B4B35OSY: Operační systémy

Lekce 9: Vstup/výstup, ovladače

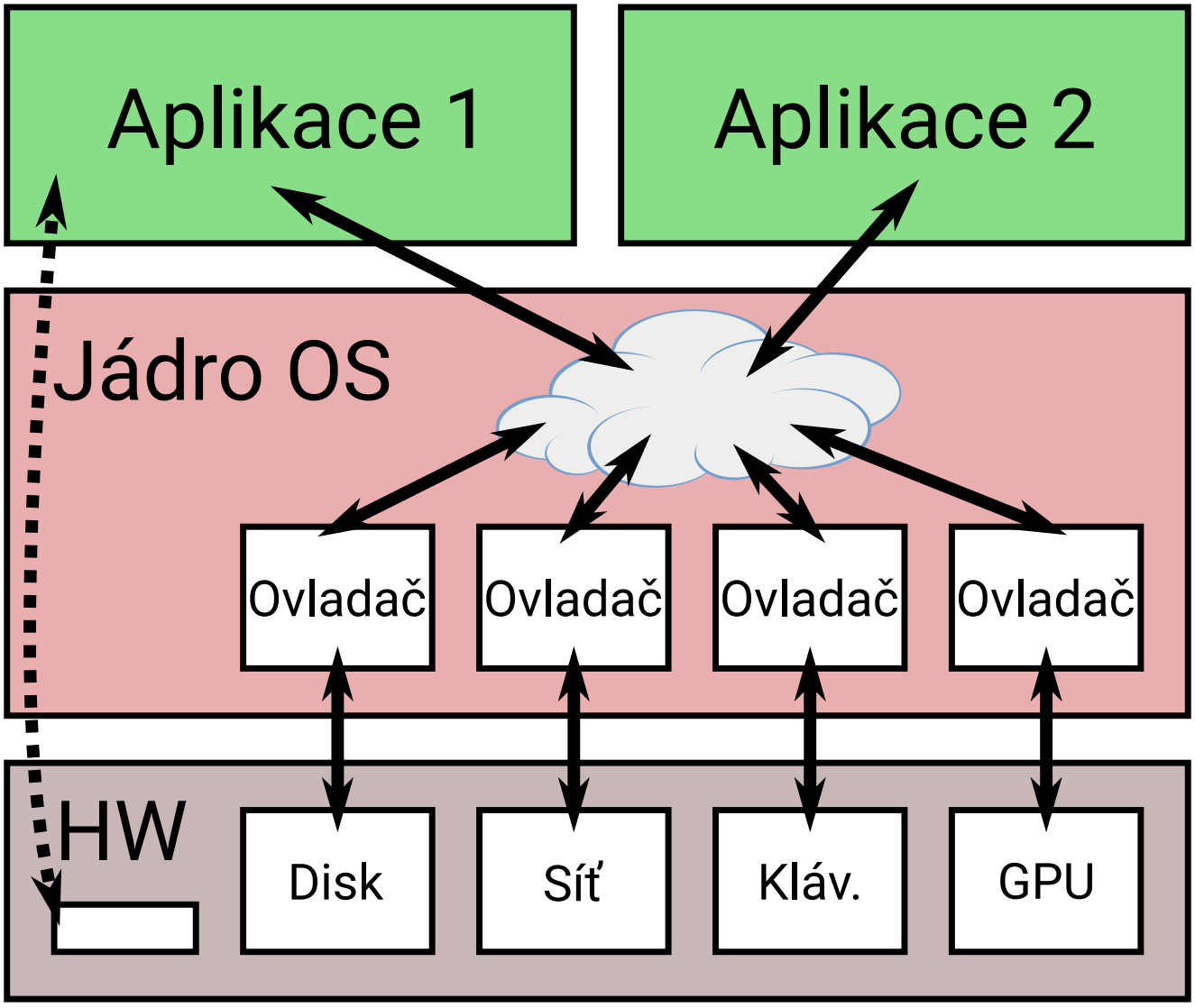
Michal Sojka
michal.sojka@cvut.cz

November 30, 2017

Osnova

Outline

Vstup a výstup v OS



Vstup a výstup

Input/Output (IO)

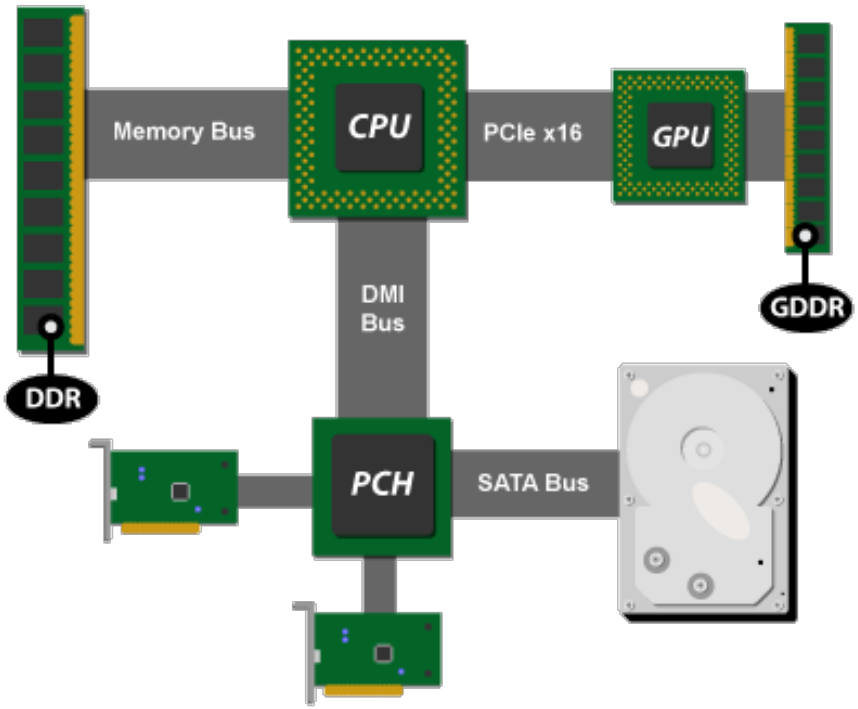
- Způsob, jak počítač komunikuje s okolním světem
  - Datová úložiště (disky)
  - Sítě
  - Klávesnice, monitor, ...
- Uživatelská aplikace nemá přímý přístup k periferiím
  - Aplikace, která nepoužívá služby jádra OS může pouze číst a zapisovat do virtuální paměti
- Pro přístup k periferiím musí používat služby OS, které
  - zajišťují „bezpečné“ **sdílení** periferií mezi aplikacemi a
  - **abstrahují** hardwarové detaily a poskytují jednotné API pro všechny periferie stejné třídy.
  - K tomu využívají služeb **ovladačů zařízení**, které naopak řeší všechny detaily práce s konkrétním hardwarem.

Outline

Úložiště

- HW pro ukládání velkého množství dat
- Není možné číst data po jednotlivých bytech, ale po tzv. blocích či sektorech
- Pevný disk – velikost bloku 512 B, 4 kB, . . .
  - Rotační
  - Solid-state (SSD)
- Flash paměť – někdy lze číst po bytech, ale mazat jde jen po blocích – typicky 128 kB
  - Typicky v embedded zařízeních
  - Základem pro SSD disky

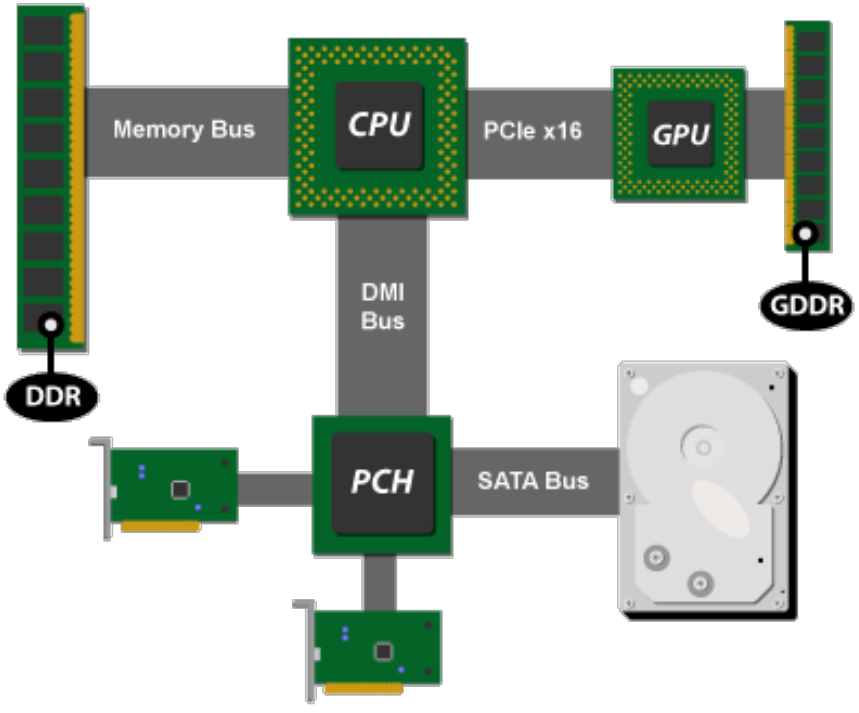
Model HW



Platforma Intel's P55. Zdroj: ArsTechnica

- Pevný disk je malý počítač, který komunikuje s hlavním CPU pomocí sběrnice.
- Přístup k disku je řádové pomalejší než přístup k paměti
- CPU posílá příkazy, disk je autonomně vykonává
- Používá se tzv. *Direct Memory Access* (DMA), také označovaný jako *Bus Master*.
  - Data proudí do paměti bez zásahu software v CPU

Model HW



Platforma Intel's P55. Zdroj: ArsTechnica

- Typické příkazy:
  - Ulož do sektorů 123456–123460 data z paměti na adrese 0x2f003200
  - Načti 32 sektorů počínaje č. 7654 a ulož je do paměti na adresu 0x302f1200
- Disky umí zpracovávat víc příkazů najednou (typicky 32)
  - Interně provádí optimalizace (např. změna pořadí vykonávání či slučování požadavků).
  - O dokončení operace je CPU informováno přerušením.

