PROGRAMACION CONCURRENTE

Sincronización basada en memoria compartida: Monitores

Introducción

En este tema nos centramos en la gestión de la comunicación y la sincronización en memoria compartida mediante monitores.

- Un programa concurrente *necesita* utilizar estructuras globales para que los procesos puedan comunicarse.
  - Simplifica la la comunicación entre dos procesos, esto ocurre cuando uno de los dos procesos escribe sobre una estructura global y el otro lee la misma.
  - La comunicación se realiza mediante variables compartidas
  - Para que la comunicación se realice de forma correcta, se necesita introducir n mecanismo de sincronización.
- Existen dos tipos de sincronización básicos:
  - *exclusión mutua*
    - Las operaciones no son *atómicas*
  - condiciones de sincronización
    - que establecen un orden preciso en la ejecución de las acciones.

Inconveniente de los semáforos

- Los semáforos tienen algunas debilidades:
  - La omisión de una primitiva puede corromper la operación de un sistema concurrente.
  - El control de la concurrencia es responsabilidad del programador.
  - Las primitivas de control se encuentran esparcidas por todo el sistema.
- Estas debilidades nos pueden llevar a errores fácilmente
  - Errores transitorios
  - La ejecución de cada sección critica debe comenzar con wait() y terminar con signal() sobre un semáforo
  - No se puede restringir el tipo de operacion sobre el recurso
  - Se debe incluir todas las sentencias criticas en la sección critica
  - Tanto la sección critica como la sincronización se implementan usando las mismas primitivas y luego son difíciles de identificar
  - Los programas que usan semáforos son difíciles de mantener,
  - El código de sincronización esta repartido entre diferentes procesos

Monitor

- La noción de monitor está muy influenciada por la programación modular.
  - "Encapsula recursos compartido"
  - Cuando los recursos no pueden ser arbitrariamente utilizado por los procesos del sistema.
    - Cualquier proceso que quiera utilizar el recurso tiene que pedir permiso al monitor
    - Cuando ya no necesite el recurso, debe devolver el permiso
  - La petición y devolución se implementan a través de los procedimientos de acceso exportados por el monitor
  - El monitor contiene las variables que representan el estado del recurso y las operaciones de acceso al mismo como si fuera un tipo abstracto de datos.
  - La diferencia fundamental con los tipos es que, por dentición, cualquier instrucción del monitor se ejecuta en exclusión mutua.
  - Se usan variables de tipo condición para implementar las condiciones de sincronización

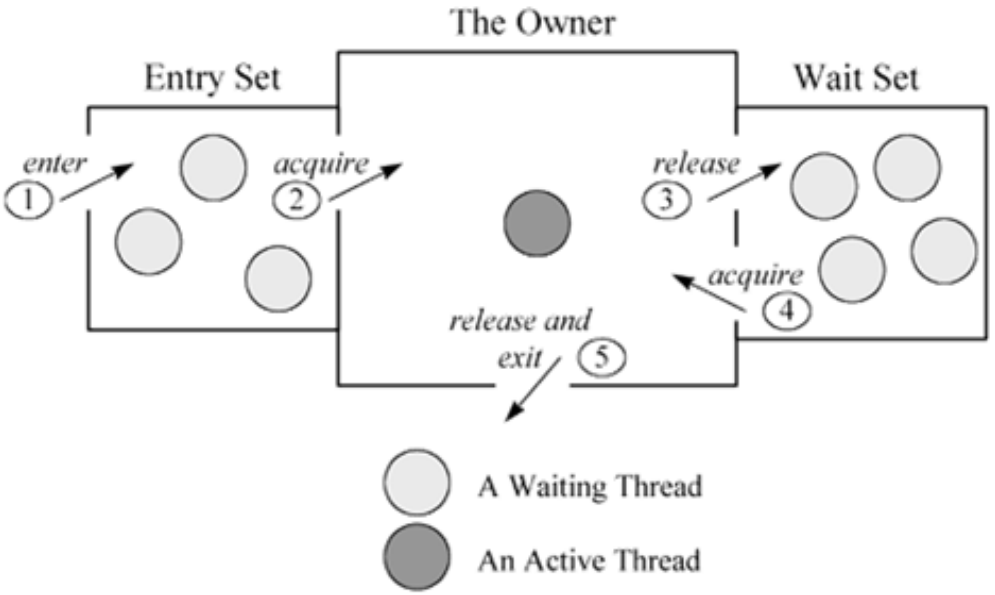
Monitor

- Son módulos que encierran los recursos o variables compartidas como componentes internos privados y ofrece una interfaz de acceso a ellos que garantiza el régimen de exclusión mutua.
- La declaración de un monitor incluye:
  - Declaración de las constantes, variables, procedimientos y funciones que son **privados** del monitor (solo el monitor tiene visibilidad sobre ellos).
  - Declaración de los procedimientos y funciones que el monitor **exporta** y que constituyen la interfaz a través de las que los procesos acceden al monitor.
  - **Cuerpo del monitor**, constituido por un bloque de código que se ejecuta al ser instanciado o inicializado el monitor. Su finalidad es inicializar las variables y estructuras internas del monitor.
- El monitor garantiza el acceso al código interno en régimen de exclusión mutua. El monitor tiene asociada una lista en la que se incluyen los procesos que al tratar de acceder al monitor son suspendidos.

