1 Rozdělení OS, architektura a komponenty OS. Základní funkce OS

Rozdělení

Podle úrovně sdílení CPU

- Jednoprocesový - např. MS DOS; v daném čase je v paměti aktivní jeden program
- Multiprocesový - např. UNIX, WinNT; aby se využilo systémových zdrojů (sdílení CPU, paměti) je v hlavní paměti více než jeden program. V současnosti nejčastěji v systémech pro více uživatelů.

Podle typu interakce/požadavků na odezvu

- Dávkový systém - uživatelské úlohy se zadávají jako sekvenční dávky, během zpracování není interakce mezi uživatelem a jeho úlohou.
- Interaktivní - dovolí uživatelům interakce s jejich úlohami
- OS reálného času - konkrétní aplikace mají přísné požadavky na čas odpovědi. OS se těmto požadavkům podřizují. Vhodné pro multimédia, virtuální realitu. 2 typy:
  - Hard real time: zaručuje odezvu systému v ohraničeném čase
  - Soft real time: RT úlohy mají prioritu před vším ostatním, avšak nezaručuje odezvu v ohraničeném čase.

Dle velikosti HW

- superpočítač
- telefon
- čipová karta

Míra distribuovanosti

- Klasické - centralizované 1 and more CPU
- Paralelní
- SÍŤOVÉ
- Distribuované virtuální uniprocesor - Uživatel neví kde běží programy, kde jsou soubory

Další možnosti dělení OS:

- podle počtu uživatelů
  - jedno / více uživatelské
- podle funkcí
  - univerzální
  - specializované

Architektura

OS = jádro + systémové nástroje

Dělíme na

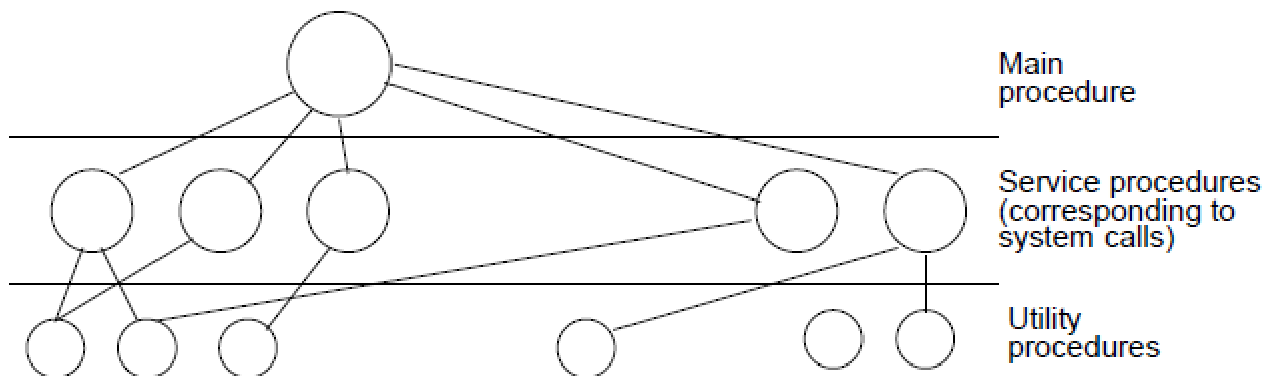
- Monolitické jádro – jádro je jeden funkční celek (GNU/Linux)
- Mikrojádro – malé jádro, oddělitelné části pracují jako samostatné procesy v user space (GNU/Hurd)
- Hybridní jádro - kombinace (Win XP, Win Vista...)

Monolitické jádro

- Jeden spustitelný soubor
- Uvnitř moduly pro jednotlivé funkce (filesys, procesy)
- Jeden program, řízení se předává voláním podprogramů
- Příklady: UNIX, Linux, MS DOS

Typickou součástí jádra je např. souborový systém

Linux je monolitické jádro OS, s podporou zavádění modulů za běhu systému



Mikrojádro

- Model klient – server
- Většinu činností OS vykonávají samostatné procesy mimo jádro (servery, např. systém souborů)
- Poskytuje pouze nejdůležitější nízkoúrovňové funkce
  - Nízkoúrovňová správa procesů
  - Adresový prostor, komunikace mezi adresovými prostory
  - Někdy obsluha přerušení, vstupy/výstupy
- Pouze mikrojádro běží v privilegovaném režimu
  - Méně pádů systému

