Universitatea Politehnică Bucureşti

Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

**Sisteme de Operare**

Driver Kernel Interface

**Profesor coordonator:** Stefan Stancescu

**Studenţi:** Taifas Emil-Valentin

Rosca Mihai

**Grupa:** 431A (CTI)

Anul universitar

2014-2015

Cuprins

1. Interfata Kernel …………..…………………………………….3

1.1 Kernel – Generalitati ……………………………………3

1.2 Functiile Kernel …….…………………………………3

1.3 Serviciile Interfetei Driver-Kernel ….…………………4

2. Linux Streams …………………………………………………6

2.1 Alocarea memoriei prin LiS ……………………………7

2.2 Functii unitare prin LiS ……….………………………..8

3. Linux API – Generalitati ……………………………………….9

4. Detalierea Driverelor Interfetei Kernel ………………………..10

Driver – Kernel Interface

**1.Interfata Kernel**

**1.1 Kernel – Generalitati**

Centrul oricarui sistem de operare este kernelul si acesta cuprinde toate utilitatile necesare pentru ca acel computer sa functioneze.

Linux este numele unui kernel din spatele sistemului de operare, si anume este un fisier executabil care se incarca la bootare si se executa.

Kernelul este deschis de boot loaderul incarcat si executat de BIOS la pornirea calculatorului. Kernelul se ocupa cu alocarea de resurse pentru un proces, cum ar fi spatiul de memorie RAM necesar si timpul de executie al procesorului, toate aceste resurse fiind eliberate odata cu incheierea procesului la care facem referire, aceasta actiune intra tot in atributiile kernelului.3

**1.2 Functiile Kernel**

Dupa incarcare, kernelul are in vedere urmatoarele functii:

-managementul dispozitivelor; asigura conditiile optime pentru o buna functionare a lor, a memoriei si a proceselor.

- fluxul de informatii este controlat astfel incat aplicatiile sa comunice bine cu sistemul.

-mai are control aspura unor functii ca :

* spatiul de swap – memoria hard-disk-ului folosita de kernel;
* sistemul de fisiere – sistemul de ierarhizare al fisierelor pe disk.
* programele demon – reprezinta procese speciale ce sunt pornite dupa incarcarea sistemului de operare, asemenea serviciilor din Windows;

Interfata Driver-Nucleu cuprinde serviciile ce sunt puse la dispozitia componentelor, de catre kernel, servicii executate in kernel si apelate de driver.

**1.3 Serviciile Interfetei Driver-Kernel**

Interfata Driver-Kernel cuprinde 2 tipuri de servicii:

* Servicii generale .
* Servicii specifice unor anumite familii de procesoare.

Serviciile generale reprezinta serviciile ce sunt accesibile oricarui driver indiferent de procesor, si cuprind dupa cum urmeaza:

* + - Asteptarea exacta: reprezinta serviciul prin care driverele asteapta pentru o perioada scurta de timp anumite permisiuni sau resurse.

Sintaxa este:

void udelay(long micro\_secs); si este definita in <linux/delay.h> care nu foloeste jiffies.

* + - Timeout: reprezinta serviciul folosit de diverse driver prin intermediul caruia verifica daca o anumita actiune inceputa se va termina inainte de o anumita perioada data.
    - Mascarea intreruperilor: reprezinta serviciul prin care un driver impiedica intreruperea unei anumite portiuni din executie.
    - Gestiunea evenimentelor: reprezinta serviciul asigurat de kernel tuturor driverelor cu prioritate ridicata. Propagarea evenimentului de la un driver de prioritate mare incepe odata cu apelul driverului respectiv catre handlerul unui driver de prioritate mai mica.
    - Alocarea memoriei: reprezinta serviciul de alocare a memoriei driverelor care au implementata o alocare dinamica de memorie. Valoare memoriei ocupate de driver ii este necunoscuta kernelului, deci eliberarea ei se va ocupa acelasi driver care a alocat-o.

Sintaxa functiei de alocare/eliberare de memorie:

void \*kmalloc(size\_t nbytes, int type) ;

void kfree(const void \*ptr) ;

void \*vmalloc(unsigned long size);

void vfree(void \*ptr) ;

* + - Dezactivarea “preemptiva” a firelor de executie: reprezinta serviciul prin care executia unui thread este intrerupta la aparitia unui alt thread cu prioritate mai mare, threadul din urma poate rula pana la sfarsitul executiei sau numai un anumit interval de timp.
    - I/E : reprezinta serviciul ce asigura functii optimizate pentru transferul de biti.
    - Sincronizarea: reprezinta serviciul ce se ocupa de managementul apelurilor indreptate catre acelasi serviciu din diferite trasee de executie. Traseul de executie al DKI este lansat de kernel la pornirea acestuia, si functioneaza ca un mecanism in urmatoarele cazuri:

1. normale – toate apelurile catre porniri sau opriri de drivere se fac din interiorul traseului de executie al DKI.
2. speciale – cazuri in care pornirile sau opririle de drivere nu fac parte din procesul de initializare al driverelor din kernel .

