REDES DE SENSORES INALÁMBRICOS

CONTIKI OS – PARTE I

Javier Schandy
Inst. de Ingeniería Eléctrica, Facultad de Ingeniería
Universidad de la Republica (Uruguay)

AGENDA

- Introducción a RTOS
  - ¿Qué son?
  - ¿Por qué usarlos?
- Contiki OS
  - Introducción
  - Principales características
  - Estructura de directorios
  - Procesos / Protothreads
  - Eventos
  - Tutorial: Primera aplicación
  - Timers
  - Multithreading
  - File System

INTRODUCCIÓN A RTOS

¿QUÉ ES UN RTOS?

- Sistemas operativos diseñados para servir aplicaciones de tiempo real.
- Manejan los recursos hardware y ejecución de aplicaciones.
- Tiempos de ejecución conocidos
- Scheduler permite al programador elegir orden de ejecución de las aplicaciones

Contiki

The Open Source OS for the Internet of Things

TinyOS

freeRTOS

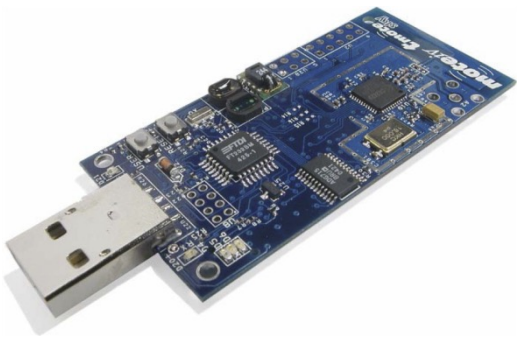
¿POR QUÉ USAR UN RTOS?

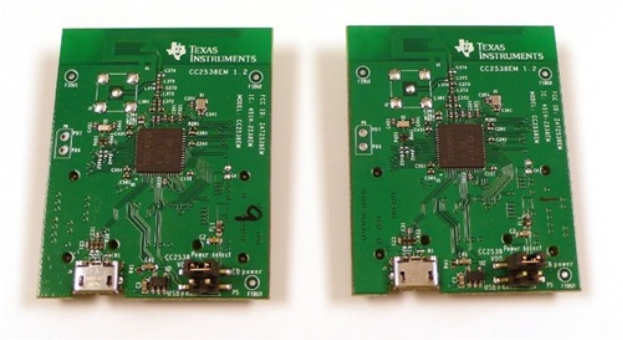
- Reduce en gran medida la dificultad de programación para aplicaciones complejas.
- Modularidad – Permite reuso de código en otros proyectos.
- Portabilidad – HAL.
- Stacks de comunicación y drivers integrados.
- Multitasking permite un uso eficiente del CPU.
  - Disminuye tiempo de respuesta.
  - Facilita el desarrollo.

CONTIKI OS

CONTIKI OS - INTRODUCCIÓN

- Sistema Operativo Open Source.
- Portado a un gran numero de plataformas y MCUs.
- Reducido footprint en memoria (10kB RAM 40kB ROM incluyendo todo el Stack de red).
- En permanente desarrollo por una comunidad muy activa (ahora Contiki-NG)
- Entorno de programación y simulación





CONTIKI OS – PRINCIPALES CARACTERÍSTICAS

- Uso eficiente de memoria.
- Provee un stack IP completo y soporta gran variedad de protocolos en las distintas capas (UDP, TCP, 6LoWPAN, RPL, CoAP).
- Manejo eficiente de modos de LPM de los MCU.
- Sleepy Routers: Manejo del ciclo de trabajo de la radio (ContikiMAC, CXMAC).
- CFS – Coffee File System para manejo de Flash.
- Escrito en standard C
- Simulador Cooja.

