Comunicación y sincronización

- Son conceptos relacionados con la interacción entre los procesos
- La **comunicación** se refiere al paso de información de un proceso a otro
- La **sincronización** corresponde al cumplimiento de las dependencias temporales entre diferentes procesos
- Son conceptos muy relacionados entre si
- Estudiaremos la comunicación y los mecanismos de sincronización que son necesarios

Comunicación

- Mecanismos
  - Variables compartidas
    - Objetos a los que puede acceder más de un proceso
    - Fáciles de usar si existe memoria compartida
  - Paso de mensajes
    - Intercambio de datos entre dos procesos por medio de un mensaje que se envían
- La elección del mecanismo depende del lenguaje. En los lenguajes de TR se suelen ofrecer ambos
- Con un mecanismo es posible implementar el otro

Comunicación basada en memoria compartida

- Precisa de métodos de sincronización adicionales:
  - Exclusión mutua : Secuencia de instrucciones que deben ejecutarse en una operación indivisible (sección crítica)
  - Sincronización condicional: Un proceso necesita esperar a que se cumpla alguna condición que controla otro proceso
- Vamos a estudiar:
  - Mecanismos: Distintas formas de implementar estos tipos de sincronización
  - Construcciones usadas en los lenguajes de programación que facilitan la sincronización entre los procesos

Exclusión mútua y sincronización condicional (mecanismos)

- Espera ocupada
  - La tarea se bloquea en un bucle
  - Única posibilidad entre distintos procesadores
- Suspender y reanudar
  - Dos operaciones para bloquear y despertar
  - Variante con memoria
- Semáforos
  - Elementos externos al proceso
  - Usan un contador. Al hacerse negativo bloquea.
  - Operaciones P (esperar) y V (señalizar)

Construcciones en los lenguajes

- Regiones críticas condicionales
  - Permite ejecutar una sección de código bajo exclusión mutua
- Monitores
  - Agrupa las distintas operaciones de una sección crítica
- Objetos protegidos
  - Monitores mas evolucionados con distintos tipos de acceso

Regiones críticas condicionales

- Construcción del lenguaje que define un bloque de código como sección crítica
- Pueden haber varios bloques de código asociados a la misma sección crítica
- También se dispone de una condición de acceso para acceder a la sección crítica
- Soporta pues tanto el uso competitivo como el cooperativo

Regiones CC (ejemplo)

```
program buffer_example
    type buffer_t is record
        slots: array(1..N) of character;
        size: integer range 0..N;
        head, tail: integer range 1..N;
    end record;

    buffer: buffer_t;
    resource buf: buffer;

    process producer;
        ...
        loop
            region buf when buffer.size < N do
                -- place char in buffer
            end region
            ...
        end loop;
    end

    process consumer;
        ...
        loop
            region buf when buffer.size > 0 do
                -- take char in buffer
            end region
            ...
        end loop;
    end
end
```


Monitores

- En las secciones críticas el código de una sección queda desperdigado
- Los monitores agrupan todos los bloques de código en una único elemento
- Las variables a las que se accede bajo exclusión mutua quedan ocultas
- Todos los procedimientos se ejecutan bajo exclusión mutua
- Pueden disponer de condiciones de acceso con las operaciones `wait` y `signal`

Semánticas de `wait` y `signal`

1. Una operación `signal` sólo se permite si es la última acción que realiza un proceso antes de salir del monitor
2. Una operación `signal` tiene como efecto secundario la ejecución de una instrucción `return` del procedimiento del monitor, es decir, saldría del procedimiento
3. Una operación `signal` que desbloquea otro proceso tiene como efecto secundario el bloquear al proceso llamante. La ejecución del proceso se reanudará sólo cuando se libere el monitor
4. Una operación `signal` que desbloquea otro proceso no lo desbloqueará hasta que salga del procedimiento actual y libere el monitor

Monitores (ejemplo)

```
monitor buffer;
    export append, take;
    var (* declaracion de las variables utilizadas *)

    procedure append (I: integer);
        ...
    end;

    procedure take (I: integer);
        ...
    end;

begin
    (* Inicializacion de las variables del monitor *)
end
```

Objetos protegidos

- Encapsulan unos datos y permite el acceso sólo a través de subprogramas protegidos
- Mejora el uso de las condiciones de acceso
- Ofrece exclusión mutua de escritura y de lectura
- Permite tres tipos de subprogramas
  - Procedimiento, con ejecución bajo exclusión mutua
  - Entradas, con condición de acceso
  - Funciones, permitiendo varias lecturas simultaneas

Objetos protegidos (ejemplo)

```
protected type Semaforo is
  function get_cont return integer;
  procedure set_cont(val: integer);
  entry wait;
  procedure signal;
private
  Counter : integer := 1;
end Semaforo;

protected body Semaforo is
  function get_cont return integer is
  begin
    return Counter;
  end get_cont;

  procedure set_cont(val: integer) is
  begin
    Counter := val;
  end set_cont;

  entry wait when Counter > 0 is
  begin
    Counter := Counter - 1;
  end wait;

  procedure signal is
  begin
    counter := counter + 1;
  end signal;
end Semaforo;
```

