Enginyeria de Sistemes TIC

Xarxes de Comunicació

Pràctica 3 - Protocol d'enllaç / Transferència fiable (connexió manual)

Francisco del Águila López

Setembre 2022

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

1 Objectiu

L'objectiu d'aquesta pràctica és definir un mòdul en C (capa Frame) que implementi un protocol fiable per sobre del nivell d'accés al medi (capa Lan). Aquest protocol permetrà un intercanvi de dades entre dos dispositius en una xarxa d'àrea local basada en un canal de comunicació infraroja amb morse.

2 Punt de partida

Es disposa de diferents mòduls entre pràctiques anteriors i la llibreria libpbn.a:

Block_Ether: És qui implementa la capa física. Aquest mòdul es farà servir essencialment pel mòdul Lan. En aquesta pràctica, la interacció amb el mòdul Ether serà nula, ja que ho farà el mòdul Lan, respectant estrictament una estructura per capes.

Serial: Permet enviar i rebre dades pel port sèrie implementant un driver basat en interrupcions i cues.

Block_serial: Permet enviar i rebre blocs de bytes pel port sèrie. Utilitza el mòdul Serial.

Timer: Aquest mòdul permet realitzar accions temporitzades gràcies a la instauració de funcions de *callback* temporitzades.

Error_Morse: Permet la detecció d'errors associats a missatges formats per blocs de bytes.

Lan: Implementació d'un protocol d'accés al medi infraroig.

3 Protocol fiable

Per simplificar la pràctica, s'implementarà el protocol de parada i espera. D'aquesta manera, no es podrà transmetre el nou missatge fins que no s'hagi reconegut l'anterior. La implementació del protocol es basarà en els autòmats vists a les classes de teoria.

Una consideració a tenir en compte per implementar un protocol fiable és que es fa necessari el concepte d'establiment de connexió. Amb l'establiment de connexió s'aconsegueix que tant el transmissor com el receptor sincronitzin els identificadors de les trames inicials de l'intercanvi de dades. L'establiment de connexió sincronitza entre transmissor i receptor el número de trama que s'enviarà (per part d'un dispositiu) amb el número de trama que s'espera rebre (per part de l'altre dispositiu). Per altra banda, quan la connexió està establerta, els dispositius només "atenen" les trames del dispositiu amb qui hi ha intercanvi de missatges, fent que els missatges rebuts d'altres dispositius siguin ignorats. Per implementar aquest establiment de connexió, el mòdul Frame (també anomenada capa Frame) oferirà a la seva API les funcions frame_connect() i frame_disconnect().

3.1 Callback de final de transmissió

El mòdul Ether disposa a la seva API d'un callback que es pot instal·lar per indicar el final de transmissió quan s'ha cridat a la funció *ether_block_put()*. Aquest callback indica quan queda el canal alliberat, coincidint amb la finalització de la transmissió del propi dispositiu. Aquest callback s'ha traslladat a la capa Lan i queda disponible per a la capa Frame. L'indicador de final de transmissió pot ser molt útil per considerar-lo un esdeveniment que hagi de processar un autòmat que gestioni els estats de la transmissió d'un missatge. En particular, els protocols fiables requereixen d'un temporitzador un cop s'inicia la transmissió, però la durada d'aquest temporitzador es fa molt dispersa ja que per una banda els missatges poden tenir durades molt diferents i per altra, la capa Lan (accés al medi) també pot afegir variabilitat en el moment de transferir el missatge a la capa Ether, augmentant el temps total de transmissió vist des de la capa Frame. Per contra, la durada d'aquest temporitzador és molt concreta a partir del moment del final de la transmissió del missatge. Definir un autòmat per a la transmissió dels missatges que contempli diferents estats (transmissió iniciada però no finalitzada, o bé transmissió iniciada i finalitzada) en funció de l'esdeveniment de final de transmissió, dóna lloc a una implementació millor.

3.2 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. També poden encarregar-se de realitzar altres funcions com l'establiment de la connexió o tancament de la connexió.

