PROGRAMACION CONCURRENTE

I.1 Motivación de la programación concurrente

J.M. Drake

Notas:

SISTEMAS CONCURRENTES

- La crisis del software.
- Programas secuenciales, concurrentes y de tiempo real.
- Programación concurrente.
- Programación de tiempo real.
- Entornos hardware para programación concurrente.
- Ejemplo de un problema concurrente.

Notas:

LA CRISIS DEL SOFTWARE.

- Conjunto de tópicos relacionados con la problemática asociada al desarrollo de software:

  "Construir una aplicación software es una tarea mucho más compleja de lo que parece al iniciarla"

- Aspectos de esta problemática son:
  - **Resposiveness:** No satisfacen las espectativas del usuario.
  - **Reliability:** Presentan fallos y su depuración es muy difícil.
  - **Cost:** El costo es difícil de evaluar y mas alto de lo esperado.
  - **Modificability:** Son productos muy rígido y difíciles de mantener.
  - **Timeless:** Requieren para su ejecución mas tiempo del previsto.
  - **Transportability:** Hay problemas para migrar entre plataforma.
  - **Efficiency:** Sólo utilizan una parte de la capacidad de hardware.

Notas:

Desde 1968 se identifica con la denominación **"crisis del software"** el conjunto de tópicos relacionados con la problemática asociada con el desarrollo de software. Aspectos identificados con esta crisis son:

* **Responsiveness:** Frecuentemente los sistemas basados en computador no satisfacen las expectativas que tiene el usuario.

* **Reliability:** Los programas suelen presentar fallos, y su depuración es muy difícil de garantizar.

* **Cost:** El costo del software es de evaluación difícil, y es habitual que resulte mas caro de lo que se preveía.

* **Modifiability:** Los programas son productos muy rígidos y de difícil modificación. El costo de su mantenimiento es muy alto.

***Timeless:** El desarrollo del software requiere siempre más tiempo que el previsto.

***Transportability:** Cuando se traslada un software de un equipo a otro, siempre se presentan problemas de adaptación.

***Efficiency:** Los programas utilizan solo una fracción pequeña de la capacidad del hardware en que se ejecuta.

El espíritu de la crisis del software se puede resumir en la frase: **"Construir una aplicación software es una tarea mucho más compleja de lo que de antemano parece"**

Un pequeño paquete con algún millar de líneas de código es es asequible mentalmente a un programador,sin embargo, un paquete mediano o grande con cientos de miles de líneas, desborda la capacidad de cualquier programador. En estos casos se hace crítico la problemática de la crisis del software.

CAUSAS DE LA CRISIS DELSOFTWARE.

- Causas profundas de la crisis del software son:
  - La metodología en cascada que linealiza el proceso de desarrollo.
  - La metodología de modularización estructurada hace que el software sea inflexible y difícil de mantener.
  - Los programadores no tienen formación en ingeniería software.
  - Las empresas e instituciones tienen inercia a introducir las innovaciones.
  - La estructura secuencial de Von Newmann no se adapta a los problemas que se abordan.

Notas:

Causas profundas que dan lugar a la crisis del software son:

* El paradigma de **diseño estructurado** de software es de naturaleza lineal, y es difícil evaluar a nivel de proyecto el efecto posterior de las decisiones que se toman en cada fase del ciclo de vida de un software.

* El conjunto de programadores que actualmente desarrollan software **no tienen una formación o no aplican la ingeniería software**.

* Las empresas y organizaciones que desarrollan software tienen una gran **inercia a introducir novedades** de eficacia demostrada.

* La **estructura secuencial de Von Newmann** y el estilo de programación que induce, no es el adecuado a los problemas que se abordan.

PROGRAMAS Y LÍNEAS DE FLUJO DE CONTROL.

- Un programa se compone de:
  - Sentencias: Establecen las actividades (operaciones y verificaciones) que ejecuta el sistema.
  - Línea de flujo de control (Thread): Establece el orden en que se ejecutan las sentencias.

- Según las características de la línea de flujo de control los programa se clasifican en:
  - Secuenciales.
  - Concurrentes.
  - Tiempo real.

Notas:

Un programa se compone de un conjunto de sentencias (operaciones y verificaciones) y una **línea de flujo de control** (threads of control). La línea de flujo de control establece de acuerdo con la estructura del propio programa y de los datos que maneja, el orden en que deben ejecutarse las sentencias.

De acuerdo con el número de líneas de flujo de control y los requerimientos temporales que se exigen en ellas, los programas se pueden clasificar en tres tipos:

* Programas secuenciales.
* Programas concurrentes.
* Programas de tiempo real.

