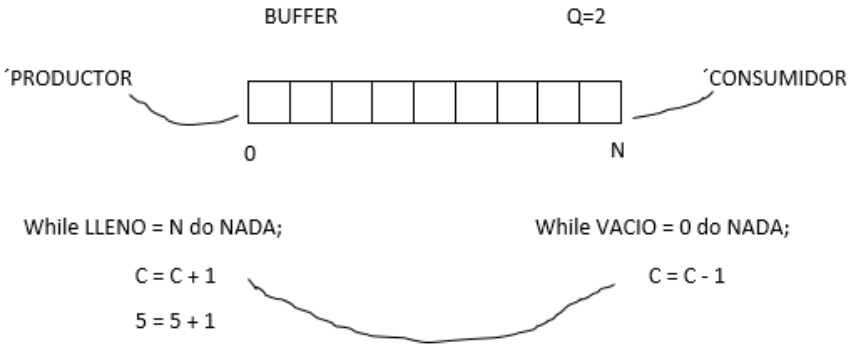
TEMA-3

RESUMEN DE CLASE TEORICA

Nombre | Nombre del curso | Fecha

SINCRONIZACION DE PROCESOS

PRODUCTOR – CONSUMIDOR (Problemas clásicos de sincronización)



QP	T0	Reg = C	Reg = 4	-Lee el valor de C de la MP y la carga al registro
	T1	Reg + 1	Reg = 5	-Reg + 1 pero como se le termina el tpo (Q=2)
QC	T2	Reg = C	Reg = 4	-El consumidor carga e valor de C en el registro
	T3	Reg = 1	Reg = 3	-Reg – 1, consume y se le termina el Q
	T4	C = Reg	Reg = 3	-Se ejecuta el productor, C=Reg guarda elemento
QP	T5	Guarda elemento	Tendría	-En el T5 el productor se ejecuta el Consumidor y hace lo mismo que guardar en 5to lugar para que este bien. C=Reg.

C es el recurso compartido, la misma variable, la misma dirección de memoria, esto quiere decir que el productor y el consumidor no pueden hacer la operación al mismo tiempo sí está en un sistema de tiempo compartido, porque pueden existir interrupciones. Que ocurriría si en el T0 (tiempo cero) comenzamos a ejecutar:

Esto no es correcto, porque la variable C no se actualiza antes de pasar al otro proceso. Lo que provoca que las posiciones se sobreescriban, es decir guarda un elemento en donde ya había otro.

Este es el problema cuando los procesos salen de foma expulsiva, el Kernel puede ser interrumpido en medio de una operación, lo que provoca valores incorrectos. Esto sucede cuando la variable es compartida o sea que en el caso de los hilos los procesos son cooperativos, los procesos PRODUCTOR – CONSUMIDOR son cooperativos, el CONSUMIDOR depende del PRODUCTOR y el PRODUCTOR depende del CONSUMIDOR, si no estan sincronizados no funcionarán correctamente. Para evitar eso, cuando tenemos un recurso compartido entre dos procesos cooperativos o hilos, decimos que esa porción de código se llama SECCION CRITICA.

SECCION CRITICA

La necesidad de la sincronización es porque compartimos recursos entre dos procesos, como por ejemplo el recurso compartido en el ejemplo es la memoria, como los dos procesos entran en estado de competencia, el resultado de cuando se ejecuten va a depender del orden en el que lo hagan. Si tuvieramos un Q = 3 el proceso PRODUCTOR se ejecutaría completo y no habría problema. Lo que tenemos que asegurar es que el estado de competencia no altere el valor de la Sección Crítica.

- ✓ La **SECCION CRITICA**: es la porción de código que comparte recursos entre procesos cooperativos.

PRODUCTOR	CONSUMIDOR
S : = 1	S : = 0
Inicio Sección Crítica	Inicio Sección Crítica
C = C + 1	C = C – 1
Fin Sección Crítica	Fin Sección Crítica
	Sec restante

La **SECCION CRITICA** tiene que cumplir 3 condiciones:

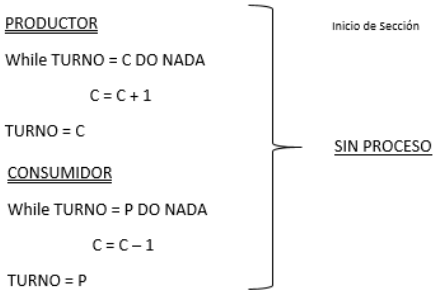
1) **MUTUA EXCLUSIÓN**: un solo proceso puede estar en su Sección Crítica, esto quiere decir que si el Productos esta trabajando con C, el Consumidor no puede trabajar en C. El Proceso que ejecuta su Sección Crítica excluye a los otros procesos.
2) **PROGRESO**: que la selección de quienes entran a la Sección Crítica va a estar dada por aquellos que quieren entrar a la S.C, es decir que solo pueden entrar en la Sección Crítica aquellos que esten compitiendo por entrar a la S.C, ningún proceso que esté en la Sección Restante puede impedir que entre.
   Def: "Solo compiten por entrar a la Sección Crítica aquellos procesos que quieren entrar en esta sección, ningún proceso que este en su Sección Restante puede impedir que otro proceso entre en su Sección Crítica".

3) **ESPERA LIMITADA:** es que todo proceso tiene que tener un **tiempo limitado de espera** para entrar a la Sección Crítica.
   Quiere decir que al inicio de la Sección Crítica hay una cola de procesos, va a ser una cola FIFO para evitar la INANICIÓN, todo proceso que quiera entrar en la Sección Crítica tiene que tener la seguridad de que alguna vez va a entrar.

   De esta forma cuando se tenga Q de tiempo lo mismo se le quitara al proceso, pero ya cuando entre el Consumidor no va a poder hacer las operaciones porque va a quedar en espera en Inicio de Sección Crítica, como no puede entrar se le quita el Q de tiempo y no se ejecuta, queda en estado de espera activa, es decir que directamente entra el Consumidor y lo que va a hacer el en T2 es preguntar por la Sección Crítica, como no puede entrar se le quita el tiempo y pasa a otro proceso, entonces el proceso Productor (que es el que está en su Sección Crítica) va a poder realizar las operaciones correctamente para obtener los valores correctos, porque el Consumidor queda en espera de su Sección Crítica. Entonces se soluciona el problema.

- ✓ Con la **SECCIÓN CRÍTICA** nos aseguramos que haga la suma y guarde el valor en la dirección de memoria, para que tengamos un resultado correcto.
  "Si un proceso entra a Sección Crítica, ningún otro proceso podrá entrar". Cuando termina el proceso de ejecutarse, recién podrá entrar el siguiente proceso de la cola.

DEKKER: hizo los primeros intentos de proteger la Sección Crítia, lo primero que dijo es: utilizamos una variable para decir si esta ocupada o desocupada y la llamó TURNO



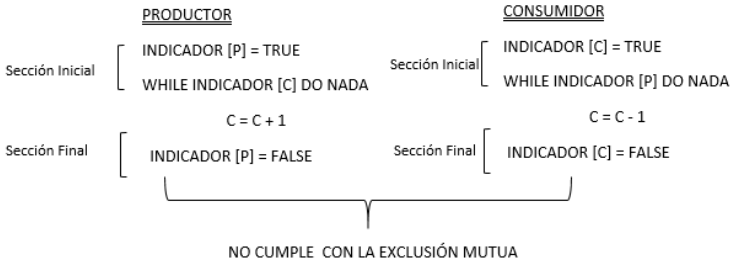
El problema de este caso es que no satisface el progreso, porque se tienen que ejecutar alternativamente porque el Productor es el que de da el TURNO al Consumidor y el Consumidor es el que le da el TURNO al Productor.

¿Qué sucede cuando el Productor produce mucho pero el Consumidor no consume?

¿Cuántas veces entra el Productor? Entra una sola vez, porque cuando se ejecuta el productor va a preguntar si es el TURNO del Consumidor, pero si este es ocioso y no quiere entrar, no podrá impedir que el Productor produzca.

Esta solución no cumple con los tres requisitos.

Entonces, en vez de darle la alternativa de uno y otro, se usa una variable booleana para averiguar donde esta el proceso, se crea la variable llamada INDICADOR.



Los dos se estan excluyendo pero ninguno entra, entonces entramos en un bloque indefinido.

- ✓ Sirve para decir donde estan, pero no sirve para impedir la EXCLUSIÓN MUTUA.

Entonces se combinaron las dos soluciones:

PRODUCTOR

INDICADOR [P] = TRUE

WHILE INDICADOR [C] AND TURNO = CONS DO NADA

(Si se cumplen las dos condiciones, quiere decir que C está en la Sección Crítica.)