3.2.1 Trames de dades

Estan formades pels següents camps:

Tipus És el camp que determina que aquesta trama és una trama de dades. Aquest camp és de la mida de 1 byte. Per tal de poder distingir-la de la resta de trames, el valor d'aquest byte serà "I".

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.

3.2.2 Trames de control de reconeixement

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. Per indicar una NO confirmació es farà servir el número de trama que no correspon.

En aquest cas, els camps són:

Tipus És un camp de 1 byte. Serveix per identificar el tipus de trama que tenim. Quedarà indicat amb el valor "R".

Numeració de trama Té mida de 1 caràcter (byte). 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 de reconeixement amb identificador "1").

Com a conclusió, aquestes trames tindran mida fixe de 2 caràcters sempre.

3.2.3 Definició de la unitat de dades de protocol en C

Per tal de treballar amb aquest tipus de trames, es definirà el següent tipus de dades:

```
typedef struct {
    morse_c_t tipus;
    morse_c_t num;
    morse_c_t payload[MAX_MSS_FRAME];
} frame_i_pdu_t;
```

per a les trames que contenen dades i

```
typedef struct {
    morse_c_t tipus;
    morse_c_t num;
    morse_c_t payload[1];
} frame_c_pdu_t;
```

per a les trames de control.

Considereu aplicar l'atribut "volatile" allà on creieu necessari.

Justifiqueu el perquè frame_c_pdu_t té un payload de mida 1.

4 Mòdul Frame

4.1 API del mòdul

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 de les anteriors pràctiques. 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 <stdint.h>
#include <stdbool.h>
#include "lan.h"

#define MAX_MSS_FRAME (MAX_MSS_LAN - 2)
typedef morse_c_t message_frame_t[MAX_MSS_FRAME];

void frame_init(morse_c_t no);

void frame_connect(morse_c_t nd);
void frame_disconnect(void);

bool frame_can_put(void);
void frame_block_put(const missatge_frame_t m);

typedef void (*fc_t)(void); //Callback
void frame_block_get(message_frame_t m);
void on_frame_received(fc_t f);

