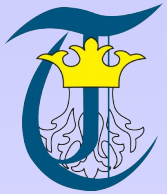


Universitatea *Transilvania* Brașov
Facultatea de Inginerie Electrică și Știința Calculatoarelor
Departamentul de Electronică și Calculatoare

Sisteme de operare distribuite și în timp real

– curs –
master SECI anul I

ș.l. dr. ing. Kertész Csaba-Zoltán

Drivere embedded

- driverele Linux sunt părți ai subsistemului IO
- subsistemul IO permite aplicațiilor să acceseze hardware de nivel jos prin intermediul unei interfețe de apeluri sistem bine definite
- tipuri de drivere Linux
  - character device
  - block device
  - network device

Dispozitive de tip caracter

- sunt drivere pentru dispozitive care pot fi accesate în mod secvențial
- accesul poate fi făcut pe octeți sau pe blocuri de diferite dimensiuni
- aplicațiile accesează aceste dispozitive prin intermediul apelurilor sistem open, read, write
- aplicațiile pot influența funcționarea dispozitivului prin apeluri ioctl
- exemple:
  - UART, SPI, I²C, USB (CDC)

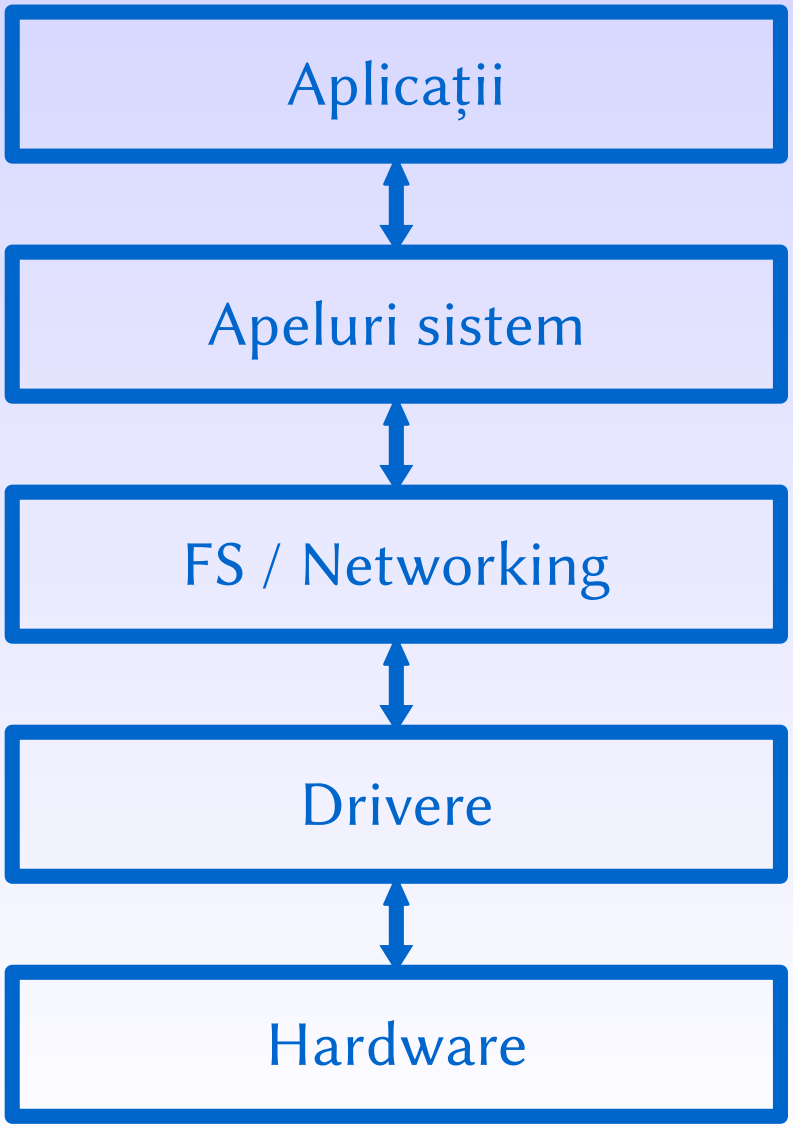
Dispozitive de tip bloc

- modelează dispozitive care pot fi accesate aleator
- accesul se face de obicei pe blocuri de date
- aceste dispozitive sunt folosite pentru stocarea unor sisteme de fișiere
- aplicațiile nu pot accesa direct driverul, ele trebuie să accese sistemul de fișiere montat deasupra driverului
- exemple:
  - HDD, CD-ROM, USB (mass storage)