PROGRAMAS SECUENCIALES.

- Es el estilo de programación que corresponde al modelo conceptual de **Von Newmann**.
- Un programa secuencial tiene una **línea simple** de control de flujo.
- Las operaciones de un programa secuencial están ordenadas de acuerdo con un **orden estricto**.
- El comportamiento de un programa es solo función de las sentencias que lo componen y del orden en que se ejecuta.
- **El tiempo** que tarda en ejecutarse cada operación **no influye** en el resultado de un programa secuencial.
- La **verificación** de un programa secuencial **es sencilla**:
  - Cada sentencia da la respuesta correcta.
  - Las sentencias se ejecutan en el orden adecuado.

Notas:

•Es el estilo de programación clásico, en el que se sigue el modelo conceptual de Von Newmann.

•Los programas secuenciales presentan una **línea simple de control de flujo**. Las operaciones de este tipo de programas están estrictamente ordenados como una secuencia temporal lineal.

•El comportamiento del programa es solo función de la naturaleza de las operaciones individuales que constituye el programa y del orden en que se ejecutan.

•En los programas secuenciales, el tiempo que tarda cada operación en ejecutarse no tiene consecuencias sobre el resultado.

•Para validar un programa secuencial se necesita comprobar:

- •La correcta respuesta a cada sentencia.
- •El correcto orden de ejecución de las sentencias.

PROGRAMAS CONCURRENTES.

- Son programas que tienen múltiples líneas de flujo de control.
- Las sentencias de un programas concurrente se ejecutan de acuerdo con un orden no estricto.
- La secuencialización de un programa concurrente es entre hitos o puntos de sincronización.
- Un programa concurrente se suele concebir como un conjunto de procesos que colaboran y compiten entre sí.
- Para validar un programa concurrente:
  - Las operaciones se pueden validar individualmente si las variables no son actualizadas concurrentemente.
  - El resultado debe ser independiente de los tiempos de ejecución de las sentencias.
  - El resultado debe ser independiente de la plataforma en que se ejecuta.

Notas:

En los programas concurrentes existen **múltiples líneas de flujo de control**. Las sentencias que constituyen el programa no se ejecutan siguiendo una ordenación que corresponde a una secuencia temporal lineal.

En los programas concurrentes el concepto de secuencialidad entre sentencias continua siendo muy importante: solo que en los programas concurrentes es de orden parcial, mientras que en los programas secuenciales era de orden estricto. En los programas concurrentes la secuencialización entre procesos concurrentes se llama **sincronización**.

En el diseño de programas concurrentes es fundamental el concepto de **proceso**. Un proceso es cada uno de los programas secuenciales con una línea de control de flujo simple que constituyen el programa concurrente. Los procesos de un programa concurrente se ejecutan de forma asíncrona e independiente durante largos períodos de tiempo. De cuando en cuando, los procesos requieren comunicar con otros procesos, bien para intercambiar datos o bien para sincronizar sus líneas de flujo de control.

Para validar un programa concurrente se requiere comprobar los mismos aspectos que en los programas secuenciales, pero con los siguientes nuevos aspectos:

•Las sentencias se pueden validar individualmente solo si no están acopladas por variables compartidas.

•Cuando existen variables compartidas, los efectos de interferencia entre las sentencias concurrentes pueden ser muy variados y la validación es muy difícil.

•Siempre que la secuencialidad entre tareas se lleve a cabo por sentencias explícitas de sincronización, el tiempo es un elemento que no influye sobre el resultado.

PROGRAMAS DE TIEMPO REAL.

- El orden de ejecución de las sentencias está establecido por las líneas de flujo de control y por los eventos externos.
- Los eventos externos no están condicionados por los criterios de sincronización establecidos en el programa.
- El tiempo físico en que se ejecutan las sentencias es parte de la funcionalidad del programa.
- La validación de los programas de tiempo real es muy compleja, requiere disponer del entorno o de un emulador.

Notas:

El orden de las sentencias es fijado por la línea (o las líneas) de flujo de control, y también por eventos asíncronos que recibe del entorno externo.

La interferencia del entorno con el programa de tiempo real se modela mediante eventos que se generan con independencia del programa. Los eventos externos no están condicionados por normas de sincronización establecidas internamente en el programa.

En los programas de tiempo real, el tiempo físico (tiempo real) que rige la evolución del entorno tiene una importancia capital al definir la funcionalidad del programa.

La validación de los programas de tiempo real es muy compleja. Para llevarse a cabo, requiere disponer de, o bien un entorno real, o bien un entorno simulado.

