Sistemas Operativos

UNIDAD 3

EXCLUSIÓN MUTUA Y SINCRONIZACIÓN CON MONITORES

Exclusión Mutua

SOLUCIONES SOPORTE SO

MONITORES

Exclusión Mutua: MONITORES

Desventajas de los semáforos

- Aunque permiten soluciones eficientes, **su uso es complicado**.
- Se basan en memoria compartida, en variables compartidas, por lo que resulta **difícil modularizar y estructurar** el código.
- El hecho de que los semáforos puedan jugar dos roles, sincronización y exclusión mutua, hace **complejo el seguimiento del código**
- Es **delicado modificar el código**, la inclusión y eliminación de sentencias puede provocar interbloqueos o hacer que la solución deje de funcionar.
- Es **difícil la depuración** de problemas complejos.
- **No se puede saber en tiempo de compilación si se accede de manera indebida** a un recurso compartido o no.
- **No se pueden proteger** los recursos críticos **de un uso indebido del programador**.

Necesitamos una nueva herramienta que:

- Permita estructurar y modularizar el código
- Manteniendo un mecanismo de control sobre la sincronización
- Debe permitir encapsular los datos compartidos de forma que no se pueda hacer mal uso de los datos por parte del programador

Exclusión Mutua: MONITORES

Un monitor es una estructura del lenguaje de programación

Consta de:

- uno o más procedimientos,
- una secuencia de inicio y
- datos que simbolizan el recurso compartido.

Es un mecanismo de abstracción de datos que

- Permite representar de forma abstracta un recurso compartido por medio de un conjunto de variables que definen su estado.
- Los procedimientos del monitor son visibles desde el exterior
- El acceso a las variables del monitor solo es posible mediante los procedimientos del monitor

Un programa concurrente que utiliza monitores tiene dos tipos de procesos

- Procesos pasivos

  Los que implementan los monitores

  Están a la espera que los activos hagan uso de los procedimientos exportados del monitor

  Solo pueden interactuar con las variables compartidas del interior del monitor a través de estos procedimientos exportados
- Procesos activos

  Los que hacen uso de los procedimientos del monitor.

Exclusión Mutua: MONITORES

Ventajas

- Los programadores de los procesos activos no tienen porque conocer la implementación del recurso compartido
- Si se modifica la forma de gestión del recurso compartido sin modificar la interfaz de procedimientos, los procesos activos no se ven afectados
- El programador del monitor no se preocupa de donde y como se usa el monitor, solo debe ocuparse de que funcione correctamente
- Se pueden implementar en módulos diferentes, lo que mejora la estructuración del código.

Exclusión mutua

Un monitor está construido de tal forma que:

- La ejecución de los procedimientos del monitor (internos o externos) no se ejecutan nunca de forma solapada
- Se impide que dos procesos activos estén ejecutando simultáneamente procedimientos del monitor, (el mismo o distintos)

Exclusión Mutua: MONITORES

Esquema de un monitor

```
monitor nombre_monitor;
  var variables locales;
  export procedimientos exportados;

  procedure proc1(parametros);
  var: variables locales;
  begin
    //codigo del procedimiento
  end;

  procedure proc2(parametros);
  var: variables locales;
  begin
    //codigo del procedimiento
  end;

begin
  //codigo de inicialización
end;
```

Un monitor se compone de:

- Conjunto de variables locales que se usan para almacenar el estado interno del recurso que representa el monitor o el estado de algún procedimiento interno.
- Estas variables no pueden variar entre llamadas sucesivas al monitor *(llamadas a uno de los procedimientos exportados)*
- El código de inicialización se ejecuta antes de la primera instrucción del programa que usa el monitor
- El conjunto de procedimientos (internos o exportados) pueden modificar las variables locales
- Los procedimientos pueden aceptar parámetros

Exclusión Mutua: MONITORES

Un proceso activo

- Ve al monitor como un conjunto de procedimientos que permiten usar un recurso compartido
- En la llamada a uno de estos procedimientos está implícito el uso que se hace del recurso
- Para invocar un procedimiento : `nombre_monitor.nombre_procedimiento(parametros)`

Decimos que:

- un proceso activo está dentro del monitor cuando está ejecutando uno de los procedimientos exportados de éste
- un proceso activo abandona el monitor cuando termina de ejecutar el código del procedimiento exportado

El acceso exclusivo en la ejecución de los procedimientos exportados del monitor se basa en:

La existencia de una **cola del monitor**

- Cuando un proceso está dentro del monitor: si otro proceso intenta acceder al monitor el código del monitor *(el generado por el compilador)* bloquea al segundo proceso en la cola del monitor.
- Cuando un proceso abandona el monitor: el código del monitor selecciona el proceso en cabeza de cola del monitor y lo desbloquea, entrando dicho proceso en el monitor.
  Si la cola esta vacía el monitor queda libre.