#endif
```

Aquest mòdul ofereix funcions equivalents a les que ofereixen els mòduls Ether o Lan. Però en aquest cas ens assegurem que el servei que ofereix és fiable.

La funció *frame_init()* inicialitza el mòdul i per tant el protocol. Cal destacar que amb aquesta inicialització queda fixada la identificació del propi node.

La funció *frame_connect()* estableix la connexió amb un altre dispositiu de la xarxa. Quan s'estableix, sincronitza la numeració de les trames tant en transmissió com en recepció. Per tant, deixarà els autòmats en l'estat inicial.

La funció *frame_disconnect()* allibera la connexió i per tant deshabilita els autòmats tant de transmissió com de recepció.

La funció *frame_can_put()* determina si es possible enviar un nou missatge. Aquesta funció donarà resultat "false" quan hi hagi algun missatge transmès pendent de ser reconegut.

La funció *frame_block_put()* permet enviar un missatge, sempre que la funció frame_can_put() sigui "true". En aquest cas, un cop establerta la connexió, s'ha de tenir en compte que la comunicació ha sigut inicialitzada per mantenir una conversa entre 2 nodes. Per tant, no és necessari indicar qui és el destinatari del missatge.

La funció *frame_block_get()* és la que s'encarrega de fer la recepció del missatge rebut,

deixant el seu contingut a la variable del tipus *message_frame_t* que es passa com a paràmetre. Actua de la mateixa manera que ho fan les funcions equivalents als mòduls Ether i Lan.

El lliurament de les dades rebudes es fa via Callback gràcies a la funció *on_frame_received()*.

Tant la part transmissora com la part receptora utilitzaran cadascuna com a buffer una única trama. En un protocol de parada / espera no es pot transmetre cap altre trama fins que aquesta no sigui reconeguda totalment. Quan es rep una trama, aquesta es lliurada directament a la capa superior, alliberant així el buffer de recepció per poder rebre la següent trama.

5 Capa d'aplicació

L'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, per tant, el nom que tindrà aquesta aplicació serà **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*.

Cada bloc de dades rebut (cada missatge) es presentarà per pantalla del terminal com una línia nova. De la mateixa manera, cada missatge transmès també es presentarà per pantalla com una línia nova.

5.1 Requeriments de la capa d'aplicació

5.1.1 Part transmissora morse - recepció canal sèrie

La part de l'aplicació transmissora morse consisteix en rebre pel port sèrie el que escriu l'usuari amb el teclat fins trobar el caràcter sentinella (del port sèrie) [enter] (polsat per l'usuari). En aquest moment l'aplicació analitzarà el missatge i cridarà a les funcions de la capa Frame que cregui oportunes. Per sol·licitar la transmissió d'un missatge pel canal morse es fan servir les funcions (*frame_can_put()* i *frame_block_put()*). Si la funció *frame_can_put()* retorna un "False", l'aplicació quedarà bloquejada en aquest punt fins que es retorni un "True".

Els missatges que es poden rebre pel port sèrie tenen el següent format:

[Missatge][enter] Correspon a la sol·licitud transmissió d'un missatge cap a l'altre dispositiu. Després de la transmissió d'aquest missatge pel canal morse es farà un eco pel port sèrie del mateix missatge amb el format "local: [missatge]". Aquest missatge d'eco ha de formar una nova línia a pantalla, per tant s'ha d'enviar també els caràcters de formatació (retorn de carro i/o avançament de línia) per a que quedi correctament visible a pantalla. Si es desitja visualitzar per pantalla cada pulsació, l'aplicació picocom té un flag per poder activar un eco local.

>[ND][enter] Correspon a la sol·licitud d'establiment de connexió cap el dispositiu amb identificador ND. El caràcter inicial especial '>' determina que es vol establir una nova connexió. La recepció d'aquest missatge determina la crida a la funció *frame_connect()*.

/[enter] Correspon a la sol·licitud d'alliberament de connexió. En aquesta ocasió es farà la crida a la funció *frame_disconnect()*.

5.1.2 Part receptora morse - transmissió canal sèrie

Aquesta part és molt més simple que l'anterior. Consisteix en definir un callback que s'executarà en el moment que hagi un missatge disponible per part de la capa Frame. La feina que ha de fer aquest callback és rebre el missatge pròpiament dit a través de la funció *frame_block_get()* deixant el contingut del missatge en una variable de tipus *message_frame_t*. Posteriorment, enviarà aquest missatge a través del port sèrie per a que pugui ser visualitzat per la pantalla del PC. El format d'aquest missatge que s'envia pel port sèrie serà "remot: [missatge]". Aquest missatge també ha de formar una nova línia a pantalla, per tant s'ha d'enviar juntament amb el missatge els caràcters de formatació (retorn de carro i/o avançament de línia) per a que quedi correctament visible a pantalla.

