Programación de Sistemas de Control en Tiempo Real

Joaquín Aranda Almansa
Dpto. Informática y Automática

Planificación de sesiones

- 4 sesiones
- Veremos:
  - Conceptos generales de S.T.R.
  - Lenguajes de programación
  - Ingeniería del software aplicada a la programación de S.T.R.
  - Concurrencia y control de procesos
  - Sistemas operativos
  - Ejemplos

Bibliografía

- Apuntes suministrados de introducción a los "Sistemas de Tiempo Real"
- Phillip Laplante. "Real-Time Systems Design and Analysis". IEEE Press, 1992
- Stuart Bennett. "Real-Time Computer Control: an introduction". Prentice-Hall, 1994.
- Alan Burns & Andy Wellings. "Real-Time Systems and their Programing Languages". Addisson Wesley, 1990.

Primera sesión

- Definición de S.T.R. y algunos ejemplos
- Clasificación y estructuras
- Consideraciones sobre los lenguajes de programación
- Algunos lenguajes de programación

Definiciones y ejemplos

Concepto de Sistema de T.R. (1/3)

- Oxford Dictionary of Computting
  - » Un sistema en el cual el tiempo que tarda en producirse la salida es significativo. Esto es así debido a que la entrada corresponde a un cambio en el mundo físico y la salida está relacionada con ese mismo cambio. El lapsus de tiempo entre la entrada y la salida debe ser lo suficientemente pequeño para que la respuesta temporal del sistema sea aceptable.

Concepto de Sistema de T.R. (2/3)

- Young:
  - » Un procesamiento activo de información o sistema que tiene que responder a una entrada externa en un periodo de tiempo finito especificado.
- Cooling:
  - » Los sistemas de tiempo real producen una respuesta correcta en un tiempo límite definido. Si el computador excediera ese tiempo límite se produciría una degradación del sistema y/o malos resultados

Concepto de Sistema de T.R. (3/3)

- Otras definiciones:
  - » Un sistema de tiempo real lee entradas de una planta y envía señales de control a la misma en un tiempo determinado por las características operacionales de la planta.
- Programa en tiempo real
  - » Un programa en el cual las correctas operaciones dependen de los resultados lógicos computacionales y del tiempo que se tarda en producir estos resultados.

Ejemplos de S.T.R. (1/3)

- Sistema de navegación de un avión
  - Cada 5 milisegundos el softtware recibe pulsos de los acelerómetros x, y, z. Conversores A/D proporcionan al software los angulos de alabeo, cabeceo y guiñada, indicando la orientación del avión cada 40 milisegundos. Cada segundo se recibe información de la temperatura. La tarea del software es calcular el vector de velocidad real basado en las lecturas de orientación y aceleración y con distintos factores de compensación (efecto de la temperatura) cada 40 milisegundos.
  - ¿Como se puede crear un programa con las cuatro diferentes velocidades de ejecución del sistema de navegación y como comunicar estos programas?

Ejemplos de S.T.R. (2/3)

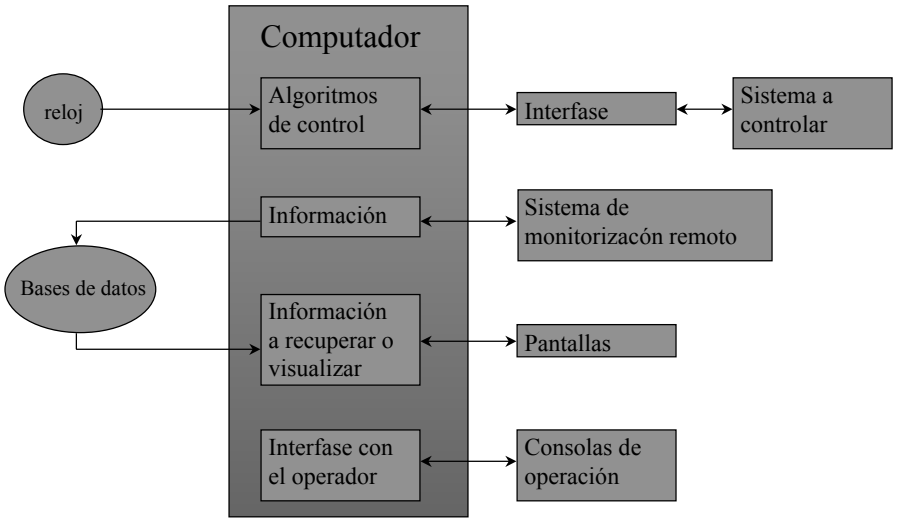
- Monitorización de una planta nuclear
  - Se manipulan dos sucesos señalados mediante interrupciones (por simplicidad).
  - El primer suceso es desencadenado por cualquiera de las señales de alguno de los puntos de seguridad que indican una brecha en la seguridad. El sistema debe responder a esta señal en 1 segundo.
  - El otro suceso indica que el núcleo nuclear alcanza una sobre-temperatura. Esta señal se debe atender en 1 milisegundo.
  - En este sistema se necesita un mecanismo que asegure que el indicador de "fusión inminente del reactor" puede interrumpir a cualquier otro proceso.

Ejemplos de S.T.R. (3/3)

- Sistema de reserva de billetes
  - Se decide que para prevenir retrasos y descontentos de los clientes el tiempo para cualquier transición debe ser menor a 15 segundos sin permitir que se produzca una sobre-reserva.
  - En cualquier instante pueden varios agentes intentar acceder a la base de datos y reservar simultáneamente el mismo vuelo.
  - Es necesario disponer de mecanismos de bloqueo de registros y de comunicación.
  - ¿Como satisfacer todo?

Clasificación y estructura

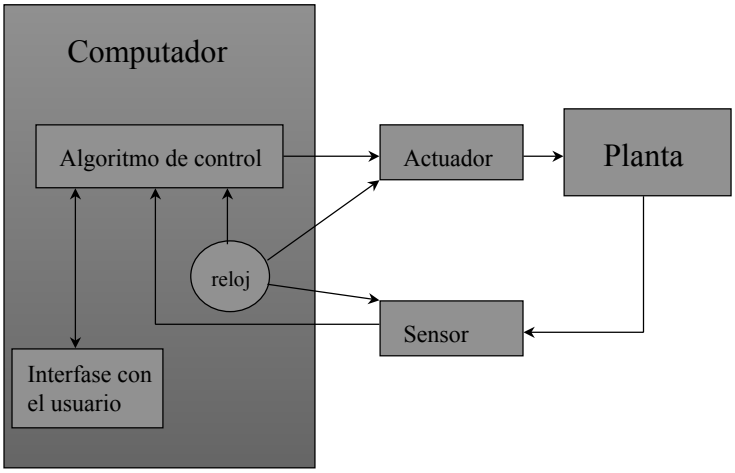
Estructura de un S.T.R.



Clasificación de los S.T.R.

- Sistemas basados en reloj
  - » La sincronización se hace en función del tiempo
- Sistemas basados en sucesos (aperiódicos)
  - » La sincronización se hace en función de sucesos que ocurren, como cierre de una válvula.
- Sistemas interactivos
  - » La relación entre acciones computador y evolución del sistema es más amplia que las anteriores.

Sistemas basados en reloj



Sistemas basados en sucesos

- Ejemplos de acciones en respuestas a sucesos:
  - » Cierre de una válvula cuando se alcanza un nivel.
  - » Parada de un motor al alcanzar una posición.
  - » Indicación de alarma.
- Generalmente se incluyen los tiempos máximos en los que se debe responder a los sucesos.

Sistemas interactivos

- Ejemplos: cajeros automáticos, sistemas de reserva de billetes, etc.
- La especificación de tiempo se expresa en términos de un tiempo de respuesta que no sea superior a un cierto valor.
- La respuesta en un determinado instante depende del estado interno del computador.

Clasificación de los S.T.R. según las restrinciones temporales

- Tiempo real estricto:
  - » Sistemas en los que se deben satisfacer las limitaciones temporales en todas y cada una de las ocasiones
- Tiempo real en sentido amplio
  - » Sistemas en los que un fallo ocasional en la limitación temporal no supone que el sistema no funcione correctamente.

Clasificación de los programas

- Secuencial
- Multitarea
- Tiempo real

Programas secuenciales

- Las acciones a realizar se ordenan de forma consecutiva
- La validación solo requiere que:
  - Cada sentencia defina una acción establecida
  - Las distintas estructuras del programa produzca una secuencia de sucesos establecidos

Programación multitarea

- Las acciones no son necesariamente disjuntas en el tiempo (aunque estan formados por tareas secuenciales)
- La validación, si no hay variables compartidas, se hace para cada tarea

Programación de tiempo real

- Difiere de los otros en que la secuencia de algunas acciones no se determina por el diseñador sino por los sucesos que ocurran en el entorno.

Estructura de los programas de T.R. (1/5)

- Una caracterísitica común a todos los programas de T.R. es que se tienen que ejcutar continuamente, por lo que la estructura natural es un lazo infinito:

```
PROGRAM ControlTR
BEGIN
   WHILE NOT ElFinDel Mundo DO
      BEGIN
      Tarea de control
      END
END
```


Estructura de los programas de T.R. (2/5)

- La "Tarea de control" se tiene que sincronizar con el mundo exterior. Una forma de hacerlo es mediante polling (sondeo):
  - Si el suceso no ha ocurrido el programa continua con otras acciones antes de volver a comprobar otra vez si se ha producido.
  - El programa de forma continua repite la comprobación hasta que ocurre el suceso.
- Ventaja: se puede considerar que el programa realiza una única tarea.
- Inconveniente: Se debe asegurar que todas las tareas se puedan realizar en un tiempo especificado.

Estructura de los programas de T.R. (3/5)

- Un programa con tres tareas (que se pueden considerar una sola):
  - Supervisar un teclado
  - Efectuar acciones de control
  - Actualizar un display

```
PROCEDURE Control
BEGIN
    IF KeyFlag THEN Keyboard
    IF ControlFlag THEN Control
    IF DisplayFlag THEN Display
END
```


Estructura de los programas de T.R. (4/5)

- Sincronización a unos instantes determinados de tiempo:

```
PROCEDURE Control
BEGIN
  LOOP
    WHILE NOT Digin(TiempoDeMuestreo) DO
          Esperar a que ocurra la señal de tiempo
    END
    Acción de control
  END
END
```

Estructura de los programas de T.R. (5/5)

- Si se tiene un reloj que produce interrupciones periódicas

```
PROCEDURE Control
BEGIN
  REPEAT
   Time=GetTime
  UNTIL Time>SiguientePeriodo
  DO Control
   Acciones de control
  NEXT SiguientePeriodo=Time+IntervaloMuestreo
END
```


Consideraciones sobre los Lenguajes para Sistemas de Tiempo Real

Cualidades de un Lenguaje para S.T.R.

- Seguridad
- Entendibilidad
- Flexibilidad
- Simplicidad
- Portabilidad
- Eficiencia

Seguridad

Se mide en términos de la efectividad en la detección de errores

- Soporte para la modularidad
- Forzar la declaración de variables
- Un rango de tipos de datos adecuado, incluyendo tipos sub-rangos
- Sintaxis no ambigua

Entendibilidad

Es una medida de lo fácil que es entender la operación de un programa sin tener que recurrir a documentación externa.

Beneficios:

- Reducción en el coste de documentación.
- Una detección de errores más fácil.
- Un mantenimiento más fácil.

Flexibilidad

Un lenguaje debe proporcionar todas las características necesarias para expresar todas las operaciones que se requieren sin tener que utilizar complicadas construcciones o tener que insertar código a nivel ensamblador.

Simplicidad

- La simplicidad contribuye a la seguridad
- Reduce el coste y los errores

Asociado con la simplicidad está la consistencia: no se deben imponer restrinciones arbitrarias en la utilización de ninguna característica del lenguaje

Portabilidad

Es deseable como un medio de acelerar el desarrollo, reducir el coste y aumentar la seguridad.

Es dificil de conseguir.

Eficiencia

En los sistemas de tiempo real se deben garantizar un cierto funcionamiento junto con unas restrinciones temporales.

- Velocidad
- Compactación

- Seguridad
- Coste de escritura y mantenimiento

Características de los lenguajes S.T.R.

- Mecanismos versátiles de paso de parámetros
- Una fuerte definición de tipos de variables
- Manipulación de excepciones
- Tipos de interrupciones
- Modularidad

Paso de parámetros

- Por valor
- Por referencia
- Variables globales

- Otros: por nombre y por resultado del valor

  (estos son particulares de ALGOL-60 y ALGOL-W)

Paso de parámetros por valor

El valor del parámetro es copiado en el parámetro formal del procedimiento

```
int abs(int x)
{
  if (x<00)
   return(-x);
  else
   return(x);
}
```

Paso de parámetros por referencia

La dirección del parámetro es pasada de la rutina de llamada al procedimiento invocado

Un ejemplo en FORTRAN al pasar parámetros por referencia

```
SUBROUTINE PROMEDIO(PRO,X,Y)
REAL PRO, X, Y
PRO=(X+Y)/2
RETURN
END

PROGRAM TEST
REAL X,Y
X=4.0
Y=3.0
CALL PRO(4,X,Y)
END
```


Variables globales versus paso de parámetros (1/2)

Variables globales

- Se ven desde todos los módulos
- Las referencias se pueden hacer mas directa y rápidamente

Paso de parámetros

- El interface entre módulos esta claramente definido
- Pero la lista de parámetros puede ser larga

Variables globales versus paso de parámetros (2/2)

- Ejemplo de problema en el paso de parámetros:

```
procedure funcion(a,b,c,d:integer,var o:integer);

……

call funcion(x,y,z,q,o)
```

- Se puede dar una situación en la que solo parte de parámetros estén copiados en la pila y sea interrumpido el proceso.

Recursividad (1/2)

- Un procedimiento se puede invocar a sí mismo

```
procedure mcd(x,y:integer);
begin
  if (y=0) then
   writeln(x)
  else
   mcd(y,(x mod y))
end;
```

Recursividad (2/2)

Lenguajes que permiten recursividad deben soportar procedimientos re-entrantes.

Un procedimiento re-entrante es aquel que puede ser utilizado por varias tareas que se ejecutan concurrentemente en un sistema multitarea.

