**SINCRONIZACION DE PROCESOS**

A raíz de la multiprogramación, han surgido problema en lo que respecta a utilización de recursos, compartidos por varios procesos. Para ello han elaborado varias soluciones factibles para poder establecer una sincronización de estos sin que traiga aparejados inconvenientes, aunque estos puedan suceder. En esta unidad se analizarán todos estos aspectos y se vera implementado en la parte práctica como se debería resolver o no está sincronización.

**EL PROBLEMA**

|  |  |
| --- | --- |
| **PROCESO A** | **PROCESO B** |
| X=1 | X=10 |
| X=X+1 | X=X\*2 |
| PRINT X | PRINT X |
| RESULTADO OBTENIDO: | |

En este caso en el ejemplo el **RESULTADO OBTENIDO**, va a estar sujeto a el orden que se le dé a la ejecución de cada proceso.

Tenemos varios elementos que están involucrados en este dilema, **RECURSO CRITICO, REGION CRITICA Y SECCION CRITICA**.

* **SECCION CRITICA**: Es aquella que involucra el recurso critico en cuestión, desde que se toma hasta que se libera *(puedo tomar el recurso, pero utilizarlo más tarde)*.
* **REGION CRITICA**: Es el código en cuestión, donde involucra estos dos factores *(SECCION Y RECURSO*).
* **RECURSO CRITICO**: Es el recurso compartido por varios procesos, define la instancia de cuando se utiliza.

**SEMAFOROS**

Los SEMAFOROS son herramienta de sincronización que no requiere espera activa.

Se generan a través de una *SYSCALL* y son pares **ATOMICOS** o instrucciones indivisibles, que no se pueden interrumpir una vez que se ejecutan.

Estos pares atómicos pueden ser:

* P() y V().
* DOWN() y UP().
* WAIT() y SIGNAL().

Las primeras de cada tipo decrementan *(ósea pide el recurso, intentando apropiarse*) y las segundas incrementan liberando el recurso.

Al crearse un semáforo, este nace con un valor entero el cual indica la cantidad de ese tipo de recurso.

Estados del SEMAFORO:

* **POSITIVO**: Disponible, el valor define la cantidad de ese tipo.
* **CERO**: Ocupado.
* **NEGATIVO**: Ocupado y el valor absoluto representa la cantidad de procesos aguardando ese recurso.

**ALGORITMO**

|  |  |
| --- | --- |
| **P**(SEMAFORO)  {  SEMAFORO - -  SI (SEMAFORO < 0)  { DUERMO  ENCOLAR  }  } | **V**(SEMAFORO)  {  SEMAFORO + +  SI (SEMAFORO <=0)  { DESPIERTO  DESENCOLO  }  } |

|  |  |  |
| --- | --- | --- |
|  | *CODIGO* |  |
| REGION CRITICA | P(SEMAFORO A) | SECCION CRITICA |
| CODIGO  A++ (RECURSO CRITICO)  CODIGO |
| V(SEMAFORO A) |
|  | *CODIGO* |  |

*“Se define entonces que un proceso es* ***COLABORATIVO*** *cuando su* ***REGION CRITICA*** *y* ***SECCION CRITICA*** *son casi iguales, cuanto más alejadas estén* ***MENOS COLABORATIVO*** *será ese proceso con el resto”*.

**RESOLUCION DE EJERCICIO**

En los casos que analizaremos son un instante o snapshot (fotografía) de un momento determinado del ambiente. Donde daremos las condiciones actuales de todos los semáforos, como así también la sentencia que ejecuta cada uno de los procesos.

|  |  |  |
| --- | --- | --- |
| **PROCESO A** | **PROCESO B** | **PROCESO C** |
| P(X) | P(Y) | P(Z) |
| V(Y) | V(Y) | V(X) |
| V(X) |  | V(Z) |
| SITUACION INICIAL SEMAFOROS X,Y=0 Z=1 | | |

Estableceremos una secuencia de análisis en donde ejecutaremos (PROCESO A, PROCESO B y PROCESO C)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PROCESO** | **SENTENCIA** | **SEMAFORO** | | | **ESTADO** |
| **X** | **Y** | **Z** |  | |
|  |  | 0 | 0 | 1 | VALOR INICIAL | |

Se establece el valor inicial del ejercicio.