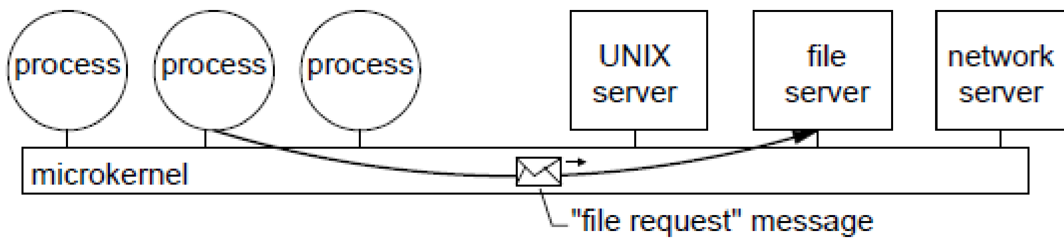
Výhody

- vynucuje modulární strukturu
- Snadnější tvorba distribuovaných OS (komunikace přes síť)

Nevýhody

- Složitejší návrh systému
- Režie

Příklady: QNX, Hurd, OSF/1, MINIX, Amoeba



Komponenty OS

- správa procesů
- správa hlavní paměti
- soubory
- správa zařízení - I/O subsystém
- síť (networking) .. viz KIV/UPS
- ochrana a bezpečnost
- uživatelské rozhraní

- Procesy; proces = běžící program; potřebuje minimálně:
  - Čas CPU = být vykonán
  - Paměť = mít kde běžet
  - Vstupy a výstupy = soubory a I/O zařízení
- Správa hlavní paměti
  - Alokace a dealokace paměti podle potřeby
  - Udržuje informace, která část paměti je používána a kým
- Soubory
  - Vytváření/ rušení souborů/adresářů
  - Mapování souborů na vnější paměť
  - Rozvrhování diskových operací
- I/O podsystém
  - Správa paměti pro buffering, caching, spooling
  - Společné rozhraní ovladačů zařízení
  - Ovladače pro specifická zařízení
- Síť
- Ochrana a bezpečnost = ke zdrojům smí přistupovat pouze autorizované procesy
- Uživatelské rozhraní - CLI, GUI

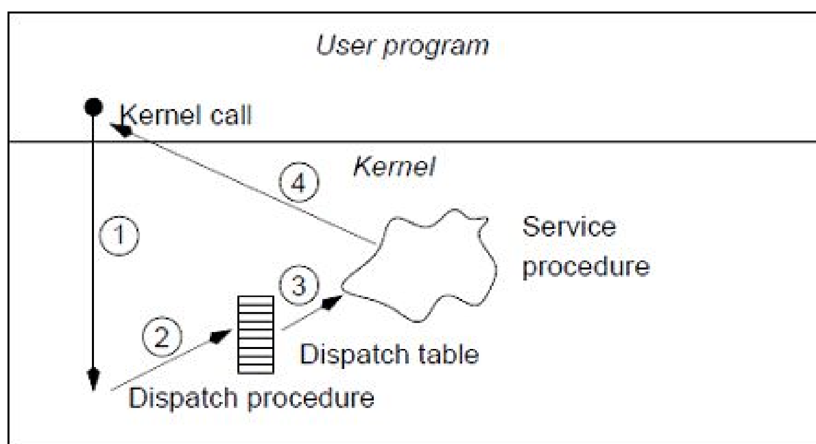
2
Vyvolání služeb OS, zpracování přerušení

Většina moderních CPU: 2 režimy:

- Privilegovaný - režim jádra, všechny instrukce jsou povoleny, v tomto běží pouze OS
- Uživatelský - některé instrukce (I/O) jsou zakázány, v tomto běží ostatní programy

Aby uživ. aplikace mohla provádět zakázané instrukce, musí požádat OS o jejich provedení následujícím způsobem (Vyvolání služby):

- Aplikace si uloži parametry pro službu na předem určené místo (registry, zásobník,...)
- Zavolá metodu volání služby jádra (např. v C je to syscall( ))
- Ta podle parametrů zjistí jaká složba je požadována, která přepne CPU do režimu jádra, a vyvolá obsluhu
- Ta službu provede a při návratu do uživatelského programu přepne CPU zpět do uživatelského režimu



Programovací jazyky skrývají volání funkcí jádra tak, aby vypadaly jako běžné knihovní funkce.

Zpracování přerušení

Přerušení je událost, která pokud nastane, je třeba ji v krátkém čase obsloužit. Je tedy přerušeno vykonávání instrukcí, provedena obsluha události a po té se pokračuje v předchozí činnosti. Provedení tohoto je v režimu jádra.

Druhy:

- HW přerušení (vnější) - generuje jej I/O zařízení, asynchronně, například přijetí paketu, stisknutí klávesy,... Stará se o ně řadič přerušení (priority atd)
- Vnitřní přerušení - generuje proces, například dělení nulou, výpadek stránky, ...
- Softwarové - synchronní, generováno aplikací jako takovou, záměrně, volání služby OS je příkladem

Přerušení na Wiki (http://cs.wikipedia.org/wiki/P%C5%99eru%C5%A1en%C3%AD)

Vektor přerušení - ukazatel, kde pro daný typ přerušení (a jeho číslo) začíná program pro jeho obsluhu.

