UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE ESPECIALIZAÇÃO EM INTERNET DAS COISAS
ATIVIDADE PRÁTICA COM O CC2650 LAUNCHPAD

Professor: Guilherme de S. Peron
Tema: Programação Concorrente e Periféricos Básicos no Contiki
Data: 11/08/2018

Objetivos

A seguinte atividade visa introduzir a utilização dos seguintes conceitos para o Contiki:

- Estrutura básica de um software e Makefile;
- Utilização de LEDs e botões;
- Utilização de timers;
- Programação concorrente básica.

Atividade 1 – Hello World

1. Utilizando o Code Composer, abra o arquivo `hello-world.c` disponível na pasta `examples/hello-world`.
   OBS.: Para saber mais sobre a estrutura de diretórios do Contiki leia [1, Seção 3.3].

2. Identifique as estruturas de um projeto para o Contiki conforme [1, Seção 3.5.1].
   (a) Enumere o nome dos processos contidos no arquivo;
   (b) Identifique o(s) processo(s) que serão executados assim que o Contiki for inicializado.

3. Crie uma cópia desse arquivo na mesma pasta e o renomeie para `embarcados1.c`. Em seguida, modifique o arquivo `embarcados1.c` de acordo com o exemplo Hello World e Timer dos slides e compile o novo código. Observe que na configuração atual o arquivo `embarcados1.c` não será compilado pois não está adicionado no `Makefile`.
   DICA: Para adicionar, observe [1, Seção 3.5.2].

4. Grave a aplicação no kit de desenvolvimento e observe a saída da porta serial utilizando o `moserial` (115200-8-N-1).

Atividade 2 – Temporização de Processos

O Contiki é um sistema operacional em tempo real que trabalha de maneira cooperativa, cuja entidade básica de execução é chamada de processo. Cada processo é executado de maneira sequencial até liberar o processador de maneira voluntária. Além disso, cada processo é executado sempre que algum evento aconteça, por exemplo, na ocasião do recebimento de um pacote de dados, ou com a finalização de um temporizador.

As seguintes macros são utilizadas para indicar uma espera em um processo. Isso faz com que o processador seja liberado para execução de outro processo:

- `PROCESS_WAIT_EVENT();` ⟶ Aguarda por um evento
- `PROCESS_WAIT_EVENT_UNTIL();` ⟶ Aguarda por um evento com uma condição
- `PROCESS_YIELD();` ⟶ Equivalente a `PROCESS_WAIT_EVENT()`
- `PROCESS_PAUSE();` ⟶ Libera o processador temporariamente

1. No arquivo `embarcados1.c`, crie um novo processo chamado `blink_process` tomando como base o processo `hello_world_process`.

2. Crie um novo `etimer` chamado `et_blink` e modifique o `blink_process` para que ele alterne apenas o led verde a cada 2 segundos.

   As funções disponíveis para manipulação dos leds são:

   - `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 a linha `#include "dev/leds.h"` ao seu código.

3. Adicione a macro `PROCESS()` para o processo recém criado e, para que o processo passe a executar a partir da inicialização do Contiki, adicione o processo recém criado à macro `AUTOSTART_PROCESSES`.

4. Verifique o funcionamento no kit.

5. Crie agora um terceiro processo chamado `proc3_process` para imprimir uma mensagem na porta serial a cada 10 segundos. Para tal, crie um novo `etimer` chamado `et_proc3`.

6. Verifique o funcionamento no kit.

Atividade 3 – Comunicação entre Processos

Além de aguardar eventos de temporização, cada processo do Contiki também é capaz de enviar eventos para os 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);`

Os eventos assíncronos (gerados pela função `process_post`) são armazenados em uma fila de eventos. Dessa forma, quando o processador estiver livre, o primeiro evento da fila será entregue a seu respectivo processo, que passará a executá-lo. Já um evento síncrono (gerado pela função `process_post_synch`) pode ser considerado uma chamada de função, pois é entregue imediatamente ao processo de destino, que passa a executá-lo a partir da chamada de função. O processo que enviou o evento, por sua vez, irá continuar sua execução apenas quando o processo que recebeu a notificação liberar o processador.

