Sistemas Embebidos II

uIP

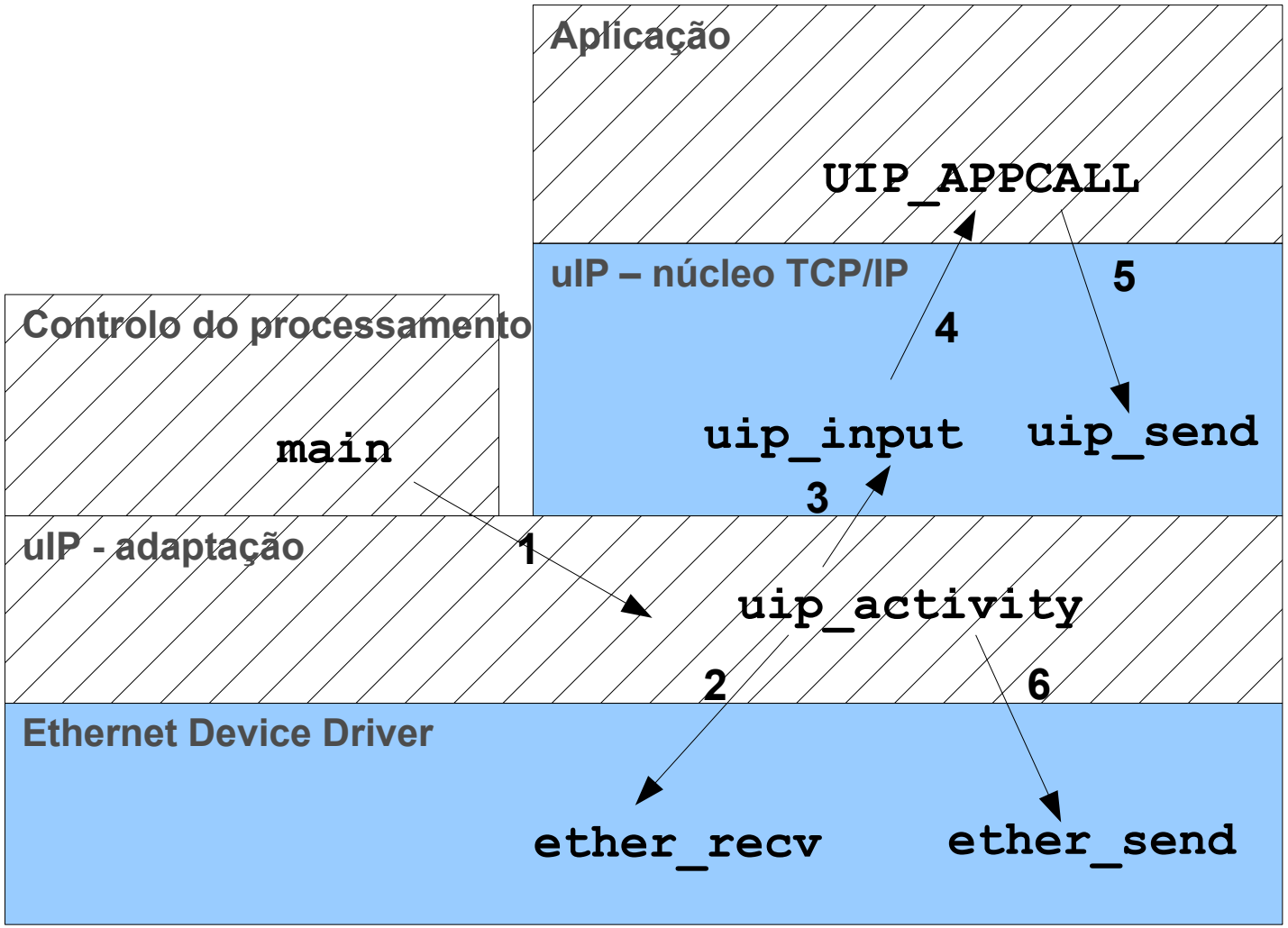
Stack TCP/IP para sistemas embebidos

Ezequiel Conde
ezeq@cc.isel.ipl.pt

Características

- Independente do Sistema Operativo.
- Programação da aplicação do tipo *event-driven*.
- Concebido para adaptação a diversos suportes de programação e de hardware.
- Suporta apenas uma interface.
- Manipula um pacote de cada vez.

Arquitectura (1)



Arquitectura (2)

- O uIP pressupõe a existência e duas funções de acesso ao controlador Ethernet – uma para enviar e outra para receber dados.
- O programa de utilizador desenvolve-se a partir de uma função principal com nome pré-definido - UIP_APPCALL (para ligação estática).
- A função principal da aplicação é chamada pelo uIP quando ocorre algum evento.
- Os eventos são despoletados pela receção de um pacote ou pelo fim de uma temporização.
- A aplicação inquire o uIP para saber que evento ocorreu e agir em conformidade.

