KIV Operační systémy

V/V zařízení

Verze 1.0

Konfigurace směrovače konzolí

- Např. viz cvičení KIV/PSI, směrovač lze konfigurovat po připojení konzolového portu směrovače do sériového portu počítače
  - BIOS sice poskytuje funkce pro práci se sériovými porty, ale jsou to rutiny pro real-mode, ve kterém není izolace procesů
  - Komunikace se sériovým portem se tak odehrává pomocí portů, na x86 instrukcemi in a out, což jsou privilegované operace
  - S periferií tak může ve skutečnosti komunikovat pouze jádro OS a procesům periferii jenom virtualizuje

Tiskárna

- Pokud bychom např. místo směrovače uvažovali lokální tiskárnu na (dnes již de-facto u PC nepoužívaném LPT), pak by současný přístup několika procesů k fyzicky jedné tiskárně vedl k promíchání příkazů a nesmyslnému výstupu z tiskárny
- Jádro OS tedy vedle virtualizace hw zajišťuje i celistvost prováděných (tiskových) úkolů
  - U tiskárny serializuje tiskové úlohy
  - U sériového portu povolí komunikaci jenom jednomu procesu

Disk

- Co když bude chtít několik procesů číst z disku?
- Stejné jako s tiskárnou a sériovým portem
- OS zde ale má možnost optimalizace/prioritizace
  - Např. budeme mít magnetický disk s plotnami a záznamovou hlavou
  - OS pak může přeuspořádat požadavky na disk tak, aby se s hlavou hýbalo co nejméně => tišší a rychlejší chod
    - Dělal např. Novell Netware
    - Dnes na to mají disky Native Command Queuing

VGA – provedení akce

- VGA je hw, který lze ovládat jak pomocí přerušení, tak pomocí zápisů na porty
- Např. změna grafického režimu

```
mov ax, 13h          ;320x200px a 256 barev v paletě
int 10h
```

- Např. změna barvy v paletě

```
outportb(0x3c8, 1);          //číslo barvy v paletě
outportb(0x3c9, 255);        //červená složka barvy
outportb(0x3c9, 0);          //zelená složka barvy
outportb(0x3c9, 0);          //modrá složka barvy
```

VGA – počkání na dokončení

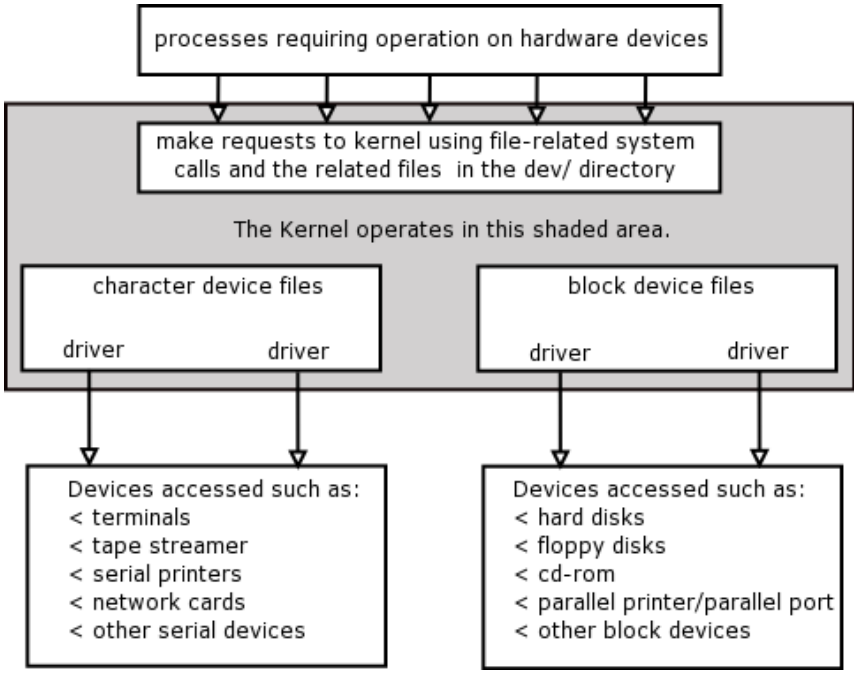
- V době elektronových obrazovek, bylo dobré měnit paletu v době, kdy se svazek elektronů přesouval z konce obrazu na jeho začátek
  - I z dnešního pohledu jde stále o to, že se synchronizuje FPS s obnovovací frekvencí monitoru
- VGA to ovšem neoznámí přes IRQ, takže je nutné dělat polling stavové informace

```
do { //čeká, dokud se svazek nevrátí zpět
} while (inportb(0x3DA) & 8); //nekreslí se obrazovka


do { //čeká, dokud se svazek nezačne opět vracet
 } while (!(inportb(0x3DA) & 8)); //kreslí se obrazovka
```

