|  |  |
| --- | --- |
|  | **2014** |
|  | Grupa 433ARădulescu Vicențiu Mihail   |

|  |
| --- |
| **Colectia de drivere dispozitiv la linux, Drivere USB la Linux** |
|  |

#  Cuprins:

**Colectia de drivere dispozitiv la linux**

**A.Protocolul USB**

**B.Drivere USB la Linux:**

**1) Noțiuni introductive**

**2)Bazele USB-ului**

**3)Comunicațile USB -ului**

**4)Scrierea driverelor USB**

**5)Controlul erorilor**

**Bibliografie**

**Concluzii**

**Protocolul USB**

Protocolul USB este bazat pe pachete de date,un pachet fiind format din cadre de date.Aceste cadre sunt de 4 biți până la 8 biți de date.

Trasnmiterea serială către magistrală are loc după bitul cel mai puțin semnificativ(little endian) din cadru,deci ultimul bit transmis din cadru va fi cel mai semnificativ.

Exista patru tipuri de pachete:pachetul de semnalizare,de date ,cel special și cel de dialog.

Pachetul de semnalizare:

-toate transferurile încep prin trimiterea de către gazdă a unui pachet de semnalizare.

-pachet format din 32 biți înparțit astfel:



-câmpurile SYNC și PID ,fiecare cu câte un octet vor fi descrise mai târziu.

-aici în câmpul PID se defines 4 categorii de pachete:

\*pachetul de ieșire OUT poartă datele de la gazdă la dispozitiv.

\*pachetul de intrare IN poartă datele de la dispozitiv la gazdă

\*pachetul de comandă SETUP vizează un anumit nod.Nodul reprezintă o secțiune a unei funcții USB adresabilă individual.

\*pachetul de început de cadru este difuzat tuturor dispozitivelor.

-este emis de gazdă la începutul tranziției de semnalizare.

-dispozitivele decodifică și redirecționează acest pachet, dar nu trimit semnalul de recepție a pachetului.

-structura acestui pachet:



-câmpul Număr cadru conține numărul cadrului care este atribuit de gazdă în mod crescător de la 0 la 7FFH,după care începe din nou de la 0.

Pachetul de date:

-conține informația propiu-zisă transferată în sistemele USB.

-structura pachetului de date:



-corectudinea câmpului de date este asigurată prin câmpul de verificare ciclică de 16b aflat la sfârșitul pachetului.

-în PID avem 2 categorii de câmpuri de date DATA0 și DATA1.Cele două categorii formează între emițător și receptor un sistem adițional de verificare a erorilor.Emițătorul oscilează între DATA0 și DATA1 pentru a indica că a recepționat o confirmare validă a recepției pachetului precedent.

Pachetul de dialog:

-pachetul se mai numește pachetul handshake

-sunt folosite pentru a raporta starea unui transfer de date,pentru a indica recepția cu succes a datelor sau pentru a întoarece valori care indică acceptarea/respingerea unei comenzi sau o stare de HALT la dispozitiv.

-compus din două câmpuri SYNC și PID:



-câmpul PID:



\*handshake ACK indică emițatorul că pachetul de date a fost recepționat fără erori.

\* handshake NAK indică faptul că o funcție nu a fost capabilă să recepționeze date de la gazdă sau că o funcție nu are date de transmis gazdei.

\* handshake STALL este emis de o funție ca răspuns la un pachet de semnalizare IN sau după o tranziție de date OUT,indicând că funcția nu este capabilă să emită sau să recepționeze date.

-gazda nu poate răspunde cu pachet NAK sau STALL.

Câmpul de sincronizare SYNC:

-apar la începutul oricărui pachet,fiind primul câmp.

-este constituit dintr-o serie de biți care produc un șir dens de tranziții utilizând schema de codificare NRZI cerută de standardul USB,ele(tranzițiile)permit dispozitivelor conectate să-si reseteze ceasul și să se sincronizeze cu gazda.

--câmpul este o serie de 3 tranziții plus o marcă cu lătimea de 2 impulsuri,datele câmpului au succesiunea de valori 00000001.



Succesiunea datelor și forma semnalului din câmpul SYNC.