**2.Linux Streams**

Linux Streams ( LiS ) este o arhitectura care ofera o alta interfata intre kernel si anumite drivere numite Streams Drivers; difineste un model de comunicare intre servicii pentru sistemele Linux/Unix.

Streams: creeaza un flux de date. Acesta se compune din 3 parti;un varf, unul sau mai multe module si un driver.

Cateva dintre driverele Streams sunt urmatoarele :

-clone-drvr – acest driver ajuta LiS la implementarea functiei

de deschidere clona ( ‘clone open’ ajuta la deschidrea unui

dispozitiv dintr-un fond comun de drivere ).

Se afla in /dev/clone\_drvr

-link-drvr – creaza o legatura intre driverele Stream si

driverele de retea. Se afla in /dev/ldl

-loop-around – realizeaza o conexiune intre 2 fluxuri de date

astfel incat datele inscrise intr-un flux pot fi accesate din

celalalt flux. Se afla in /dev/loop.1, /dev/loop.2.

-mini-mux – driverul este folosit de LiS pentru a isi autotesta

procedurile.

Se afla in /dev/minimux.1, /dev/minimux.2.

-printk – driverul primeste mesaje scrise si le printeaza cu

ajutorul functiei printk din kernel.

Se afla in /dev/printk.

-sad – driverul realizeaza o gestiune a functiilor LiS.

Se afla

in /dev/sad.

**2.1Alocarea memoriei prin LiS**

Sintaxa functiilor de alocare/eliberare de memorie este :

void \*ALLOC(int nrbytes) ;

void \*ALLOCF(int nrbytes, char \*tag) ;

void FREE(void \*ptr) ;

O alta varianta este aceea de a folosi alocatorii de memorie din kernel, exemplele ajung intr-un apel al functiei “kmalloc”:

GFP\_ATOMIC, GFP\_KERNEL, GFP\_DMA

void \*lis\_alloc\_atomic(int nrbytes) ;

void \*lis\_alloc\_kernel(int nrbytes) ;

void \*lis\_alloc\_dma(int nrbytes) ;

void \*lis\_free\_mem(void \*mem\_area) ;

Un driver Stream poate crea fire de executie:

pid\_t lis\_thread\_start(int (\*fcn)(void \*),void \*arg, const char \*name);

int lis\_thread\_stop(pid\_t pid);

Accesarea porturilor I/E se face cu ajutorul sintaxei:

Int check\_region(unsigned int from, unsigned int extent);

void request\_region(unsigned int from, unsigned int extent, const char \*name) ;

void release\_region(unsigned int from,unsigned int

extent);

Secventele de printare

int printk(const char \*fmt, ...) ;

int sprintf(char \*bfr, const char \*fmt, ...) ;

int vsprintf(char \*bfr, const char \*fmt, va\_list args) ;

**2.2Functii unitare prin LiS**

void lis\_atomic\_set(lis\_atomic\_t \*atomic\_addr,int val) ;

int lis\_atomic\_read(lis\_atomic\_t \*atomic\_addr) ;

void lis\_atomic\_add(lis\_atomic\_t \*atomic\_addr,int amt) ;

void lis\_atomic\_sub(lis\_atomic\_t \*atomic\_addr,int amt) ;

void lis\_atomic\_inc(lis\_atomic\_t \*atomic\_addr) ;

void lis\_atomic\_dec(lis\_atomic\_t \*atomic\_addr) ;

int lis\_atomic\_dec\_and\_test(lis\_atomic\_t \*atomic\_addr);

Cateva interfete LDI kernel ofera posibilitatea LDI-ului sa urmareasca si sa raporteze informatii despre utilizarea device-ului.

Alte interfete LDI kernel permit modulelor kernel sa aiba acces la operatii cum ar fi deschiderea, citirea si scrierea unui device tinta.

Linux API este spatiul API kernel-utilizator, care permite programelor in spatiul utilizatorului sa acceseze resursele sistemului si serviciile Linux Kernel.

**3.API-ul**

**3.1Generalitati**

Aici vom detalia Driverele Interfetei Kernel. Multe dintre aceste functii sunt identice cu cele din Kernel C, si vor fi in multe configuratii doar ambalaje pentru acestea. In configuratiile non-kernel, acestea vor fi sustinute direct de HAL sau de cod pentru a emula comportamentul dorit.