Monitor

- Un monitor es como un edificio que contiene una habitación especial que puede ser ocupada por un solo hilo a la vez. La habitación por lo general contiene algunos datos.
- Desde el momento que un hilo entra en este espacio hasta el momento en que sale, tiene acceso exclusivo a todos los datos en la habitación.

- Entrar en el edificio del monitor se llama "entrar en el monitor."
- Al entrar en la sala especial en el interior del edificio se llama "adquirir el monitor" (aquire).
- Ocupando la habitación se llama "poseer el monitor,“ (own) y salir de la habitación que se llama " liberación del monitor“ (release).
- Dejando todo el edificio "salir del monitor“ (exit)



Sintaxis del monitor.

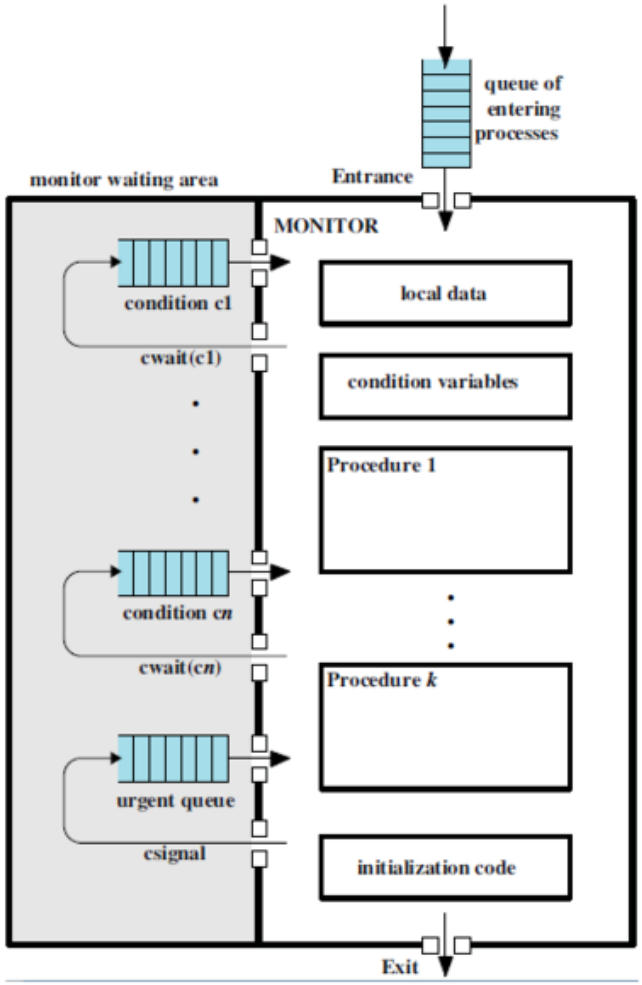
```
monitor Identificador;
                    -- Lista de procedimientos y funciones exportadas
    export Identificador_procedimiento_1;
    export Identificador_procedimiento_2;
    ....
        -- Declaración de constantes, tipos y variables internas
    ....
        -- Declaración de procedimientos y funciones
    ....
    begin
        -- Sentencias de inicialización
    ....
    end;
```

Definición

- Un monitor es una estructura que agrupa la representación y la implementación del acceso a un recurso compartido.

Procedimientos del Monitor

- Tiene un interfaz, que especifica las operaciones exportadas, y un cuerpo que contiene las variables de estado del recurso y la implementación del interfaz.
  - Los procedimientos exportados son el interfaz del monitor, es decir, lo único visible y accesible desde los procesos en su entorno.
  - Alguno de estos procedimientos modifican el estado del monitor



Código de inicialización

Se ejecuta una vez, al iniciar el monitor, con el fin de inicializar las variables permanentes.
Si vemos al monitor como un objeto, debemos notar que se trata de un objeto pasivo, es decir no tiene un hilo interno, y la inicialización se realiza con la ejecución del constructor.

Aspectos característicos del monitor.

