Xarxes de Comunicació

Pràctica 2 - Protocol d'accés al medi / Xarxa d'àrea local

Francisco del Águila López

Octubre 2013

Escola Politècnica Superior d'Enginyeria de Manresa
Universitat Politècnica de Catalunya

1 Objectiu

L'objectiu d'aquesta pràctica és definir un mòdul en C que implementi un protocol no fiable en el nivell d'enllaç quan el medi és un canal comú compartit.

2 Punt de partida

2.1 Mòdul Ether

La natura de la transmissió per àudio en Morse fa que el canal àudio sigui únic tant en el cas d'una comunicació punt a punt com en el cas de comunicacions en xarxa local. Aquest fet provoca que les comunicacions a nivell d'enllaç tinguin més sentit a nivell de blocs de bytes que a nivell de byte. Transmetre un bloc de bytes de forma compacta facilita la construcció de la trama (unitat de dades d'enllaç), la seva delimitació i el seu posterior tractament. Per aquest motiu la capa física original que oferia servei a nivell de byte es transformarà per oferir servei a nivell de bloc de bytes. Aquesta modificació inicial de la capa física servirà també per ampliar el conjunt de funcions ofertes i disposar, per exemple, de la funció *Ether_bloc_can_put()* que determina si el pot transmetre o no un missatge.

Així, el fitxer de capçalera del nou mòdul *Ether* és:

```
#ifndef ETHER_H
#define ETHER_H
#include <inttypes.h>
#include <stdbool.h>

typedef uint8_t *block_morse;
typedef void (*ether_callback_t)(void);

void ether_init(void);

bool ether_can_put(void);
void ether_block_put(const block_morse b);

bool ether_can_get(void);
void ether_block_get(block_morse b);
void on_message_received(ether_callback_t m);
void on_finish_transmission(ether_callback_t f);

#endif
```

typedef uint8_t *block_morse És una definició de tipus que contempla el bloc de caràcters codificats en ASCII que s'enviaran pel canal Morse. La capa d'enllaç serà la responsable de crear una variable en forma de taula de *block_morse* amb el contingut dels caràcters que es voldrà enviar o rebre. En el moment de definir la mida d'aquesta taula, es considerarà el valor màxim de la cua existent al mòdul *Ether* per poder enviar els caràcters. En aquest cas, considerarem un valor de 32 caràcters (incloent els bytes afegits de control). El motiu de la nova definició de tipus és perquè els valors vàlids que pot contenir aquest *block_morse* són: els caràcters corresponents als números 0..9 i els caràcters corresponents a les lletres majúscules A..Z

void ether_init(void) Serveix per inicialitzar el canal físic.

bool ether_can_put(void) Es farà servir per comprovar si és possible la transmissió d'un *block_morse*. En el cas que s'està treballant (transmissió punt a punt), vindrà determinada essencialment per si s'està rebent algun *block_morse* enviat per un altre node.

void ether_block_put(const block_morse b) Aquesta funció es fa servir per transmetre un *block_morse*. Se li passa com a paràmetre l'apuntador a taula de *uint8_t* on són els caràcters que es vol transmetre. S'ha de considerar que l'indicador de fins a on està plena aquesta taula ve determinat pel byte *null* que farà de sentinella.

bool ether_can_get(void) És una funció que indicarà quan està disponible tot un *block_morse*. Fins que no ha arribat tot el block no es fa certa. Funciona diferent

a la original, que indicava disponibilitat de lectura a partir del moment que es rebia el primer caràcter.

void ether_block_get(block_morse b) Aquesta funció ofereix la possibilitat de llegir el contingut del *block_morse* rebut. El protocol d'enllaç ha de reservar una taula de tipus *block_morse* i aquesta funció la omplirà amb el contingut del *block_morse* rebut. El paràmetre que se li passa és precisament l'apuntador a aquesta taula. El sentinella de final de dades serà igualment el caràcter *null*.

void on_message_received(ether_callback_t m) Aquesta és una funció que permet instal·lar una funció de callback que serà cridada quan sigui rebut un missatge. Si s'instal·la el callback la funció ether_can_get() perd la seva utilitat. A més, utilitzant aquest callback evitem la necessitat d'haver d'estar contínuament enquestant el fet de la rebuda d'un missatge.

void on_finish_transmission(ether_callback_t f) Permet instal·lar un callback que serà cridat quan es produeix la finalització de la transmissió d'un missatge.

