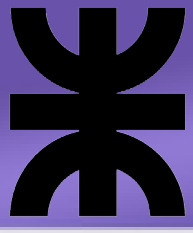


Sistemas de Tiempo Real

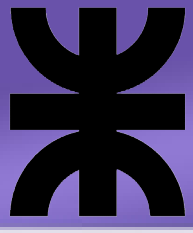
Introducción

- Distintos tipos de Sistemas:
  - Sistemas de uso general (ofimática, CAD)
  - Sistemas Multimedia (audio/video/juegos)
  - Sistemas de control (horno, motor, proceso)
- Implementación:
  - Sistemas de uso general → Linux/Windows
  - Sistemas Multimedia → Streaming
  - Sistemas de control → Real Time Systems

Introducción

- Sistemas de Tiempo Real:
  - Un sistema informático en tiempo real es aquel en el que la **corrección del resultado*** depende tanto de su validez lógica como del instante en que se produce.
- Determinismo:
  - El tiempo de ejecución de los programas de tiempo real debe estar acotado en el caso más desfavorable para cumplir las restricciones temporales. No debe ser necesariamente rápido.

** Cualidad de correcto*



Introducción

Velocidad o rapidez ≠ Determinismo

- Acotado en tiempo para el caso más desfavorable.
- Su velocidad debe ser acorde con la planta con la que se está interactuando.

Introducción

En los sistemas de control existen cuatro niveles jerárquicos de software:

- Adquisición de datos / actuadores
  - Ejecuta el software cuando llega una interrupción
- Algoritmos de control (PID)
  - Algoritmo de control digital, toma datos y envia a los actuadores. Se ejecuta periodicamente (periodo de muestreo)

Introducción

- Algoritmos de supervisión (trayectorias)
  - Capa opcional de control a un nivel superior. Se ejecutan también periódicamente, aunque normalmente con un periodo de muestreo mayor que los algoritmos de control de bajo nivel.
- Interfaz de usuario, registro de datos, etc.
  - No existen restricciones temporales estrictas, por lo que se ejecuta cuando los niveles inferiores no tienen nada que hacer.

Introducción

- El paralelismo es una característica inherente de los sistemas en tiempo real.
- Mientras un programa convencional se divide en funciones que se ejecutan según un orden preestablecido, un sistema en tiempo real se divide en tareas que se ejecutan en “paralelo”.
- La ejecución en paralelo se consigue como la sucesión rápida de actividades secuenciales (multitasking)