- Las estructuras de datos internas del monitor cuya finalidad es ser compartidas por múltiples procesos concurrentes, solo pueden ser inicializadas, leídas y actualizadas por código propio del monitor.
- Los únicos componentes del monitor públicos (visibles desde módulos externos) son los procedimientos y funciones exportadas.
- El monitor garantiza el acceso mutuamente exclusivo a los procedimientos y funciones de la interfaz. Si son invocados concurrentemente por varios procesos, solo la ejecución de un procedimientos del monitor es permitido.
- Los procesos no atendidos son suspendidos hasta que la ejecución del procedimiento atendido acabe.
- Dado que todo el código relativos a un recurso o a una variable compartida está incluido en el módulo del monitor, su mantenimiento es mas fácil y su implementación es mas eficiente.

Aspectos característicos del monitor.

Los **aspectos característicos** de la estructura monitor son:

- Las estructuras de datos compartidas son definidas dentro del monitor, y son inicializadas y gestionadas por componentes definidos dentro del propio monitor.
- Las estructuras de datos no son directamente visibles desde el exterior del monitor, y solo pueden ser accedidas a través de los procedimientos exportados por el propio monitor.
- El acceso mutuamente exclusivo es forzado automáticamente por el código del monitor. No es posible acceder a los datos fuera del régimen de exclusión mutua.
- Los monitores localiza en su código todos los componentes del programa relativos a una variable compartida. El mantenimiento de los programas basados en monitores es mucho más simple que en los basados en regiones críticas.

Exclusión Mutua del Monitor

- Los monitores distinguen entre exclusión mutua (que se tiene por defecto) y condiciones de sincronización.
  - Si dos o mas procesos intentar ejecutar simultáneamente algún procedimiento del monitor, solo uno de ellos tiene éxito, y el resto espera en una estructura de cola (que puede ser FIFO), que llamamos cola de entrada al monitor.

Protección de los datos en el monitor

- Las variables permanentes del monitor son inicializadas con la inicialización del monitor y solo accedidas por los procedimientos que están en el monitor; estos procedimientos usan estas variables para gestionar los recursos y mantener el estado del monitor.
- Mientras que las variables locales, a cada procedimiento del monitor, son iniciadas con cada llamada a los procedimientos y se usan para cálculos intermedios.
- El monitor se comunica con el resto del sistema a través de los parámetros de cada procedimiento.

Sincronización en Monitores

Primitivas de sincronización en monitores

- La sincronización facilita la activación del bloqueo, de un hilo o proceso, según un conjunto de condiciones, esto es realizado de diferentes maneras, según sea:
  - Un **semáforo**, la sincronización se realiza por la cuenta de la variable semáforo
  - Un **monitor**, la sincronización se realiza según una condición, que es representada por datos protegidos del monitor.
- En ambos casos, las variables representan a los recursos y/o condiciones.

Sincronización en Monitores

- Ejemplo de un monitor, cono planificador de un único recurso:

```
monitor recurso;
        var   ocupado:    boolean;
              noocupado: condicion;

        procedure adquirir;
                {if ocupado then noocupado.wait; ocupado := true}

        procedure liberar;
                {ocupado := false; noocupado.signal;}

        begin
                ocupado := false /* valor inicial*/
        end;
```

Condiciones de Sincronización del Monitor

- Se implementan específicamente, se utilizan las variables de tipo condición.
  - Cada variable condición representa una cola que esperan que la condición correspondiente se satisfaga
  - Las operaciones para las variables de tipo condición son las siguientes:

Operación	Significado
delay(c)	suspende el proceso al final de la fifo **c**
resume(c)	reactiva al proceso que está en la cabeza de **c** si **c** está vacía no tiene efecto.
empty(c)	función booleana: **empty(c)**='está vacía c'

Por defecto, las variables condición son colas FIFI

Condiciones de Sincronización del Monitor

- Diferencia entre implementaciones con semaforos y con monitor
  - Los semáforos son variables mas expresivas que las condiciones, un semáforo tiene un valor además de servir como medio para suspender a los procesos.
  - En contraposición, las condiciones no tienen un valor asociado por lo que se hace necesario el uso de variables adicionales (para hacer la cuenta en el buffer limitado).

Acción	Semáforos	Monitores
suspensión del productor	wait(sleido)	if not leido then delay(cleido)
reactivación del consumidor	signal(sdatos)	leido := false; resume(cdatos)
suspensión del consumidor	wait(sdatos)	if leido then delay(cdatos)
reactivación del productor	signal(sleido)	leido := true; resume(cleido)

Variables “Condition”

- El monitor resuelve el acceso seguro a recursos y variables compartidos, pero no tiene las capacidades plenas de sincronización.
- Las variables tipos **condition** que solo pueden declararse dentro de un monitor proporciona a los monitores la capacidad de sincronización plena.
- **condition** es un tipo de variable predefinido que solo puede intanciarse en los monitores

var inviables: **condition**;

  - **Valores**: No toma ningún tipo de valor, pero tiene asociada una lista de procesos suspendidos
  - **Operaciones**:
    - **delay** suspende el proceso que lo ejecuta y lo incluye en la lista de una variable Condition.
    - **resume** Reactiva un proceso de la lista asociada a una variable Condition.
    - **empty** Función que retorna True si la lista de procesos de la variable está vacía.

