MŰEGYETEM 1782

Linux alapú beágyazott rendszerek

Rövid közvéleménykutatás

Operációs rendszerek

Emlékeztető

- Informatika 1 / Operációs rendszerek tárgyban részletesebben hallhattatok róla
- Miért is felel egy operációs rendszer?
  - Erőforráskezelés
    - Fájl, memória
  - Hardverelérés biztosítása
  - Felhasználói programok felé kezelő interfész biztosítása

Operációs rendszer rétegei

Userspace

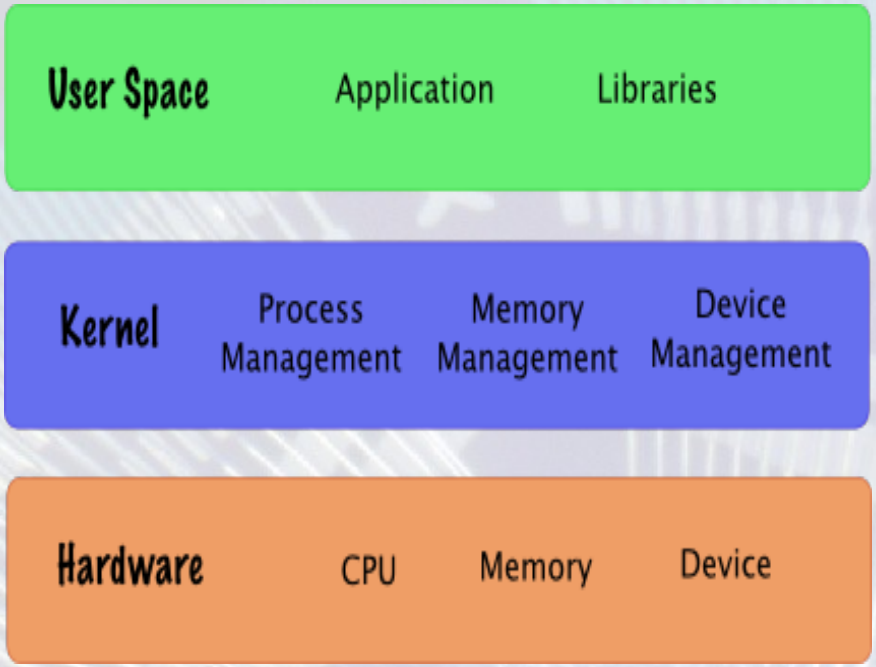
– Az összes felhasználói program

Kernel

– Hardverhozzáférés, absztrakciós réteg nyújtása

Hardver

– Amin fut a rendszer, nem igazán az OS része, viszont különböző szolgálatásokat nyújtania kell



Hardver réteg

Milyen támogatás szükséges hardver oldalról?

- Szigorúan véve semmilyen, de a modern operációs rendszerek megkövetelik a következőket:
  - Virtuális memóriakezelés – pl. x86 laptábla
    - Logikai és fizikai cím
  - Memóriavédelem – pl. x86 descriptorok
    - Ne férhessen bárki bármihez
  - Utasításvédelem – privilegizált utasítások
  - Felderíthetőség
    - ez mi lehet?

Hardver réteg

Felderíthetőség

- Az operációs rendszer feladata az egységes hardverkezelés biztosítása
  - Ne a userspace fejlesztőnek kelljen a hardverspecifikus dolgokkal törődnie
- Ehhez szükséges, hogy tudjuk, milyen a hardverkiépítése a rendszernek
  - Pl. PCI busznál minden eszköz rendelkezik egy kiolvasható ID-vel, ez egyértelműen azonosít

Kernel réteg

Milyen támogatásokat nyújt a kernel réteg?

- Hardver absztrakció – illesztőprogram (driver)
  - Azonos funkcionalitású, de különböző hardvereket egységesen lehessen kezelni
- Erőforráskezelés
  - Memória, merevlemez, processzor
- Processzkezelés
  - Felhasználói programok ütemezése
- Programozói felület biztosítása user space felé

Userspace réteg

Milyen támogatásokat nyújt a userspace réteg?

- Ez a legegyszerűbb: amit akarunk.
  - Pl. GUI, segédprogramok
  - A „megszokott” C-C++ alkalmazásfejlesztés
- C programkönyvtár
  - Kernel által nyújtott programozói felületet teszi elérhetővé
  - Mit csinál pl. a printf() függvény?
    - A kernel megfelelő függvényeit (pl. konzolra írás) fogja meghívni
      - » Ha jó a hardver absztrakció, lényegtelen, hogy az egy soros port, videokártya vagy valami más.

A továbbiakban ezen alapok Linux-specifikus oldalát nézzük meg.

Linux operációs rendszer

Mit nevezünk Linuxnak?

- Elsősorban egy operációs rendszer kernelt
  - Mely ingyenes, nyílt forrású
  - Rendkívül sok platformon elérhető
    - Többek közt a soft core FPGA processzorokon is (ezért foglalkozunk vele ebben a tárgyban :-)): MicroBlaze, NIOS II, OpenRISC
  - Ennek következtében bárki belenyúlhat, ha van egy ötlete, amit kipróbálna, könnyen csinálhat prototípust
    - Cserébe viszont nem olyan egységes, mint pl. egy Windows, ahány telepítés, annyiféle lehet...