Câmpul identificator de pachet PID

-apar la începutul oricărui pachet,fiind al doilea câmp cu o lungime de 8 biți.

-primi 4 biți indica tipul pachetului.Cei mai semnificativi doi biți specifică tipul pachetului,cei mai puțin semnificativ împart pachetele în categorii.



-ultimi 4 biți sunt de verificare pentru a afirma acuratețea primilor 4.

**Noțiuni**

**introductive**

 USB-ul (universal serial bus) este legătura între computer-ul principal și un

număr de device-uri periferice.A fost creat original pentru a înlocui o gamă largă încetă ,diferită de bus-uri(paralelă,serial și conecțiile tastaturii)cu un singur tip de bus unde toate device-urile se pot conecta.

 USB-ul a crescut mult peste aceste conectări încete și acum suportă orice tip de device care poate fi conectat la PC.

 Topologic,un subsistem USB nu este precum un bus ,ci mai degrabă un arbore construit din mai multe legături punct la punct.Legăturile sunt 4 fire cablate(ground,power si 2 fire de semnal) care conectează un device ca un hub,precum Ethernet.Controlerul USB de host este ocupat să întrebe fiecare USB device dacă are o data de transmis.Din cauza topologiei,un driver USB nu poate să trimită date fară ca înainte să fie întrebat de controlerul de host. Această configuratie oferă un sistem plug and play usor de implementat,unde device-urile pot fi automat configurate de computerul host.

 Bus-ul are câteva caracteristici interesante,precum ambilitatea ca un device să ceară o bandă fixă pentru transferurile de date,în scopul de o oferi suport video si audio a I/O.Altă caracteristică a USB-ului este că acționează doar ca un canal de comunicare între dispozitiv și gazdă(host),fară a fi nevoie ca datele sa aibă un anumit sens sau structură .

 Specificațile protocolului USB definesc un set standard care orice dispozitiv a unui specific tip poate urmări.Dacă un dispozitiv folosește acel standard,atunci un driver special pentru acest dispozitiv nu este necesar.Aceste tipuri diferite sunt numite clase si consist în lucruri precum dispozitivele de stocare, mous, tastatură,dispositive internet si modem-uri.Alte tipuri de dispositive care nu intră în aceste clase ,avem nevoie de un anumit driver special care să fie scris pe acel device.Dispozitivele video si dispozitivele USB seriale sunt un bun exemplu unde nu exista un standard anume si avem nevoie de un driver pentru fiecare dispozitiv diferit de la fabricanți diferiți.

 Kernel-ul linux suportă 2 tipuri principale de dispositive USB:driver pe sistemul gazdă si drivere de pe un dispozitiv.

 Driver-ele USB pentru un sistem gazdă(host)controlează dispozitivele USB care sunt introduse înăuntru,din punct de vedere a gazdei(o gazdă comuna USB este un computer desktop).

 Driver-ele USB pentru un dispozitiv,controlează cât acel singur dispozitiv se uită la computer-ul gazdă ca un device USB.Aceste drivere sunt mai numite “USB gadget drivers”.

Privire de ansamblu asupra suportului USB



**Bazele USB-ului**

Viteza de transfer la USB:

Viteză mică:pană la 1.5Mps,de la USB 1.0

Viteză medie:pană la 12Mps,de la USB 1.1

Viteză mare:pană la 480Mbps,de la USB 2.0

Descriptori USB:

Dispozitiv-reprezintă dispozitivele conectate la bus-ul USB

 EX:speaker-ul USB cu butoanele de volum

Configurația-reprezintă starea dispozitivului

 EX:Activ,Standby,Initializare

Interfața-dispozitive logice

 EX:speaker,butoanele de control al volumului

Endpoints-pipe-uri unidireconale de comunicare

 Pot fi IN(dispozitiv la computer) sau OUT(computer to dispozitiv)



Endpoint-urile:

Un USB endpoint poate purta date într-o singură directivă,ori de la host computer către dispozitiv sau de la dispozitiv către computerul host.Există patru tipuri de endpoint-uri care descriu transmiterea datelor:

CONTROL:

-folosit pentru a configura dispozitivul,pentru a lua informația despre ea,pentru a trimite comandă către ea,pentru a retrage informați asupra stări.

