Programación Concurrente – Tres Arroyos

Semáforos

Convenciónes:

- Utilizar S_nombreDelSemaforo;
- En caso de proteger una variable v es aconsejable usar la nomenclatura S_v;
- Arreglos de semáforos: Sem s_arreglo[1..N];

Definición:

Variable con valor entero no negativo.

Tipos de semáforos:

Generales:

Valor del semáforo entre 0 y N.

Binarios:

Valor del semáforo entre 0 y 1.

Operaciones básicas P(S) y V(S):

Generales:

P(S): <await(S>0) S=S-1;>

V(S): <S=S+1;>

Binarios:

P(S): <await(S>0) S=S-1;>

V(S): <await(S<1) S=S+1;>

Exclusión Mutua y Sincronización:

Exclusión Mutua:

Sem S=1;

```
Process P [p:1..N]
        …
        P(S);
         Sección Crítica
        V(S);
        …
End Process P;
```

Sincronización:

P1 debe esperar a P2 para continuar.

```
Sem S=0;
```

```
Process P1
        …
        P(S);
        …
End Process P1;
```

```
Process P2
        …
        V(S);
        …
End Process P2;
```

Inicialización dependiendo del caso:

Es común que los semáforos para exclusión mutua se inicialicen en un valor mayor a cero, mientras que los de sincronización se inicializan en 0.

Comunicación entre procesos utilizando semáforos, modelo Productor - Consumidor:

Es deseable que los procesos se comuniquen, por lo tanto un proceso dejara algún dato en memoria que otro proceso deberá tomar, el modelo a seguir es el de productor – consumidor.

Ejemplo: Existen un conjunto de clientes que son atendidos por un empleado para obtener el estado de un trámite, los cliente interactúan con el empleado de a uno, cada cliente llega, entrega su DNI al empleado, esté de acuerdo al DNI le retorna el estado del trámite.

Este ejercicio puede resolverse de dos formas:

Solución con buffer de capacidad "infinita".

El modelo productor – consumidor en general propone un buffer donde los productores depositan datos que los consumidores retiran.

```
Cola q;
estados = Array [1..N] of String;
Sem S_cola=1;
Sem S_atencion=0;
Sem S_espera[1..N];
Sem S_estados[1..N];
```

```
Process Cliente [c=1..N]
 Tipo_DNI Mi_DNI;
 String miEstado;

  P(S_cola);
   Push(q, [Mi_DNI,c]);
  V(S_cola);
  V(S_atencion);
  P(S_espera[c]);
  P(S_estados[c]);
   miEstado=estados[c];
  V(S_estados[c]);
End Process Cliente;
```

```
Process Empleado
Tipo_DNI  DNI_Cli;
Int c;
String estado;

  While (haya clientes){
        P(S_atencion);
         P(S_cola);
         [DNI_Cli,c]=pop(q);
        V(S_cola);
        estado = Buscar(DNI_Cli);
         P(S_estados[c]);
         estados[c]=estado;
         V(S_estados[c]);
        V(S_espera[c]);
 }
End Process Empleado;
```

Solución con buffer de capacidad 1.

Hay algunas situaciones en que el tamaño del buffer está limitado a 1, por lo tanto la comunicación se da en una variable simple.

```
Tipo_DNI DNI;
Int cliente;
estados = Array [1..N] of String;
Sem S_variable=1;
Sem S_atencion=0;
Sem S_espera[1..N];
Sem S_estados[1..N];
```

```
Process Cliente [c=1..N]
 Tipo_DNI Mi_DNI;
 String miEstado;

 P(S_variable);
 DNI=Mi_DNI
 cliente=c;
 V(S_atencion);
 P(S_espera[c]);
 P(S_estados[c]);
  miEstado=estados[c];
 V(S_estados[c]);
End Process Cliente;
```

```
Process Empleado
String estado;

  While (haya clientes){
        P(S_atencion);
        estado =Buscar(DNI);
         P(S_estados[cliente]);
         estados[cliente]=estado;
         V(S_estados[cliente]);
        V(S_espera[cliente]);
        V(S_variable);
 }
End Process Empleado;
```

En ambos casos se puede ver que hay un semáforo para cada cliente contenido en un arreglo indexado por el identificador del cliente, es muy útil en el caso que se quiera interactuar con un proceso particular.

En el primer caso los clientes envian su DNI al empleado encolándolo, esto permite que en caso que los clientes puedan hacer otra cosa después de encolarse lo puedan hacer.

En el segundo caso los clientes que van llegando deben esperar a que el empleado atienda al cliente que esta siendo atendido antes de poder cargar su DNI en la variable, en este caso la exclusión mutua se da entre dos procesos, el cliente es el que hace el P(S_variable) mientras que el empleado es el que hace el V(S_variable), es por eso que se debe tener cuidado de cuando hacer el V para no liberar las variables antes de que estas hayan terminado de usarse.

Productores o consumidores de cantidad de ítems finita.

