

# ZOS – OPAKOVÁNÍ

L. Pešička



# ZÁKLADNÍ PRAVIDLO

Důležité je znát nejen **fakta**,  
ale **porozumět** jim  
a zasadit je do kontextu celého OS

Př.

algoritmus Second Chance využívá bitu Referenced  
tak, že ... (**fakta**)

a kdy Second Chance použijeme? (**porozumění**)  
když je RAM plná a potřebujeme uvolnit rámec pro  
stránku, kterou potřebujeme mít v paměti



# OS – PROSTŘEDÍ PRO BĚH UŽIVATELSKÝCH APLIKACÍ

**Uživatelská aplikace (proces) nemá přímý přístup k HW.**

Proč? Co by se mohlo stát?

mohla by třeba smazat celý disk, i s OS

Jak ale tedy zařídit např. přístup k disku?

**systemovým voláním**

- **požádám** OS, aby pro mě akci vykonal
- OS **zkontroluje**, zda na to mám **oprávnění** a pokud ano, tak ji **vykoná**, nestane se tedy nic nepatřičného



# SYSTÉMOVÉ VOLÁNÍ

- někam, typicky do registrů uložím:
  - jakou službu OS chci volat (číslo služby)
  - jaké další parametry musím k dané službě uvést (chci-li otevřít soubor, musím říct jaký, chci-li zobrazit řetězec, musím říct na jaké adrese začíná a jak je dlouhý)
- požádám o systémové volání:
  - sysenter
  - nebo INT 0x80
- systém se následně přepne do privilegovaného režimu a jádro vykoná požadovanou službu



# VEKTOR PŘERUŠENÍ

motivace:

Stisknu klávesu na klávesnici, jak systém pozná, jaký podprogram má vyvolat?

od HW zařízení vede **drát** do řadiče přerušení  
jak z drátu číslo 5 pozná, že chceme vykonat  
podprogram na adrese 20100 jako obsluhu přerušení?

- ⇒ vektor přerušení – představuje pole
- ⇒ index do pole – číslo drátu
- ⇒ obsah pole – adresa podprogramu
- ⇒ tedy zde `vektor_preruseni[5] = 20100`



# JAKÁ HW ZAŘÍZENÍ SE TYPICKY POUŽÍVAJÍ A GENERUJÍ PŘERUŠENÍ?

- časovač
  - každý tick časovače – obsluha přerušení (navýší počet tiků, určí, zda není potřeba např. pustit plánovač a přeplánovat proces)
- klávesnice
  - zadáme znak na klávesnici a potřebujeme, aby jej někdo zpracoval
- síťová karta
  - přišel packet s daty, je třeba jej zpracovat
- disk
  - mám data, o která bylo žádáno



# PROČ JE DŮLEŽITÉ CHRÁNIT VEKTOR PŘERUŠENÍ?

- s hodnotami ve vektoru přerušení může manipulovat jen systém..
- jak by ho mohl využít virus?

např. na indexu 5 budeme mít adresu podprogramu, který se vykoná při vstupu z klávesnice, např. 20 100

kdyby se škodlivému viru povedlo změnit na indexu 5 na hodnotu např. 100 000, kde je náš škodlivý program, který uloží kód klávesy někam a na konci by zavolał původní kód na adrese 20 100



# CO ZNAMENÁ INT x?

- instrukce v assembleru pro x86 procesory, která generuje sw přerušeni
- x je v rozsahu 0 až 255
- paměť od 0 je 256 4bytových ukazatelů (celkem 1KB), říkají kam skočit v paměti pro podprogram obsluhující přerušeni – vektor přerušeni
- HW interputy jsou mapovány na dané vektory prostřednictvím **programovatelného řadiče přerušeni**
- INT 0x80
  - v 16kové soustavě 80, dekadicky 128
  - pro vykonání systémového volání
  - do 32bit EAX registru se dá číslo funkce, kterou chceme





# PREEMPTIVNÍ X NEPREEMPTIVNÍ

- pokud je proces **nepreemptivní**, nemůžeme ho přerušit a vystřídat jiným procesem, když on nechce, tedy v nevhodný okamžik – uvnitř kritické sekce
- většina systémů dnes používá preemptivní procesy – musíme řešit kritické sekce
- zatímco uživatelské procesy jsou preemptivní, jádro OS může být nepreemptivní (kooperativní), novější jádra např. Linuxu můžeme zkompilovat i jako preemptivní (je třeba lépe uvnitř ošetřit)



# TYPY JÁDRA OS

- monolitické: příklad Linux
  - souborový systém – typicky v jádře
  - vliv na stabilitu celého systému
  - ale může být fs i v userspace – FUSE
- mikrojádro: příklad GNU/Hurd
  - kolekce serverů běží na mikrojádro GNU/Hurd nebo L4
  - servery: souborový systém, síť, ...
- hybridní: Microsoft Windows



