Sistemas Embarcados

Programação Concorrente e Periféricos Básicos no Contiki

Prof. Guilherme de S. Peron

peron@utfpr.edu.br

Curso de Especialização em Internet das Coisas (CEIoT)

11 de Agosto de 2018

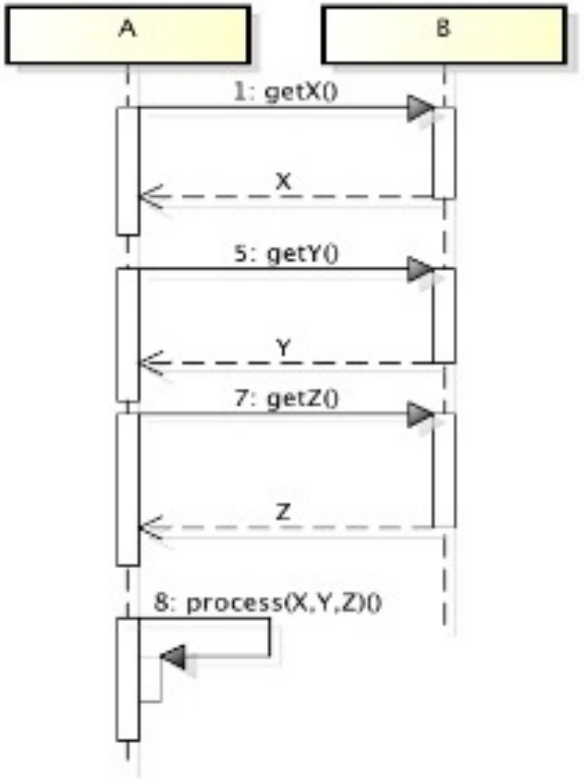
Introdução

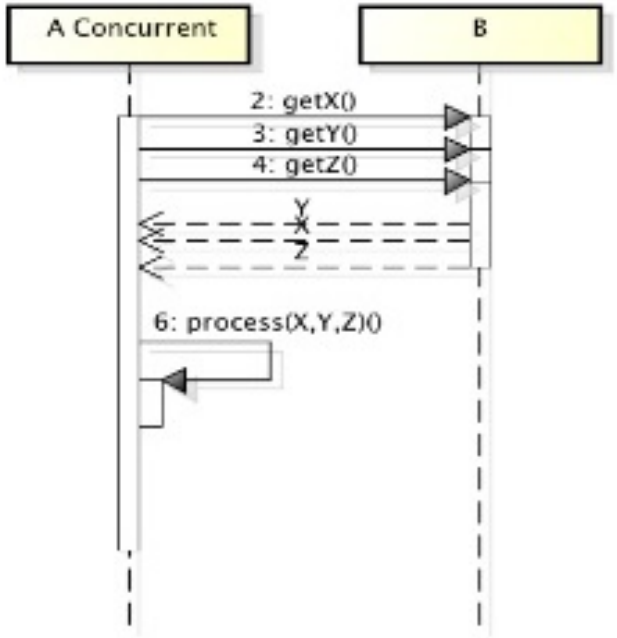
Paradigma de Programação do Contiki

Suponha que você precise programar um nó sensor que:

- A cada 125 ms leia um valor de temperatura e envie para o nó central (operação que leva de 15-40 ms);
- A cada 300 ms leia um valor de luminosidade e envie para o nó central (operação que leva $\sim$70ms);
- Receba dados de setpoint de temperatura, confirmando com um pacote de ACK a cada recebimento (o ACK deve ser enviado dentro de 100 ms);
- Implemente um algoritmo PID que utiliza os dados de temperatura e luminosidade dos sensores, cujo tempo de processamento é de 40 ms, com período de amostragem de 100 ms.