La responsabilidad de bloquear al proceso es del monitor, no del proceso activo.

Exclusión Mutua: MONITORES

Ejemplo:

- Dos procesos deben incrementar el valor de una variable compartida y poder examinar su valor en cualquier momento.

Necesitamos

- Una variable permanente representa la variable compartida
- Un procedimiento para incrementar el valor
- Un procedimiento para consultar el valor
- Inicializar la variable compartida

```
monitor incremento;
  var i:integer;
  export inc,valor;

  procedure inc;
  begin
    i:=i+1;
  end;

  function valor:integer;
  begin
    valor:=i;
  end;

begin
  i:=0;
end;
```

```
incremento.inc();
v:=incremento.valor();
```

Exclusión Mutua: MONITORES

Un monitor incluye **variables de condición** que sólo son accesibles desde dentro del monitor.

Estas variables de condición proporcionan el mecanismo de sincronización entre procesos.

Permiten bloquear/desbloquear un proceso que está dentro del monitor

- El procedimiento que invoca al monitor se bloquea cuando *(el código de un procedimiento de monitor)* se detecta/cumple una condición bloqueante en el recurso compartido
- Cuando otro proceso *(al usar un procedimiento)* modifica el estado del recurso generando una condición de desbloqueo, liberará al proceso bloqueado en la condición.

Una variable de condición se declara dentro del monitor

```
var
   cond:condition;
   array_cond: array [1..5] of condition;
```

Los valores de las variables condición **no son accesibles por el programador del monitor**.

Son la representación de colas FIFO, inicializadasvacías en su declaración

Exclusión Mutua: MONITORES

Operaciones sobre las condiciones

- Operación ***delay(C)*** *o cwait(C)*
  - Permite bloquear al proceso que ejecuta el monitor en la cola de la condición C
  - Se bloquea incondicionalmente al final de la cola FIFO
  - Realiza las siguientes acciones

```
• liberar la exclusión mutua del monitor
• bloquear el proceso llamante en la cola C
```

- Operación ***resume(C)*** *o csignal(C)*
  - Permite desbloquear al proceso cabecera de la cola de la condición C
  - Si la cola está vacía la operación no hace nada
  - Realiza las siguientes acciones

```
• liberar la exclusión mutua del monitor
• desbloquea al proceso en la cola C que tendrá
  prioridad sobre los procesos en cola de monitor
• se bloquea el proceso llamante en la cola de cortesía
```