API-ul este definit de header-ul cyg/hal/drv\_api.h .

**cyg\_drv\_isr\_lock**

Descriere:

Opreste trimiterea de intreruperi, prevenind astfel toate ISR-urile functionale. Aceasta functie tine un contor pentru numarul de apelari.

**4.Detalierea Driverelor Interfetei Kernel**

**cyg\_drv\_isr\_unlock**

Functie:

void cyg\_drv\_isr\_unlock()

Argumente: Niciunul

Rezultat: Niciunul

Nivel: DSR

Descriere:

Reporneste trimiterea de intreruperi., dandu-I voie ISR-ului sa functioneze. Aceasta functie decrementeaza contorul tinut de cyg\_drv\_isr\_lock(), si permite retrimiterea de intreruperi cand acesta ajunge la 0.

Functie:

void cyg\_drv\_dsr\_lock()

Argumente: Niciunul

Rezultat: Niciunul

Nivel: Thread

Descriere:

Opreste programarea de DSR-uri. Aceasta functie tine un contor pentru numarul de apelari.

**cyg\_drv\_dsr\_unlock**

Functie:

void cyg\_drv\_dsr\_unlock()

Argumente: Niciunul

Rezultat: Niciunul

Nivel: Thread

Descriere:

Reporneste programarea de DSR-uri. Aceasta functie decrementeaza contorul incrementat de . DSR-urilor le este permisa trimiterea doar cand contorul ajunge la 0.

**cyg\_drv\_mutex\_init**

Functie:

void cyg\_drv\_mutex\_init(cyg\_drv\_mutex \*mutex)

Argumente: mutex - pointer to mutex to initialize

Rezultat:Niciunul

Nivel: Thread

Descriere:

Initializeaza pointerul mutex ca fiind argumentul mutex.

**cyg\_drv\_mutex\_destroy**

Functie:

void cyg\_drv\_mutex\_destroy( cyg\_drv\_mutex \*mutex )

Argumente: mutex - pointer to mutex to destroy

Rezultat: Niciunul

Nivel: Thread

Descriere:

Distruge mutexul pointat de argumentul mutex.

**cyg\_drv\_mutex\_lock**

Functie:

cyg\_bool cyg\_drv\_mutex\_lock( cyg\_drv\_mutex \*mutex )

Argumente: mutex – pointeaza catre ce are mutexul de blocat

Rezultat: ADEVARAT daca threadul a solicitat blocarea, falsa daca nu. Nivel: Thread

Descriere:

Incearca blocarea mutex-ului pointat de argumentul mutex. Daca mutex-ul este deja blocat de un al thread, atunci acest thread va astepta pana acel thread termina. Daca rezultatul acestei functii este FALS, atunci threadul a fost scos din asteptare de alt thread. In acest caz, mutex-ul nu va fi fost inchis.

**cyg\_drv\_mutex\_trylock**

Functie:

cyg\_bool cyg\_drv\_mutex\_trylock( cyg\_drv\_mutex \*mutex )

Argumente: mutex – pointeaza catre ce are mutexul de blocat

Rezultat: ADEVARAT daca mutexul a fost blocat, falsa daca nu. Nivel: Thread

Descriere:

Incearca blocarea mutex-ului pointat de argumentul mutex fara avertisment. Daca mutex-ul este deja blocat de alt thread, atunci aceasta functie returneaza FALS. Daca functia poate bloca mutex-ul fara avertisment, atunci rezultatul returnat este ADEVARAT.

**cyg\_drv\_mutex\_unlock**

Functie:

void cyg\_drv\_mutex\_unlock( cyg\_drv\_mutex \*mutex )

Argumente: mutex – pointeaza catre ce are mutexul de deblocat

Rezultat: Niciunul

Nivel: Thread

Descriere:

Deblocheaza mutexul pointat de argumentul mutex. Daca exista threaduri in asteptare pentru blocarea acestuia, unul din ele este activat pentru a i se permite sa il primeasca.

**cyg\_drv\_mutex\_release**

Functie:

void cyg\_drv\_mutex\_release( cyg\_drv\_mutex \*mutex )

Argumente: mutex - pointer to mutex to release

Rezultat: Niciunul

Nivel: Thread

Descriere:

Elibereaza toate threadurile in asteptare a mutexului pointat de argumentul mutex. Aceste threaduri returneaza FALS din cyg\_drv\_mutex\_lock()si nu vor fi preluat mutexul. Aceasta functie nu actioneaza asuprea threadurilor care au deja mutexul in cadrul lor.

