Sistemas Embebidos II

Semestre de Verão de 2010/2011
Quinta atividade prática

1ª Parte

eCos - instalação do uIP

Seguindo os diapositivos sobre uIP, criar a aplicação servidora de eco aí exemplificada.

Estabeleça os parâmetros da rede (endereço IP e máscara) adequados ao seu ambiente de teste.

A forma mais fácil consiste em ligar o *kit* diretamente a um computador, criando uma rede separada.

O Windows ao detetar a ligação atribui automaticamente um endereço. Com o comando IPCONFIG pode visualizar as interfaces e os endereços atribuídos.

No Linux é necessário configurar a interface. Por exemplo, atribuir o endereço 192.168.1.1 assim:

```
$sudo ifconfig eth0 192.168.1.1
```

Com o comando ping pode verificar o correto funcionamento do novo sistema. O uIP processa adequadamente os pacotes ICMP usados pelo ping.

```
$ping 192.168.1.2
```

Com a seguinte aplicação cliente, a executar no computador, teste devidamente a aplicação servidora de eco, em execução no *kit*.

```
#include <string.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char * argv[]) {
      char buffer1[] = "abcdefghijklmnopqrstuvwxyz", buffer2[sizeof buffer1];

      struct sockaddr_in serv_addr; int serv_len, sock;
      int i, result;

      printf("Client\n");
      if (argc != 3) {
            printf("usage: %s <xx.xx.xx.xx> <port>\n", argv[0]);
            exit(1);
      }

      sock = socket(AF_INET, SOCK_STREAM, 0);
      if (sock < 0) {
            perror("socket");
            exit(1);
      }
```

```
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_addr.s_addr =  inet_addr(argv[1]);
        serv_addr.sin_port = htons(atoi(argv[2]));
        memset(serv_addr.sin_zero, 0, sizeof(serv_addr.sin_zero));
        serv_len = sizeof(serv_addr);

        result = connect(sock, (struct sockaddr *)&serv_addr, serv_len);
        if (result < 0) {
              perror("connect");
              exit(1);
        }
        for (i = 0; i < 10; ++i) {
              size_t n_write, n_read;
              n_write = send(sock, buffer1, sizeof(buffer1), 0);
              n_read = recv(sock, buffer2, sizeof(buffer2), 0);
              printf("[%d]%s\n", n_read, buffer2);
              if (memcmp(buffer1, buffer2, n_read) == 0)
                    printf("== \n");
              else
                    printf("!= \n");
              memset(buffer2, 0, sizeof(buffer2));
        }

        close(sock);
}
int main(void) {
        diag_printf("eCos, uip, test\n\r");
        uip_setup();
        main_application_init();
        while (1) {
              uip_activity();
        }
}
```

2ª Parte

Suporte para canais de comunicação (Sockets)

O uIP não tem capacidade de armazenamento de dados e o seu desenho pressupõe a intervenção da aplicação a cada evento de rede - a chegada de um pacote, a repetição de envio, a confirmação de envio, etc. Este modelo dificulta a programação da aplicação, já que obriga à manutenção de estado entre eventos. Por exemplo, não é fácil utilizar o modelo de múltiplos canais *byte stream* suportado pelo protocolo TCP, sem uma abordagem estruturada.

Nesta etapa é criada uma camada que, assente sobre a interface do uIP, oferece à aplicação uma interface para estabelecimento e utilização de canais de dados do tipo *byte stream*. Esta camada possui internamente, os mecanismos de sincronização necessários para que as *threads* da aplicação bloqueiem em espera passiva, pelo progresso da comunicação.

Interface de utilização:

- Iniciação do módulo de comunicações.
  ```
  void net_init(void);
  ```
- Aceitar uma ligação num porto.
  ```
  Socket * net_server_accept(int port);
  ```
- Estabelecer ligação com o servidor e porto indicados.
  ```
  Socket * net_client_connect(char * server, int port);
  ```

- Eliminar uma ligação.
```
void net_close(Socket * sock);
```
- Enviar de dados numa ligação previamente estabelecida.
```
size_t net_send_block(Socket * sock, char * buffer, size_t length);
void net_send_char(Socket * sock, char c);
```
- Receber dados numa ligação previamente estabelecida.
```
size_t net_recv_block(Socket * sock, char * buffer, size_t length);
char net_recv_char(Socket * sock);
```

Como exercício, sugere-se a implementações das funções **net_send_block** e **net_send_char** omissa no código disponibilizado.