C = C + 1

INDICADOR [P] = FALSE

CONSUMIDOR

INDICADOR [C] = TRUE

WHILE INDICADOR [P] AND TURNO = PROD DO NADA

C = C - 1

INDICADOR [C] = FALSE

- Si el INDICADOR [C] es verdadero significa que C esta en la Sección Crítica.
- TURNO esta en C, significa que le toca ejecutar a C.

O sea estamos seguros que C está en la Sección Crítica, esto lo hace el INDICADOR.

SOLUCIÓN DE PETERSON: Solución para N

Utiliza un array booleano para todos los procesos y un turno de array numérico para el número de identificador de proceso.

Este algoritmo se conoce como el ALGORITMO DE LA PANADERIA, porque lo que dice es, que todo proceso que quiere entrar se le da un valor booleano para avisar que entra y un número que es la posición en la cola.

El problema que tiene cualquiera de estos procesos es que son procesos de ESPERA ACTIVA, porque para esperar tenemos que hacer algo, es decir que el proceso queda esperando sin poder hacer nada en el estado bloqueado, ocupando recursos sin hacer nada positivo, cuando en realidad el proceso esta en un estado de competencia, eso quiere decir que no va a la cola de BLOQUEADOS sino que va a la cola de LISTOS por lo que sigue dando vuelta tratando de competir por algo que no puede alcanzar.

DIJKSTRA

Dijo porque no utilizamos instrucciones atómicas con ayuda del hardware.

- INSTRUCCIÓN ATOMICA: es aquella instrucción que no puede ser interrumpida mientras se este ejecutando.

El S.O consta de dos Instrucciones Atómicas, que son cerradura o cerrojo e insertando.

  - **TAS**: Testear, Asignar y Settear
  - **TSL**: Tester, Settear y Bloquear

    Tanto TSL y TAS son instrucciones del Kernel

SEMAFOROS

"Es una variable del tipo entero a la cual solamente se puede acceder a traves de dos operaciones atómicas: ESPERA – WAIT(S) y SEÑAL – SIGNAL(S)."

¿Qué operaciones podemos hacer?

Solamente podemos crear un Semáforo, asignarle un valor y dos operaciones atómicas.

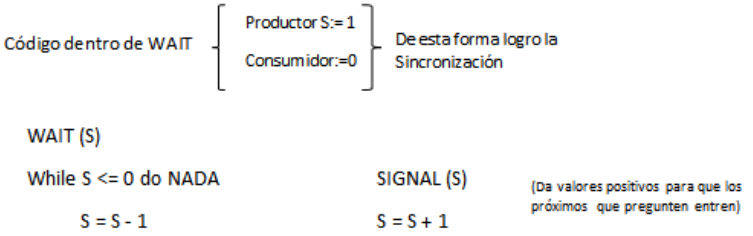
- **WAIT:** como la palabra lo dice, el proceso espera de acuerdo al valor de un semáforo.
- **SIGNAL**: cuando el proceso sale de la SC, emite una señal y modifica el valor de un semáforo y si hubiera desbloquea un proceso.

WAIT (S)

S.C

SIGNAL (S)

En vez de tener un código, utilizamos estas dos Instrucciónes: WAIT es para saber si puedo entrar a la Sección Crítica y SIGNAL es para avisar que salí de la Sección Crítica.

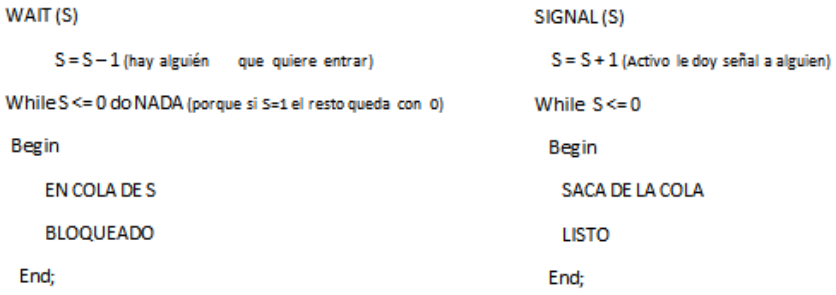


Es importante que se ejecute primero el productor porque sino el consumidor no va a hacer nada.

Este es un SEMAFORO DE ESPERA ACTIVA: es decir que sigue usando ciclos de CPU sin hacer nada.