Obsluha:

- Přijde signalizace přerušení
- Je dokončena právě prováděná instrukce v CPU
- Adresa další instrukce je uložena na zásobník
- Do PC (Program Counter - ukazatel CPU na další instrukci) je nastavena instrukce, na kterou ukazuje odpovídající vektor

přerušení)

- Přepnutí kontextu (Obsluha je samostatný proces)
- Obsluha uloží stav registrů a obecně stav CPU do zásobníku
- Provede obsluhu přerušení
- Provede (I)RET
- Plánovač opět přepne kontext podle dat v zásobníku a původní proces pokračuje v činnosti (kromě zpoždění tedy neví, že přerušení proběhlo, musí tedy naslouchat všem výstupům obsluh přerušení, které jej zajímají)

3 Proces, implementace procesu, konstrukce pro vytváření procesů

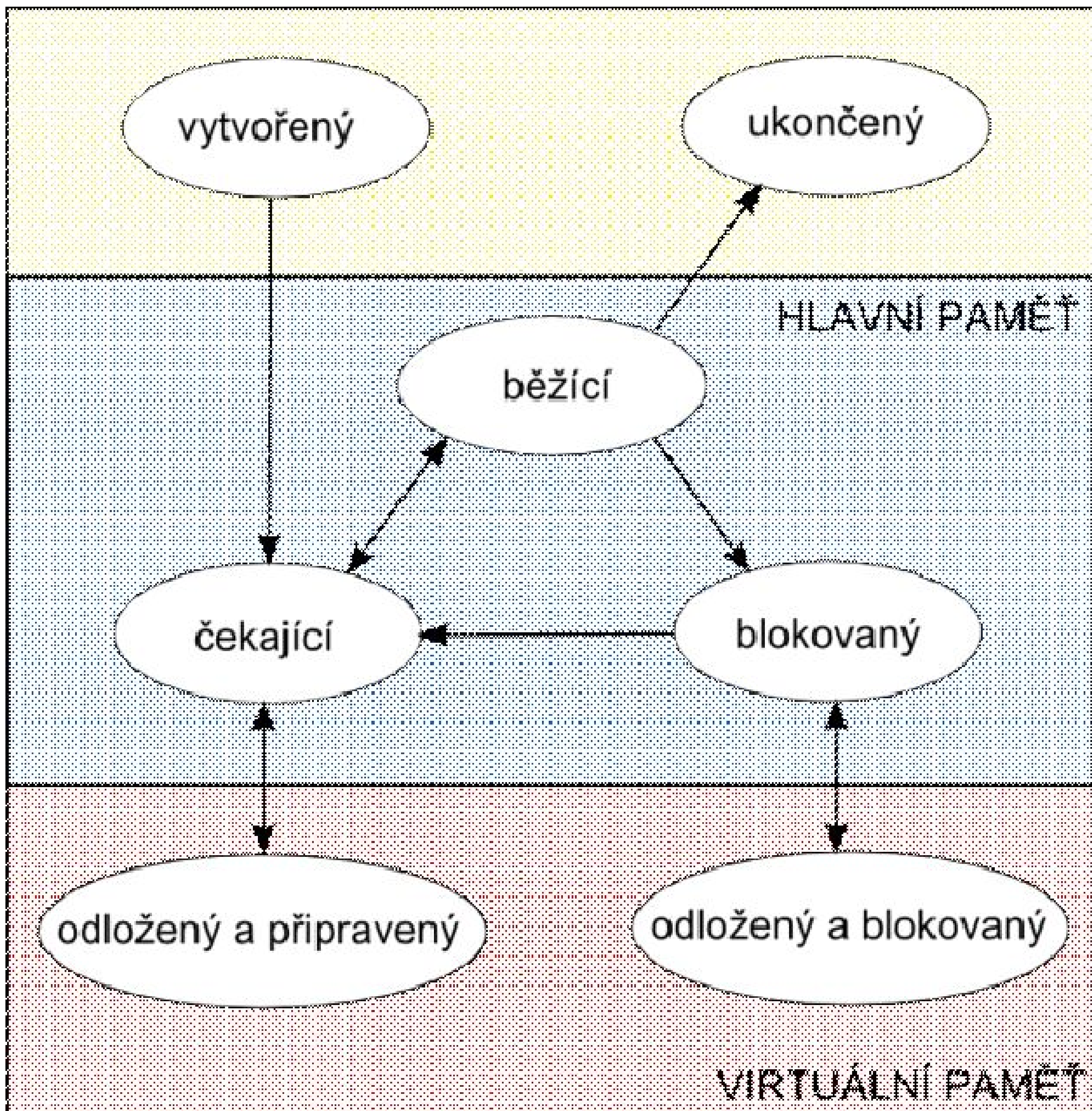
Proces = program, který běží

- Má vlastní adresový prostor, ve kterém běží
- V adresním prostoru má kód spustitelného programu, data, zásobník
- S procesem sdruženy registry a další informace potřebné k běhu programu
- Stavy: běžící, připraven, blokován