Linux operációs rendszer

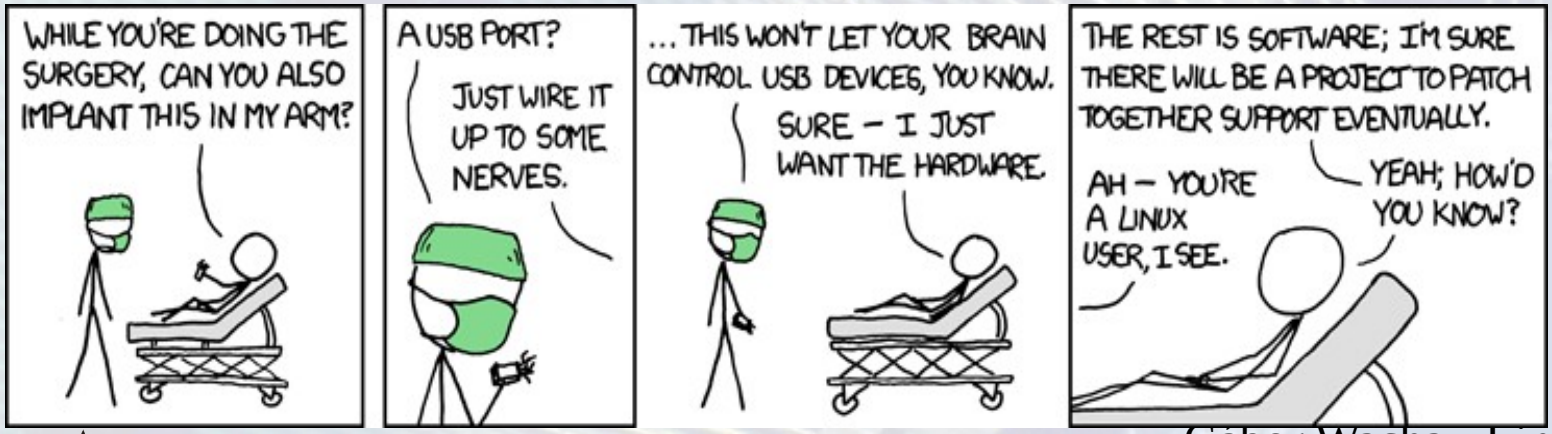
Mit nevezünk Linuxnak?

- Gyakran a hozzátartozó userspace alkalmazásokat is
  - Mivel ezek majdnem mindig ugyanazok
    - A „majdnem”-et az Android adja, ami szintén Linux kernelen alapszik
- Az „összerakott” kernel és userspace alkalmazásokból álló rendszert is Linuxnak szokás hívni
  - Persze kivéve az Androidot, mert azt Androidnak hívjuk
  - Egy feltelepíthető, összecsomagolt rendszert pedig disztribúciónak

Linux operációs rendszer

A Linux előnyei

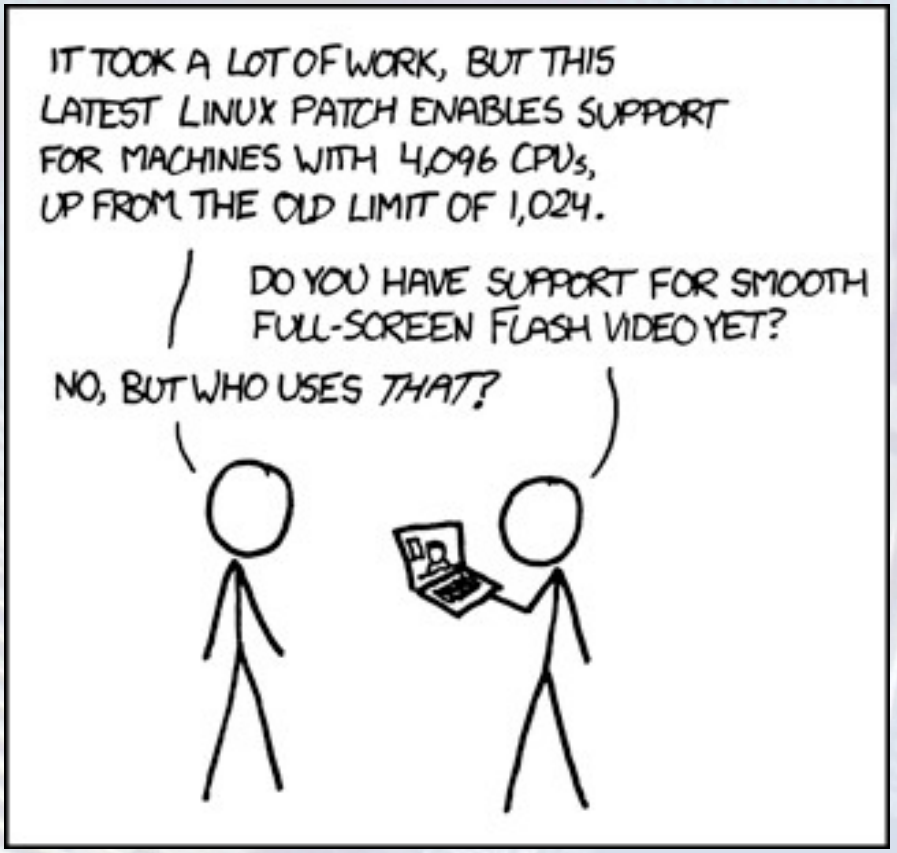
- Kiváló „kísérleti rendszer”
  - Aránylag jól dokumentált
  - Elég sok rendszeren fut
  - Elég könnyen lehet módosítani
  - Így aztán ha valami új technológiát talál ki valaki (pl. új processzütemezőt), van egy alkalmas platformja hozzá