Asignación dinámica

- Importante para construir y mantener las pilas.
- Estructuras de datos dinámicas (listas, arboles) se benefician de la claridad y economía de memoria a costa de velocidad.

Tipos de variables (1/3)

- Cada variable y constante debe de ser de un tipo (entero, real, carácter, …).
- Se prohibe que se mezclen diferentes tipos en las operaciones y asignaciones.

Tipos de variables (2/3)

Ventajas de una estricta utilización de tipos

- Fuerzan al programador a ser preciso sobre la forma de los datos.
- Forma de prevenir truncaciones o redondeos no queridos.

Ejemplo de un problema en FORTRAN:

```
      DO 20 I=1.5
      …
20    CONTINUE
```

Tipos de variables (3/3)

Inconvenientes de una débil utilización de tipos

- Si la declaración de tipo no es estricta se pueden dar problemas causados por truncaciones o redondeos.

Ejemplo de promoción de tipo en C:

```
int i,j;
float k,l,m;
…
j=i*k+m
```


Manipulación de excepciones

- Excepciones: errores y situaciones anómalas que pueden ocurrir en la ejecución de un programa.
- Ada tiene muchas facilidades de manipulación de excepciones.
- En C, Pascal y Modula-2 se pueden implementar como librerías.

Tipos de datos abstractos (1/2)

- Posibilidad de representar y manipular tipos de datos que no son tipos estandares soportados por el lenguaje.
- Tipos numerados en Ada

```
type DIA is (LUNES, MARTES, MIERCOLES, JUEVES,
  VIERNES, SABADO, DOMINGO);
type VOLTAJE is delta 0.01 range 0.0 .. 5.0;
type COLOR is ();
```

- Definición de tipo en C

```
typedef char cadena[16];
```


Tipos de datos abstractos (2/2)

Inconveniente:

- Solo permiten el agrupamiento de clases similares de datos
- Pero en Ada, C, Pascal y Modula-2 se proporciona un mecanismo para organizar un conjunto de datos diferentes en un único tipo de dato.

```
Empleado= RECORD
                Nombre : Array [0..25] of CHART;
                Apellido : Array [0..25] of CHART;
                NumSeg : CARDINAL;
          END
```

Programación orientada a objeto

- Lenguajes: C++, Smalltalk interpretado, Ada++, DRAGOON.
- Los L.O.O. proporcionan características para buenas técnicas de ingeniería del software.
- *Polimorfismo, herencia*
- Desventaja: mayor tiempo de ejecución.

Modularidad

- Desde la perspectiva de ingeniería del software: es deseable tener programas altamente modulares.

Algunos lenguajes de programación

BASIC

- Mediados de los 60 en el col. Dartmouth.
- Interpretado no se suele utilizar en S.T.R.
- Tiene débil declaraciones de tipo
- Pocas facilidades para el diseño modular
- No hay mecanismos de paso de variables
- Todas las variables son globales

FORTRAN

- Data de 1957.
- Primeras versiones no aceptan recursión. En posteriores aceptan código re-entrante.
- En los primeros S.T.R. había mucho código en ensamblador.
- No es estricto en la definición de tipos.
- Se utiliza para S.T.R. científicos, aeronáuticos y aplicaciones de control de procesos
- Muchas versiones: FORTRAN IV, FORTRAN 77, ISO FORTRAN 90, ...

C

- Data de 1971
- Manipula objetos específicos de la máquina (bytes, bits, direcciones,…)
- Dispone de variables globales y paso de parámetros por valor.
- Paso por referencia se simula mediante los tipos puntero.
- Es modular y recursivo

C: tipos de variables especiales

- Registros: indica que será utilizada frecuentemente
- Volátiles (VOLATILE): no son optimizadas por el compilador
- Estáticas (STATIC): pueden proporcionar comunicación privada entre módulos
- Constantes (CONSTANT)

C: ejemplo implementación pila

```
#define MAXIMIZE 1000 /*tamaño maximo de la pila */
static int top;
static char pila[MAXSIZE]   /*se define la pila*/
void int()
{
   top=0;   /* se pone a 0 el puntero cabeza pila*/
}
int pop(void)
{
   int temp;
   temp=pila[top];  /*elimina los datos de la pila*/
   top--;     /*decrementa el puntero de la cabeza*/
   return(temp);
}
void push(int dato)
{
   top++;      /*incrementa el puntero a la cabeza*/
   pila(top=dato);   (/*pone el dato en la pila
}
```

C: manipulación de excepciones

- Se hace mediante señales.
- Además se tiene:
  - » *setjmp:* es una macro que guarda la información del entorno
  - » *longjmp*: función de librería que vuelve a situar el programa en el estado que tenía cuando se llamó a *setjmp*.

C: ejemplo manipulación de excepciones

```
#include <signal.h>             /*donde estan setjmp y longjmp*/
typedef long jmp_buf[16];            /*define tipo de entorno*/
jmp_buf env;                    /*donde se retiene el entorno*/
main()
{
   …                         /*se realiza algún procedimiento*/
   if (setjmp(env)==0)      /*vuelta normal o desde longjmp?*/
        process();          /*procesamiento normal*/
   else
        abort_process()    /*se vuelve con error de longjmp*/
}
void process()
{
   int valor;             /*devuelve el valor a setjmp */
   …                      /*se realiza algún procesamiento*/
   if (no_error)
        …                 /*procesamiento normal sin error*/
   else
        logjmp(env,valor);        /*error, se aborta*/
}
```

C: inconvenientes y ventajas

- No es estricto en la definición de tipos.
- Poco estandar. Estandarización con el libro de Kernighan y Ritchie (1990) y el ANSI-C.
- Es bueno para T.R. ya que proporciona estructuración y flexibilidad sin complejas restricciones del lenguaje.

PASCAL (1/2)

- Diseñado por Niklaus Wirth en 1968
- Primer compilador en 1971
- Estándar ANSI/IEEE, aunque diversos dialectos
- Estricta definición de tipos
- Recursividad
- Estructura de datos dinámicas
- Datos abstractos mediante tipos enumerados
- Alto grado de modularidad

PASCAL (2/2)

Argumentos en contra expuestos por Kernighan

- Difícil manipulación cadenas
- Carencia de variables estáticas, de inicialización y forma de comunicación no jerárquica, destrozan la "localidad"
- Los procedimientos y funciones se presentan en un orden no natural
- Carencia de compilación separada, impide grandes aplicaciones y difícil utilización de librerías
- No se controla el orden de evaluación de expresiones lógicas
- Sentencia CASE incompleta, falta cláusula por defecto
- Carece de herramientas para ensamblar grandes programas, falta la inclusión de archivos

MODULA-2

- En 1975, N.Witrh, desarrolla Modula
- En 1977 desarrolla Modula-2 (entre Pascal y Modula)
- Modula-2 incluye los aspectos de Pascal y extiende el lenguaje con el concepto de módulo o paquete.

Otras características:

- Concepto de proceso o hebra de ejecución
- Capacidades a bajo nivel
- Procedimiento que hace asignar variables dinámicamente
- Interfaces definidas claramente entre módulos
- Se eliminan muchos de los inconvenientes de Pascal
- Adecuado para S.T.R. debido a su orientación multitarea

ADA (1/5)

- Diseñado para aplicaciones de tiempo real
- Es un lenguaje de consenso
- Elegido por el DoD (Depart. de Defensa USA)

Consideraciones en el desarrollo:

- Fiabilidad y mantenimiento del programa
- Programación como actividad humana
- Eficiencia del programa

ADA (2/5)

Fiabilidad y mantenimiento del programa

- Verificación de tipo en tiempo de compilación
- Módulos compilables separadamente
- Manipulación de excepciones explícita
- Verificación estricta del tipo y prevención de conversiones de tipo

ADA (3/5)

Programación como actividad humana

- Tipos de datos abstractos
- Mas primordial la entendibilidad del programa que la facilidad de construcción
- Intento de sintaxis compacta (pero no se logró)

ADA (4/5)

Eficiencia del programa

- Poco logrado
- Los primeros compiladores generaban código pobre (en términos de tiempo de ejecución)
- El modelo multitarea/multiprogramación hacen prever una operación eficiente

ADA (5/5)

Resumen

- Definición estricta de tipos
- Asignación dinámica
- Recursividad
- Capacidad explícita de manipulación de excepciones
- Compilación por separado
- Adecuado para S.T.R.
- Poco eficiente en algunas aplicaciones

ENSAMBLADOR

Ventaja:

- Control directo del hardware
- Algunos programadores realizan código más eficiente que el de algunos compiladores

Inconvenientes

- No estructurado
- Grandes diferencias de una máquina a otra: no portable
- Difícil y tedioso
- Propenso a errores

Generación de código

Tan importante como el código escrito por el programador es el generado por el compilador

Optimizar salida del compilador:

velocidad, utilización memoira y registros, saltos

Los programadores deben conocer el compilador que utilizan

Segunda sesión

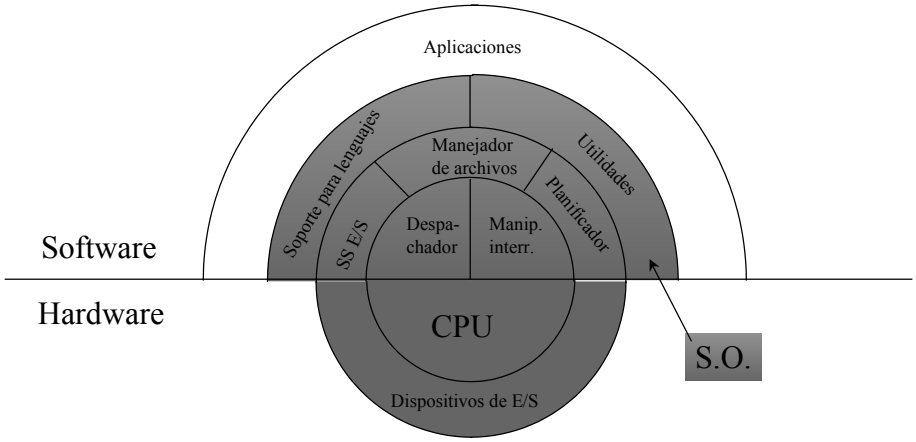
- Sistemas Operativos
- Programación concurrente

Sistemas Operativos

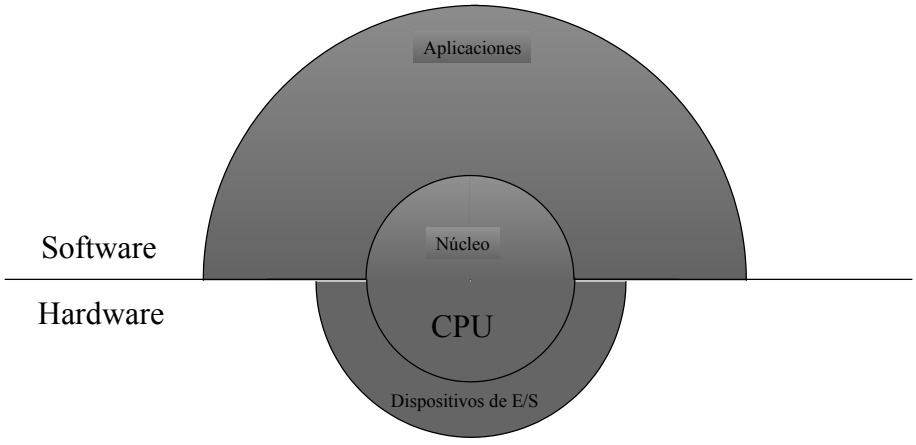
Sistemas Operativos

- Un sistema operativo convierte el hardware del sistema en una máquina virtual con unas características definidas por el sistema operativo.
- El acceso al hardware y a los dispositivos de E/S es a través del sistema operativo.