Les funcions ether_block_put() i ether_block_get() no retornen res per simplificar, però es podria fer una implementació que retornessin la quantitat real de bytes que han pogut enviar o rebre. Això serviria per poder fer un control dels possibles errors que pugui haver. De la mateixa manera, només hi ha com a paràmetre l'apuntador a la taula de bytes, però es podria afegir també la mida fins on està plena aquesta taula. En el nostre cas no es fa necessària aquesta característica ja que el sentinella ja fa aquesta funció. En canvi es fa imprescindible en el cas que la transmissió fos binaria i el caràcter *null* es podria confondre amb una dada més o amb el sentinella.

2.2 Mòdul Serial

Per facilitar la comunicació serie, també s'ofereix aquest mòdul

```
#ifndef SERIAL_H
#define SERIAL_H

#include <inttypes.h>
#include <stdbool.h>

void serial_init(void);

uint8_t serial_get(void);
void serial_put(uint8_t c);
bool serial_can_read(void);

#endif
```

Recordeu que aquest mòdul fa de *driver* de la interfície sèrie. Disposa d'un *buffer* de 32 bytes.

void serial_init(void) Inicialitza les comunicacions sèrie.

bool serial_can_read(void) Indica si hi ha bytes disponibles per ser llegits en el *buffer* de recepció.

unit8_t serial_get(void) Buida un caràcter del buffer de recepció. S'ha de cridar quan hi ha dades disponibles.

void serial_put(uint8_t c) Envia per port sèrie el caràcter que se li passa com a paràmetre.

2.3 Mòdul timer

Aquest mòdul és el mateix que es planteja a la pràctica de "Control semafòric de cruïlla amb comunicació morse: mestre" de l'assignatura de Programació de Baix Nivell. El fitxer de capçalera és el següent

```
#ifndef TIMER_H
#define TIMER_H

/*
 * This module implements a time dispatcher with a resolution
 * of 10 ms. It is based on callbacks. That is, functions which
 * are called after a specific (temporal) event ocurred.
 */
#define TIMER_MS(ms) (ms/10)
#define TIMER_ERR    -1

typedef void (*timer_callback_t)(void);
typedef int8_t timer_handler_t;

void timer_init(void);
void timer_cancel(timer_handler_t h);
void timer_cancel_all(void);
timer_handler_t timer_ntimes(uint8_t n, uint16_t ticks, timer_callback_t f);
timer_handler_t timer_every(uint16_t ticks, timer_callback_t f);
timer_handler_t timer_after(uint16_t ticks, timer_callback_t f);

#endif
```

S'ofereix un servei de temporització amb aquest mòdul. Essencialment consisteix en executar una funció f() planificant la seva execució per d'aquí a k ms. El nombre màxim de ticks és un uint16_t que amb una resolució de 10ms dóna un temps màxims de 655 segons.

void timer_init(void) Inicialitza el mòdul. Cal cridar-la com a mínim una vegada abans d'usar el mòdul. Només pot cridar-se amb les interrupcions inhabilitades.

void timer_cancel(timer_handler_t h) Cancel·la l'acció planificada identificada per h. Si h no és un handler vàlid, no fa res.

void timer_cancel_all(void) Cancel·la totes les accions planificades del servei.

timer_handler_t timer_after(uint16_t ticks, timer_callback_t f) Planifica la funció f() per ser executada al cap de ticks ticks. Retorna un handler que identifica aquesta acció planificada o bé val TIMER_ERR en cas que l'acció no es pugui planificar per alguna raó.

timer_handler_t timer_every(uint16_t ticks, timer_callback_t f) Planifica la funció f() per a ser executada cada ticks ticks de manera indefinida.

timer_handler_t timer_ntimes(uint8_t n, uint16_t ticks, timer_callback_t f) Planifica la funció f() per a ser executada cada ticks ticks n vegades. En cas que n sigui zero s'interpreta que la funció ha de ser cridada indefinidament.