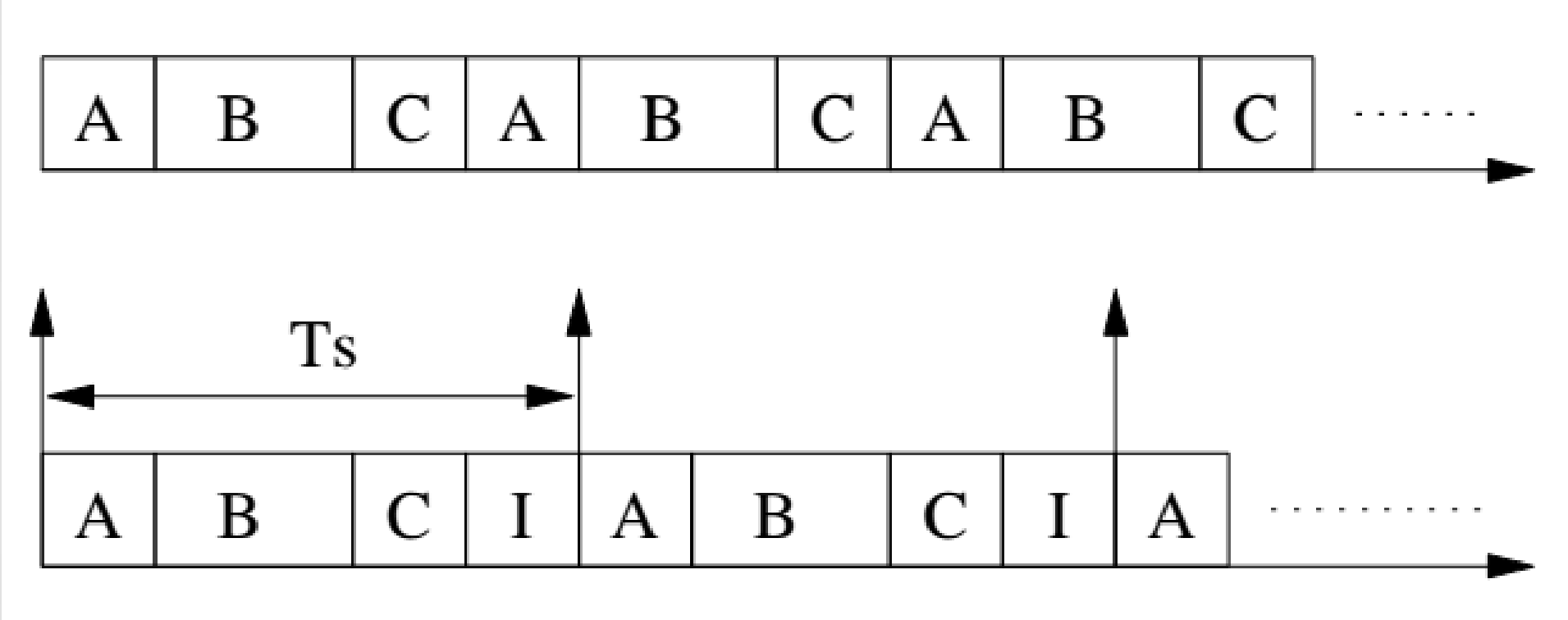
Introducción

- Contamos con varias técnicas para ejecutar tareas en paralelo:
  - Procesamiento secuencial (bucle de scan)
  - Interrupciones (background / foreground)
  - Multitarea cooperativo
  - Multitarea expropiativo (preemptive)

Procesamiento secuencial

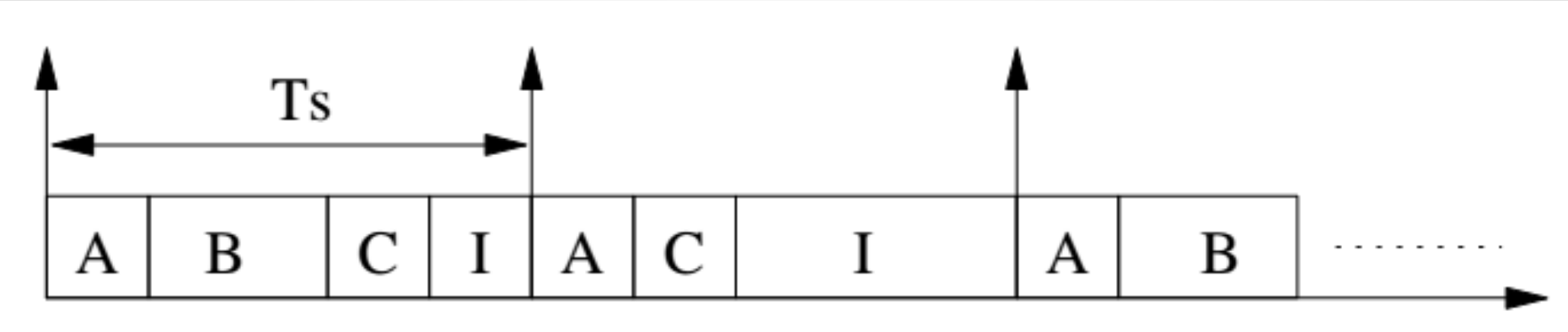
- Las tareas se ejecutan una tras otra en orden preestablecido.
- Condición:
  - No debe bloquearse esperando evento externo asíncrono (¿cómo maneja una lectura de un conversor A/D?)
  - La suma de todos los tiempos máximos de ejecución de las tareas debe ser menor al Ts (tiempo de muestreo)
  - Si necesitamos un Ts exacto y determinado se debe utilizar un temporizador