Linux operációs rendszer

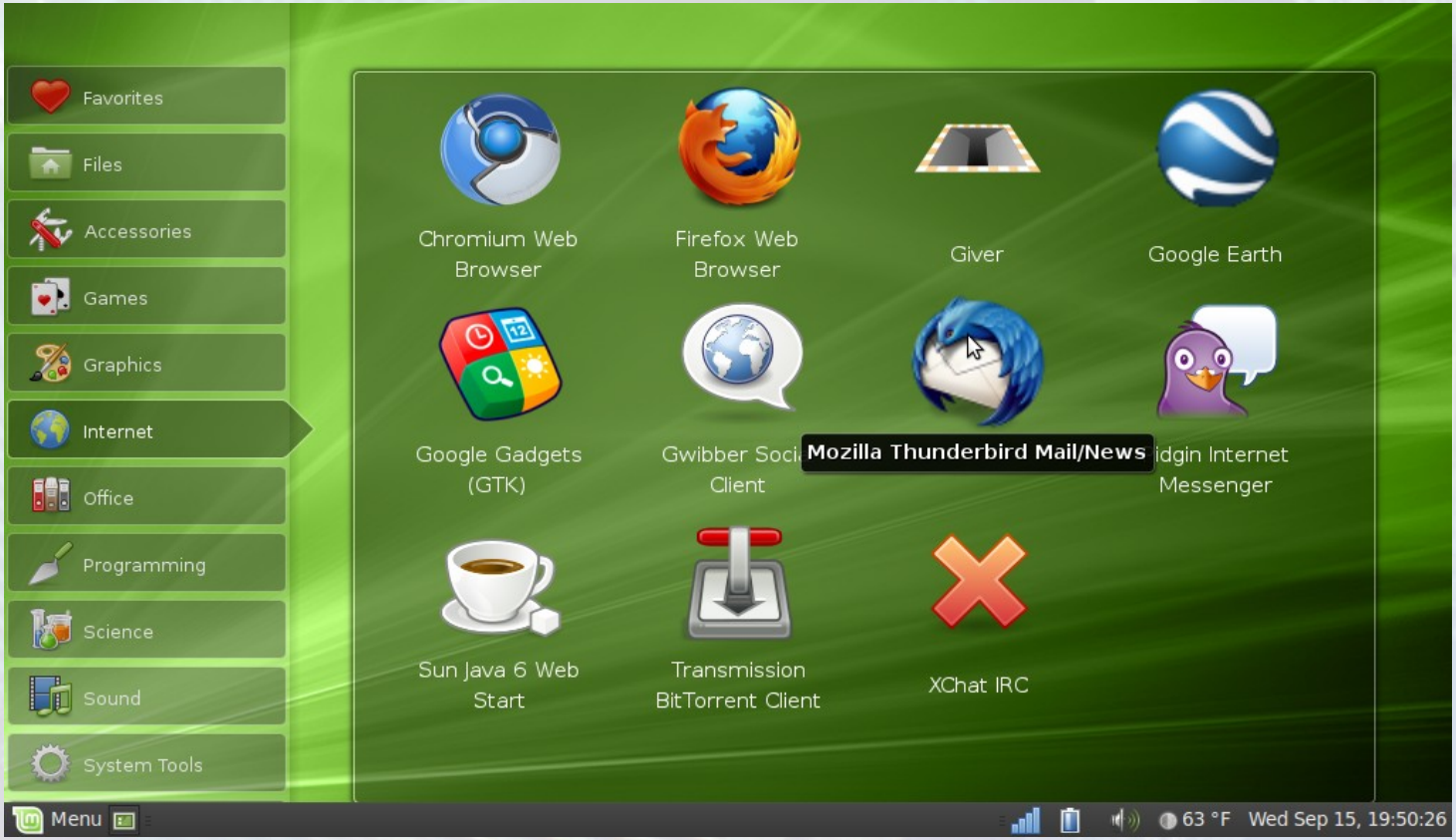
A Linux hátrányai

- Rengeteg, külön dolgozó fejlesztő
  - Nem egységes
  - Sokszor egymás ellen dolgoznak, vagy újra feltalálják a kereket
- Túl gyorsan változik, folyamatosan frissíteni kell a tudást
- Valamint a jobb oldali kép magáért beszél...



Linux operációs rendszer

Desktopon



Linux operációs rendszer

Telefonon

Linux operációs rendszer

Beágyazott alkalmazás

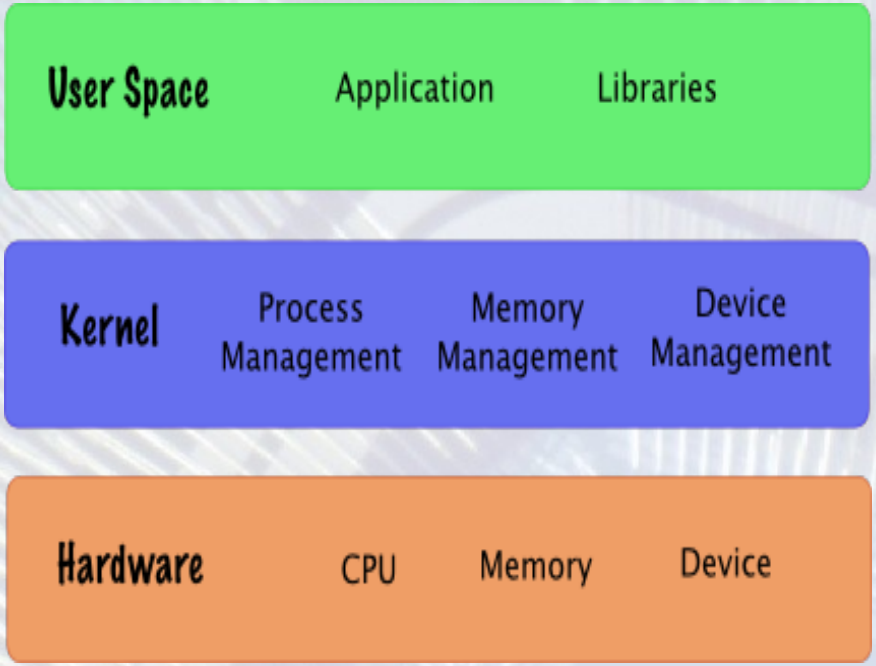
```
[    2.284263] mmc0: new SD card on SPI
[    2.317537] Freeing unused kernel memory: 5348K (c04b5000 - c09ee000)
[    2.327598] mmcblk0: mmc0:0000 SA02G 1.83 GiB
[    2.380671]  mmcblk0: p1 p2
Starting logging: OK
Initializing random number generator... done.
Starting network...
udhcpc (v1.21.1) started
grep: /etc/resolv.conf: No such file or directory
Sending discover...
Sending discover...
Sending select for 192.168.2.101...
Lease of 192.168.2.101 obtained, lease time 7200
deleting routers
route: SIOCDELRT: No such process
adding dns 192.168.2.1
Starting ProFTPD: [   12.230825] warning: `proftpd' uses 32-bit capabilities (legacy support in use)
done

Linux@Atlys (C) wachag -- IP address: 192.168.2.101

logsys login:
```