2.4 Mòdul Error_Morse

Aquest mòdul és el que s'ha desenvolupat a la pràctica anterior.

2.5 Generació de número aleatori

En aquest protocol d'accés al medi hi ha la necessitat d'esperar un temps aleatori. Per generar aquest temps es farà servir la funció rand() de la llibreria <stdlib.h>. Mireu la documentació a [avr-libc].

3 Protocol d'accés al medi

El protocol que s'ha d'implementar és un CSMA (accés múltiple amb detecció de portadora). Per tant, si en el moment que una estació vol transmetre es detecta que el canal està ocupat s'esperarà un temps aleatori i ho tornarà a intentar. Si detecta canal lliure realitzarà la transmissió.

Els nodes de la xarxa local seran tant transmissors com receptors.

No es farà cap tipus de control dels errors. Per tant aquesta capa oferirà un servei no fiable. L'única gestió que es farà amb els erros és que si es detecta una trama errònia es descartarà aquesta trama directament.

Existeix la necessitat d'identificar tots els nodes existents a la xarxa local. Per resoldre aquest problema s'assignarà a cada node una adreça corresponent a un caràcter morse. D'aquesta manera amb un únic byte identificarem als nodes. Es recorda que els caràcters morse és l'abecedari en majúscules i els números del 0 al 9.

3.1 Recepció de missatges

El comportament que ha de tenir el protocol en el moment de rebre un missatge és el següent:

- Analitza si el missatge rebut (trama) és un missatge vàlid. Això ho farà comprovant els bits de redundància. Si no és vàlid el descarta i no fa res més.
- Analitza si el destinatari del missatge és ell. Si no és així descarta el missatge i no fa res més.
- Finalment extreu les dades i l'adreça d'origen. Entrega aquestes dades a la capa superior.

3.2 Transmissió de missatges

El comportament del protocol per transmetre un missatge és el següent:

- Un missatge per transmetre arriba quan la capa superior fa una crida a lan_block_put(). Quan això passa, primer construeix la trama que ha d'enviar. Testeja si la pot transmetre. En cas positiu el transmet i acaba.
- En cas negatiu calcula un temps aleatori entre 1 i 10 segons, i activa un event temporal (amb el mòdul timer) per tornar a fer l'intent de transmissió passat aquest temps aleatori.
- Repeteix aquest procés 3 vegades com a màxim fins que aconsegueix la transmissió o bé descarta aquesta transmissió i encén el led indicant l'error.

3.3 Unitat de Dades de Protocol

La unitat de dades de protocol (PDU) en aquest cas rep el nom de trama. Un dels aspectes més rellevants en la especificació d'un protocol és la definició de com han de ser aquestes trames. En aquesta pràctica només hi ha un tipus de trama, ja que no hi ha necessitat de trames de control.

La trama està formada pels següents camps:

Adreça d'origen És el camp corresponent a identificar el node propi. Es reserva un caràcter morse pel seu valor. S'ha de gestionar adequadament aquestes adreces per evitar la duplicitat d'adreces a la mateixa xarxa local.

Adreça de destí És el camp corresponent a identificar el node a qui se li envia la informació. Es reserva un caràcter morse pel seu valor.

Camp de dades És un camp de mida variable múltiple de Byte, que contindrà les dades que s'han de transportar. En general, contindrà la unitat de dades de protocol de la capa superior, però en el cas de la pràctica, la capa superior directament serà la capa d'aplicació, per tant contindrà els missatges que es volen transmetre.

Camp de FCS És un camp que ocupa 2 caràcters. Conté el Checksum / CRC calculat segons la pràctica anterior. Per conveni es considera que el caràcter (byte) de més pes s'envia primer i el segon caràcter és el de menys pes. FCS: Seqüència de Comprovació de Trama.