Bloková a znaková zařízení

- Bloková zařízení – ovladač zařízení komunikuje s hw pomocí bloků dat velkých několik bytů
  - Např. u disků se čtou a zapisují celé sektory
  - Pro urychlení práce se zařízením, je-li to možné, se cachují právě tyto bloky dat
- Znaková zařízení
  - Komunikuje se sekvenčně po jednom bytu, nemá cache



- Každý hw má své id; id může mít strukturu
  - Např. typ zařízení a pořadí ve skupině daného typu

http://www.learnlinux.org.za/courses/build/internals/ch06s05.html

I/O API

- Programátor nepracuje přímo s konkrétním hw, ale volá funkce operačního systému
  - Např. může volat funkce pro čtení a zápis do souboru, přičemž je konkrétní, pro uživatelský proces virtualizovaný hardware identifikovaná pomocí souborového deskriptoru
  - Tj. dochází k využití infrastruktury, kterou jsme si již ukázali u souborových systémů
  - Dále si tedy ukážeme, co se děje na úrovni kernelu, který pracuje se skutečným, nevirtualizovaným hw

I/O subsystém

1. Po zavolání příslušného API, OS vytvoří I/O požadavek (request), který předá dál (block) I/O subsystému.
2. Volající thread se pozastaví, než bude I/O požadavek dokončen.
3. I/O požadavek se zařadí do I/O fronty. Až přijde na řadu, ovladač provede požadovanou akci s hw.
4. Po dokončené se příslušný thread převede do stavu runnable.
    - U síťové karty je operace dokončena ihned po předání příkazů hw. Ale např. u disku může hw provádět akci dále a její dokončení teprve oznámí – např. pomocí IRQ.

Ne2000 (NIC)

- Jednoduchý, referenční design chipsetu síťové karty od Novellu, který se stal populárním

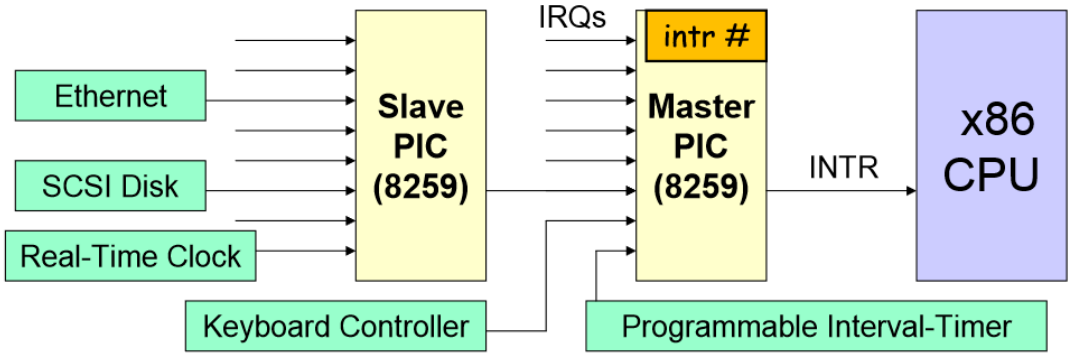
- Odesílání dat
  - Kombinace odesílání příkazů na konkrétní porty a čekání, až budou konkrétní stavové bity shozeny či nastaveny
- Přijímání dat
  - Nejprve hw vygeneruje IRQ, čímž oznámí přijetí dat
  - A teprve poté se vykoná obsluha, která data přečte

Obsluha IRQ

- Jakmile dojde u konkrétního hw k nějaké události, kterou je třeba obsloužit pomocí sw, tento hw vygeneruje tzv. interrupt request – IRQ
  - Stisk klávesy, data přijatá síťovou kartou, hodiny, dělení nulou, atd.
- IRQ je tedy hw signál, který je zaslán CPU
- CPU následně zastaví aktuálně vykonávaný program a vykoná obsluhu přerušení, která přísluší danému IRQ
  - K čemuž použije tabulku vektorů přerušení, protože na tyto čísla jsou namapované čísla jednotlivých IRQ

PIC

- Programmable Interrupt Controller
- Dokáže obsloužit 8 IRQ, používaly se proto 2
  - Master PIC
    - IRQ0 (hodiny) až IRQ 7 (LPT1); IRQ2 je kaskádově Slave PIC
  - Slave PIC
    - IRQ8 (RTC) až IRQ15 (sekundární ATA kanál)



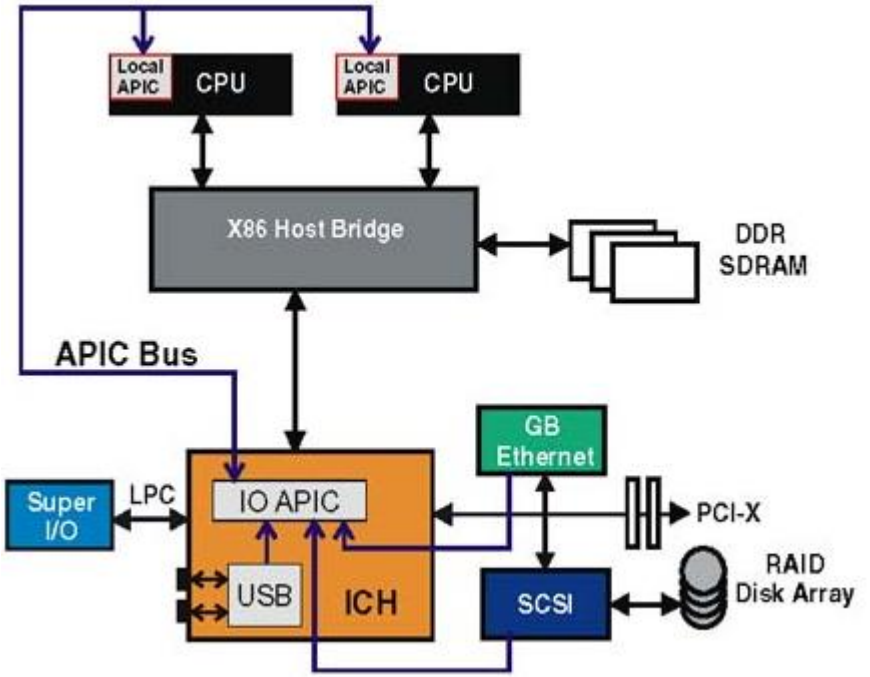
http://www.cs.columbia.edu/~junfeng/10sp-w4118/lectures/l04-syscall-intr.pdf

IRQ konflikt/sdílení

- Co se stane, když budou chtít dvě periferie sdílet to samé IRQ?
  - Např. zvuková karta a tiskárna na LPT1 budou sdílet IRQ5
  - Pokud se nebudou zařízení používat najednou, může to projít, pokud bude obsluha přerušení správně reagovat (známé jako IRQ sdílení)
  - Anebo také může dojít k tomu, že se celý počítač „zamrzne“
- Systémově se situace vyřešila pomocí
  - Message-Signaled Interrupts – zařízení už nepředpokládá, že má dedikovanou linku do PIC, ale že je tato linka možná sdílená, a proto po ní vysílá své ID
  - Advanced PIC

Advanced PIC

- Umožňuje směrovat mnohem více IRQ než PIC
- Umožňuje konstrukci SMP
  - Každý CPU má svůj Local APIC
    - 224 IRQ
    - prvních 0-31 IRQ je vyhrazeno
      - Vyžaduje Message-Signaled Interrupts
      - Někdy se jim říká Virtual IRQs
  - V systému je ještě I/O APIC
- (Naprogramováním) Lze směrovat jednotlivé IRQ na jednotlivé CPU



http://flylib.com/books/en/3.356.1.153/1/

Top Half

- Přerušení musí být obslouženo co nejrychleji, jinak nám může „počítač zmrznout“ stejně jako u IRQ konfliktu
- Když po IRQ CPU spustí příslušnou obsluhu přerušení, tato obsluha vykoná jenom to nejnutnější a zbytek činnosti se odloží na pozdější dobu
  - Tj. nevykonaná práce přidá se někam do fronty
  - Ale hw se řekne, že jeho IRQ bylo úspěšně obslouženo!
- Obsluha přerušení vykonává jenom tzv. „Top Half“

Bottom Half

- V systému běží několik „démonů“, kteří periodicky vykonávají odložené činnosti, které vznikly v důsledku Top Half obsluhy přerušení
  - Odložená činnost (deferred work) se jmenuje Bottom Half
  - Např. UDP datagram nedostanete jak přijde, ale až se vykoná příslušná Bottom Half!
- Ne každá Bottom Half má stejnou důležitost => tj. prioritu
- Obsluhu Bottom Half můžeme rozdělit do dvou skupin:
  - Kritická – v Linuxu se jí říká SoftIRQ
  - Plánovatelná – v Linuxu se jí říká Tasklet a vykonává ji SoftIRQ

Linux: SoftIRQ vs Tasklet

- SoftIRQ
  - Staticky alokovaný a přiřazený konkrétnímu CPU
  - Obsluhuje časově kritické události, časovače (a v Linuxu síť a SCSI)
  - SoftIRQ stejného typu může zároveň běžet na několika CPU v SMP
  - Jeden ze SoftIRQ vykonává tasklety
- Tasklet
  - Lze je deklarovat staticky i dynamicky
  - Tasklety mohou běžet zároveň, ale musí být různého typu
    - Tj. může být spuštěn maximálně jeden Tasklet konkrétního typu
  - V Linuxu se dále dělí na dvě priority HI_SOFTIRQ a TASKLET_SOFTIRQ
  - Musí dokončit svou činnost atomicky, nelze ho uspat
  - Poběží na tom samém CPU, které ho naplánovalo

Linux: Workqueue

- (deferred work) SoftIRQ a Tasklet „co nejdříve“ vykonává jádro při různých příležitostech
  - Např. při dokončení poslední obsluhy přerušení (ISR), nicméně konkrétní „kdy“ závisí na konkrétní architektuře a jádru
- (delayed work) Workqueue je také odložená práce, ale už nemusí být vykonána „co nejdříve“
  - Je to mechanismus, kterým si jádro řekne, že daná práce se udělá někdy v budoucnu
- Každá Workqueue má v jádře svůj vlastní thread
  - Respektive, může mít jeden thread na každý jeden CPU v SMP
- Protože každý ovladač nepotřebuje svou vlastní Workqueue, v systému existuje tzv. Shared WorkQueue

IRQ Polling

- Např. IRQ hodin, dělení nulou nebo výpadek stránky budou vždy obslouženy pomocí přerušení
- Ale jiná IRQ mohou být obsloužena pomocí tzv. pollingu
  - Např. viz příklad s diskem u I/O subsystému, driver nemusí čekat až disk oznámí dokončení akce pomocí IRQ, ale může periodicky číst (tzv. poll) jeho stavové bity
  - Nelze sice automaticky povědět, že IRQ je špatný a okamžitě zavržení hodný
  - Ale, máme-li obsluhovat obrovské množství I/O požadavků, polling řeší několik aspektů, nad kterými je třeba se zamyslet

Interrupt vs. Polling

- Při IRQ pollingu ovladač čeká ve smyčce, dokud není práce vyřízena – respektive, dokud nemá prázdnou frontu I/O požadavků
- Když se generuje velké množství IRQ CPU musí držet krok co do počtu změn kontextu tam a zpět
  - Při stovkách tisíc IRQ za sekundu ho ale hw nakonec „uštve“
  - U tzv. short-wait by se zase I/O thread uspal, plánovač pak spustí jiné vlákno a to začne přepisovat cache CPU, což zpomalí I/O thread, až bude čekat na opětovné nahrání svých dat do cache
  - Interrupt coalescing – hw generuje IRQ až od nějakého počtu událostí nebo dojde k timeoutu, po který se na ně čeká

DMA - motivace

- Direct Memory Access
- Čtení dat ze zařízení pomocí portů a přerušení může fungovat rozumně pouze tehdy, pokud se jedná o malý počet dat
  - Např. modem s rychlostí 9600 baudů vyvolá přerušení cca každé 2ms při přenosu cca 1 znaku každou milisekundu
  - I disketová mechanika přenáší příliš velký objem dat
  - A co teprve moderní disk nebo síťová karta?
  - => potřebujeme mechanismus, který přenese data z paměti systému do paměti zařízení – DMA

DMA - použití

- V systému je omezený počet DMA kanálů a nelze je sdílet
  - Ovladač musí vždy počkat, až je některý kanál volný
  - A ne každé zařízení může použít libovolný DMA kanál
  - => jádro si udržuje seznam DMA kanálů a mj. ke každému zná jeho číslo a stav, zda je volný
- CPU zahájí přenos dat a dále se pak může věnovat jiné činnosti – zařízení oznámí konec přenosu přes IRQ
  - Legacy hardware (disketa, paralelní port, IrDA,…) používají 16-bitovou adresu – DMA je pak omezen na prvních 16MB fyzické paměti
  - DMA s PCI sběrnicí ve výchozím režimu používá 32-bitovou adresu, ale v double-address cycle mapping umí 64-bitovou adresu