MicroBlaze alapú Linux rendszerek

Emlékeztető

Linux operációs rendszer rétegei

- User space: saját programjaink
- Kernel: Linux kernel
- Hardver: Xilinx MicroBlaze processzoros rendszer

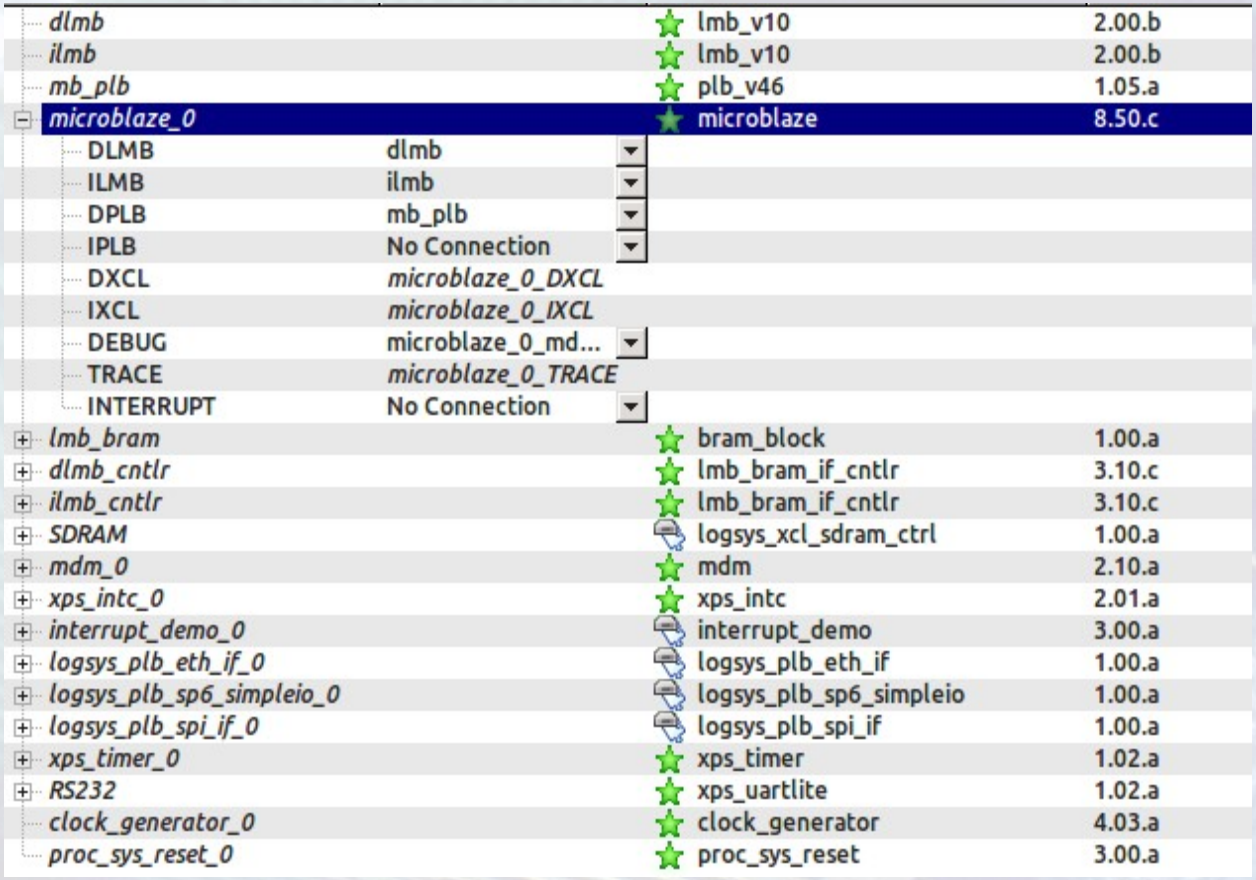


Hardver réteg

Minimális Linux alapú EDK rendszer

- MicroBlaze processzor
  - MMU-val: Memory Management Unit – virtuális memóriakezelés, memóriavédelem
- Külső memória
  - Blokk RAM-ban nem férünk el :-(
- Soros port
  - Hogy lássunk is valamit...
- Megszakításkezelő
  - A legtöbb eszközt megszakításos módon kezeli a Linux
- Időzítő
  - A processz ütemezéshez kell

Hardver réteg



Hardver réteg

Felderíthetőség

- A PLB és az AXI buszok nem támogatják a buszra illesztett eszközök detektálását
  - Viszont menet közben nem kerül rá új eszköz, az egész buszrendszer statikus
- A buszrendszert és az illesztett eszközöket a kernel réteg számára egy leíró fájl – az eszközfa (*device tree*) – fogja mutatni

Hardver réteg

Eszközfa

- Célja hasonló, mint az MHS fájlé
  - És ez is generálható automatikusan :-)
- Felsorolja, milyen buszra milyen eszközök csatlakoznak
  - Ezt belefordítjuk majd a Linux kernelbe – lehet tudni, hogy melyik memóriacímen mi érhető el