Luego procederemos a ejecutar cada proceso en el orden establecido. Recuerden que estaremos trabajando con el algoritmo de la página 2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PROCESO** | **SENTENCIA** | **SEMAFORO** | | | **ESTADO** |
| **X** | **Y** | **Z** |  | |
|  |  | 0 | 0 | 1 | VALOR INICIAL | |
| A | P(X) | -1 | 0 | 1 | WAIT (X) | |

Cuando ejecutamos la sentencia de P() recuerden que decrementamos en 1 el semáforo en cuestión y analizamos el SI, en este caso el valor del semáforo queda negativo indicando que ese recurso esta **OCUPADO** y posee un proceso esperando ese recurso, como vemos en nuestro ejemplo que el proceso **A** esta aguardando que se libere el recurso **X**, por ende este es encolado en **WAIT**. Cuando una sentencia deja a un proceso en estado **WAIT** o (Duerme y Encola), este proceso abandona la **CPU** y deja de procesar, continuando con el siguiente proceso en cuestión (B).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PROCESO** | **SENTENCIA** | **SEMAFORO** | | | **ESTADO** |
| **X** | **Y** | **Z** |
|  |  | 0 | 0 | 1 | VALOR INICIAL |
| A | P(X) | -1 | 0 | 1 | WAIT (X) |
| B | P(Y) | -1 | -1 | 1 | WAIT (Y) |
| C | P(Z) | -1 | -1 | 0 | OK |
|  | V(X) | 0 | -1 | 0 | OK 🡪 DESPIERTA AL PROCESO “A”, VOLVIENDO A LA READY QUEUE |
|  | V(Z) | 0 | -1 | 1 | OK – “C” END – FINALIZA |
| A | V(Y) | 0 | 0 | 1 | OK 🡪 DESPIERTA AL PROCESO “B”, VOLVIENDO A LA READY QUEUE |
|  | V(X) | 1 | 0 | 1 | OK – “A” END – FINALIZA |
| B | V(Y) | 1 | 1 | 1 | OK – “B” END – FINALIZA |

Si analizamos este ejercicio podemos decir que todos los procesos terminan con éxito y su traza de ejecución es A🡪 B🡪C🡪A🡪B.

Existen otras posibilidades de ejecución, si!!!! Veamos un caso más, plantearemos la secuencia C,A,B.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PROCESO** | **SENTENCIA** | **SEMAFORO** | | | **ESTADO** |
| **X** | **Y** | **Z** |
|  |  | 0 | 0 | 1 | VALOR INICIAL |
| C | P(Z) | 0 | 0 | 0 | OK |
|  | V(X) | 1 | 0 | 0 | OK |
|  | V(Z) | 1 | 0 | 1 | OK – END |
| A | P(X) | 0 | 0 | 1 | OK |
|  | V(Y) | 0 | 1 | 1 | OK |
|  | V(X) | 1 | 1 | 1 | OK – END |
| B | P(Y) | 1 | 0 | 1 | OK |
|  | V(Y) | 1 | 1 | 1 | OK – END |

Como vemos en este ejemplo la ejecución es mucho mas limpia y sin interrupciones. La traza seria C🡪A🡪B.

**Que condiciones se pueden dar en este tipo de ejercicios:**

* Que todos los procesos finalicen con éxito.
* Que algunos finalicen y otros no, pudiendo quedar en las siguientes situaciones:
  + **STARVATION o INHANICION**: Esta condición se da cuando un proceso debe esperar un recurso por un tiempo excesivo, y queda en WAIT.
  + **DEADLOCK o ABRAZO MORTAL**: Esta condición se da cuando dos procesos producen un entre cruzamiento de recursos, EJ: Si tenemos dos procesos A y B y a estos posee un R1 y R2 respectivamente. Esto se da porque el PA quiere el R2 que lo tiene PB y PB quiere el R1 que lo tiene el PA y ninguno de estos los libera, entonces se produce el DEADLOCK. Se necesitan ciertas condiciones:
    - **MUTUA EXCLUSION**: Existencia de al menos un recurso compartido por los procesos, al cual solo puede acceder uno simultáneamente.
    - **RETENCION Y ESPERA**: Al menos un proceso PA ha adquirido un recurso R1, y lo retiene mientras espera al menos un recurso R2 que ya ha sido asignado a otro proceso.
    - **ESPERA CIRCULAR**: Dado el conjunto de procesos PA...PZ (subconjunto del total de procesos original), PA está esperando un recurso adquirido por PB, que está esperando un recurso adquirido por PC,... ,que está esperando un recurso adquirido por PD, que está esperando un recurso adquirido por PE. Esta condición implica también la condición de ***RETENCION Y ESPERA***.
    - **NO OCURRE EXPROPIACION**: Los recursos no pueden ser expropiados por los procesos, es decir, los recursos solo podrán ser liberados voluntariamente por sus propietarios (el sistema operativo no puede quitarle un recurso al proceso).