Operación delay.

- Suspende el proceso que la ejecuta (que ha invocado el procedimiento del monitor en que se encuentra) y lo introduce en la cola asociada a la variable Condition.

  delay (variableCondition) ;

- Semántica de la operación:
  - Cuando un proceso ejecuta la operación delay, se suspende **incondicionalmente**.
  - Cuando un proceso ejecuta la operación delay, se libera la capacidad de acceso que dispone el proceso.
  - En la cola de una variables Condition pueden encontrarse suspendidos un número ilimitado de procesos.

Operación resume.

- Ejecutada sobre una variable tipo Condition, se reactiva uno de los procesos de la lista asociada a la variables.

  resume(variableCondition);

- Semántica:
  - Si la cola de la variable está vacía, equivale a una operación null.
  - Esta operación tiene en su definición la inconsistencia de que tras su ejecución existen dos procesos activos dentro del monitor, lo que contradice su principio de de operación. Diferentes modelos del procedimiento resume, que han sido utilizados:
    - **Reactiva y Continua**: El proceso activado permanece suspendido hasta que el proceso que ha invocado resume libera el monitor.
    - **Reactivación_Inmediata**: La ejecución de la operación reasume, supone la liberación del monitor por parte del proceso que la realiza. El único proceso activo dentro del monitor es el proceso que se ha reactivado.
    - **Reactiva y espera**:El proceso que ejecuta resume se bloquea en una cola especial denominada “urgent”, y el proceso activado es el que tiene el acceso sobre el recurso. Cuando el proceso suspendido sobre la cola “urgent” se activa ejecuta la sentencia que sigue a la sentencia resume que la bloqueó.

Disciplinas de Reactivación

- Cuando el proceso que está en el monitor cambia el estado de este, ahora se satisface alguna condición que antes no era cierta
- Se ejecuta una instrucción resume sobre dicha condición, que despierta un proceso de entre los que están esperando.
- La imposition de la exclusion mutua hace que cuando un proceso, *que esta dentro del monitor*, ejecuta una instrucción de reactivación, *resume*, **haya que decidir que proceso continua en el monitor:**

El reactivador o el proceso reactivado

  - Pueden considerarse distintas estrategias para resolver este problema que influyen fuertemente en el modo de implementar las condiciones de sincronización en el monitor.

Disciplinas de Reactivación

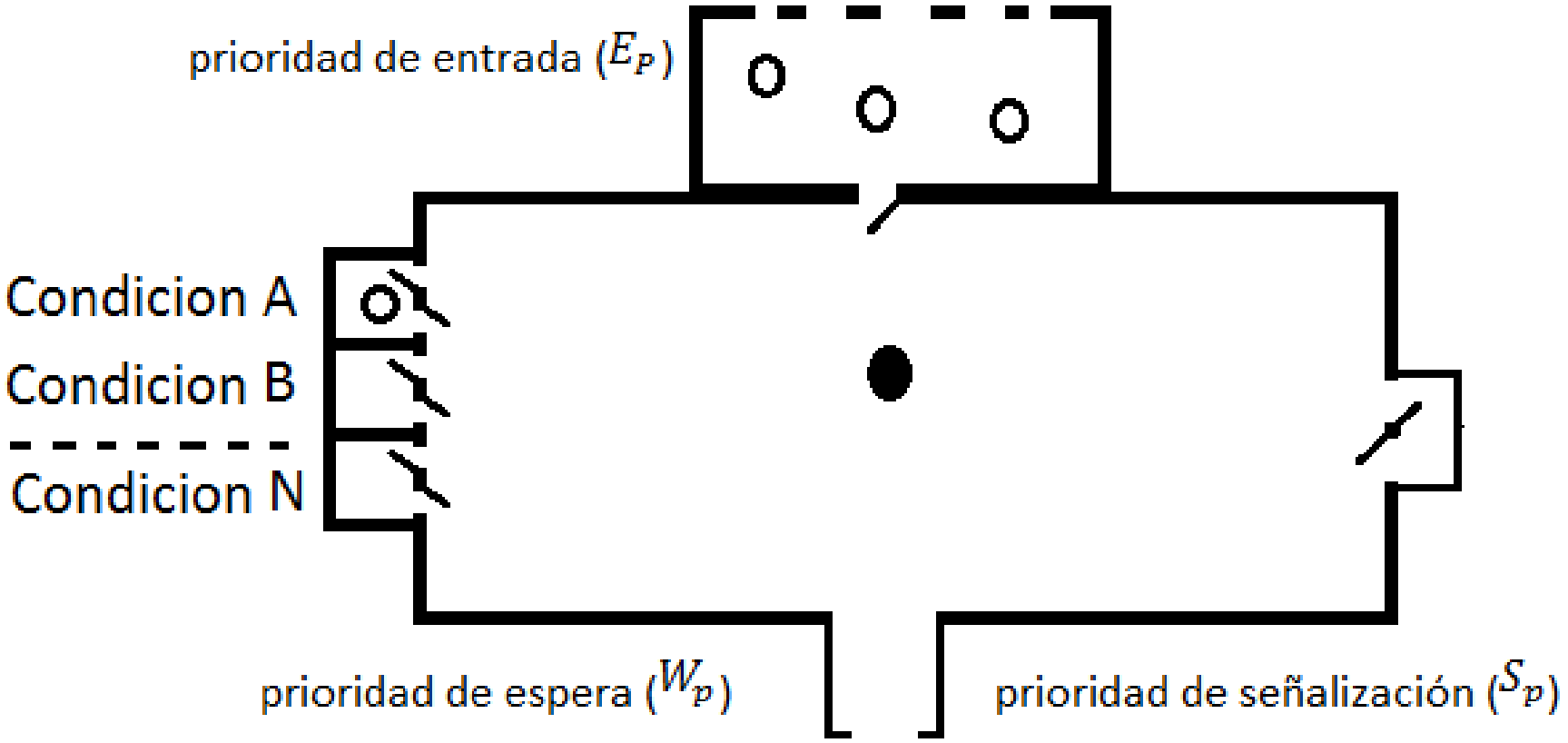
Resume-and-Exit (RE)