APLICACIONES DE LOS PROGRAMAS CONCURRENTES

- Aplicaciones clásicas:
  - Programación de sistemas multicomputadores.
  - Sistemas operativos.
  - Control y monitorización de sistemas físicos reales.
- Aplicaciones actuales:
  - Servicios WEB.
  - Sistemas multimedia.
  - Cálculo numérico.
  - Procesamientos entrada/salida.
  - Simulación de sistemas dinámicos.
  - Interacción operador/máquina (GUIs)
  - Tecnologías de componentes.
  - Código móvil.
  - Sistemas embebidos.

Notas:

Tradicionalmente existen tres tipos de **aplicaciones** en las que se ha utilizado los programas concurrentes:

•Los programas concurrentes que se necesitan para programar las **arquitecturas multicomputadoras**.

•Los **sistemas operativos** para multiprogramación tienen por objeto la ejecución concurrente de múltiples programas compartiendo, CPU, periferia, y sistema de almacenamiento de información.

•En los programas dedicados a **control o supervisión de entornos físicos reales** (que tienen naturaleza concurrente y no determinista), su diseño basado en un programa concurrente conduce a una mayor modularidad y claridad de diseño.

La naturaleza y los modelos de interacción entre procesos de un programa concurrente, fueron estudiados y descritos por Dijkstra (1968), Brinch Hansen (1973) y Hoare (1974). Estos trabajos constituyeron los principios en que se basaron los sistemas operativos multiproceso de la década de los 70 y 80.

VENTAJAS DE LA PROGRAMACION CONCURRENTE.

- Proporciona el modelo más simple y natural de concebir muchas aplicaciones.
- Facilita el diseño orientado a objeto de las aplicaciones, ya que los objetos reales son concurrentes.
- Hace posible compartir recursos y subsistema complejos.
- En sistemas monoprocesador permite optimizar el uso de los recursos.
- Facilita la programación de tiempo real, ya que se pueden concebir como procesos cuya ejecución se planifican de acuerdo con la urgencia.
- Permite reducir los tiempos de ejecución sobre plataformas multiprocesadoras.
- Facilita la realización de programas fiables por despliegue dinámico de los procesos en los procesadores.

Notas:

Los beneficios que se obtienen al adoptar un modelo de programa concurrente son:

•El programa concurrente es un modelo mas natural en la mayoría de las aplicaciones del mundo real, y que refleja mejor la concurrencia que se produce en el dominio del problema. Un diseño que haga énfasis en la simultaneidad de los procesos, conduce a una mayor simplicidad y facilidad de compresión.

•Estructurar un programa como conjunto de procesos concurrentes interactuantes, genera una gran claridad sobre lo que cada proceso debe hacer y cuando debe hacerlo.

•La concepción del software como programa concurrente puede conducir a una reducción del tiempo de ejecución. Cuando se trata de un entorno monoprocesador, la concepción concurrente permite solapar los tiempos de entrada /salida o de acceso al disco de unos procesos con los tiempos de ejecución de CPU de otros procesos. Cuando el entorno es multiprocesador, la ejecución de los procesos es realmente simultánea en el tiempo, y esto reduce el tiempo de ejecución del programa.

•La concepción concurrente de un software permite una mayor flexibilidad de planificación (scheduling), y con ella, los procesos con plazos límites de ejecución más urgentes pueden ser ejecutados antes de otros procesos menos urgentes.

•La concepción concurrente del software permite un mejor modelado previo del comportamiento (performances) del programa, y en consecuencia un análisis más fiable de las diferentes opciones que requiera su diseño.

CONCEPTO DE SISTEMAS DE TIEMPO REAL.

- Características esenciales de un sistema de tiempo real:
  - El orden de ejecución de las operaciones depende de:
    - Los eventos externos que recibe del entorno.
    - Del tiempo que transcurre.
  - El cumplimiento de plazos temporales en las respuestas es parte de la especificación funcional.

- De acuerdo con la severidad de los requerimientos temporales:
  - Sistemas de tiempo real estricto (Hard real-time): El incumplimiento de un plazo es un fallo irrecuperable.
  - Sistemas de tiempo real laxos: (Soft real-time): Los requerimientos temporales se cumplen en promedio.

Notas:

Los aspectos esenciales de un sistema de tiempo real son:

•El orden de la ejecución de las sentencias es determinado por la estructura del propio programa, por el transcurrir del tiempo físico, o por los eventos recibidos del entorno externo al computador.