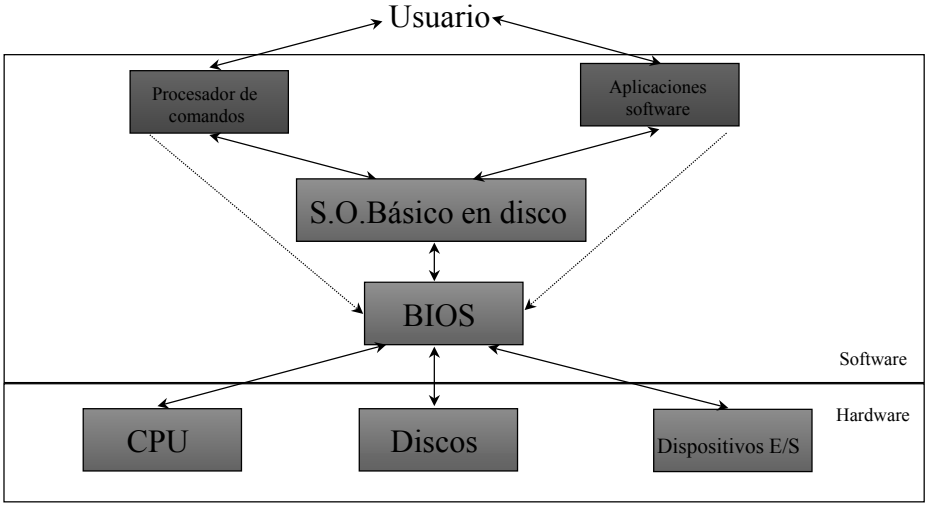
Sistema Operativo de propósito general



Sistema Operativo mínimo



Estructura de un Sistema Operativo

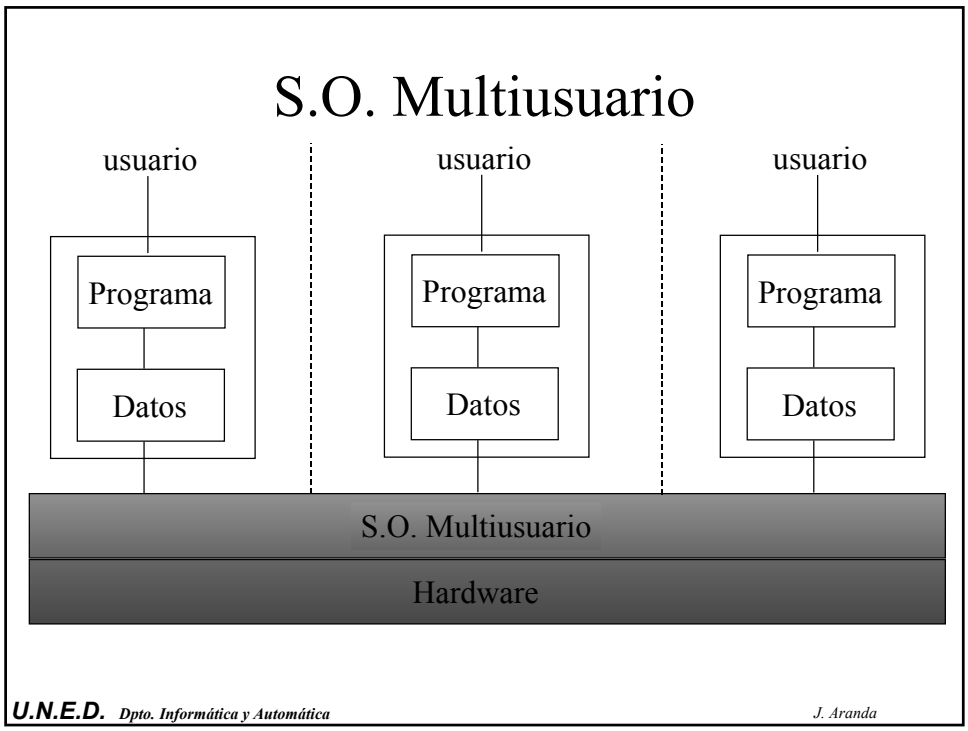


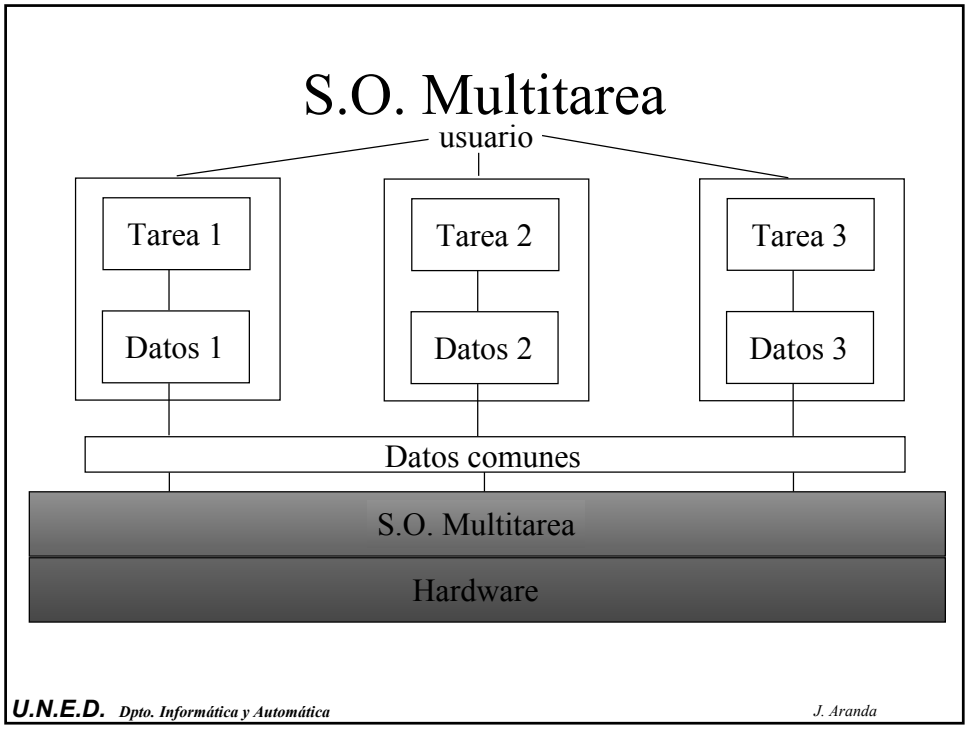
Tipos de S.O.

Poca diferencia entre S.O. de tiempo real y otros.

La confusión está entre:

- Multiprogramación o multiusuario
- Multitarea


S.O. Multiusuario
usuario
usuario
usuario
Programa
Programa
Programa
Datos
Datos
Datos
S.O. Multiusuario
Hardware
U.N.E.D. Dpto. Informática y Automática
J. Aranda

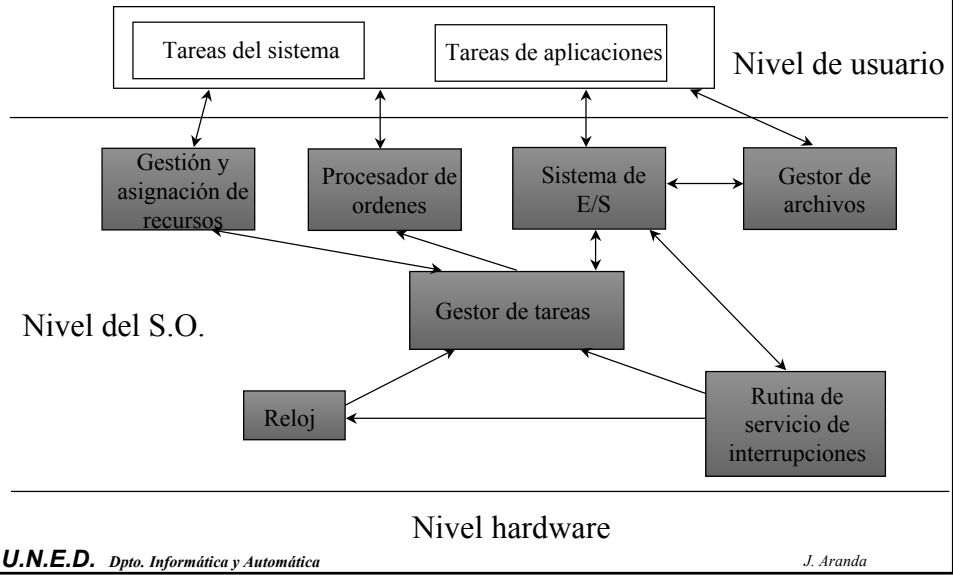

S.O. Multitarea
usuario
Tarea 1
Tarea 2
Tarea 3
Datos 1
Datos 2
Datos 3
Datos comunes
S.O. Multitarea
Hardware
U.N.E.D. Dpto. Informática y Automática
J. Aranda

S.O. multitarea de tiempo real (1/2)

Tiene que soportar la compartición de recursos y requerimientos temporales de las tareas y funciones.

- **Gestor de tareas**: asignación de memoria y planificación de tareas.
- **Gestor de memoria**: controla la asignación de memoria.
- **Control de recursos**: Control de todos los recursos compartidos, además de la memoria y el tiempo de CPU.
- **Comunicación y sincronización entre tareas**: Proporciona mecanismo seguros de comunicación y sincronismo

S.O. multitarea de tiempo real (2/2)



Estrategias de planificación (1/3)

Planificación de tiempo de asignación de una única CPU

- Cíclica

  Cada tarea utiliza la CPU el tiempo que necesite. Cuando no requiere más la CPU el planificador la asigna a la siguiente tarea en la lista

- Pre-desocupada

  Hay muchas estrategias, todas utilizan la posibilidad de interrupciones.

Estrategias de planificación (2/3)

Criterios para la planificación

- Eficacia: Porcentaje de tiempo medio de utilización
- Rendimiento: Número de procesos completado por unidad de tiempo
- Tiempo de regreso: Intervalo que transcurre desde que un proceso se crea o presenta hasta que se completa por el sistema
- Tiempo de espera: Tiempo que el proceso espera hasta que se le concede un procesador
- Tiempo de respuesta a un suceso

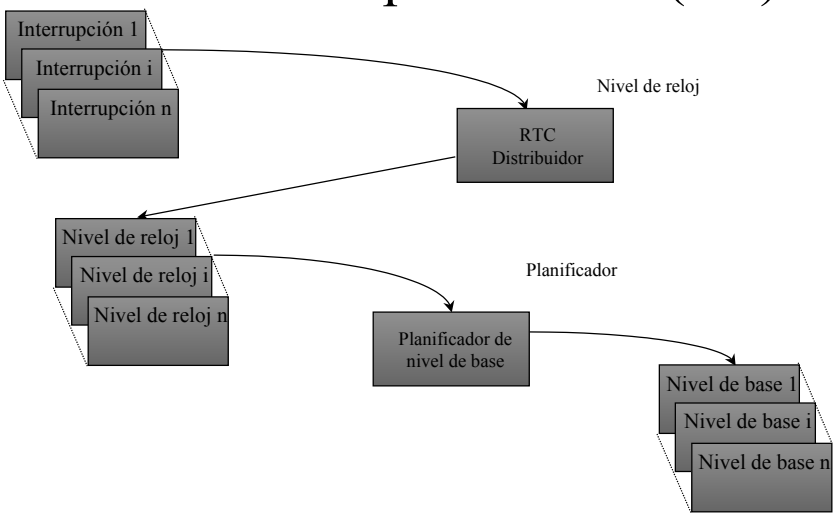
Estrategias de planificación (3/3)

- Por expropiación
- FCFS: primero en llegar primero en ser servido
- SJF: primera la tarea más corta
- SRT: tiempo que queda más corto
- RR: prioridad circular

Planificación para S.T.R. (1/3)

- Tareas cíclicas
- Tareas que se tienen que hacer en un instante determinado: Guiadas por sucesos
  - Cierre de una válvula
  - Parada de un motor
  - Una alarma

Planificación para S.T.R. (2/3)



Planificación para S.T.R. (3/3)

Las tareas se suelen dividir en tres amplios niveles de prioridad

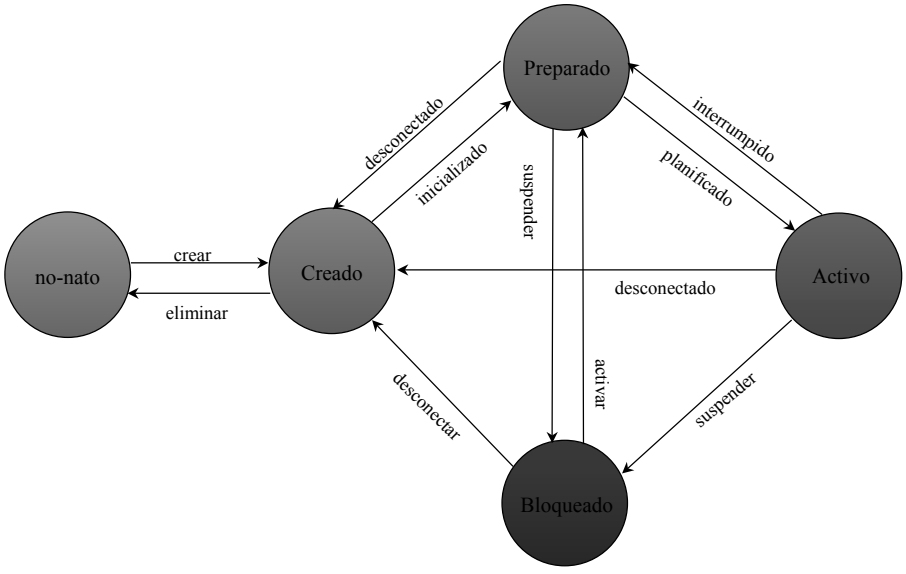
- **Nivel de interrupción**: Rutinas de servicio de las tareas y dispositivos que requieren una respuesta muy rápida (reloj de tiempo real y el distribuidor de nivel de reloj).
- **Nivel de reloj**: Tareas que requieren un procesamiento repetitivo (tomas de datos de un sistema de control) y las que requieren una sincronización muy precisa. La tarea de menor prioridad es el planificador de nivel de base.
- **Nivel de base**: Tareas que no tienen un tiempo límite para realizarse.

Administración de tareas

Funciones básicas de la administración de tareas:

- Mantener un registro del estado de cada tarea
- Planificar la asignación de tiempo de la CPU para cada tarea.
- Realizar la conmutación de contexto.

Administración de tareas: estados

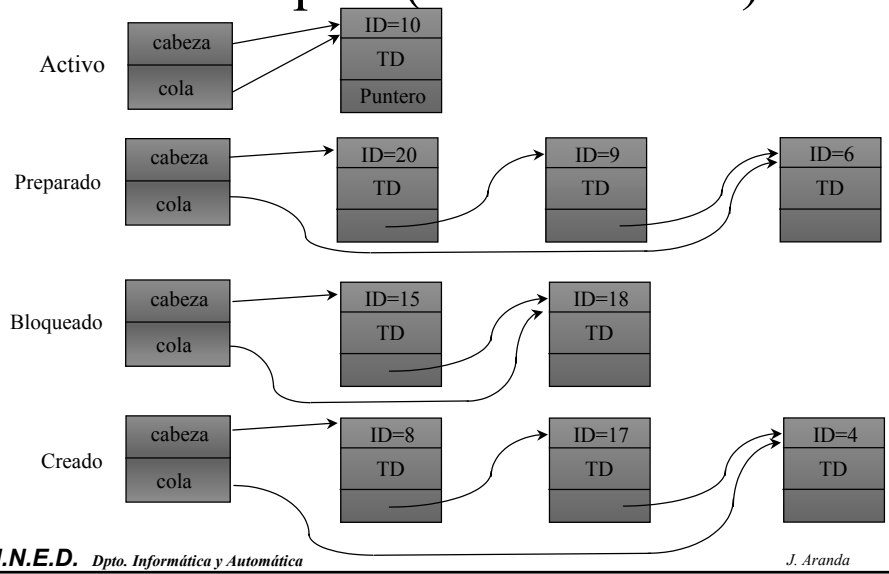


Administración de tareas: descriptor

La información sobre el estado de cada tarea se tiene en el **descriptor de tareas** (TD):

- Identificador de la tarea (ID)
- Prioridad de la tarea (P)
- Estado actual de la tarea
- Area para almacenar el entorno volátil (o un puntero a la misma)
- Puntero a la siguiente tarea en la lista

Administración de tareas: descriptor (lista de tareas)