CONTIKI OS – ESTRUCTURA DE DIRECTORIOS

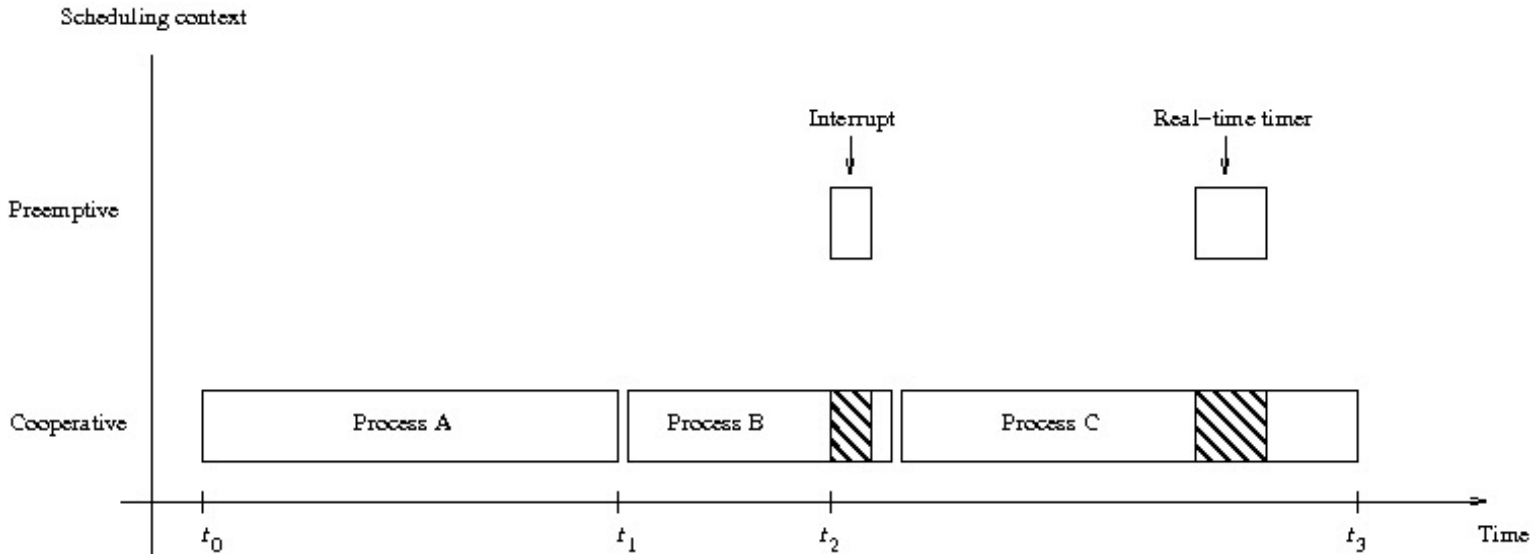
- contiki/core
  - System source code, includes
  - net: MAC, RDC, Rime, IP
  - sys: Processes
  - cfs: File System
- contiki/cpu
  - CPU-specific code: one subdirectory per CPU
- contiki/platform
  - Platform-specific code:
    - platform/sky/contiki-conf.h
    - platform/sky/contiki-sky-main.c

CONTIKI OS – ESTRUCTURA DE DIRECTORIOS

- contiki/dev
  - Radios, sensorses, ethernet
- contiki/examples
  - Gran variedad de ejemplos para las distintas plataformas.
- contiki/apps
  - System apps (telnet, shell, deluge, servreghack, tunslip)
- contiki/tools
  - e.g. cooja, start with "ant run"
  - tools/sky contains serialdump (start with "./serialdump-linux -b115200 /dev/ttyUSB0") and other useful stuff
- contiki/regression-tests

CONTIKI OS - PROCESOS

- El código en Contiki puede correr en dos contextos:
  - Cooperativo
  - Preemptivo
- Todos los programas de Contiki son procesos y corren en modo cooperativo.