  (hasonló az *xparameters.h* funkciójához)

```
logsys_plb_eth_if_0: ethernet@84000000 {
»       compatible = "xlnx,logsys-plb-eth-if-1.00.a";
»       device_type = "network";
»       interrupt-parent = <&xps_intc_0>;
»       interrupts = <3 2>;
»       reg = <0x84000000 0x10000>;
»       xlnx,include-dphase-timer = <0x0>;
} ;
logsys_plb_sp6_simpleio_0: sp6-simpleio@83000000 {
»       compatible = "xlnx,logsys-plb-sp6-simpleio-1.00.a";
»       interrupt-parent = <&xps_intc_0>;
»       interrupts = <6 2>;
»       reg = <0x83000000 0x10000>;
»       xlnx,gpio-enable = <0x1>;
»       xlnx,gpio-width = <0xd>;
»       xlnx,include-dphase-timer = <0x1>;
} ;
logsys_plb_spi_if_0: spi@85000000 {
»       #address-cells = <1>;
»       #size-cells = <0>;
»       clock-frequency = <60000000>;
»       compatible = "xlnx,logsys-plb-spi-if-1.00.a";
»       interrupt-parent = <&xps_intc_0>;
```

Hardver réteg

Eszközfa

- Fa-struktúra
- Csomópontok és csomópontok tulajdonságai
  - Lényeges tulajdonságok:
    - *compatible*: milyen drivert rendeljen hozzá a kernel
    - *reg*: milyen memóriacímen érhető el a buszon
    - *interrupts*: hányadik megszakítási vonalon van

Kernel réteg

Rendszerbetöltő

- A lefordított Linux kernel ugyanolyan ELF futtatható, mint amit az SDK-val lehet csinálni
  - Csak **sokkal** nagyobb méretű (~10 Mbyte)
- Blokk RAM-ba nem fér bele → külső memória kell
- Az XMD (vagy az SDK) JTAG-on keresztüli letöltése nagyon lassú (~ tíz perc)
  - Érdemes máshonnan betölteni → **rendszerbetöltő** használata

Kernel réteg

Rendszerbetöltő

- A rendszerbetöltő valamilyen külső forrásról (SD kártya, hálózat, flash) a fő memóriába (pl. SDRAM) másolja a Linux futtathatót fájlt, majd futtatja
  - Innentől kezdve a kernel fut
  - Pl. Logsys SP6: SD kártya bootloader + Linux
- Két szintű rendszerbetöltés is lehetséges:
  - Blokk RAM-ból fut egy első szintű betöltő, mely a FLASH memóriából betölt egy szabványosabb, komolyabb betöltőt, amely aztán hálózatról letölti a Linux futtathatót
  - Pl. Xilinx Zynq: FSBL + uboot

Kernel réteg

A Linux kernel feladatai

- Eszközök felismerése és hozzájuk meghajtóprogram rendelése
  - MicroBlaze PLB/AXI rendszeren device tree leírásból
- Meghajtóprogramok segítségével hardverabsztrakció nyújtása
- Memóriakezelés
- Alkalmazások ütemezése
- Betöltés után vezérlés átadása a userspace alkalmazásoknak

Userspace réteg

Minimális userspace réteg

- A userspace réteg tartalmazza a saját alkalmazásainkat
  - Tehát kell pl. a C programkönyvtár
- Valahogyan el kell tudni érni a hardver platform eszközeit
  - **Userspace és kernel között valamilyen kommunikációs interfész szükséges – ez lesz a driver feladata**
- Segédalkalmazások – fájl másolás, mozgatás, stb.
- Szükséges egy „első processz”, amely a kernel indítása után inicializálja a userspace szolgáltatásokat (pl. mérésadatgyűjtőnél webszervert indít)

Userspace réteg

init – az első processz

- A kernel betöltés után ezt a felhasználói programot fogja elindítani
- Ez már userspace alkalmazás (~olyan, mint amilyet amúgy írni szokott az ember C-ben)
- Feladata, hogy elindítsa az összes általunk igényelt szolgáltatást
  - FTP szerver, távoli hozzáférés, saját alkalmazásunk
- Legvégül egy login promptot ad vissza

Userspace réteg

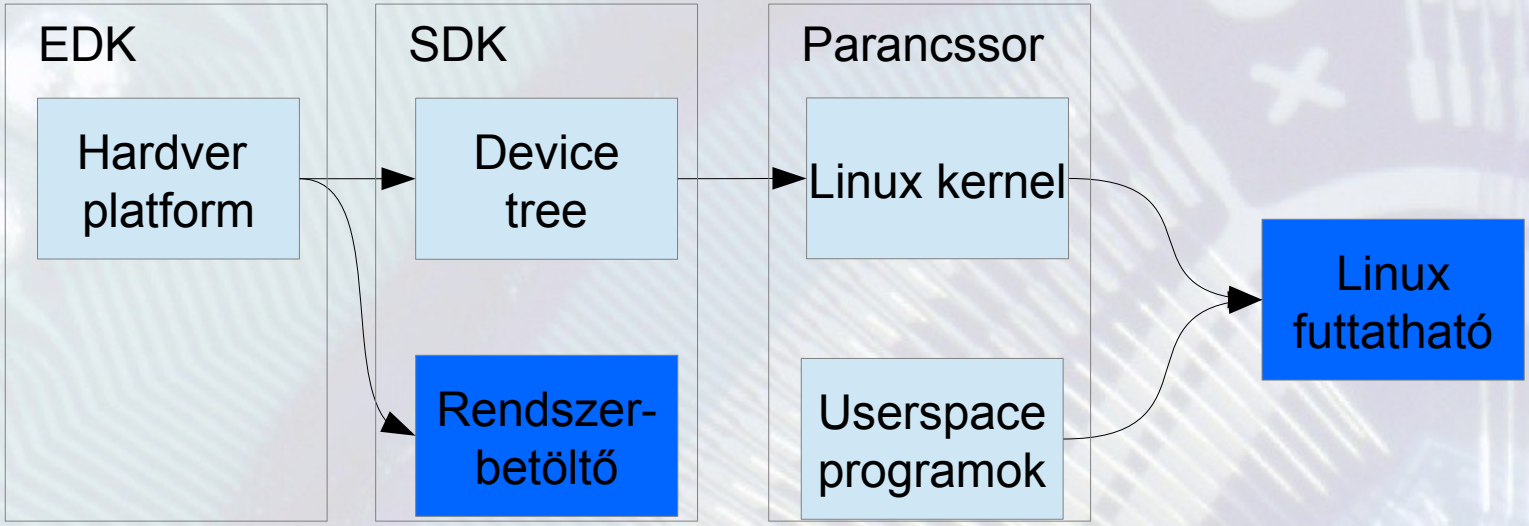
Userspace réteg

- A felhasználói alkalmazásokat, segédprogramokat, programkönyvtárakat egy ún. *gyökér fájlrendszerbe* (root filesystem) gyűjtik
- MicroBlaze alapú Linux rendszer esetén
  - a kernel
  - a device tree
  - és a gyökér fájlrendszer

  egy darab futtatható binárisba lesz összefordítva.