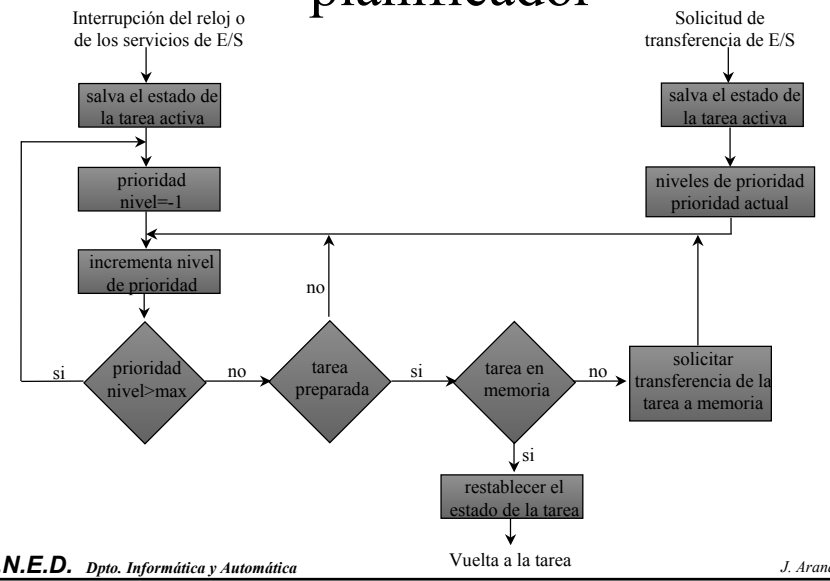


Administración de tareas: planificador

Se entra en el planificador/despachador bajo dos condiciones:

- Debido a una interrupción del reloj o a otra interrupción que señale que se completó una solicitud de entrada/salida
- La suspensión de una tarea debido a un retardo de la misma, completando o solicitando una transferencia de entrada/salida

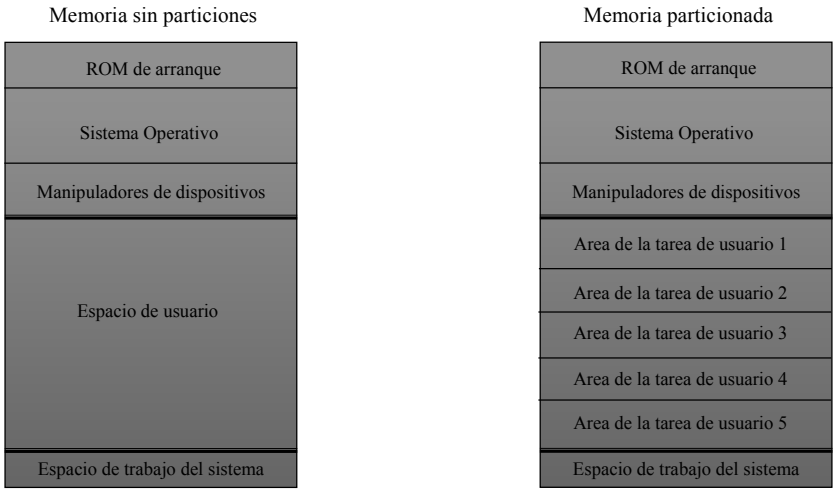
Administración de tareas: planificador



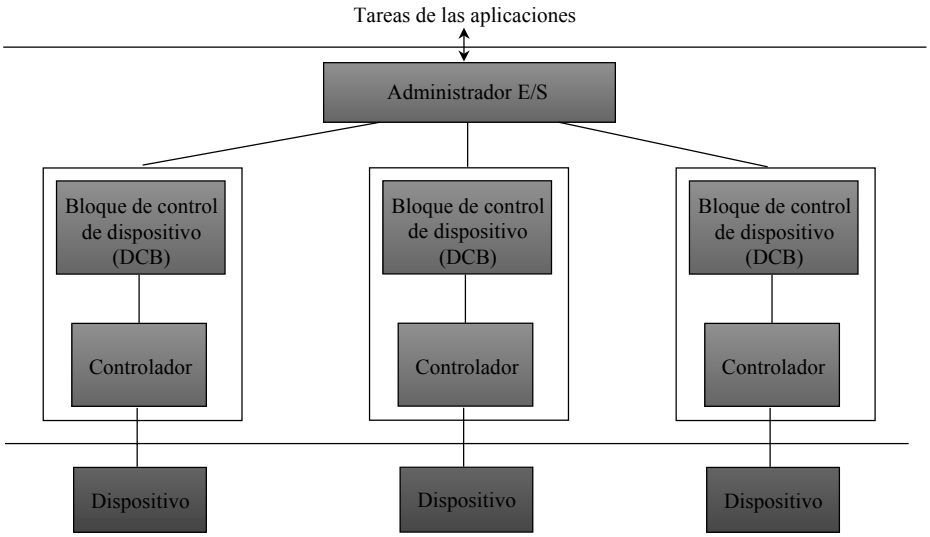
Administración de la memoria (1/2)

- En aplicaciones de control el software es estático, no se crea y elimina dinámicamente.
- La administración de memoria es más sencilla que para multi-programación.
- Se pueden usar particiones estáticas.

Administración de la memoria (2/2)



Subsistema de Entrada/Salida



Programación concurrente

Programación concurrente

Se conoce por programación concurrente a la rama de la informática que trata de las notaciones y técnicas de programación que se usan para expresar el paralelismo potencial entre tareas y para resolver los problemas de comunicación y sincronización entre procesos

Exclusión mutua: región crítica (1/4)

Ejemplo: variable x compartida entre los procesos A y B

Problema de los Jardines:

Se quiere tener constancia del número de visitantes que hay en un Jardín.

Hay dos puertas y asociado a cada puerta un proceso.

Cuando entra una persona el proceso P1 incrementa x: x:=x+1

Cuando sale una persona el proceso P2 decrementa x: x:=x-1

Exclusión mutua: región crítica (2/4)

Instrucciones para la actualización de la variable:

- copiar el valor de x en un registro del procesador
- incrementar el valor del registro
- almacenar el resultado en la dirección donde se guarda x

En esta situación se puede producir una mala actualización de la variable x.

Exclusión mutua: región crítica (3/4)

- P1 carga el valor de x en un registro de su procesador
- P2 carga el valor de x en un registro de su procesador
- P2 decrementa el valor de su registro
- P1 incrementa el valor de su registro
- P1 almacena el valor de su registro en la dirección de memoria de x
- P2 almacena el valor de su registro en la dirección de memoria de x

Exclusión mutua: región crítica (4/4)

- Se denomina **Sección crítica** a aquellas partes de los procesos concurrentes que no pueden ejecutarse de forma concurrente o, también, que desde otro proceso se ven como si fueran una única instrucción.
- Las secciones críticas se agrupan en clases, siendo **mutuamente exclusivas** las regiones críticas de cada clase.
- La exclusión se consigue con protocolos que **bloqueen** el acceso a la sección crítica mientras está siendo utilizada por un proceso.

Bloqueo mediante el uso de variables compartidas (1/4)

```
module ExclusionMutua1
var flag:boolean;
process P1
begin
  loop
       while flag=true do
       (*Espera a que quede libre el dispositivo*)
       end;
       flag:=true;
      (*uso del recurso. Sección crítica*)
      flag:=false;
      (* resto del proceso*)
 end
end P1;
```

```
process P2
begin
 loop
       while flag=true do
       (*Espera a que quede libre el dispositivo*)
       end;
       flag:=true;
      (*uso del recurso. Sección crítica*)
      flag:=false;
      (* resto del proceso*)
 end
end P2;
```

```
begin
  flag:=false
  cobegin
       P1;
       P2;
  coend
end
```

Este programa no resuelve el problema de la exclusión mutua, ya que la comprobación y la puesta del indicador son operaciones separadas y en ese caso se puede entrelazar el uso del recurso por ambos procesos

Bloqueo mediante el uso de variables compartidas (2/4)

```
module ExclusionMutua2
var flag1,flag2:boolean;
procedure bloqueo(var mi_flag,su_flag:boolean);
begin
  mi_flag:=true; (*intención de usar el recurso*)
  while su_flag do; end (*espera liberar recurso*)
end bloqueo;

procedure desbloqueo(var mi_flag:boolean);
begin
  mi_flag:=  false;
end desbloqueo;
```

```
process P1
begin
  loop
       bloqueo(flag1,flag2);
       (*uso del recurso. Sección crítica*)
       desbloqueo(flag1);
       (* resto del proceso*)
 end
end P1;

process P2
begin
 loop
       bloqueo(flag2,flag1);
       (*uso del recurso. Sección crítica*)
       desbloqueo(flag2);
       (* resto del proceso*)
 end
end P2;
```

```
begin
  flag1:=false
  flag2:=false
  cobegin
       P1;
       P2;
  coend
end
```


Bloqueo mediante el uso de variables compartidas (3/4)

- Un problema con la solución anterior es que se puede producir el **INTERBLOQUEO**: Los dos procesos llaman al "bloqueo" simultáneamente. Los dos indicadores quedan activados y los dos procesos a la espera de que se libere el recurso. Pero esto no se puede dar ya que ninguno entra en la sección crítica.

Bloqueo mediante el uso de variables compartidas (4/4)

- Una solución al interbloqueo

```
procedure bloqueo(var mi_flag,su_flag:boolean);
begin
    mi_flag:=true;
    while su_flag:=false;
        mi_flag:=false;
        mi_flag:=true;
    end
end bloqueo
```

- Problema: Puede que un proceso deje su sección crítica y vuelva a entrar mientras que el otro proceso desactiva su indicador en la sección de bloqueo. El que un proceso no pueda progresar porque se lo impida otro se denomina **CIERRE** (lockout).

Algoritmo de PETERSON

```
module ExclusionMutua_Peterson
var flag1,flag2:boolean;
    turno:integer;
procedure bloqueo(var
     mi_flag,su_flag:boolean,su_turno:integer);
begin
  mi_flag:=true;
  turno:=su_turno;
  while su_flag and (turno=su_turno) do; end
end bloqueo;

procedure desbloqueo(var mi_flag:boolean);
begin
  mi_flag:=  false;
end desbloqueo;
```

```
process P1
begin
  loop
       bloqueo(flag1,flag2,2);
      (*uso del recurso. Sección crítica*)
      desbloqueo(flag1);
      (* resto del proceso*)
 end
end P1;

process P2
begin
 loop
      bloqueo(flag2,flag1,1);
      (*uso del recurso. Sección crítica*)
      desbloqueo(flag2);
      (* resto del proceso*)
 end
end P2;
```

```
begin
  flag1:=false;
  flag2:=false;
  cobegin
      P1;
      P2;
  coend
end
```

Algoritmo de DEKKER

```
module ExclusionMutua_Dekker
var flag1,flag2:boolean;
    turno:integer;
procedure bloqueo(var
     mi_flag,su_flag:boolean,su_turno:integer);
begin
  mi_flag:=true;
  while su_flag do (*otro proceso en la sec. cr.*)
    if turno=su_turno then mi_flag:=false;
    while turno=su_turno do
       (*espera a que el otro termine*)  end
    mi_flag:=true;
   end
  end
end bloqueo;
procedure desbloqueo(var
     mi_flag:boolean;su_turno:integer);
begin
  turno:=su_turno;
  mi_flag:=  false;
end desbloqueo;
```

```
process P1
begin
  loop
      bloqueo(flag1,flag2,2);
      (*uso del recurso. Sección crítica*)
      desbloqueo(flag1);
      (* resto del proceso*)
 end
end P1;

process P2
begin
 loop
      bloqueo(flag2,flag1,1);
      (*uso del recurso. Sección crítica*)
      desbloqueo(flag2);
      (* resto del proceso*)
 end
end P2;
```

```
begin
  flag1:=false;
  flag2:=false;
  turno:=1;
  cobegin
      P1;
      P2;
  coend
end
```


SEMÁFOROS

- Los semáforos binarios fueron introducidos por Dijkstra en 1968.
- Un semáforo binario es un indicador (S) de condición que registra si un recurso está disponible o no.
- Toma dos valores:
  - 0 (no está diposnible) y
  - 1 (está disponible)

Operaciones sobre Semáforos

- Inicializar
  - inicializa (S:SemaforoBinario; v:ineger);
    Pone el valor del semáforo al valor de v (0 o 1)
- Espera (wait o P)
  - espera (S)

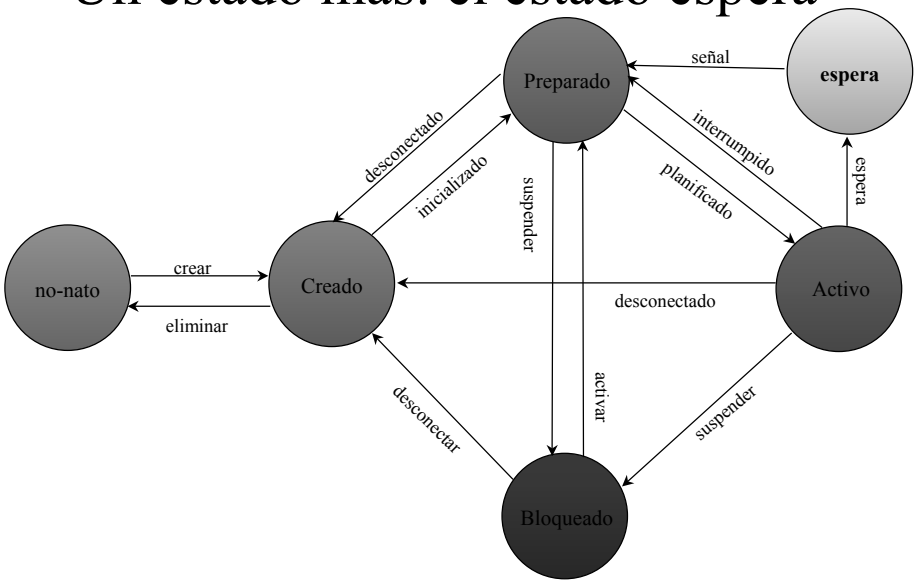
    if S=1 then S:=0

    else suspender la tarea que hace la llamada y ponerla en la cola de tareas.
- Señal (signal o V)
  - señal (S)

    if la cola de tareas está vacía then S:=1

    else reanudar la primera tarea de la cola de tareas

Un estado más: el estado espera



Exclusión mutua con semáforos

- Se usa "espera" como procedimiento de bloqueo antes de la sección crítica.
- Se usa "señal" como desbloqueo

```
process P1
begin
    loop
      espera(S);
      Sección Crítica
      señal(S);
      (*resto del proceso*)
    end
end P1
```


SINCRONIZACIÓN con semáforos

- "espera" y "señal" se ejecutan en dos procesos separados
- El que ejecuta "espera" se bloquea hasta que el otro ejecuta "señal"

SINCRONIZACIÓN
con semáforos

```
module Sincronización;
var sincro:semaforo;

process P1 (*proceso que espera*)
begin
    …
    espera(sincro);
    …
end P1;

process P2 (*proceso que señala*)
begin
    …
    señal(sincro);
    …
end P2;

begin (*sincronización*)
    inicializa(sincro,0);
    cobegin
        P1;
        P2;
    coend
end Sincronización
```


SINCRONIZACIÓN
(problema del productor-consumidor)

```
module Productor_consumidor;
var BufferComun:buffer;

process Productor;
    var x:dato;
    begin
        loop
            produce(x);
            while Lleno do
                (*espera*)
            end;
            Poner(x);
        end
    end Productor;

process Consumidor;
    var x:dato;
    begin
        loop
            while Vacio do
                (*espera*)
            end;
            Tomar(x);
            Consume(x);
        end
    end Consumidor;

begin
    cobegin
        Productor;
        Consumidor;
    coend
end Productor_consumidor;
```

SINCRONIZACIÓN (problema del productor-consumidor)

Esta solución no es satisfactoria:

- Las funciones Poner(x) y Tomar(x) utilizan el mismo buffer -> exclusión mutua.
- Ambos procesos utilizan una espera ocupada cuando no pueden acceder al buffer.