-se transmit date simple și mici.

-fiecare dispozitiv are un control de endpoint(endpoint 0),folosit pentru a configura dispozitivul.

-protocolul USB garantează că o să avem destulă bandă.

INTERRUPT:

-transferă mici cantități de date la o rată fixă de fiecare dată când host-ul cere dispozitivului data.

-USB protocol garantează o bandă rezervată pentru transferuri.

-principala metodă de transport pentru mous si tastatură USB.

BULK:

-folosită pentru a transfera cantități mari

-USB protocol nu garantează o bandă rezervată pentru transferuri dar garantează că nu voi fi pierdute date

-folosit in general la printere,dispositive de stocare și de internet

ISOCHRONOUS:

-transferă tot cantitați mari de date,dar datele nu sunt garantate că vor fi transmise

-folosite de dispozitve care pot să se ocupe cu pierderi de date precum dispositive video sau audio

Structura unui endpoint (domenile folositoare pentru dispozitivele de scriere)

a) bEndpointAddress: adresa USB a endpoint-ului

-include direcția endpoint-ului

-poți folosi bitmask-ul USB\_ENDPOINT\_DIR\_MASK pentru a spune dacă este USB\_DIR\_IN  sau USB\_DIR\_OUT  endpoint.

Exemplu:if ((endpoint­>desc.bEndpointAddress &

USB\_ENDPOINT\_DIR\_MASK) == USB\_DIR\_IN)

b) bmAttributes:prezintă tipul endpoint-ului

-poți folosi bitmaskul  USB\_ENDPOINT\_XFERTYPE\_MASK USB \_ENDPOINT\_XFER\_ ISOC, pentru a spune ce fel de tip este USB\_ENDPOINT\_XFER\_BULK ,  USB\_ENDPOINT\_XFER\_INT sau USB\_ENDPOINT\_XFER\_CONTROL.c) wMaxPacketSize:numărul maxim de octeți care un endpoint poate să aibă.Dacă se folosesc dimensiuni mai mari,data o sa fie înparțita în bucați de n wMaxPacketSize

d) bInterval:dacă acest endpoint este de tipul întrerupere,atunci aceasta valoare este intervalul pentru acest endpoint.

-timpul între cererile de întrerupere pentru acel endpoint si se măsoară în milisecunde

Interfața :

-fiecare interfață incapsulează o funcție de mare nivel.Ex:flux video,flux audio,butoanele de control la tastatură

-avem nevoie de un driver penru fiecare interfață

-capabilitatea de a avea setări alternative:fiecare interfață USB poate avea setări diferite de parametru.Ex :bandă diferită pentru interfața audio

-setarea alternativă sunt deobicei folosite în controlul periodic al endpoint-urilor,precum în avea diferite endpoint-uri care ocupă diferite parți de bandă USB.

Structura unui interfețe:

-această structură este ce nucleul USB îi lasă driver-elor USB ,apoi driver-ul USB este responsabil de control

-interfața USB este descrisă in Kernel cu structura usb\_interface structure

Importante domenii in structură:

a) struct usb\_host\_interface \*altsetting- listă de setinguri alternative care poate fi selectată pentru aceea interfața

b) unsigned num\_altsetting-numărul de setinguri alternative

c) int minor-cel mai mic număr de care interfața este legată

Configurația

Interfețele sunt introduse in configurați.

Configurațile reprezintă starea unui device .Ex: Activ, Standby, Initializare.

Ele sunt descrise de structura usb\_host\_config,dar driverele nu au nevoie să acceseze această structură.

Dispozitivele:

Ele sunt reprezentate de structura  usb\_device .Multe drivere folosesc funcția  interface\_to\_usbdev() pentru a accesa structura usb\_dispozitiv de la structura usb\_interface care este dată de nucleul USB.

**Comunicațiile USB-ului**

Cererile bloc:

Orice comunicație intre host și dispozitiv este facut asincron prin cererile bloc ale USB-ului numite in engleza USB Request Blocks (urbs).