- En esta disciplina se obliga a que la instrucción de reactivación, ***resume*, sea siempre la ultima que ejecuta el proceso reactivador antes de salir del monitor**
- En este caso, el comportamiento del monitor es sencillo:
  - El proceso reactivador sale del monitor y entra el reactivado.
  - Si no hay ningún proceso que reactivar, entra un proceso de la cola de entrada al monitor.
  - Finalmente, si no hay nadie esperando a la entrada, se desbloquea la entrada al monitor.
- El problema de esta disciplina es que tiende a generar procedimientos en los monitores muy particionados ya que, habitualmente las operaciones de tipo *resume* no ocurren al final de los procedimientos.

Resume-and-Exit (RE)
Retorno Forzado (DR)

- La operación resume se implementa con exit
- La operación resume que es el desbloqueador (del proceso que ejecuto el delay) debe ejecutar return
- El monitor ha sido cedido al proceso desbloqueado
- Aquí el proceso desbloqueado no tiene que comprobar la condición por la que se ejecutó el resume

Modelo grafico del Monitor



Disciplinas de Reactivación

Resume-and-Wait (RW)

- En este caso el proceso reactivador **sale del monitor una vez que ha realizado la operación *resume* dando prioridad al proceso despertado**.
  - La ventaja de esta solución es que el proceso despertado sabe que entre su reactivación y su incorporación al monitor, ningún otro proceso ha modificado la condición por la que tuvo acceso al mismo, por lo que puede continuar su ejecución directamente
- Esta solución hace que el proceso reactivador vuelva a la cola de entrada, por lo que tiene que competir con el resto de los procesos del entorno para entrar en el monitor.
  - En este sentido es un poco injusta, ya que el proceso reactivador ya estaba dentro del monitor y se le trata igual que a los que estaban fuera.
  - Una solución un poco mas justa es (RUW)

Resume-and-Wait (RW)
Desbloquear y Esperar (DE)

- El proceso desbloqueador sede el monitor y la exclusión-mutua (resumen and wait) al proceso desbloqueado, para esto:
  - El proceso desbloqueador se bloquea, en la cola de entrada al monitor, después de ejecutar resume
  - El proceso desbloqueador tiene que volver a competir por su condición para salir del monitor
- La condición para el proceso desbloqueado no ha cambiado
- El proceso desbloqueador tiene que competir por el monitor, con los procesos que esperan para entrar al monitor. Esta condición puede no ser justa.

Disciplinas de Reactivación

Resume-and-Urgent-Wait (RUW)

- Esta disciplina es como la anterior salvo que el proceso reactivador no se transfiere a la cola de entrada, sino a una cola especial, llamada de urgentes (cortesía), formada por procesos que tienen prioridad para entrar en el monitor sobre los que están en la cola de entrada.

Resume-and-Urgent-Wait (RUW)
Desbloqueo y Espera Urgente (DU)

- Esta política es también llamada *resume an urgent wait,* soluciona la falta de equidad de la politica DE.
  - Se asocia al monitor una nueva cola, **“cola de cortesía”**
  - El proceso desbloquedor desbloquea al proceso desbloqueado
  - El proceso desbloqueador se bloquea en la cola de cortesía, y sede el monitor al desbloqueado.
  - El proceso dessbloqueado desbloquea al proceso desbloqueador que está en la cola de cortesía antes de abandonar el monitor.

