,    espera[N]= ([N] 0);
int resultados[N];     cola C;

Process Cliente[id: 0..N-1]
{ secuencia S;
  while (true)
    { --generar secuencia S
      P(mutex);
      push(C, (id, S));
      V(mutex);
      V(pedidos);
      P(espera[id]);
      --ver resultado de resultados[id]
    }
}

Process Servidor[id: 0..1]
{ secuencia sec;   int aux;
  while (true)
    {   espera su turno
        inicia reloj
        while (no termine el tiempo)
          { P(pedidos);
            P(mutex);
            pop(C, (aux, sec));
            V(mutex);
            resultados[aux] = resolver(sec);
            V(espera[aux]);
          }
    }
}

Process Reloj
{  while (true)
     { espera inicio
       delay(5 hs);
       avisa final del tiempo
     }
}
```

EJERCICIO 5

Usaremos un semáforo *inicio* para avisar al reloj que debe comenzar a correr las 5 horas. Una variable booleana *FinTiempo* para indicar que el tiempo termino.

```
sem mutex = 1,  pedidos = 0,  espera[N] = ([N] 0),  inicio = 0;
int resultados[N];   cola C;   bool finTiempo = false;
```

Process Cliente[id: 0..N-1]

```
{ secuencia S;
  while (true)
    { --generar secuencia S
      P(mutex);
      push(C, (id, S));
      V(mutex);
      V(pedidos);
      P(espera[id]);
      --ver resultado de resultados[id]
    }
}
```

Process Servidor[id: 0..1]

```
{ secuencia sec;  int aux;
  while (true)
    {  espera su turno
       inicia reloj
       while (no termine el tiempo)
         { P(pedidos);
           P(mutex);
           pop(C, (aux, sec));
           V(mutex);
           resultados[aux] = resolver(sec);
           V(espera[aux]);
         }
    }
}
```

Como manejamos el turno de cada servidor

Process Reloj

```
{  while (true)
     { P(inicio);
       delay(5 hs);
       finTiempo = true;
       V(pedidos);
     }
}
```

El servidor actual puede esperar en un ÚNICO semáforo tanto el pedido de un cliente como el fin del reloj → se le avisa por medio del semáforo *pedidos*

EJERCICIO 5

Cada servidor tendrá un semáforo *turno* donde se demora hasta que deba trabajar, uno inicializado en 1 (el que inicia trabajando) y el otro en 0 (el que inicia dormido).

```
sem mutex = 1,  pedidos = 0,  espera[N] = ([N] 0),  inicio = 0,  turno[2] = (1, 0);
int resultados[N];   cola C;   bool finTiempo = false;
```

Process Cliente[id: 0..N-1]

```
{ secuencia S;
  while (true)
    { --generar secuencia S
      P(mutex);
      push(C, (id, S));
      V(mutex);
      V(pedidos);
      P(espera[id]);
      --ver resultado de resultados[id]
    }
}
```

Process Servidor[id: 0..1]

```
{ secuencia sec;  int aux;
  while (true)
    {  P(turno[id]);
       finTiempo = false;
       V(inicio);
       while (no termine el tiempo)
         { P(pedidos);
           P(mutex);
           pop(C, (aux, sec));
           V(mutex);
           resultados[aux] = resolver(sec);
           V(espera[aux]);
         }
    }
}
```

¿Cómo sabe cuando hasta cuando iterar?

Process Reloj

```
{ while (true)
    { P(inicio);
      delay(5 hs);
      finTiempo = true;
      V(pedidos);
    }
}
```

EJERCICIO 5

Cuando pasa el *P(pedidos)* es porque el reloj avisó que termino el tiempo (*finTiempo* = *true*) y/o hay pedidos en la cola → en base a eso despierta al otro o atiende pedido.

```
sem mutex = 1,  pedidos = 0,  espera[N] = ([N] 0),  inicio = 0,  turno[2] = (1, 0);
int resultados[N];   cola C;   bool finTiempo = false;

Process Cliente[id: 0..N-1]
{ secuencia S;
  while (true)
    { --generar secuencia S
      P(mutex);  push(C, (id, S));  V(mutex);
      V(pedidos);
      P(espera[id]);
      --ver resultado de resultados[id]
    }
}

Process Reloj
{  while (true)
    { P(inicio); delay(5 hs); finTiempo = true;
      V(pedidos);
    }
}

Process Servidor[id: 0..1]
{ secuencia sec;  int aux;   bool ok;
  while (true)
    {  P(turno[id]);  finTiempo = false;  V(inicio);
       ok = true;
       while (ok)
        { P(pedidos);
          if (finTiempo) { ok = false;
                           V(turno[1-id]);
                         }
          else { P(mutex);  pop(C, (aux, sec));  V(mutex);
                 resultados[aux] = resolver(sec);
                 V(espera[aux]);
               }
        }
    }
}
```

Si termino el tiempo entonces marca la salida del *while* interno y despierta al otro servidor

EJERCICIO 5

```
sem mutex = 1,  pedidos = 0, espera[N] = ([N] 0),  inicio = 0,  turno[2] = (1, 0);
int resultados[N];   cola C;   bool finTiempo = false;
```

Process Cliente[id: 0..N-1]

```
{ secuencia S;
  while (true)
    { --generar secuencia S
      P(mutex);
      push(C, (id, S));
      V(mutex);
      V(pedidos);
      P(espera[id]);
      --ver resultado de resultados[id]
    }
}
```

Process Reloj

```
{  while (true)
    { P(inicio);
      delay(5 hs);
      finTiempo = true;
      V(pedidos);
    }
}
```

Process Servidor[id: 0..1]

```
{ secuencia sec;  int aux;   bool ok;
  while (true)
    {  P(turno[id]);
       finTiempo = false;
       V(inicio);
       ok = true;
       while (ok)
         { P(pedidos);
           if (finTiempo) { ok = false;
                            V(turno[1-id]); }
           else { P(mutex);
                  pop(C, (aux, sec));
                  V(mutex);
                  resultados[aux] = resolver(sec);
                  V(espera[aux]);
                }
         }
    }
}
```