Enginyeria de Sistemes TIC

Xarxes de Comunicació

Pràctica 3 - Protocol d'enllaç / Transferència fiable

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 fiable en el nivell d'enllaç. L'enllaç que es considera és punt a punt entre dos dispositius.

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 oferides 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 l'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 *\0* 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 l'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 *uint8_t* de manera que *block_morse* sigui l'apuntador d'aquesta taula. Aquesta funció omplirà la taula amb el contingut del missatge rebut. El sentinella de final de dades serà igualment el caràcter *\0*.

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 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. Això es fa per simplificar-les, 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 del possibles errors. De la mateixa manera, només hi ha un únic paràmetre de la funció, que és l'apuntador a la taula de *uint8_t*, però es podria afegir també com a paràmetre la mida de 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 *\0* es podria confondre amb una dada més o bé 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 5 ms. It is based on callbacks. That is, functions which
 * are called after a specific (temporal) event ocurred.
 */
#define TIMER_MS(ms) (ms/5)
#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.

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 primera pràctica.

3 Protocol unidireccional

En un primer moment simplificarem el problema de la creació d'un protocol fiable considerant que la transferència d'informació va en un sol sentit. Això no vol dir que un dispositiu només sigui transmissor i l'altre receptor, sinó que els dos dispositius seran transmissors i receptors però que els missatges a nivell d'aplicació (les dades que es volen transmetre) només aniran en un sol sentit.

Per tant, el rol de transmissor el tindrà el dispositiu que vol enviar missatges cap a l'altre (encara que hagi de rebre les trames de confirmació) i el rol de receptor el tindrà qui rep els missatges (encara que hagi de transmetre les trames de confirmació).

Degut al poc retard que hi ha a la comunicació i sobretot a que l'ample de banda (velocitat de transmissió) és molt baix en el canal Morse que es fa servir, s'implementarà un protocol de tipus parada / espera. En aquest protocol no és possible la transmissió d'una nova trama de dades fins que no s'hagi confirmat correctament la rebuda de la trama de dades anterior.

3.1 Dispositiu transmissor

Aquest dispositiu haurà d'implementar la màquina d'estats vista a classe que implementa un protocol fiable. S'ha d'implementar la màquina d'estats que considera tant els errors en les trames com la pèrdua de trames.

3.2 Dispositiu receptor

En aquest cas s'implementarà la màquina d'estats receptora que considera errors i pèrdua de trames.

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 el cas de la pràctica, es pot classificar els tipus de trames en:

1. *Trames de dades:* Són les que transporten la informació corresponent als missatges que es volen transmetre. Tenen un camp específic on viatgen aquestes dades.
2. *Trames de control:* No contenen cap tipus d'informació. Només es fan servir per gestionar el correcte funcionament del protocol. En el cas de la pràctica són les trames encarregades de realitzar les confirmacions de la rebuda de les dades.

3.3.1 Trames de dades

Estan formades pels següents camps:

Numeració de trama És el camp corresponent a identificar les trames. En el cas del protocol parada / espera, només caldria un únic bit per identificar les trames, però el canal físic de comunicació està basat en caràcters morse, per tant es reservarà un caràcter (Byte) per identificar les trames. Els caràcters que es faran servir per identificar les trames són "0" i "1".

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 Checksum És un camp que ocupa 2 caràcters. Conté el Checksum 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.

3.3.2 Trames de control

Són trames de mida molt petita. Es fan servir per indicar la confirmació o reconeixement de les trames de dades enviades. Per simplificar i aprofitar la numeració de trames només es defineixen trames de confirmació on el número de trama que es confirma indicarà si realment és la confirmació de la trama de dades que correspon o bé està indicant una NO confirmació.

Un aspecte a tenir en compte és la distinció entre trames de dades i trames de control quan un dispositiu rep una trama. Aquesta distinció es fa necessària quan la transferència de informació és bidireccional. Es podria considerar que si el camp de dades té mida nul·la, la trama rebuda és una trama de control però per simplificar aquesta distinció aprofitarem l'excés de bits del camp de numeració de trama per definir nous valors de numeració de trama i que al mateix temps indiquin que són trames de control.

En aquest cas, els camps són:

Numeració de trama Té mida d'un caràcter (byte). Per confirmar la trama de dades "0" aquest camp contindrà el caràcter "A". Per confirmar la trama de dades "1" aquest camp contindrà el caràcter "B". D'aquesta manera, aquest camp compleix la doble funció de identificar les trames que el volen confirmar i permetre la distinció entre trames de dades i de control. Es recorda que en cas de voler enviar una NO confirmació s'enviarà una confirmació de la trama amb la identificació que no correspon (Si la trama de dades rebuda té identificador "0" i es vol enviar una NO confirmació, s'enviarà una trama de control amb identificador "B").

Camp de Checksum És un camp que ocupa 2 caràcters. Conté el Checksum 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.

Com a conclusió, aquestes trames tindran mida fixe de 3 caràcters sempre.