- Para entender lo que sucede hay que tener presente que cada proceso tiene uno de tres estados: bloqueado, listo o en ejecución.
- Con esta estrategia el proceso desbloqueador ha sedido el tiempo que el SO le ha otorgado al proceso desbloqueado.

Disciplinas de Reactivación

Resume-and-Continue (RC)

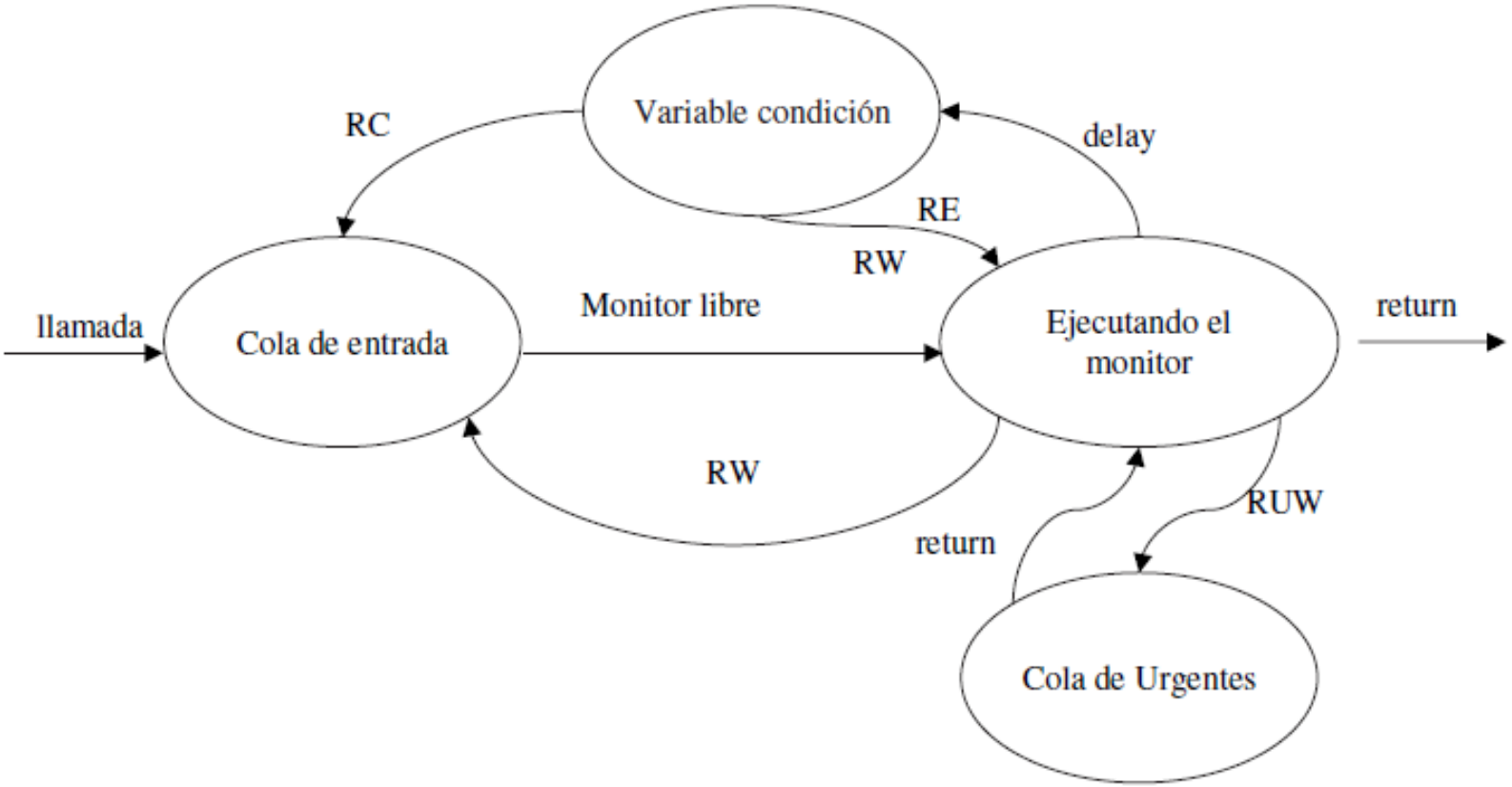
- Según esta estrategia el proceso reactivador sigue ejecutando el monitor y el despertado se va a la cola de entrada.
- La principal diferencia con la solución anterior es que desde que el proceso fue despertado hasta el momento en el que entra en el monitor, han podido entrar en este un numero arbitrario de procesos.
- Por lo tanto, es posible que la condición por la que se le despertó haya cambiado (el estado del monitor puede haber sido modificado por otro proceso) y, en ese caso, tendrá que volver a suspenderse.