Neste caso, o servidor de eco para teste poderá ser programado como se segue:

```
#include <stdlib.h>
#include <stdio.h>
#include "net.h"

static char buffer[100];

int main(int argc, char * argv[]) {
      int i;
      Socket * sock;

      diag_printf("Server\n");

      while (1) {
            sock = net_server_accept(1234);
            for (i = 0; i < 10; ++i) {
                  size_t n_write, n_read;
                  n_read = net_recv_block(sock, buffer, sizeof(buffer));
                  n_write = net_send_block(sock, buffer, n_read);
                  diag_printf(".");
            }
            net_close(sock);
      }
      return 0;
}

void cyg_user_start( void ) {
      net_init();
}
```

Código anexo: **net.tar (net.h net.c fbuffer.h fbuffer.c**)

3ª Parte

Servidor Web

O servidor WEB aceita ligação no porto 80, processa um pedido HTTP e fecha a ligação.

```
      while (1) {
            Socket * socket = net_server_accept(80);
            stream_set_device(&stream, &stream_socket, socket);
            http_process(&stream);
```

```
            net_close(socket);
      }
```

O processamento do pedido consiste em isolar o campo URI (Universal Resource Identifier), procurar na tabela de recursos do servidor - **`http_table`** e invocar a função associada. Esta função irá actuar sobre o sistema e responder com uma mensagem HTTP contendo um texto em formato HTML.

```
void http_process(Stream * stream) {
      char request[256];
      char line[40];
      char * formdata;
      char * uri;

      /* ler a primeira linha */
      stream_getline(stream, request, sizeof(request));
      TRACE("Request <%s>\r\n", request);

      /* esperar pela linha em branco */
      while ( stream_getline(stream, line, sizeof(line)))
            ;

    char * p;
      uri = p = request + 4;
      while( *p != ' ' && *p != '?' )
            p++;
      if( *p == '?' )
            formdata = p + 1;
      *p = 0;
      if (formdata != NULL ) {
            while( *p != ' ' )
                  p++;
            *p = 0;
      }

      TRACE("table: %p...%d\r\n", http_table, http_table_len);
      http_table_entry * entry;
      int i;
      for (entry = http_table, i = 0; i < http_table_len; ++i) {
            TRACE("try %p: %s\r\n", entry, entry->pattern);
            if (strcmp(uri, entry->pattern) == 0) {
                  TRACE("calling %p: %s\r\n", entry, entry->pattern);
                  if (entry->handler(stream, uri, formdata, entry->arg))
                        break;
            }
            entry++;
      }
      if (i >= http_table_len) {
            TRACE("Not found %s\r\n", uri);
            http_send_html(stream, NULL, NULL, page_not_found);
      }
}
```

A tabela de recursos contém, por cada entrada, a string de identificação do recurso, o ponteiro para a função de processamento do recurso e um parâmetro de contexto para essa função.

```
#define LED_PIN   15

int led_page(Stream * stream, char *uri, char *formdata, void *arg );
```

```
http_table_entry http_table[] = {
      { "/led.html", led_page, LED_PIN }
};

int http_table_len = sizeof_array(http_table);
```

Mensagem de recurso não encontrado:

```
static char page_not_found[] =
"<head><title>Page not found</title></head>\r\n"
"<body><h2>O recurso pretendido nao esta disponivel.</h2></body>\r\n";
```