Základní stavy procesů

Následující stavy procesů se vyskytují ve všech víceúlohových systémech:

- vytvořený (created) - proces je vytvořen buď příkazem uživatele (u terminálu), nebo na žádost operačního systému o provedení služby, či na žádost jiného procesu (rodiče)
- připravený (ready) nebo čekající (waiting) - připravený pro vstup do stavu běžící, čeká pouze na přidělení procesoru
- běžící (running) - procesu je přidělen procesor a právě se provádí příslušné programy
- blokovaný (blocked) - proces je převeden do tohoto stavu v případě, kdy čeká na dokončení nějaké vstupně-výstupní operace, případně na skončení jiného procesu, uvolnění zdroje, synchronizační primitivum a podobně
- ukončený (terminated) - proces skončil



Implementace procesu

OS udržuje tabulku nazývanou TABULKA PROCESŮ - každý proces v ní má položkou nazývanou PCB (Process Control Block). PCB obsahuje všechny informace, které musejí být uchovány, je-li proces přepnut ze stavu „běžící“ do „připraven“ nebo „blokován“ - tak aby bylo proces možné znovu spustit. Konkrétní obsah se liší mezi systémy, ale většina obsahuje:

- Správa procesu
  - Identifikátory
  - Stavová informace procesoru
  - Stav procesu
  - Plánovací parametry procesu
  - Odkazy na rodiče a potomky
  - Nastavení meziprocesové komunikace
- Správa paměti

    - Popis paměti = ukazatel, velikost, přístupová práva
    - Úsek paměti s kódem programu
    - Data
    - Zásobník
- Správa souborů
    - Prostředí (aktuální adresář, …)
    - Otevřené soubory

Konstrukce

```
pid = fork();
if (pid == 0)
  jsem potomek;
else
  jsem rodič;
```

4
Paralelní procesy, prostředky pro popis paralelních procesů, vlákna

- Všechen SW běžící na počítači je organizován jako množina sekvenčních procesů
- Proces = běžící program včetně obsahu čítače instrukcí, registrů, proměnných, běží ve vlastní paměti
- Koncepčně má každý proces vlastní virtuální CPU

Při pseudoparalelním běhu je v jednu chvíli aktivní pouze jeden proces. Po nějakém čase se OS rozhodne pozastavit běh procesu a spustit běh dalšího. Po dostatečně dlouhém čase všechny procesy vykonají část své činnosti.

Obsah

- 1 Základní stavy procesů
- 2 PCB
- 3 Vlákna
- 4 Precedenční graf

Základní stavy procesů

Následující stavy procesů se vyskytují ve všech víceúlohových systémech:

- vytvořený (created) - proces je vytvořen buď příkazem uživatele (u terminálu), nebo na žádost operačního systému o provedení služby, či na žádost jiného procesu (rodiče)
- připravený (ready) nebo čekající (waiting) - připravený pro vstup do stavu běžící, čeká pouze na přidělení procesoru
- běžící (running) - procesu je přidělen procesor a právě se provádí příslušné programy
- blokovaný (blocked) - proces je převeden do tohoto stavu v případě, kdy čeká na dokončení nějaké vstupně-výstupní operace, případně na skončení jiného procesu, uvolnění zdroje, synchronizační primitivum a podobně
- ukončený (terminated) - proces skončil

vytvořený
ukončený
HLAVNÍ PAMĚŤ
běžící
čekající
blokovaný
odložený a připravený
odložený a blokovaný
VIRTUÁLNÍ PAMĚŤ

Lepší obrázek z přednášek:

Precedenční grafy

- Acyklický orientovaný graf
- Běh procesu Pi je vyjádřen orientovanou hranou grafu
- Vztahy mezi procesy znázorněny spojením hran

Abstraktní primitiva fork, join a quit

- Conway 1963
- Jeden z prvních mechanismů, možnost obecného popisu paralelních aktivit
- Funkce primitiv:

    - fork X - provedení primitiva „fork x“ způsobí spuštění nového vlákna od příkazu označeného návěštím x; nové vlákno bude běžet paralelně s původním vláknem.
    - quit - ukončí vlákno
    - join t, Y - atomicky provede: t:=t-1; if t = 0 then goto Y;
- Abstraktní primitiva cobegin a coend
    - Explicitně specifikuje sekvenci programu, která má být spuštěna paralelně
    - Cobegin má formát:

```
cobegin
```

C1 || C2 || … || Cn

```
coend
```

- Výsledkem je vytvoření samostatného vlákna pro všechna Ci
- Každé Ci běží nezávislé na ostatních vláknech v konstrukci cobegin / coend
- Program pokračuje za coend až po skončení posledního Ci

Vlákna

- Vlakna v procesu sdileji adresni prostor, otevrene soubory (atributy procesu)
- Vlakna maji soukromy citac instrukci, obsah registru, soukromy zasobnik
- Mohou mit soukrome lokalni promenne
- Puvodne vyuzivana zejmena pro VT vypocty na multiprocesorech (kazdé vlakno vlastni CPU, spolecna data)

PCB

= process control block

OS (konkrétně správce procesů) udržuje tabulku nazývanou tabulka procesů

- PCB obsahuje všechny info potřebné pro opětovné spuštění procesu
- Konkrétní obsah PCB – různý
- Pole správy procesů, správy paměti, správy souborů

Položky PCB

- Identifikátor procesu, uživatele
- Stavová informace procesoru (registry, PC, SP, styv CPU)
- Plánovací parametry procesu
- Odkazy na rodiče a potomky
- Čas spuštení, čas spotřebovaný na CPU
- Nastavení meziprocesorové komunikace
- Popis paměti, data, zásobník
- Otevřené soubory, aktuální pracovní adresář...

PCB

Pointer	Process state
Process number	
Program counter	
Registers	
Memory limits	
List of open files	
...	

Vlákna

Jeden proces se může skládat z jednoho nebo více vláken.

Vlákna v procesu sdílejí adresní prostor, ale mají vlastní čítač instrukcí, obsah registrů, zásobník. Taky mohou mít soukromé lokální proměnné.

- interaktivní procesy (něco beží na pozadí a zároveň probíhá komunikace s uživatelem)
- www server (každý klient jedno vlákno)
- textový procesor (vstup dat, formátoovávání textu)

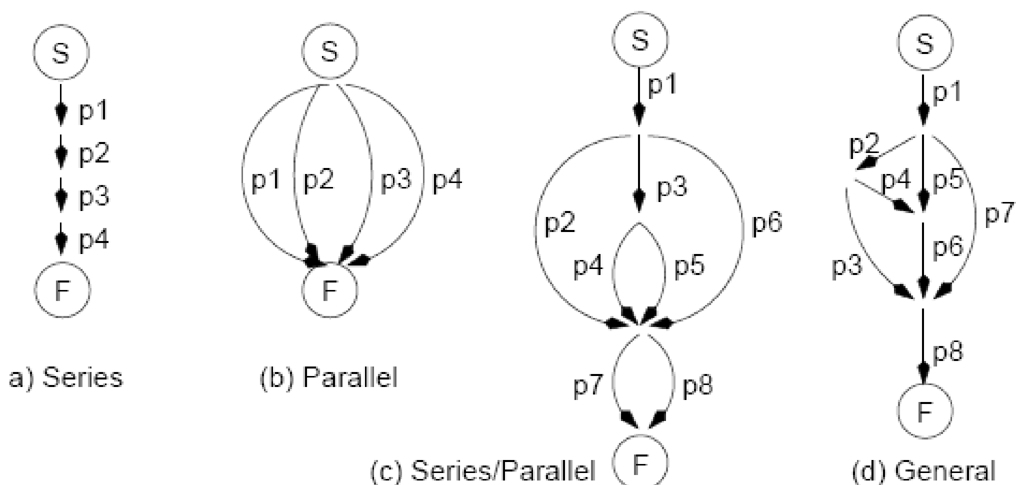
Multithreading - více vláken. Proces začíná s jedním vláknem, další si vytváří za běhu.

Režije na vytvoření vlákna a přepnutí kontextu je menší než u procesů.

Vlákna popisujeme pomocí Precedenčních grafů

Precedenční graf

Acyklický orientovaný graf, popisuje vztahy mezi procesy.



a) Series (b) Parallel (c) Series/Parallel (d) General

Lze také zapsat pomocí funkcí:

- P(a,b) - paralelní běh 2 vláken (a,b)
- S(a,b) - sériový běh 2 vláken (a,b)

např: S (p1, S(P(p2, P(S(p3, P(p4,p5)), p6)), P(p7,p8))

5
Problém kritické sekce

Časový souběh

Pokud procesy sdílejí společnou paměť, kterou čtou a zapisují, může nastat časový souběh. Hledání časových souběhů v reálných programech není jednoduché, protože časový souběh se obvykle projevuje nedeterministicky a programy běží po většinu času bez problémů.