Resume-and-Continue (RC)
Desbloquea y Continúa (DC)

- Hay un proceso vivo dentro del monitor llamado **desbloqueador**.
- Este desbloquea (resume) a un proceso bloqueado, que llamamos proceso **desbloqueado**.
- El proceso **desbloqueador continua su ejecución** hasta que sale del monitor o se bloquea en una variable de condición
- En ese momento el proceso desbloqueado es seleccionado y continua su ejecución en la **instrucción siguiente al delay** que ejecuto al bloquearse.
- Aquí no es posible garantizar que la condición por la que se ejecutó el resume siga siendo cierta y hay que verificarla

Disciplinas de Reactivación

En la Figura se muestran estas disciplinas. Comentadas en las filminas anteriores



Características fundamentales de esta operación son:

- Cuando se ejecuta la operación resume sobre una variable que tiene su cola asociada vacía, equivale a una operación **null.**
- Un problema potencial que lleva consigo la ejecución de esta operación es que de ella resultan dos procesos activos dentro del monitor, uno el que la ejecuta que obviamente debe estar con acceso al monitor para poder llevarla a cabo, y otro el proceso de la cola asociada a la variable tipo "condition" que se activa. A este problema se le han dado dos
- soluciones alternativas:
  - **1. Reactiva_y_continua**: En este caso se supone que el proceso que se activa permanece aún suspendido hasta que justamente el proceso que ejecuto "resume“ concluya el uso del monitor. Esta alternativa fue utilizada por Mitchell (1979) en la implementación del lenguaje concurrente Mesa.
  - 2. **Reactivación_inmediata**: En este caso el proceso que realiza la operación "resume" abandona inmediatamente del monitor, y deja que el proceso que se ha reactivado, sea el único que permanece de forma exclusiva dentro del monitor. Esta es la variante que mas frecuentemente se ha adoptado, y es la que consideraremos en lo que sigue. En este caso, la sentencia resume **debe ser siempre la última del procedimiento** del monitor en que se encuentra.
- De esta última alternativa existen diferentes variantes que han sido utilizadas en diferentes lenguajes: Pascal Concurrente (Brinch Hansen, 1975), Pascal Plus (Welsh y Bustard, 1979). Andrews (1991) demostró todos estos protocolos para la operación "resume" son equivalentes, ya que con cada uno de ellos se pueden simular la operación del otro.

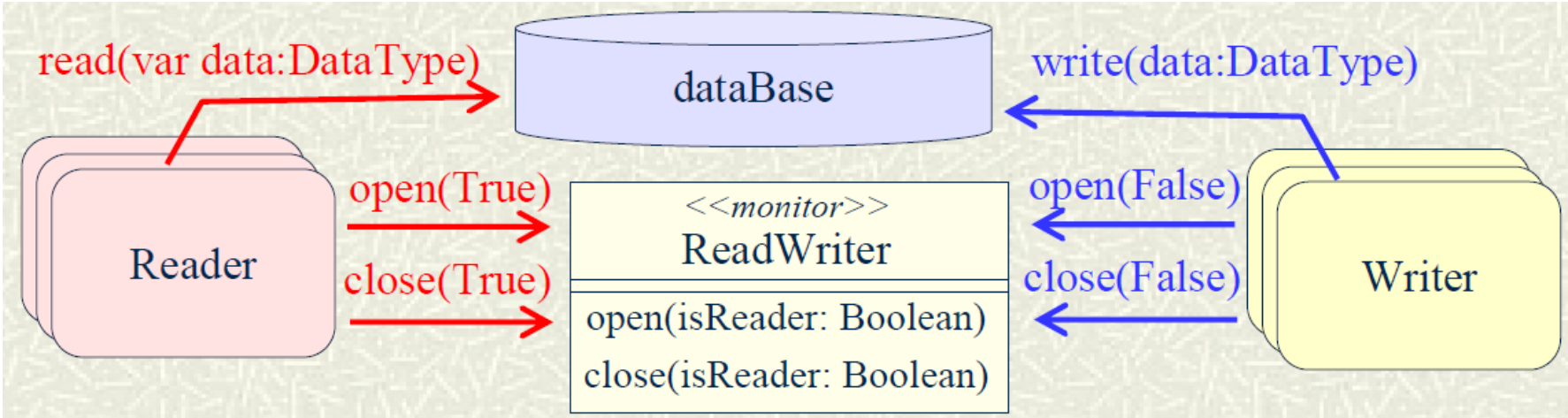
Problema de Anidación de llamadas en monitores

- Son sistema estructurado como colección jerárquica de monitores, donde los procesos de un monitor pueden llamar a los procesos de otro monitor.
- Los problemas que presenta este escenario son:
  - Una llamada de monitor anidada se suspende en el último monitor. La E.M. en el último se abandonará pero no en el monitor desde el que se llama.
  - Los procesos que intenten llamar a procedimientos de cualquier monitor de la cadena se bloquearán.
  - Se obtiene menor concurrencia, por lo que el rendimiento del sistema es menor.