Dispozitive de tip rețea

- astfel de drivere deși modelează dispozitive care sunt accesate secvențial sunt tratate ca o clasă sperată, ele interacționând cu stiva de protocoale de rețea
- aplicațiile nu au acces direct la aceste drivere
- numai subsistemul de rețea poate interacționa cu ele
- exemple:
  - Ethernet

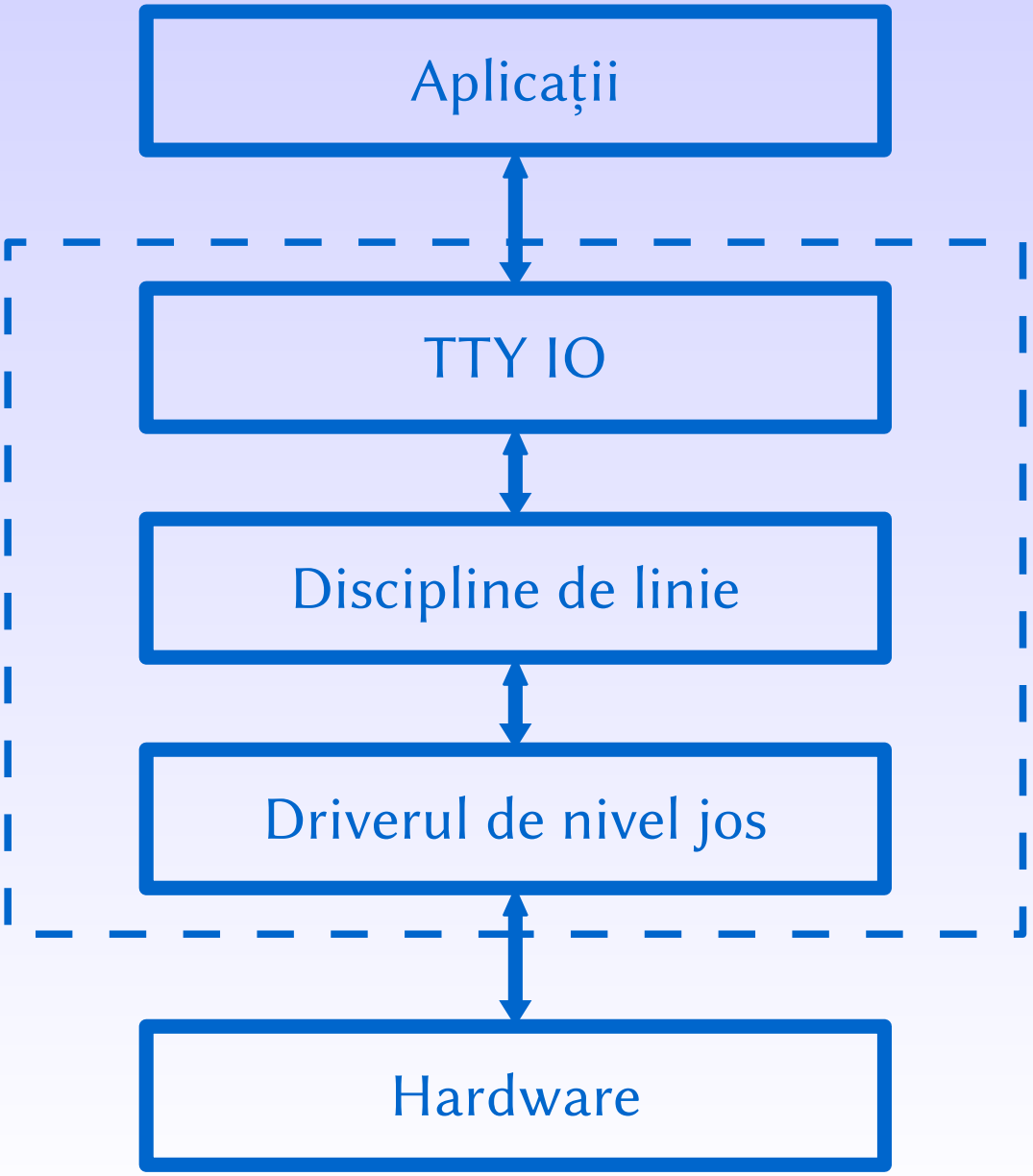
Arhitectura sistemului de drivere



Driverul de serială

- interfața serială în Linux este strâns legată de subsistemul TTY
- nivelul TTY este o clasă speciala de dispozitive de tip caracter care se intercalează deasupra driverului de nivel jos
- TTY tratează toate dispozitivele seriale indiferent de modul cum este folosit interfața
- aplicațiile nu accesează direct driverul de serială, ci driverele TTY

Subsistemul TTY



Discipline de linie

- fiecare dispozitiv de tip TTY este asociat cu o disciplină de linie, care tratează modul cum sunt transmise datele pe linia serială
- implicit disciplina de linie (integrate în fiecare kernel) este N_TTY, care oferă un terminal simplu pe serială
- pot fi implementate și alte discipline de linie pentru protocoalele mai complicate folosite pe interfețe seriale (X.25, PPP/SLIP)

Terminale

- fiecare proces în Linux are asociat un terminal de control
  - procesul asociază intrarea standard (stdin) și ieșirea standard (stdout) respectiv ieșirea pentru erori (stderr) cu acest terminal de control
- subsistemul TTY împreună cu subsistemul de management a proceselor alocă automat terminalul de control a proceselor
- CTTY asociat unui proces poate fi o interfață serială, tastatura, monitorul în mod text

Pseudo-terminale

- datorită faptului că nu există suficiente interfețe seriale pentru terminalele de control a tuturor proceselor, subsistemul TTY implementează și o serie de pseudo-terminale (PTY)
- aceste pseudoterminale pot fi folosite și pentru IPC
  - terminalele PTY pot fi redirecționate între procese
  - procesele pot fi pornite cu terminale de control ascunse (la care utilizatorul nu are acces direct)