4 Implementació mòdul Lan

El mòdul Lan és on s'ha d'implementar el protocol de xarxa local que s'ha definit en aquesta pràctica. Aquest mòdul farà servir els mòduls Ether, Error_Morse i Timer. Aquest mòdul ha d'oferir funcions (servei) per permetre una comunicació no fiable entre diferents nodes de la xarxa local, per tant el fitxer de capçalera podria ser el següent

```
#ifndef LAN_H
#define LAN_H
#include <inttypes.h>
#include <stdbool.h>
#include <pbn.h>

typedef void (*lan_callback_t)(void);

void lan_init(uint8_t no);

bool lan_can_put(void);
void lan_block_put(const block_morse b, uint8_t nd);

//bool lan_can_get(void);
//uint8_t lan_block_get(block_morse b);
void on_lan_received(lan_callback_t l);

#endif
```

Com es pot observar, aquest mòdul ofereix les mateixes funcions que ofereix el mòdul Ether. Però en aquest cas permet identificar amb quin node es vol realitzar la comunicació.

La funció lan_init(no) inicialitza el mòdul i per tant el protocol. Té com paràmetre *no* l'adreça del node.

Quan es vol transmetre un bloc de dades utilitzant el mòdul Ether, es fa servir la funció ether_block_put() que requereix com a paràmetre un apuntador a una taula de tipus *block_morse*. Per aquest motiu s'ha de reservar un espai de memòria on estigui aquesta taula. Aquesta reserva de memòria es podria fer dinàmicament (*malloc()*)o estàticament (variable global). Per simplicitat es farà estàticament, per tant en el mòdul s'ha de definir

una variable global que serà la taula de *block_morse* que s'utilitzarà per crear la trama de transmissió.

De la mateixa manera, quan es vol rebre un bloc de dades utilitzant el mòdul Ether, es fa servir la funció ether_block_get(), per tant també s'ha de definir una variable global que serà la taula de *block_morse* per a la recepció.

En aquest protocol es fan successius intents de transmissió mentre es troba el canal ocupat. El número màxim de reintents serà de 3.

La funció lan_block_put(b, nd) té dos paràmetres. El primer és la taula block_morse que es vol transmetre. El segon és l'adreça del node a qui va dirigida.

La funció lan_block_get() té com a paràmetre la taula block_morse on es recollirà el missatge. També retorna un resultat que és l'adreça del node que ha enviat el missatge. Aquesta funció està molt relacionada amb lan_can_get() ja que aquesta última indica quan hi ha disponible un block de dades per llegir i per tant quan es pot cridar a lan_block_get().

Una alternativa a implementar les funcions lan_can_get() i lan_block_get() és instal·lar una funció de callback on_lan_received() que serà cridada just quan hi hagi dades disponibles. Per aquest motiu les dues funcions anteriors estan comentades en el fitxer de capçalera, ja que la implementació recomanable i elegant és fer servir les funcions de callback.

4.1 Implementació de la capa lan

Cada node pot estar en dos possibles estats: "pendent_enviar" o "esperant". En l'estat "esperant" simplement espera que la capa superior li demani l'enviament d'algun missatge. Mentrestant està pendent de la rebuda de possibles missatges. Si el missatge que rep no és per ell el descarta i no fa res més, però si el missatge que rep sí que és per ell, activa la crida del callback corresponent i/o indica que hi ha missatges pendents de ser llegits.

Si la capa superior li demana l'enviament d'un missatge, el node passa a l'estat "pendent_enviar" fins que aconsegueix l'enviament. Si ho aconsegueix just en el mateix moment, torna a l'estat "esperant" de nou. Però si això no és possible, ha de tornar-ho a intentar 2 vegades més programant events temporitzats. Quan ho aconsegueix torna a l'estat "esperant", de igual manera que si no ho aconsegueix amb els intents disponibles però indica la condició d'error.

Si el node està en "pendent_enviar" no permet que la capa superior li sol·liciti un nou enviament.