**cyg\_drv\_cond\_init**

Functie:

void cyg\_drv\_cond\_init( cyg\_drv\_cond \*cond,

cyg\_drv\_mutex \*mutex )

Argumente:

cond—conditia de initializare a variabilei

mutex—mutexul este asociat cu variabila conditiei

Rezultat: Niciunul

Nivel: Thread

Descriere:

Inițializa variabila conditie indicata de argumentul cond. Argumentul mutex trebuie indica un mutex cu care această variabilă condiție este asociată. Un fir de executie poate aștepta doar la această variabilă condiție atunci când a blocat deja mutexul asociată. Așteptare va determina daca mutex va fi deblocat, iar când firul este redeschis, acesta va solicita automat mutexul înainte de a continua.

**cyg\_drv\_cond\_destroy**

Functie:

void cyg\_drv\_cond\_destroy( cyg\_drv\_cond \*cond )

Argumente: cond - condition variable to destroy

Rezultat: Niciunul

Nivel: Thread

Descriere:

Distruge variabila conditiei pointata de argumentul cond.

**cyg\_drv\_cond\_wait**

Functie:

void cyg\_drv\_cond\_wait( cyg\_drv\_cond \*cond )

Argumente:

cond - condition variable to wait on

Rezultat: Niciunul

Nivel: Thread

Descriere:

Așteptați un semnal pe variabila conditie indicat de argumentul cond. Firul trebuie să fi blocat mutexul asociat înainte de așteptare pe această variabilă condiție. În timp ce așteaptă firul, mutex-ul va fi deblocat, și va fi re-blocat înainte de intoarcerea functiei. Este posibil ca firele de așteptare pe o variabila condiție să reporneasca sponta. Din acest motiv, este necesar să se utilizeze această Functie într-o buclă care re-testează starea de fiecare dată când se întoarce.

**cyg\_drv\_cond\_signal**

Functie:

void cyg\_drv\_cond\_signal( cyg\_drv\_cond \*cond )

Argumente:

cond – variabila conditie la semnal

Rezultat: Niciunul

Nivel: DSR

Descriere:

Semnaleaza variabila conditie indicata de argumentul cond. Dacă există fire in asteptare pe această variabilă, cel puțin una dintre ele va fi trezita. In unele configurații, nu poate fi nici o diferenta intre acest Functie și cyg\_drv\_cond\_broadcast ().

**cyg\_drv\_cond\_broadcast**

Functie:

void cyg\_drv\_cond\_broadcast( cyg\_drv\_cond \*cond )

Argumente:

cond – variabila conditiei catre care se face emisia

Rezultat: Niciunul

Nivel: DSR

Descriere:

Semnaleaza variabila de conditie pointata de argumentul cond. Daca exista threaduri in asteptare pentru aceasta variabila, ele vor fi activate.

**cyg\_drv\_interrupt\_create**

Functie:

void cyg\_drv\_interrupt\_create(

cyg\_vector\_t vector,

cyg\_priority\_t priority,

cyg\_addrword\_t data,

cyg\_ISR\_t \*isr,

cyg\_DSR\_t \*dsr,

cyg\_handle\_t \*handle,

cyg\_interrupt \*intr

)

Argumente:

vector—vector la care sa fie atasat

priority--queueing priority

data--data pointer

isr--interrupt service routine

dsr--deferred service routine

handle--returned handle

intr—obiectul de intreruperi

Rezultat: Niciunul

Nivel: Thread

Descriere:

Crearea unui obiect de intreruperi si returnarea unui mijloc. Obiectul contine informatii despre care vector de intreruperi sa foloseasca ISR si DSR care va fi apelat dupa ce obiectul de intreruperi este atasat. Obiectului de intreruperi va fi alocata memorie prin parametrul intr. Acest obiect de intreruperi nu este atasat imediat; el trebuie atasat impreuna cu apelul functiei cyg\_interrupt\_attach().

**cyg\_drv\_interrupt\_delete**

Functie:

void cyg\_drv\_interrupt\_delete( cyg\_handle\_t interrupt )

Argumente: interrupt—intrerupere catre delete

Rezultat: Niciunul

Nivel: Thread

Descriere:

Detaseaza intreruperea de vector si elibereaza memoria din intr pentru a fi refolosita de cyg\_drv\_interrupt\_create().

**cyg\_drv\_interrupt\_attach**

Functie:

void cyg\_drv\_interrupt\_attach( cyg\_handle\_t interrupt )

Argumente: interrupt—intrerupere catre atasare

Rezultat: Niciunul

Nivel: ISR

Descriere:

Ataseaza intreruperea vectorului astfel incat intreruperile vor fi livrate ISR-ului cand apare intreruperea.

