

Správa zariadení

= vytvoriť nejaký menej komplikovaný interface nad zariadením

Hardware správy zariadení

- zariadenie v princípe pozostáva z 2 častí: radič (controller) + samotné zariadenie
- radič zabezpečuje preklad z protokolu používanom na zariadení na protokol, aký používa zbernica
- programátor teda programuje len radič
- radič môže byť úplne jednoduchý, ale občas to býva aj celý procesor atď
- radič obsahuje nejaké registre – radič sa ovláda tak, že sa mu do registra uložia nejaké dáta, ktoré radič pochopí ako príkaz

Porty a mapovanie pamäti

a./ porty majú vlastný adresný priestor

b./ pri mapovaní pamäte je jeden adresný priestor

c./ kombinácia a+b - „malé dáta“ (príkazy) sa prenášajú priamo na porty, väčšie dáta sa mapujú do pamäti

- z hľadiska programátora to vyzerá easy, ale v realite viaczbernicových počítačov to musí byť zabezpečené priamo na úrovni hardware

Direct Memory Access

- pozri v slajde
- význam: na jednu operáciu sa dá preniesť veľa dát priamo z radiča do pamäte (bez účasti procesora)

Prerušenia

- zariadenia nežiadajú o prerušenie procesor, ale interrupt controller
- procesor má len jednu linku na prerušenie – rozlišovanie, ktoré zariadenie požiadalo o prerušenie sa deje na interrupt controlleri
- ako to funguje: zariadenie požiadá o prerušenie -> interrupt controller vyšle prerušenie do procesora -> procesor začne vykonávať prerušenie, spýta sa interrupt controllera aké vlastne prerušenie nastalo

Software správy zariadení

- úlohy:
 - umožniť komunikáciu so zariadením v „jednoduchšom jazyku“, zjednotiť ovládanie podobných zariadení (Floopy, CD, SCSI Disk, ATA Disk, ...)
 - pomenovať zariadenia
 - spracovávať chyby tak, aby som sa dozvedel len o tých najväznejších
 - prezentovať asynchrónne operácie ako synchronne (vo Win sa však dajú robiť aj asynchrónne operácie)

- buffering – nie každá 1 operácia od užívateľa hneď vyvolá operáciu na zariadení a vice-versa
- pridelovanie zariadení procesom
 - zariadenia sa delia na blokovito orientované zariadenia (napr. disk – tieto zariadenia sú zdieľateľné) a znakovito orientované zariadenie (napr. tlačiareň – je tam „prúd“ bajtov a ona ich interpretuje v takom poradí, v ako prichádzajú – nezdieľateľné)
- software správy zariadení sa dá rozdeliť na 4 vrstvy:
(pozri v slajde)

Prerušovacie podprogramy

- krátke úseky kódu, ktoré reagujú na prerušenia z jednotlivých zariadení

Deadlocky

- deadlock: situácia, keď jeden program z nejakej množiny čaká na udalosť, ktorú môže spraviť len iný program z tej istej množiny (=> čakajú navzájom na seba)
- na úrovni chyby programátora sa ním OS nezaobera
- OS sa nimi ale musí zaoberať na úrovni „nezdieľateľných“ zariadení => môže sa stať, že dva procesy, ktoré o sebe vôbec nevedia sa dostanú do deadlocku. Takúto situáciu musí vyriešiť operačný systém.
- podmienky nutné aby vznikol deadlock, sú:
 1. nezdieľateľné prostriedky – každý prostriedok môže byť v 1 čase pridelený len 1 procesu
 2. môžem žiadať aj keď už vlastním – proces môže zažiadať o 1 prostriedok a keď ho dostane, môže žiadať ďalší
 3. neexistuje preempcia – procesu nie je možné odobrať prostriedky skorej ako ich uvoľní
 4. cyklické čakanie – proces A čakajúci na prostriedok čaká na proces B (ktorý prostriedok vlastní), ten čaká na proces C a proces C čaká na proces A...
- vznik deadlocku môže závisieť od poradia, ako sa procesy naplňujú (pozri slajdy „Graf procesov a prostriedkov (2)“ a „Graf procesov a prostriedkov(3)“)

Stratégie prístupov k deadlockom

Algoritmus pštrosa

- strčiť hlavu do piesku a tváriť sa ako lietadlo
- môže to byť vhodný prístup
 - ak deadlocky vznikajú len veľmi zriedka, alebo ak jeho riešenie by bolo moc zložité, moc drahé, ...
 - ak deadlock považuje výhradne za chybu programátora spolupracujúcich prostriedkov
- napr. v Unixe existuje teoretický deadlock – max. počet bežiacich threadov... Ak je tabuľka procesov plná, proces, ktorý chce vytvoriť nový thread čaká...; vo Windowsoch nastane výnimka