# INFO O SYSTÉMECH (WIKIPEDIA)

## Windows 7

<b>Web:</b>	<a href="#">Windows 7</a>
<b>Vyvíjí:</b>	Microsoft
<b>Rodina OS:</b>	Windows NT
<b>Druh:</b>	Uzavřený vývoj
<b>Aktuální verze:</b>	Service pack 1 SP1 / 15.3.2011
<b>Způsob aktualizace:</b>	<a href="#">Windows Update</a>
<b>Správce balíčků:</b>	<a href="#">Windows Installer</a>
<b>Podporované platformy:</b>	x86, x86_64
<b>Typ kernelu:</b>	Hybridní jádro
<b>Implicitní uživatelské rozhraní:</b>	Grafické uživatelské rozhraní
<b>Licence:</b>	Microsoft EULA
<b>Stav:</b>	finální verze

## Linux



<b>Rodina OS:</b>	Unix-like
<b>Aktuální verze:</b>	3.2 / 4. ledna 2012
<b>Podporované platformy:</b>	IA-32, x86-64, PowerPC, ARM, m68k, DEC Alpha, SPARC, hppa, IA-64, MIPS, s390 a další
<b>Typ kernelu:</b>	Monolitické jádro
<b>Implicitní uživatelské rozhraní:</b>	GNOME, KDE, Xfce a jiné
<b>Licence:</b>	GNU GPL a jiné
<b>Stav:</b>	Aktuální



# PŘÍSTUPOVÁ PRÁVA

## ○ základní Unixová

- u, g, o, .. a
- r, w, x
- `chmod 777 ahoj.txt`
- `chown`

## ○ ACL

- daleko komplexnější, např. NTFS (včetně např. dědění)
- seznam spjatý s každým souborem

User/group	name	rights
user	pepa	Read, write
user	tom	read
group	students	read



# IDENTIFIKÁTORY

- PID            id procesu, getpid()
- PPID          id rodiče, getppid()
- TID            id vlákna
- UID            id uživatele

příkazy: id, ps



# NOVÝ PROCES VYKONÁVAJÍCÍ NĚJAKÝ PROGRAM

- Windows:

  - CreateProcess()**

    - proces bude vykonávat kód dle uvedeného programu

- Linux:

  - kombinace **fork()** a **exec()**

    - fork – nový proces (nový PID), ale stejný kód, liší se jen návratovým kódem forku

    - exec – nahradí kód, který daný proces vykonává

(exec většinou tvar `execv`, `execve` aj.)



# JAK FUNGUJE SHELL?

- shell – příkazový interpret, známe `/bin/bash`
- shell čeká na zadání příkazu
- shell se naforkuje
- co mu zadáme na příkazovou řádku pustí execem
- čeká na dokončení tohoto potomka (není-li `&`)
- a pak se cyklus znovu opakuje



# SYSTEMY S ČASOVÝM KVANTEM

Jakou vlastnost potřebuje mít operační systém, aby mohl plánovat procesy po časových kvantech?

⇒ musí mít časovač

každý tick časovače – hw přerušování (drát, přerušovací vektor, obsluha přerušování)

v rámci obsluhy přerušování zvýší počítadlo tiků

po určitém počtu tiků – uplynulo kvantum –  
plánovač zjistí kdo dál poběží – dispatcher přepne kontexty procesů





## I-UZLY

- jeden i-uzel odpovídá jednomu souboru
- název souboru není součástí i-uzlu
- v adresáři je přiřazení (název souboru, číslo i-uzlu)

mějme v adresáři následující položky:

ahoj.txt 60

cau.txt 60

kuk.txt 61

ahoj.txt, cau.txt – hardlinky (příkaz ln) na stejný i-uzel, tedy na stejný obsah souboru



# FAT TABULKA

disk – posloupnost datových bloků

rozdělení disku:

- boot sektor
- alokační tabulka souborů (FAT)
  - obvykle 2 kopie hned za sebou
- Kořenový adresář
  - kořen hierarchie adresářů



# NTFS

- fragmenty
- chceme se přiblížit kontinuální alokaci, která je nejvýhodnější
- v ideálním případě 1 soubor – 1 fragment

pojem defragmentace disku



- u FAT, NTFS





# DOPORUČENÍ

- courseware – prezentace přednášek
- courseware – nahlédnout i KIV/OS, příp. KIV/PPR
- courseware – opakování
- portál s otázkami (<http://students.kiv.zcu.cz/teri/>)

## OPAKOVÁNÍ !!

 [Opakování základních znalostí](#) (797KB) 

 [Semaforey opakování](#) (604KB) 

 [PametProces precist](#) (116KB) 

### Doporučuji:

[Portál s otázkami na procvičení](#) (0KB)