•El resultado de un cálculo concreto es correcto no solo si funcionalmente es correcto, sino también si se obtiene dentro del plazo de tiempo físico especificado.

De acuerdo con la severidad de los requerimientos temporales que se establecen, los sistemas de tiempo real se clasifican en:

•**Sistemas de tiempo real estrictos (Hard-real time):** Son aquellos que tienen establecidos plazos temporales estrictos de respuesta, de forma que si alguno de ellos no se satisface, se produce un fallo catastrófico irrecuperable. Estos son sistemas que deben diseñarse utilizando estrategias de peor caso.

•**Sistemas de tiempo real laxos (Soft-real time):** Son aquellos que aunque también requieren el cumplimiento de plazos temporales de repuesta, su incumplimiento no supone un fallo catastrófico irrecuperable, sino solo una reducción de las prestaciones del sistema, y posiblemente asociado con él, un incremento de la carga de computación que requiere la recuperación del fallo.

CARACTERÍSTICAS DE SISTEMAS DE TIEMPO REAL.

- Naturaleza empotrada.
- Fuerte interacción con el entorno.
- Restricciones temporales.
- Interacciones asíncronas.
- Naturaleza reactiva.

Notas:

Aspectos que suelen estar asociados con los sistemas de tiempo real, son:

•**Naturaleza empotrada:** Los programas de tiempo real se presentan en sistemas electrónicos empotrados en equipos físicos (Embedded system), tales como controladores de sistemas robotizados, equipos de control físicos, instrumentación inteligente, etc.. Muy frecuentemente, los programas de tiempo real deben estar romeados en el sistema.

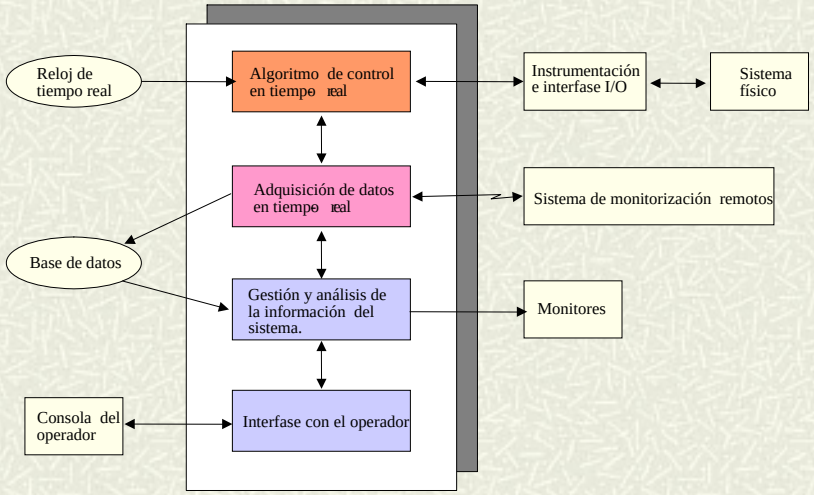
•**Fuerte interacción con el entorno:** La característica más relevante de un sistema de tiempo real es su necesidad de interacción con un entorno físico externo, que es de donde surge los requerimientos de tiempo real. Esto supone que deben ser soportados por computadores especiales, con una componente de entrada/salida muy relevante.

•**Restricciones temporales:** Ya sea de forma estricta o laxa, los sistemas deben responder a los eventos externos dentro de unos plazos especificados de tiempo físico.

•**Control de sistemas físicos:** Los sistemas de tiempo real corresponden a menudo con sistemas de control automáticos, en los que de acuerdo con los datos que se adquieren, el sistema toma decisiones de control de forma automática y sin intervención humana.

•**Naturaleza reactiva:** La mayoría de los sistemas de tiempo real son de naturaleza reactiva, esto es, son sistemas gobernados por eventos, en los que los diferentes componentes del software se activan de acuerdo con estímulos recibidos desde el entorno.

EJEMPLO DE SISTEMA DE TIEMPO REAL.



Estructura típica de un sistema computarizado de tiempo real.

Notas:

No todos los componentes de un sistema software de tiempo real tiene requerimientos de tiempo real. Solo un número frecuentemente reducido de ellos presentan requerimientos explícitos de respuesta temporal. Una estructura típica de un sistema de tiempo real es la que se muestra en la figura.

ENTORNOS HARDWARE PARA PROGRAMACIÓN CONCURRENTE.

- Un programa concurrente correcto debe funcionar en cualquier plataforma.
- La plataforma puede ser importante para establecer los modelos de fallo.
- Entornos habituales:
  - Entorno monoprocesador con multiprogramación.
  - Entorno multicomputador con memoria compartida.
  - Entorno distribuido.

