

RESUMEN TEORÍA SISTEMAS EMPOTRADOS Y DE TIEMPO REAL:

ÍNDICE DE CONTENIDOS:

TEMA 1: INTRODUCCIÓN A SISTEMAS EMPOTRADOS Y DE TIEMPO REAL

TEMA 2: PLANIFICACIÓN EN SISTEMAS DE TIEMPO REAL

TEMA 3: SISTEMAS OPERATIVOS EN TIEMPO REAL (RTOS)

TEMA 4: MICROCONTROLADORES

TEMA 5: ARDUINO

TEMA 6: MICROROS Y FREERTOS EN ARDUINO

TEMA 1: INTRODUCCIÓN A SISTEMAS EMPOTRADOS Y DE TIEMPO REAL

Sistema empotrado: Sistema de comunicación diseñado para realizar una o algunas pocas funciones de control (Arduino, Esp8266 y LoRa).

Ventajas: Costes reducidos, diseño modular, corto tiempo de respuesta, facilidad en la integración y accesibilidad.

Inconvenientes: Cifrado débil, diseños cerrados, puertos de entrada y salida, seguridad descuidada y ataques DOS/DDoS.

Sistema de tiempo real: Sistema informático en el que se aseguran los tiempos de ejecución por cada tarea.

Características: **Determinismo** (asegurar que la tarea siempre dura lo mismo), **responsividad** (tiempo que tarda en ejecutarse la tarea desde que se le ordenó), **confiabilidad** (el sistema debe seguir funcionando a pesar de catástrofes o fallos), **operación a prueba de fallos** (en caso de fallo, preservar datos y ejecutar tareas de mayor prioridad).

Clasificación: **Estricto** (necesidad de respuesta dentro de un límite, control de vuelo), **no estricto** (se permiten pérdidas aunque deben cumplirse, sistema de adquisición de datos), **firme** (se permiten pérdidas pero no implica beneficios al descartarse la respuesta, sistema multimedia).

Ejemplos: Computador de navegación del Apolo 11, Rover Perseverance, automoción tradicional y moderna, aviación, Boeing 737 MAX-8.

TEMA 2: PLANIFICACIÓN EN SISTEMAS DE TIEMPO REAL

STR según flujo de ejecución: **Monotarea** (un flujo de ejecución, bucle infinito, muestreo de entradas, sencillos pero poco flexibles, difícil añadir nueva funcionalidad), **multitarea** (conjunto de tareas, procesos concurrentes, control de recursos, comunicación entre tareas).

Tareas según forma de ejecución: **Periódicas** (activaciones repetidas a intervalos de tiempo, $0 < t_{\text{cómputo}} < p_{\text{ejecución}} < p_{\text{activación}}$), **esporádicas** (activaciones aleatorias con requisitos temporales críticos, separación mínima p' , plazo de ejecución d' , tiempo de cómputo c'), **aperiódicas** (sin requisitos temporales rígidos).

Tareas según semántica: **Críticas** (su fallo puede ser catastrófico), **opcionales** (refinan tareas críticas o monitorizan el estado del sistema).

Planificador de tareas: Subsistema encargado de decidir qué tareas se ejecutan en el procesador y en qué momento.

Tipos de planificadores: **Cíclicos** (orden y secuencia de tareas a mano), **prioridades estáticas** (RM \Rightarrow tareas con prioridad alta, DM \Rightarrow tareas con plazo más corto), **prioridades dinámicas** (EDF y LLF).

Planificación cíclica de tareas: **Ventajas** (determinista y predecible, no hay sobrecarga, implementación sencilla, procesos por turnos, ejecución según programador, sistema operativo por ejecución cíclica), **desventajas** (diseño de planes complejo, no son fáciles de mantener y actualizar).

Una condición necesaria para que el conjunto se planificable el conjunto de procesos periódicos $\{P_1, P_2, \dots, P_n\}$ con requisitos temporales representados por p_i, c_i, d_i es que la utilización del procesador U sea menor o

igual que uno ($U = \sum_{i=1}^k \frac{c_i}{p_i} \leq 1$).

No es un planificador óptimo, excepto cuando las tareas periódicas son simples: $\sum_{i=1}^N \frac{c_i}{p_i} < N \left(2^{\frac{1}{N}} - 1 \right)$.

En la política de planificación de Rate Monotonic, todos los deadlines de N tareas periódicas están

garantizados si: $U \leq N \left(2^{\frac{1}{N}} - 1 \right)$.

Para calcular la utilización mínima garantizada para N tareas: $U_o = N \left(2^{\frac{1}{N}} - 1 \right)$.

Utilización mínima garantizada: $U \leq U_o(N)$ (deadlines están garantizados), $U > 1$ (deadlines no están garantizados), $U > U_o(N) \leq 1$ (no se sabe nada, hay que realizar el cronograma temporal).

TEMA 3: SISTEMAS OPERATIVOS EN TIEMPO REAL (RTOS)

RTOS: Sistema operativo multitarea diseñado para ejecutar aplicaciones de tiempo real.

Características: Comportamiento predecible y determinista, garantiza la ejecución de la tarea en un tiempo determinado, implementa herencia de prioridades y existen dos tipos (Soft y Hard RTOS).

Funciones básicas: Manejo de tareas, interrupciones y tiempo, sincronización y planificación de tareas, administración de memoria y excepciones.

RTOS ideal: Menor latencia en cambio de contexto y tratar interrupciones, menor tamaño del kernel, mayor flexibilidad en algoritmos y rango de arquitectura, y soporte para mono o multi núcleo.

Latencia: Intervalo de tiempo entre estímulo y respuesta.

Tipos: **Interrupción** (tiempo entre que se genera y es atendida), **planificación** (tiempo entre que debe ejecutarse y realmente se ejecuta).

CFS (Completely Fair Scheduler): Maximiza el uso de la CPU permitiendo el uso interactivo de la máquina (minimiza latencia en procesos interactivos), **3 colas de ejecución** (SCHED_NORMAL, SCHED_BATCH, SCHED_IDLE), **3 políticas de tiempo real** (SCHED_FIFO, SCHED_RR, SCHED_DEADLINE).

Valor de prioridad: **Usuario** (100-139), **tiempo real** (1-99).

Nice: [-20,19], prioridad = 20+Nice.

Reloj hardware: Mantiene la gestión del tiempo cuando el equipo está apagado.

Reloj Software: Realtime (reloj del sistema, se puede corregir la fecha), monotonic (sin ajustes, avanza por ticks y usado para medir duraciones entre eventos).

RT Linux: Permite tareas de tiempo real y regulares, minimiza latencia y código del Kernel, SO de tiempo real estricto, extensiones SMP, fuerza interrupción de threads, permiten dormir a los locks, elimina secciones críticas, herencia de prioridad, sistema de tiempo real duro, latencia estable con picos.

Herramientas para medir la latencia: **Cyclicttest** (mide latencia en sistemas operativos y a nivel de planificador), **stress** (genera sobrecarga computacional en el sistema), **hackbench** (testea y estresa el planificador), **bonnie++** (testea I/O y sistemas de ficheros).

TEMA 4: MICROCONTROLADORES

Microcontrolador: Circuito integrado programable capaz de ejecutar las órdenes grabadas en su memoria compuesta por varios bloques que cumplen una tarea específica, y que incluye CPU, memoria y periféricos de entrada/salida.

Características: Realizan tareas específicas, disminuyen coste económico y consumo de energía.

Componentes: CPU, ALU, memoria de programa y de datos, puertos de entrada/salida, timers y contadores, interrupciones, conversión analógico/digital, interfaz de comunicación.

Recursos auxiliares: Circuito de reloj, PWM, puertos de comunicación, sistema de protección, estados de reposo, watchdog.

Arquitectura Von Neumann: Una sola memoria de datos e instrucciones y un único bus de dirección, datos y control.

Arquitectura Harvard: Memoria de datos y de programas, buses y memoria segregados, menos instrucciones RISC.

Clasificación según bits: 4/8 bits, 16/32 bits.

Clasificación según memoria: **ROM** (programa ya grabado en la memoria), **PROM** (solo se programa una vez), **EPROM** (memoria programable), **EEPROM** (sustituto de EPROM), memoria flash.

Técnicas de alto rendimiento: **Harvard** (memorias separadas, buses distintos y paralelismo), **RISC** (instrucciones reducidas y de tamaño fijo), **pipelining** (instrucción en varias etapas, ejecutar etapas de varias instrucciones a la vez).

Ejemplos: PIC 16F877A, ATmega328, Attiny85, ESP8266, STM32F103C8T6, STM32.

TEMA 5: ARDUINO

Arduino: Compañía de hardware y software libre (conjunción de processing y wiring).

Características: Circuitos/placas de bajo coste, basada en AVR Atmel-8bits, diversas cantidades de memoria flash, pines y funciones.

Shield: Placa de circuito impreso que se coloca sobre la placa Arduino para extender su funcionalidad.

setup(): Función que se utiliza para inicializar datos o puertos, y que se ejecuta una única vez al inicio del programa.

loop(): El contenido de esta función se ejecuta repetidamente mientras la placa arduino sigue encendida.

Bootloader: Código que reside en un espacio reservado de la memoria del MCU de Arduino.

Serial.begin(SPEED): Configurar velocidad del puerto serie.

Serial.println(MESSAGE): Imprimir mensajes.

Serial.println(value): Imprimir valor por el puerto serie.

pinMode(pin,mode): Configurar pin digital para leer o escribir.

digitalRead(pin): Lee un valor digital HIGH o LOW en un pin establecido como entrada.

digitalWrite(pin,value): Escribe un valor digital HIGH o LOW en un pin establecido como salida.

Señal digital: Tipo de señal en que cada símbolo que codifica se puede analizar a través de valores discretos.

Señal analógica: Los valores de la tensión del voltaje varían constantemente durante el tiempo.

PWM: Simula comportamientos analógicos en los pines digitales (duty cycle).

analogWrite(): Genera una señal PWM en un pin digital [0-255].

millis(): Número de milisegundos desde que la placa empezó a ejecutar el sketch actual.

micros(): Número de microsegundos desde que la placa empezó a ejecutar el sketch actual.

delay(): Pausa el sketch durante la cantidad de milisegundos especificados (no libera la CPU del microcontrolador).

delayMicroseconds(): delay en el rango [3-16383].

Sensores: LED, Botón/Switch, HC-SR04 (pulseIn devuelve el tiempo hasta que aparece un pulso), LCD 16x2.

Watchdog: Mecanismo de seguridad que provoca un reset del sistema, en caso de que éste se haya bloqueado o no responda al comportamiento deseado.

Incluir librería en el sketch: #include <avr/wdt.h>.

Desactivar el watchdog y configurarlo: wdt_disable(); wdt_enable(tiempo);.

Actualizar watchdog para no resetear: wdt_reset();.

Resistencia pull-up: La entrada está a HIGH hasta detectar pulsación (LOW).

Resistencia pull-down: La entrada está a LOW hasta detectar pulsación (HIGH).

Sensores: TMP36, DHT-11/20, LED RGB, PIR, zumbador.

ArduinoThread: No ejecuta threads en paralelo, planifica y maneja tareas periódicas, tiempos fijos o variables entre ejecuciones, organiza el código limpiamente y esconde la complejidad de cada thread, sin un thread se bloquea todo el programa también (primera aproximación a multitarea en mono-core y sin delays).

Polling: Sondear un estado de manera continua y constante (técnica ineficiente).

Interrupciones: Mecanismos del microcontrolador para responder a eventos (suspensión temporal -> ISR -> reanudación).

Interrupciones hardware: Cambios en los pines físicos de entrada (LOW, HIGH, CHANGE, RISING, FALLING), llamada (attachInterrupt(digitalPinToInterrupt(pin), ISR, mode);), anular interrupción (detachInterrupt(interrupt)), desactivar ejecución de interrupciones (noInterrupts()), reactivar interrupciones (interrupts()).

Interrupciones software: Timers y contadores, generadas pasado un tiempo determinado, sensibles a la precisión y exactitud del reloj interno del sistema.

Volatile: Directiva del compilador asociada a definición de variables.

Comunicación serie: Proceso de envío de datos de un bit a la vez, de forma secuencial, sobre un canal de comunicación o bus (asíncrona o síncrona).

Comunicación UART: Ventajas (2 líneas para envío y recepción, comunicación asíncrona, velocidades entre

300 bps y 460 kbps, mecanismo simple), **desventajas** (problemas a más de 15 metros, comunicación limitada a 2 dispositivos, velocidad acordada al inicio).

Comunicación I2C: **Ventajas** (mayor velocidad que comunicación serie, muy extendido, 2 líneas, múltiples líderes y seguidores, ACK/NACK permite confirmación de la transferencia), **desventajas** (más lento que SPI, hardware más complicado, dataframe limitado a 8 bits).

Comunicación SPI: **Ventajas** (mayor velocidad, full-dúplex, no está limitado a palabras de 8 bits, hardware más sencillo y multitud de sensores compatibles), **desventajas** (requiere más líneas, direccionamiento a través de líneas específicas, longitud del mensaje conocido por ambas partes, sin ACK en la transmisión, sólo un único nodo líder).

Otros tipos de comunicaciones: CANBUS (intercambio entre unidades de control electrónicas), Ethernet, Wi-Fi, MQTT, Node-Red (interconecta diferentes dispositivos del sistema).

TEMA 6: MICROROS Y FREERTOS EN ARDUINO

MicroROS (OFERA): Habilitar el comportamiento colaborativo entre robots con una alta capacidad de rendimiento y microcontroladores con grandes restricciones computacionales.

Objetivos: Integración sin problemas, facilidad en la portabilidad, asegurar mantenimiento a largo plazo.

Características: Crea nodos, publicador/subscriptor, ejecutar en diferentes RTOS y MCUs, compilación cruzada, QoS, uso eficiente de memoria, software libre, soporta microcontroladores de 32 bits.

FREERTOS: Kernel de tiempo real basado en hilos (tareas).

Beneficios: Abstracción de información del tiempo, escalabilidad, modularidad, desarrollo en equipo, tiempo "idle".

Características: "Pre-emptive", tareas con prioridad, colas, semáforos, tick/idle, interrupciones anidadas.

Tarea: xTaskCreate(Tarea,"Tarea",P.Res,Par,Pr,Punt);.