Detekcia a zotavenie

- pridel'ujem tak ako procesy žiadajú a začnem riešiť, až keď vznikne deadlock
- **detekcia:**
 - nájsť v grafe procesov a zariadení orientovaný cyklus (=cyklické čakanie). Toto je ale jednoduché len vtedy, ak mám len 1 kus z každého zariadenia.
 - všeobecný postup (ak mám viac ks z 1 zariadenia):
 - vektor E (e_1, e_2, \dots, e_n) – hovorí, koľko existuje kusov z každého zariadenia
 - vektor A (a_1, a_2, \dots, a_n) – hovorí, koľko kusov z každého zariadenia je k dispozícii (nepridelených)
 - matica C - current (počet procesov \times n) – ktorý proces má koľko kusov z každého zariadenia pridelených
 - matica R – request – ktorý proces žiada o koľko kusov z každého zariadenie
 - potom porovnávam vektor A s riadkami matice R : ak platí, že každé $a_i \leq r_{mi}$ tak to znamená, že tento proces m časom skončí \Rightarrow riadok C_m pripočítam k vektoru $A \Rightarrow$ takto skúšam a ak dokážem „vynulovať“ celú maticu tak nie som v deadlocku
- **zotavenie:**
 - ok, zistil som že mám deadlock. Čo teraz?
 1. zotavenie odobratím prostriedku
 - niektorému procesu zoberiem prostriedok – je dosť pravdepodobné že on dostane chybovú hlášku a bude na to vedieť zareagovať (napr. modem – software je pripravený na to, že sa mu rozpadne spojenie – čiže ak mu modem zoberieme, sw na to bude pripravený)
 2. rollback
 - návrat do posledného checkpointu (keď ešte deadlock nebol) („undo pred deadlock“)
 3. ukončenie procesu
 - ukončím 1. proces v deadlocku, ak to nepomôže ukončím ďalší, ...
 - „si v deadlocku tak skap!“
 - proces vlastne padne bez vlastnej viny

Vyhýbanie sa deadlockom opatrným pridel'ovaním prostriedkov

= algoritmus bankára

- nepridelím, aj keď by som mohol (ak viem dopredu, že by mohol vzniknúť deadlock)
- chcem sa vyhnúť deadlocku
- aby sme to vedeli spraviť, musíme vedieť, aké prostriedky budú procesy potrebovať \Rightarrow proces musí na začiatku oznámiť, čo bude potrebovať
- treba rozlišovať bezpečné a nebezpečné stavy
- pri interaktívnych OS sa veľmi nepoužíva – nie je možné vedieť dopredu kto bude čo potrebovať

Prevenia negovaním niektorej zo 4 podmienok

- pomením pravidiel v OS tak, aby nenastala niektorá z podmienok vzniku deadlocku
- podľa toho, ktorú podmienku budeme negovať:
 1. **prostriedky sú zdieľateľné - spooling**
 - virtualizácia zariadenia
 - každý, kto zariadenie potrebuje dostane jeho virtuálnu kópiu – a tie potom obsluži niekto iný

- spravidla pri neinteraktívnych výstupných zariadeniach (tlačiareň, ...); ale napr. v sálových počítačoch to mali aj pri neinterakt. vstupných zariadeniach (dierne štítiky)
- 2. môžem žiadať len vtedy, ak nič nevlastním**
 - to by ale znamenalo, že každý proces môže mať len 1 zariadenie. zlepšenie: žiadať o viac zariadení naraz (hladný filozof, pokiaľ nemá k dispozícii obidve vidličky, nebude 1 blokovať...)
 - vo všeobecnosti sa nepoužíva
- 3. preempcia**
 - odobrať zariadenie
- 4. nevznikne cyklus**
 - zavediem poradie, v akom sa zariadenia pridelujú (očísľujem zariadenia – napr. 1.scanner, 2.plotter, 3.CD ROM)
 - proces môže žiadať aj ak už nejaké zariadenie má, ale môže žiadať len o zariadenie s vyšším číslom (ak máš plotter, môžeš žiadať o CDROM, ale nie o scanner)
 - vo všeobecnosti sa nepoužíva (bolo by dosť problematické vymyslieť nejaký univerzálny spôsob číslovania)

Starvation

- to starve (angl.) - umrieť od hladu
- je to situácia, keď nejaký proces sa nedostáva k slovu, ale nie je to deadlock (napr. pri shortest job first sa bude stále niečo predbiehať pred môj (dlhý) proces a ten sa nedostane k slovu)



This work is licensed under a [Creative Commons Attribution-NonCommercial-Share Alike 3.0 License](http://creativecommons.org/licenses/by-nc-sa/3.0/).
Original source: <http://matfyz.adammuller.sk/opsys>