Soluciones: Utilización de semáforos

SINCRONIZACIÓN (productor-consumidor con semáforos)

```
module Productor_consumidor;
var BufferComun:buffer;
AccesoBuffer, Nolleno,
Novacio:semaforo;

process Productor;
    var x:dato;
    begin
      loop
        produce(x);
        espera(AccesoBuffer);
        if Lleno then
          señal(AccesoBuffer);
          espera(Nolleno);
          espera(AccesoBuffer);
        end;
        Poner(x);
        señal(AccesoBuffer);
        señal(Novacio);
      end
    end Productor;

process Consumidor;
    var x:dato;
    begin
      loop
        espera(AccesoBuffer);
        if Vacio then
          señal(AccesoBuffer);
          espera(Novacio);
          espera(AccesoBuffer);
        end
        Tomar(x);
        señal(AccesoBuffer);
        señal(Nolleno);
        Consume(x);
      end
    end Consumidor;

begin
    inicializa(AccesoBuffer,1);
    inicializa(Nolleno,1);
    inicializa(Novacio,0);
    cobegin
      Productor;
      Consumidor;
    coend
end Productor_consumidor;
```

Vesrsión general de semáforos (1/2)

- Se inicializa con el número N de recursos
- "espera" y "señal" se diseñan para impedir el acceso al recurso cuando el semáforo es menor o igual a cero.
- Cuando se solicita y obtiene un recurso el semáforo se decrementa
- Cuando se libera un recurso se incrementa

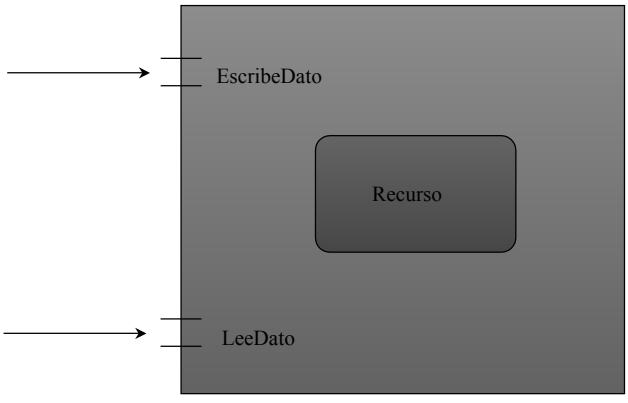
Vesrsión general de semáforos (2/2)

- inicializa (S:Semaforo; v:integer)
  Poner el valor del semáforo al valor de v(N)
  numero_suspendidos:=0
- espera(S)
  if S>0 then S:=S-1
  else
    numero_suspendidos:=numero_suspendidos+1;
    suspender la tarea que hace la llamada y ponerla en la cola de tareas
- señal(S)
  if numero_suspendidos>0 then
    numero_suspendidos:=numero_suspendidos-1
    pasar al estado listo un proceso suspendido
  else S:=S+1

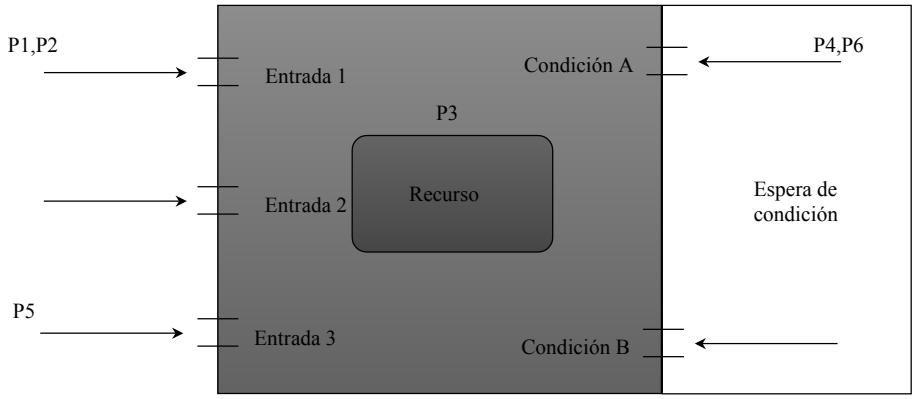
MONITORES (1/3)

- Un monitor es un conjunto de procedimientos que proporcionan el acceso con exclusión mutua a un recurso o conjunto de recursos compartidos por un grupo de procesos.

MONITORES (2/3)



MONITORES (3/3)



Monitor: implementación de un semáforo

```
monitor:semaforo;

from condiciones import condicion, espera,
    señal;
export
    sespera, sseñal;
var
    ocupado: boolean;
    libre:condicion;


procudere sseñal(var libre:condicion);
    begin
        ocupado:=false;
        señal(libre);
    end sseñal;
```

```
procedure sespera(var libre:condicion);
begin
    if ocupado then
        espera(libre);
        ocupado:=true;
    end
end sepsera

begin
    ocupado:=false;
end
```

Monitor: solución al problema del productor-consumidor

```
monitor: productor-consumidor;

from condiciones import
     condicion,espera,señal;
export poner,tomar;
const tamaño=32;
var
     buff:array[0..tamaño-1] of dato;
     cima,base:0..tamaño-1;
     espacio,itemdisponible:
          condicion;
     numeroenbuffer:integer;
```

```
procedure poner(item:dato);
  begin
    if numeroenbuffer=tamaño
         then espera(espacio)
         end;
    buff[cima]:=item;
    numeroenbuffer:=
         numeroenbuffer+1;
    cima:=(cima+1) mod tamaño;
    señal(itemdisponible);
  end poner;
```

```
procedure tomar(var item:dato);
  begin
    if numeroenbuffer=0 then
       espera(itemdisponible);
    end;
    item:=buff[base];
    numeroenbuffer:=
         numeroenbuffer-1;
    base:=(base+1) mod tamaño;
    señal(espaciodisponible);
  end tomar;

begin (*inicialización*)
  numeroenbuffer:=0;
  cima:=0;
  base:==0;
end
```


MENSAJES (1/3)

- Permiten sincronización y comunicación entre procesos y dar una solución al problema de la concurrencia.
- Las operaciones básicas son:
  - enviar(mensaje)
  - recibir(mensaje)

MENSAJES (2/3)

La comunicación por mensajes requiere que se establezca un enlace entre el receptor y el emisor.

- Cómo y cuántos enlaces se pueden establecer entre los procesos
- La capacidad del mensaje del enlace
- El tipo de los mensajes

MENSAJES (3/3)

La implementación varía dependiendo de:

- El modo de nombrar los procesos
- El modelo de sincronización
- Almacenamiento y estructura del mensaje

Modos de nombrar los mensajes

- Comunicación directa
  - enviar(Q,mensaje)
  - recibir(P,mensaje)
- Comunicación indirecta

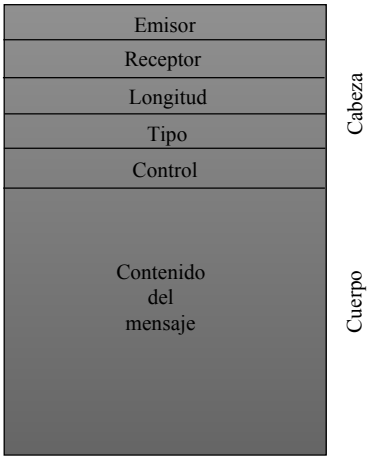
  Los mensajes se envían a un **buzón**:
  - enviar(buzónA,mensaje)
  - recibir(buzónA,mensaje)

Modelos de sincronización

- Síncrona: El proceso que envía sólo prosigue su ejecución cuando el mensaje ha sido recibido.
- Asíncrona: El proceso que envía sigue su ejecución sin preocuparse de si el mensaje se recibe o no.
- Invocación remota: El proceso que envía el mensaje sólo prosigue su ejecución cuando ha recibido una respuesta del receptor.

Estructura de los mensajes

- Longitud fija
- Longitud variable
- De tipo indefinido



Semáforo binario con mensajes

```
module semaforo;

type mensaje=…; (tipo del mensaje*)
const nulo=…; (*mensaje vacío*)

procedure espera(var S:semaforo);
  var temp:mensaje;
  begin
    recibir(Sbuzon,temp);
    S:=0;
  end espera;

procedure señal (var S:integer);
  begin
    enviar(Sbuzon,nulo);
  end señal;
```

```
procdure inicializa (varS:integer; valor:boolean);
  begin
    if valor=1 then
      enviar(Sbuzon,nulo);
    end
    S:=valor;
  end inicializa;

begin
  crear_buzon(Sbuzon);
end {semaforo}
```

INTEBLOQUEO

```
P1                  P2
.......             .......
espera(S1)          espera(S1)
    espera(S2)          espera(S2)
.......             .......
.......             .......
    señal(S2)           señal(S2)
senal(S1)           senal(S1)
end P1              end P2
```

```
P1                  P2
.......             .......
espera(S1)          espera(S2)
    espera(S2)          espera(S1)
.......             .......
.......             .......
    señal(S2)           señal(S1)
senal(S1)           senal(S2)
end P1              end P2
```


Caracterización del interbloqueo

- Exclusión mutua
- Retención y espera
- No existencia de expropiación
- Espera circular

Métodos para evitar interbloqueo

- Prevención de interbloqueos
- Evitación de interbloqueos

Prevención de interbloqueos

Evitar alguna condición que llevan al interbloqueo

- Retención y espera: Hacer que se pida todo o nada
- No existencia de expropiación: Que sí se permita expropiación
- Espera circular: Se ordenan los recursos y se impone que los recursos se pidan en orden ascendente.

Evitación de los interbloqueos

No se evita toda posibilidad de interbloqueo, sino que cuando se ve que se puede producir se soslaya.

- Técnica: algoritmo del banquero

Algoritmo del banquero (1/4)

Proceso	Usados	Posibles necesarios	Máximos necesarios
P1	5	0	5
P2	6	0	6
P3	4	0	4
Total disponibles		10	

Algoritmo del banquero (2/4)

Proceso	Usados	Posibles necesarios	Máximos necesarios
P1	2	3	5
P2	1	5	6
P3	3	1	4
Total disponibles		4	

Estado seguro

Algoritmo del banquero (3/4)

Proceso	Usados	Posibles necesarios	Máximos necesarios
P1	3	2	5
P2	4	2	6
P3	2	2	4
Total disponibles		1	

Estado no seguro

Algoritmo del banquero (4/4)

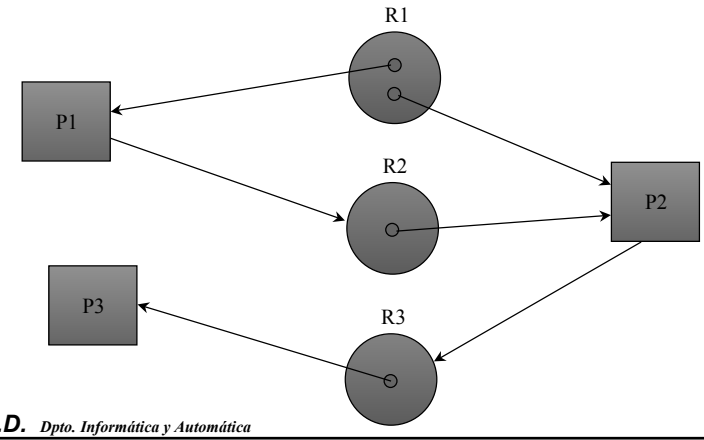
- Se permiten las condiciones de exclusión mutua, retención y espera, y de no existencia de expropiación
- Los procesos solicitan el uso exclusivo de los recursos necesitan; mientras esperan a alguno se les permite mantener los recursos de que disponen sin que se les pueda expropiar
- Los procesos piden los recursos al sistema operativo de uno en uno
- El sistema puede conceder o rechazar cada petición
- Una petición que no conduce a un estado seguro se rechaza y cada petición que conduce a un estado seguro se concede

Detección de los interbloqueos

- Se utiliza en los sistemas en los que se permite que se produzca el interbloqueo o que no se comprueba las condiciones del mismo
- Se utilizan algoritmos de detección y recuperación.

