Opakování ZOS 2016

L. Pešička

Verze: 12. ledna 2016, 16:30

Průběžně upravováno

Obsah

- Úvod
- Systémové volání
- Přerušení

Důležité

- Přečtete si také opakování 2015
- Jsou tam důležité věci a nemá smysl je znovu opisovat

Systémové volání

Definice:

- **Mechanismus používaný aplikacemi k volání služeb operačního systému.**

Důvod:

- V uživatelském režimu není možné celou řadu věcí vykonat – není přímý přístup k HW, nelze tedy přímo otevřít soubor, číst z něj a zapisovat do něj.
- Pokud aplikace takovou činnost požaduje, nezbývá jí, než požádat o danou službu operační systém.
- Operační systém zkontroluje, zda má aplikace pro danou činnost oprávnění a pokud ano, požadovanou činnost vykoná. (Kontrola může být např. podle ACL, zda má uživatel procesu právo zapisovat do souboru).

Realizace systémového volání

Dvě možnosti:

- Softwarové přerušení – instrukce INT číslo, např. INT 0x80
- Speciální instrukce CPU – sysenter aj.

Realizace systémové volání SW přerušením

1. MOV EAX, číslo služby
2. MOV EBX, další parametry…
3. INT 0x80

1. Do registru EAX dáme číslo požadované služby
2. Do dalších registrů další potřebné parametry (liší se službu od služby)
3. INT 0x80 – instrukce pro sw přerušení
   (V rámci sw přerušení se CPU přepne do privilegovaného režimu)

Jak zjistím, jaké služby jsou k dispozici?

Např. na http://syscalls.kernelgrok.com/ pro Linux

- Je zde vidět číslo volání (dáme do registru EAX)
- Požadavané parametry co uvést v dalších registrech

Linux Syscall Reference

Show All entries Search:

#	Name	Registers						Definition
		eax	ebx	ecx	edx	esi	edi	
0	sys_restart_syscall	0x00	-	-	-	-	-	kernel/signal.c:2058
1	sys_exit	0x01	int error_code	-	-	-	-	kernel/exit.c:1046
2	sys_fork	0x02	struct pt_regs *	-	-	-	-	arch/alpha/kernel/entry.S:716
3	sys_read	0x03	unsigned int fd	char __user *buf	size_t count	-	-	fs/read_write.c:391
4	sys_write	0x04	unsigned int fd	const char __user *buf	size_t count	-	-	fs/read_write.c:408
5	sys_open	0x05	const char __user *filename	int flags	int mode	-	-	fs/open.c:900
6	sys_close	0x06	unsigned int fd	-	-	-	-	fs/open.c:969
7	sys_waitpid	0x07	pid_t pid	int __user *stat_addr	int options	-	-	kernel/exit.c:1771
8	sys_creat	0x08	const char __user *pathname	int mode	-	-	-	fs/open.c:933

Není to nepohodlné, pamatovat si čísla služeb?

- Ano, je.
- Proto jsou k dispozici funkce, které „obalí“ jednotlivá systémová volání, takže si nemusíme pamatovat jejich čísla.

Např:

getpid(), open(), fork(), creat(), execve()

V Linuxu – nápověda sekce 2, systémová volání:

man 2 open, man 2 fork, man 2 execve

Knihovní funkce

- Někdy se nevolá systémové volání přímo.
- Aplikace zavolání **knihovní funkci** a teprve ona zavolá systémové volání
- Výhoda – skryje podrobnosti o systémovém volání na různých platformách, může provést další užitečné činnosti.

- Např. volání fopen() z stdio.h
- Linux: man 2 open x map 3 fopen

Systémové volání – práce s procesy

Pamatujte si:

volání	popis
fork()	Vytvoří nový proces
wait()	Čeká na dokončení procesu
waitpid(id)	Čeká na dokončení konkrétního procesu
_exit()	Ukončení procesu
execve()	Spustí jiný program v rámci aktuálního procesu

Systémové volání – práce se soubory

volání	popis
creat()	Vytvoří soubor
open()	Otevře soubor v požadovaném režimu (čtení, zápis)
close()	Uzavře soubor
read()	Čtení ze souboru
write()	Zápis do souboru
lseek()	Posune ukazovátko v souboru – pro přímý přístup (z libovolné pozice)

Soubory sekvenční – můžeme číst a zapisovat jen postupně (nemají lseek)
Soubory s přímým přístupem – čtení a zápis na libovolnou pozici v souboru (lseek)

Dnes většina souborů s přímým přístupem, tj. můžeme se v něm posouvat.
Systémové volání open() vs. knihovní funkce fopen().

Systémové volání – práce s pamětí

volání	popis
mmap()	Paměťově mapovaný soubor
munmap()	Odstraní mapování paměťově mapovaného souboru
mlock()	Zamkne paměťovou stránku v paměti, aby nemohla být stránkována do swapu
munlock()	Odemkne stránky v daném adresním rozsahu
brk(), sbrk()	Nastavení konce datového segmentu

Systémové volání – práce se zprávami

název	popis
msgsnd()	Poslání zprávy
msgrcv()	Příjem zprávy

Důležitý pojem IPC – InterProcess Communication
Meziprocesová komunikace:

- Zasílání zpráv
- Sdílená paměť
- roury

Přerušení

Definice:

Metoda pro (asynchronní) obsluhu událostí, kdy procesor přeruší vykonávání sledu instrukcí, vykoná obsluhu přerušení a pak pokračuje v předchozí činnosti.

Rozdělení:

- HW přerušení (vnější) – obsluha HW zařízení
- SW přerušení – synchronní, instrukcí INT x v kódu procesu
- Vnitřní přerušení (výjimky) – procesor oznamuje chyby při vykonávání instrukcí

HW přerušení

Příklady:

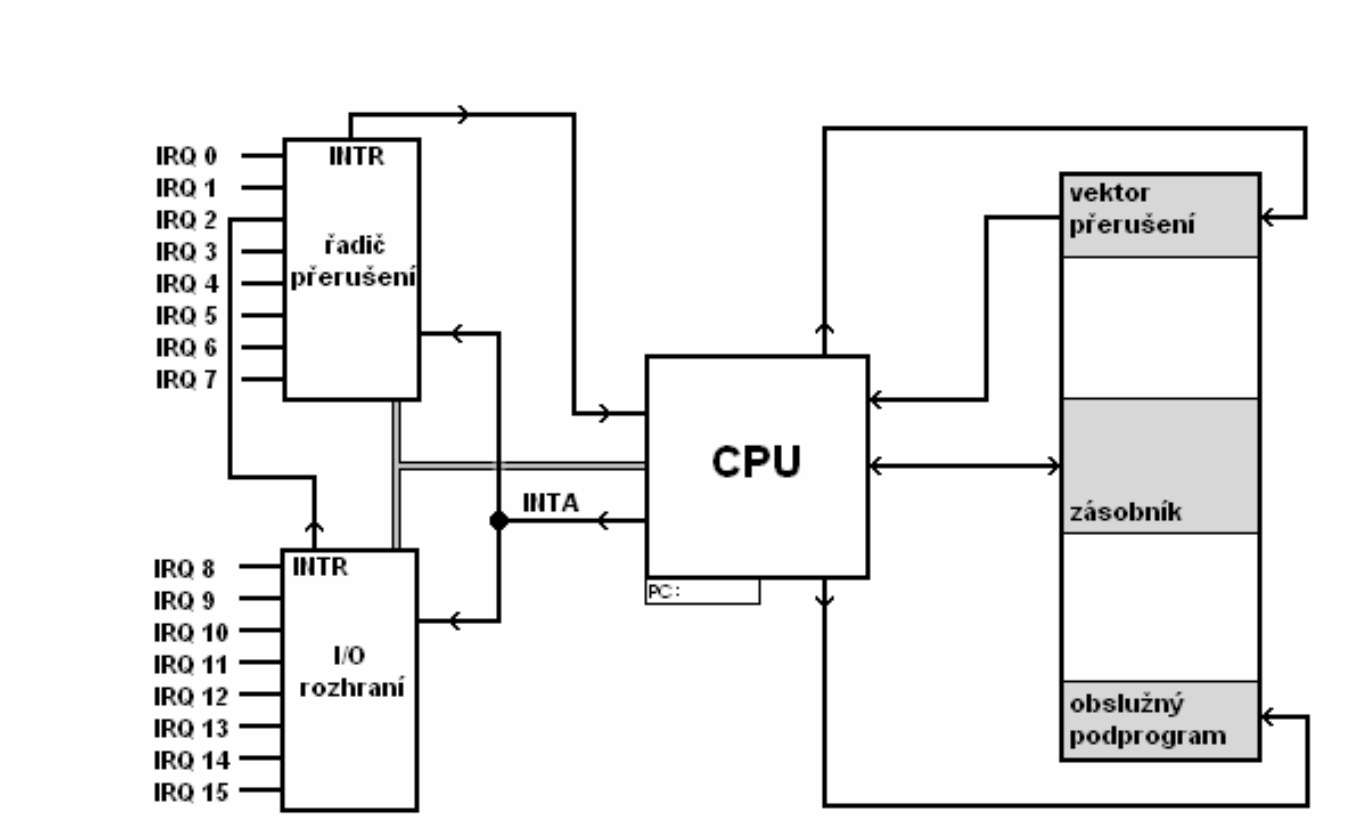
- Časovač (timer) – např. kontrola, zda neuběhlo časové kvantum
- Stisknutí klávesy na klávesnici
- Pohyb myši
- Disk signalizuje, že má k dispozici požadovaná data

HW zařízení žádá operační systém o pozornost („věnuj se mi“)

HW přerušení - zpracování

1. Vnější zařízení vyvolá požadavek o přerušení.
2. I/O rozhraní vyšle signál IRQ na řadič přerušení.
3. Řadič přerušení vygeneruje signál INTR – „někdo“ žádá o přerušení a vyšle ho k procesoru.
4. Procesor se na základě maskování rozhodne obsloužit přerušení a signálem INTA se zeptá, jaké zařízení žádá o přerušení.
5. Řadič přerušení identifikuje zařízení, které žádá o přerušení a odešle číslo typu přerušení k procesoru.
6. Procesor uloží stavové informace o právě zpracovávaném programu do zásobníku.
7. Podle čísla typu příchozího přerušení nalezne ve vektoru přerušení adresu příslušného obslužného podprogramu.
8. Vyhledá obslužný podprogram obsluhy přerušení v paměti a vykoná ho.
9. Po provedení obslužného programu opět obnoví uložené stavové informace ze zásobníku a přerušený program pokračuje dál.

HW přerušení - zpracování



Zdroj: wikipedia

SW přerušení

- Instrukcí INT x, kde x je číslo 0 – 255
- Vyvolání služby OS v Linuxu: INT 0x80

- SW přerušení se používá jako mechanismus pro systémové volání
- Při vyvolání přerušení se procesor přepne do privilegovaného režimu, kdy je možné vykonávat všechny instrukce (např. IN, OUT)
- Je synchronní – tj. nastane, když začneme zpracovávat instrukci INT daného procesu

SW přerušení - zpracování

1. Do zásobníku se uloží stavové informace o právě zpracovávaném procesu.
2. Zakáže se další přerušení.
3. Procesor zjistí vektor přerušení (dle operandu za instrukcí INT).
4. Nalezne obslužný podprogram (dle tabulky vektorů přerušení) a vykoná ho.
5. Po návratu z podprogramu obnoví uložené stavové informace o přerušeném programu.

Pozn.: často se na zásobník uloží jen návratová adresa, uložení dalších registrů na zásobník je už potom úkolem obslužné rutiny.

Vnitřní přerušení (výjimky)

- Dělení nulou
- Neplatná instrukce
- Nedostupnost koprocesoru (matematický koprocesor)
- Výpadek stránky (stránka je ve swapu místo v RAM)
- Nepřístupný segment

Tabulka vektorů přerušení

Motivace:

Procesor ví číslo přerušení, ale to mu nestačí. Potřebuje znát, kde leží **adresa obslužného podprogramu**, který má při daném přerušení vykonat.

Tedy mapování:

f(i)= adresa_obsluzneho_podprogramu

kde i je číslo přerušení (0 až 255)

Toto mapování zprostředkuje tabulka vektorů přerušení.

Tabulka vektorů přerušení

Příklad:

Procesor narazí v procesu na instrukci INT 0x80.

Na indexu 0x80 najde adresu obslužného podprogramu, na kterou skočí pro obsluhu daného přerušení.

V tomto konkrétním případě je to u Linuxu vstupní bod do jádra monolitického systému, kde se následně dle hodnoty v registru EAX zavolá příslušné systémové volání.

Tabulka vektorů přerušení

- Datová struktura
- 256 položek (0-255), po 4B, tedy velikost 1KB
- Od adresy O do adresy 1023
- Toto planí v reálném módu CPU

- V protected módu CPU
- IDT (Interrupt Descriptor Table)
- Pole 8bytových deskriptorů (místo 4B v předchozím případu)
- Naplněná IDT tabulka 2KB (256 x 8B)

Tabulka vektorů přerušení

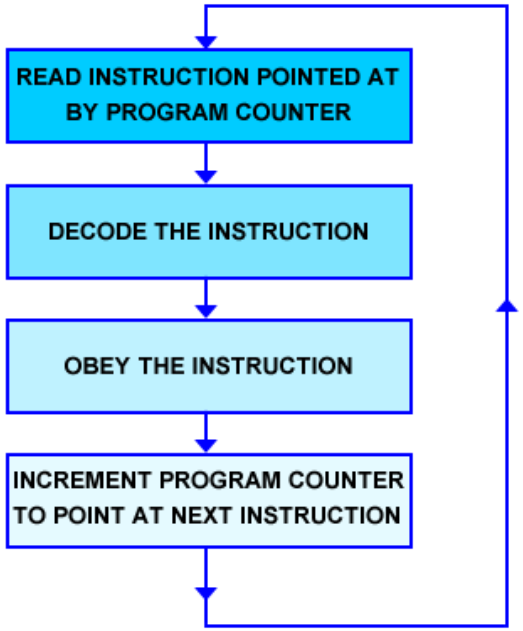
Definice:

Tabulka vektorů přerušení je datová struktura, ve které se uschovávají vektory přerušení.

Vektor přerušení – adresa (první instrukce) podprogramu pro obsluhu daného přerušení.

Čítač instrukcí

- Říká se mu obecně program counter (PC)
- Reálně jde o dvojici registrů CS:EIP
- Ukazuje do paměti na instrukci, která je na řadě pro vykonávání na CPU



Location	Instruction
0	LOAD R0 1
1	ADD R0 1
2	WRITE 0 R0
3	READ R1 0
4	LOAD R2 4
5	COMPARE R0 R2
6	JUMPLT 2
7	EXIT
8	LOAD R1 4

Program Counter	3