  - de ex.: telnet, ssh

Driverul de serială în Linux

- interfața UART are un driver în kernelul Linux
- driverul accesează dispozitivul prin intermediul structurilor:
  - uart_driver
    - interfața comună pentru apeluri sistem
  - uart_state
    - fiecare port serial are asociat această structură care conține starea actuală
  - uart_port
    - configurația hardware
  - uart_ops
    - funcții low level de acces

Folosirea serialei în Linux

- driverul tty se accesează cu funcțiile open (cu opțiuni specifice TTY), read, write (operații pe șiruri de caractere blocante, cu timeout)
- controlul interfeței seriale se realizează printr-o structură termios
  - permite setarea ratei de transfer, formatul cadrului de date (nr. biți de date, de stop, de paritate)
  - permite asociere unei discipline de linii
  - pentru setarea flagurilor din termios se folosește funcția tcsetattr

Driverul de Ethernet

- driverul de rețea în Linux este o clasă de drivere speciale
- aceste drivere nu se accesează prin intermediul fișierelor, ci prin intermediul socketților oferite de subsistemul rețea
- Linux oferă suport implicit pentru o interfață de driver Ethernet (fiind orientat mult pe rețea) prin intermediul structurii net_device
- driverul oferă o rutină de probă: kernelul probează toate driverele de rețea existent la pornire

Pornirea driverului Ethernet

- la pornirea sistemului kernelul apelează rutina de probă cu o structură net_device umplută cu valori implicite
- dacă rutina de probă găsește dispozitivul asociat atunci completează structura cu valori corespunzătoare și pune la dispoziția subsistemului de rețea rutinele de transfer de date asociate dispozitivului

Transmisia datelor pe Ethernet

- modul de transmisie este dependent de dispozitivul folosit, de aceea și funcțiile de transfer prezentate subsistemului de rețea trebuie realizate de BSP
- kernelul creează cozi de mesaje prin intermediul căreia subsistemul de rețea comunică cu driverul