Procesamiento secuencial



Procesamiento secuencial

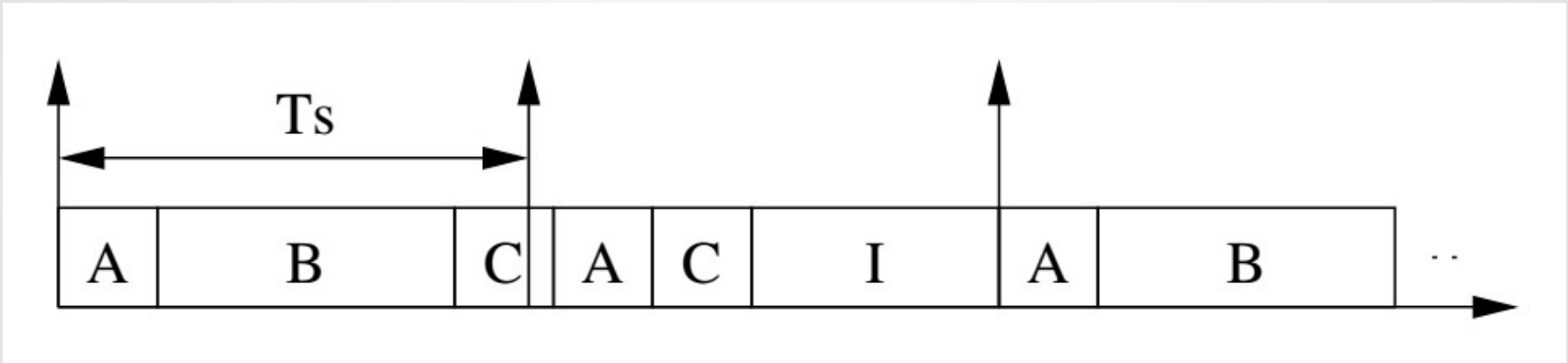
- Tareas con distinto período de muestreo (Ts)



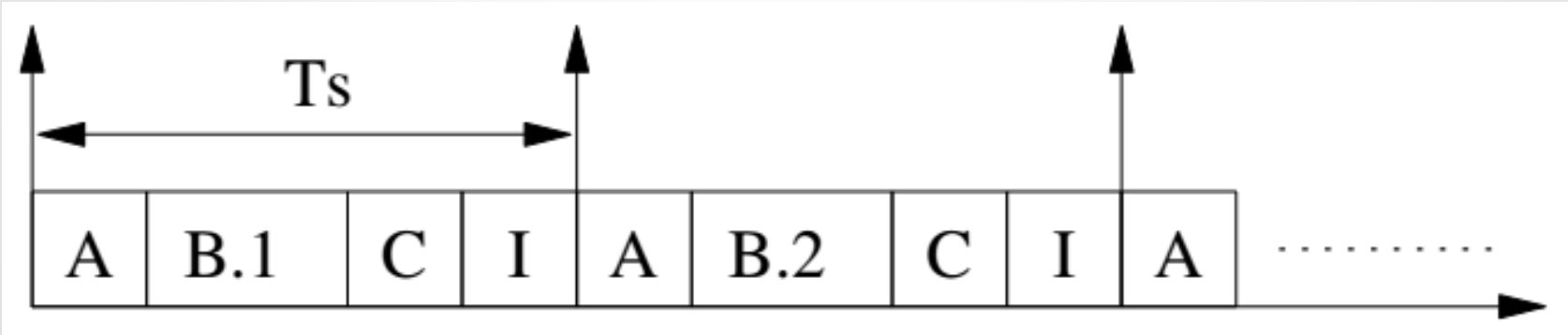
- La tarea B tiene un período de 2 Ts

Procesamiento secuencial

- La tarea B tiene un período de ejecución demasiado largo:

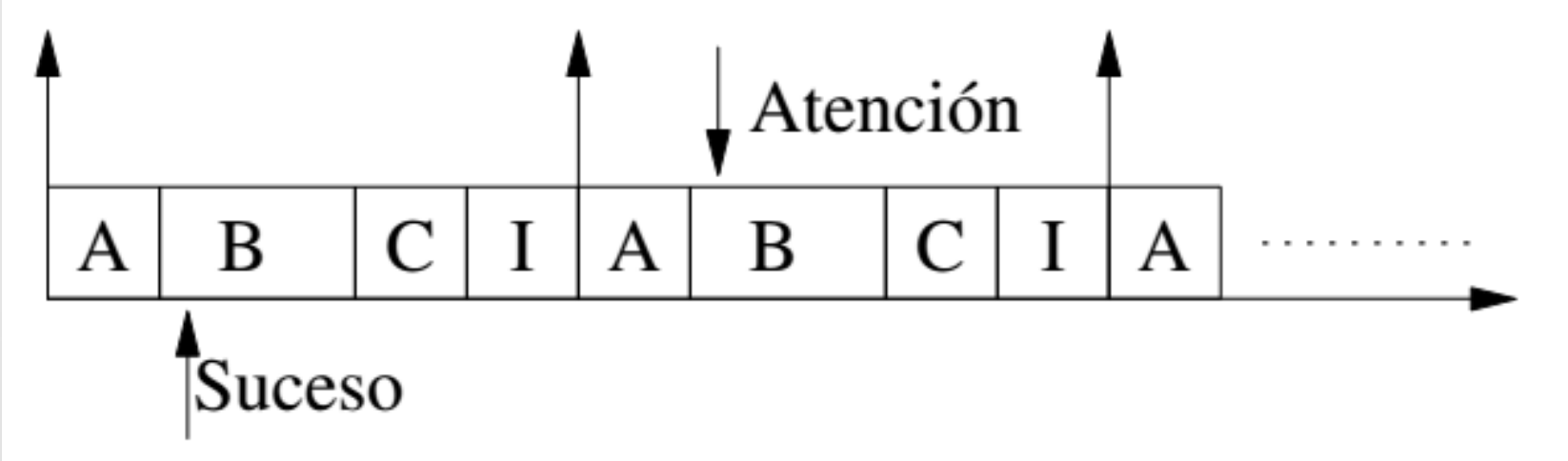


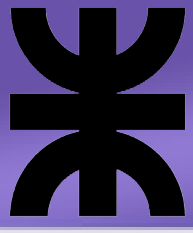
- División de la tarea (o cambio de procesador?):



Procesamiento secuencial

- Otros Problemas:
  - Varias tareas con Ts NO múltiplos de Ts básico y varios tiempos de ejecución
  - Latencia (tiempo entre evento externo y ejecución de la tarea que lo implementa) → Ts en el peor caso