Fiecare endpoint poate să se ocupe cu un rând de urbs și fiecare urb are handler de finalizare.Un driver poate aloca mai multe urb-uri pentru un singur endpoint,sau să reutilizeze același urb pentru diferite endpoint-uri.

Sunt asemănătoare cu pachetele în rețeaua de comunicații.

Durata unui ciclu de viața a unui urb



Structura unui urb:

a) struct usb\_device \*dev- dispozitivul cărui i se transmite urb-ul

b) unsigned int pipe- informația despre endpoint-ul din dispozitivul target

c)int status-transferă starea

d) unsigned int transfer\_flags-instrucțiunile pentru tratarea usb-ului

e) void \* transfer\_buffer-pointerul alocat buffer-ului pentru a fi folosit să trimită date catre dispozitiv sau pentru a primi date de la dispozitiv și trebuie creat cu kmalloc().

f) dma\_addr\_t transfer\_dma –buffer detransfer de date cand DMA este folosit

g) int transfer\_buffer\_length-transferă lunigmea buffer-ului

h) int actual\_length-lungimea adevarată a datei transmise sau trimise de un urb

i) usb\_complete\_t complete-handler de finalizare cand un transfer este complet

j) void \*context-date care pot fi flosite in finalizarea unui handler

k) unsigned char \*setup\_packet –pachetul de setup transferat înainte ca datele din buffer-ul de transmisie

l) dma\_addr\_t setup\_dma-la fel ca la unsigned char \*setup\_packet dar când pachetul de setup este transferat cu DMA

m) int interval-intervalul de alegere a Urb-ului

n) int error\_count- numărul de transferuri sincrone care raportează o eroare

o) int start\_frame – setează sau returnează numărului ințial de frame

p) struct usb\_iso\_packet\_descriptor –permit unui singur urb să

iso\_frame\_desc[0] / / definească mai multe transferuri asincrone

q) int number\_of\_packets-numărul de buffere asincrone de transfer folosite

Crearea de pipe-uri,funcțiile inițializării unui domeniu de pipe al unei structuri urb

1)pipe-uri de control- usb\_sndctrlpipe(), usb\_rcvctrlpipe()2)pipe-uri bulk- usb\_sndbulkpipe(), usb\_rcvbulkpipe()3) pipe-uri de întrerupere- usb\_sndintpipe(), usb\_rcvintpipe()4) pipe-uri asincrone- usb\_sndisocpipe(), usb\_rcvisocpipe()

Crearea și eliberarea urb-urilor

Structura struct urb nu trebuie creat niciodată static într-un driver sau într-o altă structură,din cauza faptului că ar distruge schema referință de numărare folosit de nucleul USB pentru urb-uri.El trebui creat cu un call la funcția usb\_alloc\_urb.Această funcție are prototipul: struct urb \*usb\_alloc\_urb(int iso\_packets, int mem\_flags).

Pentru a elibera un urb folosim această funcție: void usb\_free\_urb(struct urb \*urb)

Inițializarea întreruperilor urb

Funcția usb\_fill\_int\_urb este o funcție de tip help pentru a inițializa cum trebuie un urb să fie trimis la o întrerupere endpoint a unui dispozitiv USB.

void usb\_fill\_int\_urb (

struct urb \*urb, // urb-ul care trebuie inițializat

struct usb\_device \*dev, // driver-ul la care trebuie trimis urb-ul

unsigned int pipe, // pipe-ul

void \*transfer\_buffer, // bufferul de transmisie

int buffer\_length, // transfera dimensiunea bufferului

usb\_complete\_t complete, //  handler-ul de completare

void \*context, // context (pentru handler)

int interval // programul interval ului

);

Alocarea bufferelor DMA

Poți folosi funcția usb\_buffer\_alloc()  pentru a aloca un buffer consistent DMA

void \*usb\_buffer\_alloc (

struct usb\_device \*dev, // device-ul

size\_t size,  // lungimea bufferului

gfp\_t mem\_flags, // kmalloc() flags

dma\_addr\_t \*dma // (iesirea)adresa DMA

);

Exemplu:

buf = usb\_buffer\_alloc(dev­>udev,

  count, GFP\_KERNEL, &urb­>transfer\_dma);

Pentru a utiliza aceste buffere,se folosește URB\_NO\_TRANSFER\_DMA \_MAP sau URB\_NO\_SETUP\_DMA\_ MAP  setări pentru urb ­> transfer\_ flags  pentru a indica că  urb­>transfer\_dma  sau  urb­> setup\_dma este valid la trimitere.