Příklad - Přístup do souboru

- 2 procesy chtějí vytvořit soubor a zapsat do něj
- 1. proces – zjistí, že soubor není
- ... přeplánování ...
- 2. proces – zjistí, že soubor není, vytvoří a zapíše
- 1. proces – pokračuje, vytvoří a zapíše znehodnotí činnost druhého procesu

Řešení - časový souběh by nenastal, pokud by čtení+modifikace proběhli atomicky, tj. jako jedna nedělitelná operace. Zařídit HW většinou není praktické. SW řešení - v jednom okamžiku dovolíme číst a zapisovat společná data pouze jednomu procesu.

Kritická sekce

= místo programu, kde je prováděn přístup ke společným datům

Dobré řešení musí mít 3 vlastnosti:

1. Vzájemné vyloučení: žádné 2 procesy nesmějí být současně uvnitř své kritické sekce
2. Proces běžící mimo svou kritickou sekci nesmí blokovat jiné procesy
3. Žádný proces nesmí na vstup do své kritické sekce čekat nekonečně dlouho

6

Prostředky pro synchronizaci procesů

Zakázání přerušení

- V systému se sdílením času přepíná CPU mezi procesy pouze jako důsledek přerušení
- Zakážeme-li přerušení, k přepínání nedochází
- Nejjednodušší řešení, možné pouze v jednoprocesorovém systému
- Není dovoleno v uživatelském režimu

Řešení s aktivním čekáním Základní předpoklady o systému:

1. Zápis a čtení ze společné datové oblasti jsou nedělitelné operace
2. Kritické sekce nemohou mít přiřazenou prioritu
3. Relativní rychlost procesů je neznámá
4. Proces se může pozastavit mimo kritickou sekci

Průběžně se testuje, zra už může proces do kritické sekce vstoupit - plýtvání časem CPU

Petersonovo řešení

- Na začátku není v kritické sekci žádný proces
- Proces 0 volá enter_CS(0)
    - Nastaví interested[0]:=true, turn:=0;
    - Protože interested[1]=false, nebude čekat ve smyčce
- Pokud proces 1 volá enter_CS(1)
    - Nastaví interested[1]:=true, turn:=1;
    - Bude čekat ve smyčce, dokud se interested[0] nenastaví na false (voláním leave_CS(0))
- Co kdyby oba procesy volaly enter_CS téměř současně?
    - Oba nastaví interested na true
    - Oba nastaví turn na své číslo; „téměř souběžný zápis“ se ale provede sekvenčně, tj. nejdříve nastaví turn jeden, hodnota bude přepsána druhým
    - Oba se dostanou do while, proces 0 projde, proces 1 aktivně čeká

Spin-lock s instrukcí TSL

Většina současných počítačů má instrukci, která otestuje hodnotu a nastaví paměťové místo v jedné nedělitelné operaci (TSL - „Test and Set Lock“) - HW podpora

- Proměnná zámek - na počátku 0
- Proces, který chce vstoupit do KS otestuje
    - Pokud 0 nastaví na 1 a vstoupí do KS
    - Pokud 1 čeká
- Problém časového souběhu (pokud by TSL nebyla atomická):
    - Jeden proces přečte, vidí 0
    - Druhý proces je naplánován, přečte, vidí 0, nastaví na 1, vstoupí do KS

    - Po naplánování první zapíše 1 a máme 2 procesy v KS
- Řešení vyžaduje HW podporu

Další možnosti

- Semafory
- Mutexy - někdy nepotřebujeme použít schopnost semaforů čítat, tj. stačila by nám pouze jejich schopnost zajistit vzájemné vyloučení => mutex
- Monitory

Předávání zpráv - primitiva send a receive

1. Čeká-li primitivum send na převzetí zprávy příjemcem, je blokující (synchronní). Ve většině systémů je neblokující.
2. Pokud při zavolání recese není ve frontě žádná zpráva, může se recese buď zablokovat, nebo se může vrátit s chybou. Ve většině systémů je blokující.

7 Semafory, jejich použití a implementace

Semafor = proměnná, obsahuje nezáporné celé číslo

Semaforu, lze přiřadit hodnotu pouze při deklaraci

Nad semafory pouze operace P(s) a V(s):

- Operace P(S): pokud S>0, sníží S o 1, jinak pozastaví proces
- Operace V(S): pokud je nad semaforem S zablokovaný jeden nebo více procesů, vzbudí jeden z nich (náhodně), jinak zvýší S o 1

Operace P i V jsou nedělitelné (atomické) akce, tj. jakmile započne akce nad semaforem, nikdo k němu nemůže přistoupit. Když se několik procesů pokouší přistoupit současně ke stejnému semaforu, operace se provedou sekvenčně v libovolném pořadí.