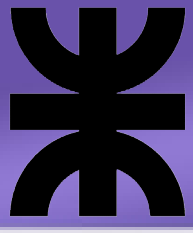




Procesamiento secuencial

Resumen

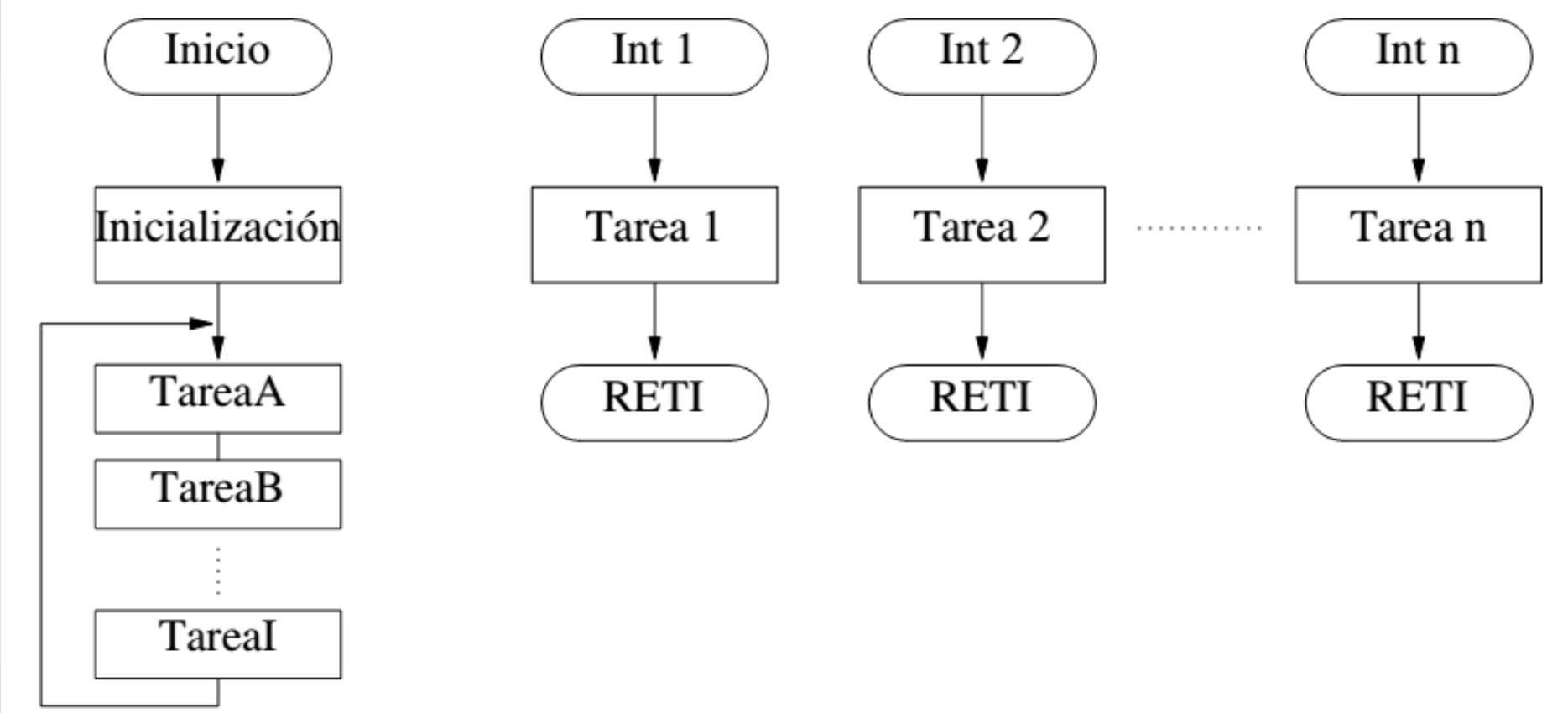
- Ventajas
  - Fácil implementación y depuración
  - Fácil compartir datos
  - Mayor eficiencia (no hay cambio de contexto)
- Desventajas:
  - Latencia asociada al bucle scan
  - Difícil para tareas con distinto Ts
  - Solo para sistemas sencillos

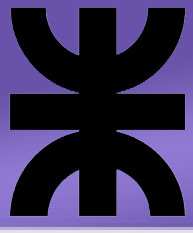


Interrupciones

- Usa interrupciones para esperar eventos externos asíncronos y disminuir latencia
- Existen dos tipos de tareas:
  - Foreground
  - Background (atención de interrupción)

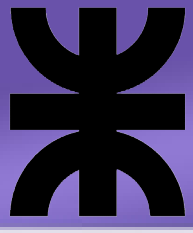
Interrupciones





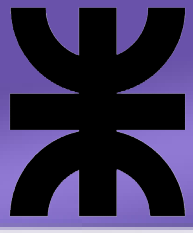
Interrupciones

- Ventajas:
  - Latencia → tiempo máximo tarea background (deshabilitación interrupciones)
  - No tiene problemas con distintos Ts