Dispozitive USB

- USB este o magistrală de comunicație serială master-slave
- masterul (USB host) conține un USB controller driver care controlează dispozitivele conectate pe magistrală
- toată comunicația pe USB este pornită de host
- dispozitivele pot transmite date numai la cerere
- lățimea de bandă poate fi alocată de host fiecărei dispozitiv în parte

Dispozitive USB

- Linux oferă suport atât pentru host-uri USB cât și pentru dispozitive slave
- pe partea host kernelul Linux oferă implicit drivere pentru controllere USB existente
- pentru dispozitive slave (gadget-uri) Linux oferă drivere generice pentru dispozitivele din clasele: mass storage (stickuri USB, harddisk extern), HID (tastatură, mouse, touchpad) și CDC (modem, bluetooth, rețea)

Driverul USB

- driverul de controller
  - oferă abstractizare a controllerului USB și oferă API către gadgeturi
  - oferă o interfață independentă de gadget pentru driverele de nivel înalt
- driverul de gadget
  - este driverul low level dedicat dispozitivului USB
  - fiecare dispozitiv necesită un driver de gadget
- gadgeturile oferă funcționalități similare cu alte dispozitive, de aceea drivere de gadget se integrează și cu aceste drivere (rețea,serială,...)

Module kernel

- modulele de kernel sunt subsisteme ale kernelului care sunt adăugate dinamic la un kernel aflat în rulare
- folosirea modulelor reduce dimensiunea kernelului, pentru că numai subsistemele ce sunt folosite vor fi încărcate
- modulele oferă un API standard, care permite încărcarea/descărcare lor de către kernel și extind apelurile sistem standard pentru comunicarea cu aplicațiile

Module API

- funcții de intrare și ieșire
  - modulul trebuie să ofere funcțiile module_init() și module_exit(), care sunt apelate de kernel la încărcarea respectiv descărcarea modulului
- pasarea parametrilor
  - la încărcarea modulelor, acestora pot fi pasate parametri într-o manieră similară cu linia de comandă
  - parametrii trebuie declarați de către modul prin niște tabele de asociere (nume parametru – variabilă)

Module API

- reference count
  - fiecare modul trebuie să stocheze într-o variabilă numărul de încărcări ai modulului respectiv
  - când acest număr cade la 0, modulul poate fi descărcat
  - pentru a evita referirile la module inexistente toate apelurile către module trebuie să treacă prin referințe tratate de kernel
- declararea licenței
  - modulele Linux au obligația de a declara licența la pornire

Încărcarea/descărcarea modulelor

- kernelul oferă apeluri sistem pentru încărcarea, descărcare și accesul modulelor
- există și programe standard pentru manipularea modulelor
  - insmod: încearcă linkarea modulului la kernelul în rulare prin rezolvarea tuturor simbolurilor exportate de kernel
  - modprobe: încearcă să încarce modulele de care depinde un modul (modules.dep) după care încarcă modulul
  - rmmod: descarcă modulul (numai dacă reference count = 0)

Folosirea driverelor în forma modulelor

- driverele pot fi compilate în forma modulelor
- astfel se vor încărca numai driverele folosite în sistem
- kernelul exportează simboluri pentru accesare tuturor funcțiilor oferite de drivere
- când se execută un apel către o astfel de funcție (exportată dar nerezolvată), kernelul va încerca să încarce modulul aferent (prin executarea funcțiilor de probă din clasa driverelor)

Board Support Package

- software folosit pentru inițializarea dispozitivelor hardware întâlnite în sistemul încorporat
- implementează rutinele specifice hardware-ului ce vor fi folosite de kernel și drivere
- BSP ascunde toate detaliile legate de procesor și placa în care este încorporată, permițând astfel o portare ușoară a sistemului de operare și a driverelor aferente
- BSP (µC) = HAL (PC)

Componentele BSP

- suport pentru procesor
  - tratează caracteristicile specifice procesoarelor embedded (MIPS, ARM, PowerPC, etc...)
- suport pentru periferice
  - bootloader
  - harta memoriei
  - timere
  - PIC (Programmable Interrupt Controller)
  - RTC (Real Time Clock)
  - serială (UART) pentru consolă și debug
  - magistrale (ISA, PCI)
  - DMA
  - Power management