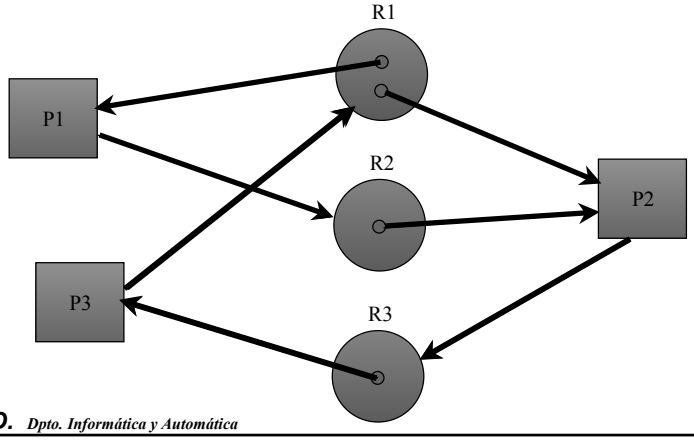
Grafos de asignación de recursos

Grafo dirigido que indica las asignaciones de los recursos a los procesos y las peticiones que estos realizan



Grafos de asignación de recursos

Interbloqueo: dos ciclos
(R1,P1,R2,P2,R3,P3,R1) y (R1,P2,R3,P3,R1)



Recuperación de interbloqueos (1/2)

Se suelen tomar dos opciones:

- Reiniciar uno o mas de los procesos bloqueados
- Expropiar los recursos de algunos de los procesos bloqueados

Recuperación de interbloqueos (2/2)

Para reiniciar hay que tener en cuenta:

- La prioridad del proceso
- El tiempo de procesamiento utilizado y el que resta
- El tipo y número de recuros que dispone
- El número de recursos que necesita para finalizar
- El número de otros procesos que se verían involucrados con su reiniciación.

Interbloqueo en Sistemas de Tiempo Real

- OJO: En algunos sistemas de Tiempo Real el interbloqueo puede tener resultados inaceptables, por lo que no se puede permitir que se presente dicha situación.

Tercera sesión

- El ciclo de vida software

El ciclo de vida del software

Fases del ciclo de vida del software

- Fase conceptual
- Fase de requerimientos
- Fase de diseño
- Fase de programación
- Fase de verificación
- Fase de mantenimiento

Fase conceptual

- Determinación de las necesidades y de las metas globales.
- No se hacen requerimientos formales ni decisiones sobre software o hardware
- El propósito es:
  - Identificar las necesidades del producto y las metas
  - Producir estudios de viabilidad

Fase de requerimientos (1/3)

- Se prepara documento de requerimientos y especificaciones del software
- Realizado por el "cliente"
- Contiene información sobre que tiene que hacer el producto:
  - tiempo/capacidad de procesamiento
  - interfaz de usuario
  - requerimientos de precisión
  - Interfaces software y hardware
  - ...

Fase de requerimientos (2/3)

Requerimientos funcionales

- Son los verificados directamente en la ejecución del programa:

En 1 milisegundo, el subsistema de navegación debe suministrar todos los datos compensados de la aceleración al computador principal, que los leerá mediante DMA. Estos datos no se pueden actualizar mientras el computador principal los esté leyendo

Requerimientos no funcionales

- Son características del sistema que no se pueden verificar mediante la ejecución de un programa: tipo de máquina, lenguaje de programación, estilo de programación, mantenibilidad, modularidad, seguridad, …

El sistema se debe codificar en ANSI-C siguiendo las adecuadas prácticas de ingeniería del software. No se permiten sentencias "GOTO".

Fase de requerimientos (3/3)

- Los objetivos claves de esta fase son:
- Definir todas las interfaces hardware y software
- Escribir el documento de requerimientos
- Escribir el plan de verificación
- Preparar la planificación del proyecto

Fase de diseño

- Conversión del documento de requerimientos a una especificación de diseño detallada.
- Tareas de esta fase:
  - Partición del software en módulos
  - Preparar un documento de diseño detallado
  - Desarrollar casos de tests.

Fase de programación

Comienza cuando se finaliza la fase de diseño, aunque en la práctica hay solapación, y termina cuando se integra el software.

- Se escribe el código
- Se depura el software
- Se desarrollan casos de test automatizados
- Se integran los módulos

Fase de verificación

Comienza cuando ha cesado completamente la fase de programación

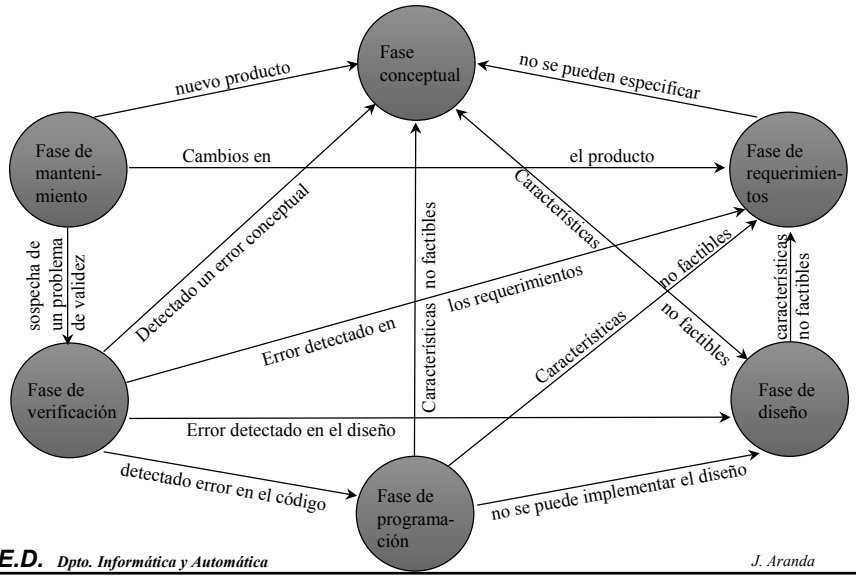
- Se realiza la validación del software
- Se preparan los informes de verificación

Fase de mantenimiento

Comienza cuando el software se ha validado.

- Distribución del producto
- Soporte a los clientes
- Continuación de correcciones de los errores de los programas

Transiciones no-temporales en el ciclo de vida del software



Especificaciones y diseño de sistemas de tiempo real

Especificación vs. diseño

- Especificación: realizada por el cliente, dice lo que debe de hacer el software
- Diseño: realizada por el analísta o diseñador, dice como debe hacer el software lo que se le pide.
- CASE (Computer aided software engineering)

Especificaciones

Lenguaje humano

Especificación de un sistema aeronáutico

- El software recibirá cada 5 milisegundos pulsos de los acelerómetros x, y,z. Mediante DMA, los convertidores A/D proporcionan al software los ángulos de roll, pitch, yaw; cada 40 milisegundos, estos ángulos indican la orientación del avión.
- El software recibe información de la temperatura cada segundo.
- La tarea del software es calcular el vector de velocidad real basandose en la orientación y la lectura de los acelerómetros, tiene en cuenta factores de compensación (como la temperatura) cada 40 mseg. Cada segundo se actualiza la información de velocidad en un display del piloto.

Especificaciones

Especificaciones matemáticas

Sistema aeronáutico

- Las medidas de los acelerómetros son

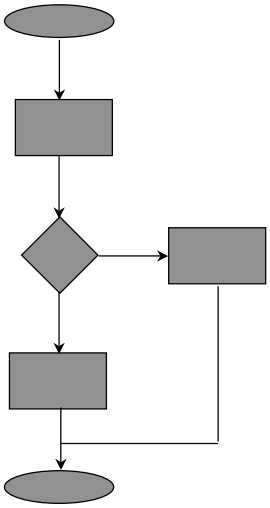
  $a_x=a+\delta a$
- Las medidas de los giros

  $\omega_x=\omega+\delta\omega$
- Ecuaciones del sistema

  $x'(t)=Fx(t)+Bu(t)$

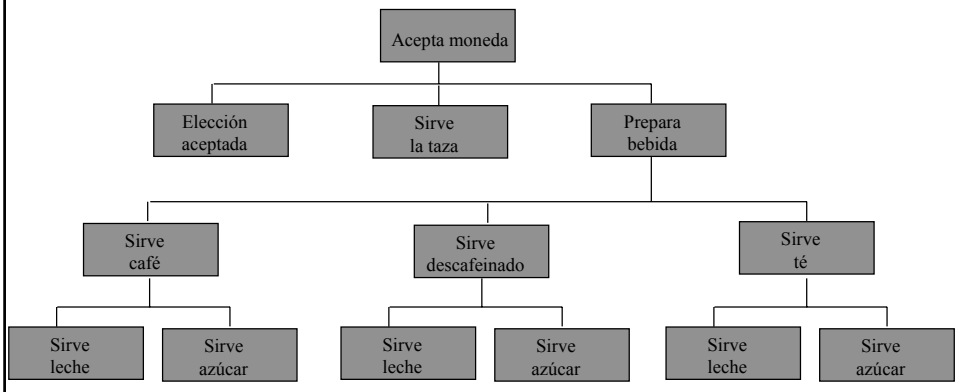
Diagramas de flujo

- Utiles para programas de 5.000 a 10.000 instrucciones, pero no mayores.
- En multitarea, describen cada tarea separadamente, pero no represetan la interacción entre procesos, ni la conducta temporal.



Arboles de llamadas o diagramas de estructuras

- Expendedora de café



Seudocódigo y lenguajes de diseño de programación

- Usados para especificar sistemas de una forma muy cercana a la escritura del código.
- Mas abstractos que los lenguajes de alto nivel
- Algunos lenguajes, como Ada o Modula-2, se han utilizado como LDP.
- Se adaptan mejor para hacer los programas.
- En contra, es una clase mas de lenguaje de programación. La utilización de abstraciones a alto nivel no evita los errores.

Seudocódigo y lenguajes de diseño de programación

Cajero automático

```
begin
  Acepta la tarjeta
  Display"Entre su código"
  If código no es correcto
   begin
    Display "Disculpe"
    Exit/eject tarjeta
   end
  else
   begin
    Display "Pulse una tecla"
    Acepta tecla de función
    If Tecla=consulta de
   cuenta
```

```
begin
 Display estado de la cuenta
 Exti/Eject tarjeta
end
else
 begin
  if tecla=retirar fondos
   begin
    Display "¿Que cantidad?
    Acepta cantidad
    if catindad > saldo
     begin
      display "Disculpe"
      exit/eject tarjeta
     end
```

Autómatas de estados finitos

Métodos de represetar AEF:

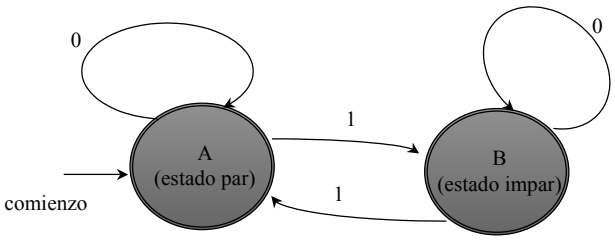
- Matemáticamente
- Con diagramas
- Representación matricial.

Un AEF consiste de un conjunto finito de estados descritos por círculos etiquetados, uno de los cuales es el estado inicial y uno (o mas) son el estado terminal.

Autómatas de estados finitos

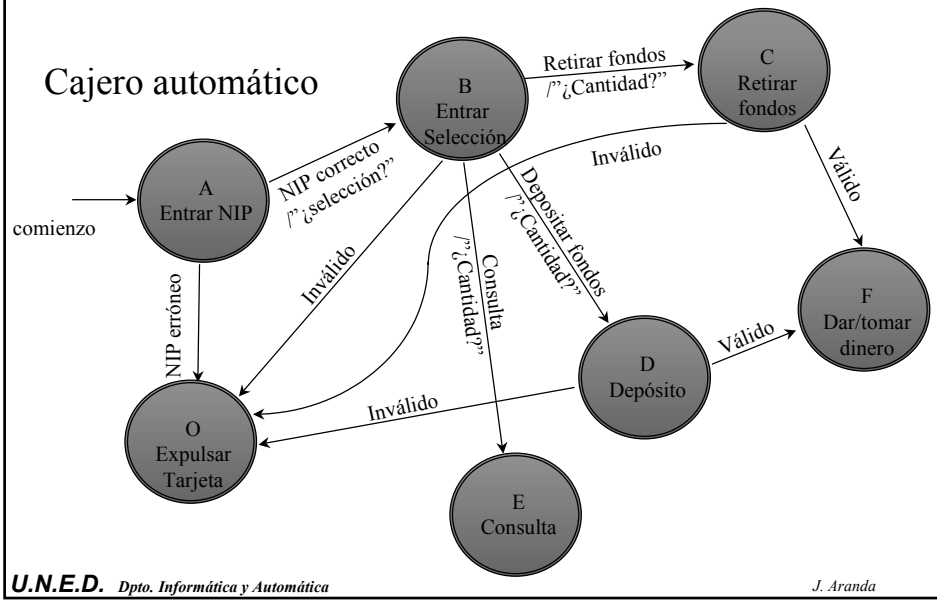
- AEF para chequear la paridad en una cadena de bits.

Entrada	Estado actual A	B
0	A	B
1	B	A



Autómatas de estados finitos

Cajero automático



Autómatas de estados finitos

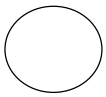
Cajero automático

Entrada	A	B	C	D
NIP erróneo	0/	NP	NP	NP
NIP correcto	B/"¿selección?	NP	NP	NP
Retirar fondos	NP	C/"¿cantidad?	NP	NP
Consulta	NP	E	NP	NP
Depósito	NP	D/"¿cantidad?	NP	NP
Válido	NP	NP	F	F
Inválido	NP	NP	O	O

Diagramas de flujos de datos

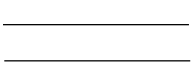
- Se propusieron por DeMarco en 1978 como una herramienta de análisis estructurado para el modelado de sistemas software
- Hatley y Pribhai en 1987 lo extiende para permitir modelar sistemas de tiempo real.

Diagramas de flujos de datos



Procesos

Productores o consumidores de datos



Almacenamiento de datos

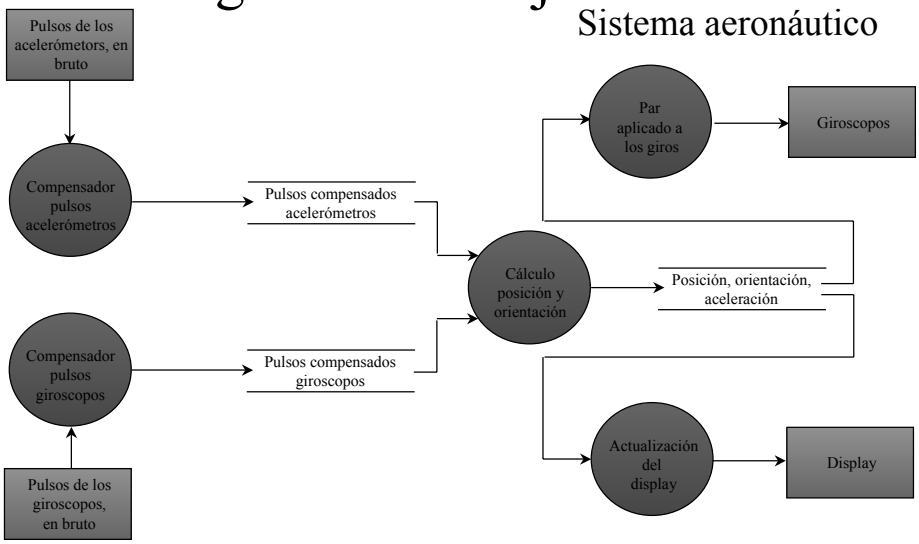
Flujo de datos

Reglas de DeMarco

- Identificar todas las entradas de la red y los flujos de salida. Dibujarlos alrededor del diagrama
- Trabajar los caminos desde las entradas a las salidas, volver atrás desde las salidas a las entradas, o desde la mitad.
- Etiquetar cuidadosamente todos el interfaz de flujos de datos
- Etiquetar los procesos en términos de sus entradas y salidas.
- Ignorar la inicialización y la terminación
- Omitir (inicialmente) detalles de caminos de errores triviales.
- No mostrar el flujo de control o información de control.