Arquitectura (3)

```
main() {
    uip_setup();
    while (1)
        uip_activity();
        ...
}
```

O processamento do stack TCP/IP é feito diretamente a partir do programa principal. A função uip_activity é não bloqueante, isto é, executa todo o processamento possível em cada invocação e retorna. Paralelamente, podem ser executadas outras atividades

uIP - Adaptação

```
uip_setup() {
    uip_setethaddr(ether_addr);
    uip_ipaddr(ipaddr, 10,0,0,4);
}

uip_activity() {
    uip_len = ethernet_recv(uip_buf, UIP_BUFSIZE);
    if (uip_len > 0) {
        uip_input();
        if (uip_len > 0)
            ethernet_send(uip_buf, uip_len);
    } else if (timer_expired(&periodic_timer))
        for (i = 0; i < UIP_CONNS; i++)
            uip_periodic(i);
}
```

Este módulo contém software de iniciação e de adaptação do pacote uIP ao device driver Ethernet.

1.Verificar se foi recebido pacote da rede.
2.Verificar temporizações.
3.Do processamento podem resultar dados para serem enviados.

Arquitectura (4)

uIP - Núcleo

```
uip_input() {
    …
    UIP_APPCALL();
}

#define uip_newdata()
    (uip_flags & UIP_NEWDATA)

void uip_send(void * data, int size) {
    ...
    memcpy(uip_sappdata, data, size);
    ...
}
```

uIP - Aplicação

```
UIP_APPCALL() {
  if (uip_connected())
    …
  if (uip_newdata())
    …
  if (uip_acked())
    uip_send(data, size);
    …
  if (uip_rexmit())
    …
  if (uip_poll())
    …
}
```

O uIP é executado por um módulo de software sem dependências da infraestrutura de processamento.
Dispõe de duas interfaces:

- aplicação:
  - pesquisa de eventos, uip_send, ...
- device driver:
  - uip_input, uip_periodic, ...

O programa de utilizador é organizado segundo o modelo de máquina de estados. Em cada invocação age de acordo com o evento ocorrido e com o estado da aplicação retido em dados estáticos.

Recepção de dados

- O núcleo uIP chama a aplicação ao receber novos dados.
- A aplicação deteta a existência de novos dados com uip_newdata().
- uip_appdata() e uip_datalen() indicam o inicio e a dimensão dos dados recebidos em uip_buffer.
- o uIP não memoriza dados. Ao retornar da aplicação, assume que os dados foram consumidos.
- Eventualmente, a aplicação vai armazenando os dados e guardando estado para processamento posterior.

Envio de dados

- Como o uIP não tem capacidade de armazenamento, antes de enviar dados, a aplicação deve determinar a máxima dimensão de um segmento uip_mss().
- A aplicação deve ser capaz de reproduzir os dados enviados para o caso de retransmissão.
- Os dados são transmitidos em dois passos: primeiro são depositados no *buffer* uip_buffer pela aplicação (uip_send), depois da função da aplicação retornar é que são enviados para o controlador Ethernet.
- Os dados a transmitir são produzidos como resposta a dados recebidos ou por iniciativa da aplicação. Do ponto de vista do funcionamento do programa são sempre produzidos como resposta a um evento produzido por uip_input() ou uip_periodic().

Tipos de dados importantes (1)

- **`eth_addr`** – representa um endereço Ethernet.

```
struct uip_eth_addr {
  u8_t addr[6];
};
```

- **`uip_ipaddr_t`** – representa um endereço IP.

```
typedef u16_t uip_ip4addr_t[2];
typedef u16_t uip_ip6addr_t[8];
#if UIP_CONF_IPV6
typedef uip_ip6addr_t uip_ipaddr_t;
#else /* UIP_CONF_IPV6 */
typedef uip_ip4addr_t uip_ipaddr_t;
```

Tipos de dados importantes (2)

- uip_conn – contém os dados de uma ligação.

```
struct uip_conn {
    uip_ipaddr_t ripaddr;   endereço IP remoto
    u16_t lport;            porto local
    u16_t rport;            porto remoto
    u8_t rcv_nxt[4];        o próximo número de sequência a receber
    u8_t snd_nxt[4];        o último número de sequência enviado
    u16_t len;              a dimensão do segmento previamente enviado
    u16_t mss;              a máxima dimensão de um segmento nesta ligação
    u16_t initialmss;       a dimensão inicial de um segmento
    u8_t sa;                para o calculo do timeout de retransmissão.
    u8_t sv;
    u8_t rto;               timeout de retransmissão
    u8_t tcpstateflags;     flags com o estado da ligação
    u8_t timer;             timer de retransmissão
    u8_t nrtx;              número de retransmissões do último segmento
    uip_tcp_appstate_t appstate;
                    para registo de estado da aplicação relativo a esta ligação
};
```

Manipulação de endereços

- uip_sethostaddr, uip_gethostaddr – endereço IP.
- uip_setdraddr, uip_getdraddr – default router.
- uip_setmask, uip_getmask – mascara de rede.
- uip_setethaddr – definir endereço Ethernet.

Interface device driver

- uip_input – processar pacote IP recebido.
- uip_periodic – processar uma ligação após timeout.

Interface ARP

- **`uip_arp_out()`** – Enviar pedido ARP.
- **`uip_arp_arpin()`** – Processar resposta ARP.
- **`uip_arp_ipin()`** – Aproveitar informação de endereços contida num pacote IP normal.