Caracteristici BSP

- nu există un standard pentru HAL/BSP în Linux
- fiecare arhitectură are propria implementare într-un subdirector din arch/ și fișierele header în include/asm-*XXX*/ (*XXX* fiind numele procesorului)
- referirea la HAL în restul kernelului este făcut prin secvențe condiționale la compilare
- configurarea folosirii acestor secvențe este realizată odată cu configurarea kernelului (make config / make menuconfig / make xconfig) prin selectarea opțiunilor specifice arhitecturii

Configurare BSP pentru placa de dezvoltare

- se poate realiza o setare prealabilă a elementelor din configurarea kernelului pentru a placă de dezvoltare aparte
- în arch/XXX/ se inserează un fișier Kconfig (sau config.in în cazul kernelului 2.4) cu opțiunile specifice procesorului XXX
- în fișierul Kconfig se poate adăuga opțiuni specifice unei plăci de dezvoltare, de ex:

```
ifdef CONFIG_EUREKA
LIBS += arch/mips/eureka/eureka.o
SUBDIRS += arch/mips/eureka
LOADADDR := 0x80000000
endif
```

Opțiuni de configurare

- se pot crea și opțiuni de configurare specifice plăcii

```
dep_bool 'Support for EUREKA board' CONFIG_EUREKA

if ["$CONFIG_EUREKA"="y" ]; then
    choice 'Eureka Clock Speed' \
        "75 CONFIG_SYSCLK_75 \
        100 CONFIG_SYSCLK_100" 100
fi

...

if ["$CONFIG_EUREKA"="y" ]; then
    define_bool CONFIG_PCI y
    define_bool CONFIG_ISA y
    define_bool CONFIG_NONCOHERENT_IO y
    define_bool CONFIG_NEW_TIME_C y
fi
```

Interfața bootloader

- rutina cea mai specifică procesorului
- fiecare procesor prezintă o arhitectură diferită în privința pornirii sistemului, astfel fiecare procesor și fiecare sistem încorporat în parte are un secvență de boot specifică
- secvența de boot poate fi integrată în kernel, aceasta legându-se la adrese specifice, sau poate fi realizat într-un program separat care la rândul său va apela kernelul
  - majoritatea nucleelor RTOS încorporează secvența de bootare

Bootloader pentru Linux

- Linux pornește de la premiza că se rulează din memoria RAM (moștenit de la 386)
  - există excepții prin patchul XIP (eXecute In Place) pentru arhitecturi ce nu oferă suport pentru aceasta, dar implică modificări severe în kernel
- pentru Linux trebuie neapărat să existe un bootloader care inițializează sistemul până când ajunge la un punct comun pentru a putea continua kernelul Linux implicit

Funcțiile bootloaderului

- funcții obligatorii
  - inițializarea procesorului, a controlerului de memorie și a dispozitivelor hardware esențiale pentru încărcarea kernelului (de ex. flash)
  - încărcarea kernelului: copiere din dispozitivul de stocare în memorie
- funcții opționale
  - depind de sistemul încorporat în care este folosit
  - poate inițializa alte dispozitive
  - poate pasa argumente kernelului pentru o configurare dinamică

Secvența de bootare
(la un sistem încorporat)

- bootarea
  - bootloaderul pornește din flash
  - se inițializează registrele de bază și cache-ul (dacă e cazul)
  - se verifică memoria și dispozitivele hardware de pe placă (POST)
- relocatarea
  - bootloaderul se autocopiază din flash în RAM (pentru rulare mai rapidă)
  - la copiere se poate efectua și o dezarhivare dacă bootloaderul era comprimat în flash

Secvența de bootare

- inițializarea dispozitivelor
  - se inițializează dispozitivele de bază necesare pentru interacțiunea cu utilizatorul (consola)
  - se poate inițializa și alte dispozitive necesare pentru preluarea kernelului (flash, USB, Ethernet)
- UI
  - se prezintă o interfață utilizatorului prin intermediul căreia se poate selecta imaginea de kernel ce trebuie încărcată
- Încărcarea kernelului
  - se copiază kernelul în memorie (+ initrd)