Přístup aplikací k úložišti

- Aplikace typický nepřistupují k úložišti přímo, ale skrze **souborový systém** (viz příští přednášku)
- OS optimalizuje přístup k úložišti:
  - Spravuje vyrovnávací paměť pro rychlejší přístup k datům na disku
  - OS sám předem načítá data o kterých předpokládá, že budou brzy potřeba
  - Pro pomalé rotační disky:
    - Slučuje požadavky aplikací do větších
    - Rozvrhuje, kdy který požadavek vykonat – optimalizace přejezdů hlaviček – tzv. IO scheduler.

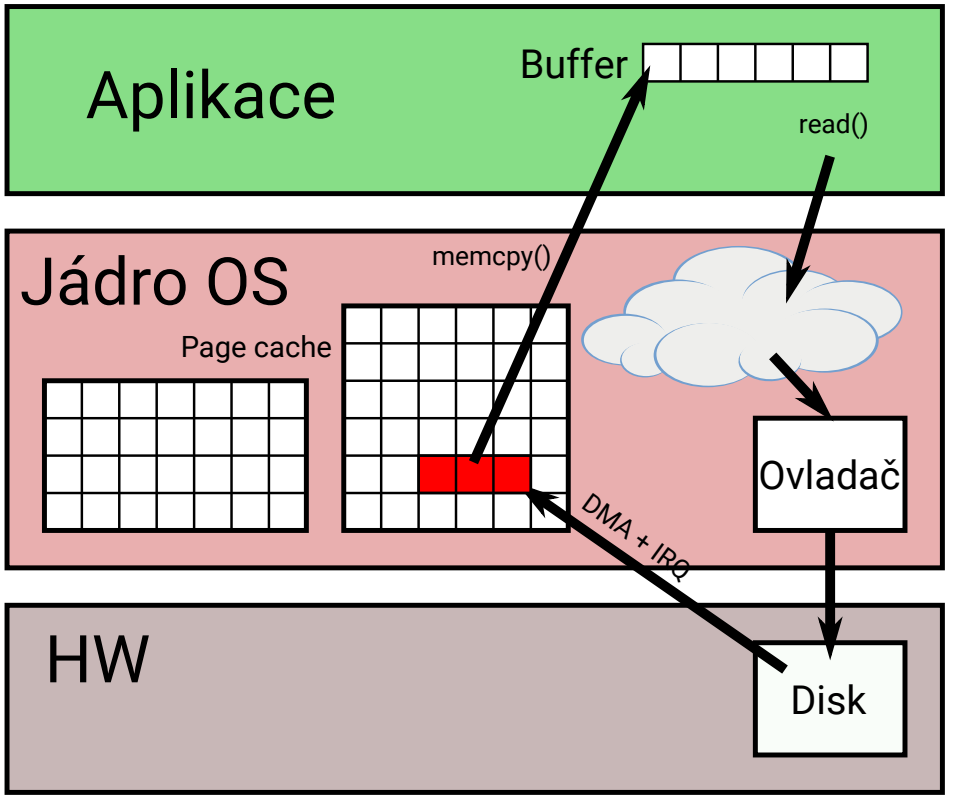
Stránková vyrovnávací paměť

Page cache – název používaný Linuxem pro vyrovnávací paměť disku

- Data čtená z disku resp. zapisovaná na disk jsou uchovávána v paměti pro případné další použití
- OS se snaží využít veškerou volnou paměť jako diskovou cache
- Spravována po stránkách (4 kB)
  - I když starší disky používaly 512 B sektory, OS (téměř) vždy načítá celé 4 kB.

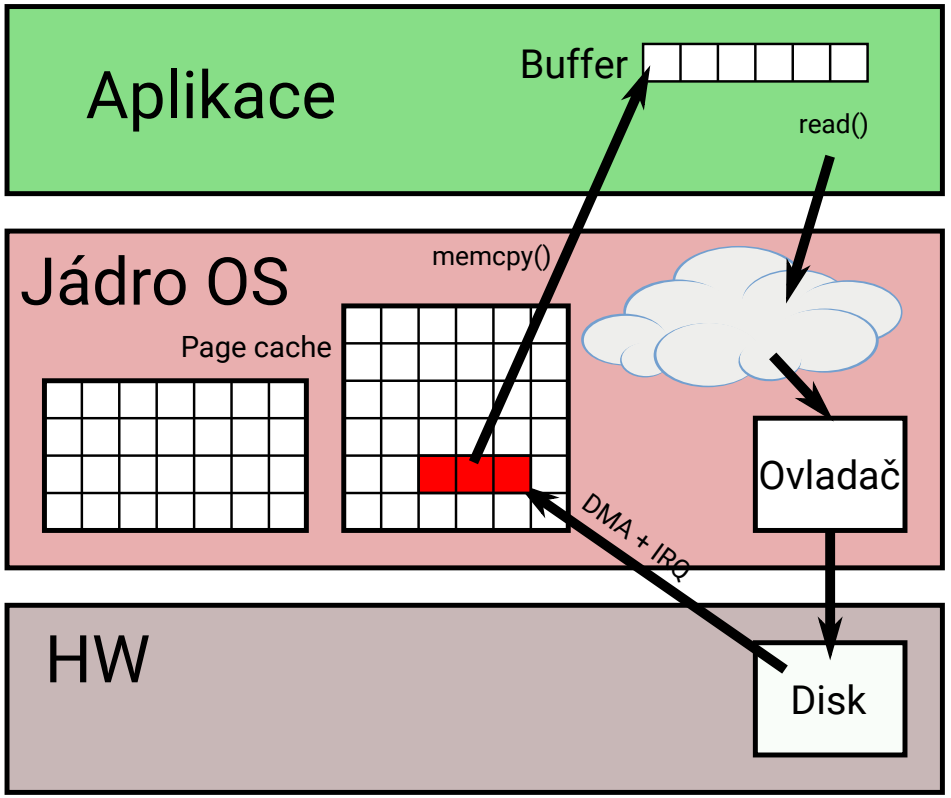
Čtení a zápis

- Čtení z disku:
  1. Aplikace zavolá read()
  2. Disky se pošle příkaz pro načtení dat a uložení do page cache
  3. Když disk načítání dokončí (IRQ), OS zkopíruje data z page cache do paměti aplikace



Čtení a zápis

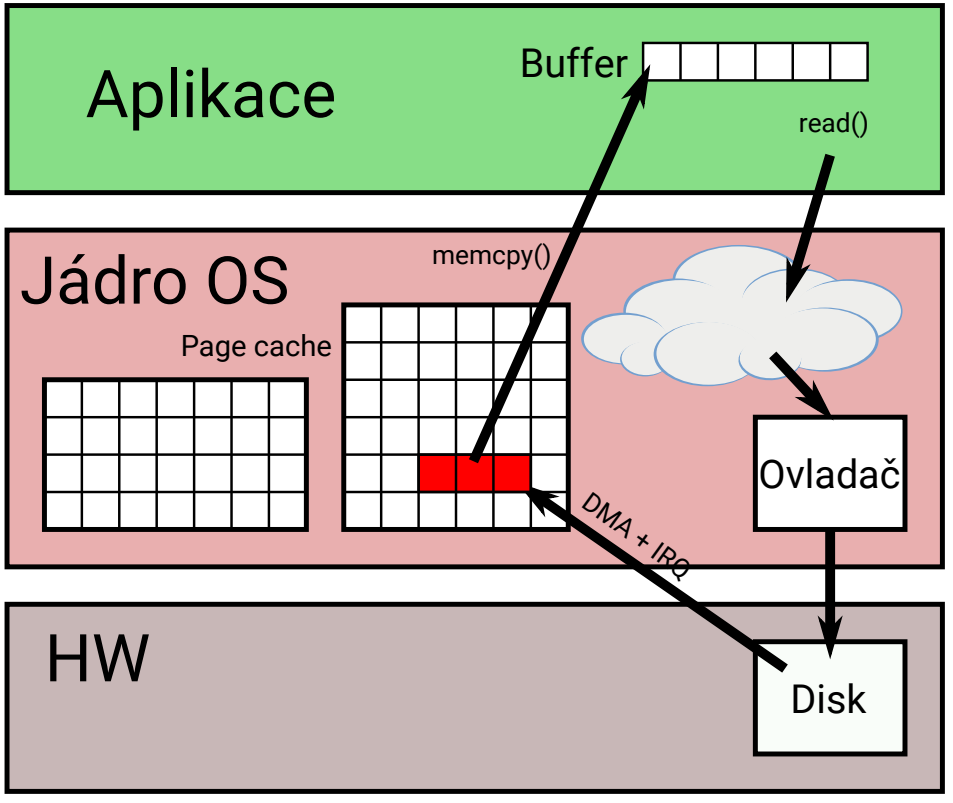
- Zápis na disk:
  1. Aplikace zavolá write()
  2. Data se zkopírují do page cache
  3. Čas od času OS zapisuje „špinavé stránky“ na disk.
     - Zápis se dá vynutit voláním fsync() (Linux)



Pozn.: *fsync()* vs. *fflush()*

Čtení a zápis

- Zápis na disk:
  1. Aplikace zavolá write()
  2. Data se zkopírují do page cache
  3. Čas od času OS zapisuje „špinavé stránky“ na disk.
     - Zápis se dá vynutit voláním fsync() (Linux)

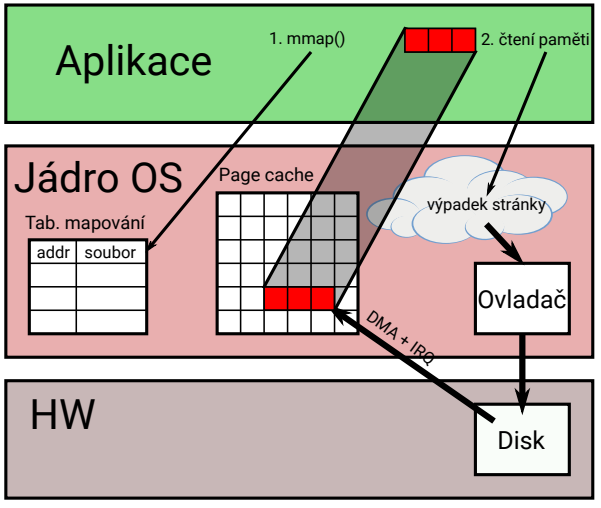


Pozn.: *fsync()* vs. *fflush()*
fsync() ukládá data z page cache na disk, *fflush()* ukládá data z bufferu aplikace (schovaný v libc) do page cache.