4 Implementació mòdul Frame

El mòdul Frame és on s'ha d'implementar el protocol fiable 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 fiable, per tant el fitxer de capçalera podria ser el següent

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

typedef void (*frame_callback_t)(void);

void frame_init(void);

bool frame_can_put(void);
void frame_block_put(const block_morse b);

//bool frame_can_get(void);
void frame_block_get(block_morse b);
void on_frame_received(frame_callback_t l);

#endif
```

Com es pot observar, aquest mòdul ofereix les mateixes funcions que ofereix el mòdul Ether. Però en aquest cas ens assegurem que el servei que ofereix és fiable.

En aquesta pràctica s'implementarà el protocol de manera unidireccional, com ja s'ha explicat en l'apartat anterior. Això implica que s'ha de definir un dispositiu transmissor i un altre dispositiu receptor. Això implica que el dispositiu transmissor només farà servir les funcions frame_can_put() i frame_block_put(), en canvi el dispositiu receptor només farà servir les funcions que permet instal·lar el *callback* on_frame_received() (o bé frame_can_get()) i frame_block_get() en les seves aplicacions. Aquest fet simplifica les màquines d'estat que s'han de crear en el mòdul de tal manera que encaixen exactament amb les màquines d'estat vistes a classe de teoria. Quan es faci un disseny bidireccional aquestes màquines d'estat s'han de modificar lleugerament per determinar si estan en mode de transmissió o de recepció.

La funció frame_init() inicialitza el mòdul i per tant el protocol. Això implica que deixarà les màquines d'estat en l'estat inicial. En l'estat inicial el protocol comença la transmissió amb el número de trama "0", per tant el receptor espera rebre aquesta trama, que reconeixerà amb un número de trama "A".

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 *uint8_t* amb apuntador *block_morse* que

s'utilitzarà per crear la trama de transmissió. Com ja s'ha comentat varies vegades, al ser un protocol de parada / espera només és necessari l'espai per una única trama, ja que no es podrà transmetre cap altre fins que aquesta no sigui reconeguda totalment.

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 *uint8_t* amb apuntador *block_morse* per a la recepció.

En aquest protocol es fan successives retransmissions cada vegada que hi ha un problema amb la trama de dades enviada. El número de transmissions successives totals que es farà amb una mateixa trama serà de 3. Si després de 3 intents no s'ha aconseguit la transmissió, s'encendrà el led indicant la condició d'error i acaba el programa.

El valor de *timeout* del temporitzador el fixarem de manera que no es produeixin *timeouts* abans de temps.

5 Capa d'aplicació

El protocol d'enllaç 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.

La aplicació serà la més simple possible i consisteix en la transmissió de missatges de text entre els dos dispositius. Es a dir, s'implementarà un xat. Els dos dispositius Arduino estaran connectats pel canal Morse entre ells i és on es fa servir el protocol d'enllaç dissenyat. Cadascun dels Arduino estarà connectat a un PC a través del canal sèrie. El PC farà la funció simplement de terminal amb una aplicació de terminal com pot ser el *picocom o cutecom.*

Ja que les capes inferiors estan estructurades en forma de blocs de informació per fer l'aplicació el més eficient possible, els missatges també es consideraran en blocs. Per aquest motiu no es farà la transmissió de cada caràcter rebut pel port sèrie sinó que es farà la transmissió de tot un bloc quan es detecti el caràcter especial de retorn de carro. Cada bloc de dades rebut (cada missatge) es presentarà per pantalla del terminal com una línia nova.

5.1 Implementació de la capa d'aplicació

5.1.1 Aplicació transmissora

Aquesta aplicació consisteix en que tot el que arribi pel port serie es va guardant en una taula de *uint8_t* amb apuntador *block_morse* fins trobar un retorn de carro. En aquest moment es fa una crida a la funció frame_block_put() i es passa com a paràmetre l'apuntador d'aquesta taula, que serà un *block_morse*. Quan la funció frame_can_put() retorna True vol dir que el missatge s'ha transmès correctament. En cas que es produeixi algun

error, el mòdul Frame s'encarrega d'encendre el led. Quan la funció frame_can_put() retorna True també vol dir que es pot enviar el següent missatge que es rebi pel port sèrie.

5.1.2 Aplicació receptora

En el cas de no fer servir el *callback*, l'aplicació receptora estarà esperant fins que la funció frame_can_get() retorna True. En aquest moment vol dir que es pot cridar a la funció frame_block_get() i omplirà una taula de *uint8_t* amb apuntador *block_morse* que se li passa com a paràmetre. En el cas de fer servir el *callback*, l'aplicació receptora instal·larà el *callback* a partir del moment que vulgui rebre missatges. Quan el *callback* sigui cridat (produït per la rebuda d'un missatge des de la capa inferior) el primer que ha de fer és cridar a la funció frame_block_get() i així omplir la taula de *uint8_t* amb apuntador *block_morse* que se li passa com a paràmetre.

La següent acció és buidar pel port serie el contingut d'aquesta taula, incloent el retorn de carro i salt de línia que farà que es visualitzi correctament.

6 Treball pràctic

1. Determina un valor adequat pel *timeout*.
2. Dissenya l'aplicació transmissora suposant que existeix el mòdul Frame.
3. Dissenya l'aplicació receptora suposant que existeix el mòdul Frame.
4. Dissenya les funcions de recepció del mòdul Frame. Comprova l'aplicació receptora i el seu correcte funcionament amb un programa transmissor de test que generi les trames de manera concreta.
5. Dissenya les funcions de transmissió del mòdul Frame. Comprova l'aplicació transmissora i el seu correcte funcionament amb un programa receptor que generi les trames de manera concreta.
6. Comprova el correcte funcionament de les aplicacions transmissora i receptora comunicant-se entre elles.
7. Fusiona les aplicacions transmissora i receptora en una única. Les màquines d'estat que s'han definit al mòdul Frame han de compartir l'event de la rebuda de missatges i saber diferenciar si el missatge és per l'autòmat receptor o per l'autòmat transmissor. D'aquesta manera amb una única aplicació es pot rebre i transmetre.
8. *Opcional:* Implementa una arquitectura de comunicacions de 4 capes incloent per sota de la capa Frame la capa Lan de la pràctica anterior. Realitza les modificacions oportunes en les funcions dels fitxers de capçalera dels mòduls que es fan servir.