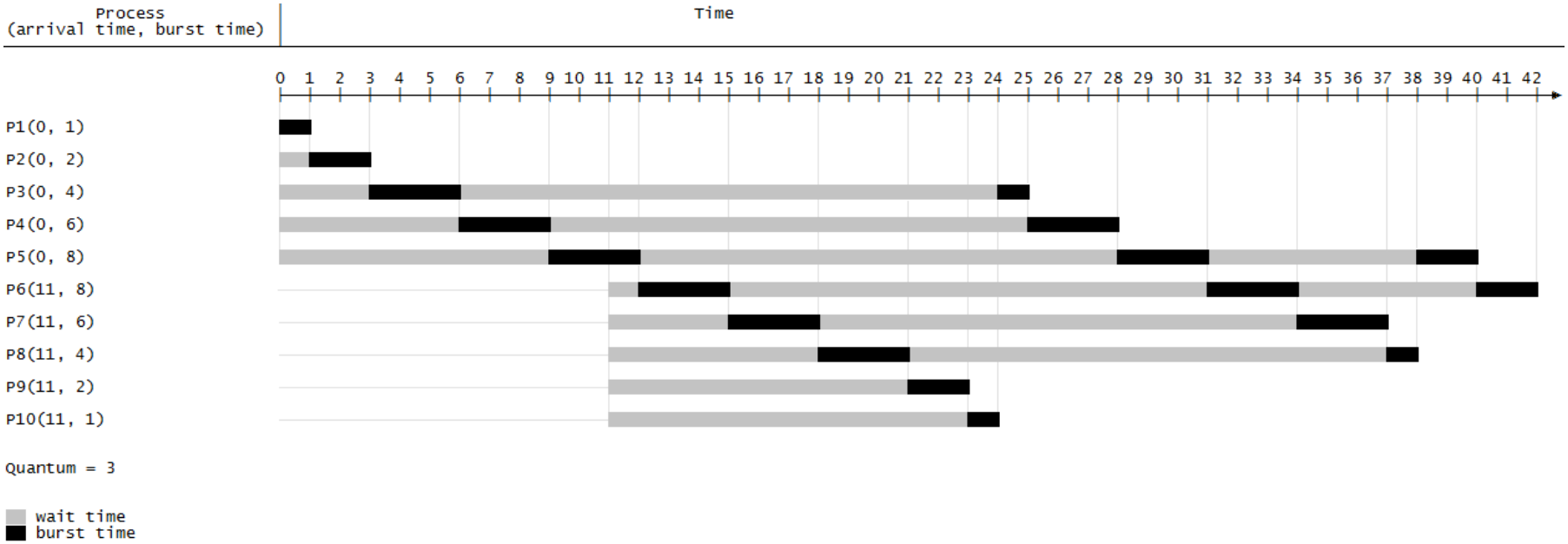
Paradigma de Programação do Contiki



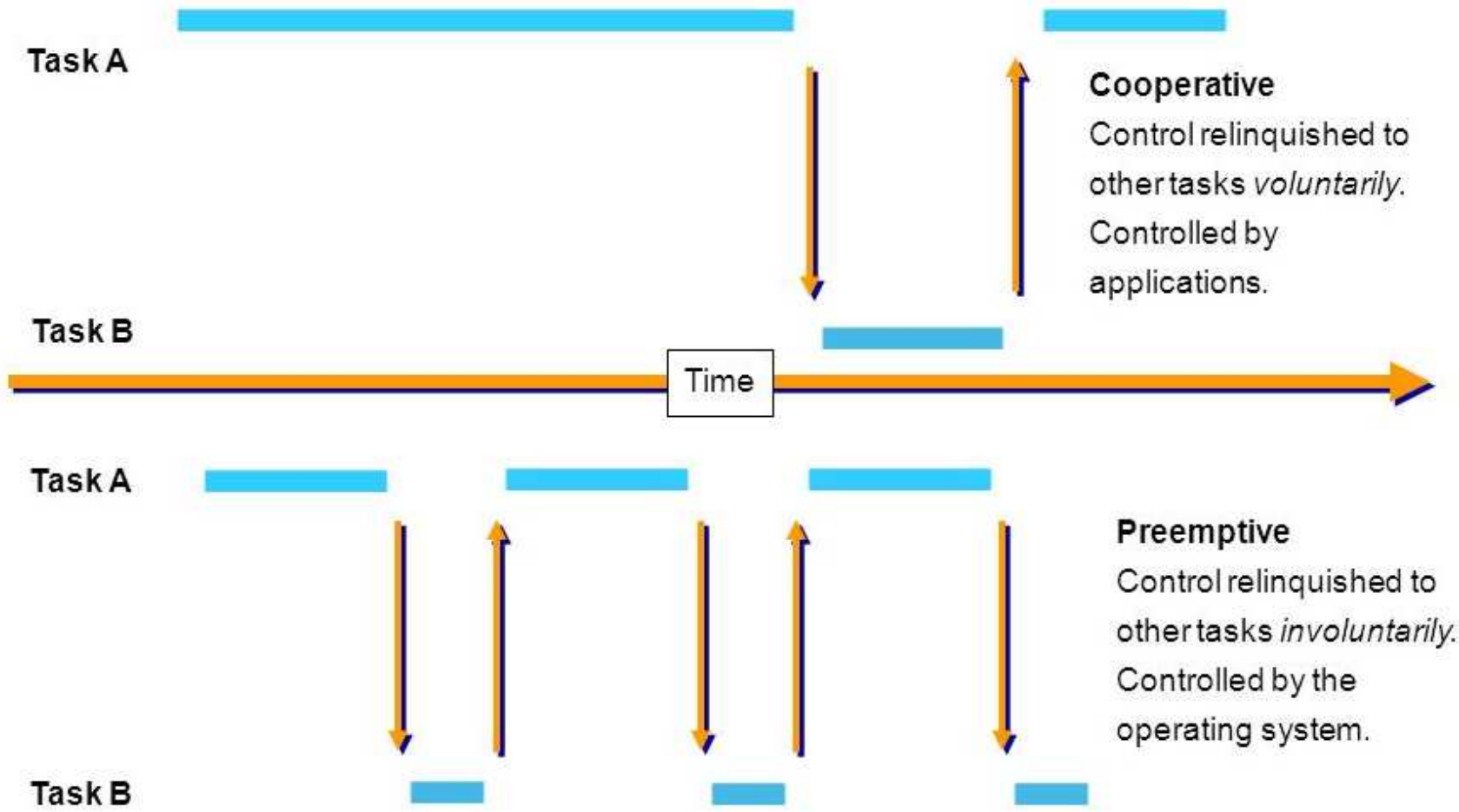


Escalonamento Round-Robin

Solução: Dividir o tempo do processador para as diferentes tarefas.