6 Implementació del mòdul Frame

Per tal de fer la implementació del protocol de manera progressiva, el punt de partida serà un mòdul Frame que faci de curtcircuit (o pont) entre l'aplicació i la capa Lan. Anomenarem a aquest mòdul *frame_fake*. La principal utilitat d'aquest mòdul és poder testejar l'aplicació final sense disposar encara de la implementació final del mòdul Frame.

En aquest mòdul *frame_fake* no s'implementarà cap mecanisme de gestió de la connexió. Per tant, la connexió entre 2 dispositius es farà quan l'usuari corresponent de cada dispositiu introdueixi la comanda d'establiment de connexió ">[ND][enter]". Tant l'usuari que pren la iniciativa d'establir la connexió amb un altre dispositiu com l'usuari amb qui es vol establir la connexió han d'introduir la comanda per tenir la connexió establerta.

El següent pas és definir 2 mòduls Frame a partir de *frame_fake*. En un d'ells s'implementarà el protocol corresponent a l'enviament de missatges amb la màquina d'estats (autòmat) transmissora vista a classe. Anomenarem a aquest mòdul *frame_tx*. En l'altre s'implementarà el protocol corresponent a la recepció de missatges, igualment amb la màquina d'estats receptora vista a classe. En aquest cas s'anomenarà *frame_rx*. En un dispositiu Arduino injectarem l'aplicació amb el mòdul frame_tx i en l'altre dispositiu injectarem l'aplicació amb el mòdul frame_rx. La conseqüència d'això és que els dispositius tindran una comunicació fiable únicament en un sentit, però en l'altre sentit no. En el sentit de comunicació fiable, el node amb *frame_tx* enviarà trames de dades cap

a l'altre node i rebrà trames de reconeixement. Per contra, el node amb *frame_rx* rebrà trames de dades i transmetrà trames de reconeixement.

Un cop tenim aquests 2 mòduls Frame perfectament operatius, el pas següent és fusionar els 2 autòmats en un únic mòdul Frame anomenat *frame*. Això seria extraordinàriament simple si els dos autòmats fossin totalment independents, ja que implicaria cridar un o l'altre en funció de l'esdeveniment que passi. El problema apareix quan hi ha esdeveniments que poden estar associats als dos autòmats, com seria la rebuda d'una trama. Quan passa això, es poden establir dues estratègies:

1. Processar el missatge rebut per part dels 2 autòmats. Cada autòmat descartarà o no el missatge en funció del seu contingut.
2. Pre-processar el missatge rebut per distingir si és una trama de dades o de reconeixement. Un cop feta la distinció, s'ha de llençar l'esdeveniment "rebuda trama dades" per a que el processi l'autòmat de recepció o bé l'esdeveniment "rebuda trama reconeixement" per a que el processi l'autòmat de transmissió.

6.1 Autòmat de transmissió

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 serà il·limitat per facilitar així la implementació de l'autòmat. En aquest cas, qui s'encarregarà de determinar que la comunicació és inviable serà el mateix usuari. Per altra banda, les úniques trames que rebrà aquest autòmat seran trames de control per reconèixer les trames de dades prèviament enviades.

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

6.2 Autòmat de recepció

Les funcions d'aquest autòmat són lliurar el missatge correctament rebut a la capa d'aplicació i enviar el corresponent reconeixement a l'altre node. Aquest reconeixement s'envia amb trames de control de tipus reconeixement.

7 Treball pràctic

Per a cada sessió crea un "tag" de subversion per tal de "congelar" la versió del projecte. La documentació oficial de subversion la trobareu a [1] i en particular la creació de "tags" a [2]

7.1 Sessió 1

1. Dibuixa el graf corresponent al transmissor adaptant el graf de les transparències de classe amb els noms de les funcions i esdeveniments corresponents a aquesta pràctica.
2. Dibuixa el graf corresponent al receptor adaptant el graf de les transparències de classe amb els noms de les funcions i esdeveniments corresponents a aquesta pràctica.
3. Implementa l'aplicació final considerant que el mòdul Frame estigui disponible, anomena-la *xat*.
4. Implementa el mòdul *frame_fake* que faci de pont (curtcircuit) entre l'aplicació i la capa Lan. Testeja el correcte funcionament de l'aplicació *xat*.

7.2 Sessió 2

1. Implementa el mòdul *frame_ tx*.
2. Implementa el mòdul *frame_ rx*.
3. Testeja el correcte funcionament combinat de *frame_ tx* i *frame_ rx* amb l'aplicació de *xat* i/o el que consideris oportú.
4. Implementa el mòdul *frame* i comprova el seu correcte funcionament.

7.3 Sessió 3

1. Implementa la gestió de la connexió i incorpora-la a la capa Frame.

Referències

[1] `https://svnbook.red-bean.com/`

[2] `https://svnbook.red-bean.com/en/1.2/svn.branchmerge.tags.html`