# Operációs rendszerek

## *Input/Output*

## *1.1*

Pere László (pipas@linux.pte.hu)

PÉCSI TUDOMÁNYEGYETEM TERMÉSZETTUDOMÁNYI KAR

INFORMATIKA ÉS ÁLTALÁNOS TECHNIKA TANSZÉK

# Blokkos és karakteres I/O

A perifériák két nagy csoportba sorolhatóak az adatátvitel szempontjából:

- Blokkos eszközök: az adatok mozgatása adatblokkok szerint történik, az adatblokkok egymástól függetlenül, tetszőleges sorrendben olvashatóak és írhatóak.
- Karakteres eszközök: karaktersorozatokat (streams) küldenek vagy fogadnak blokkszerkezet nélkül. Nincs címzési, keresési szolgáltatás.

A felosztás nem tökéletes, vannak eszközök, amelyek nem illeszkednek egyik modellbe sem.

# Sebesség

A perifériák sebessége igen tág határok közt változhat:

| | |
|---|---|
| Billentyűzet | $10$ byte/s |
| Modem | $7$ kB/s |
| ISDN | $16$ kB/s |
| Printer | $100$ kB/s |
| USB | $1,5$ MB/s |
| IDE disk | $5$ MB/s |
| FireWire | $50$ MB/s |
| PCI bus | $528$ MB/s |

# Sebesség

A tág határok közt mozgó adatátviteli sebesség alapvetően meghatározza azt, hogy az operációs rendszernek milyen módon kell kezelnie az egyes perifériákat.

A különféle sebességű perifériákat az operációs rendszer általában különféle prioritások bevezetésével kezeli.

A perifériák sebességét azonban nem kizárólag a sebesség határozza meg, léteznek olyan perifériák, ahol az adatvesztés megengedett és léteznek olyanok, ahol nem.

# Device controller

Az I/O eszközök általában mechanikus és elektronikus részekből állnak. Az elektronikus rész neve eszközkezelő (*device controller*) v. adapter.

A device és a controller közt általában alacsonyszintű csatoló (*low level interface*) található.

Az alacsonyszintű csatoló felépítését és működését általában szabványok írják le, hogy a számítógépet és a perifériát gyártó vállalatok megfelelően kommunikáló eszközöket hozzanak létre.

# Példák

Az alacsonyszintű csatoló általában a laikusok számára is közismert:

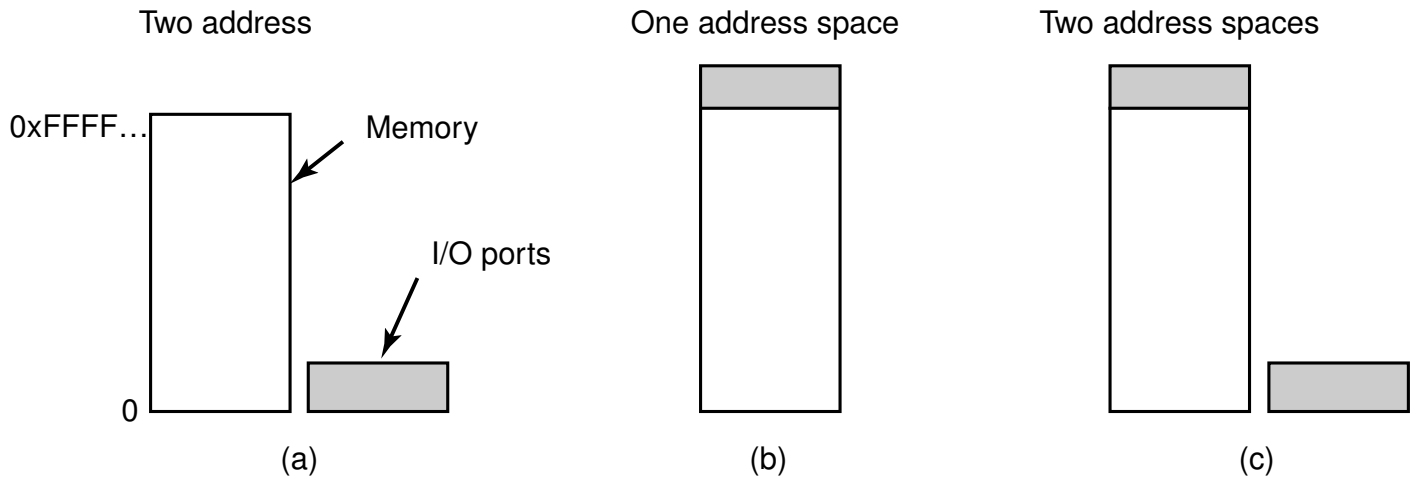
- Párhuzamos port/controller. A device lehet egy összetett lézernyomtató, de a centronics a low level interface.
- USB controller. Az eszköz amit rákapcsolunk sokféle lehet, de a low level interface USB szabvány szerinti.
- Alaplapra integrált videomeghajtó/controller. A monitor sokféle lehet, de a low level interface SVGA szabványú.

