Co-funded by the
Erasmus+ Programme
of the European Union

Sistemas Embebidos

Arquitecturas de software embebido

Arquitecturas de software embebido

- Evento: situación que debe ser atendida (ej: cambio en una entrada, recepción de un dato, etc.)
- Handler: código que se encarga de gestionar el evento
- En términos generales, la arquitectura define cómo se detectan los eventos y de qué forma se llama al handler.

Arquitecturas de software embebido

- Round Robin (RR)
- Máquinas de estado
- Round Robin con interrupciones
- Encolado de funciones
- Sistemas operativos en tiempo real

Factores de decisión

- ¿Qué tiene que hacer el sistema?
- Tiempos de respuesta.
- ¿Cuántos eventos diferentes hay que atender?
- Tiempos críticos.

Round Robin

- Se utiliza un bucle infinito
- Se chequean los eventos
- A partir del chequeo se ejecuta el handler

Round Robin

```
void main()
{
    while (true)
    {
        if (/*Device A needs attention*/)
        {
            // Take care of device A
        }

        if (/*Device B needs attention*/)
        {
            // Take care of device B
        }
        ...
        if (/*Device N needs attention*/)
        {
            // Take care of device N
        }
    }
}
```

Round Robin

- Prioridades:
  - Ninguna: Todos los eventos son iguales
  - Cada handler debe esperar su turno.
- Tiempo de respuesta (peor caso):
  - Una vuelta al bucle (gestionando todos los eventos restantes primero)
  - Malo para todos los eventos si uno solo requiere procesamiento pesado

Round Robin

- Ventajas:
  - Sencillez: única tarea, sin interrupciones.
- Desventajas:
  - Solución frágil: agregar un nuevo handler modifica tiempos de respuesta restantes y puede provocar pérdida de deadlines.

Interrupciones

- ¿Qué es una interrupción?
- ¿Qué pasa si dos interrupciones ocurren a la misma vez ?
- ¿Qué ocurre si se deshabilitan las interrupciones y se genera una IRQ?
- ¿Puede una interrupción caer en medio de una línea de C?

Round Robin con interrupciones

- Se utiliza un bucle infinito
- Se chequean los eventos a partir de flags (banderas)
- A partir del chequeo se ejecuta el control

Round Robin con interrupciones

```
bool fDeviceA = false;
bool fDeviceB = false;

void interrupt handlerA()
{
    // Take care of device A
    fDeviceA = true;
}

void interrupt handlerB()
{
    // Take care of device B
    fDeviceB = true;
}
```

```
void main()
{
    while (true)
    {
        if (fDeviceA)
        {
            // Handler device A
            fDeviceA = false;
        }

        if (fDeviceB)
        {
            // Handler device B
            fDeviceB = false;
        }
    }
}
```

Round Robin con interrupciones

- Al agregar interrupciones:
  - ISR realiza la respuesta inicial
  - Resto realizado por funciones llamadas en el bucle
  - ISR indica con banderas la necesidad de procesado

Round Robin con interrupciones

- Presenta mayor flexibilidad:
  - Respuesta de tiempo crítico disparadas por interrupciones y realizadas en ISR.
  - Código con tiempo de procesamiento “largo” ubicado en handlers.

Round Robin con interrupciones

- Ventajas:
  - Trabajo realizado en ISR tiene mayor prioridad.
  - Tiempo de respuesta de ISR estable si el código cambia.
- Desventajas:
  - ISRs y handlers comparten datos
  - Tiempo de respuesta de handler no estable cuando el código cambia.

Encolado de funciones

- Similar a Round Robin con interrupciones
  - Trabajo dividido en ISR y handlers
  - ISR encola su propio handler
- Características de la cola de funciones:
  - FIFO: First Input First Output (cola clásica)
  - Alternativamente, un ISR de mayor prioridad podría encolar al principio para tener prioridad.

Encolado de funciones

```
void interrupt ISR_A()
{
    // Handler inicial para A
    queue_add(handle_event_A);
}

void interrupt ISR_B()
{
    // Handler inicial para B
    queue_add(handle_event_B);
}
...
void interrupt ISR_N()
{
    // Handler inicial para N
    queue_add(handle_event_N);
}
```

```
void (*task)();

void (main)
{
    while (true)
    {
        if ((task = queue_get()) != NULL)
        {
            task();
        }
    }
}
```

Máquinas de estado

- Las máquinas de estado pueden usarse en conjunto con cualquier arquitectura de software, y permiten dividir una función para procesar en partes más chicas, o combinar con la arquitectura para esperar sin bloquear.
- Sirve para implementar un sistema de multitasking cooperativo.

Máquinas de estado

```
void handler()
{
    static int state = STEP_1;

    switch (state)
    {
        case STEP_1:
            // Do step 1
            step = STEP_2;
            break;
        case STEP_2:
            // Do step 2
            step = STEP_N;
            break;
        case STEP_N:
            // Do step N
            step = STEP_1
            break;
    }
}
```

Sistema operativo en tiempo real

- Multitasking preemptivo
- Tareas priorizadas y ejecutadas por scheduler.
- Prioridades
- Tareas se bloquean si esperan eventos o recursos
- Tiempo de respuesta: suma de los tiempos de ISR + tareas de mayor prioridad

Sistema operativo en tiempo real

- Ventajas:
  - Estabilidad cuando el código cambia: agregar una tarea no afecta el tiempo de respuesta de las tareas de mayor prioridad.
  - Muchas opciones disponibles: pagas y libres

- Desventajas:
  - Complejidad: mucho dentro de RTOS, algo para usarlo.
  - Requerimientos mínimos y overhead del RTOS.

Conclusión

- ¿Qué arquitectura elegir?
  - Siempre la más sencilla que cumpla con los requerimientos de tiempos y escalabilidad de lo que se quiere construir. Se deben considerar posibles requerimientos futuros.

- Bibliografía
  - An Embedded Software Primer - David E. Simon (Capítulo 5)