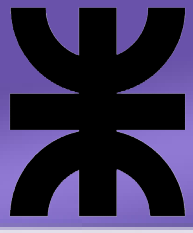
Interrupciones

- Desventajas:
  - Necesita soporte interrupciones de hardware
  - Cada tarea de background debe tener asociada una interrupción
  - Rutinas background deben ser cortas (se lleva parte de la tarea de background a foreground)
  - Problemas de concurrencia
  - Depuración más compleja
  - Incoherencia de datos



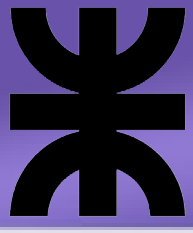
RTOS

- ¿Qué es un RTOS?
  - Sistema Operativo para Sistemas de Tiempo Real
  - Herramientas para que los programas cumplan compromisos temporales definidos por el programador.
  - Sistema con comportamiento determinista.
  - Se emplea cuando hay que administrar varias tareas simultáneas con plazos de tiempo estrictos.



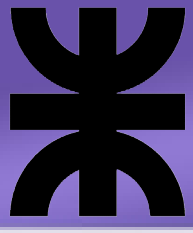
RTOS

- ¿Por qué usar un RTOS?
  - Multitarea: simplifica la escritura del código.
  - Escalabilidad: Al tener ejecución concurrente de tareas se pueden agregar las que hagan falta (insertarlas correctamente en el esquema de ejecución del sistema)
  - Reutilización de código: Si las tareas tienen pocas o ninguna dependencia, es fácil incorporarlas a otras aplicaciones.
  - No manejar el tiempo a mano. Los RTOS se encargan del manejo de timers y esperas, ayudando al programador



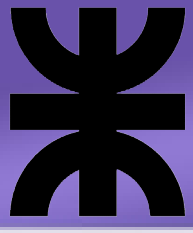
RTOS

- ¿Por qué usar un RTOS?
  - Tarea Idle. Si no hay tareas que necesiten procesador, se ejecuta una llamanda idle
  - Cumplir requisitos temporales estrictos. Los RTOS aseguran que ocurrido el evento, la respuesta suceda en un tiempo acotado. (Cuidado, esto no lo hace por si mismo, solo da herramientas al programador)



RTOS

- ¿Por que NO usar RTOS?
  - Uso de tiempo de CPU para cambio de tareas
  - Uso de memoria de código para implementar RTOS
  - Uso de memoria de datos
  - El programador NO decide cuando ejecutar cada tarea, queda en manos del planificador.
  - Con errores en las reglas de ejecución, puede hacer que los eventos se procesen fuera de tiempo o ni se ejecuten

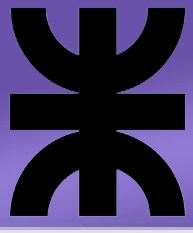


RTOS

- Tareas
  - Unidad básica de planificación.
  - Funciones escritas como si cada una fuera la única que utiliza la CPU.
  - Las tareas NO deben terminar nunca.
  - Si deja de ser necesaria, se elimina explícitamente
  - Se inicializan indicando su prioridad, memoria, función, etc.