**FORMAS GRAFICAS**

|  |  |
| --- | --- |
| PROCESO  RECURSO | PROCESO  RECURSO  CUANDO LA FLECHA ES SALIENTE DE UN RECURSO HACIA UN PROCESO, INDICA QUE EL PROCESO POSEE ESTE RECURSO “**ASIGANCION**” |
| PROCESO  RECURSO  CUANDO LA FLECHA ES SALIENTE DE UN PROCESO HACIA UN RECURSO, INDICA QUE EL PROCESO ESTA SOLICITANDO ESTE RECURSO “**SOLICITUD**” *(si al finalizar un ejercicio de semáforos algún proceso queda en esta condición, podemos decir que se encuentra en STARVATION)* |  |
| ***ABRAZO MORTAL FORMA GRAFICA***  RECURSO 1  PROCESO A  PROCESO B  RECURSO 2 | |

**ABRAZO MORTAL – ESTRATEGIAS PARA AVITARLO *(Esto no quiere decir que no suceda)***

* ***No hacer nada***
* ***Prevención***
  + Prevenir alguna de las cuatro condiciones.
  + Es una decisión de diseño.
* ***Evasión***
  + Cada vez que un proceso pide un recurso, se analiza el estado futuro del sistema.
  + Algoritmo del banquero.
* ***Detección***
  + En intervalos regulares de tiempo se buscan deadlocks.
  + Si los hubiera, se los trata.

**ALGORITMO DEL BANQUERO**

Los procesos al ingresar al sistema declaran la cantidad máxima de cada uno de los recursos que podrá requerir.

Cuando un proceso solicita un conjunto de recursos, el sistema debe deter­minar si la asignación de dichos recursos dejará al sistema en un estado seguro. En caso afirmati­vo, los recursos se asignarán; en caso contrario, el proceso tendrá que esperar hasta que los otros procesos liberen los suficientes recursos.

Estructuras necesarias:

* Matriz de peticiones máximas (MxN)
* Matriz de recursos asignados (MxN)
* Matriz de necesidad (MxN)
* Vector de recursos totales (Mx1)
* Vector de recursos disponibles actuales (Mx1)

Ante cada solicitud se debe simular la asignación, actualizando las estructuras adecuadas, y luego analizar si existe una secuencia segura.