# Kapcsolattartás

Az operációs rendszer és a periféria az eszközkezelő írható illetve olvasható regiszterein keresztül kommunikálnak egymással. Az eszközkezelő regiszterei:

- Parancsregiszterek: az operációs rendszer ezek írásával adhat utasításokat az eszköznek.
- Állapotregiszterek: az operációs rendszer ezek olvasásával értesülhet az eszköz állapotáról.
- Adatbuffer: ezek írásával és olvasásával valósulhat meg a kommunikáció, az adatcsere.

# A perifériakezelés




1. ábra. (a) Memóriába ágyazott, (b)különálló és (c)hibrid I/O

# Különálló I/O

Minden regiszterhez rendelünk egy címet az I/O címterületben és azokat speciális utasításokkal olvassuk/írjuk.

A processzor külön handshake vezetéken jelzi, hogy a címbusz tartalma perifériára vonatkozik.

**Példa:**

```
IN    cx, 1000ff
OUT   120000, cx
```

# Memoriába ágyazott I/O

A memóriába ágyazott (*memory mapped*) I/O esetén minden I/O regiszterhez rendelünk egy memóriacímet, amelyeket a megszokott módon írunk, olvasunk.

Azt, hogy memória, vagy I/O utasításról van -e szó, maga a cím dönti el.

**Példa:**

```
MOV cx, ffff18
MOV ffffa9, cx
```

# Hibrid módszer

A hibrid módszer használata esetén bizonyos perifériacímeket memóriába ágyazott, más regisztereket különválasztott módon érünk el.

Általában a parancs és állapotregisztereket külön, az adatregisztereket pedig memóriába ágyazott módon érjük el.