Suponer que se tienen P productores que deben producir entre todos N ítems. Existe el compromiso de un productor de producir el ítem i-esimo.

```
Int cantidad=0;
Sem S_Cantidad=1;

Process Productor [p:1..P]
Int cant;
        P(S_Cantidad);
        cant=cantidad;
        While(cant<N){
                cantidad++;
                V(S_Cantidad);
                "Posible uso de cant"
                Produce Item;
                 P(S_Buffer);
                   Push(buffer,Item);
```

```
                V(S_Buffer);
                P(S_Cantidad);
                cant=cantidad;
        }
        V(S_Cantidad);
End Process Productor;
```

Un ejemplo similar es el de C consumidores consumiendo N ítems, en este caso existe el compromiso de un consumidor de consumir el ítem i-esimo, aunque el item no esté listo el consumidor se compromete a consumirlo.

```
Int cantidad=0;
Sem S_Cantidad=1;

Process Consumidor [c:1..C]
Int cant;
        P(S_Cantidad);
        cant=cantidad;
        While(cant<N){
                cantidad++;
                V(S_Cantidad);
                "Posible uso de c"
                P(S_Buffer);
                   Item = pop(buffer);
                 V(S_Buffer);
                Consume Item;
                P(S_Cantidad);
                cant = cantidad;
        }
        V(S_Cantidad);
End Process Consumidor;
```

Notar que se usa una variable local cant para cargar el valor de la variable global cantidad, esto es útil en caso que el valor de la variable cantidad sea necesitado nuevamente, sería deseable que sea el mismo valor que se leyó al principio.

Otro aspecto a tener en cuenta es el compromiso del productor o consumidor con el ítem i-esimo, en programación secuencial es común producir o consumir un ítem y luego incrementar la cantidad, en este caso el incremento se realiza antes de realizar la acción.

Errores comunes: Busy waiting.

```
Sem S_Buffer=1;
```

```
Queue buffer;

Process Productor
        Produce Item;
        P(S_Buffer);
          Push(buffer,Item);
        V(S_Buffer);
End Process Productor;
```

```
Process Consumidor
 Boolean consume=false;
        While (not consume){
            P(S_Buffer);
            If(not empty(buffer)){
                Item = pop(buffer);
                consume=true;
            }
            V(S_Buffer);
        }
End Process Consumidor;
```

Barreras:

Es muy común que varios procesos deban esperarse para continuar su ejecución.

Si fueran solo 2 procesos:

```
semllega1=0, llega2=0;

processWorker1
        …
        V(llega1);
        P(llega2);
        …
End Worker1;
```

```
processWorker2
        …
        V(llega2);
        P(llega1);
        …
End Worker2;
```

Si fueran N procesos:

```
Int contador=0;
Sem S_contador =1;
Sem S_espera =0;

Process P [p:1..N]
        …
        P(S_contador);
        contador++;
        if(contador<N){
                V(S_contador);
                P(S_espera);
```

```
        }else{
                For i=1..N-1{
                        V(S_espera);
                }
                contador=0;
                V(S_contador);
            }
        …
End Process P;
```

Grupos:

En ocasiones los procesos deben agruparse para realizar una tarea en particular.

Ejemplo: Los empleados de una fabrica se agrupan en grupos de a G empleados, cada grupo luego realiza una tarea asignada.

```
Int contador=0;
Int nroGrupo=1;
Sem S_contador =1;

Process P [p:1..N]
Int miGrupo;
        …
        P(S_contador);
        miGrupo=nroGrupo;
        contador++;
        if(contador==G){
                contador=0;
                nroGrupo++;
        }
        V(S_contador);

        "Realiza la tarea correspondiente al grupo miGrupo"
        …
End Process P;
```

Consideraciones de grupos y barreras:

Muchas veces luego de armar un grupo es posible que los procesos deban esperarse, suponiendo el ejemplo anterior los empleados del mismo grupo podrían esperarse para comenzar a realizar la tarea todos juntos.

Es común que se quiera hacer ambas cosas al mismo tiempo, mientras se arma el grupo realizar la barrera, es conveniente no hacerlo de esta forma, sino primero asignar el grupo y

luego la barrera ya que la barrera de un grupo es independiente de la asignación del grupo y podrían interferir no permitiendo que se maximice la concurrencia, además el código en ocasiones es mas complejo.

Relojes:

Ejemplo: Se tiene un banco con una sola caja, los clientes llegan se encolan y esperan ser atendidos, si pasados 15 minutos un cliente no fue atendido se retira.

```
Cola q;
estados = Array [1..N] of String = “”;
Sem S_cola=1;
Sem S_atencion=0;
Sem S_reloj[1..N]=0;
Sem S_esperar[1..N]=0;
Sem S_estado[1..N]=0;

    Process Cliente [c:1..N]
        …
        P(S_cola);
         push(q,c);
        V(S_cola);
        V(S_atencion);
        V(S_reloj[c]);
        V(S_esperar[c]);

        P(S_estado[c]);
        If (estado[c]==”atención”){
         “ser atendido por el empleado”;
        }
        V(S_estado[c]);
        …
    End Process Cliente;


    Process Empleado
    Int cliente;
        …
        While(haya clientes para atender){
            P(S_atencion);
            P(S_cola);
             cliente=pop(q);
            V(S_cola);
            P(S_estado[cliente]);
            If (estado[cliente]==””){
```

```
                    estado[cliente]="atencion";
                    V(S_estado[cliente]);
                    V(S_esperar[cliente]);

                    " atender cliente";
                }else{ V(S_estado[cliente]);}

        }
        …

End Process Empleado;

Process Reloj [r:1..N]
        …
        P(S_reloj[r]);
        P(S_estado[r]);
        If (estado[c]==""){
                estado[c]="irse";
                V(S_esperar[r]);
        }
        V(S_estado[r]);
        …
End Process Reloj;
```

Desventajas de los semaforos:

Sincronización y exclusión mutua mezclada en el código.

Problemas al olvidarse un P o V.

Complejidad al escalar el problema.

Ejercicio:

Se tiene una fábrica de muebles que debe fabricar M muebles, cada mueble tiene 2 partes, existen E1 empleados que fabrican la parte 1 y E2 empleados que fabrican la parte 2, existe además un ensamblador que se encarga de ensamblar una parte 1 y una parte 2 para armar un mueble. Los empleados una vez que no tienen más trabajo dejan de trabajar.