MicroBlaze alapú Linux rendszer

Linux rendszer létrehozásának lépései



Ezekről a lépésekről későbbiekben (következő előadáson) részletesen lesz szó.

Linux boot folyamat

Az indulás lépései MicroBlaze rendszer alatt

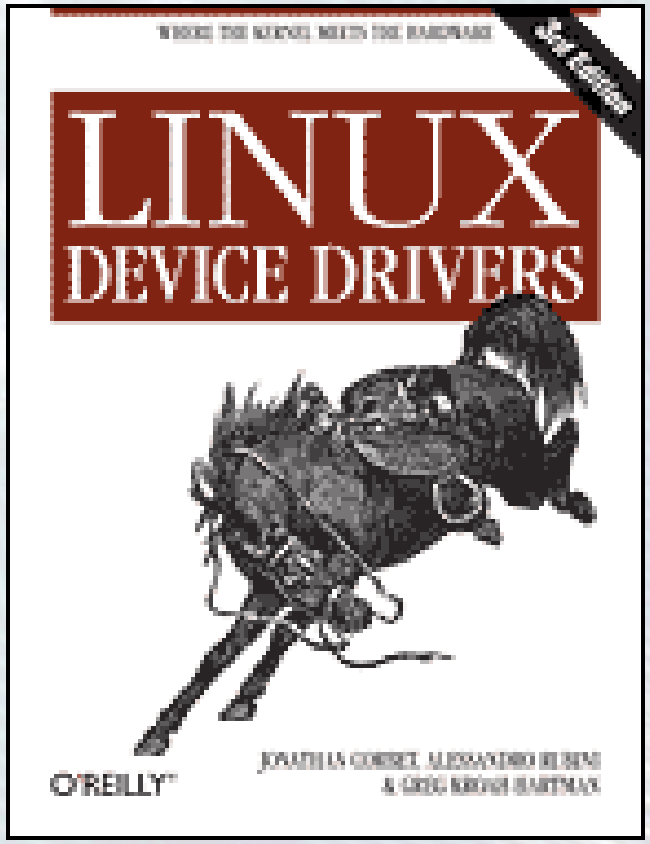
1. Blokk RAM-ból betöltött alkalmazás beolvassa a Linux kernelt
   SD kártyáról, hálózatról, flash memóriából vagy egyéb helyről
2. Linux kernel elindul, inicializálja a hardvert
3. A Linux kernel átadja a vezérlést az első userspace alkalmazásnak, az *init*-nek
4. Az init inicializálja a userspace környezetet (pl. FTP szervert indít, stb.)
5. Az init elindítja az általunk megadott alkalmazást vagy a login promptot
6. Miénk a vezérlés – saját alkalmazás vagy interaktív parancssor

Linux alapú perifériahozzáférés

Linux alapú rendszer létrehozásának lépései

- Hardvertervezés – a tárgy jelentős része erről szólt
  - Áramkör vagy FPGA terv
- Board bringup – következő alkalommal lesz róla szó
  - „Indításképes” Linux rendszer létrehozása
- **Driverfejlesztés**
  - **A hardver perifériáinak elérhetővé tétele a userspace számára**
- Alkalmazásfejlesztés – más tárgyak szólnak erről
  - A hardvert a driveren keresztül elérő userspace alkalmazás fejlesztése
- Vezérlő felület fejlesztése – más tárgyak szólnak erről
  - Magas szintű alkalmazás

Ajánlott irodalom



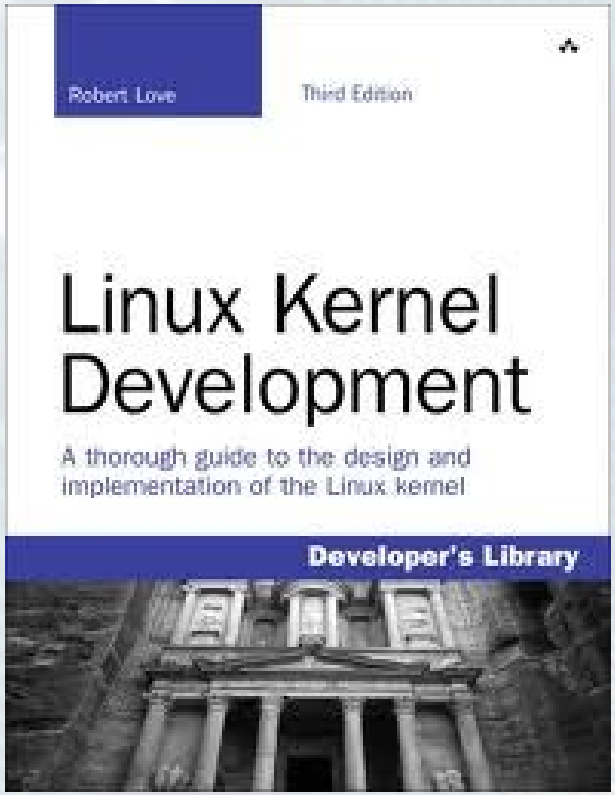
***Linux Device Drivers*,**

Jonathan Corbet, Alessandro Rubini, és Greg Kroah-Hartman

http://lwn.net/Kernel/LDD3/

- 2.6.10-es kernelhez, ma már 3.14-nél tartunk...
- Rengeteg a „zavaros” rész
- Az alapok ma is igazak, de rengeteg az API változás

Ajánlott irodalom



***Linux Kernel Development*,**

Robert Love

- 2010-es kiadás
- 2.6.32 körüli kernelek
- Nem ingyenes
- http://www.amazon.com/Linux-Kernel-Development-3rd-Edition/dp/0672329468

Ajánlott irodalom

LWN kernel cikkei

- http://lwn.net/Kernel/
- Kernel API dokumentációk (általában elavult)
- Bejelentések
- Kernel belső működés

LKML lista

- https://lkml.org/

Kernel newbies

- http://kernelnewbies.org/

Linux forrás dokumentációja

- *Documentation/* könyvtár
  - Gyakran elavult, töredékes
- Adott header fájl
  - Példa: *linux/mutex.h*
- Példaprogramok
  - Netes keresés
  - *samples/* könyvtár

Linux driverfejlesztés

A Linux kernel – www.kernel.org

- ~ 600 Mbyte C forrás
- Ennek jelentős része driver
- Több részből áll:
  - Processzkezelés
  - Memóriakezelés
  - Filerendszerek
  - **Eszközkezelés ← ezzel foglalkozunk majd**
  - Hálózat
- Moduláris felépítés – l. később

Linux driverfejlesztés

Kernel modul

- Az egyes meghajtóprogramok (driverek) menet közben betölthetőek, eltávolíthatóak
  - Csak azt töltjük be, ami szükséges
  - Nem *main()* függvényünk lesz, hanem inicializáló (~konstruktor) és lebontó (~destruktor)
- Egy modul erősen párhuzamos és eseményvezérelt. Példák:
  - Több alkalmazás hozzáférhet user spaceből – párhuzamos
  - Megszakítás lekezelése – eseményvezérelt

Példa modul

```
#include
<linux/module.h>
int onload(void)
{ return 0;}
void onunload(void)
{}
module_init(onload);
module_exit(onunload
);
```

←szükséges header

←belépési pont, tetszőleges névvel!

← kilépési pont, tetszőleges névvel!

← jelezzük, mi a belépés

← jelezzük, mi a kilépés

Hardverhozzáférés

Hardverhozzáférés

- Ritka (gyakorlatilag nincs) az olyan modul, amely nem nyúl hardverhez
- Hardverhozzáférés új problémákat hoz be:
  - Tönkre lehet tenni a hardvert (l. régi VGA CRTC regiszterei)
  - Le lehet blokkolni a rendszert
  - Csökkenteni lehet a rendszer biztonságát (pl. DMA támadás)

Hardverhozzáférés

IO portok

- A periféria regiszterei külön *in* és *out* CPU utasításokkal érhetőek el (tipikusan x86 architektúra, de pl. az általatok tanult 8085 is), akár másfajta buszcikluson keresztüli elérést is jelenthet.

Memóriára leképezett IO

- A periféria területe adott memóriatartományra van leképezve, memóriaműveletekkel érhető el. Tipikus a beágyazott rendszereknél, illetve PCI eszközöknél.

Hardverhozzáférés

Konkurencia

- Párhuzamosan több modul futhat
  - Egyszerre több modul próbálhatja elérni ugyanazt a hardvert
    - Ez nem feltétlen kívánatos
- Egy modult párhuzamosan több alkalmazás használhat
  - Ugyanaz a modul adott időpontban párhuzamosan többször próbálhatja elérni ugyanazt a hardvert
    - Ez sem szerencsés mindig

Hardverhozzáférés

Konkurens viselkedés feloldása

- Párhuzamosan több modul hardverelérése:
  - Felhasználás előtt az elérni kívánt IO memóriarégió lefoglalása – ez kötelező
- Ugyanannak a modulnak párhuzamos elérése:
  - Szemafor, mutex – ez ránk van bízva
    - Nem fogunk vele foglalkozni

Hardverhozzáférés

Tipikus működés

- Modul inicializáló függvény lefoglalja az IO memória régiót
- Modul használja
- Modul lebontó függvény felszabadítja az IO régiót

Hardverhozzáférés

```
void * cim;
int onload(void) {
   struct resource * port =
request_mem_region(0x73000000,16,"sajatmodulom
”);
   cim=ioremap(0x73000000,16);
   // lefoglalás megtörtént, használhatom
   iowrite8(0x55,cim);
}
… // unload-kor meg felszabadítom
```

Hardverhozzáférés

Hardverbáziscím lekérdezése

- Egyszerű, op. rendszer nélküli rendszereknél *xparameters.h*
- Viszont a Linux kernel hordozható (szeretne lenni)
- Nem is feltétlenül létezik az a hardveregység, amit a mi driverünk kezelne
  - Szükséges a hardver réteg felismerhetősége

Hardverhozzáférés

Felderíthetőség a Linux kernelben

- Szeretnénk, ha az eszközkezelőnk csak akkor töltődne be (vagy legalábbis értesülne róla), ha megjelenik egy általa támogatott eszköz a buszon.
  - Pl. Logsys USB kábel: csak akkor töltődjön be a hozzá tartozó driver, ha csatlakoztattuk
- Ehhez a hardver felderíthetősége szükséges
  - PCI, USB, stb. oldalon ez hardveresen megvalósul, de pl. a PLB és AXI nem támogatja
    - Emlékeztető: erre jó a device tree (eszközfa)

Platform eszközvezérlők

Platform eszközvezérlők

- Olyan eszközvezérlők, melyek a *device tree* alapján találják meg, melyik eszközt vezérlik
  - Emlékeztető – egy eszköz bejegyzés:

```
logsys_plb_sp6_simpleio_0: sp6-simpleio@83000000 {
compatible = "xlnx,logsys-plb-sp6-simpleio-
1.00.a";
reg = <0x83000000 0x10000>;
} ;
```

compatible: ez alapján fogunk eszközvezérlőt hozzárendelni

reg: ez az az IO terület, amit el akarunk érni

Platform eszközvezérlők

Anatómiája

- Megadjuk a kernel számára, milyen *compatible* tulajdonságú eszközt szeretnénk támogatni
- Megadjuk, melyik függvény meghívását szeretnénk amikor egy ilyen eszközt talál a rendszer (*probe*) és melyiket, mikor az eszközt eltávolítják (*remove*).
  - Mindjárt látunk is rá példát
- A modul inicializáló és lebontó függvényeket kiváltja!