**Példa:** IBM PC

- 640k-1M memóriába ágyazott I/O
- 0-64k I/O portok

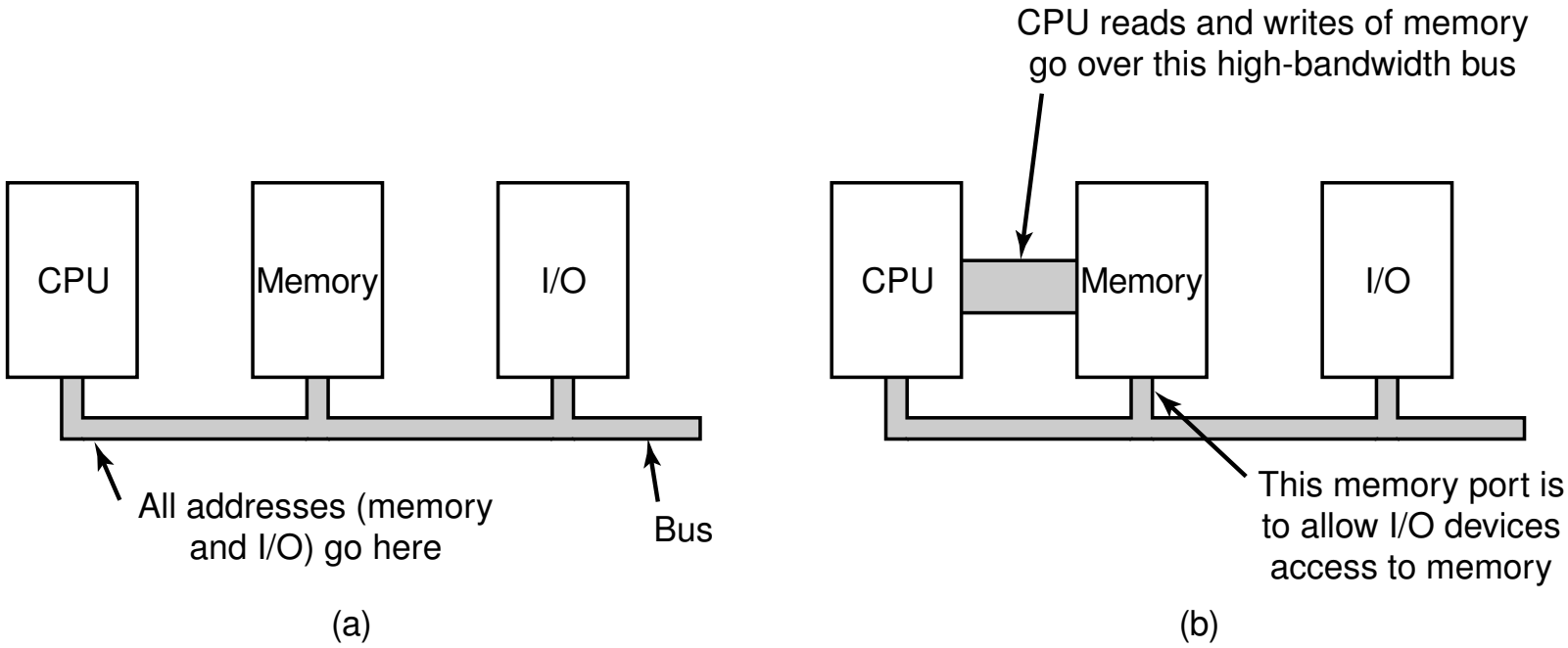
# A memóriába ágyazott I/O előnyei

- Az I/O eléréséhez nincsen szükséges speciális gépi kódú utasításokra, az eszköz regiszterei egyszerű változók, a kapcsolattartás megírható pl. C nyelven.
- Védelem könnyen megoldható, az I/O-t tartalmazó címeket egyszerű memóriavédelemmel el kell zárni az adott programok elől.
- Eszközönként szétválasztható védelem.
- A kód egységes lehet memóriában és periférián dolgozó programok esetében.

# A memóriába ágyazott I/O hátrányai

- A processzorban található cache gondot jelenthet, amit meg kell oldani. (Disable caching a page table-ben)
- A memóriának és az I/O eszközöknek a teljes címtartományt dekódolniuk kell (különben szellemképes lesz).
- Nehezen beilleszthetőek a sémába a modern architektúrák, ahol külön nagysebességű buszt építünk a memória felé.

# Külön busz




2. ábra. A processzor felé külön buszrendszert építünk

# DMA

A CPU gyors perifériák esetén nem elegendően gyors, mert sok gépi kódú utasításból áll az adatmásoló ciklus, ráadásul pazarlás a CPU-t ilyen feladatokra használni.

A közvetlen memóriahozzáférés (DMA, *direct memory access*) megvalósítására adatmozgató társprocesszort használunk.

A DMA egység adatmozgatásra készített áramkör, amely a buszokat éppen úgy képes vezérelni, mint a CPU.

A DMA áramkörök általában támogatják a ciklusloppásos és a burst módot.

