Enginyeria de Sistemes TIC

Xarxes de Comunicació

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

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 ampliar el mòdul *lan* fet a la pràctica anterior per aconseguir millors prestacions i funcionalitats. Les funcionalitats requerides són les següents:

- L'API millora la funció *lan_block_put()* acceptant que es pugui cridar sempre, sense la comprovació prèvia de *lan_can_put()*, però a canvi de convertir-la en una funció bloquejant fins que es doni la condició que es pugui transferir el missatge.
- Opcional: Eliminar de la API la funció *lan_can_put()* de manera que es pugui cridar *lan_block_put()* sense problemes quan es vulgui, però que sempre sigui una funció no bloquejant. Per aconseguir aquesta funcionalitat, la manera de fer-ho és implementant una cua de missatges.
- Afegir a la API *lan* el callback de final de transmissió del missatge, tal i com té implementada la API del mòdul ether.
- Donar la possibilitat de transferir missatges amb diferents prioritats. Això significa que la crida a *lan_block_put()* admetrà com a paràmetre el grau de prioritat del missatge. Per tant, entre tots els possibles missatges que l'autòmat de transmissió de la *lan* tingui pendents d'enviar, agafarà el de prioritat més alta primer.

2 Implementació

2.1 lan_block_put() bloquejant

Aquesta funcionalitat es pot considerar un mecanisme de seguretat del mòdul *lan*, ja que evita un comportament erràtic del sistema en el cas que la funció *lan_block_put()* sigui cridada quan *lan_can_put()* no ho permetria. D'altra banda, permet alhora altres dissenys on el comportament bloquejant de la funció sigui l'esperat.

S'ha de tenir en compte que sota certes circumstàncies aquest bloqueig pot provocar un bloqueig de tot el sistema. Un exemple seria fer la crida a la funció dins d'una interrupció. Si la condició de desbloqueig depèn d'una altra interrupció, aquesta altra interrupció no es donarà mai ja que les interrupcions a l'AVR estan desactivades mentre s'executa una interrupció.

2.2 Cua de missatges

En aquest cas, s'ha de fer una implementació anàloga al modul cua vist en assignatures anteriors. La principal diferència és que els elements de la cua són de tipus *message_lan_t*. Cal tenir en compte a la implementació que la condició per diferenciar entre cua plena i la cua buida no es pot fer deixant un element buit. Surt molt més a compte definir una variable booleana que distingeixi entre cua plena i buida quan els 2 índexs coincideixen.

2.3 Callback de final de transmissió

La implementació d'aquest callback al mòdul lan és tan simple com traslladar el callback del mòdul *ether* oferint-lo a la API de la *lan*, sense que el mòdul lan el processi. La manera de presentar-lo a la API del *lan* podria ser tal com ho fa el *ether*, afegint la funció *on_finish_transmission()*. Per reforçar la idea que aquest callback està associat al missatge que es vol transmetre, la instal·lació es farà afegint-lo com a paràmetre de la funció *lan_block_put()*. Per tant el nou prototipus de la funció *lan_block_put()* serà el següent:

```
//Callback when finish transmission
typedef void (*lc_fintx_t)(void);

void lan_block_put (const message_lan_t m, morse_c_t nd, lc_fintx_t cf);
```

2.4 Missatges amb prioritat

La implementació prèvia de la *lan* admet 1 missatge en espera de ser transmès. La implementació amb una cua admet més de 1 missatge però no es pot gestionar la prioritat. Per

poder gestionar prioritats, la *lan* ha d'admetre com a mínim 1 missatge per a cadascuna de les prioritats que estiguin definides. Això implica que s'ha de disposar d'un buffer on estiguin aquests missatges. Per altra banda, existeixen implementacions de cues amb prioritats, però per la seva complexitat i natura marxen dels objectius de l'assignatura, encara que són la solució més adequada per aquest problema. En conclusió, es proposa una solució de mínims, senzilla i simple que permeti complir amb l'objectiu plantejat.

La solució de mínims consisteix en tenir missatges amb 2 prioritats (High i Low) i un buffer on guardar un missatge per a cada prioritat. Un cop definida aquesta estructura de dades, cal definir les operacions sobre aquesta estructura de dades que permeti incorporar i treure els missatges d'aquest buffer. D'aquesta manera, la crida a la nova funció *lan_block_put()* incorporarà nous missatges amb prioritat al buffer, mentre l'autòmat de transmissió de la lan, *tx_automat_lan()*, els traurà tenint en compte la prioritat per transferir-los al *ether*.

Les operacions d'incorporar i treure són equivalents a les operacions d'encuar i desencuar en una cua. Per tant, se seguirà un model equivalent al d'una cua. Per fer-ho més estructurat encara, aquest buffer es definirà en un nou mòdul que anomenarem *lan_buffer*. Degut a que la possibilitat de necessitar més d'un buffer d'aquest tipus és remota, el mòdul s'implementarà com un objecte tipus singleton.