**ARITMETICA DEL BANQUERO**

**MATRIZ DE MAXIMOS – MATRIZ DE ASIGNACION = MATRIZ DE NECESIDAD**

**MATRIZ DE MAXIMOS = MATRIZ DE ASIGNACION + MATRIZ DE NECESIDAD**

**MATRIZ DE ASIGNACION = MATRIZ DE MAXIMOS – MATRIZ DE NECESIDAD**

**DISP. TOTAL = DISPONIBLE ACTUAL + COLUMNAS DE MATRIZ DE ASIGNACION *(correspondiente a cada recurso)***

***Ejemplo matrices:***

En el presente ejercicio contamos con las siguientes matrices (***MAXIMAS, ASIGNADOS, DISPONIBLE TOTAL***). Se deberá averiguar las matrices de (***TODAVIA NECESITA y DISPONIBLE ACTUAL***)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **PETICIONES MAXIMAS** | | | |  | **RECURSOS ASIGNADOS** | | | |  | **DISPONIBLE TOTAL** | | |
|  | **R1** | **R2** | **R3** |  |  | **R1** | **R2** | **R3** |  | **R1** | **R2** | **R3** |
| **P1** | 2 | 0 | 1 |  | **P1** | 1 | 0 | 0 |  | 2 | 3 | 2 |
| **P2** | 1 | 3 | 1 |  | **P2** | 0 | 1 | 1 |  |  |  |  |
| **P3** | 1 | 1 | 1 |  | **P3** | 1 | 0 | 0 |  |  |  |  |
|  |  |  |  | ASIGNADO | | 2 | 1 | 1 |  |  |  |  |
| **TODAVIA NECESITA** | | | |  |  |  |  |  |  | **DISPONIBLE ACTUAL** | | |
|  | **R1** | **R2** | **R3** |  | DISP. TOTAL – ASIGNADO = | | | | | **R1** | **R2** | **R3** |
| **P1** | 1 | 0 | 1 |  |  |  |  |  |  | 0 | 2 | 1 |
| **P2** | 1 | 2 | 0 |  |  |  |  |  |  |  |  |  |
| **P3** | 0 | 1 | 1 |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

Como el algoritmo del banquero va analizando cada una de las peticiones de los procesos en cuestión, bloqueara aquellos que pongan en riesgo al sistema (*ya que al no poder satisfacer su petición*), siempre tratara de evitar la **INSEGURIDAD** del sistema y dará prioridad a los procesos a los cuales puede satisfacer su demanda de **TODAVIA NECESITA** con el objetivo de que finalicen y así puedan devolver al sistema el **MAXIMO** de recursos solicitado desde el comienzo.

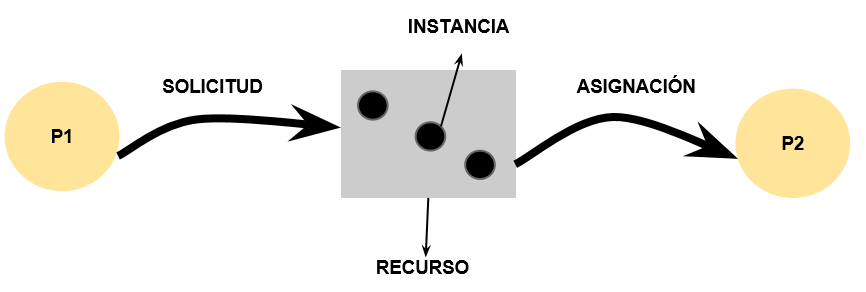
Entonces analicemos la asignación que realizaría el sistema *(basándonos en la matriz de* ***TODAVIA NECESITA****),* podría así asignar la petición P1(1,0,1) ?? NO, ya que nosotros contamos con un **DISP. ACTUAL** de (0,2,1), cosa que pondría en riesgo el sistema, entonces este lo que hace manda a **WAIT** **o BLOQUEADO** al proceso P1. De la misma manera hace lo mismo con el P2 ya que su **NECESIDAD** es de (1,2,0).

Si vemos el P3 su **NECESIDAD** es de (0,1,1) cosa que con nuestro **DISP.ACTUAL** (0,2,1) podríamos cubrir su necesidad. Desarrollemos el ejercicio.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ACCIONES** | **R1** | **R2** | **R3** |  |
| DISP.ACTUAL | 0 | 2 | 1 | INICIAL |
| P3 - TN | 0 | 1 | 1 |  |
| DISP.ACTUAL | 0 | 1 | 0 | SALE DE LA RESTA DE D.A – P3 TN |
| P3 - FIN | 1 | 1 | 1 |  |
| DISP.ACTUAL | 1 | 2 | 1 | SALE DE SUMAR EL MAX LIBERADO POR P3 + D.A |
| P2 - TN | 1 | 2 | 0 | AHORA PUEDO DESPERTAR AL P2 BLOQUEADO |
| DISP.ACTUAL | 0 | 0 | 1 | SALE DE LA RESTA DE D.A – P2 TN |
| P2 - FIN | 1 | 3 | 1 |  |
| DISP.ACTUAL | 1 | 3 | 2 | SALE DE SUMAR EL MAX LIBERADO POR P2 + D.A |
| P1 - TN | 1 | 0 | 1 | AHORA PUEDO DESPERTAR AL P1 BLOQUEADO |
| DISP.ACTUAL | 0 | 3 | 1 | SALE DE LA RESTA DE D.A – P1 TN |
| P1 – FIN | 2 | 0 | 1 |  |
| DISP.ACTUAL | 2 | 3 | 2 | SALE DE SUMAR EL MAX LIBERADO POR P1 + D.A  EL SISTEMA **ES SEGURO** Y EL DISP DEBE SER = AL TOTAL |