Notas:

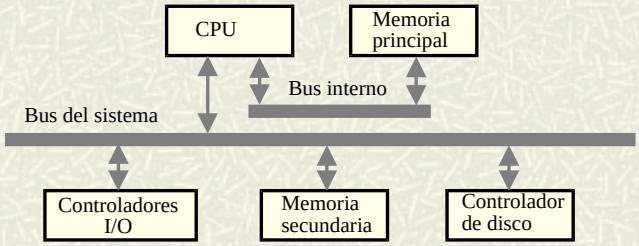
Una programa concurrente correcto debe ser funcionalmente independiente de la plataforma hardware sobre la que se instala y ejecuta. Sin embargo, buscando los posibles modelos de fallo que se pueden presentar en un programa concurrente, es conveniente considerar algunos conceptos relativos a las posibles arquitecturas hardware que se pueden utilizar.

Entornos hardware de programas concurrentes son:

- Entorno monoprocesador con multiprogramación.
- Entorno multicomputador con memoria compartida.
- Entorno distribuido.

ENTORNO MONOPROCESADOR CON MULTIPROGRAMACIÓN.

- Los procesos se ejecutan con concurrencia virtual.
- La aplicación concurrente no se ejecuta mas eficientemente.
- Razones por las que se utiliza:
  - Optimizar utilización de los recursos.
  - Servir a múltiple usuarios.
  - Conseguir un diseño mas simple y comprensible.



Componentes de un entorno monoprocesador

Notas:

Los procesos son ejecutados simultáneamente en concurrencia virtual bajo control del sistema operativo. Las sentencias de los diferentes procesos son ejecutadas de forma temporalmente entrelazadas, de acuerdo con una cierta política de planificación.

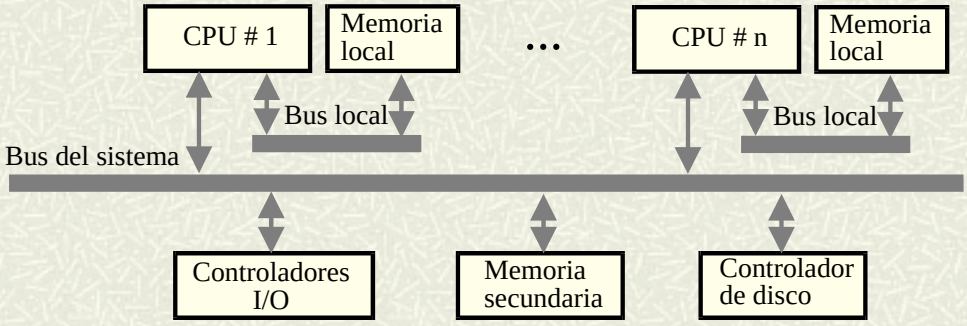
La ejecución de una aplicación concurrente en un entorno monoprocesador nunca es ejecutada más eficientemente que el correspondiente programa secuencial. Sin embargo, muy frecuentemente se utiliza la estructura concurrente en estos sistemas, por alguna de las siguientes razones:

- Optimizar la utilización de los recursos que se disponen
- Proveer un servicio interactivo a múltiples usuarios.
- Conseguir un diseño conceptualmente más comprensible y mantenible.

Este entorno es típico de sistemas minicomputadores en los que el hardware reside en una tarjeta en la que se encuentra la CPU y cierta cantidad de memoria RAM, y que a través de un bus local, se accede a tarjetas con memoria adicional y con controladores de dispositivos de entrada/salida y discos.

ENTORNO MULTIPROCESADOR CON MEMORIA COMPARTIDA.

- Los procesos se ejecutan con concurrencia física.
- El programa concurrente se ejecuta mas eficiente.
- Pueden existir problemas de coherencia entre memorias.



Componentes de un entorno multicomputador

Notas:

Es un sistema compuesto por varios procesadores instalados sobre un mismo armario físico y que a través de un bus del sistema, comparten recursos y periféricos de entrada/salida, discos, y segmentos de memoria compartida.