Vzájemné vyloučení pomocí semaforů:

- Vytvoříme semafor s počáteční hodnotou 1
- Před vstupem do KS voláme P(s), po vystoupení V(s)
- Je-li libovolný proces v KS, je s=0, jinak s=1

Implementace

- Procesům poskytneme možnost se pozastavit při operaci P
- Aktivace operací V

S každým semaforem s je sdruženo:

- Celočíselná proměnná s.c, dovolíme jí nabývat i záporných hodnot, pak absolutní hodnota s.c vyjadřuje počet blokovaných procesů
- Binární semafor s.mutex (mutual exclusion) pro vzájemné vyloučení při operaci nad semaforem
- Seznam blokovaných procesů s.L

Proces, který nemůže dokončit operaci P bude zablokován a uložen do seznamu blokovaných procesů. Pokud při operaci V není seznam prázdný, vybere se ze seznamu jeden proces a odblokuje se.

Pozor, mutex není binární semafor. Mutex může na rozdíl od semaforu odemknout pouze vlákno, které ho zamklo.

8 Monitory

Monitor je synchronizační primitivum, které se používá pro řízeni přístupu ke sdíleným prostředkům. Jeho zvláštností je, že jde o speciální konstrukci programovacího jazyka (musí ho tedy implementovat překladač), typicky implementovanou pomocí jineho synchronizačního primitiva. Výhodou monitoru oproti jiným primitivum je jeho vysokoúrovňovost - snadněji se používá a je bezpečnější. Při jeho použití je méně pravděpodobné, že programátor udělá chybu.

- Monitor je speciální typ modulu, ve kterém jsou sdružena data a procedury, které s nimi mohou manipulovat
- Monitory musí umět rozpoznat překladač a přeložit je do odpovídajícího kódu
- Procesy mohou volat procedury monitoru, ale nemohou přímo přistupovat k datům monitoru
- V monitoru může být v jednu chvíli AKTIVNÍ pouze jeden proces; ostatní jsou při pokusu o vstup do monitoru pozastaveny.
- Monitory poskytují speciální typ proměnné nazývané podmínka
- Podmínky mohou být definovány a použity pouze uvnitř monitoru
- Nad podmínkami jsou definovány dvě operace:
  - c.wait - volající bude pozastaven nad podmínkou c. Pokud je některý proces připraven vstoupit do monitoru, bude mu to dovoleno
  - c.signal - pokud existuje jeden nebo více procesů pozastavených nad podmínkou c, reaktivuje jeden z pozastavených procesů, tj. bude mu dovoleno pokračovat v běhu uvnitř monitoru pokud nad podmínkou nespí žádný proces, nedělá nic

Pozor, pokud by signál pouze vzbudil proces, běžely by v monitoru dva. Možná řešení:

- Hoare - proces volající c.signal se pozastaví a vzbudí se až poté co předchozí rektivovaný proces opustí monitor nebo se pozastaví
- Hansen - Signal smí být uveden pouze jako poslední příkaz v monitoru. Po volání signal musí proces opustit monitor

V Javě jsou monitory implementovány pomocí klíčového slova synchronized. Zde je problém více vláken v monitoru vyřešen tak, že čekající může běžet až poté, co proces volající signál opustí monitor.

9

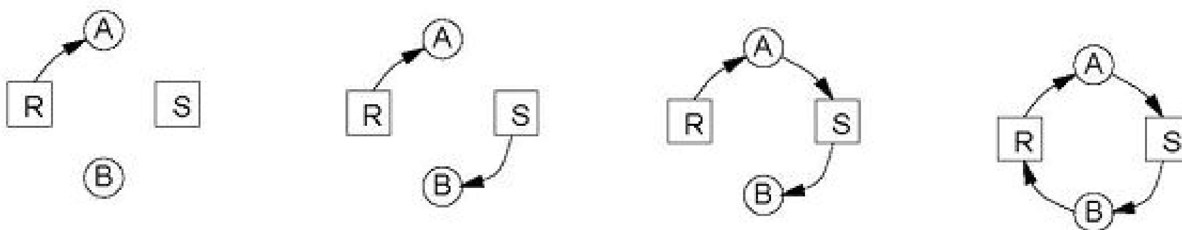
Problém uvíznutí procesů, graf alokace zdrojů

V množině procesů nastalo uvíznutí, jestliže každý proces množiny čeká na událost, kterou může způsobit pouze jiný proces množiny. Protože všechny čekají, žádný z nich událost nevygeneruje, nevzbudí jiný proces.