Interface da aplicação

- **`uip_newdata()`** dados recebidos
- **`uip_acked()`** dados confirmados
- **`uip_rexmit()`** pedido de retransmissão
- **`uip_poll()`** ligação inactiva; passagem do tempo
- **`uip_connected()`** ligação estabelecida
- **`uip_aborted()`** ligação abortada pelo remoto
- **`uip_closed()`** ligação fechada
- **`uip_timedout()`** ligação abortada por execesso de retransmissões

- **`uip_appdata(), uip_datalen()`** definem zona de dados recebidos.
- **`uip_send(ptr, size)`** enviar dados na ligação actual.
- **`uip_mss()`** saber a dimensão máxima dum segmento TCP.
- **`uip_listen(port)`** aceitar ligação
- **`uip_connect(ip, port)`** estabelecer ligação

Conversão

- **`HTONS`** – converte valores a 16 bits do formato host para o formato rede (para constantes).
- **`htons`** – o mesmo que a anterior mas para variáveis.
- **`ntohs`** – converte do formato rede para o formato host.
- **`uip_ipaddr(addr, byte0, byte1, byte2, byte3)`** – forma um endereço IP a partir de quatro bytes.

Variáveis importantes

- **`uip_buffer`** – ponteiro para o buffer que aloja todo o pacote recebido ou a enviar.
- **`uip_len`** - dimensão do pacote que se encontra em uip_buffer.
- **`uip_conn`** – representa a ligação actual.

uIP Adaptação - iniciação

```
static struct uip_eth_addr ether_addr = {
    {0x02, 0x65, 0x7A, 0x65, 0x71, 00}
};

void uip_setup() {
    uip_ipaddr_t ipaddr;

    timer_set(&periodic_timer, CLOCK_SECOND / 2);
    timer_set(&arp_timer, CLOCK_SECOND * 10);

    ethernet_init(ether_addr.addr);

    uip_init();

    uip_setethaddr(ether_addr);

    uip_ipaddr(ipaddr, 10,0,0,4);
    uip_sethostaddr(ipaddr);
    uip_ipaddr(ipaddr, 10,0,0,0);
    uip_setdraddr(ipaddr);
    uip_ipaddr(ipaddr, 255,255,255,0);
    uip_setnetmask(ipaddr);
}
```

Iniciar os Timers

Iniciar o driver Ethernet

Definir endereço MAC da interface.

Definir endereço da interface.

Definir endereço da gateway.

Definir mascara da rede.

uIP Adaptação - actividade (1)

```
#define BUF ((struct uip_eth_hdr *)uip_buf)

void uip_activity() {
    int i;
    uip_len = ethernet_recv(uip_buf, UIP_BUFSIZE, CLOCK_SECOND / 2);
    if (uip_len > 0) {
        if (BUF->type == htons(UIP_ETHTYPE_IP)) {
            uip_arp_ipin();
            uip_input();
            if (uip_len > 0) {
                uip_arp_out();
                ethernet_send(uip_buf, uip_len);
            }
        }
        else if (BUF->type == htons(UIP_ETHTYPE_ARP)) {
            uip_arp_arpin();
            if (uip_len > 0) {
                ethernet_send(uip_buf, uip_len, 10);
            }
        }
```

Verificar se há pacote recebido

Actualizar cache ARP (aproveitar)

Processar pacote IP

Tratar do endereço de destino

Processar um pedido ARP

uIP Adaptação - actividade (2)

```
    } else if (timer_expired(&periodic_timer)) {
        timer_reset(&periodic_timer);
        for (i = 0; i < UIP_CONNS; i++) {
            uip_periodic(i);
            if (uip_len > 0) {
                uip_arp_out();
                ethernet_send(uip_buf, uip_len);
            }
        }
        for (i = 0; i < UIP_UDP_CONNS; i++) {
            uip_udp_periodic(i);
            if (uip_len > 0) {
                uip_arp_out();
                ethernet_send(uip_buf, uip_len);
            }
        }
        if (timer_expired(&arp_timer)) {
            timer_reset(&arp_timer);
            uip_arp_timer();
        }
    }
}
```

Processar ligações TCP: pedir novos dados, retransmitir ou terminar ligações

Processar ligações UDP

Vai eliminar entradas antigas da tabela ARP

uIP Aplicação

```
#include <cyg/kernel/kapi.h>
#include <string.h>

#include "uip-conf.h"
#include "uip.h"

static void main_application_init() {
    uip_listen(HTONS(1234));
}

void main_application(void) {
    if (uip_newdata())
        uip_send(uip_appdata, uip_datalen());
}

int main(void) {
    diag_printf("Test uip\n\r");
    uip_setup();
    main_application_init();
    while (1)
        uip_activity();
}
```

Ficheiro de configuração do uIP

Aceitar ligação no porto 1234

A aplicação foi chamada por dados recebidos?

Ecoar os dados recebidos

Iniciar uIP: endereços, hardware

Iniciar a aplicação

Manter a actividade

Referências

- http://www.sics.se/~adam/uip/index.php/Main_Page