Entonces se decide encolar a los procesos que quieren entrar, y creó los SEMAFOROS DE TIPO CONTADOR ("S": indica la cantidad de procesos que pueden entrar en la S.C)

SEMAFOROS DE TIPO CONTADOR (es un método de Sincronización, para proteger la S.C)



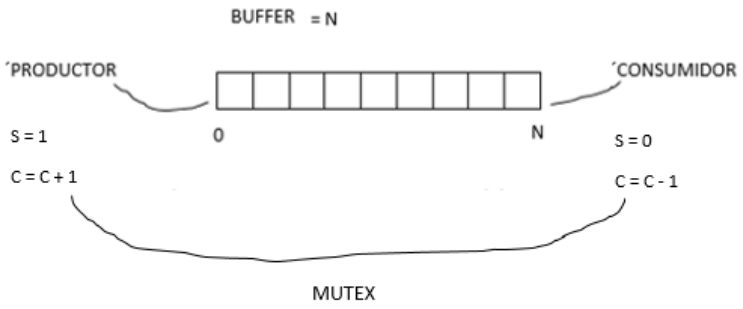
El valor negativo de S es la cantidad de procesos que están en la cola de S (bloqueado).

SEMAFORO BINARIO

Es un semáforo que solo puede tener solo dos valores 0(cero) o 1(uno). Que sería un semáforo cerradura, hoy en día se lo conoce como MUTEX.

PROBLEMAS CLASICOS DE SINCRONIZACION

1. **PROBLEMA PRINCIPAL PRODUCTOR – CONSUMIDOR** (o Problema del Buffer Limitado)



(Porque vale para el Productor o vale para el Consumidor).

También necesitamos un Semáforo para el buffer: (Área de Memoria)

BUFFER (N = LLENO, O = VACIO)

- ✓ Mínimamente necesitamos dos semáforos, una para el área de memoria (BUFFER) y otro un contador que nos permite manejar la longitud del Array.
- ✓ En un proceso se va a tener tantos semáforos como recursos compartan o Secciones Críticas se tengan.
- ✓ Se da en todos los casos, por ejemplo: cuando tenemos que enviar caracteres del teclado a un proceso, el teclado produce caracteres y eso lo tiene que consumir el proceso que solicito esos caracteres.

2. **LECTOR O ESCRITOR**

Cuando se tiene una variable compartida o un archivo compartido, la Sección Crítica está dada cuando alguien quiere modificar ese dato o modificar el archivo.

Necesitamos dos Semáforos, por un lado, para contar la cantidad de lectores que hay en el Sistema hasta que llegue el escritor. Una vez que llega el escritor tenemos que cerrar el ingreso de lectores.

```
ESCRITOR = 0
LECTORES
  LECTOR = LECTOR + 1
  SI HAY LECTOR
     ESCRITOR WAIT (ESCRITOR)
     MUTEX (LECTOR)
   LECTOR = LECTOR - 1
END
```

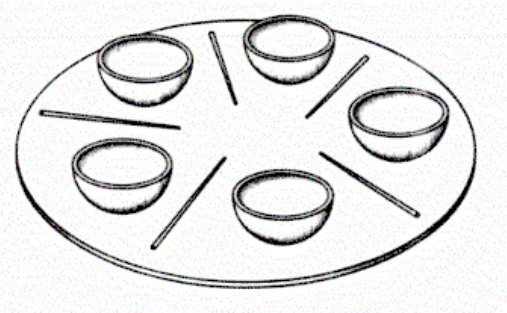
```
ESCRITOR (Semáforo mutex)
IF LECTORES = 0 then

   SIGNAL (ESCRITOR)
RETURN
```

- Qué los Lectores no entres si hay un escritor (necesitamos un semáforo ahí) MUTEX.
- En el sistema se puede tener muchos Lectores, pero solo un Escritor a la vez.

3. FILOSOFOS COMENSALES

El problema de los Filósofos Comensales es que, tenemos una mesa donde solo pueden estar a la vez 4 Filósofos, y sus estados pueden ser: tienen hambre (INICIO Sección Crítica) o están comiendo (están en la Sección Crítica) o están pensando (Sección Restante).



(Mientras comen los PARES, los IMPARES no comen).

*CONDICION: El Filósofo que quiere comer pregunte primero por el palillo de la derecha, si está libre lo tome y pregunte por el palillo de izquierda, y si está libre lo tome y coma.