Como parâmetros dessas duas funções temos:

- `struct process *p`: que representa o nome do processo, conforme declarado na macro `PROCESS()`. Esse parâmetro pode receber um `PROCESS_BROADCAST` para que todos os processos ativos recebam o evento;

- `process_event_t ev`: que representa o número do evento. O Contiki suporta 256 tipos de eventos diferentes. Todos os eventos com números menores que 128 podem ser definidos livremente. Porém, existem alguns eventos reservados:

```
#define PROCESS_EVENT_NONE          128
#define PROCESS_EVENT_INIT          129
#define PROCESS_EVENT_POLL          130
#define PROCESS_EVENT_EXIT          131
#define PROCESS_EVENT_CONTINUE      133
#define PROCESS_EVENT_MSG           134
#define PROCESS_EVENT_EXITED        135
#define PROCESS_EVENT_TIMER         136
```

  Observe que o evento 136 já foi utilizado no exemplo anterior para o timer. Para uma explicação detalhada sobre o gerenciamento de processos do Contiki, veja [2].

- `void *data`: representa um ponteiro para uma estrutura de dados relacionada ao evento (pode ser nulo se desnecessário).

1. Adicione as duas macros a seguir ao arquivo `embarcados1.c`, as quais serão utilizadas para envio de eventos:

```
#define LED_PING_EVENT          (44)
#define LED_PONG_EVENT          (45)
```

2. Adicione também um quarto processo ao arquivo `embarcados1.c` chamado `pong_process`.

3. Altere os processos `hello_world_process`, `blink_process` e `proc3_process` para que, a cada vez que um `PROCESS_EVENT_TIMER` ocorra, estes processos enviem um evento `LED_PING_EVENT` para o processo `pong_process`. Imprima pela porta serial uma mensagem indicando que o processo está enviando uma mensagem de ping.

   DICA: Utilize o parâmetro `void *data` para indicar qual processo enviou o ping. Por exemplo, para indicar que o processo `blink_process` enviou um evento `LED_PING_EVENT` para o processo `pong_process` podemos fazer:

   ```
   process_post(&pong_process, LED_PING_EVENT, (void*)(&blink_process));
   ```

   Em outras palavras, vamos modificar o código para que os processos se comuniquem entre si, em um formato de pingue-e-pongue.

4. Altere o processo `pong_process` para que ele responda com um evento `LED_PONG_EVENT` para qualquer processo que lhe enviou um `LED_PING_EVENT`.

   DICA: Para saber para qual processo enviar a resposta, recupere essa informação da estrutura do parâmetro `void *data`. Por exemplo:

   ```
   process_post((struct process*)data, LED_PONG_EVENT, NULL);
   ```

5. Adicione ao processo `pong_process` a funcionalidade de imprimir uma mensagem na serial sempre que o processo receber um evento do tipo `LED_PING_EVENT`. Exemplo de mensagem:
   Pong: Recebido ping do processo (nome do processo);

6. Altere os processos `hello_world_process`, `blink_process` e `proc3_process` para que imprimam uma mensagem quando receberem o evento de resposta `LED_PONG_EVENT`.

7. Verifique o funcionamento no kit.

Atividade 4 – Botões

Vamos agora trabalhar com os dois botões disponíveis no kit de desenvolvimento. O procedimento utilizado aqui servirá de base para trabalhar com qualquer sensor futuramente, configurando processos para responder a partir de um evento externo.

1. Faça o download do arquivo `buttons.c` disponível no Moodle e copie-o para a pasta `examples/hello-world`.

2. Por default, o CC2650 configura o botão direito como reset do processador. Vamos modificar essa configuração alterando o arquivo `platform/srf06-cc26xx/contiki-conf.h`. Procure o seguinte define e altere seu valor para `0`:

   ```
   #DEFINE BUTTON_SENSOR_CONF_ENABLE_SHUTDOWN 0
   ```

3. Identifique as estruturas utilizadas no arquivo `buttons.c` e teste seu funcionamento no kit de desenvolvimento.

4. Adicione agora um quinto processo ao arquivo `embarcados1.c` baseado no processo `read_button_process`. Modifique o código para que este processo também se comunique com o `pong_process` seguindo a mesma lógica de pingue-e-pongue.

5. Verifique o funcionamento no kit.

Referências

[1] A. L. Colina, A. Vives, A. Bagula, M. Zennaro, and E. Pietrosemoli, *IoT in five Days*. E-Book, june 2016, rev 1.1. [Online]. Available: https://github.com/marcozennaro/IPv6-WSN-book/releases/

[2] A. Dunkels, "Processes - contiki-os/contiki wiki," https://github.com/contiki-os/contiki/wiki/Processes, Agosto 2015, acessado em 2016-08-12.