RTOS Cooperativo *vs.* RTOS Preemptivo



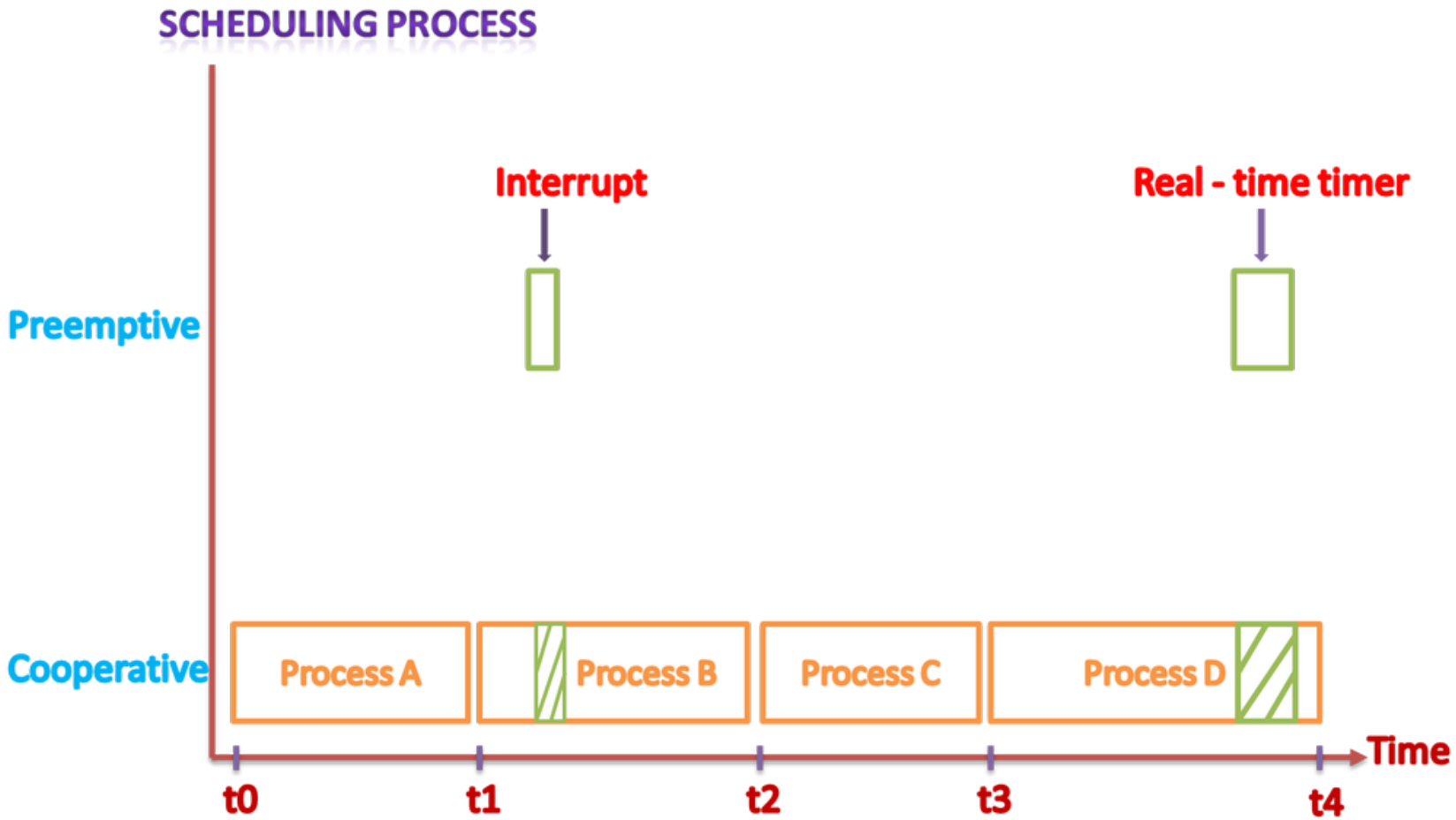
RTOS Cooperativo *vs.* RTOS Preemptivo

Qual tipo de RTOS é melhor?

Processos no Contiki

- No Contiki, todos os programas são chamados processos.
- Os processos no Contiki rodam no contexto cooperativo, ao passo que as interrupções e temporizadores de tempo-real rodam no contexto preemptivo.
  - Códigos cooperativos são organizados sequencialmente, sendo executado até o fim antes que o próximo evento agendado inicie.
  - Os códigos preemptivos podem parar os códigos cooperativos a qualquer momento. Ou seja, são executados com maior prioridade.

Processos no Contiki



Estrutura de um Processo

- Um processo no Contiki é composto de duas partes:
  - **Bloco de Controle:** gravado na RAM com as informações sobre nome, estado atual e um ponteiro para a thread do processo.
  - **Thread:** que é o código do processo em si, gravado na ROM.
- O bloco de controle é declarado através da macro `PROCESS()`. Exemplo para um processo Hello World:

```
PROCESS(hello_world_process, "Hello world process");
```

  - `hello_world_process` é o **nome** do processo, que é a variável usada para acessá-lo.
  - `"Hello world process"` é sua **descrição**, usada para debug.

Thread de Processo

- A thread contém o código do processo, o qual será executado sempre que chamada pelo escalonador.