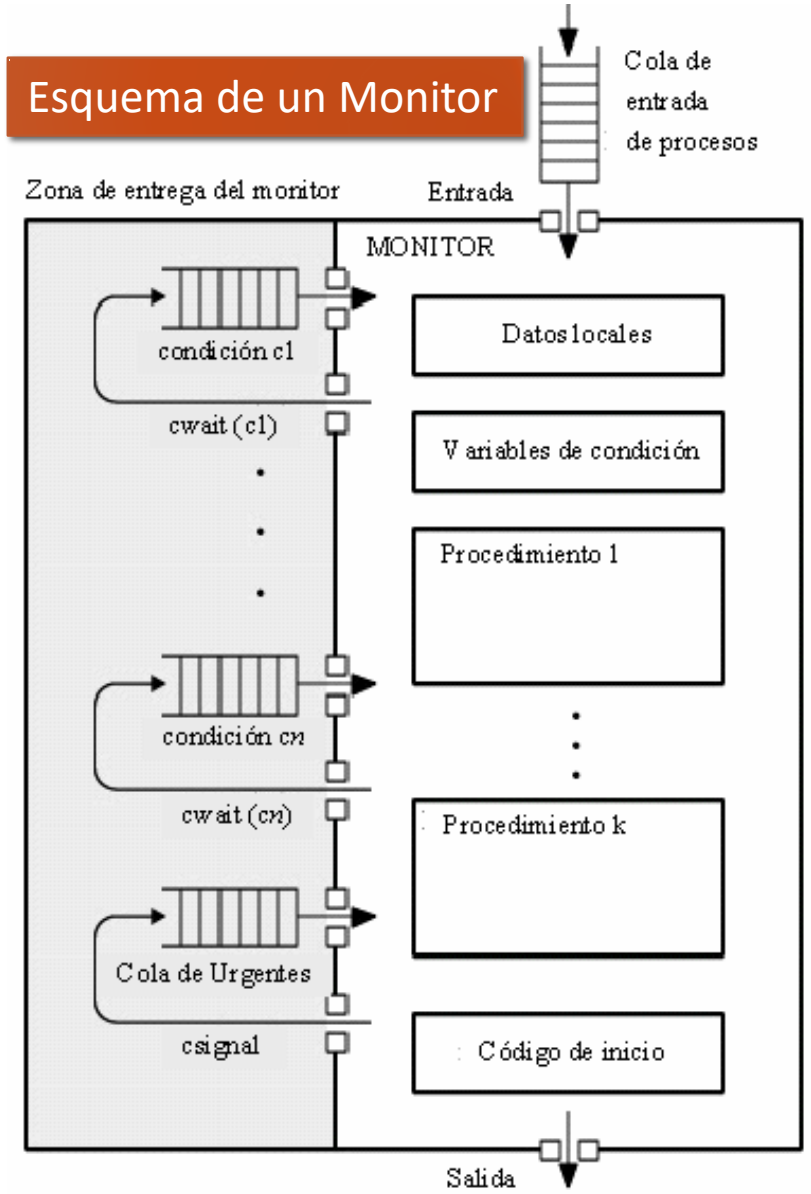
Exclusión Mutua: MONITORES

La **cola de cortesía** o de urgentes

- El monitor tiene una cola FIFO *(cola de cortesía)* en la que se bloquean todos los procesos que ceden el monitor como consecuencia de la ejecución de *resume(C)*
- Esta cola tiene prioridad frente la cola de monitor
- Si hay procesos esperando en ambas colas, serán desbloqueados primero los procesos en cola de cortesía.

La función ***empty(C)***

- En algunos casos es útil conocer si bajo una condición existen o no procesos bloqueados
- La función booleana empty(C) devolverá true o false si la cola de la condición C está llena o vacía respectivamente.



Exclusión Mutua: MONITORES

Productor/Consumidor con buffer limitado:

```
/*program productor_consumidor*/
monitor monitorPC;
char buffer[N];  //Buffer Acotado
int sigent, sigsal;
int contador;
condition cLleno, cVacio;
export añadir,coger;

void añadir(char x){
        if (contador ==N) delay(cLleno);
        buffer[sigent] = x;
        sigent=(sigent + 1) % N;
        contador++;
        resume(cVacio);
        }

void coger(char x){
        if (contador ==0) delay(cVacio);
        x = buffer[sigsal];
        sigsal=(sigsal + 1) % N;
        contador--;
        resume(cLLeno);
}
{
        sigent=0; sigsal=0; contador=0;
}
```

```
void productor(){
        char x;
        while true {
                producir(x);
                 buffer.añadir(x);
                }
        }


void consumidor() {
        char x;
        while true {
                buffer.coger(x);
                consumir(x);
                }
        }

void main(){
        monitor buffer = new(monitorPC);
        parbegin(productor, consumidor);
}
```

Exclusión Mutua: MONITORES

Ejercicio

Implementar un semáforo binario utilizando monitores

Necesitaremos:

- Las operaciones ***wait y signal***
- Una variable condición ***csem*** para bloquear a aquellos procesos que ejecuten un ***wait*** sobre nuestro semáforo
- Una variable booleana ***sem*** para indicar si el semáforo está abierto o cerrado
- La operación ***empty(condición)*** que Pregunta si hay procesos en cola de la condición.

Exclusión Mutua: MONITORES

Semáforo binario implementado con un monitor

```
monitor sbinario;
  bool sem; //true→cerrado
  condition csem;
  export wait,signal;

  void wait(){
    if sem delay(csem);
    sem=true;
  }

  void signal(){
    if !empty(csem)resume(csem);
    else sem=false;
  }

begin
  sem=false; //Abierto
end;
```

```
sbinario.wait();

sbinario.signal();
```

Exclusión Mutua: MONITORES

El barquero (Hoare)

Un proceso, al que vamos a llamar el barquero, se encarga de transportar pasajeros de una parte a otra de un río. El resto de procesos, a los que llamaremos pasajeros, utilizan la barca para cruzar. Escribir el algoritmo del barquero y los pasajeros utilizando monitores como herramienta de sincronización para que se cumplan los siguientes requisitos:

- Los pasajeros deben esperar en la orilla izquierda hasta que llegue el barquero.
- Cuando el barquero llegue a la orilla izquierda, debe esperar a que haya al menos un pasajero preparado para subir.
- Cuando hayan subido 10 pasajeros o, si eran menos, todos los que hubiesen esperando, la barca debe iniciar la travesía.
- Al llegar a la orilla derecha el barquero avisará a los pasajeros, que bajarán de la barca.
- Cuando hayan bajado todos los pasajeros, el barquero iniciará el viaje de regreso a la orilla izquierda con nuevos pasajeros.

```
monitor Barco {
   var
      numOrilla,numBarca:integer;

      cEsperaPasajeros,
      cColaDeEspera,
      cColaDeBajada,
      cZarpar,
      cBajar  :condition

   export
      PermitirSubir,PermitirBajar,
      EsperaEnCola, Subir, Bajar;

   void PermitirSubir(){
      if(numOrilla==0)
            delay(cEsperaPasajeros)
      resume(cColaDeEspera);
      delay(cZarpar);
   }

   void PermitirBajar(){
      resume(cColaDeBajada);
      delay(cVolver);
   };

   ....
```

```
....
void EsperaEnCola(){
   NumOrilla++;
   resume(cEsperaPasajeros)
   delay(cColaDeEspera);
}

void Subir(){
   numOrilla--;
   numBarca++;
   if (numBarca<=10 && numOrilla>0)
      resume(cColaDeEspera);
   else
      resume(cZarpar);
   endif
}

void Bajar(){
   delay(cColaDeBajada);
   numBarca--;
   resume(cColaDeEspera);
   if numBarca==0
      notify(cVolver);
}

{ //init Monitor
   numOrilla=0;
   numBarca=0;
};

} // monitor
```

```
void Barquero(){
   while true{
      Barco.PermitirSubir();
      /*Cruzar Rio*/
      Barco.PermitirBajar();
      /*Volver a por mas*/
   }

void P1() … P(N){
   Barco.EsperaEnCola();
   Barco.Subir();
   /*Cruzando Rio*/
   Barco.Bajar();
}

void main(){
   Barco miBarco=new(Barco);
   parbegin(Barquero, P1, …. Pn);
end
```

Exclusión Mutua: MONITORES

La versión de Hoare *(monitores con señalización)* tiene los siguientes inconvenientes:

- Si el proceso que hace ***resume*** hacen falta dos cambios de proceso: para suspender el proceso y para reanudarlo cuando corresponda.
- Cuando se efectúa ***resume,*** *y no ha terminado su proceso,* se coloca en la cola de cortesía. Si el procedimiento debe ejecutar un **delay** después del **resume** que lo colocó en cortesía, la sincronización puede fallar.

Versión de Lampson y Redell

(Monitores con notificación y difusión)

- La primitiva ***resume*** se sustituye por ***cnotify***
- Permiten mensajes de difusión o broadcast. ***cbroadcast***.
- Establece temporizadores en las colas de condición

cnotify(C)

Cuando un proceso ejecuta *cnotify* sobre una condición, provoca que la cola de esa condición sea notificada, pero el proceso continúa su ejecución hasta salir del monitor (terminación del procedimiento o espera en condición)

Cuando se notifica una cola de condición, el proceso cabecera de la cola será puesto en ejecución (Listos) cuando se libere el monitor.

Si la cola de Listos no es round robin (FIFO) no hay garantía de que no se cuele en el monitor otro proceso que altere la condición, por lo que el proceso notificado deberá volver a comprobarla (if → while)

con temporizador

Se pueden mejorar con un temporizador asociado a cada cola de condición.

Cuando vence el timer el proceso pasa a (Listo) independientemente de que fuera notificada la condición.

Esto garantiza que no habrá inanición si falla un proceso antes de notificar pero obliga a volver a determinar si se cumple la condición.

croadcast©

Permite sacar a todos los procesos de una condición. Útil si no se cuantos son o deben salir todos.

Exclusión Mutua: MONITORES

El barquero (Lampson)

Un proceso, al que vamos a llamar el barquero, se encarga de transportar pasajeros de una parte a otra de un río. El resto de procesos, a los que llamaremos pasajeros, utilizan la barca para cruzar. Escribir el algoritmo del barquero y los pasajeros utilizando monitores como herramienta de sincronización para que se cumplan los siguientes requisitos:

- Los pasajeros deben esperar en la orilla izquierda hasta que llegue el barquero.
- Cuando el barquero llegue a la orilla izquierda, debe esperar a que haya al menos un pasajero preparado para subir.
- Cuando hayan subido 10 pasajeros o, si eran menos, todos los que hubiesen esperando, la barca debe iniciar la travesía.
- Al llegar a la orilla derecha el barquero avisará a los pasajeros, que bajarán de la barca.
- Cuando hayan bajado todos los pasajeros, el barquero iniciará el viaje de regreso a la orilla izquierda con nuevos pasajeros.

```
monitor Barco {
   var
      numOrilla,numBarca:integer;

      cEsperaPasajeros,
      cColaDeEspera,
      cColaDeBajada,
      cZarpar,
      cBajar  :condition

   export
      PermitirSubir,PermitirBajar,
      EsperaEnCola, Subir, Bajar;

   void PermitirSubir(){
      if(numOrilla==0)
            delay(cEsperaPasajeros)
      cnotify(cColaDeEspera);
      delay(cZarpar);
   }

   void PermitirBajar(){
      cbroadcast(cColaDeBajada);
      delay(cVolver);
   };

   ....
```

```
....
void EsperaEnCola(){
   NumOrilla++;
   cnotify(cEsperaPasajeros)
   delay(cColaDeEspera);
}

void Subir(){
   numOrilla--;
   numBarca++;
   if(numBarca<=10 && numOrilla>0)
      cnotify(cColaDeEspera);
   else
      cnotify(cZarpar);
   endif
}

void Bajar(){
   delay(cColaDeBajada);
   numBarca--;
   if numBarca==0
      notify(cVolver);
}

{ //init Monitor
   numOrilla=0;
   numBarca=0;
};

} // monitor
```

```
void Barquero(){
   while true{
      Barco.PermitirSubir();
      /*Cruzar Rio*/
      Barco.PermitirBajar();
      /*Volver a por mas*/
   }

void P1() … P(N){
   Barco.EsperaEnCola();
   Barco.Subir();
   /*Cruzando Rio*/
   Barco.Bajar();
}

void main(){
   Barco miBarco=new(Barco);
   parbegin(Barquero, P1, …. Pn);
end
```

Exclusión Mutua: MONITORES

El barquero (Lampson)

Un proceso, al que vamos a llamar el barquero, se encarga de transportar pasajeros de una parte a otra de un río. El resto de procesos, a los que llamaremos pasajeros, utilizan la barca para cruzar. Escribir el algoritmo del barquero y los pasajeros utilizando monitores como herramienta de sincronización para que se cumplan los siguientes requisitos:

- Los pasajeros deben esperar en la orilla izquierda hasta que llegue el barquero.
- Cuando el barquero llegue a la orilla izquierda, debe esperar a que haya al menos un pasajero preparado para subir.
- Cuando hayan subido 10 pasajeros o, si eran menos, todos los que hubiesen esperando, la barca debe iniciar la travesía.
- Al llegar a la orilla derecha el barquero avisará a los pasajeros, que bajarán de la barca.
- Cuando hayan bajado todos los pasajeros, el barquero iniciará el viaje de regreso a la orilla izquierda con nuevos pasajeros.

```
monitor Barco {
   var
      numOrilla,numBarca:integer;

      cEsperaPasajeros,
      cColaDeEspera,
      cColaDeBajada,
      cZarpar,
      cBajar  :condition

   export
      PermitirSubir,PermitirBajar,
      EsperaEnCola, Subir, Bajar;

   void PermitirSubir(){
      while(numOrilla==0)
            delay(cEsperaPasajeros)
      cnotify(cColaDeEspera);
      delay(cZarpar);
   }

   void PermitirBajar(){
      cbroadcast(cColaDeBajada);
      delay(cVolver);
   };

   ....
```

```
....
void EsperaEnCola(){
   NumOrilla++;
   cnotify(cEsperaPasajeros)
   delay(cColaDeEspera);
}

void Subir(){
   numOrilla--;
   numBarca++;
   if(numBarca<=10 && numOrilla>0)
      cnotify(cColaDeEspera);
   else
      cnotify(cZarpar);
   endif
}

void Bajar(){
   delay(cColaDeBajada);
   numBarca--;
   if numBarca==0
      notify(cVolver);
}

{ //init Monitor
   numOrilla=0;
   numBarca=0;
};

} // monitor
```

```
void Barquero(){
   while true{
      Barco.PermitirSubir();
      /*Cruzar Rio*/
      Barco.PermitirBajar();
      /*Volver a por mas*/
   }

void P1() … P(N){
   Barco.EsperaEnCola();
   Barco.Subir();
   /*Cruzando Rio*/
   Barco.Bajar();
}

void main(){
   Barco miBarco=new(Barco);
   parbegin(Barquero, P1, …. Pn);
end
```

Fin

UNIDAD 3

EXCLUSIÓN MUTUA Y SINCRONIZACIÓN CON MONITORES