CONTIKI OS - PROCESOS

- Están compuestos por:
  - Bloques de control (pocos bytes)
    - Almacenados en RAM
    - Información de Runtime
      - Nombre
      - Estado
      - Puntero al thread
    - Se declara: **`PROCESS(rsi_process, "Hello world process")`**

```
struct process {
    struct process *next;
    const char *name;
    int (* thread)(struct pt *, process_event_t, process_data_t);
    struct pt pt;
    unsigned char state, needspoll;
};
```

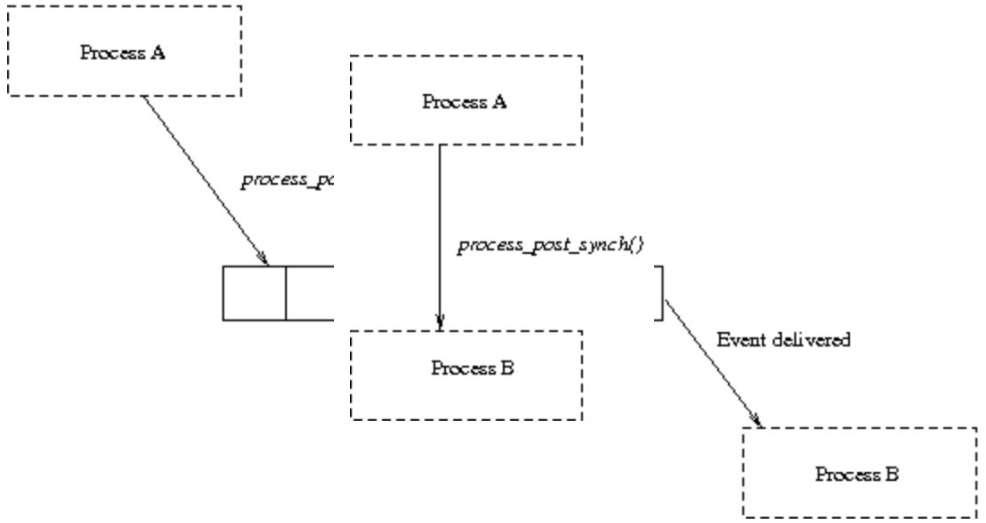
CONTIKI OS - PROCESOS

- Están compuestos por:
  - Process Threads (protothreads)
    - Almacenados en ROM
    - Utilizados por el usuario
    - Contiene el código de ejecución del proceso
    - Invocado por el scheduler de procesos

```
PROCESS_THREAD(rsi_process, ev, data){

    PROCESS_BEGIN();

    printf("Clase Contiki RSI\n");

    PROCESS_END();
}
```

CONTIKI OS - EVENTOS

- Los eventos son los disparadores de los procesos
- Hay dos tipos:
  - Asíncronos
    - Se postean mediante la función **`process_post()`**
    - Se encolan en la cola de eventos del kernel
    - Se pueden postear a un proceso o a todos
  - Síncronos
    - Se postean mediante la función **`proccess_post_synch()`**
    - Se entregan inmediatamente (funcionalidad equivalente a llamar una función)
    - El destinatario puede ser un único proceso
    - El proceso que recibe no sabe que el evento fue síncrono
- Tienen identificadores de evento que se pasan a los procesos (msg, continue, timer, exit, init, etc.)



CONTIKI OS – LLAMADAS A PROCESOS

- El scheduler llama a los procesos en los que se postearon eventos, o a los que se le hizo process_poll(). Se les pasa:
  - Identificador de evento
  - Puntero a datos
- Comenzar un proceso:
  - **`process_start()`**
    - Comprueba que no se haya creado previamente
    - Crea el bloque de control
    - Lo añade a la lista de procesos activos
    - Le cambia el estado a PROCESS_STATE_RUNNING
    - Se le envia un evento PROCESS_EVENT_INIT (inicialización)
  - **`AUTOSTART_PROCESS()`**
    - Se llama cuando el sistema bootea.
    - Es la forma más común de iniciar procesos

CONTIKI OS – LLAMADAS A PROCESOS

- Terminar un proceso:
  - Se le avisa a los demás procesos para que liberen los recursos asociados
    - Evento del tipo PROCESS_EVENT_EXITED
  - Se quita de la lista de procesos activos
  - Formas de terminar un proceso:
    - Cuando llega a **`PROCESS_END()`**
    - Cuando lo terminan invocando la función **`process_exit()`**