- Exemplo para um processo Hello World:

```
PROCESS_THREAD(hello_world_process, ev, data)
{
    PROCESS_BEGIN();
    printf("Hello, world\n");
    PROCESS_END();
}
```

Protothreads

- Protothreads são uma forma de liberar o processador enquanto um processo espera por algo acontecer.

- No Contiki temos as seguintes protothreads:

`PROCESS_BEGIN();`	Declara o início de uma thread
`PROCESS_END();`	Fim de uma thread
`PROCESS_EXIT();`	Força o fim de um processo
`PROCESS_WAIT_EVENT();`	Aguarda por um evento
`PROCESS_WAIT_EVENT_UNTIL();`	Aguarda por um evento, com condição
`PROCESS_YIELD();`	Equivalente a `PROCESS_WAIT_EVENT()`
`PROCESS_WAIT_UNTIL();`	Aguarda por uma condição
`PROCESS_PAUSE();`	Libera o processador temporariamente

Eventos

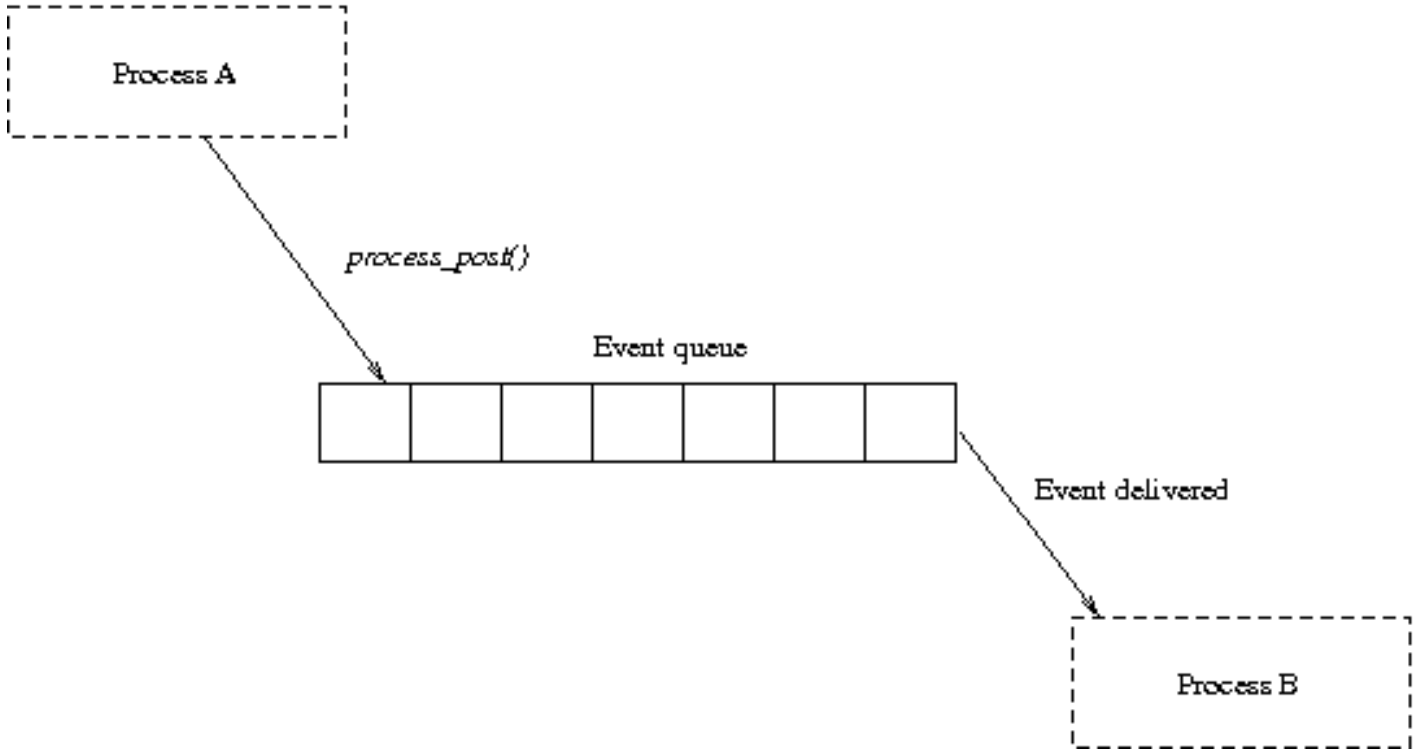
- No Contiki, um processo roda quando recebe um "evento".
- Existem dois tipos de eventos possíveis: assíncronos e síncronos.
- Cada processo do Contiki é capaz de enviar eventos para outros processos utilizando as seguintes funções:

```
int process_post(struct process *p, process_event_t ev,
                 void *data);

int process_post_synch(struct process *p, process_event_t ev,
                       void *data);
```