Ahora que ocurre, si el palillo de la izquierda está ocupado deberá liberar el de la derecha, porque quiere decir que no puede comer. Si pregunta por el palillo de la derecha y no lo puede obtener no hay problema, sigue al inicio de la Sección Crítica sino si toma el palillo de la derecha (WAIT) entra a la Sección Crítica, va a preguntar WAIT (i + 1) palillo de la izq, si no lo puede tomar, tiene que liberar al otro, o sea no puede agarrar un recurso y retenerlo.

4. BARBERO DORMILON (Caso práctico del Spool)

Se trata de una barbería donde hay un solo sillón donde duerme el barbero y tenemos una LISTA para los clientes. Entonces tenemos un MUTEX, los clientes pueden entrar mientras el BUFFER está libre, cuando el BUFFER esté lleno se cierra la puerta y no entran más clientes y en el sillón de la barbería solo va a entrar uno, o sea que tenemos otra variable MUTEX, cuando entra el primero tiene que hacer un SIGNAL al barbero para que se despierte y lo atienda.

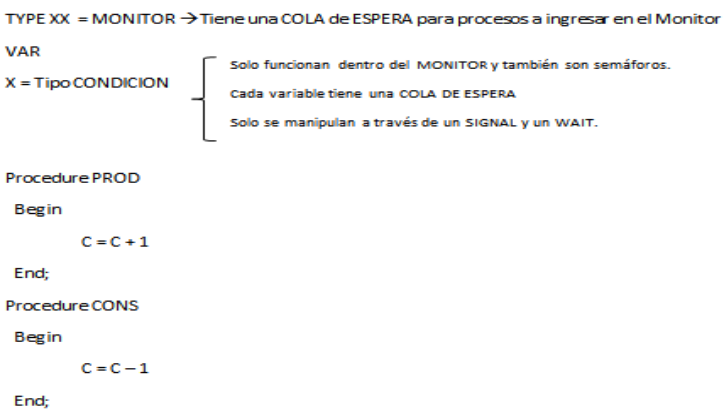
MONITOR

El problema de trabajar con Semáforos es que si no se usan bien las dos instrucciones atómicas (WAIT y SIGNAL) podemos producir que los procesos que quedan dormidos en el semáforo sufran de inanición porque nadie le da señal, nadie incrementa sino tienen un valor positivo, no van a entrar a su Sección Crítica.

Entonces los lenguajes de programación de alto nivel crearon una estructura lingüística llamada MONITOR.

MONITOR: es una construcción lingüística (construcción de un lenguaje de alto nivel) que lo interpreta el compilador y cuando este traduce el lenguaje a un programa ejecutable maneja los WAIT y SIGNAL.

Es decir, que en vez que el programa haga WAIT y SIGNAL, solo declara procedimientos con todas las Secciones Críticas que tenga. Primero se declara un MONITOR:



- Cuando nosotros programemos lo que vamos a hacer, es llamar al procedimiento XX PROD que quiere decir que está dentro del MONITOR cuando se lo compile, lo que el compilador hará es poner WAIT el semáforo que sea, él mismo va a crear semáforo y va a hacer SIGNAL semáforo, o sea él va a traducir como corresponde manejando el semáforo como corresponde, nosotros lo único que hacemos dentro de esta construcción lingüística, es declarar todo.
- Puede ocurrir que nosotros necesitemos sincronizar procesos cooperativos, por ejemplo, en una construcción Lingüística LECTOR – ESCRITOR, que el Escritor no puede escribir si hay Lectores, entonces vamos a tener una condición dentro del BEGIN y END (procedimiento correspondiente). En estos casos podemos declarar las variables de TIPO CONDICION, estas variables solo funcionan dentro del monitor como las Variables Locales solo existen dentro de la estructura de monitor.
- En el MONITOR solo un proceso puede estar activo, es decir solo un procedimiento se puede estar ejecutando. Esto quiere decir que se excluyen mutuamente los procedimientos.
- Tiene prioridad LA COLA de la variable del TIPO CONDICION, porque ya está dentro del semáforo.

LINUX →LOG: es un registro transaccional, es decir que cuando se mandan comandos al S.O, esos comandos con el dato que se va a analizar se guardan en LOG.

PUNTOS DE VERIFICACION: si se descubre que hay un error antes del punto de verificación se puede hacer un ROLL – BACK, es decir lo que se hizo, para que las bases de datos no queden en forma inconsistente.

WINDOWS →Regedit

"LOG": es ir guardando las transacciones y tener puntos de verificación"