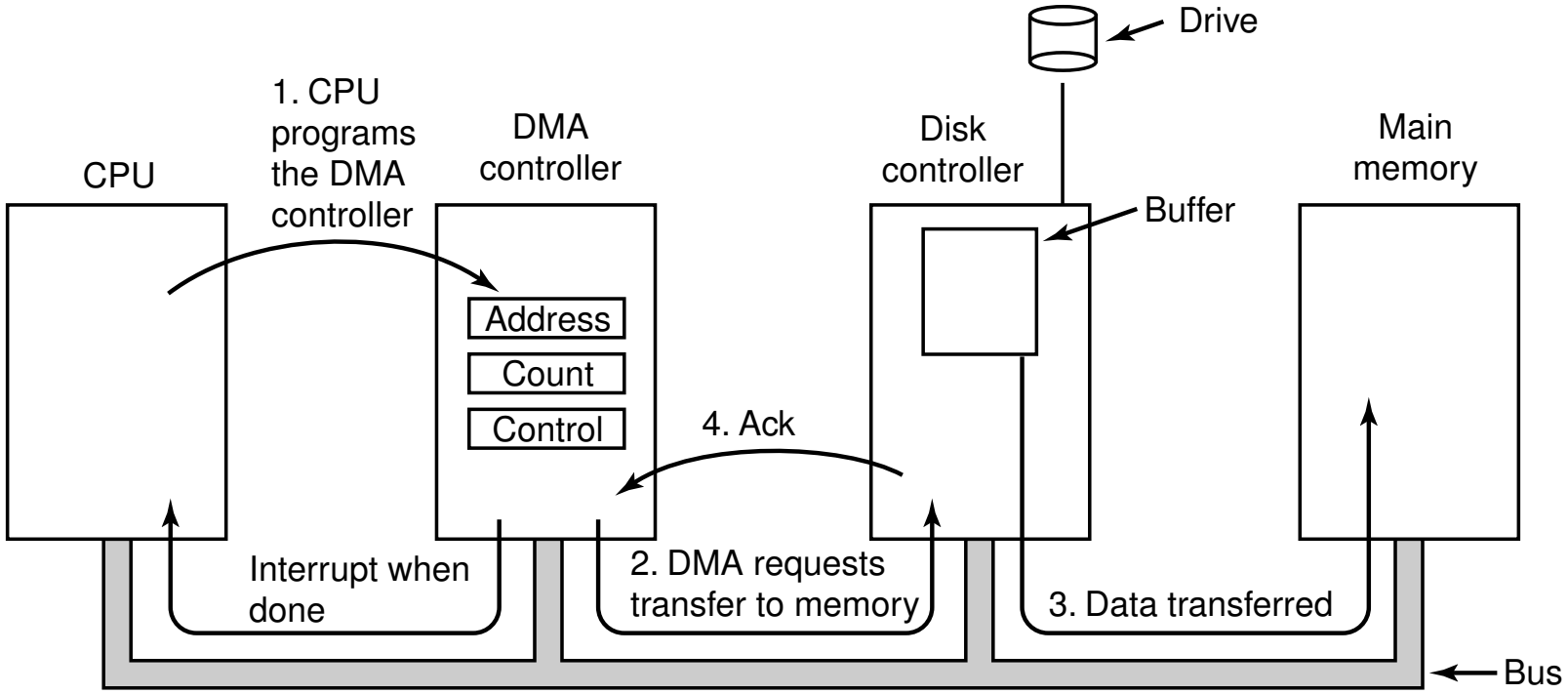
# DMA regiszterek

A DMA saját regiszterekkel rendelkezik, amelyeket a CPU perifériáként használ. A DMA működését vezérlő regiszterek akövetkezők:

1. Címregiszter: a művelet elején a mozgatandó adatblokk elejére mutat.
2. Számlálóregiszter: a mozgatandó adatblokk méretét határozza meg, folyamatosan csökken.
3. Vezérlő regiszter: I/O cím, irány stb.

Több regiszterkészlet esetén több csatornáról beszélünk.

# A DMA működése




3. ábra. A DMA regisztereinek működése

# Közvetlen/közvetett DMA

A DMA működése alapjában véve kétféleképpen szervezhető:

1. Közvetlen vezérlés: a DMA csak összeköti az I/O eszközt a memóriával. Minden adatmozgatás egy ciklus hosszú, az I/O és a memória közt történik az adatmozgatás.
2. Közvetett vezérlés: az adatmozgatás a DMA-n keresztül történik, a DMA olvas, aztán ír, minden művelet két ciklus hosszú.
   Rugalmasabb: elképzelhetők a mem->I/O, I/O->I/O, mem->mem stb. irányok is.

# DMA címzési rendszere

A DMA egység általában fizikai címeket kezel. Ez természetesen nagymértékben megnehezíti az operációs rendszer írójának feladatát.

Ha az MMU a memória és nem a processzor része (volna), a DMA képes (volna) virtuális címeket kezelni.

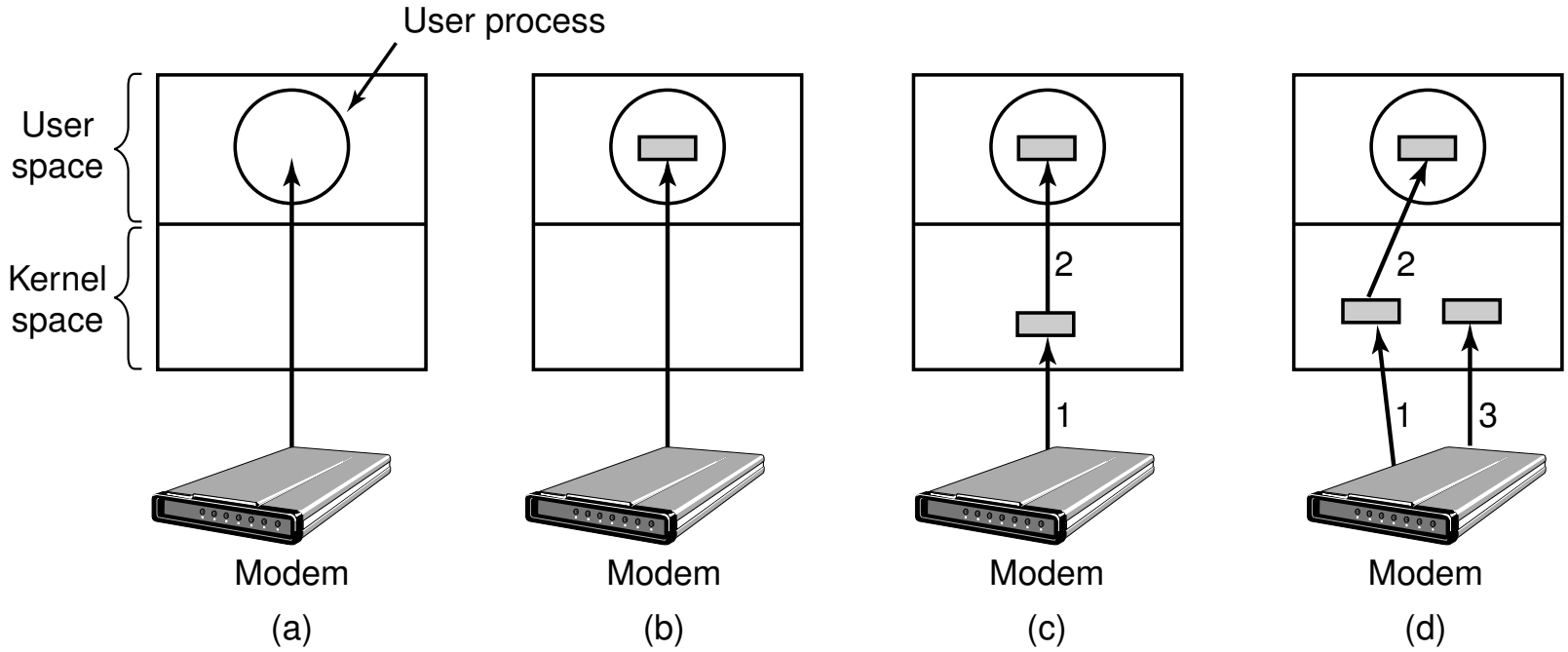
Ebben az esetben a számítógép buszrendszerén virtuális címek (volnának) találhatók, a számítógép drágább (volna).