Pentru a elibera aceste buffere:

void usb\_buffer\_free (

struct usb\_device \*dev,  // device-ul

size\_t size,  // lungimea bufferului

void \*addr,   // adresa CPU al bufferului

dma\_addr\_t dma  // adresa DMA al bufferului

);

După crearea și inițializarea unui urb se realizează trimiterea urb-ului astfel: int usb\_submit\_urb(

struct urb \*urb, // urb-ul de trimis

int mem\_flags); // kmalloc() flaguri

mem\_flags-este folosit pentru alocarea internă realizată de usb\_submit\_urb().

Urb-urile trimise pot fi anulate folosind usb\_unlink\_urb()(asincron) sau

 usb\_kill\_urb() (sincron).

Handler-ul de finalizare

Handler-ul de finalizare este chemat în context-ul de întrerupere ,în 3 situați.Verifică valoarea erorii în urb­>status

Cele 3 situații:

-după ce transferul de date este completat cu succes

urb­>status == 0

-eroare apăruta în timpul transferului

-urb-ul a fost deconectat de nucleul USB

Implementarea Handler-ului de finalizare :

void (\*usb\_complete\_t)(

struct urb \*, //  completul urb

struct pt\_regs \*   // valori ale registrelor înaintea

 // întreruperi

);

**Scrierea driverelor USB**

 Driverele trebuie anunțate de dispozitivele care îl suportă în structura

usb\_device\_id .La suportul de dispozitiv este nevoie de un spațiu de utilizare pentru a cunoaște care modul să încarce(descarce),iar pentru kernel care cod al driver-ului trebuie executat,când dispozitivul este inserat sau scos.

 Majoritatea driverelor USB\_DEVICE() pentru a crea structuri.Aceste structuri sunt apoi înregistrate cu MODULE\_DEVICE\_TABLE(usb, xxx).

Înegistrarea unui driver USB

Principala structură care toate driverele USB trebuie să creeze este struct usb\_driver.Acestă structură trebuie umplută de driver-ul USB și de un numar mare de funcții și variabile care descriu drive-ul USB si și codul nucleului USB.

a) const char \*name-nume unic la driver folosit în general la nume de modul

b) const struct usb\_device\_id \*id\_table-tabelul deja declarat cu MODULE\_DEVICE\_TABLE().

c) int (\*probe) (struct usb\_interface \*intf,const struct usb\_device\_id \*id)

d) void (\*disconnect) (struct usb\_interface \*intf)-deconectare de apel invers

Structuri opționale:

e) int (\*suspend) (struct usb\_interface \*intf,

 pm\_message\_t message);

int (\*resume) (struct usb\_interface \*intf);-ponteri folosiți pentru a suspenda sau a se întoarce la driverul USB.Sunt apelate când dispozitivul este suspendat/reluat de nucleul USB

f ) void (\*pre\_reset) (struct usb\_interface \*intf);

 void (\*post\_reset) (struct usb\_interface \*intf);

Apelate de usb\_reset\_composite\_device() după și înainte de un reset a portului USB.

Înregistrarea și anularea înregistrări

Se folosește  usb\_register() pentru a înregistra driverul.

Ex: /\* Example from drivers/usb/input/mtouchusb.c \*/

static struct usb\_driver mtouchusb\_driver = {

        .name           = "mtouchusb",

        .probe          = mtouchusb\_probe,

        .disconnect     = mtouchusb\_disconnect,

        .id\_table       = mtouchusb\_devices,

};

static int \_\_init mtouchusb\_init(void)

{

        dbg("%s ­ called", \_\_FUNCTION\_\_);

        return usb\_register(&mtouchusb\_driver);

}

Pentru a anula înregistrarea se utilizează  usb\_deregister()

Ex: /:\* Example from drivers/usb/input/mtouchusb.c \*/

static void \_\_exit mtouchusb\_cleanup(void)

{

 dbg("%s ­ called", \_\_FUNCTION\_\_);

        usb\_deregister(&mtouchusb\_driver);

}

Funcția probe() este apelată de nucleul USB să vadă dacă driverul vrea să ocupe o interfată particulară pe un device.Apoi driverul trebuie să verifice informația despre dispozitiv.Dacă decide să se ocupe de interfață ,funcția probe() o să returneze 0,astfel o să returneze o valoare negativă.