Per facilitar-vos la feina, a continuació disposeu de porcions de codi que us poden ser d'utilitat:

```
typedef struct {
  volatile bool free;
  lc_fintx_t cf;
  lanpdu_t pdu;
} pdu_unit_t;

typedef struct {
  pdu_unit_t high;
  pdu_unit_t low;
} buff_lan_t;

typedef enum {low, high} priority_t;

...

void buff_empty(void);
bool buff_is_empty(void);
bool buff_is_full(priority_t p);
void buff_put(const lanpdu_t *const pdu, lc_fintx_t cf, priority_t p);
lc_fintx_t buff_get(lanpdu_t *const pdu);
```

buff_empty() buida el buffer. En el moment d'implementar-la s'ha de ser eficient amb el concepte buidar.

buff_is_empty() Retorna true si no hi ha res al buffer. Això implica que totes les pdu siguin de la prioritat que siguin s'han buidat.

buff_is_full() Retorna true si no hi ha disponibilitat d'incorporar una pdu amb la prioritat indicada segons el paràmetre.

buff_put() Incorpora al buffer la pdu que se li passa per referència, així com el seu callback de final de transmissió associat. Prèviament s'ha de comprovar la disponibilitat d'espai per a aquesta pdu. Si no hi ha espai, el comportament serà arbitrari. La implementació d'aquesta funció implica la còpia de la pdu que es passa per referència cap al buffer, ocupant l'espai que li correspon.

buff_get() Recupera del buffer la pdu que tingui prioritat més alta i la col·loca a dins de la pdu que es passa per referència. També retorna la funció de callback associada de final de transmissió. L'espai ocupat pel missatge recuperat quedarà disponible. Prèviament s'ha de comprovar que el buffer no estigui buit, sinó el comportament serà arbitrari.

Observeu que els tipus *lanpdu_t* i *lc_fintx_t* han de ser coneguts tant pel mòdul **lan** com pel mòdul **lan_buffer**. Resoleu adequadament aquesta necessitat.

3 Nova API del mòdul lan

Després d'incorporar aquestes funcionalitats, la nova API del mòdul *lan* quedarà de la següent manera:

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

#define MAX_MSS_LAN (MAX_ME_ETH - 4)

typedef morse_c_t message_lan_t[MAX_MSS_LAN];

void lan_init(morse_c_t no); //no is origin node ID

//Callback when finish transmission
typedef void (*lc_fintx_t)(void);
typedef enum {low, high} priority_t;
bool lan_can_put(priority_t p);
void lan_block_put(const message_lan_t m, morse_c_t nd, priority_t p, \
        lc_fintx_t cf);

//Callback when a message is received
typedef void (*lc_messrx_t)(void);
morse_c_t lan_block_get (message_lan_t m);
void on_lan_received (lc_messrx_t cm);

#endif
```

4 Treball pràctic

Saber quines tasques i en quin ordre s'han de fer determina molt l'optimització del temps dedicat a la implementació d'aquestes funcionalitats extres. El punt de partida són tots els mòduls de la pràctica anterior. A mode de guia, teniu a continuació quines poden ser aquestes tasques i l'ordre a seguir

1. Marqueu la versió actual de la pràctica al sistema de control de versions amb una etiqueta que us serveixi per recuperar la versió que teniu fins ara. La manera de fer-ho més adequadament és creant el que s'anomena un tag de subversion (consulteu la documentació).
2. Modifiqueu el mòdul *lan* per incorporar la funcionalitat *lan_ block_ put()* bloquejant
3. Testegeu i comproveu el correcte funcionament del mòdul lan amb un programa de test que forci el bloqueig de la funció *lan_ block_ put()*. Anomeneu al programa de test *t_ lan_ bq*. Un cop tot estigui funcionant correctament, marqueu amb un nou tag la versió actual al sistema de control de versions.
4. *Opcional:* Incorporeu la cua de missatges al mòdul *lan*. Això implica:
   a) La creació d'un nou mòdul anomenat *msg_ queue*.
   b) Un programa de test anomenat *t_ msg_ q* per testejar aquesta cua.
   c) La modificació del mòdul *lan* incorporant la cua.
   d) Un programa de test anomenat *t_ lan_ q* per testejar el mòdul *lan* amb la nova cua.
5. Modifiqueu el mòdul *lan* per incorporar la funcionalitat del callback de final de transmissió.
6. Testegeu i comproveu el correcte funcionament del callback de final de transmissió amb un programa de test anomenat *t_ lan_ cf*.
7. Quan tot funcioni correctament, marqueu la nova versió amb un tag al svn.
8. Dissenyeu el mòdul *lan_ buffer*. Estructureu adequadament les definicions de tipus que necessiten tant *lan_ buffer* com *lan*.
9. Testegeu i comproveu el correcte funcionament del mòdul lan_buffer amb un programa de test anomenat *t_ lan_ buff*. En aquest cas, no cal incorporar en el test el mòdul lan.
10. Modifiqueu el mòdul *lan* incorporant el mòdul *lan_ buffer* per dotar de prioritat a la funció *lan_ block_ put()*.
11. Testegeu i comproveu el correcte funcionament del mòdul *lan* amb un programa de test anomenat *t_ lan*. Aquest programa de test pot ser una modificació del programa principal de la pràctica anterior. Quan tot funcioni marqueu la versió amb un nou tag indicant el final d'aquesta nova pràctica.

12. *Opcional:* Modifiqueu el mòdul lan_buffer per incorporar una cua per a cada prioritat enlloc de 1 sol missatge. Un cop fet això s'obté una implementació simple però subòptima d'una cua de missatges amb 2 prioritats. Com sempre, s'han de realitzar els testos adequats.

Referències

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