***Ejemplo gráfico:***

Para el caso de los ejercidos en modo grafico el armado de las matrices sería el siguiente, teniendo en cuenta que:



Ejercicio:

R3 O

R1 O

P3

P2

P1

R2 OO

Con el modo grafico se pueden armar todas las matrices sin necesidad de realizar ninguna resta, suma, etc. Entonces como lo hacemos o que hay que tener en cuenta a la hora de la construcción de esta. La dimensión de las matrices estará dada en función de la cantidad de Procesos (M) y la cantidad de Recursos (N).

* ***MATRIZ DE MAXIMOS***: Serán todas las flechas entrantes y salientes del proceso, con referencia a cada recurso.

|  |  |  |  |
| --- | --- | --- | --- |
| **MATRIZ DE MAXIMOS** | | | |
|  | **R1** | **R2** | **R3** |
| **P1** | 1 | 1 | 0 |
| **P2** | 1 | 1 | 1 |
| **P3** | 0 | 0 | 1 |

* **MATRIZ DE ASIGNACION**: Serán todas las flechas salientes de un recurso hacia un proceso.

|  |  |  |  |
| --- | --- | --- | --- |
| **MATRIZ DE ASIGNACION** | | | |
|  | **R1** | **R2** | **R3** |
| **P1** | 0 | 1 | 0 |
| **P2** | 1 | 1 | 0 |
| **P3** | 0 | 0 | 1 |

* **MATRIZ DE TODAVIA NECESITA**: Serán todas las flechas salientes de un proceso hacia un recurso.

|  |  |  |  |
| --- | --- | --- | --- |
| **MATRIZ DE NECESIDAD** | | | |
|  | **R1** | **R2** | **R3** |
| **P1** | 1 | 0 | 0 |
| **P2** | 0 | 0 | 1 |
| **P3** | 0 | 0 | 0 |

* **DISPONIBLE TOTAL**: Surge de la cantidad de puntos “Instancias” que posee cada recurso.

|  |  |  |
| --- | --- | --- |
| **DISPONIBLE TOTAL** | | |
| **R1** | **R2** | **R3** |
| 1 | 2 | 1 |

* **DISPONIBLE ACTUAL**: Surge de resta la cantidad de flechas salientes de un recurso a su cantidad de instancias.

|  |  |  |
| --- | --- | --- |
| **DISPONIBLE ACTUAL** | | |
| **R1** | **R2** | **R3** |
| 0 | 0 | 0 |

Si observamos a simple vista este ejercicio teniendo ese Disp. Actual parecería que el sistema seria ***INSEGURO***. Pero hay que analizar la necesidad de los procesos en cuestión. Vemos que el P3 ya cumplió su cometido y estaría en condiciones de liberar su ***MAXIMO***. Entonces desarrollemos el ejercicio.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ACCIONES** | **R1** | **R2** | **R3** |  |
| DISP.ACTUAL | 0 | 0 | 0 | INICIAL |
| P3 - FIN | 0 | 0 | 1 |  |
| DISP.ACTUAL | 0 | 0 | 1 | SALE DE SUMAR EL MAX LIBERADO POR P3 + D.A |
| P2 - TN | 0 | 0 | 1 | PUEDO TOMAR EL P2 YA QUE HE BLOQUEADO AL P1 |
| DISP.ACTUAL | 0 | 0 | 0 | SALE DE LA RESTA DE D.A – P2 TN |
| P2 - FIN | 1 | 1 | 1 |  |
| DISP.ACTUAL | 1 | 1 | 1 | SALE DE SUMAR EL MAX LIBERADO POR P2 + D.A |
| P1 - TN | 1 | 0 | 0 | PUEDO DESPERTAR AL P1 |
| DISP.ACTUAL | 0 | 1 | 1 | SALE DE LA RESTA DE D.A – P1 TN |
| P1 - FIN | 1 | 1 | 0 |  |
| DISP.ACTUAL | 1 | 2 | 1 | SALE DE SUMAR EL MAX LIBERADO POR P1 + D.A  EL SISTEMA **ES SEGURO** Y EL DISP DEBE SER = AL TOTAL |