5 Capa d'aplicació

El protocol de xarxa local d'aquesta pràctica oferirà servei a les capes superiors. En aquest cas, la capa superior ja serà directament la capa d'aplicació. Per tant per poder fer una aplicació final complerta s'ha de definir que es vol que faci la aplicació.

L'aplicació consistirà en un xat entre tots els nodes de la xarxa local. La interfície d'usuari serà el port serie que es connectarà a un ordinador amb el picocom com aplicació terminal o bé l'aplicació cutecom que permet un control més precís del que s'està fent i una presentació més estructurada. Per tant, per la banda del PC s'aprofiten aplicacions ja acabades. Falta definir l'aplicació a la banda del AVR.

Els requeriments de l'aplicació a la banda del AVR són els següents:

- La visualització de resultats cap a l'usuari és per la via d'escriure en el port serie. Per tant cada línia que s'escriu en el port serie serà tant els missatges rebuts com els missatges enviats. Les línies estaran acabades en un retorn de carro
- En aquestes línies s'ha d'indicar qui és l'origen i el destí del missatge de manera que els primers caràcters de la línia seran "No->Nd:" on No és el caràcter morse (uint8_t) corresponent a l'adreça del node origen i Nd és el caràcter morse corresponent a l'adreça del node destí.
- La resta de la línia serà el contingut del missatge.
- La rebuda de missatges serà a través del port sèrie igualment. Aquest missatges són els que es volen transmetre per part del node en qüestió. Amb la intenció de simplificar al màxim, quan es vulgui enviar un missatge a algun node, la manera de indicar-ho serà: teclejar l'adreça del node destí, teclejar ":" i seguidament el missatge acabat amb un retorn de carro "\n". Un cop fet això es mostrarà una línia amb el format indicat en el segon punt. Si es produeix un error s'escriurà "ERROR" per pantalla i es tornarà a començar. Opcionalment es pot fer servir l'enviament de la lletra "r" en minúscula per indicar un reset en l'enviament i tornar a començar un nou missatge. Quan s'enviï "r" el missatge de resposta serà "RESET".

5.1 Implementació de la capa d'aplicació

Des del punt de vista de la transmissió, s'ha anar processant els caràcters que es rebent pel port serie fins trobar un retorn de carro o "r". Quan es rep retorn de carro vol dir que hi ha un missatge per poder ser enviat. Si el missatge té el format adequat se solicita a la capa lan la seva transmissió i no se surt d'aquí fins que es fa possible la transmissió. Això vol dir que lan_can_put() és cert.

Des del punt de vista de la recepció i aprofitant la implementació a través del callback, l'aplicació tan sols ha de presentat el missatge rebut pel port sèrie en el format especificat anteriorment.

Si per contra, es fa una implementació no basada en callback, el programa principal ha d'anar contínuament consultant si hi ha dades disponibles amb lan_can_get() i quan és així, presentar-les pel port sèrie.

6 Treball pràctic

1. Dibuixa el graf de la maquina d'estat corresponent a la transmissió a la capa d'aplicació i a la capa lan.
2. Defineix les possibles funcions privades del mòdul lan i del programa principal per estructurar millor el disseny de manera que quedi el més simple possible.
3. Dissenya l'aplicació final suposant que existeix el mòdul Lan.
4. Dissenya les funcions de recepció del mòdul Lan. Comprova l'aplicació receptora i el seu correcte funcionament amb un programa transmissor de test que generi les trames de manera concreta. Considera que no existeix el camp de checksum.
5. Dissenya les funcions de transmissió del mòdul Lan. Comprova l'aplicació transmissora i el seu correcte funcionament amb un altra node transmissió que provoqui la ocupació del canal aleatoriament. Considera que no existeix el camp de checksum.
6. Amplia els apartats 3 i 4 amb el checksum.
7. Comprova el correcte funcionament combinant la transmissió i la recepció.

Referències

[avr-libc] `http://www.nongnu.org/avr-libc/user-manual/modules.html`