Diagramas de flujos de datos

Sistema aeronáutico



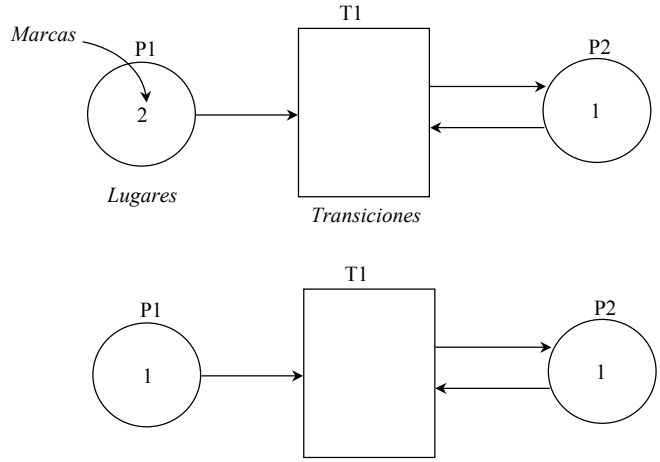
Diagramas de flujos de datos

Extensiones de Hatley y Pribhai

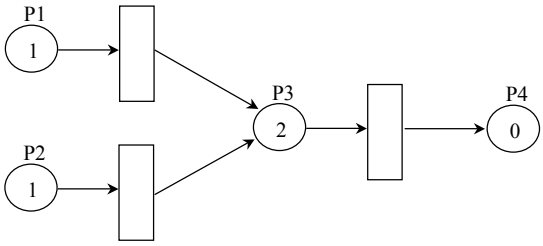
Se incorporan otros dos tipos de represetanción del sistema

- **Diagrama de flujo de control:** Diagrama que muestra el flujo de las señales de control a través del sistema.
- **Especificaciones de control:** es un autómata de estados finito en representación de diagrama y matricial.

Redes de Petri

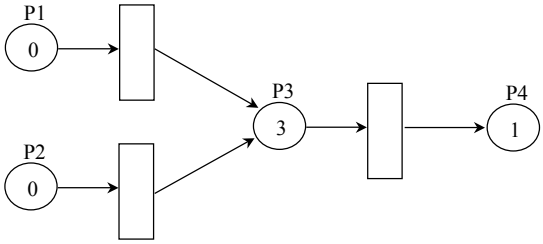


Redes de Petri



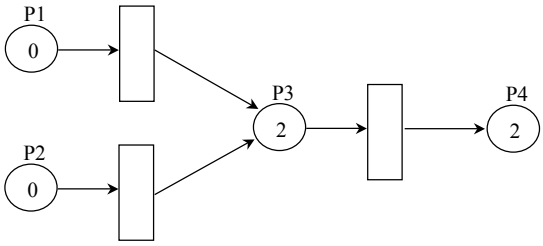
	P1	P2	P3	P4
mo	1	1	2	0

Redes de Petri



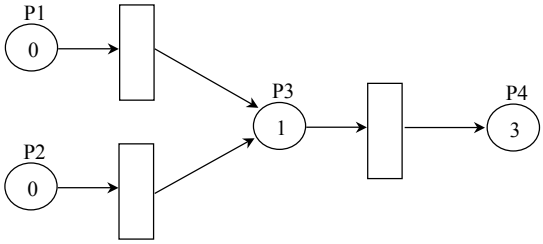
	P1	P2	P3	P4
mo	1	1	2	0
m1	0	0	3	1

Redes de Petri



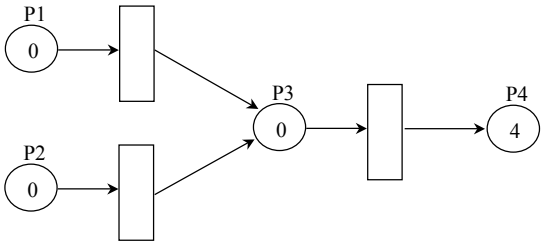
	P1	P2	P3	P4
mo	1	1	2	0
m1	0	0	3	1
m2	0	0	2	2

Redes de Petri



	P1	P2	P3	P4
mo	1	1	2	0
m1	0	0	3	1
m2	0	0	2	2
m3	0	0	1	3

Redes de Petri

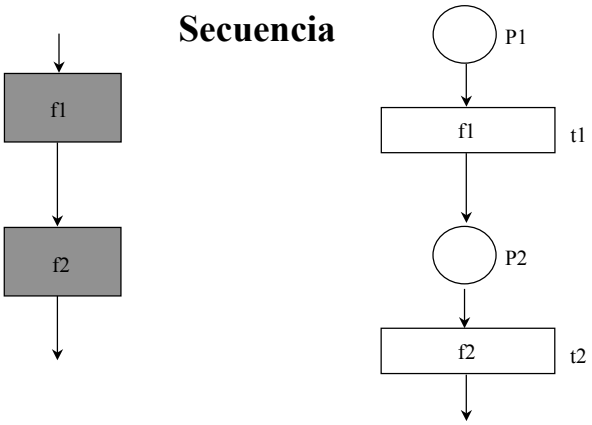


	P1	P2	P3	P4
mo	1	1	2	0
m1	0	0	3	1
m2	0	0	2	2
m3	0	0	1	3
m4	0	0	0	4

Redes de Petri

Analogía entre los diagramas de flujo y las redes de Petri

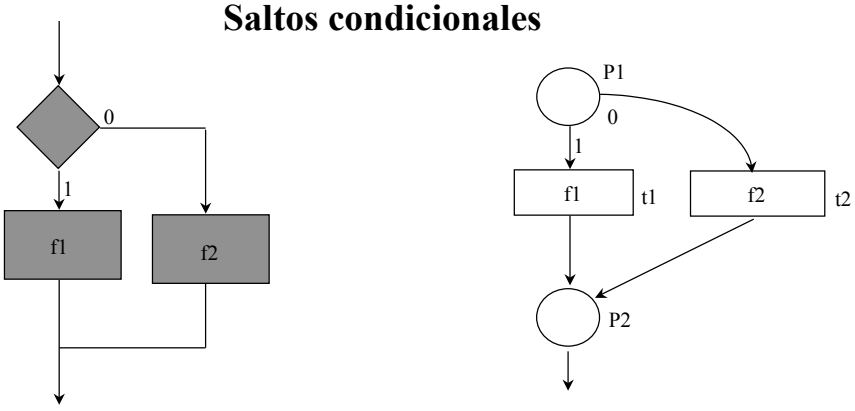
Secuencia



Redes de Petri

Analogía entre los diagramas de flujo y las redes de Petri

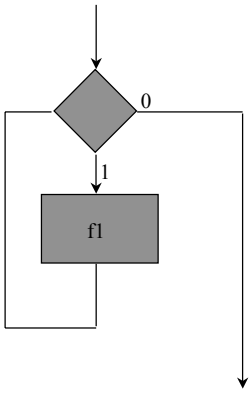
Saltos condicionales

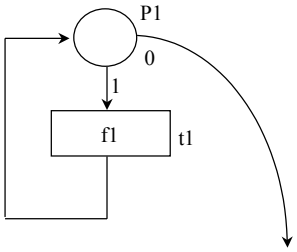


Redes de Petri

Analogía entre los diagramas de flujo y las redes de Petri

Bucles



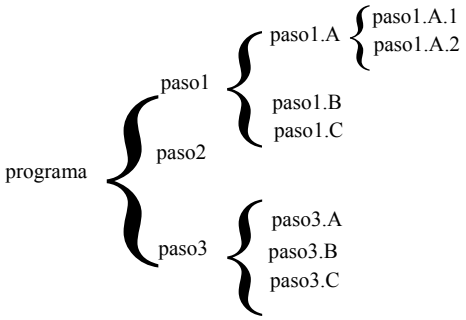


Notación de Warnier-Orr

La semántica o notación de Warnier-Orr es una representación metodológica similar a los diagramas de estructuras pero con algunas mejoras.

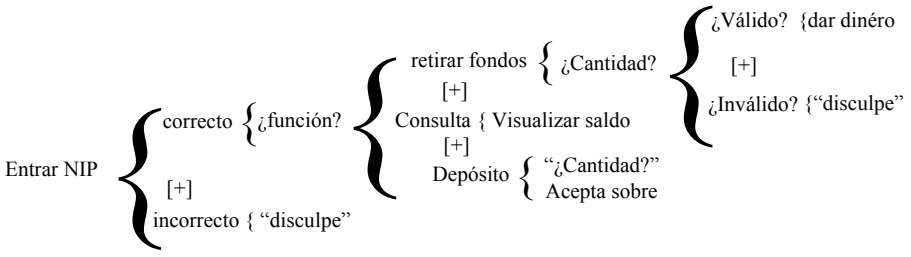
- Se puede utilizar para representar tanto estructuras como procesos
- Utiliza una notación teórica mas rigurosa
- Utiliza llaves {, + para o inclusiva y [+] para o exclusiva

Notación de Warnier-Orr



Notación de Warnier-Orr

Cajero automático



Diagramas de estados

- Los diagramas de estado de Harel (1988) combinan los Autómatas de estados finitos con los diagramas de flujo de datos y una característica denominada comunicación de "broadcast", que es una forma de describir las operaciones síncronas y asíncronas.:

Diagramas de estados = AEF + "niveles de detalle" + ortogonalidad ("tareas separadas") + comunicación "broadcast"

Diagramas de estados

Componentes:

- Los AEF se representan en la forma normal
- Los detalles se representan por lo que hay en el interior de los estados
- Las comunicaciones "broadcast" se representan por flechas etiquetadas, como en los AEFs
- La ortogonalidad se representa por líneas punteadas separando los estados
- Los símbolos a,b,…z representan los sucesos que desencadenan las transiciones, de la misma forma que se representan las transiciones en los AEFs
- Las letras minúsculas entre paréntesis representan condiciones que se deben cumplir para que se produzca la transición

Diagramas de estados

Detalle

- Los D.E. estimulan el diseño top-down de los módulos.
- Para cualquier módulo (estado en un AEF) aumenta los detalles al describir los estados internos.

Diagramas de estados

Ortogonalidad

- Representa la concurrencia en el sistema para procesos que corren aisladamente, se llaman estados AND.
- Se representa por líneas punteadas

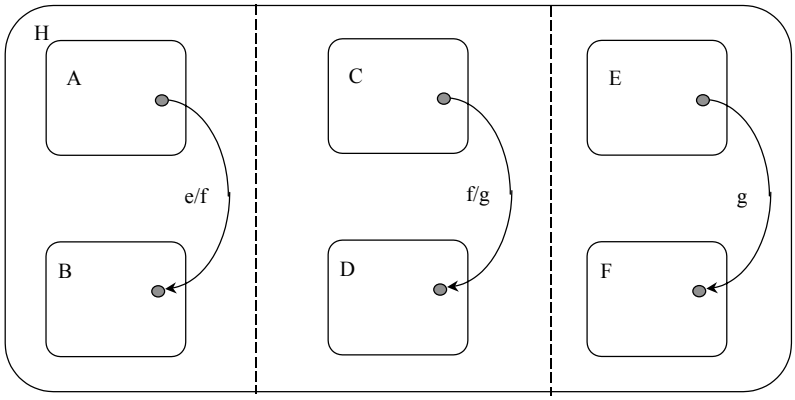
Diagramas de estados

Comunicación “broadcast”

- Representa la transición de estados ortogonales basados en el mismo suceso.
- Reacción en cadena: sucesos que pueden desencadenar otros sucesos.

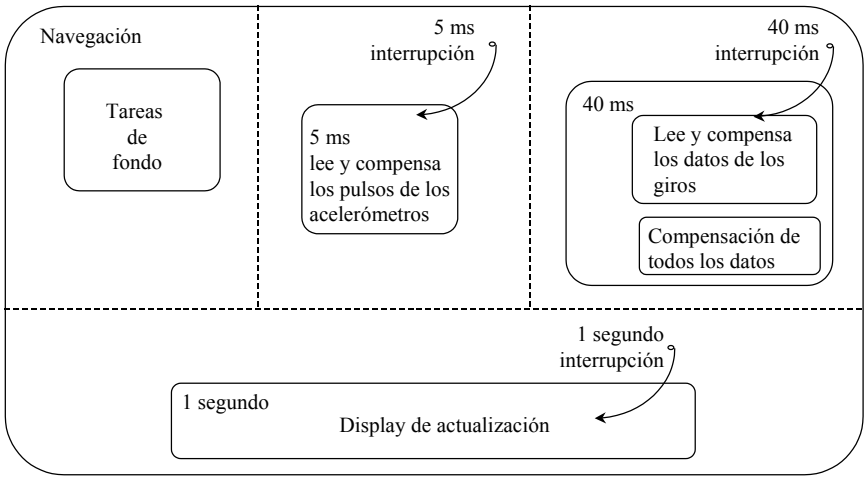
Diagramas de estados

Reacción en cadena



Diagramas de estados

Sistema de navegación



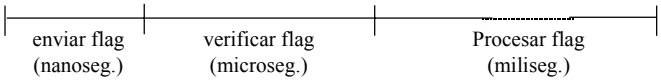
Análisis del funcionamiento y optimización del sistema

Calculo del tiempo de respuesta

- Tiempo que transcurre entre el momento en el que el suceso señala que se produce la tarea y el momento en el que la tarea completa el procesamiento.
- Este tiempo depende del tipo de sistema que se involucra.