Propuestas de solución

- Prohibir llamadas anidadas.
- Liberar exclusión mútua en todos los monitores implicados en la cadena y bloquear proceso.
  - Una vez señalado, el proceso necesitará readquirir el acceso exclusivo a todos los monitores.
  - Requerirá que el invariante del monitor se establezca antes de cualquier llamada que pueda bloquear.
- Monitores = herramienta de estructuración para recursos compartidos. E.M. → sólo forma de preservar integridad del recurso.
  - Hay casos en los cuales las operaciones de un monitor pueden ejecutarse concurrentemente sin efectos adversos.
  - Definir construcción que permita especificar que ciertas operaciones se podrán ejecutar concurrentemente y la exclusión mútua se liberará.

Problema de los lectores y escritores




```
process type Reader;
  var theData:DataType;
begin
  repeat
    ReadWriter.open(True);
       read(theData:DataType);
    ReadWriter.close(True);
     ...       - - Usa el dato
  forever;
end;
```

```
process type Writer;
  var theData:DataType;
begin
  repeat
     ...       - - Crea el dato
    ReadWriter.open(False);
       write(theData:DataType);
    ReadWriter.close(False);
  forever;
end;
```

Comparación entre modelos de concurrencia.

- Utilizamos dos criterios complementarios de comparación:
  - **Poder expresivo**: Es la capacidad de la primitiva para implementar algoritmos o resolver problemas de sincronización en programas concurrentes.
    - Si una primitiva tiene capacidad de implementar otra, tiene al menos su capacidad expresiva.
    - Todas las primitivas que hemos estudiado tienen la misma capacidad expresiva,
  - **Facilidad de uso**: Criterio subjetivo que se refiere a aspectos, tales como:
    - Como de natural es el uso de la primitiva.
    - Como de fácil es combinar la primitiva con otras sentencias del lenguaje.
    - Cuan proclive es la primitiva para que el programador cometa errores.

Facilidad de uso de las primitivas de sincronización.

- La sincronización a través de semáforos presenta la capacidad plena, pero su facilidad de uso es muy pobre por su bajo nivel de abstracción y su gestión distribuida.
- Los Monitores incrementan la abstracción y concentran su gestión, pero al final requieren variables “Condition” que son también de muy bajo nivel de abstracción aunque permanecen localizadas en el monitor.
- La invocación de procedimientos remotos es de alto nivel de abstracción y de uso muy seguro, aunque presenta dos problemas:
  - Implementa los objetos pasivos con un modelo de módulo activo que no corresponde a lo que el programador espera.
  - Conduce a implementaciones ineficientes como consecuencia del gran número de cambios de contextos.
- No hay una solución absolutamente favorable. Para una programación segura, mantenible y eficiente se requiere combinar varios tipos de mecanismos.

Primitivas

1. **Los semáforos y las variables de condición son de un nivel muy bajo, a efectos de ser consideradas adecuadas para un lenguaje de programación de propósito general.**
   - Esta afirmación no es contraria al hecho de reconocer a los semáforos su carácter de componente básico que todas los sistemas lo van a utilizar a bajo nivel.
   - El semáforo juega en este aspecto un papel equivalente al de la sentencia GO TO en programación secuencial.
2. **Los método de paso de mensajes (incorporando la facilidad de condiciones de guarda) representan un mecanismo más abstracto y unificado para la programación concurrente.**
   - El modelo de lenguaje a que dá lugar estas primitivas es más natural y seguro, y además simplifica considerablemente el lenguaje al eliminar el uso de variables compartidas.
3. **La invocación remota de procedimientos es la abstracción de alto nivel más efectiva del tipo de paso de mensajes.** La invocación remota tiene un gran número de ventajas respecto de la comunicación sincrona:
   - No requiere introducir el concepto intermedio de canal o buzón.
   - No requiere simetría en las sentencias Select.
   - El lenguaje resulta de mas simple al utilizar una sintaxis similar para la invocación de procedimiento remotos y de procedimientos locales.
4. **Las estructuras tipo Monitor constituyen un método efectivo de encapsular recursos compartidos**. Constituye la mejor abstracción de un recurso si excluimos su modelado mediante un proceso. No obstante, presenta el problema de requerir variables condicionadas, para que cubra la totalidas de las posibilidades de sincronización.