 Funția disconect() este apelată de nucleul USB când un driver n-o să mai controleze dispozitivul și o să faca curățenie.

Transferuri USB fară URB

Kernel-ul ofera două funcții ajutătoare usb\_bulk\_msg() și usb\_control\_ msg() care fac posibile transferurile simple bulk și controlul mesajelor,făra să fie nevoie de a crea sau reutiliza o structură urb,de a inițializa,de a trimite și fară a aștepta terminarea unui handler.

Aceste funcții sunt sincrone și o să facă codul tău să stea.Nu trebui apelate dintr-un context de întrerupere .Nu poți închide cererile,dacă nu poți trata un Urb itern.Verificăte dacă funcția disconnect() poate să aștepte aceste funcți să se termine.

**Controlul erorilor**

Comunicația între calculator(host) și dispoztive(hub)se face pe baza unui protocol descris în tabelul de mai jos.

|  |  |  |  |
| --- | --- | --- | --- |
| Câmp de sincronizare(8biți) | Indentificator al pachetului (8biți) | Date(0,1.....1032octeți | Control erori CRC(16 biți) |

Controlul erorilor este realizat cu ajutorul unui cod reductant ciclic,CRC(Cyclic Redundacy Code),care asigură transmisia corectă a informației.Mecanismul CRC asigură o rată de eroare a transmisiei de un bit eronat la 10 miliarde.După terminarea transferului de date,are loc confirmarea datelor și verificarea corectitudinii transmisiei lor.

**Algoritmul de calcul CRC**

La aplicația USB se folosește polinomul CRC-5-USB :$x^{5}+x^{2}+1$ cu cadrul 100101

La aplicațiile CDMA,Bluetooth,IrDA,MMC,SD utilizăm polinomul CRC-16-CCITT :$x^{16}+x^{12}+x^{5}+1$ cu cadrul 10001000000100001

Pentru rezolvare se va folosi aritmetica polinomială de tipul modulo 2,în care nu există transport la adunare și nici înprumut la scădere.Se va folosi de asemenea un polinom generator G(x).Acest polinom are atât bitul cel mai semnificativ cât și cel mai puțin semnificativ 1.Se dorește sa se adauge un număr de biți la sfârșitul unui cadru,ce are un polinom notat M(x) astfel încât M(x) divide G(x).Fie r gradul lui G(x). Se adaugă r biți de 0 la capătul mai puțin semnificativ al cadrului astfel încât acesta va corespunde polinomului xr M(x).Se împarte șirul de biți ce corespund lui G(x) într-un șir de biți corespunzător lui xr M(x), utilizând împărțirea modulo 2.Se scade restul (care are întotdeauna r sau mai puțini biți) din șirul de biți corepunzător lui xr M(x),

utilizând scăderea modulo 2. Rezultatul este cadrul cu suma de control ce va fi transmis (acesta este divizibil modulo 2 cu G(x)).

**Bibliografie:**

-><http://free-electrons.com/doc/books/ldd3.pdf>

-><http://free-electrons.com/doc/linux-usb.pdf>

->wikipedia

**Concluzii:**

-interfața USB este cea mai folosită interfață în conectarea mouse-ului,imprimantei sau a scanerului

-scrierea driverelor USB la Linux nu este o sarcină prea dificilă precum sar crede,driverul împreuna cu alte drivere USB ,îti dau destule exemple pentru a înțelege să creezi un driver funcțional într-un timp destul de scurt

-mai ușor de dezvoltat un driver USB pe o gazdă(precum computerul adică host-ul) datorită flexibiliăți și posibilitatea de a avea kituri de dezvoltare și debugging.