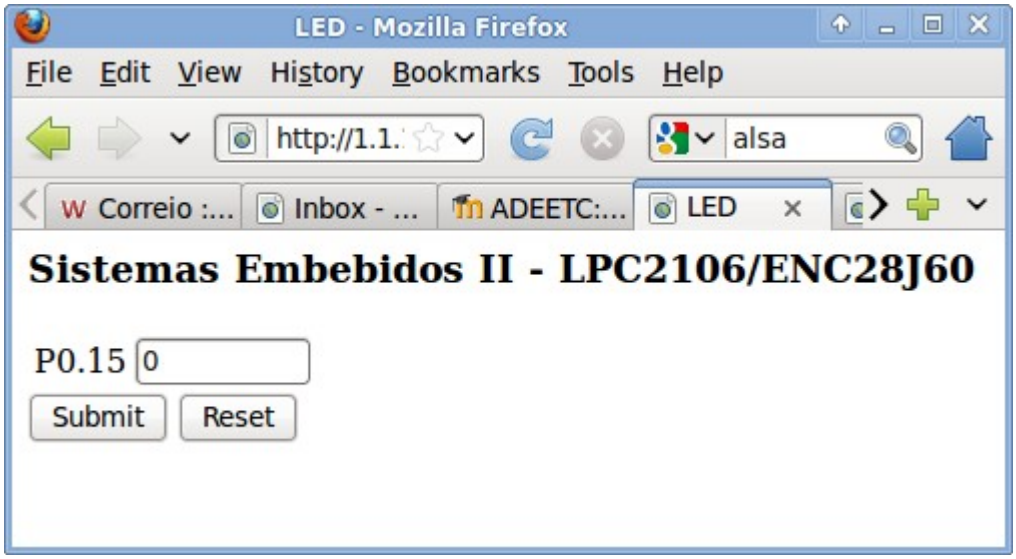
Quando a página contém controlos HTML do tipo “input” e é premido um botão do tipo “submit”, o Browser produz uma mensagem HTTP, em que no campo URI são concatenados os valores dos outros campos input. A função de processamento do recurso começa por processar os valores dos controlos de input, eventualmente actua sobre o sistema, e em seguida gera um texto HTML com valores actualizados para resposta.

```
int led_page(Stream * stream, char *uri, char *formdata, void *arg ) {
      char string[33];
      cyg_uint32 value;

      if (formdata != 0) {
            char * formlist[10], * p;
            formdata_parse(formdata, formlist, sizeof_array(formlist));
            p = formlist_find(formlist, "led");
            if (p != 0)
                  if (*p == '1')
                        HAL_WRITE_UINT32(CYGARC_HAL_LPC2XXX_REG_IO_BASE +
                              CYGARC_HAL_LPC2XXX_REG_IOSET, 1 << (int)arg);
                  else
                        HAL_WRITE_UINT32(CYGARC_HAL_LPC2XXX_REG_IO_BASE +
                              CYGARC_HAL_LPC2XXX_REG_IOCLR, 1 << (int)arg);
      }
      html_begin(stream);
      html_head(stream, "LED", "");
      html_body_begin(stream, "");
      {
            html_heading(stream, 3, "Sistemas Embebidos II - LPC2106/ENC28J60");
            html_form_begin(stream, uri, "" );
            {
            html_table_begin(stream, "" );
            {
                  html_table_row_begin(stream, "" );
                  {
                        html_table_data_begin(stream, "" );
                        stream_printf(stream, "P0.%d", (int)arg);
                        html_table_data_begin(stream, "" );
                        HAL_READ_UINT32(CYGARC_HAL_LPC2XXX_REG_IO_BASE +
                              CYGARC_HAL_LPC2XXX_REG_IOSET, value);
                        sprintf(string, "%d", (value >> ((int) arg) & 1));
                        html_form_input(stream, "text", "led", string,"size=8");
                  }
                  html_table_row_end(stream);
            }
            html_table_end(stream);

            html_form_input(stream, "submit", "submit", "Submit", "");
            html_form_input(stream, "reset", "reset", "Reset", "");
```

```
            }
            html_tag_end(stream, "font");
            html_form_end(stream);
        }
        html_body_end(stream);
        html_end(stream);
        return 1;
}
```

Com base no código anterior construa uma aplicação eCos que permite manipular um LED através WEB Browser.

A imagem da página gerada por este programa é a seguinte:



Código anexo: **`web.tar (http.h http.c stream.h stream.c`**)