# Bufferelés

Akár használunk DMA-t, akár nem, a device controller általában rendelkezik átmeneti tárolóval. Ennek célja kettős:

- Sokszor szükség van arra, hogy az adatok egy tömegben megjelenjenek a controllerben, hogy hibaellenőrzést végezhessünk (pl. floppy controller, hálózati csatoló CRC).
- Előfordulhat, hogy az adatok azonos ütemben érkeznek a low level interface felől, a számítógép belső busza pedig foglalt. Ez még fordítva is problémát okozhat.

# Bufferelés




4. ábra. A bufferelés módozatai

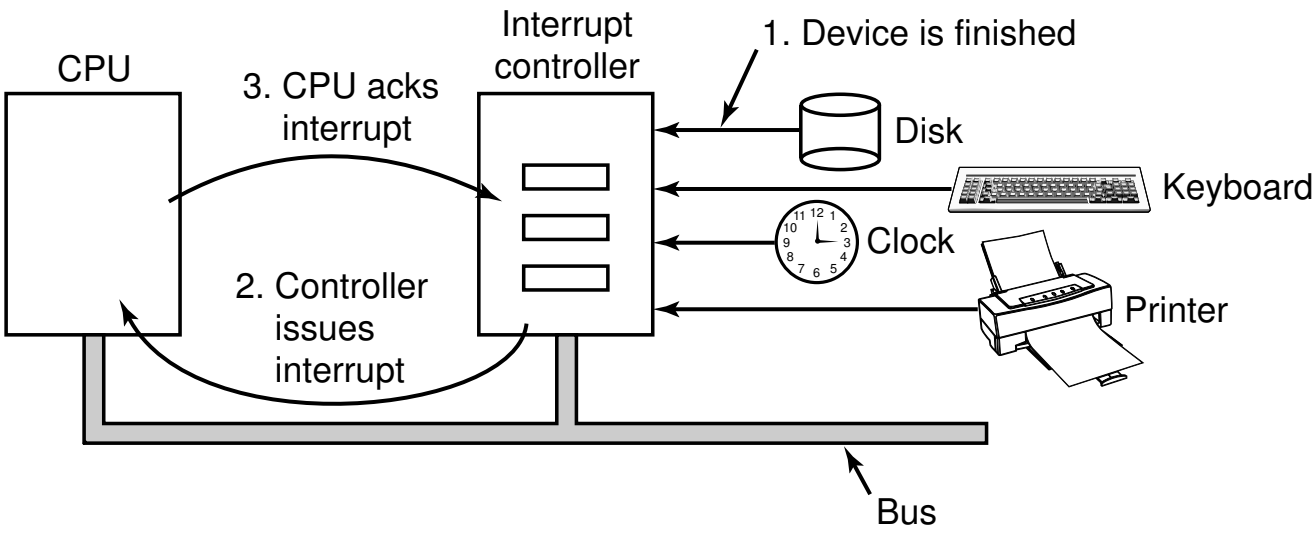
# Megszakítás

Ha az I/O eszköznek kiszolgálásra van szüksége a processzortól, a legtöbb rendszerben megszakíthatja annak pillanatnyi munkáját (IRQ, *interrupt request*) és átadhatja a vezérlést az operációs rendszer adott részére (megszakításkezelő, *interrupt handler*).

Ez a megszakításkérés és megszakításkezelés lényege.

Általában a CPU is képes megszakítást kérni önmagától (page fault, floating point exception, software irq stb.).

# Megszakításkérés




5. ábra. A megszakításos perifériakiszolgálás

# Megszakítás menete

1. Az I/O a megszakításvezetékén jelet küld a buszon.
2. Az interrupt controller a beérkezett jeleket prioritásba rendezve fogadja.
3. Az interrupt controller megszakítást küld a CPU felé egy számmal, amely jelzi, honnan származik az IRQ.
4. A CPU az aktuális program végrehajtását megszakítja, elmenti a regisztereket.
5. A kapott számból a CPU tudja, melyik interrupt handlert kell végrehajtania.
6. Az interrupt handler a munka után engedélyezi a megszakításkezelést.

# Megszakításvektor

Az operációs rendszer része a mekszakításvektor (*interrupt vector*), amely meghatározza, hogy az egyes perifériáknak hol van a megszakításkezelője.

A megszakításvektor indexelésével a megszakítást kérő I/O eszköz kijelölheti, hogy milyen jellegű kiszolgálást kér.

Az operációs rendszer sokszor nem perifériákként, hanem perifériacsoportonként (merevlemezek, hálózati csatolók stb.) tartalmaz megszakításkezelőt.

# Prioritás

A modern arhitektúrák általában több vezetéken keresztül fogadnak megszakításkéréseket.

Egyszerre több megszakítás beérkezésének esetére egy hardver prioritási rendet használunk, amely a magasabb prioritású eszköz kiszolgálását előtérbe helyezi.

# Megszakítás tiltása

A processzornak általában módja van, hogy a megszakítások fogadását szoftveresen tiltsa.

Sok architektúrában a megszakításkezelő indulása előtt automatikusan tiltódik a megszakítások fogadása.

A processzorok általában rendelkeznek nem tiltható megszakításkérési bemenettel (NMI, non maskable interrupt) veszély esetére.

# Az I/O program

A következő oldalakon arról olvashatunk, hogy milyen elvárásaink vannak az I/O kezelést végző programmal szemben és milyen módszerekkel érhetjük el a céljainkat.

# Az I/O kezelés főbb céljai

- Eszközfüggetlenség (*device independence*) az alkalmazásprogramozó felé.

  Az eszközfüggetlenség elve szerint az egyes perifériákat lehetőleg azonos módon kell kezelni a felhasználói programoknak.

  Példa: Az állománykezelés független attól, hogy az adott állományok merevlemezen vagy CD-ROM lemezen vannak.

# Az I/O kezelés főbb céljai

- Egységes névkezelés (*uniform naming*) az alkalmazásprogramozó felé.

  Az egységes névkezelés elve szerint az eszköz felépítésétől függetlenül kezeljük az eszközök objektumait.

  Példa: Minden állomány névváltozás nélkül legyen másolható a merevlemezről a hajlékonylemezre.

# Az I/O kezelés főbb céljai

- Tartsuk a hibakezelést (*error handling*) olyan alacsony szinten, (az eszközhöz közeli szinten) amilyen alacsony szinten csak lehet.

  Az alacsony szinten tartott hibakezelés előnye, hogy kihasználhatjuk az eszközről szerzett speciális ismereteket.

# Az I/O kezelés főbb céljai

- Szinkron/aszinkron váltás: az I/O nincs szinkronban a CPU által futtatott programmal.

  Minél jobban megvalósítjuk a szinkron/aszinkron elvet, annál gyorsabban használjuk az I/O eszközöket, hiszen az egyes munkafázisok annál kevésbé állnak egymás utjába.

  Az alkalmazásprogramozó viszont szinkron módon szeret programozni, hiszen az sokkal egyszerűbb.

# Az I/O kezelés főbb céljai

- Átmeneti tárolás (*buffering*).

  Láttuk, hogy a bufferelésre szükség van, ráadásul a felső szintek (alkalmazásprogramozó) csak akkor tudja kezelni az adatokat, ha már az egész a rendelkezésére áll.

- Megosztható és dedikált eszközök szétválasztása.

  Amint láttuk az eszközök kiosztása az OS feladata, amelyet szerencsés az eszközkezelőben megvalósítani, hogy a felső szintek már az operációs rendszerben is használhassák e szolgáltatást.

# Az I/O kezelés módjai

Az I/O kezelőt a következő módszerekből választva építhetjük fel:

- Programozott I/O (*programmed I/O*) lényege, hogy a perifériát folyamatosan kérdésekkel zaklatjuk.
- Megszakításkéréses I/O lényege, hogy valahányszor kiszolgálást kér a periféria mindannyiszor megszakítást kér a processzortól.
- Megszakításkéréses I/O DMA-val: lényege, hogy a megszakításkor a processzor a DMA-val végezteti el a kiszolgálást.

# Programozott I/O

A legegyszerűbb hardvert feltételező koncepció a programozott I/O, amelynek során nem használunk DMA-t vagy IRQ-t, mindent a processzor csinál.

Az elfoglalt várakozás miatt, a módszer ma már nem használható. Nem megengedhető, hogy az I/O művelet lefoglalja a processzort, ezért jobb módszerre van szükség.

# Programozott I/O

Példa programozott I/O működésére:

1. Bemásoljuk a nyomtatandó szöveget a kernel adatterületére.
2. Ciklusban kinyomtatjuk a karaktereket (egyenként!), mindig megvárva, hogy a device controller állapotregisztere ready legyen (polling, busy waiting)
3. Visszaadjuk a vezérlést a nyomtatást kérő programnak.

# I/O IRQ-val

A módszer feltételezi, hogy a harver és az operációs rendszer fel van készítve a megszakításkezelésre.

A módszer nem használja a DMA-t, ezért inkább csak lassú eszközök esetében használjuk.

Az, hogy az adott architektúrán mely perifériákat kezeljük ezzel a módszerrel operációs rendszer, sőt beállításfüggő lehet.

A módszer hátránya, hogy igen sok megszakítást generálhat.

# I/O IRQ-val

Példa megszakításkezeléses I/O működésére:

1. Bemásoljuk a nyomtatandó karaktereket a kernel adatterületére.
2. Kiküldjük az első nyomtatandó karakter.
3. Amikor a nyomtató kinyomtatta a karakter egy megszakításkéréssel jelzi, hogy kész a következő karakter nyomtatására.

# I/O DMA-val

A módszer feltételezi, hogy a hardver és az operációs rendszer a megszakításkezelésre és a DMA kezelésre is fel van készítve.

Ez a legfejlettebb módszer, a CPU olyan keveset foglalkozik a perifériával, amilyen keveset csak lehet.

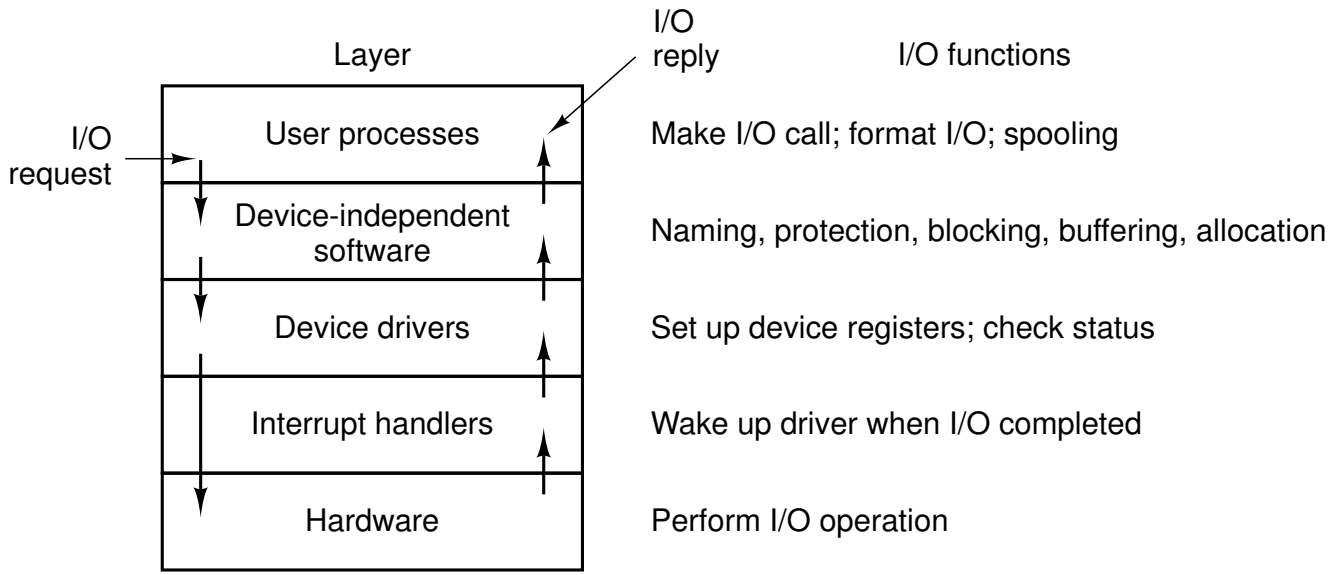
# I/O DMA-val

Példa a I/O kezelésre DMA használata esetén:

1. Bemásoljuk a nyomtatandó karaktereket a kernelterületre.
2. Felprogramozzuk a DMA regisztereit a megfelelő regiszterértékek beállításával, majd indítjuk a DMA-t.
3. Amikor a nyomtató kész a következő karakter fogadására, jelzi azt a DMA felé.
4. Amikor a nyomtatás kész, a DMA megszakítást küld a processzor felé.

# I/O szoftver

Az I/O kezelő programok többrétegűek, elkülönített feladatcsoportokkal




6. ábra. Négyrétegű I/O szoftvermodell

# Megszakításkezelő

A megszakítások kezelése meglehetősen bonyolult feladat, az tehát a szerencsés, ha a rendszer alsó szintjén kezeljük, a felső szintek elől elfedjük.

Az interrupt handler a szoftverhierarchia felső részeit blokkolni és továbbengedni tudja a megszakítások beérkezésének megfelelően. Erre használhat pl. szemaforokat.

A megszakításkezelő feladata nem egyszerű, hiszen el kell mentenie a megszakított program állapotát, kezelnie kell a megszakítást, a megfelelő módon engedélyeznie kell a felsőbb szintek működését és a folyamat végén tovább kell futtatnia valamely processzt.

# Eszközmeghajtó

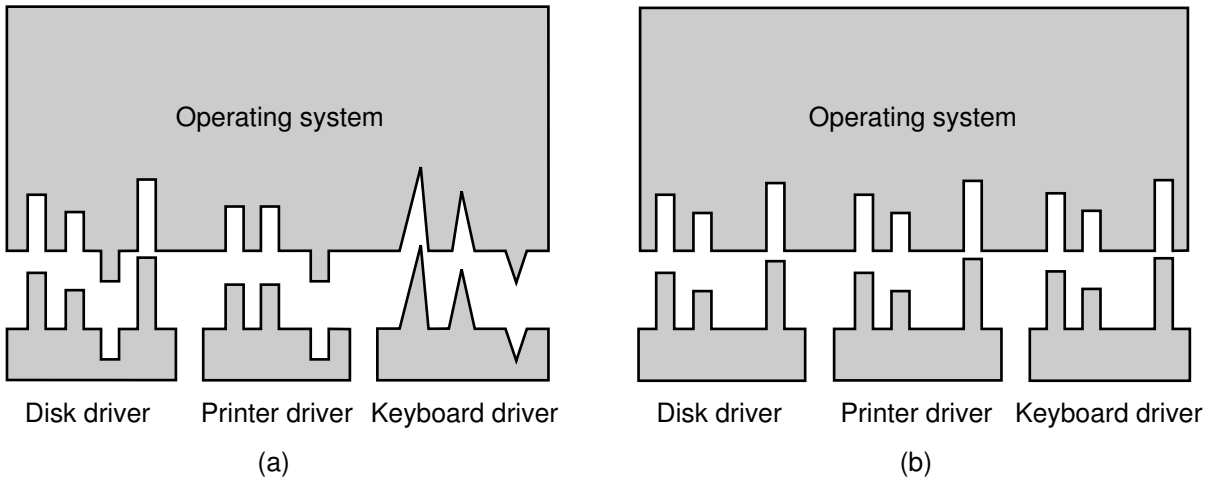
Minden I/O eszköznek szüksége van némi eszközfüggő szoftverre. Ennek neve eszközmeghajtó program (device driver).

Az eszközkezelő szoftvert általában a hw gyártó bocsátja a rendelkezésünkre – minden operációs rendszerhez külön.

Annak érdekében, hogy a eszközmeghajtó hozzáférhessen a device controller regisztereihez, általában a rendszermagban foglal helyet. Ez a rendszeromlások sorát eredményezheti.

# Eszközmeghajtó

Néhány OS (pl. Unix) monolitikus kernelt használ, ahol a device drivert be kell fordítani a kernelbe. Más rendszerek (pl. Linux) moduláris kernelt használnak, ahol a device driver futás közben betölthető.




7. ábra. Eszközmeghajtó felületek

# Eszközmeghajtó

Az eszközmeghajtó működése általában a következő lépésekre bontható:

1. A kérés paramétereinek ellenőrzése.
2. Az eszklöz foglaltságának ellenőrzése. Ha a nyilvántartás szerint szabad, meg kell vizsgálnunk, hogy üzemkész-e.
3. Parancs küldése az eszköz felé a parancsregiszterek írásával.
4. Hibaellenőrzés.
5. Eredmény ellenőrzése: vannak -e elhozandó adatok a device controllernél.

# Eszközfüggetlen szoftver

Lehetőleg minden szoftver komponenst ide kell helyeznünk, ami egységes lehet az egyes eszközök esetében.

Így ezeket az egyes eszközök közösen használhatják ezeket a szoftverelemeket.

# Eszközfüggetlen szoftver

E komponens feladatai:

- Egységes felület megvalósítása.
- Buffering. A sebességet nagymértékben növeli.
- Device independent block size biztosítása.
- Hibajelzés (programhiba a user területen, I/O eszköz hiba).
- Eszközök foglalása, odaítélése és blokkolás.