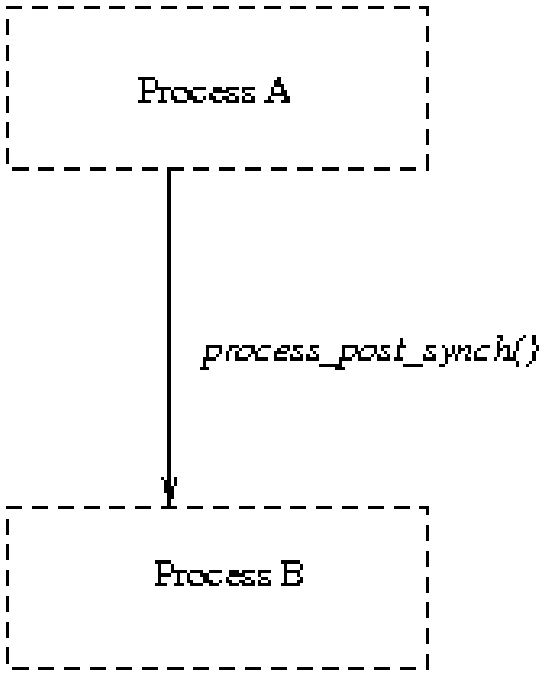
Eventos Assíncronos

- Quando um evento assíncrono é enviado (`process_post()`), este evento é enfileirado pelo processador para ser entregue ao destino quando apropriado.



Eventos Síncronos

- Um evento síncrono (`process_post_synch()`) é imediatamente enviado ao processo de destino.



Exemplo 1

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

PROCESS(example_process, "Example process");
AUTOSTART_PROCESSES(&example_process)

PROCESS_THREAD(example_process, ev, data)
{
    PROCESS_BEGIN();

    while(1) {
        PROCESS_WAIT_EVENT();
        printf("Got event number %d\n", ev);
    }

    PROCESS_END();
}
```

Hello World e Timer

```
#include "contiki.h"
#include <stdio.h>
static struct etimer et_hello;

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

PROCESS_THREAD(hello_world_process, ev, data)
{
    PROCESS_BEGIN();
    etimer_set(&et_hello, 4*CLOCK_SECOND);

    while(1) {
        PROCESS_WAIT_EVENT();
        if(ev == PROCESS_EVENT_TIMER) {
            printf("Hello world!\n");
            etimer_reset(&et_hello);
        }
    }
    PROCESS_END();
}
```

Tarefas

O roteiro de atividades está disponível no Moodle. Iremos precisar de algumas funções para manipulação dos leds do kit:

- `unsigned char leds_get(void);`
- `void leds_set(unsigned char leds);`
- `void leds_on(unsigned char leds);`
- `void leds_off(unsigned char leds);`
- `void leds_toggle(unsigned char leds);`

em que a variável `leds` pode assumir um dos valores a seguir (ou uma concatenação desses valores utilizando *bitwise or*):

- `LEDS_GREEN`
- `LEDS_RED`
- `LEDS_ALL`

OBS.: Será necessário adicionar `#include "dev/leds.h"` ao seu código.