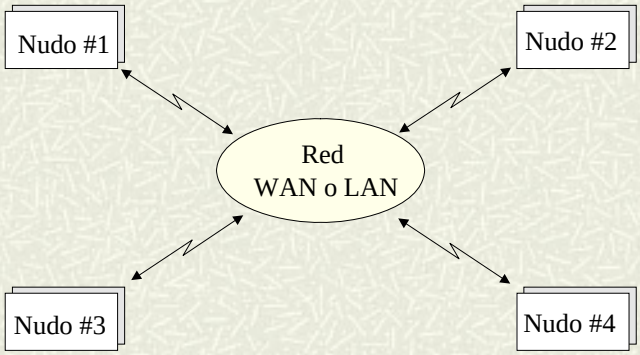
Los procesos que constituyen el programa concurrente, se ejecutan simultáneamente en los diferentes procesadores del sistema. Lo habitual es que en muchas fases de ejecución del programa, el número de procesadores sea menor que el número de procesos concurrentes, y sea necesario por ello, que varios de los procesos se ejecuten sobre un mismo procesador.

Como la concurrencia es real en estos sistemas, se consigue una disminución del tiempo de ejecución, respecto a la correspondiente versión secuencial.

En este entorno, los problemas en la ejecución de un programa concurrente, surgen como consecuencia de que dos procesos que se ejecutan simultáneamente en dos procesadores diferentes, tratan de acceder a un dato de la memoria compartida. En este caso, el controlador hardware del bus secuencia las acciones de acceso. Si el acceso al dato es una operación atómica, y las acciones de acceso están convenientemente sincronizadas en el programa concurrente, el arbitrio del controlador del bus es suficiente y el resultado será correcto. Por el contrario, si las acciones sobre los datos de memoria compartida requieren múltiples accesos independientes al bus, pueden presentarse problemas si las operaciones son de modificación del dato, y si cabe la posibilidad de que los múltiples accesos se entrelacen temporalmente.

SISTEMA DISTRIBUIDO.

- Existe concurrencia física.
- La comunicación es por mensajes.
- Problemas típicos:
  - Comunicación es costosa.
  - Coherencia de datos.
  - Entornos heterogéneos.



Entorno de programación distribuido.

Notas:

El entorno esta compuesto por un conjunto de computadores (monoprocesadores o multiprocesadores) dispersos geográficamente y que constituyen los nudos de la red distribuida. Los computadores no disponen entre si de espacio de memoria compartida, y deben intercomunicarse, intercambiando mensajes a través de una red de comunicación telemática.

Los procesos que constituyen el programa concurrente que se ejecuta sobre el sistema distribuido, deben intercambiar datos y señales de sincronismo, codificados como mensajes que se intercambian a través de la red de comunicaciones. Las operaciones de transferencia de mensajes y traducción de su formato de representación son muy costosas, y deben ser minimizadas para conseguir un rendimiento satisfactorio.

En estos sistemas, no se dispone de espacios de memoria compartida, sin embargo, es posible que en el diseño del programa concurrente se necesite disponer de variables compartidas por procesos ubicados en diferentes nudos. Esto se consigue, disponiendo de replicas del valor de las variables en los diferentes nudos, y a través de mecanismos complejos garantizar la coherencia entre la información replicada.

EJEMPLO DE PROGRAMA CONCURRENTE.

- Software embarcado de control de un coche.

Tarea	Periodo	Duración	% uso
Medida de velocidad	200 ms	40 ms	20%
Control presión carburante	400 ms	100 ms	25%
Control Válvula Carburador	800 ms	400 ms	50%

- El conjunto de tareas hacen uso del 95% de la capacidad del procesador.
- No es posible combinar secuencialmente la tres tareas.

Notas:

Considérese la estructura del programa que debe incluirse en el computador empotrado en el sistema de control de un coche. que sea capaz de llevar a cabo las siguientes tareas.

Tareas periódicas:

Considérese inicialmente que se requiere que el programa lleve a cabo los siguientes tres servicios periódicos:

Tarea M_V: Medida de velocidad (Cada 200 ms.)(Usa 40 ms de CPU)

Tarea P_C: Control de la presión de carburante. (Cada 400 ms.)(Usa 100 ms de CPU)

Tarea V_C: Control Válvula Carburador (Cada 800 ms.)(Usa 400 ms de CPU)

Dada la periodicidad de cada tarea, y la duración que cada una de ellas le supone al computador, es fácil comprobar que el computador tiene suficiente potencia de cálculo para llevarlas a cabo. Sobre un hiperperiodod de 800 ms, el uso de CPU es