Čtení a zápis bez zbytečného kopírování dat

- Aplikace může požádat OS, aby „namapoval“ stránky diskové cache do jejího adresního prostoru (v UNIXu systémové volání mmap)
- Při prvním přístupu k paměti vrácené funkcí mmap dojde k výjimce (výpadku stránky), protože nic ještě není namapováno

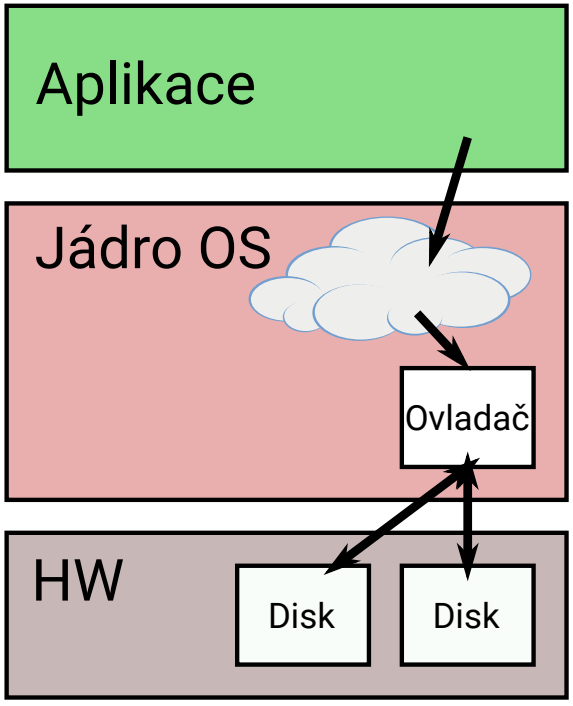
  1. OS se podívá do tabulky mapování (pro Vás viditelná v /proc/<PID>/maps), aby zjistil, jaký soubor je potřeba načíst a načte data z disku do cache
  2. Poté modifikuje stránkovací tabulku procesu a vrátí se z obsluhy výjimky na instrukci, která výjimku způsobila
  3. Tentokrát se instrukce provede úspěšně a aplikace pokračuje
- Nedostatky?



- Zápis se provádí do namapované paměti stejně jako čtení
- Čas od času OS zapisuje „špinavé stránky“ na disk.
- Pouze při použití *msync()* máte jistotu, že jsou data uloženy na disku (pro případ výpadku napájení)
- Sdílení dat jednoho souboru mezi pocesy se uskutečňuje prostřednictvím page-cache a není vázáno na uložení na disk

Disková pole

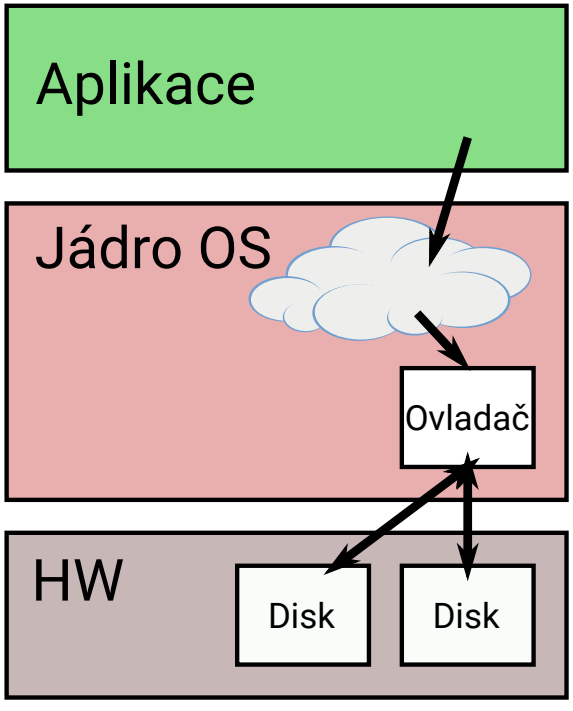
RAID – Redundant Array of Independent Disks



- Pokud se disk porouchá, přijdeme o (cenná) data
- Redundance – data jsou uložená na více místech najednou
- Možnost implementace v HW nebo v SW (OS)
- Rychlost SW implementace – čtení typicky rychlejší (paralelní čtení z více disků), zápis o něco pomalejší.
- Nahradí RAID zálohování dat?

Disková pole

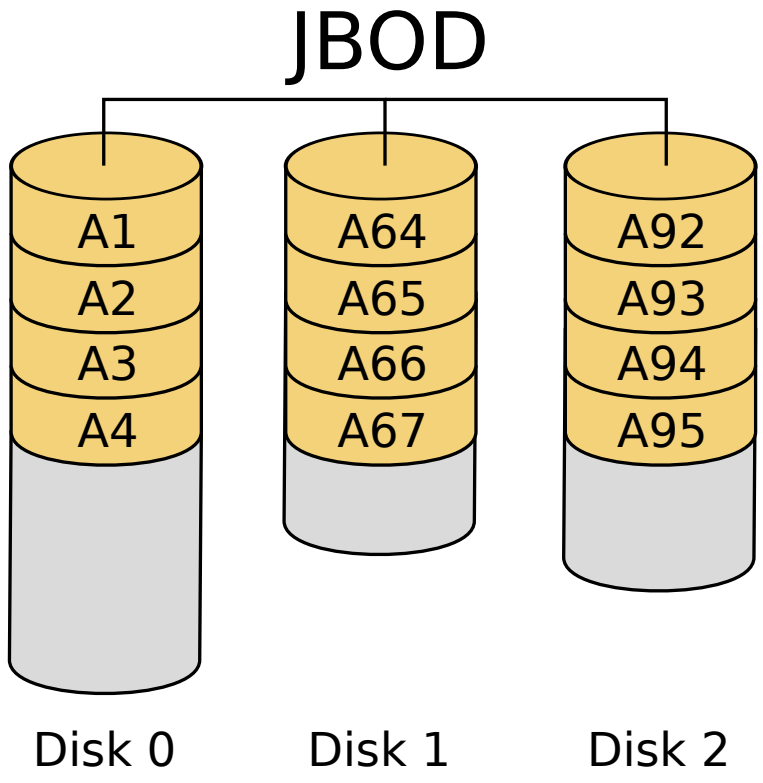
RAID – Redundant Array of Independent Disks



- Pokud se disk porouchá, přijdeme o (cenná) data
- Redundance – data jsou uložená na více místech najednou
- Možnost implementace v HW nebo v SW (OS)
- Rychlost SW implementace – čtení typicky rychlejší (paralelní čtení z více disků), zápis o něco pomalejší.
- Nahradí RAID zálohování dat?
  - Požár v serverovně – záloha na jiném místě
  - Administrátor omylem smaže data

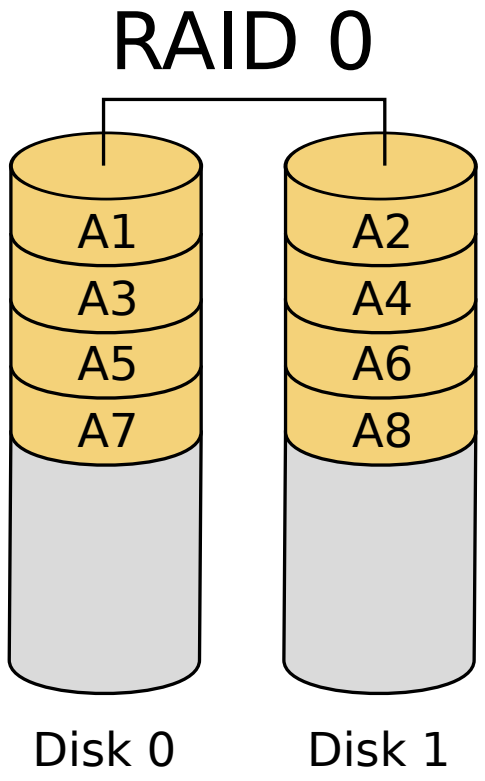
Typy diskových polí

- RAID0 – spojení více disků do jednoho virtuálního (bez redundance)
- RAID1 – zrcadlení, efektivita: 50%
- RAID5 – prokládání dat a parita, min. 3 disky, toleruje ztrátu jednoho disku, efektivita $\frac{n-1}{n}$
  $Ap = A1 \oplus A2 \oplus A3$ (xor)
  Při poruše 1. disku:
  $A1 = Ap \oplus A2 \oplus A3$
- RAID6 – toleruje ztrátu dvou disků



Typy diskových polí

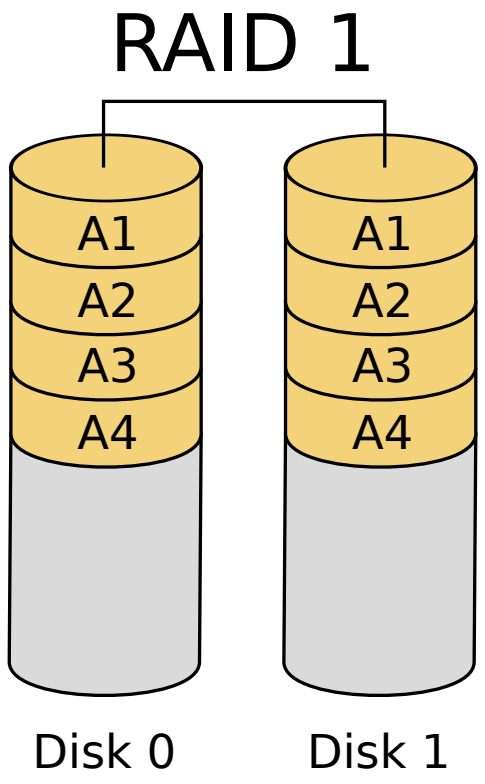
- RAID0 – spojení více disků do jednoho virtuálního (bez redundance)
- RAID1 – zrcadlení, efektivita: 50%
- RAID5 – prokládání dat a parita, min. 3 disky, toleruje ztrátu jednoho disku, efektivita $\frac{n-1}{n}$
  $Ap = A1 \oplus A2 \oplus A3$ (xor)
  Při poruše 1. disku:
  $A1 = Ap \oplus A2 \oplus A3$
- RAID6 – toleruje ztrátu dvou disků



Autor: en:User:Cburnett – Vlastní dílo, CC BY-SA 3.0

Typy diskových polí

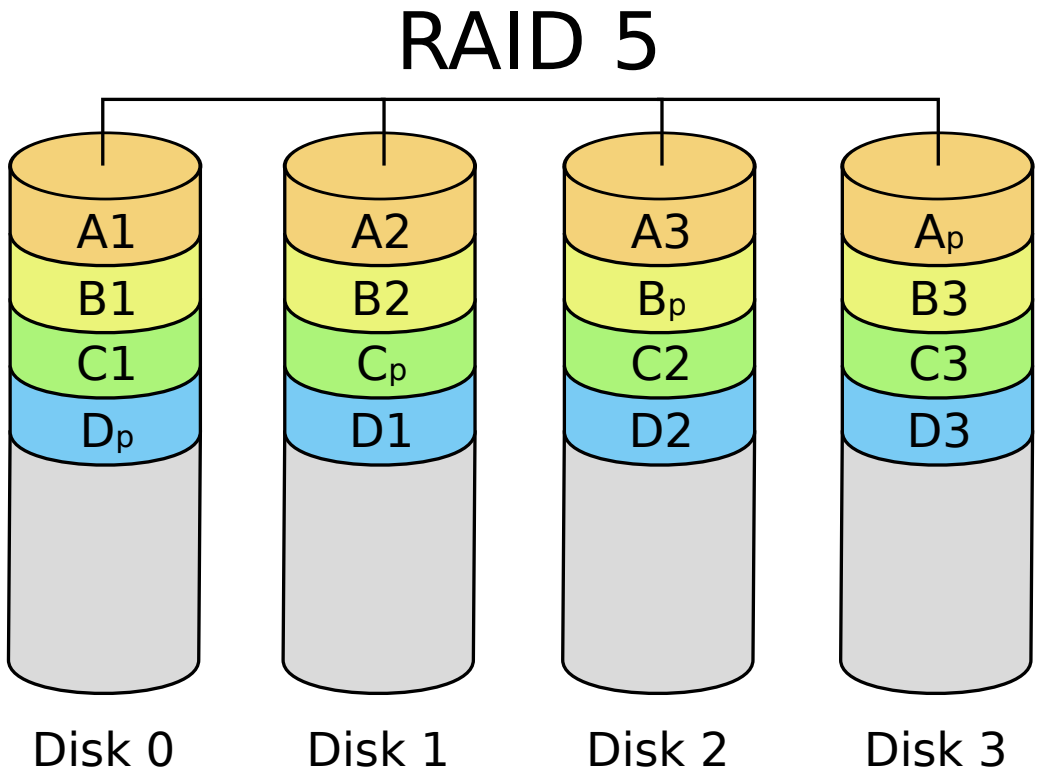
- RAID0 – spojení více disků do jednoho virtuálního (bez redundance)
- RAID1 – zrcadlení, efektivita: 50%
- RAID5 – prokládání dat a parita, min. 3 disky, toleruje ztrátu jednoho disku, efektivita $\frac{n-1}{n}$
  $Ap = A1 \oplus A2 \oplus A3$ (xor)
  Při poruše 1. disku:
  $A1 = Ap \oplus A2 \oplus A3$
- RAID6 – toleruje ztrátu dvou disků



Autor: en:User:Cburnett – Vlastní dílo, CC BY-SA 3.0

Typy diskových polí

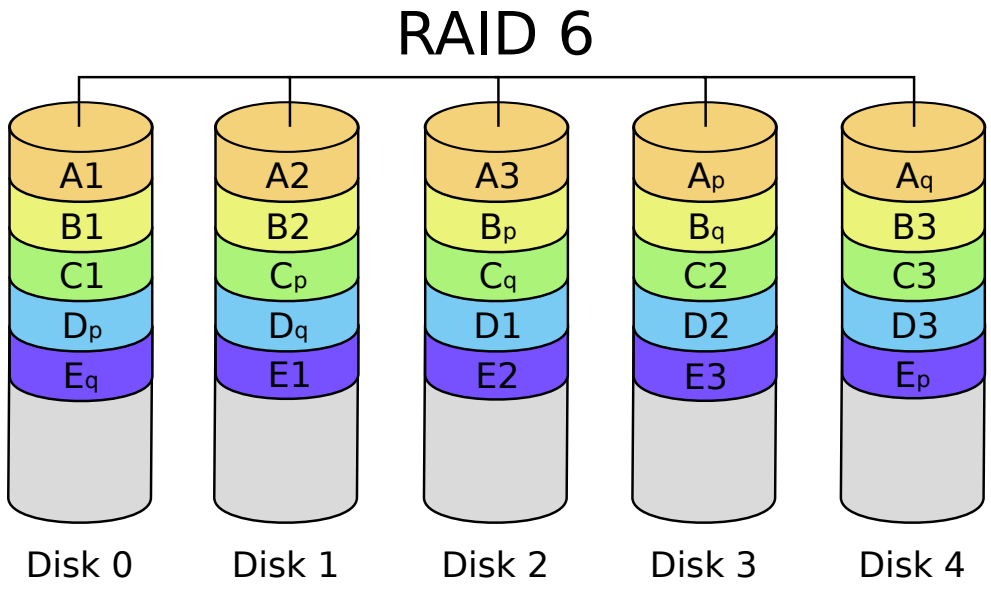
- RAID0 – spojení více disků do jednoho virtuálního (bez redundance)
- RAID1 – zrcadlení, efektivita: 50%
- RAID5 – prokládání dat a parita, min. 3 disky, toleruje ztrátu jednoho disku, efektivita $\frac{n-1}{n}$
  $Ap = A1 \oplus A2 \oplus A3$ (xor)
  Při poruše 1. disku:
  $A1 = Ap \oplus A2 \oplus A3$
- RAID6 – toleruje ztrátu dvou disků



Autor: en:User:Cburnett – Vlastní dílo, CC BY-SA 3.0

Typy diskových polí

- RAID0 – spojení více disků do jednoho virtuálního (bez redundance)
- RAID1 – zrcadlení, efektivita: 50%
- RAID5 – prokládání dat a parita, min. 3 disky, toleruje ztrátu jednoho disku, efektivita $\frac{n-1}{n}$
  $Ap = A1 \oplus A2 \oplus A3$ (xor)
  Při poruše 1. disku:
  $A1 = Ap \oplus A2 \oplus A3$
- RAID6 – toleruje ztrátu dvou disků



Autor: en:User:Cburnett – Vlastní dílo, CC BY-SA 3.0

Outline

Sítě

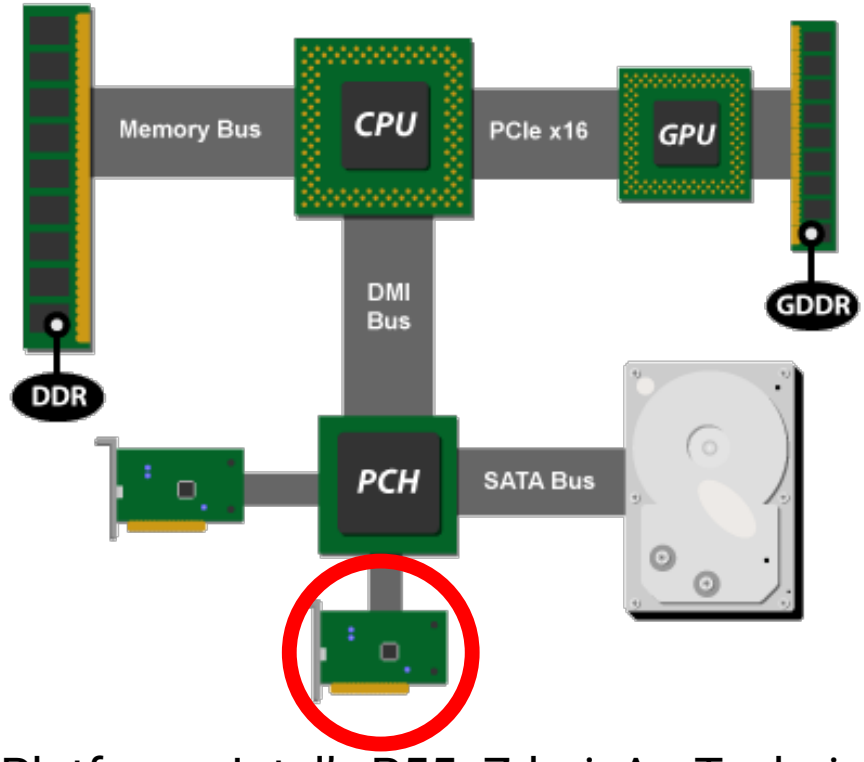
- Ethernet, Wi-Fi, Bluetooth, CAN, ...
- Virtuální sítě – VPN, ...
- Ethernet představuje základní model sítě používaný OS
  - Základní funkce: posílání a příjem rámců jsou stejné
  - Jednotlivé síťové technologie se liší především nastavováním parametrů
- OS reprezentuje síťový HW pomocí tzv. síťových rozhraní

Sítě

- Ethernet, Wi-Fi, Bluetooth, CAN, ...
- Virtuální sítě – VPN, ...
- Ethernet představuje základní model sítě používaný OS
  - Základní funkce: posílání a příjem rámců jsou stejné
  - Jednotlivé síťové technologie se liší především nastavováním parametrů
- OS reprezentuje síťový HW pomocí tzv. síťových rozhraní

- Sítě jsou velmi rychlé – dnes až 100 Gbps
- Síťový subsystém OS musí být velmi efektivní, aby OS nebyl úzkým hrdlem
- Úložiště a sítě mají z pohledu OS se sítěmi mnoho společného
  - Do nedávna nebyla efektivita diskového subsystému důležitá, ale s nástupem rychlých SSD disků nabývá na důležitosti a síťování je zde inspirací

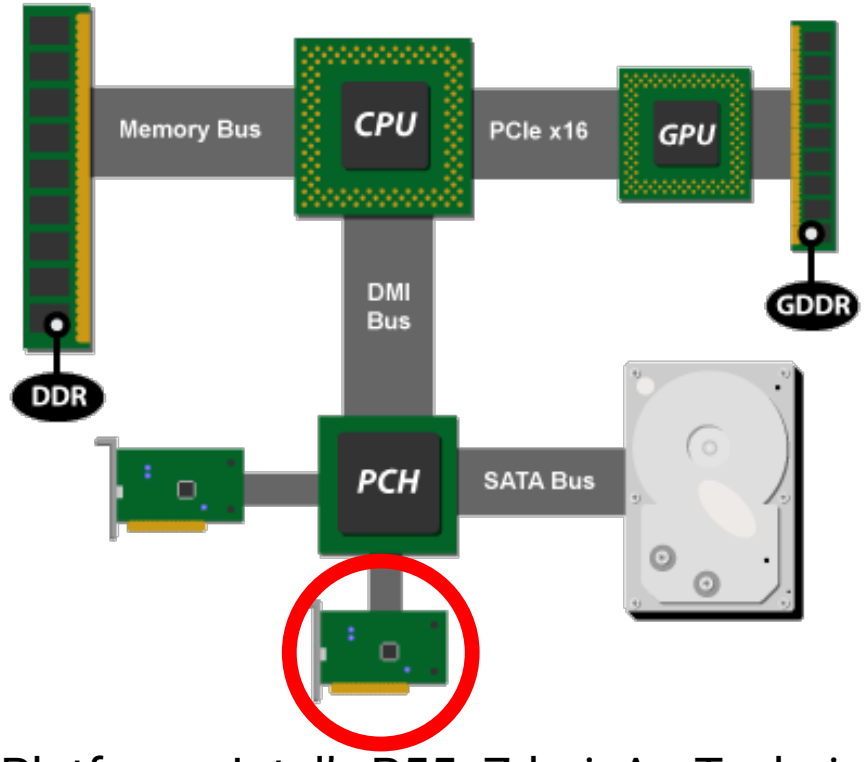
Síťový hardware



Platforma Intel's P55. Zdroj: ArsTechnica

- Síťové rozhraní je malý počítač, který komunikuje s hlavním CPU pomocí sběrnice.
- CPU posílá příkazy, síťové rozhraní je autonomně vykonává
- Používá se tzv. *Direct Memory Access* (DMA), také označovaný jako *Bus Master*.
  - Data proudí z/do paměti bez zásahu software v CPU

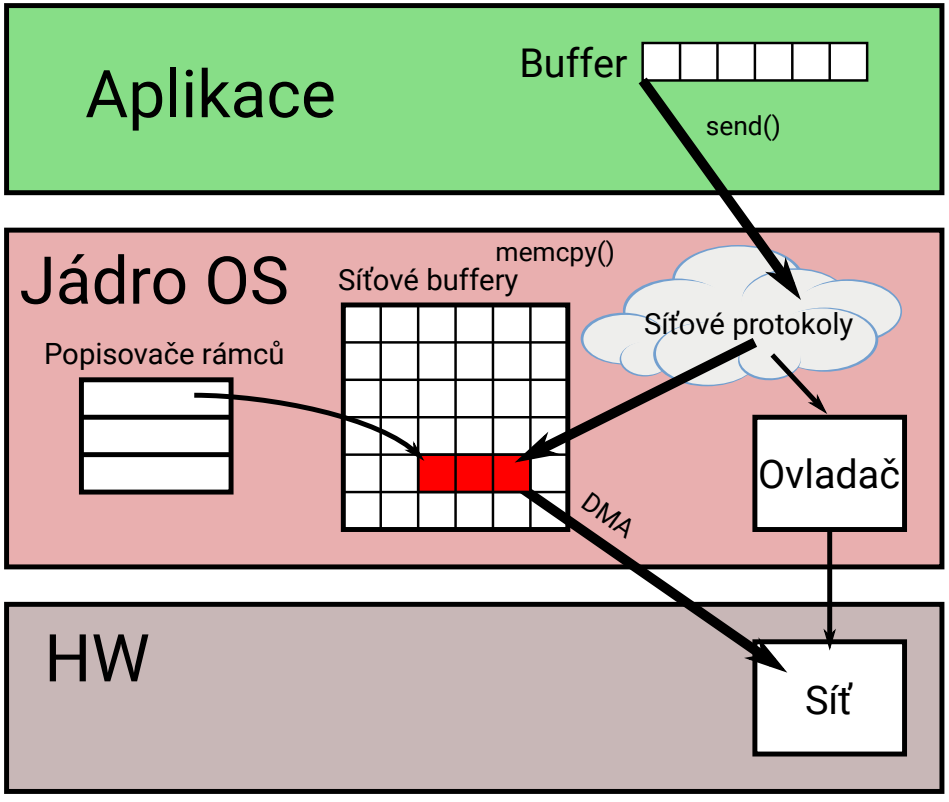
Síťový hardware



Platforma Intel's P55. Zdroj: ArsTechnica

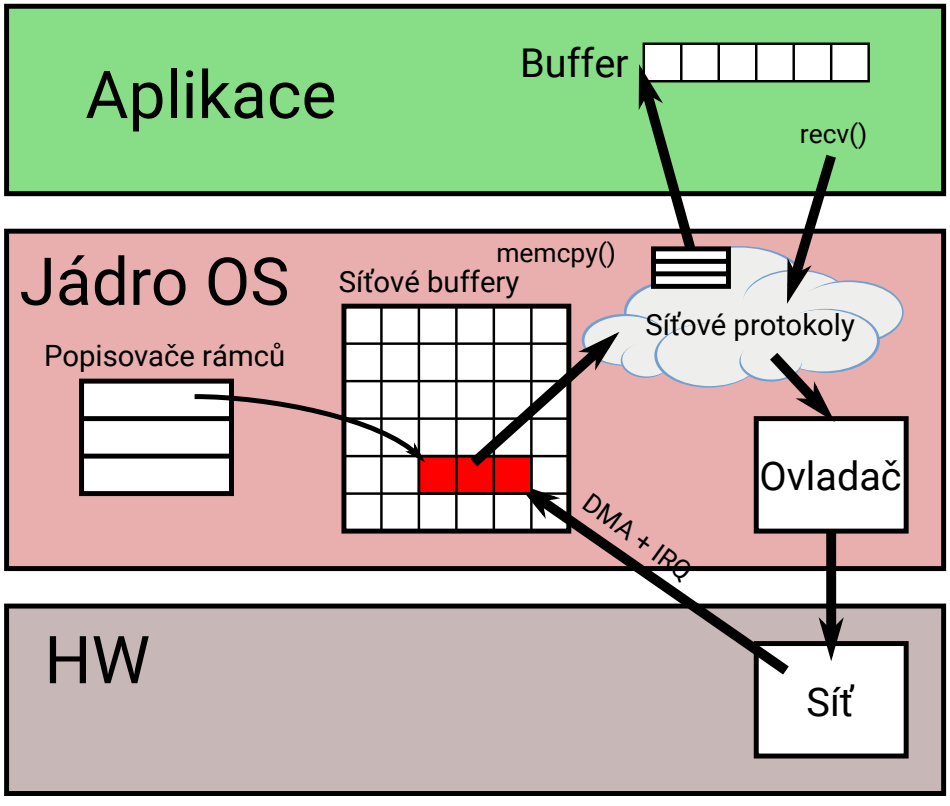
- Typické „příkazy“:
  - Pošli rámec, který je uložený na adrese 0x2f003200.
  - Pokud přijmeš rámec, ulož ho na adresu 0x302f1200.
  - Implementováno pomocí tabulky popisovačů rámců (packet descriptor table) – ovladač vytvoří v paměti tabulku ukazatelů na rámce a síťové rozhraní si jí vyčte.

Odesílání dat aplikacemi



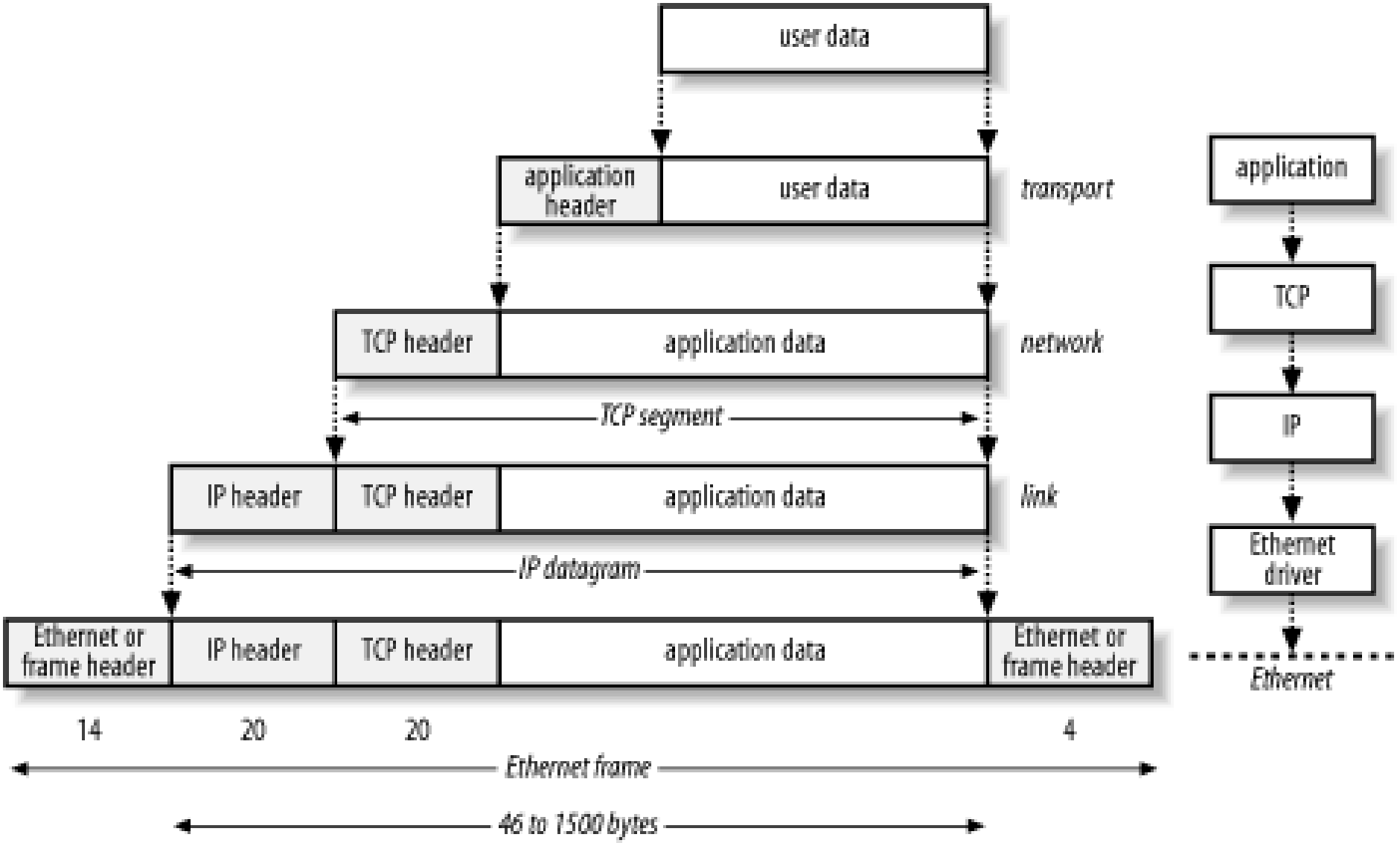
1. Aplikace zavolá send()
2. Odesílaná data se zkopírují do bufferů v jádře
3. OS (tzv. protokolový zásobník) přidá k aplikačním datům potřebné hlavičky, uloží výsledek do síťových bufferů a upozorní ovladač
4. Ovladač upraví tabulku popisovačů rámců, a dá vědět (jak?) síťovému HW, že se tabulka popisovačů změnila.
5. HW začne číst data z paměti a odešle je.

Příjem dat aplikacemi



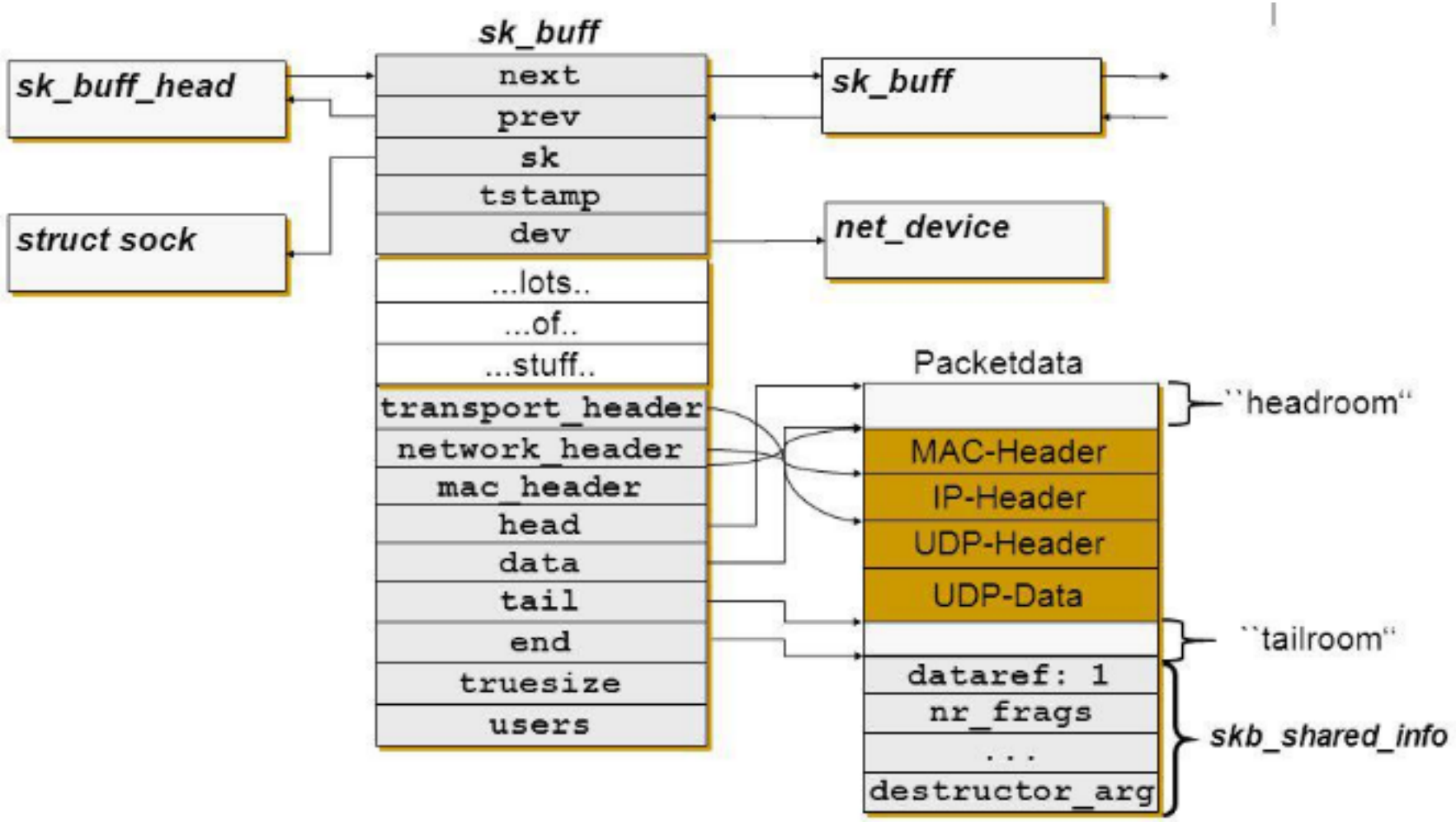
1. Aplikace zavolá recv()
2. Pokud už jsou nějaká přijatá data připravena ve frontě socketu, pokračuje se krokem 6, v opačném případě se vlákno zablokuje a čeká.
3. Při příjmu rámce ho síťové rozhraní autonomně ukládá do paměti.
4. Po dokončení příjmu je upozorněn ovladač (přerušení) a ten pak aktivuje zpracování rámce síťovými protokoly.
5. Poté je rámec zařazen do fronty patřičného soketu.
6. Přijatá data jsou nakopírována ze sítových bufferů v jádře do aplikačního bufferu.

Síťové protokoly



Datová struktura pro práci se síťovými rámci

struct skbuff v Linuxu



- Možnost přidávat hlavičky před data, bez nutnosti jejich kopírování
- Scatter-gatter DMA – hardware si umí sestavit rámec „za běhu“ z více částí

Příjem a odesílání dat bez kopírování

Zero-copy networking

- Podobný „trik“, jako s diskovou vyrovnávací pamětí
- socket(AF_PACKET, ...) + mmap()
- Síťový HW přijímá/odesílá rámce rovnou z paměti kontrolované aplikací
- Nevýhody:
  - Aplikace si musí sama řešit přidávání a odebírání hlaviček
  - Aplikace nesmí modifikovat rámce (např. kvůli chybě v programu) , které jsou v procesu odesílání.

Rozvrhování rámců při odesílání

- Prioritizace interaktivní komunikace
- Spravedlivé rozdělení šířky pásma mezi uživatele (zákazníky)
- Problém zvaný „buffer-bloat“
  - Ovladač může do odesílací fronty (popisovač rámců k odeslání) uložit velké množství rámců.
  - Síťový HW odesílá rámce v pořadí, v jakém jsou tam uvedeny.
  - Pokud je na konci fronty rámec, který by měl být odeslán přednostně, musí dlouho čekat.
  - Řešení:
    - Fronta ovladače se udržuje krátká, aby kritické rámce mohly „předbíhat“

Rozvrhování rámců při odesílání

- Prioritizace interaktivní komunikace
- Spravedlivé rozdělení šířky pásma mezi uživatele (zákazníky)
- Problém zvaný „buffer-bloat“
  - Ovladač může do odesílací fronty (popisovač rámců k odeslání) uložit velké množství rámců.
  - Síťový HW odesílá rámce v pořadí, v jakém jsou tam uvedeny.
  - Pokud je na konci fronty rámec, který by měl být odeslán přednostně, musí dlouho čekat.
  - Řešení:
    - Fronta ovladače se udržuje krátká, aby kritické rámce mohly „předbíhat“
- Moderní síťový hardware implementuje více front pro odesílání (i příjem)
  - Rámce jsou rozvrhovány (i) v hardwaru – výběr fronty
  - Využívá se ve vícejádrových systémech, kde má každé jádro samostatnou frontu a není potřeba ztrácet čas synchronizací v ovladači
  - Někdy lze využít i k prioritizaci rámců – každá fronta má jinou prioritu

Outline

Ovladač zařízení

Device driver

- Software, který ovládá konkrétní zařízení (disk, síťová karta, GPU, . . .) a nabízí zbytku OS jednotné rozhraní
- Se zařízením typicky komunikuje pomocí do paměti mapovaných registrů
- Obsluhuje přerušení od zařízení
- Ovladače bývají nejméně spolehlivou částí jádra OS
  - Chyba kdekoli v jádře OS (tedy i v ovladači) může způsobit nestability celého systému
  - Ne každý programátor ovladačů rozumí všem potřebným detailům
  - Ovladače se nedají testovat, pokud není k dispozici konkrétní HW
  - Velmi špatně se testuje obsluha chybových stavů, protože je potřeba donutit HW, aby signalizoval chybu
  - Microsoft zavedl povinné digitální podepisování ovladačů, aby měl částečnou kontrolu nad jejich kvalitou
- Dnešní OS umožňují, aby některé ovladače běžely v uživatelském prostoru (jako aplikace)

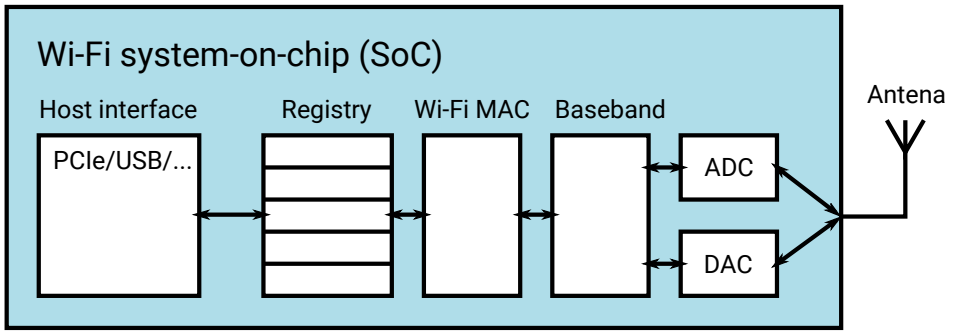
Příklad – ovladač klávesnice

1. Aplikace zavolá **getch()**/scanf()/… na standardní vstup
2. libc vyvolá systémové volání **read()** na deskriptoru souboru 0 (stdin)
3. Standardní vstup je připojen k terminálu (klávesnice + obrazovka)
4. Požadavek na vstup je tedy předán **ovladači** klávesnice
   - Ovladač klávesnice spravuje buffer znaků
5. Pokud je buffer prázdný, ovladač **uspí volající vlákno**
   - Interně k tomu použije semafor – vlákno přidá do fronty semaforu
   - Poté zavolá plánovač, aby vybral jiné vlákno, které poběží
6. Po stisku klávesy HW vyvolá **přerušení**
7. Ovladač klávesnice přerušení obslouží:
   - Přečte z HW (registru) jaká byla stisknuta klávesa a uloží ji do bufferu
   - Zavolá operaci up/post na semafor
8. Uspané vlákno aplikace se **probudí** (je stále v jádře), vyčte z bufferu ovladače stisknuté znaky a zkopíruje je do bufferu v aplikaci.
9. Provede se **návrat** ze systémového volání zpět do aplikace, funkce getch/scanf se dokončí.

Variabilita a složitost HW

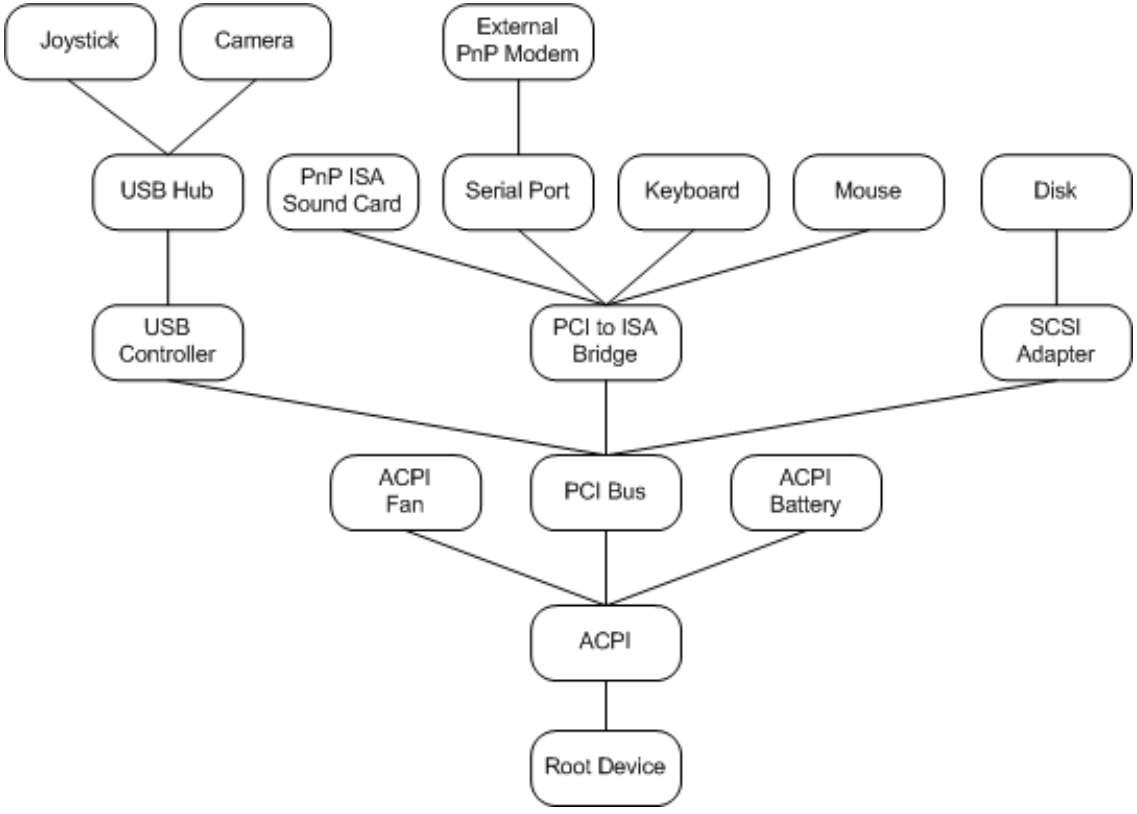
- Dnešní hardware je složitý, zařízení mohou obsahovat stovky či tisíce registrů
- I vývojáři HW mají v oblibě „Ctrl-C, Ctrl-V“ – jeden čip existuje v mnoha variantách, ale všechny mají téměř stejné registry
  - Např. Wi-Fi čip – jedna verze se připojuje k PCIe, jiná k USB
- Struktura ovladačů je modulární – chceme ovladač napsat jednou a používat pro všechny varianty čipu

Příklad:



Hierarchie ovladačů

Topologie hardwaru



Source: Microsoft

- Ovladače reflektují topologii HW
- Každý uzel má vlastní ovladač nezávislý na okolí
- Plug-and-Play (PnP)
  - Ovladač sběrnice (USB, PCI) detekuje připojená zařízení a automaticky načte potřebný ovladač zařízení

Ovladače v Linuxu

- Aplikace s ovladači komunikuje:
  - Nepřímo – síťové API, práce se soubory, stdin/out
  - Přímo – většina zařízení je reprezentována jako speciální soubor v adresáři /dev (např. sériová linka /dev/ttyUSB0).
  - Aplikace často pro přístup k souborům v /dev používají knihovny (např. libusb).
- Ovladač poskytuje aplikacím následující operace:
  - open – slouží pro „navázání spojení“ aplikace s ovladačem
  - read – čtení dat ze zařízení (např. hexdump /dev/input/mouse1)
  - write – zápis dat do zařízení (např. tty; echo XXX > /dev/pts/3),
  - ioctl – vše ostatní, co není čtení či zápis, často nastavování (man ioctl_list, ioctl_tty, . . .)
  - close – ukončení komunikace s ovladačem

Komunikace mezi ovladači (Linux)

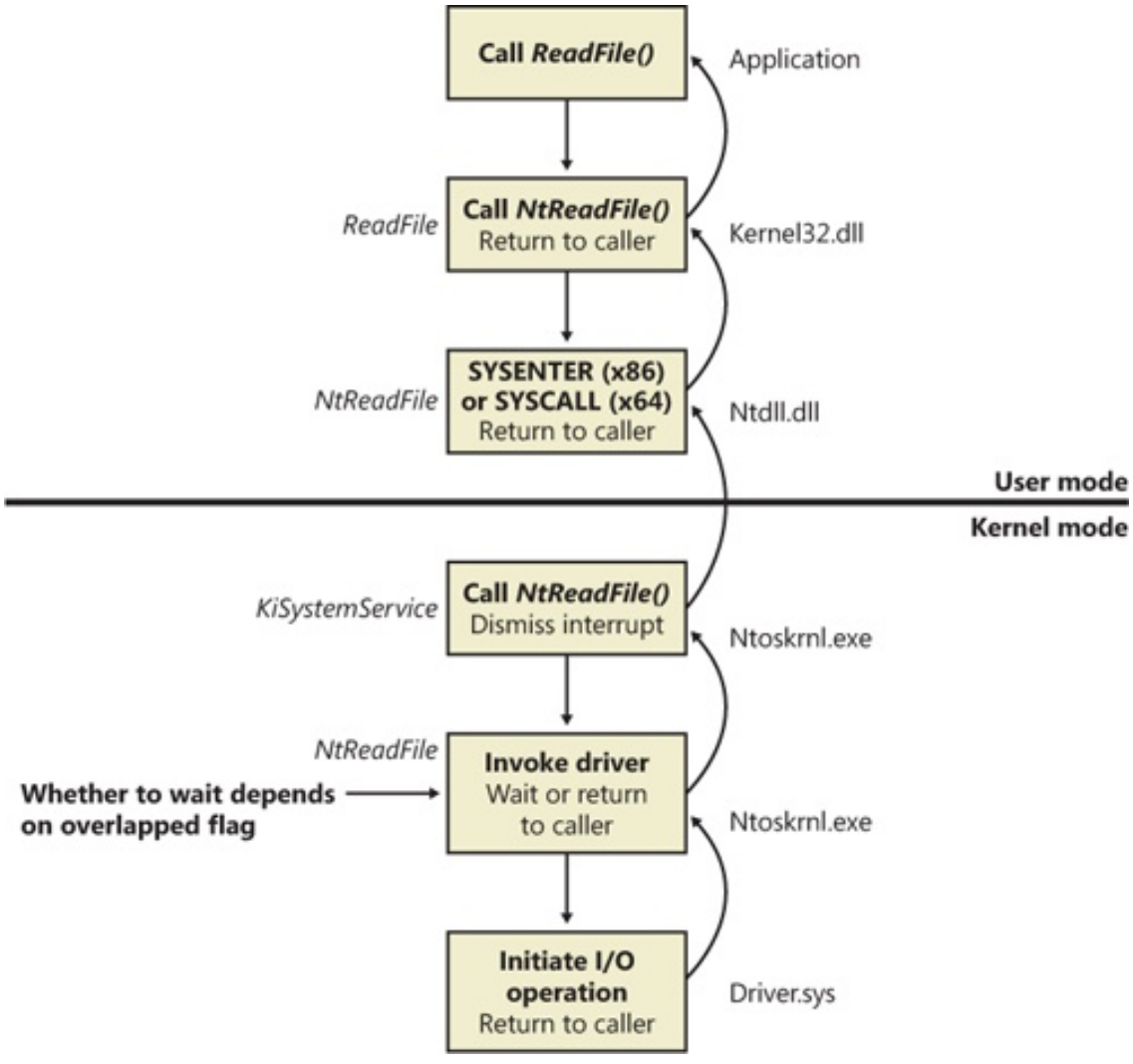
- Linux je monolitické jádro.
- Jednotlivé ovladače se volají vzájemně úplně stejně, jako se volají funkce v uživatelských aplikacích.
- Často se funkce nevolá přímo, ale přes ukazatel
  - Např. Každý ovladač si registruje ukazatel na funkci, která se má vyvolat, když aplikace zavolá read().
- Data se předávají skrze argumenty funkcí (buď přímo nebo pomocí ukazatelů).

Přístup k ovladačům ve Windows

- Z pohledu aplikace konceptuálně podobné Linuxu:

	Linux	**Windows**
Otevření ovladače	open	CreateFile
Operace s ovladačem	read, write, ioctl	ReadFile, DeviceIoControl, . . .
Uzavření ovladače	close	CloseHandle
Jmenný prostor	/dev/	\\.\
Příklad	/dev/ttyUSB0	\\.\COM6

Přístup k ovladačům ve Windows

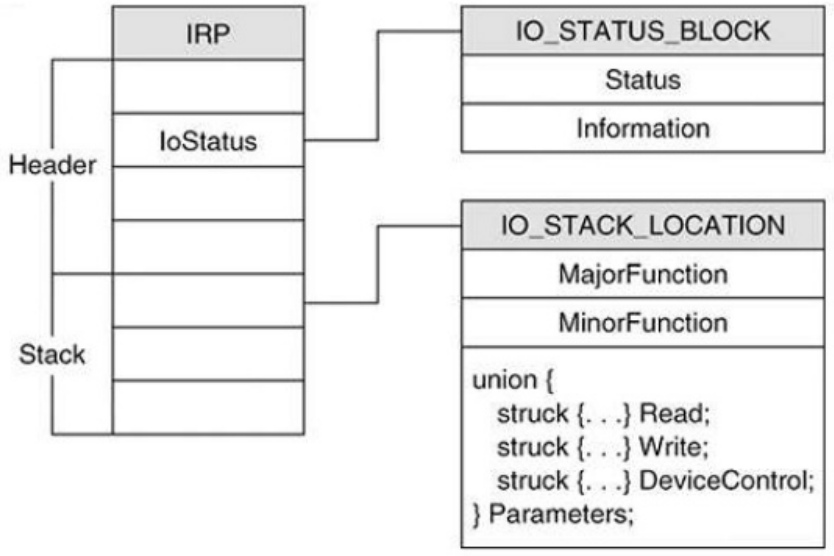


Source: Microsoft

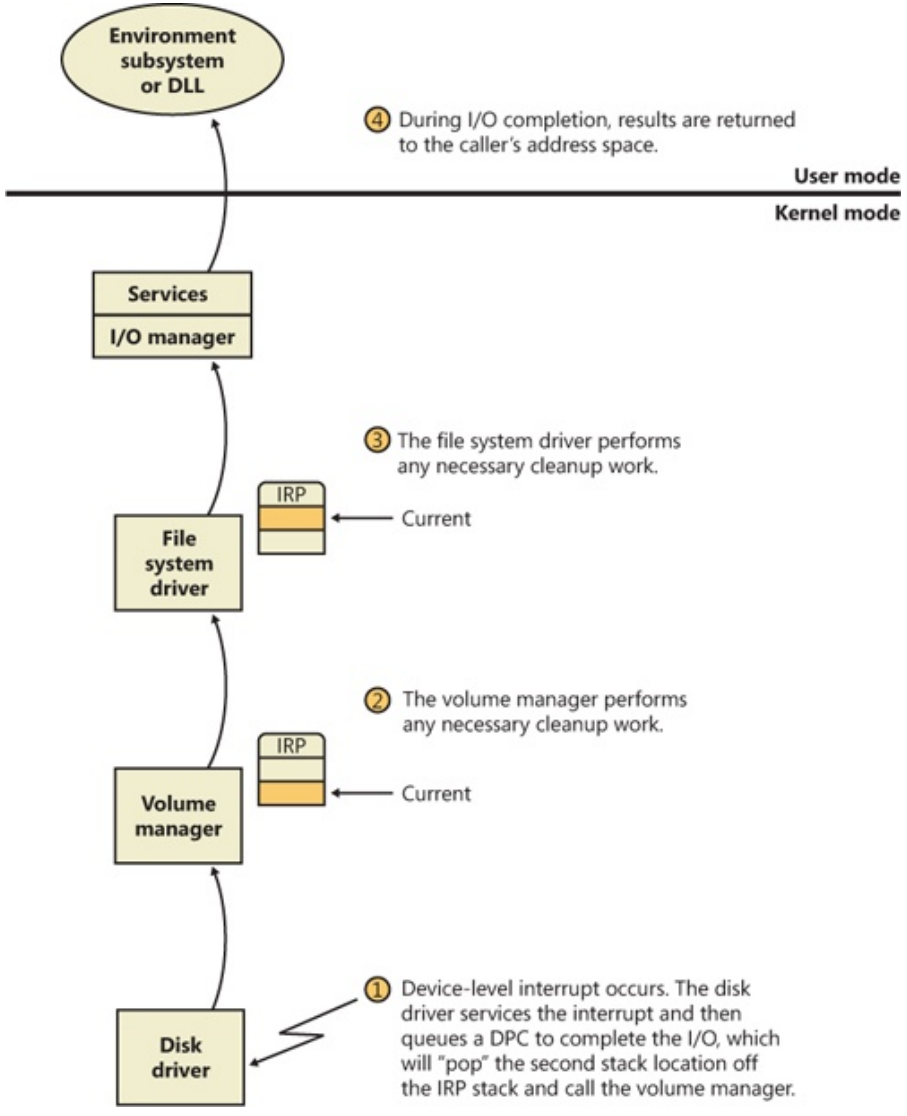
Komunikace mezi ovladači v jádře Windows

- Ovladače ve Windows nepoužívají přímé volání funkcí, ale komunikují pomocí předávání zpráv
- Windows Driver Model je navržen tak, aby bylo teoreticky možné pouštět ovladače v oddělených adresních prostorech
- Kvůli rychlosti je ale běží ovladačů v jednom monolitickém adresním prostoru jádra.
- Zprávy, které si ovladače vyměňují se nazývají **I/O request packet (IRP)**

Cesta IRP jádrem



- IRP se alokuje jen jednou
- Každý ovladač „po cestě“ má svůj slot
- File system vyplní slot pro volume manegr a pošle IRP dál.
- Po dokončení požadavku se IRP „cestuje“ zpět (obr.



Source: Microsoft

Ovladače v uživatelském prostoru

- Chyba v ovladači může způsobit pád systému
- Nekvalitní ovladače jsou také zdrojem mnoha bezpečnostních problémů
- Ovladače v uživatelském prostoru:
  - Podporovány jak Linuxem (UIO) tak Windows
  - Spouštěny jako běžná aplikace
  - Přístup k registrům HW: mmap()
  - Obsluha přerušení – OS upozorní aplikaci pokud nastalo přerušení
    - UIO:

```
int uio = open("/dev/uio0", ...);
read(uio, ...); // waits for interrupt
handle_interrupt();
```

    - Při chybě ovladače ho lze jednoduše restartovat
    - Ostatní aplikace nevolají ovladač pomocí systémových volání, ale pomocí meziprocesní komunikace (např. fronty zpráv)
- OS založené na mikrojádře mají (téměř) všechny ovladače v uživatelském prostoru