Calculo del tiempo de respuesta

Sistemas de bucle de consulta (polled loop)



El tiempo de respuesta depende de:

- los retardos del hardware involucrado en activar el flag software al dispositivo externo.
- El tiempo para que el bucle de consulta verifique el flag.
- El tiempo necesario para procesar el evento asociado al flag.

Calculo del tiempo de respuesta

Sistemas de bucle de consulta (polled loop)

En el caso de que ocurran varios sucesos a la vez, es decir que se inicialice un evento mientras se está procesando el evento anterior, entonces se empeora el tiempo de respuesta.

f tiempo necesario para chequear el flag

P tiempo para procesar el evento

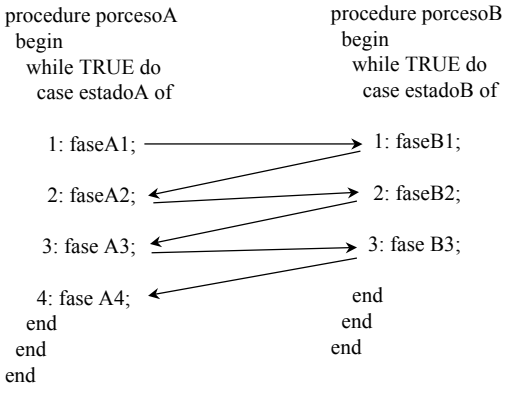
El tiempo de respuesta para los *n* eventos que colisionan está limitado por:

$$nfP$$

Calculo del tiempo de respuesta

Sistemas de corutinas:

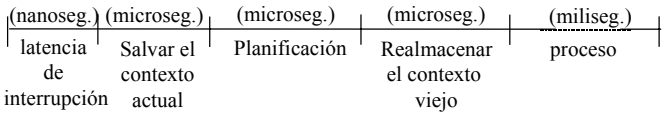
- En este caso no hay interrupciones.
- El tiempo de respuesta se calcula buscando el peor caso que se puede dar.



Calculo del tiempo de respuesta

Sistemas de interrupciones:

- En este caso el cálculo del tiempo de respuesta depende de una gran cantidad de factores: tiempo de latencia, tiempo de conmutación de contexto, tiempos de planificación/distribución.



$$R_i = L_i + C_s + S_i + R_s + A_i$$

Calculo del tiempo de respuesta

Sistemas de interrupciones:

Cálculo de los tiempos:

- C_s y R_s son los mismos para el código de cualquier aplicación.
- S_i es despreciable si la CPU está equipada con un controlador de interrupciones. Si sólo se soporta una interrupción en conjunción con un controlador de interrupciones, entonces se S_i pude medir contando las instrucciones.
- Determinación de L_i : Dos casos:
  - La tarea de mayor prioridad es la que interrumpe:
    $$L_i = L_p + \max\{L_L, L_D\}$$
  - La tarea de menor prioridad es la que interumpe:
    $$L_i = L_H$$

Latencia de la interrupción

- Retardo entre que ocurre una interrupción y la CPU comienza a actuar.
- Causas tanto hardware como software (intencionado o fortuito).
- Efectos mas pronunciados (y fáciles de encontrar sus causas) con interrupciones de baja prioridad cuando se ejecutan tareas de alta prioridad.

Latencia de la interrupción

Retardo de propagación

- Debido a las limitaciones en la velocidad de conmutación de los dispositivos digitales y el tiempo de tránsito de los electrones por el alambre. (nanoseg. o microseg.).
- Son dados por el fabricante o se pueden medir con un osciloscópio o un analizador lógico.
- Se suelen ignorar.

Latencia de la interrupción

Tiempo de ejecución de la macroinstrucción

- Macroinstrucción = microprograma (no interrumpible).
- La instrucción con el mayor tiempo de ejecución de tu código causará la latencia de interrupción máxima.
- Si se pasa por alto puede causar problemas.
- En algunos lenguajes (FORTRAN) se desactivan las interrupciones para el paso de parámetros. Esto acarrea multiples cargas, y grandes periodos con las interrupciones desactivadas.

Latencia de la interrupción

Interrupciones desactivadas

- Intencionadamente se pueden desactivar las interrupciones.
- Razones: protección de regiones críticas, rutinas de almacenamiento en memoria intermedia, conmutación de contexto, etc.

Latencia de la interrupción

Cuando una tarea de alta prioridad interrumpe a otra de baja prioridad, la latencia de la interrupción total es:

$$L_{max} = L_P + \max\{L_I, L_D\}$$

L_{max} es la latencia de interupción máxima

L_P es la latencia debido a l retardo de propagación

L_I es el tiempo de terminación mayor de una instrucción

L_D es el tiempo máximo en el que son desactivadas las interrupciones

Latencia de la interrupción

Baja prioridad interumpe a alta

- Genralmente el hardware bloquea la interrupción en el controlador de interrupciones y no se envia ninguna señal a la CPU hasta que termina la de alta prioridad.

$$L_{max} = L_H$$

donde L_H es el tiempo necesario para completar la de alta prioridad

Tiempo de carga

- Es importante conocer a priori el tiempo de carga para la selección del hardware y el método de diseño del sistema.
- Métodos de medir el tiempo de carga:
  - Mediante un analizador lógico
  - Contando instruccíones

Medida del tiempo de carga

Mediante un analizador lógico

- Es el mejor método de medir el tiempo de ejecución y la utilización completa de la CPU
- Tiene en cuenta las latencias del hardware y otros retardos no debido al tiempo de ejecución de la instrucción.
- Para poderlo usar el hardware tiene que estar disponible y codificado todo el software.

Medida del tiempo de carga

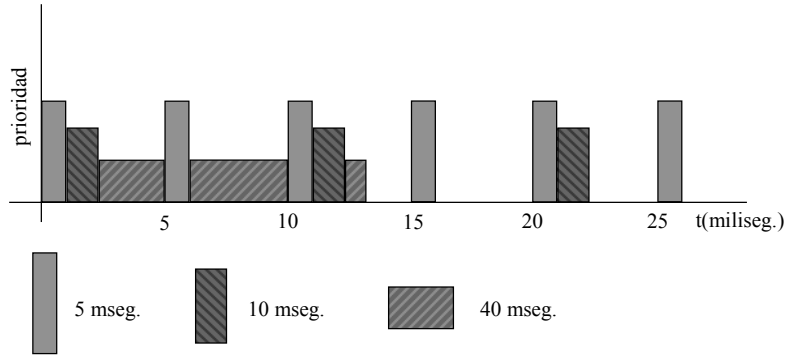
Contando instrucciones

- Si no se dispone de analizador lógico o es muy pronto, el mejor método es contar las instrucciones.
- El código tiene que estar escrito

Medida del tiempo de carga

Representación gráfica

Ejemplo: sistema de navegación



Medida del tiempo de carga

Simuladores del tiempo de ejecución de las instrucciones

- Programas de simulación para predecir el tiempo de ejecución de la instrucción y la capacidad de procesamiento de la CPU.

Reducción del tiempo de respuesta y del tiempo de carga

- Un paso preliminar en reducir el tiempo de carga es identificar el despilfarro de cálculo.

Calcular con el ciclo mas lento

Todos los cálculos se tienen que hacer a la velocidad mas lenta que se pueda tolerar

» Ej.: Verificar la temperatura de una sala grande a 1 seg. es antieconómico.

Reducción del tiempo de respuesta y del tiempo de carga

Aritmética escalada

- Por ejemplo: operar enteros es mas rápido que operar en coma flotante. Se puede multiplicar enteros por un **factor de escala** para simular operaciones en coma flotante.
- El bit menos significativo es la granularidad
- El bit mas significativo es el bit de signo.
- El número mayor que se puede representar es: $2^{n-1}-1$
- El mas pequeño es $-2^{n}-1$.

Reducción del tiempo de respuesta y del tiempo de carga

Aritmética escalada

Sistema de navegación de un avión:

- Los pulsos de los acelerómetros x, y, z se pasan a aceleraciones reales aplicando un factor de escala de 0.01.
- 0000 0000 0001 0011 representa una velocidad de 19*0.01=0.19pies/seg.
- La mayor velocidad será 327.67 y la menor -327.68

Reducción del tiempo de respuesta y del tiempo de carga

Aritmética escalada

Los números escalados se pueden sumar y restar juntos, y multiplicar por otra constante (pero no por otro número escalado)

- Como hay una relación inversa entre precisión y rango del número escalado, una cuestión natural es: ¿Como decidir la granularidad apropiada?

  Suponer un número de n-bits, se quiere representar números entre -x y x, el número correspondiente al bit menos significativo es

  $$2^{-(n-1)}x$$

  p.e.: si x=311, enonces es 0.00949.

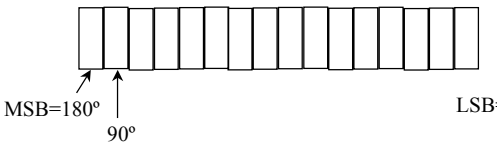
Reducción del tiempo de respuesta y del tiempo de carga

Medida angular binaria (BAM)

Es una variación de los números escalados que utiliza el hecho de que sumar 180º a un ángulo es equivalente a tomar su complemento dos. Es decir: x+180º=x-180º=-x.

Si al bit mas significativo se le asigna 180º entonces el menos significativo será:

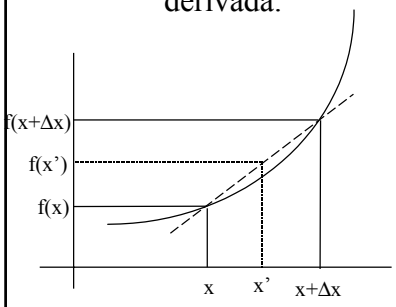
$$2^{-(n-1)}*180 \text{ grados}$$



Reducción del tiempo de respuesta y del tiempo de carga

Tablas de consulta

Se basa en la idea de la interpolación. Se van almacenado puntos y, posiblemente, el valor de la función o de su derivada.



x	f(x)
$x_0+\Delta x$	$f(x_0+\Delta x)$
.....	.....
$x_0+(n-1)\Delta x$	$f(x_0+(n-1)\Delta x)$

$$f(x')=f(x)+(x'-x)*[f(x+\Delta x)-f(x)]/\ \Delta x$$

Reducción del tiempo de respuesta y del tiempo de carga

Fundamentos de la teoría de optimización

- uso de identidades aritméticas
- reducción de fuerza bruta
- eliminación de expresiones comunes
- uso de funciones intrínsecas
- constantes "plegables" (folding")
- eliminación de invariantes en los bucles
- eliminación de induciones en el bucle
- uso de registros y caches
- eliminación de código no utilizable
- optimización del flujo de control
- propagación de constantes
- eliminación de almacenamiento no utilizado
- eliminación de variables no utilizadas
- código booleano corto
- desplegar los bucles
- fusionar bucles
- eliminación de saltos cruzados

Reducción del tiempo de respuesta y del tiempo de carga

Reducción de la fuerza bruta

Esto se refiere a la utilización de las macroinstrucciones más rápidas posibles para hacer los cálculos.

Ej.: multiplicar por potencias de 2 es como hacer un desplazamientos de bits.

Es más ra´pido multiplicar que dividir. Luego es preferible hacer x*.5 que x/2.

Eliminación de expresiones comunes

```
x := y + a * b;
y := a * b + z;
```

Se puede reemplazar por

```
t := a*b;
x:= y +t;
y := t +z;
```


Reducción del tiempo de respuesta y del tiempo de carga

Funciones intrínsecas

Son macros cuyas llamadas se sustituyen por código en la compilación.

Constantes plegables

x:=2.0*x*4.0;

se optimiza plegando 2.0*4.0 en 8.0.

- Hacerlo a mano puede hacer mas fácil la depuración.

Eliminación de invariantes del bucle

```
x:=100;
while (x>0) do
    x:=x-y+z;
```

se puede reemplazar por

```
x:=100;
t:=y+z;
while (x>0) do
    x:=x-t;
```

Reducción del tiempo de respuesta y del tiempo de carga

Uso de registros y caches

Como las operacioens registro a registro son mas rápidas, se optimiza utilizandolas.

Eliminación de código no utilizable

Código que no es alcanzado en el flujo normal de control. Por ejemplo las sentencias de depuración.

Optimización del flujo de control

```
    goto label1;
label0 y=1;
label1 goto label2;
```

Se puede reemplazar por

```
    goto label2;
label0 y=1;
label1 goto label2;
```

- Es código, que normalmente, no se genera por programadores, sino de forma automática

Reducción del tiempo de respuesta y del tiempo de carga

Propagación de constantes

Ejemplo en Pascal:

```
x:=100;
y:=x;
```

En ensamblador:

```
LOAD R1, 100
STORE R1,X
LOAD R1,X
STORE R1,Y
```

Se puede reemplazar por:

```
x:=100;
y:=100;
```

En ensamblador:

```
LOAD R1, 100
STORE R1, X
STORE R1, Y
```

Reducción del tiempo de respuesta y del tiempo de carga

Código booleano corto

Ejemplo en Pascal

```
if (x>0) AND (y>0) then
    z:=1;
```

se puede reemplazar por

```
if (x>0) then
    if (y>0) then
        z:=1;
```

Desplegar bucles

```
for i=1 to 6 do
    a[i].= a[i]*8;
```

Es reemplazado por

```
for i=1 to 6 step 3 do
begin
    a[i]:=a[i]*8;
     a[i+1]:=a[i+1]*8;
     a[i+2]:=a[i+2]*8
end
```


Reducción del tiempo de respuesta y del tiempo de carga

Fusionar bucles

```
for i=1 to 100 do
    x[i]:=y[i]*8;
for i=1 to 100 do
    z[i]:=x[i]*y[i];
```

se puede reemplazar por

```
for i=1 to 100 do
begin
     x[i]:=y[i]*8;
    z[i]:=x[i]*y[i]
end
```

Eliminación de saltos cruzados

```
case x of
    0: x:=x+1;
    1: x:=x*2;
    2: x:=x+1;
end;
```

Es reemplazado por

```
case x of
    0,2: x:=x+1;
    1: x:=x*2;
end;
```