CONTIKI OS – ESTRUCTURA DE UNA APLICACIÓN

```
<Header Files>

PROCESS(name,strname);
AUTOSTART_PROCESSES(struct process &);

PROCESS_THREAD(name, process_event_t, process_data_t){

    ----Initialization of required variables----

    PROCESS_BEGIN();

    ---Set of C statements---

    PROCESS_END();

}
```

CONTIKI OS – TUTORIAL: PRIMERA APLICACIÓN

1. Ir al directorio `contiki/examples` y crear la carpeta `rsi_helloworld`
2. Crear un archivo con el nombre `helloworld.c`
3. Crear un archivo `Makefile` con el siguiente contenido:

```
CONTIKI_PROJECT = helloworld
all: $(CONTIKI_PROJECT)

CONTIKI = ../..
CFLAGS += -DPROJECT_CONF_H=\"project-conf.h\"

include $(CONTIKI)/Makefile.include
```

CONTIKI OS – TUTORIAL: PRIMERA APLICACIÓN

4. Crear un archivo project-conf.h con el siguiente contenido:

```
#ifndef PROJECT_CONF_H_
#define PROJECT_CONF_H_

#define NETSTACK_CONF_RDC     nullrdc_driver

#endif /* PROJECT_CONF_H_ */
```

CONTIKI OS – TUTORIAL: PRIMERA APLICACIÓN

5. Escribir la aplicación en el archivo `helloworld.c`

```
#include "contiki.h"
#include "stdio.h"

PROCESS(helloworld_process, "Hello World process");
AUTOSTART_PROCESSES(&helloworld_process);

PROCESS_THREAD(helloworld_process, ev, data){
   PROCESS_BEGIN();
   printf("============ \n");
   printf("Hello World! \n");
   printf("============ \n");
   PROCESS_END();
}
```

CONTIKI OS – TUTORIAL: PRIMERA APLICACIÓN

6. Abrir una terminal, ir al directorio del proyecto y ejecutar los comandos:
   `make TARGET=native`
7. Correr la aplicación ejecutando:
   `./helloworld.native`

CONTIKI OS – TIMERS

- timer
  - Se utilizan para chequear si transcurrió un cierto período de tiempo.
  - Cuando expira no toma ninguna iniciativa, hay que consultarlo.
  - Utiliza los ticks del reloj del sistema como unidad de medida, por lo que sirven para medir períodos cortos de tiempo (tiene gran resolución).
- stimer
  - Similar a los timer, pero utiliza segundos como medida, por lo que pueden medir grandes cantidades de tiempo (menor resolución).
  - Uso: `void stimer_set(struct stimer *t, unsigned long interval)`
  - Chequeo: `int stimer_expired(struct stimer *t)`
- etimer
  - Permite generar eventos temporizados.
  - Uso:
    ```
    etimer_set(&et, CLOCK_SECOND);
    etimer_reset(&et);
    ```
  - Cuando vence, postea el evento **`PROCESS_EVENT_TIMER`** al proceso que seteó el etimer.

CONTIKI OS – TIMERS

- ctimer
  - Llaman una cierta función cuando expiran.
  - Utilizan **`clock_time()`** para obtener la hora del sistema.
  - Uso: **`ctimer_set(&timer, CLOCK_SECOND, callback, (void*) arg);`**
    `ctimer_reset(&timer);`
- rtimer
  - Permite agendar y ejecutar tareas de tiempo real.
  - Utiliza su propio módulo de reloj para permitir mayor resolución.
  - Las tareas se ejecutan en modo preemptivo.
  - Generalmente se usa para llamar un **`process_poll()`**

CONTIKI OS – TIMERS: EJEMPLO

```
PROCESS_THREAD(example_process, ev, data){
   static struct etimer et;
   static uint16_t tiempo;

   PROCESS_BEGIN();

   etimer_set(&et, 2*CLOCK_SECOND);

   while(1) {
      PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
      tiempo+=2;
      printf("Expiro el timer! Tiempo total: %d segundos \n", tiempo);
      etimer_reset(&et);
   }

   PROCESS_END();
}
```

CONTIKI OS – TIMERS: EJERCICIO

- Crear un archivo en el proyecto que implemente el código de la diapositiva anterior y probarlo.
  - ¿Qué pasa si la variable `tiempo` no es estática?
  - ¿Qué pasa si inicializo el tiempo en 0 antes de la declaración de `PROCESS_BEGIN()`?
- Implementar lo mismo que el ejemplo anterior pero utilizando un `ctimer`, donde en la función de callback se incremente el tiempo y se despliegue el mensaje en pantalla.

