TÉCNICO LISBOA

Aplicações e Tecnologias Móveis

Mestrado em Engenharia Eletrotécnica e de Computadores

Projeto - Parte 2

Autores:

Francisco Mendes (96529) francisco.m.b.mendes@tecnico.ulisboa.pt
David Fernandes (96176) david.luis.fernandes@tecnico.ulisboa.pt

Grupo 7

2022/2023 – 1º Semestre, P1

Conteúdo

1 Application description

O objetivo deste trabalho era desenvolver o código em C para simular em Cooja um sistema de um *sink* com vários *motes*, em que cada *mote* representa uma sala com um sensor de presença que liga uma luz e um sensor de temperatura que permite monitorizar a temperatura da sala. Nos tópicos seguintes vamos explicar mais detalhadamente como procedemos para tal.

2 Sensors and Actuators

O *mote* é constituído por um sensor de temperatura e um de presença, como já foi referido. Como não era possível simular com sensores físicos, utilizaram-se alguns métodos já implementados no simulador Cooja.

Nomeadamente, para simular o sensor de temperatura, criámos uma função que amostra num intervalo de tempo um valor introduzido no *serial monitor* do *mote* (ou o valor lido do sensor, no caso do sensor físico). De aproximadamente 30 em 30 segundos (é utilizado um *rand* para não haver colisão entre o envio de mensagens dos *motes*) são amostrados valores de um valor introduzido no terminal, simulando um sensor real. Após 5 amostras estes valores são enviados para o *sink* onde são guardados em memória.

Para o sensor de presença, utilizámos o botão já disponível nos *motes* do *environment*, sendo que quando clicamos no botão significa que alguém se está a movimentar na sala (o sensor foi ativado).

Também utilizámos os LED's disponíveis nos motes do *environment* como auxílio às *feactures* que queríamos implementar. Quando a temperatura da sala excede o *threshold* definido, o LED Vermelho liga. Quando a temperatura da sala é inferior ao *threshold*, o LED Verde acende e o Vermelho desliga-se.

Já o sensor de movimento ativa o LED Amarelo (ou azul), simulando assim um candeeiro.

O *sink* não apresenta quaisquer sensores, tendo apenas implementado um botão, que quando é clicado imprime no *serial monitor* as informações guardadas que os *motes* já enviaram.

3 Node & Network Architecture and Communications

Neste trabalho, decidimos simular com 3 *motes* e 1 *sink*.

A comunicação entre *motes* → *sink* é feita utilizando UDP com implementação do método de comunicação *unicast*. Para tal utilizámos como base um dos exemplos existente relativamente a este método de comunicação.

Todos os *motes* se ligam ao *sink* para enviar a informação relativamente ao seu estado. As informações enviadas são: Se a luz está ligada (1) ou não (0). E o últimos 5 valores registados

da temperatura. As informações enviadas são colocadas na estrutura apresentada de seguida, e é alocado dinamicamente um vetor de estruturas, adicionando uma nova estrutura ao vetor à medida que são feitas ligações novas de um *mote* novo (sendo verificado o IP *address* do *mote* que envia a mensagem).

```
typedef struct Vector {
  uip_ipaddr_t addr; // address of the sender mote
  int presence; // 0 = no movemment, 1 = movemment
  float temperature[5]; // last 5 values
}vector;
```

4 System Support

No código dos *motes* utilizaram-se 4 processos diferentes:

- **button_process** - Processo que verifica o movimento dentro da sala onde o *mote* se insere. No caso da nossa simulação, esta verificação é feita através do botão do *mote*, como já referimos anteriormente;
- **temperature_process** - Processo que lê os valores inseridos no *serial monitor* do *mote*. Este processo só existe na nossa simulação, caso estivesse disponível um sensor físico este processo era descartado;
- **temperature_sampling_process** - Processo que faz a amostragem dos valores do sensor (neste caso, simulado pelo *temperature_process*). Faz ainda uma pequena análise do valores (*threshold*);
- **unicast_sender_process** - Processo responsável por estabelecer a ligação com o *sink* e enviar uma mensagem, este processo é chamado dentro de outros processos.

No código do *sink* utilizaram-se 2 processos diferentes:

- **unicast_receiver_process** - Processo responsável por estabelecer a ligação com os *motes* e receber mensagens. A mensagem recebida é analisada pela função *fill_values* que guarda a informação no vetor de estruturas supramencionado;
- **button_process** - Processo que imprime as informações guardadas para visualização do usuário;

5 Energy Management

Algumas decisões de projeto foram feitas tendo em base a eficiência e limitação energética. Para começar, apenas são enviados valores de temperatura quando se acumulam 5 valores de

forma a não serem enviadas muitas mensagens. Também decidimos apenas amostrar valores de aproximadamente 30 em 30 segundos para evitar a sobrecarga de valores que podiam não ter importância. Para além disto, utilizou-se o método de comunicação *unicast*, visto que a comunicação era só unilateral e direcionada somente para um *sink*. Não havendo comunicação entre os *motes*.

6 Conclusions

Os objetivos foram cumpridos, o código desenvolvido possui todas as característica pedidas, descritas ao longo desde relatório. Este trabalho foi muito enriquecedor pois aprendemos sobre Contiki OS e a utilizar o simulador Cooja.

Referências

[1] Contiki tutorials. URL: http://anrg.usc.edu/contiki/index.php/Contiki_tutorials.

[2] Contiki wiki. URL: https://github.com/contiki-os/contiki/wiki.