Tarea M_V: (800/200) * 40 = 160 ms

Tarea P_C: (800/400) * 100 = 200 ms

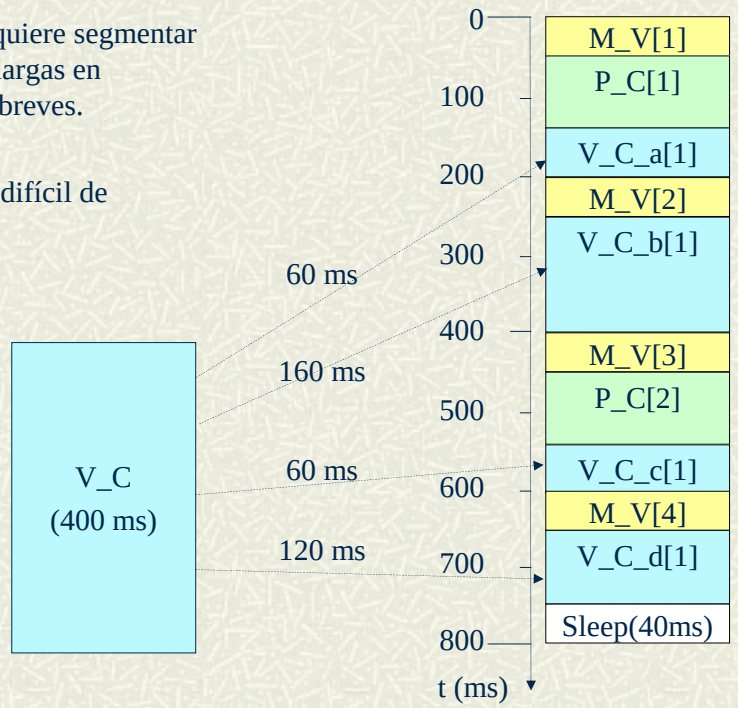
Tarea V_C: (800/800) * 400 = 400 ms

Total de CPU requerido = 760 ms

Sin embargo, una solución directa secuencial que combine las tres tareas no puede satisfacer los requerimientos establecidos.

EJEMPLO: SOLUCIÓN SECUENCIAL.

- La solución requiere segmentar las tareas mas largas en secciones mas breves.
- La solución es difícil de mantener.



Notas:

Las posibles soluciones secuenciales, se consiguen descomponiendo las tareas en subtareas con duración parcial inferior, y entrelazar temporalmente estas. Una solución posible es descomponer la tarea V_C en cuatro subtareas: V_C_1 (de duración 60 ms), V_C_2 (de duración 160 ms) y V_C_3 (de duración 60 ms) y V_C_4 (de duración 120 ms) y planificar su ejecución de acuerdo con el siguiente programa:

```
repeat
    M_V; P_C; V_C_a; M_V; V_C_b; M_V; P_C; V_C_c; M_V; V_C_d;
    sleep (40 ms);
forever;
```

Claramente, esta estrategia de programación es muy pobre y de muy difícil mantenimiento.

EJEMPLO: TAREA APERIÓDICA.

- La tarea C_S tiene la función de bloquear el cinturón de seguridad si el sensor de aceleración supera un umbral:

Naturaleza:	Aperiódica
Plazo de respuesta:	30 ms
Uso de CPU:	20 ms
Intervalo mínimo:	800 ms