Platform eszközvezérlők

```
static struct of_device_id pelda_match[] = {
 {.compatible = ”xlnx,logsys-gpio”, },
/* lista vége */ };

static struct platform_driver gpio_driver = {
   .driver = {
      .name = ”gpio_vezerlo”,
      .owner = THIS_MODULE,
      .of_match_table = pelda_match,
   },
   .probe = gpio_probe,
   .remove = gpio_remove,
};
module_platform_driver(gpio_driver);
```

Platform eszközvezérlők

***probe()* és *remove()* függvények**

- Hasonlóan képzelhető el, mint a modul inicializálás és lebontás, csak ezek olyankor hívódnak meg, amikor a kernel kompatibilis eszközt talál/kapcsol le.
- Megkapjuk az eszköz összes paraméterét
  - Például a *reg* bejegyzés értékét – le tudjuk foglalni a hozzá tartozó memóriatartományt

Platform eszközvezérlők

```
static int gpio_probe(struct platform_device *of_dev)
{
  struct resource * res;
  res = platform_get_resource(of_dev, IORESOURCE_MEM,
0);
  base_addr = devm_ioremap_resource(&of_dev->dev,res);
  // az így lekérdezett memóriaterület automatikusan
el lesz engedve
  iowrite8(0x55, base_addr);
  return 0;
}
```

Platform eszközvezérlők

```
static void gpio_remove(struct
platform_device * of_dev){
  /* itt végezhetünk minden
felszabadítást, elengedést, stb. */
}
```

Kernel és userspace közötti kommunikáció

Eddig elhangzott

- Egyszerű kernel modul
  - Inicializálás és lebontás
  - Hardverelérés „beégetett” konstansokkal
- Platform eszközkezelő
  - „Kompatibilis” hardver kezelése
  - Báziscím lekérdezése

- Viszont felhasználói programból nem tudunk adatot átadni a kernel modulnak!

Kernel és userspace közötti kommunikáció

Bevezetés

- Linux alatt a legtöbb kommunikáció fájlon keresztül történik
  - Fájl: kb. bármi, amin a következő műveletek értelmezhetőek:
    - Megnyitás
    - Írás
    - Olvasás
    - Bezárás
  - Hogy mi történik egy fájlba íráskor, az a kernel dolga
    - Lehet, hogy ténylegesen merevlemezre írás történik („klasszikusan” ez a fájl)
    - De lehet, hogy egész más lesz

Kernel és userspace közötti kommunikáció

Bevezetés

- A kernel – userspace kommunikáció is fájlon keresztül fog történni
  - A driverfejlesztőnek csak azt kell megmondani, hogy mi történjen akkor, amikor megnyitják-bezárják-írják-olvassák az adott fájlt
    - Mert végül is ez is csak adatátvitel
  - Többféle módszer létezik, ezekből csak egyet fogunk megnézni és annak is az egyszerűbb módját

Karakteres eszközök

Legegyszerűbb típusú eszközmeghajtók

- Az eszköz egy példányához egy fájlrendszer bejegyzés tartozik – általában a */dev* könyvtár alatt érhetőek el
- Byteos hozzáférés
  - user space oldalról teljesen olyan, mintha „klasszikus” fájlt látnánk
  - Kernel oldalról pedig azt látjuk, hogy írtak bele/olvastak belőle, és a mennyiséget, hogy mennyi adatot

Karakteres eszközök

Használatuk kernel oldalról – egyszerű verzió

- Megírjuk azokat a fájlműveleteket, amelyeket támogatni akarunk: írás, olvasás
- Megmondjuk, milyen néven szeretnénk, hogy létrejöjjön a */dev* könyvtár alatt a bejegyzés
- Beregisztráljuk a kernelnek

Karakteres eszközök

Írás és olvasás művelet megvalósítása

- Lényegében kapunk egy userspace címtartományba tartozó puffert, abba kell írni vagy abból olvasni
  - Userspace memóriára nem szabad közvetlenül hivatkozni, mert:
    - Nem biztos, hogy az a memóriacím éppen él (pl. másik memóriabankban van)
    - Ha létezik is, a userspace memória lapozott, előfordulhat, hogy a lap nincs bent → page fault, mely kernel kódban nem engedélyezett.
    - Felhasználó által adott mutatóban – biztonsági okokból – nem bízunk vakon.

Egyszerű karakteres eszköz

```
char msg[100] = "Helló\n";
static ssize_t mdev_read(struct file * file, char *
buf, size_t count,
loff_t *ppos) {
   copy_to_user(buf, msg, min(6, count));
   return min(count, 6);
}
struct file_operations mops = { .owner =
THIS_MODULE, .read = mdev_read,};
struct miscdevice mdev = { .name = "misctest", .minor =
MISC_DYNAMIC_MINOR,
.fops = &mops };
int misctest_init(void) { return  misc_register(&mdev);
}
void misctest_exit(void) { misc_deregister(&mdev); }
```

A következő óráról

Hogyan hozhatunk létre mi magunk Linuxot futtató rendszert?

- Szükséges hardver platform
- Eszközfa leírás generálása
- Userspace alkalmazások generálása
- Linux kernel konfigurálása és fordítása

A laborgyakorlatról

- Egyszerű, „Helló, világ” kernel modul létrehozása
- „Helló, LED” létrehozása – hardverhozzáférés „beleégetett” címekkel
- „Knight Rider” – futófény LED-eken, kernelmodulként
- Átalakítás platform eszközzé
- Egyszerű karakteres eszköz megvalósításával userspace-ből állítható GPIO működés.