**cyg\_drv\_interrupt\_detach**

Functie:

void cyg\_drv\_interrupt\_detach( cyg\_handle\_t interrupt )

Argumente: interrupt—intrerupere catre detasare

Rezultat: Niciunul

Nivel: ISR

Descriere:

Detaseaza intreruperea de vector astfel incat vectorul sa nu mai poata fi trimis ISR-ului.

**cyg\_drv\_interrupt\_mask**

Functie:

void cyg\_drv\_interrupt\_mask(cyg\_vector\_t vector )

Argumente: vector--vector to mask

Rezultat: Niciunul

Nivel: ISR

Descriere:

Programeaza controllerul de intreruperi sa opreasca livrarea de intreruperi pe vectorul dat. Pe arhitecturile care imprementeaza nivele de intreruperi de prioritate, acestea pot chiar opri toate intreruperile de nivel mic.

**cyg\_drv\_interrupt\_unmask**

Functie:

void cyg\_drv\_interrupt\_unmask(cyg\_vector\_t vector )

Argumente: vector--vector to unmask

Rezultat: Niciunul

Nivel: ISR

Descriere:

Programeaa controllerul de intreruperi sa re-primeasca intreruperi de la vectorul dat.

**cyg\_drv\_interrupt\_acknowledge**

Functie:

void cyg\_drv\_interrupt\_acknowledge( cyg\_vector\_t vector )

Argumente: vector--vector to acknowledge

Rezultat: Niciunul

Nivel: ISR

Descriere:

Face orice procese necesare pentru microcontrolerul de intreruper si in CPU pentru a opri cererea de intrerupere. Orice ISR poate avea nevoie sa programeze hardware-ul device-ului pentru a preveni repornirea trimiterilor de inreruperi.

**cyg\_drv\_interrupt\_configure**

Functie:

void cyg\_drv\_interrupt\_configure(

cyg\_vector\_t vector,

cyg\_bool\_t Nivel,

cyg\_bool\_t up

)

Argumente: vector--vector to configure

Nivel- -Nivel or edge triggered

up--rising/falling edge, high/low Nivel

Rezultat: Niciunul

Nivel: ISR

Descriere:

Programeaza microcontrolerul de intreruperi cu caracteristicile de sursei de intreruperi. Argumentul Nivel alege intre intreruperile Level sau edge-triggered. Argumentul sus alege intre high si low pentru intreruperi level triggered si intre rising si falling pentru intreruperile edge-triggered. Aceasta functie functioneaza doar cu microcontrelerele de intreruperi care pot controla acesti parametri.

**cyg\_drv\_interrupt\_level**

Functie:

void cyg\_drv\_interrupt\_Nivel( cyg\_vector\_t vector, cyg\_priority\_t Nivel)

Argumente:

vector--vector to configure

Nivel--Nivel to set

Rezultat: Niciunul

Nivel: ISR

Descriere:

Programeaza controlerul de intreruperi sa trimita intreruperea data la nivelul de proritate dorit. Aceasta functie functioneaza doar cu controlere de intreruperi care pot controla acest parametru.

Bibliografie:

1.Linux for Dummies 8th edition 2008

2.Linux from scratch, versiune 2.2 2002

3. <http://ro.wikipedia.org/linux>

4. <http://en.wikipedia.org/wiki/Linux_kernel_interfaces>

5. Linux bible 2007 edition

6. Linux Device Drivers, 3rd Edition by Greg Kroah-Hartman

<http://www.makelinux.net/ldd3>

7. The Linux Kernel Driver Interface by Greg Kroah-Hartman

<https://www.kernel.org/doc/Documentation/stable_api_nonsense.txt>

8. Oracle-Writing Device Drivers

<http://docs.oracle.com/cd/E23824_01/html/819-3196/ldi-2.html#euayr>

9. Device Driver Interface

<http://ecos.sourceware.org/docs-1.3.1/ref/ecos-ref.13.html>

Impartirea temei a fost facut in felul urmator:

1. Interfata Kernel

1.1 Kernel – Generalitati- Taifas Emil

1.2 Functiile Kernel - Rosca Mihai

1.3 Serviciile Interfetei Driver-Kernel - Taifas Emil

2. Linux Streams - Taifas Emil

3. LiS

3.1 Alocarea memoriei prin LiS - Taifas Emil

3.2 Functii unitare prin LiS - Rosca Mihai

4. Linux API – Generalitati - Rosca Mihai

5. Detalierea Driverelor Interfetei Kernel - Rosca Mihai