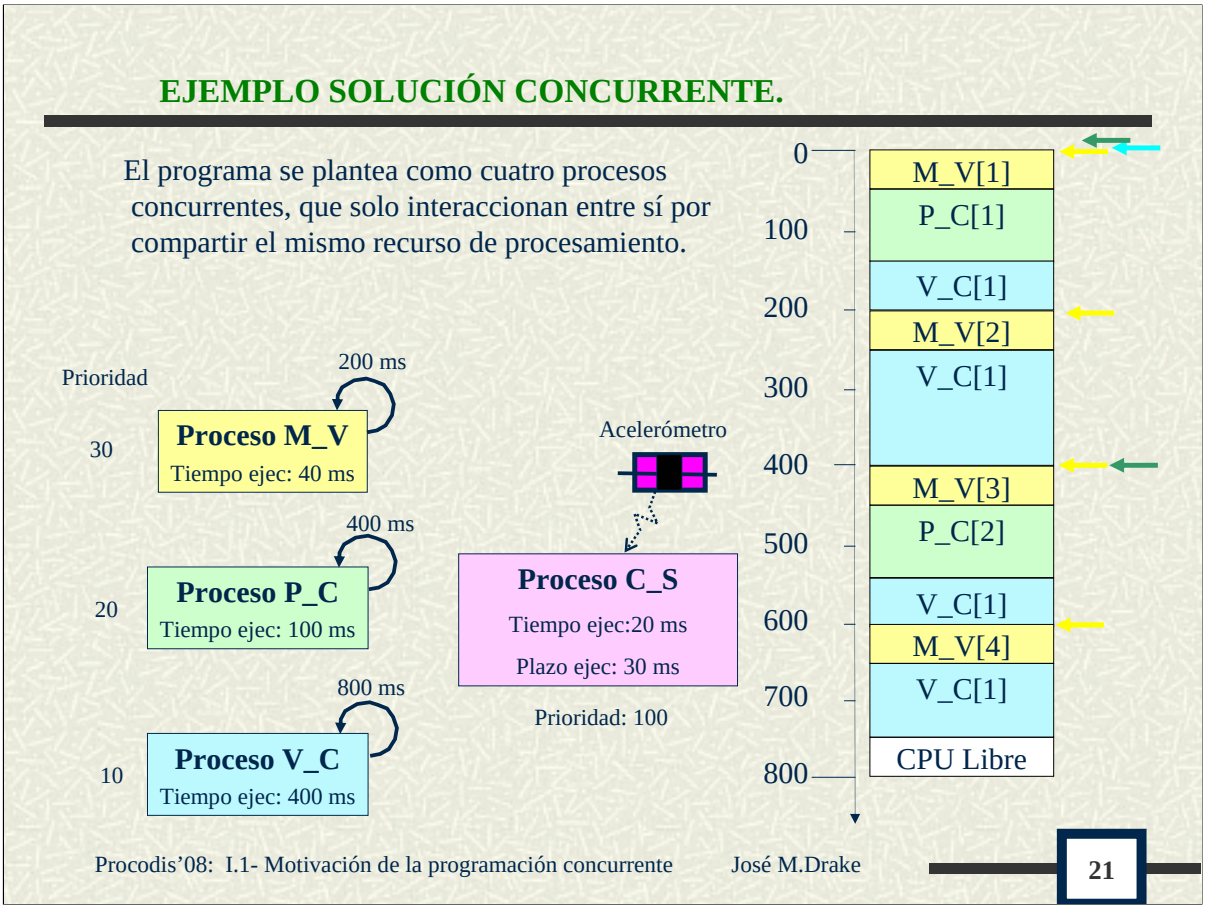
- Incorporar esta tarea a la estructura secuencial es muy difícil.
- La solución atender el evento en una rutina de interrupción.

Notas:

Consideremos ahora que el mismo sistema debe atender una nueva tarea de control de la fijación del cinturón de seguridad (C_S). Esta tarea no debe estar activa habitualmente, solo cuando un sistema inercial detecta una aceleración negativa superior a un umbral, debe operar el proceso para fijar la posición del cinturón y así mismo, cuando se detecta que esta condición desaparece, debe liberar de nuevo el cinturón. Las características temporales que definen esta tarea aperiódica son:

Retraso de respuesta desde el evento :	30 ms.
Duración de la tarea:	20 ms.
Intervalo mínimo entre eventos:	800 ms.

Incorporar esta nueva tarea en el código del programa secuencial anterior, es muy complicado. La solución tradicional a este tipo de problema, se basa en atender el evento dentro de una rutina de interrupción. Lo cual es en cierto modo una forma heterodoxa de concurrencia.



Notas:

```
program Control_coche;

  process P_M_V;                  (* Proceso de medida de la velocidad *)
  begin repeat  M_V;  sleep(160 ms);  forever; end;

  process P_P_C;                  (* Proceso de control de la presión de carburante *)
  begin repeat M_V; sleep(300 ms); forever; end;

  process P_V_C;                  (* Proceso de control de la válvula del carburador *)
  begin repeat M_V; sleep(400 ms); forever; end;

  process P_C_S;                  (* Proceso de atención del control del cinturón *)
  begin repeat  wait evento; C_S; forever; end;

begin                             (* Programa principal *)
   cobegin P_M_V; P_P_C; P_V_C;  P_C_S;  coend
end.
```

Ejemplo de programa concurrente

```
program Control_coche;

  process P_MedidaVelocidad;                (* Proceso de medida de la velocidad *)
  begin repeat  M_V;  sleep(160 ms);  forever; end;

  process P_PresiónCarburante;              (* Proceso de control de la presión de carburante *)
  begin repeat M_V; sleep(300 ms); forever; end;

  process P_ControlValvulaCarburador;       (* Proceso de control de la válvula del carburador *)
  begin repeat M_V; sleep(400 ms); forever; end;

  process P_ControlCinturan;                (* Proceso de atención del control del cinturón *)
  begin repeat  wait evento; C_S; forever; end;

begin                       (* Programa principal *)
   cobegin
            P_MedidaVelocidad;
            P_PresionCarburante;
            P_ControlValvulaCarburador;
            P_ControlCinturón;
     coend;
end.
```


Notas: