Principy počítačů a operačních systémů

Operační systém
a jeho role v počítači

Zimní semestr 2011/2012

Poděkování

Přípravě této prezentace jsem převzal a přeložil velké množství materiálu z prezentace

- Roth, A., Martin, M. **CIS 371 – Computer Organization an Design**. University of Pennsylvania, Dept. of Computer and Information Science, Spring 2009.

Dále jsem převzal část materiálu z prezentace

- Yaghob, J. **Základy operačních systémů**. Katedra SW inženýrství, Matematicko-fyzikální fakulta, Univerzita Karlova v Praze, 2007.

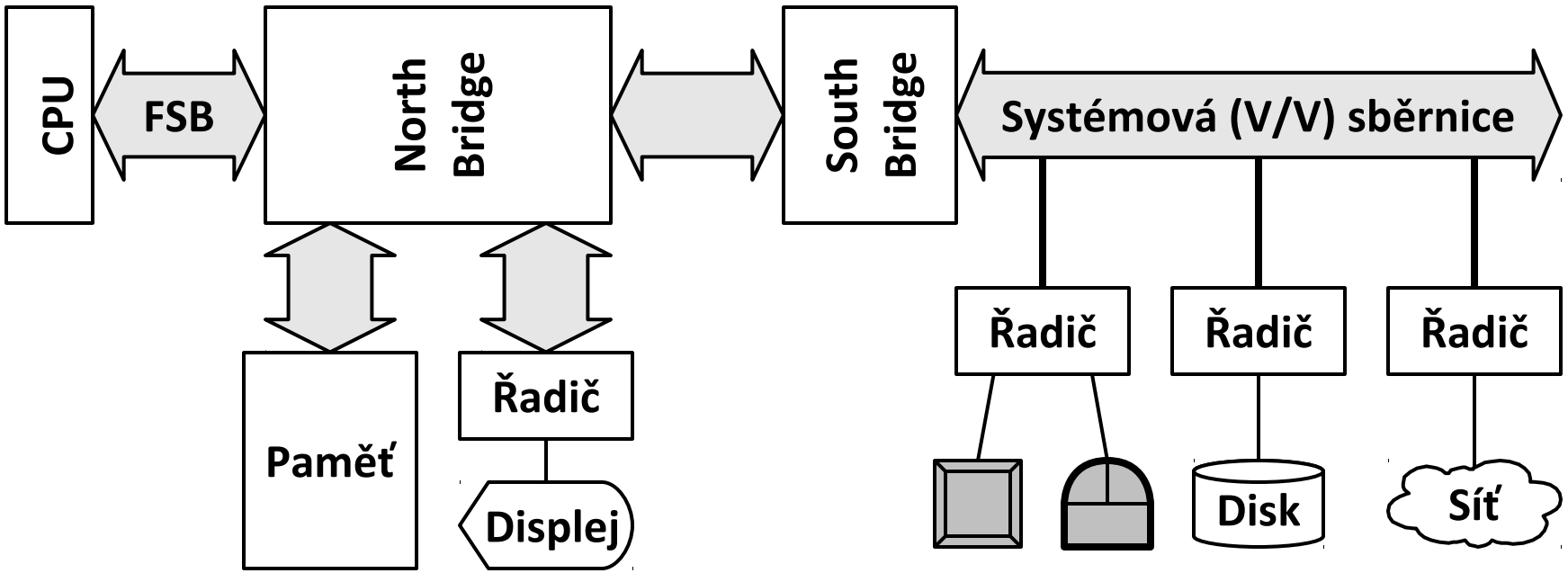
Motivace

Proč potřebujeme operační systém?

Počítač: hardware

Hardware

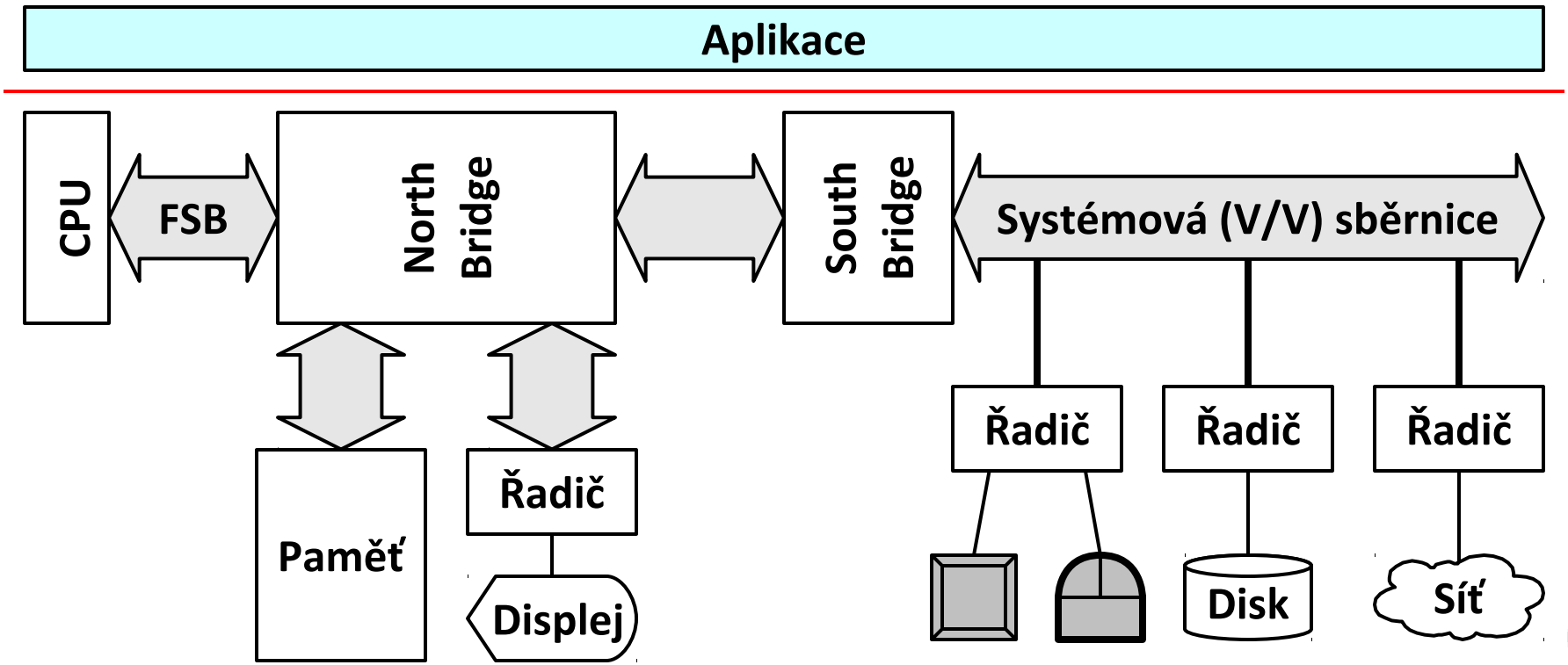
- procesor
- paměť
- čipová sada (chipset)
- periferní zařízení (vstup/výstup, úložiště dat, síť)



Počítač: + aplikační software

Počítač musí také něco užitečného dělat

- bez aplikací je počítač jen hromádka "železa"



Aplikace a holé "železo"

Programový model

- aplikace má paměť celou pro sebe
  - co když se data nevejdou do paměti?
  - co když se kód aplikace nevejde do paměti?
- aplikace má procesor celý pro sebe
- musí umět komunikovat se všemi zařízeními (všech výrobců)
  - čipová sada, klávesnice, myš, disk, grafická karta, síťová karta

Jak se aplikace dostane do paměti?

- aplikace pro načítání aplikací
  - sama musí být načtena pomocí programu v ROM (BIOS)

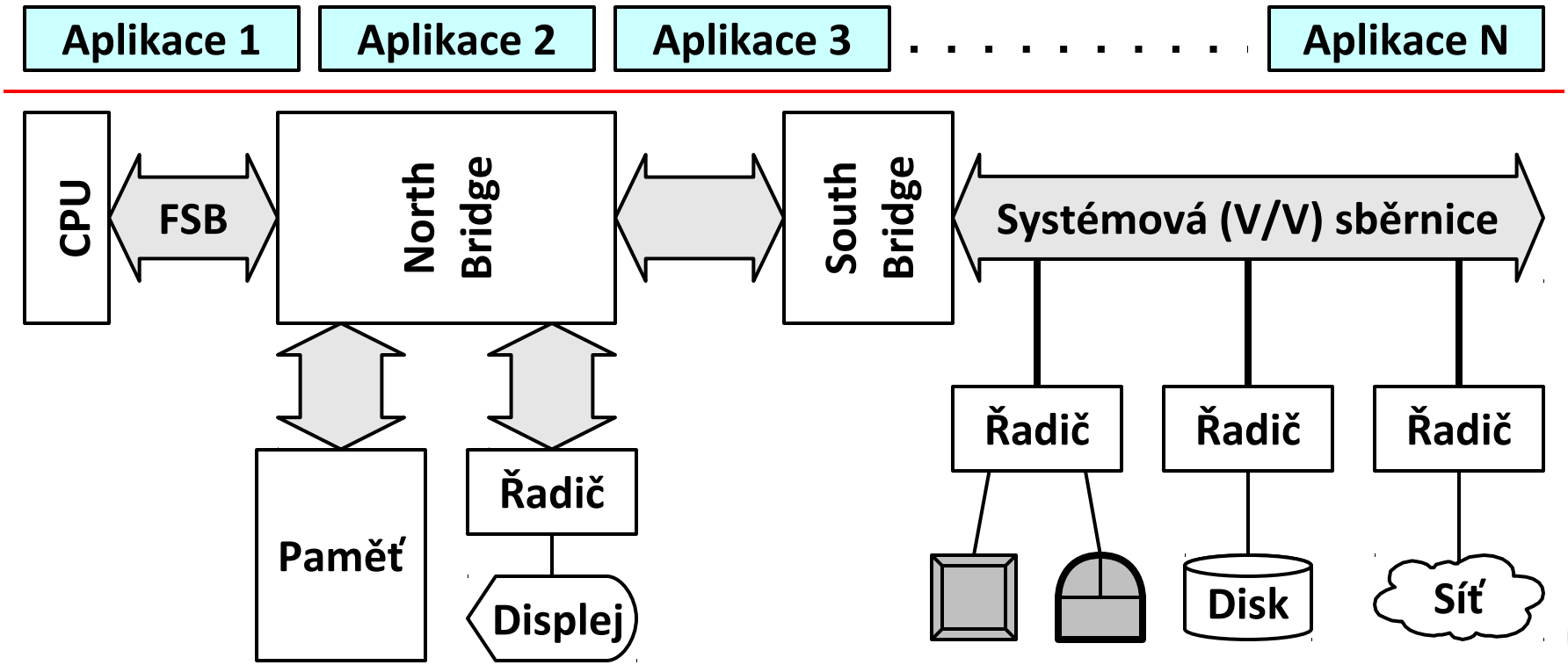
Co když chci při práci poslouchat hudbu?

- ukončím textový editor, na chvilku pustím přehrávač hudby, po chvíli ho ukončím a spustím textový editor, po chvíli...

Počítač: + aplikační software

Když chci při práci poslouchat hudbu...

- musí běžet více aplikací současně



Více aplikací a holé "železo"

Programový model

- více aplikací sdílí jednu paměť
  - kam do paměti se má/může aplikace nahrát?
  - co když jedna aplikace zapíše do paměti, kde jsou data/kód jiné aplikace?
  - co když je paměti málo?
- více aplikací sdílí jeden procesor
  - musejí se vzájemně volat, aby mohly běžet (kooperativní multitasking)
- všechny aplikace × všechna zařízení × všichni výrobci
  - nebudou si aplikace "kecat" do komunikace se zařízeními?

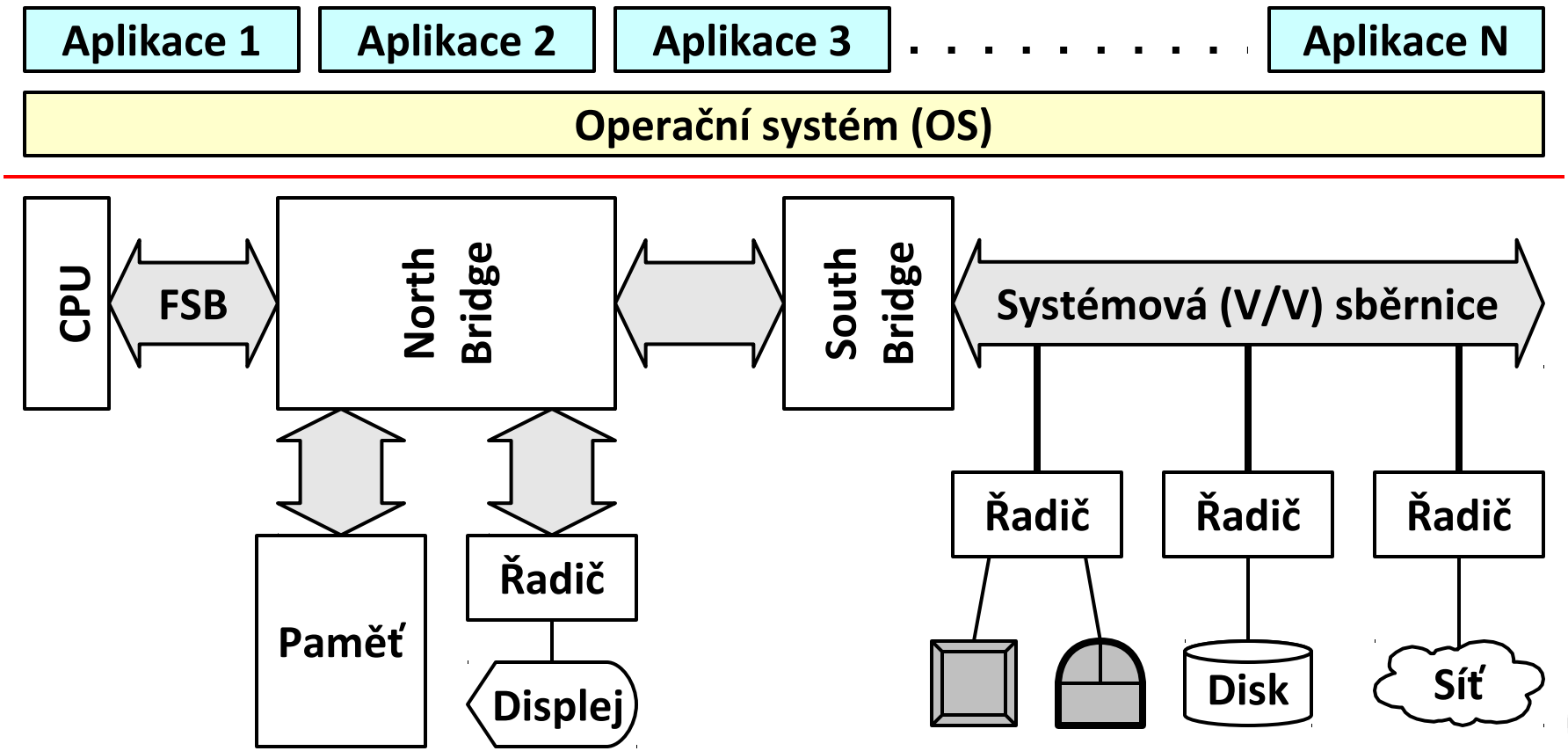
Co když chci při práci poslouchat hudbu?

- textový editor musí každou chvilku zavolat nějakou funkci v přehrávači hudby, ta přehraje úsek skladby a vrátí se zpět
  - jak ví textový editor, že je přehrávač vůbec v paměti a kde?
  - jak ví textový editor, kterou funkci volat a jak?

Počítač: + operační systém (OS)

Aby se programátoři (a uživatelé) nezbláznili

- OS odstíní aplikace od nutnosti "znát" zařízení
- OS umožní více aplikacím běžet na jednom "železe"



Více aplikací + operační systém

Programový model

- každá aplikace má "celou" paměť jen pro sebe
  - do paměti jiných aplikací nemůže, nemůže jim tedy uškodit
  - paměti je "neomezeně" mnoho (omezení velikostí adres počítače)
- každá aplikace má procesor jen pro sebe
  - procesor vykonává její instrukce, o ostatní aplikace se nestará
- pro práci se zařízeními používá abstrakce
  - streamy, soubory, síťové sockety

Co když chci při práci poslouchat hudbu?

- textový editor a přehrávač běží "současně"

Abstrakce a systémová volání

The purpose of abstraction *IS NOT* to be vague, but to create a new semantic level in which one can be absolutely precise.

E. W. Dijkstra

"The Humble Programmer", 1972

OS poskytuje abstrakci

Skrytí složitosti HW rozhraní

- soubor, adresář, síťový socket, IP adresa, stream, datagram, datum, čas, ...
  - blok na disku určen číslem stopy, povrchu a sektoru, číst/zapisovat je možné posloupnost sektorů
  - síťová karta umí vysílat/přijmat pouze rámce (bloky dat) obsahující linkové adresy a data
- nezávislé na konkrétní HW platformě

Systémová volání

- služby pro přístup k poskytovaným prostředkům
  - pro každý typ prostředku definované rozhraní
- podobné volání knihovní funkce v programu
  - proč není operační systém knihovna?

Co je to systémové volání?

Systémové volání = volání funkce OS

- aplikace předá řízení operačnímu systému pomocí speciální instrukce/posloupnosti instrukcí
  - v registrech procesoru (a na zásobníku) je předán "popis" požadavku (číslo funkce a její parametry)

MIPS

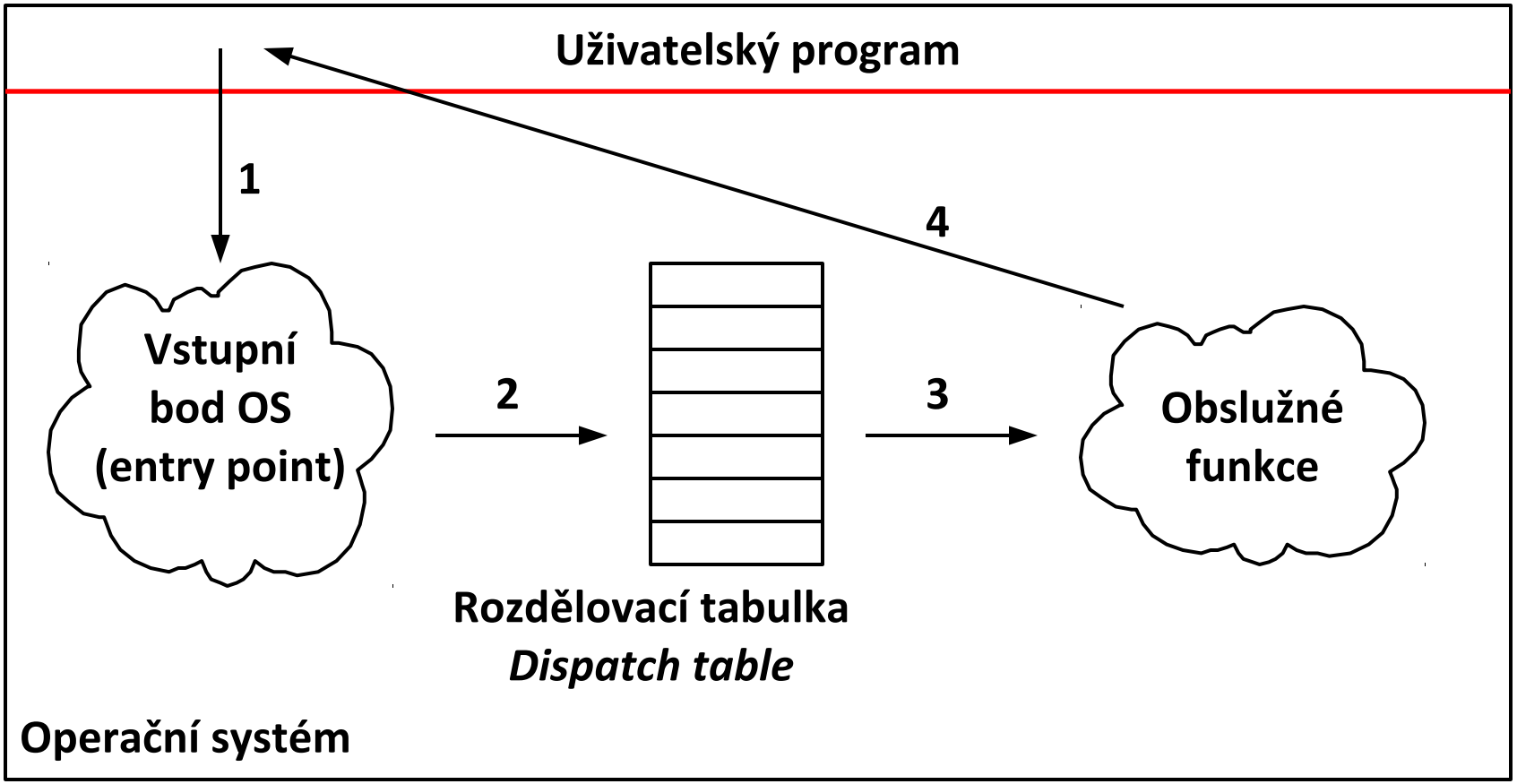
```
add $a0, $t0, $zero
li  $a1, 1
syscall 1
```

x86_64

```
mov eax, 101h
mov edx, esp
sysenter
```

```
movzwl 12(%esp),%eax
movl %eax,4(%esp)
movl $23,%eax
movl 4(%esp),%ebx
int $0x80
```

Obsluha systémových volání



Proč jsou potřeba speciální instrukce?

Běžné volání funkce v programu

- překladač zná adresy funkcí v programu, při volání vygeneruje instrukci pro volání funkce s přímou nebo nepřímou adresou

Jak se liší systémové volání od volání funkce?

- aplikace neví, na kterých adresách leží OS a jeho funkce
  - aplikace nemůže jen tak volat libovolný kód OS
  - OS je v někde v paměti, ale není to knihovna spojená s aplikací
- neznámá adresa ⇒ speciální mechanizmus volání
  - způsobí výjimku (*trap*), při které procesor skočí na (aplikaci neznámou) adresu, kde má OS svou obslužnou rutinu
- mechanizmus volání způsobí přepnutí režimu procesoru
  - kód OS běží v režimu s vyšší úrovní oprávnění
  - systémové volání umožňuje přechod přes hranici mezi uživatelským programem a operačním systémem

K čemu jsou nutná oprávnění?

Přístup k HW z jednoho místa – OS

- žádná aplikace nesmí přímo komunikovat s HW
- v robustním systému nelze spoléhat na "čestné" slovo
  - HW podpora pro zajištění exkluzivity přístupu pro OS

Uživatelský režim procesoru

- procesor vykonává instrukce (aplikace), pokud narazí na privilegovanou instrukci, vyvolá výjimku
  - přístup do paměti, kam jsou mapována zařízení, popř. instrukce in/out
  - další instrukce nutné ke správné funkci OS (později)

Privilegovaný režim procesoru

- pro běh OS – instrukce vykonávány bez omezení
- při návratu do uživatelského programu zajistí OS přepnutí procesoru zpět do uživatelského režimu

Izolace aplikací pomocí virtualizace

Vlastní procesor, paměť a zařízení
~~do každé rodiny~~ pro každou aplikaci

Vzájemná izolace aplikací

Zjednodušuje programovací model

- iluze vlastní paměti, procesoru a zařízení
- programátor nemusí explicitně řešit sdílení

Zvyšuje bezpečnost systému

- aplikace si nemohou škodit (úmyslně či neúmyslně)

Zvyšuje míru využití prostředků

- OS optimalizuje přidělování prostředků

Jak souvisí izolace s virtualizací?

- poskytuje iluzi "soukromých" výpočetních prostředků

Virtualizace procesoru

Jak mohou aplikace (a OS) sdílet procesor(y)?

- cíl: aplikace si myslí, že systém má neomezený počet procesorů

Řešení: sdílení procesoru v čase (time sharing)

- **přepínání kontextu** (*context switch*) v pravidelných intervalech
  - **preemptivní** – aplikace "neuvolňuje" procesor, OS si ho vezme "násilím", což znemožňuje aplikacím si procesor "uzurpovat" pro sebe
- kontext architektury: programový čítač (PC), registry procesoru
  - uložen/obnoven při každém přepnutí kontextu
  - co obsah paměti?
- kontext implementace: cache, prediktory skoků, ...
  - při přepnutí kontextu se ignoruje nebo vynuluje (*flush*)

Operační systém musí zajistit přepínání kontextu

- jediná HW podpora je přerušení od časovače

Virtualizace paměti

Jak mohou aplikace (a OS) sdílet paměť?

- cíl: aplikace si myslí, že systém má neomezené množství paměti

Aplikace může požadovat více paměti než je v systému...

- rozsah kódu/dat aplikace může být větší než hlavní paměť
- hlavní paměť se musí chovat jako cache
  - (pomalý) disk představuje pamět v následující úrovni hierarchie
  - velké bloky (stránky), write-back, write-allocate, LRU (aproximace)

Řešení

- 1. část: zacházet s pamětí jako s cache
  - data, co se nevejdou do paměti se ukládají na disk (*swap file*)
- 2. část: přidáme úroveň indirekce (překlad adres)
  - aplikace pracjí s jinými adresami než hlavní paměť

Virtualizace zařízení

Jak mohou aplikace sdílet zařízení?

- cíl: aplikace si myslí, že mají zařízení jen pro sebe

Řešení: opravdové sdílení, pokud je to možné

- speciální soubory, které je možné otevřít ve více aplikacích
  - znakové zařízení místo klávesnice, myši (umožňují číst data ze zařízení ve více aplikacích) nebo např. zvukové karty (čtení i zápis dat ve více aplikacích, HW při zápisu mixuje zvuk z více zdrojů)

Řešení: posun v úrovni abstrakce

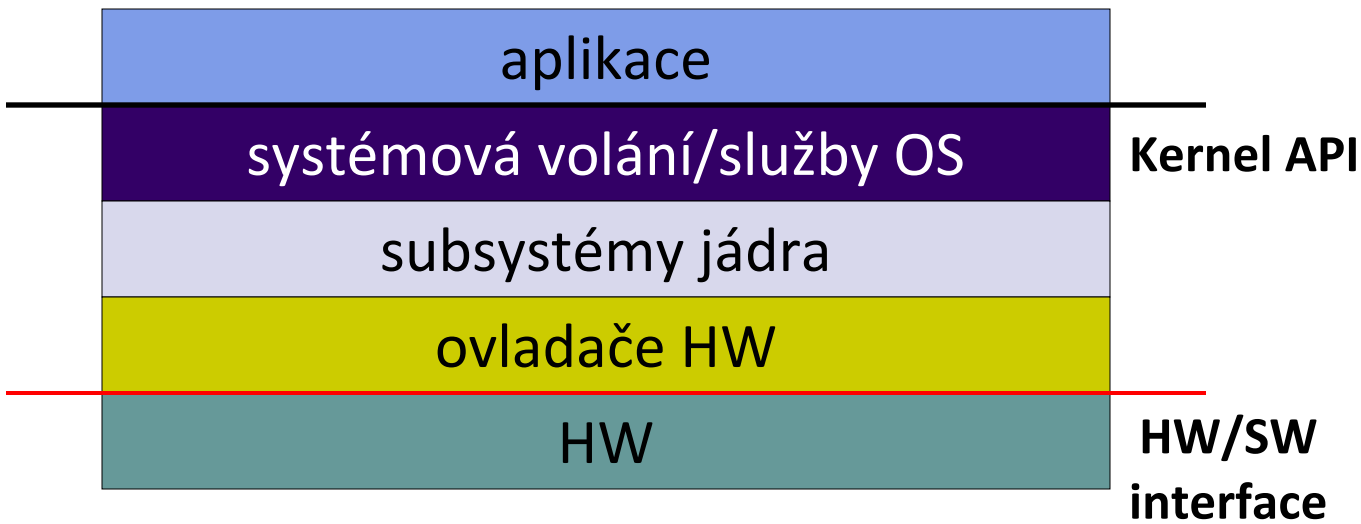
- místo zařízení OS poskytne aplikaci objekt s jiným rozhraním a operace s objektem převádí na operace se zařízením
  - soubory a adresáře místo disku (přístup ke stejným souborům z více aplikací už je jiný problém), síťové sokety místo síťové karty
  - tisková služba (na úrovni dokumentů) místo tiskárny (*spooling*)

Architektura OS

Hlavní část OS tvoří jádro (kernel)

Jádro je také (jenom) program...

- vytvořeno běžnými postupy pro návrh a tvorbu složitého SW
  - jenom chyby jsou více fatální než u aplikačních programů
- subsystémy pro různé funkce poskytované OS
  - rozdělení zodpovědnosti, zjednodušení návrhu
- různá oprávnění pro různé subsystémy
  - zvýšená odolnost proti "zlým" programům



Vybrané subsystémy OS

Správa paměti

- spravuje fyzickou paměť a implementuje virtuální paměť s využitím HW podpory procesoru

Správa a plánování procesů

- poskytuje abstrakci pro běžící aplikace ve formě procesů a vláken a plánuje jejich běh na sdíleném procesoru

Souborový systém

- poskytuje abstrakci pro ukládání dat ve formě souborů a adresářů, převádí operace se soubory na operace s diskem

Síťový subsystém

- poskytuje abstrakci pro komunikaci po síti ve formě síťových soketů, streamů a datagramů, převádí operace se sokety na operace se síťovou kartou

Ovladače zařízení

Low-level komunikace se zařízeními

- vyřizuje požadavky vlastních subsystémů (souborový systém, síťové sokety, ...) na zařízení
  - konfiguruje zařízení, posílá příkazy, čte stav, přenáší data
- používá instrukce pro čtení/zápis do (necachované) paměti nebo speciální instrukce pro vstup a výstup (in/out)
  - HW tyto přístupy přenáší po sběrnici k zařízení na cílové adrese

Konfigurace zařízení

- způsob signalizace stavu a přenosu dat
  - polling vs. interrupt driven I/O (číslo přerušení)
  - programmed I/O vs. DMA (busmastering, scatter/gather)

Obsluha výjimek a přerušení

Výjimky (*exceptions, traps*)

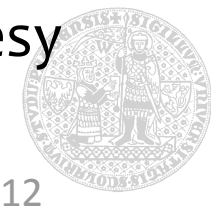
- synchronní události, způsobené běžící aplikací
  - neplatná instrukce, dělení nulou, systémové volání (trap), ...

Přerušení (*interrupts*)

- asynchronní události generované mimo procesor
  - časovač, I/O požadavek nebo odpověď, ...

Mechanizmus obsluhy stejný pro výjimky i přerušení

- při výjimce nebo přerušení procesor skočí na obslužnou rutinu (adresa či výběr adresy určen architekturou)
- OS uloží stav (registry) přerušeného programu (na zásobník), analyzuje příčinu výjimky/přerušení a zavolá obslužnou funkci
- OS obnoví stav přerušeného programu a vrátí se zpět tak, aby vykonávání programu pokračovalo od stejné/následující adresy



Architektura OS

Běhové uspořádání OS

- jeden program – monolitické jádro
  - komunikace pomocí volání funkcí
- minimální jádro + sada programů (subsystémy)
  - komunikace pomocí zpráv

Dilema při volbě architektury

- izolace subsystémů vs. efektivita OS
  - vyšší režie při komunikaci mezi izolovanými subsystémy
  - souvisí s HW latencí při změně režimu procesoru
- kompromis mezi odolností a efektivitou
  - základní typy architektur operačních systémů

Monolitická architektura OS

Jádro OS tvořeno jedním programem

- hlavní důraz na efektivitu
- všechny subsystémy mají stejná oprávnění
  - nízká režie na komunikaci mezi subsystémy
  - nevylučuje modulární design
- uživatelské programy mají minimální oprávnění
  - speciální instrukce na systémová volání mění oprávnění
  - nepoužívá se u "zavaděčů" typu CP/M, MS-DOS

Běžně používané systémy

- Linux, Solaris, Windows, ...

Mikrojádrová architektura OS

Minimální jádro + sada programů

- důraz na bezpečnost a robustnost
- subsystémy jako oddělené aplikace (servery)
  - subsystémy mají pouze minimální oprávnění
  - příliš se neliší od uživatelských programů
- jádro poskytuje pouze nejnutnější funkce
  - obsluha přerušení, komunikace mezi programy
  - při komunikaci je zajištěna kontrola oprávnění

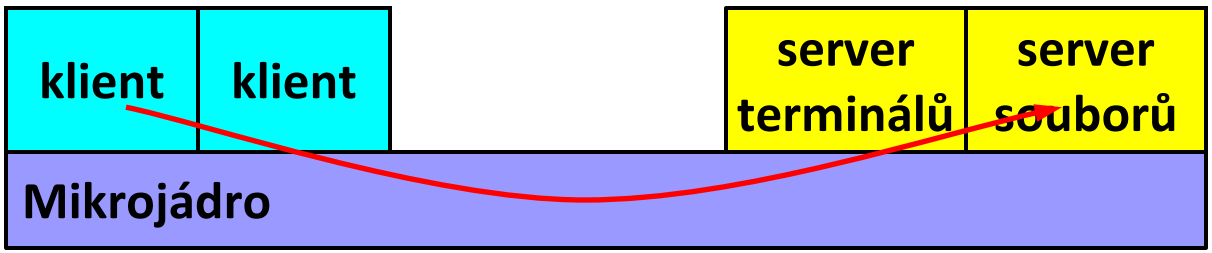
Výzkumné a embedded systémy

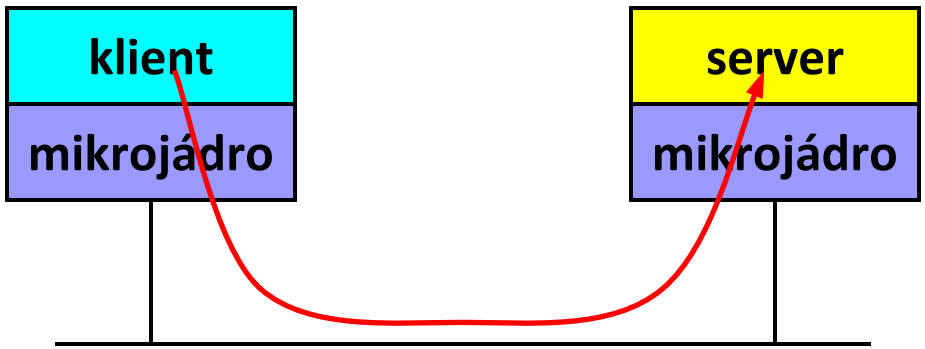
- MACH, Spring, Minix, ..., QNX, L4, Fiasco, ...

Volání služeb OS (i po síti)

Volání služby = zaslání zprávy

- zprávy mohou transparentně cestovat po síti
  - na rozdíl od lokálního přenosu mohou mít výrazně vyšší latenci, s címž nemusí program počítat





Shrnutí

Operační systém vs. aplikační program

Operační systém: privilegovaný program

- spravuje a přiděluje prostředky aplikacím
  - má přímý přístup k alokačním mechanizmům
- ví o všech běžících aplikacích
- přímo komunikuje se zařízeními (ovladače)
- ví o HW detailech nutných pro implementaci OS

Aplikační program: sladká nevědomost

- neví nic o podrobnostech HW
- neví nic o ostatních programech (a OS)
- alokace prostředků a přístup k zařízením pouze skrz OS

Role operačního systému

Abstrakce (*extended machine*)

- odděluje aplikace od low-level platformy
- poskytuje abstrakce nezávislé na hardware

Izolace (*isolation through virtualization*)

- OS virtualizuje procesor, paměť a zařízení
- virtualizace umožňuje vzájemnou izolaci aplikací

Správa prostředků (*resource manager*)

- OS přiděluje a řídí sdílení prostředků
  - CPU, RAM, přenosové pásmo, disková kapacita, ...
- maximální výkon systému jako celku při zohlednění potřeb jednotlivých aplikací