Evaluación de las barreras

- Ada optimiza la evaluación de las barreras para reducir la sobrecarga
- Una barrera se reevalúa solo en los siguientes casos:
  - Un proceso llama a una entrada de un objeto protegido y la barrera asociada hace referencia a una variable o atributo que puede haber cambiado desde que la barrera se evaluó por última vez
  - Una tarea abandona un procedimiento protegido o barrera protegida y hay tareas en la cola de espera de las entradas en las que se hace referencia a variables o atributos que pueden haber cambiado desde que la barrera se evaluó por última vez

Mecanismos de sincronización

- Señales transitorias y persistentes
  - suspend / resume
  - wait / wakeup
- Eventos
  - Son como banderas
  - Las tareas esperan un estado
- Buffers
  - Sincronización + información
- Pizarras
  - La lectura no borra el mensaje
  - Operación clear
- Broadcast
  - Lectura por todos los que esperan
- Multicast
  - Envío a un grupo
- Barreras
  - Bloquea a un grupo hasta que están todas bloqueadas

Eventos - Especificación

```
package Events is
    type Event_State is (Up, Down);

    protected type Event(Initial: Event_State := Down) is
        procedure Set;
        procedure Reset;
        procedure Toggle;
        functions State return Event_State;
        entry Wait(S: Event_State);
    private:
        entry Wait_Up(S: Event_State);
        entry Wait_Down(S: Event_State);
        Value: Event_State :I Initial;
    end Event;
end Events;
```


Eventos – Cuerpo (I)

```
 package body Events is
     protected body Event is
         procedure Set is
         begin
             Value := Up;
         end;

         procedire Reset is
         begin
             Value := Down;
         end;

         procedure Toggle is
         begin
             if Value = Down then
                 Value := Up;
             else
                 Value := Down;
             end if;
         end;

         functions State return Event_State is
         begin
             return Value;
         end;
```

Eventos – Cuerpo (II)

```
        entry Wait(S: Event_State) when True is
        begin
            if Value = Up then
                requeue Wait_Up with abort;
            else
                requeue Wait_Down with abort;
            end if;
        end;

        entry Wait_Up(S: Event_State) when Value = Up is
        begin
            null;
        end;

        entry Wait_Down(S: Event_State) when Value = Down is
        begin
            null;
        end;
    end Event;
end Events;
```


Comunicación basada en paso de mensajes

- Mecanismo alternativo a las variables compartidas
- Incorpora los mecanismos de sincronización
- Elementos que intervienen:
  - El modelo de sincronización
  - El método para indicar el destino
  - La estructura de los mensaje
- Espera selectiva
- Llamada a procedimientos remotos (RPC)

Sincronización entre procesos

- Existe una sincronización implícita
- Clasificación por el modelo de sincronización
  - Asíncrono
  - Síncrono
  - Invocación remota
- Con el modelo asíncrono se pueden construir los otros
- Inconvenientes del modelo asíncrono
  - Necesita buffers de tamaño infinito
  - Para confirmar la lectura se requiere otro mensaje
  - Se utilizan más envíos
  - Es más difícil depurar los errores

Indicación del destino

- Clasificación
  - Envío: Indicación del destino
    - Envío directo (a la tarea tarea)
    - Envío indirecto (a un buzón)
  - Lectura: Simetría
    - Envío simétrico (se indica de donde se lee)
    - Envío asimétrico (sin referencia a la fuente)
- El modelo asimétrico es adecuado para el modelo de comunicación cliente-servidor
- En la indicación indirecta pueden ser varios los que envían y los que leen

Usos con el envío indirecto

- Tipos de comunicación
  - De muchos a uno: Similar al modelo cliente-servidor
  - De muchos a muchos: Pueden leer varios procesos
  - De uno a uno: Comunicación entre dos procesos a través del buzón
  - De uno a muchos: Una petición se envía a un grupo de procesos
- Cuando los que leen son muchos puede ocurrir:
  - El mensaje lo lee solo un proceso (pool de servidores)
  - Todos los procesos reciben una copia (tolerancia a fallos)

Estructura de los mensajes

- En un lenguaje se debería permitir enviar cualquier objeto de datos
- Complicado si pueden haber punteros y estructuras dinámicas
- Algunos lenguajes restringen el envío a tipos simples o sin estructura
- Algunos lenguajes modernos eliminan esta restricción
- Si se usa SO los mensajes se suelen restringir a cadenas de bytes

Espera selectiva

- Hasta lo que hemos visto, el proceso receptor queda bloqueado hasta la recepción de un mensaje
- Esto es muy restrictivo. Puede interesar:
  - Leer simultáneamente de varios buzones
  - Leer un tipo de mensajes y dejar otros en espera
  - Variar los tipos de mensajes que se quieren leer
  - Dejar de leer al cabo de un tiempo
- Se resuelve con el mecanismo de la espera selectiva
  - En Ada se dispone de la construcción `select`
  - En las colas de de mensajes de UNIX se puede indicar el tipo (o tipos) de mensaje a leer
  - En los descriptores de UNIX se puede hacer una lectura con la operación `select`

Llamada a procedimientos remotos RPC

- Se construye con el paso de mensajes
- Es habitual en los sistemas distribuidos
- Es una extensión de la llamada a un procedimiento dentro de un proceso
- Usa el mecanismo de invocación remota
  - El mensaje que se envía incluye un identificador del procedimiento y sus parámetros
  - El mensaje de respuesta contiene el resultado de la función y los parámetros pasados por referencia