Secvența de bootare

- prepararea kernelului
  - argumente pot fi pasate kernelului
  - se completează argumentele din linia de comandă și se plasează în adrese fixe cunoscute de kernel
- se bootează kernelul
  - se va face un salt la punctul de intrarea în kernel
  - în mod normal bootloaderul nu mai este folosit, așa că zona de memorie ocupată de aceasta este eliberată de kernel în timpul mapării memoriei

Criterii de selecție a bootloaderului

- suport pentru procesor
  - grub: numai pentru x86
  - pmon, yamon, blob, etc...
- dimensiunea ocupată în flash
- suport pentru bootare din rețea
  - important pentru depanare
- existența UI în consolă
- posibilitate de upgrade
  - rutine de ștergere/scriere flash încorporat

Argumentele kernelului

- argumente pot fi pasate kernelului într-o manieră similară cu linia de comandă
- aceste argumente permit rezolvarea unor probleme hardware prin configurarea dinamică a kernelului
- lista parametrilor poate fi văzută în /proc/cmdline
- parametrii mai importanți
  - root: dispozitivul folosit ca root în FS
  - nfsroot: FS montat în rețea
  - mem: dimensiunea existentă a memoriei
  - debug: mesaje de depanare la pornirea kernelului

Maparea memoriei

- prima rutină importantă realizată de kernel
- sistemul va crea spațiul de adrese virtuale în care vor fi mapate toate memoriile și dispozitivele din sistem
- se alocă adresele fixate pentru diferite dispozitive ce nu mai pot fi alterate ulterior
- marchează zonele importante în memorie (de ex. cele alocate kernelului)
- inițializează sistemul de translatare adrese virtuale -> adrese fizice

Mapări de memorie în Linux

- processor map
  - folosit pentru politicile de MM realizate în procesor (depinde de arhitectura procesorului)
- board map
  - folosit pentru maparea dispozitivelor (de obicei acestea având niște adrese fixe)
- software map
  - aici vor fi plasate componentele software
  - este realizat în timpul linkării programelor (în cazul kernelului în arch/*XXX*/ld.script.in

Managementul întreruperilor

- fiecare sistem tratează întreruperile în mod diferite
- de obicei există un controler de întreruperi specific (PIC – Programmable Interrupt Controller)
- întreruperile hardware sunt tratate mai întâi de PIC, care poate devia întreruperile acordând prioritate hardware și metode diferite de tratare
- pentru programarea PIC trebuie să există driver special, care însă trebuie mascat prin BSP pentru portare ușoară

Tratarea întreruperilor în Linux

- Linux tratează toate întreruperile ca întreruperi logice (un număr de obicei 128 sau 256 întreruperi)
- driverul pentru PIC va redirecționa întreruperile hardware către una din aceste întreruperi logice (prin intermediul funcţiei request_irq())
- BSP va realiza interfațarea între întreruperi hard și soft prin structurile
  - hw_interrupt_type (pointeri la funcțiile de tratare hardware)
  - irq_desc_t (pointeri la handlere logice)

Funcțiile BSP pentru tratarea întreruperilor

- initialization: inițializarea întreruperii
- startup function: funcția apelată la cererea de întrerupere, activează întreruperile
- shutdown: dezactivează întreruperile
- enable: activează o întrerupere particulară
- disable: dezactivează o întrerupere particulară
- acknowledgement: apelat la începutul rutinei de tratare (dezactivează întreruperea)
- end of interrupt: marchează sfârșitul unei întreruperi (reactiveaza întreruperea)

Magistrale

- Linux având la bază 386 așteaptă că un dispozitiv extern (BIOS) să autodetecteze și să mapeze toate dispozitivele hardware în memorie (în speță cele de pe magistrala PCI)
- în cazul sistemelor încorporate, BSP trebuie să preia și acest rol
- în general BSP va emula o interfață PCI pentru portare ușoară și va conține rutine de inițializare pentru maparea în acest sens a dispozitivelor

Alte dispozitive importante tratate de BSP

- timere
  - Programmable Interval Timer: un timer este tot timpul configurat în faza de pornire pentru a fi folosit ca tick-ul sistem
  - pentru acest timer BSP trebuie să ofere o interfață
- UART
  - folosit cel mai des pe post de consolă
  - kernelul așteaptă întotdeauna prezența unei console prin care să se interfațeze cu utilizatorul (serială, ethernet, etc...)