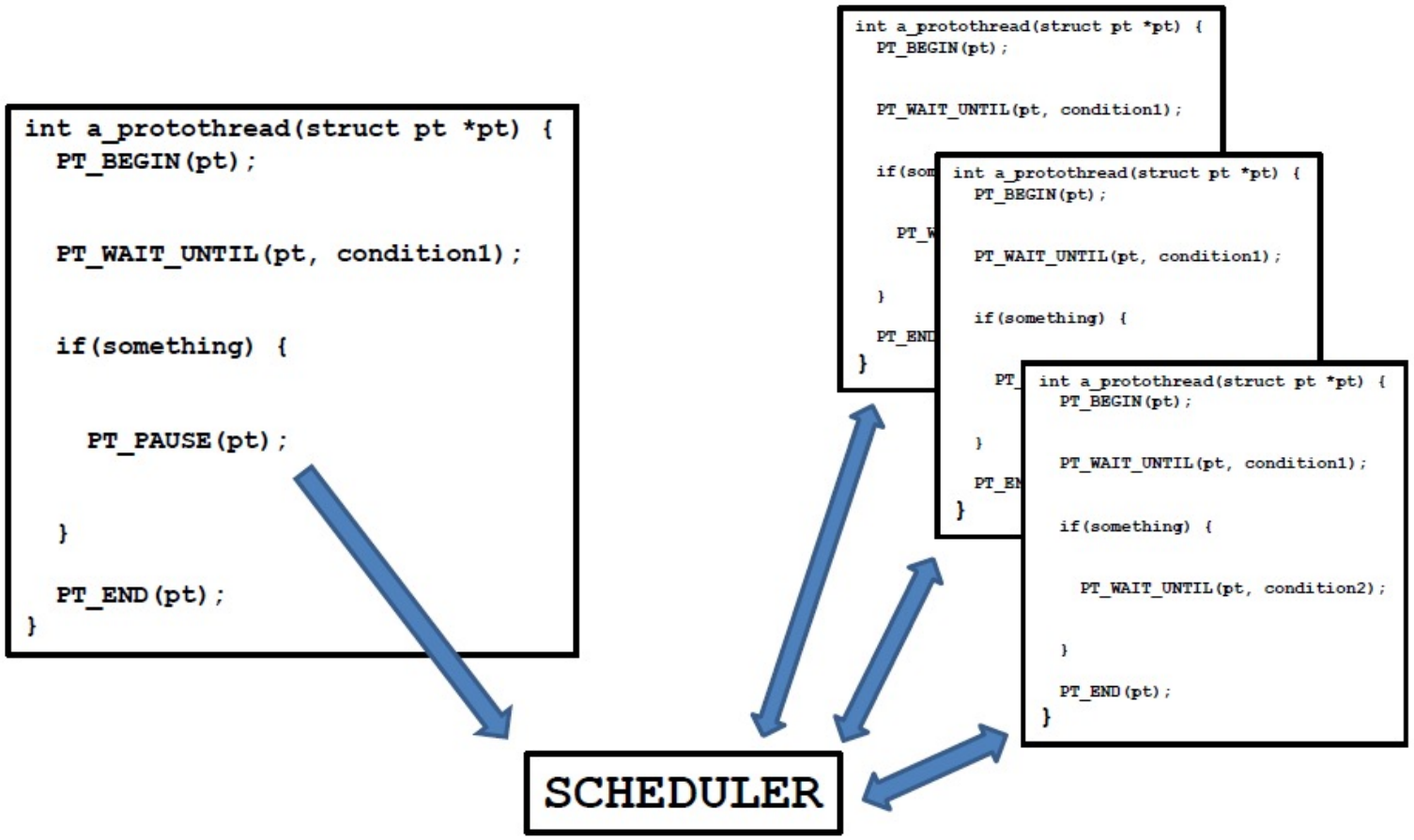
CONTIKI OS – MULTITHREADING

- Protothreads
- Librerías multithreading
  - Permiten threads preemptivos
  - Cada uno tiene su stack

```
PROCESS_THREAD(example_process, ev, data){
   PROCESS_BEGIN();
   SENSORS_ACTIVATE(button_sensor);
   while(1) {
     PROCESS_WAIT_EVENT_UNTIL(ev ==
sensors_event && data == &button_sensor);
     frecuencia *= 2;
   }

   PROCESS_END();
 }
```

```
PROCESS_THREAD(blink, ev, data) {
   PROCESS_BEGIN();
   static struct etimer et;
   while(1) {
     etimer_set(&et, frecuencia);
     PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
     leds_toggle(LEDS_ALL);
   }

    PROCESS_END();
 }
```

CONTIKI OS – MULTITHREADING



CONTIKI OS – POSTEO DE EVENTOS

- Un proceso puede postear un evento en otro proceso pasando un puntero a datos de la siguiente forma:
  1. Declarar el evento: `process_event_t event_boton;`
  2. Asignar memoria para el evento mediante la función:
     `event_boton = process_alloc_event();`
  3. Postear el proceso pasando un puntero a datos:
     `process_post(&proceso_destino,event_boton, &var);`
- El proceso `proceso_destino` recibirá un evento con `ev == event_boton`.
- Para interpretar el dato se debe castear el puntero recibido al tipo de datos correcto.
  - Ej. Si `var` es un entero de 16 bits lo puedo leer de la siguiente forma:
    `uint16_t num = *(uint16_t*)data;`

CONTIKI OS – CFS

- Sistema de archivos de Contiki.
- Provee una API para el manejo del mismo:
  - `cfs_open(filename, CFS_WRITE);`
  - `cfs_close(fd_write)`
  - `cfs_read(fd_read, buf, sizeof(message))`
  - `cfs_write(fd_write, message, sizeof(message))`
  - `cfs_seek(fd_read, sizeof(message), CFS_SEEK_SET)`
  - `cfs_remove(filename)`

CONTIKI OS – CFS

```
fd_write = cfs_open(filename, CFS_WRITE);
if(fd_write != -1) {
  n = cfs_write(fd_write, message,
      sizeof(message));

  cfs_close(fd_write);
} else {

  printf("ERROR: couldnt write to memory.\n");
}
```

Escribir

```
strcpy(buf,"empty string");
fd_read = cfs_open(filename, CFS_READ);
if(fd_read!=-1) {
   cfs_read(fd_read, buf, sizeof(message));
   printf("step 3: %s\n", buf);

   cfs_close(fd_read);
} else {

   printf("ERROR: couldnt read from memory.\n");
}
```

Leer

CONTIKI OS – COFFEE

- Es una adaptación del CFS para dispositivos de bajo consumo con memorias FLASH o EEPROM.
- Footprint muy chico.
- Agrega a la API del CFS tres funciones:
  - `int cfs_coffee_format(void)`
  - `int cfs_coffee_reserve(const char *name, cfs_offset_t size)`
  - `int cfs_coffee_configure_log(const char *file, unsigned log_size, unsigned log_entry_size)`

PLANIFICACIÓN DE CLASES

1. Introducción RSI
2. Modelado e introducción a IPv6
3. Plataforma de hardware
4. **Plataforma de software: Contiki OS I**
5. Plataforma de software: Contiki OS II
6. Capa de aplicación: CoAP / MQTT
7. Capa de red: RPL
8. Subcapa MAC
9. IEEE 802.15.4 / 6lowpan
10. Capa Física & antenas
11. IoT y las RSI

- ¿PREGUNTAS?