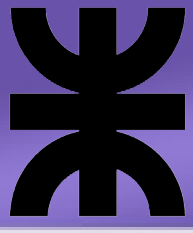
RTOS


La tarea necesita
un suceso
externo a ella
Ejecución
Hay una tarea
con mayor
prioridad
Esta es la
tarea conmayor
prioridad
Bloqueada
Lista
Ha ocurrido el suceso



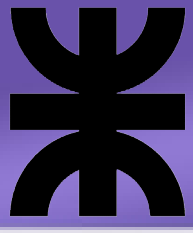
Planificadores

- **Planificador cooperativo**: las tareas de 1er plano son las responsables de llamar al planificador:
  - Cuando terminan
  - Cuando llevan demasiado tiempo ejecutando (yield)
  - Cuando se bloquean
- **Planificador expropiativo**: un temporizador genera una interrupción y la rutina de atención de interrupción ejecuta el planificador periódicamente
- Se elije la tarea lista de mayor prioridad



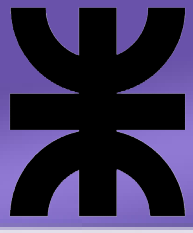
Planificadores

- Planificador cooperativo:
  - No hay problemas de concurrencia (coherencia)
  - Mas sencilla la codificación
  - No sabe la latencia de las tareas 1er plano
- Planificador expropiativo:
  - Baja latencia
  - La programación de tareas debe ser cuidadosa
  - Gestionar el acceso a recursos compartidos
  - Mas complicada la implementación



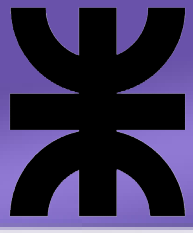
Tareas

- Tipo de tareas:
  - Tareas Periódicas: con frecuencia determinada. Generalmente se usa un timer y un contador. Ejemplo: destello de led.
  - Tareas Aperiódicas: con frecuencia intederminada. Tareas inactivas (bloqueadas) hasta que ocurre el evento de interés. Por ejemplo una parada de emergencia.
  - Tareas Contínuas: régimen permanente. Ejemplo: muestrear un buffer de recepción en espera de datos para procesar. Estas tareas deben tener prioridad menor que el resto, ya que en caso contrario podrían impedir su ejecución



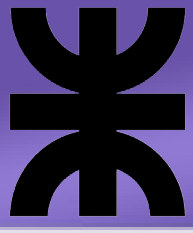
Tareas

- Tareas Periódicas:
  - La mayor parte de su tiempo en el estado bloqueado hasta que expira su tiempo de bloqueo, en este momento pasa al estado Listo.
  - Alta prioridad, para tener baja latencia.
- Tareas Aperiódicas:
  - Nos interesa que permanezca bloqueada hasta que ocurra un evento determinado (interno o externo)
  - Queremos que la tarea ceda el CPU a las de menor prioridad, ya que no va a estar haciendo nada



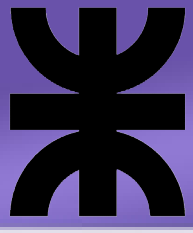
Sincronización

- Recursos de Sincronización:
  - Semáforos:
    - Su función es restringir el acceso a una sección particular del programa
    - Dos primitivas, incrementar y decrementar
    - Semáforos binarios
    - En algunos RTOS se define máximo tiempo de bloqueo



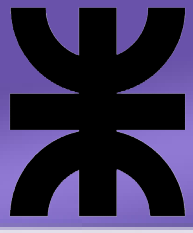
Intercambio de datos

- Recursos para Intercambio de datos:
  - Tareas son funciones de C. Aplican reglas de visibilidad.
  - Variables globales como mecanismo de intercambio.
  - Problemas de acceso al recurso: Tareas aperiódicas que esperan un dato para procesarlo
  - Necesidad de sincronización



Intercambio de datos

- Recursos para Intercambio de datos (cont.):
  - Colas de mensaje:
  - Son visibles por todas las tareas (deben ser creadas en forma global)
  - Incorporan el mecanismo de sincronización.
  - De la misma manera que un semáforo, se puede bloquear al leer / escribir datos de una cola
  - En algunos RTOS se define máximo tiempo de bloqueo



Recursos

- Administración de recursos:
  - Se debe aplicar exclusión mutua al código del recurso:
  - Deshabilitar Interrupciones
  - Suspender el planificador
  - Aumentar la prioridad de la tarea que usa el recurso
  - Uso de semáforo o mutex