Podmínky vzniku uvíznutí (musejí být splněny všechny 4, 1-3 jsou předpoklad pro 4):

- Vzájemné vyloučení: každý zdroj je buď dostupný, nebo je výhradně přiřazen právě jednomu procesu
- „hold and wait“: proces držící výhradně přiřazené zdroje může požadovat další zdroje
- Nemožnost odejmutí: jednomu přiřazené zdroje nemohou být procesu násilně odejmuty
- Cyklické čekání: musí být cyklický řetězec dvou nebo více procesů, kde každý z nich čeká na zdroj držený dalším členem

- tyto 4 podmínky mohou být modelovány pomocí orientovaných grafů = graf alokace zdrojů
- graf má 2 typy uzlů:
  - proces - zobrazujeme jako kruh
  - zdroj - zobrazujeme jako čtverec
- význam hran:
  - od zdroje k procesu: zdroj držen procesem
  - od procesu ke zdroji: proces je blokován čekáním na zdroj
- cyklus v grafu je nutnou a postačující podmínkou pro vznik uvíznutí



Strategie zacházení s uvíznutím:

1. Ignorování = „pštrosí algoritmus“
2. Detekce a zotavení (odejmutí zdroje, zrušení změn, zrušení procesu)
3. Dynamické zabrání (pečlivá alokace zdrojů - systém rozhodne zda je bezpečné požadovaný zdroj přidělit) - mělo by předejít uvíznutí (bankéřův algoritmus)
4. Prevence pomocí strukturální negace jedné z podmínek pro vznik uvíznutí

10

Klasické problémy meziprocesové komunikace – producent-konzument aj.

Meziprocesova komunikace: // tady by to chtělo rozvést a doplnit...

- Predavani zprav
- Primitiva send, receive
- Mailbox, port
- RPC
- Ekvivalence semaforu, zprav, ...
- Bariera, problem vecericich filozofu

Problem sdilene pameti

- umisteni objektu ve sdilene pameti
- Bezpecnost - globalni data pristupna kteremukoliv procesu bez ohledu na semafor
- Procesy bezi na ruznych strojich, komunikuji spolu po siti Reseni - predavani zprav

Obsah

- 1 Predavani zprav
- 2 Volání vzdálených procedur (RPC - Remote Procedure Call)
- 3 Ekvivalenty uvedených primitiv
- 4 Bariéry
- 5 Producent-konzument
- 6 Problém večeřících filosofů
- 7 Problém čtenářů a písařů
- 8 Problémy

Predavani zprav

- send, receive
- Zavedeme 2 primitiva
  - send (adresat, zprava) - odeslani zpravy
  - receive (odesilatel, zprava) - prijem zpravy
  - Send - Zprava (lib. datovy objekt) bude zaslana adresatovi
  - Receive - Prijem zpravy od urceneho odesilatele Prijata zprava se ulozi do promenne "zprava"

Synchronizace - blokující (synchronní) (recieve) / neblokující (asynchronní) (send)

- blokující send
    - čeká na převzetí správy příjemcem
- neblokující send
    - vrací se ihned po odeslání zprávy
    - většina systémů
- blokující receive
    - není-li ve frontě žádná zpráva, zablokuje se
    - většina systémů
- neblokující receive
    - není-li zpráva, vrací chybu

problémy, které nejsou u semaforů ani monitorů, zvláště při komunikaci po síti

- ztráta zprávy (potvrzení o přijetí (acknowledgement))
- ztráta potvrzení (číslování zpráv, duplicitní zprávy se ignorují)
- problém autentizace (zprávy je možné šifrovat)

lokální komunikace

- rendezvous - eliminuje frontu zpráv, send zavolán dříve než receive – odesílatel zablokován, vyvolán send i receive – zprávu zkopírovat z odesílatele přímo do příjemce
- využití mechanismu virtuální paměti paměť obsahující zprávu je přemapována z procesu odesílatele do procesu příjemce

typické aplikace typu klient - server

- WWW klient x WWW server (Apache, IIS)
- účetní aplikace x databázový systém (Oracle, MySQL)

Volání vzdálených procedur (RPC - Remote Procedure Call)

1. Klient zavolá spojku klienta, reprezentující vzdálenou proceduru
2. Spojková procedura argumenty zabalí do zprávy, pošle ji serveru
3. Spojka serveru zprávu přijme, vezme argumenty a zavolá proceduru
4. Procedura se vrátí, návrat. hodnotu pošle spojka serveru zpět klientovi
5. Spojka klienta přijme zprávu obsahující návrat. hodnotu a předá ji volajícímu

dnes nejpoužívanější jazyková konstrukce pro implementaci distribuovaných systémů a programů bez explicitního předávání zpráv

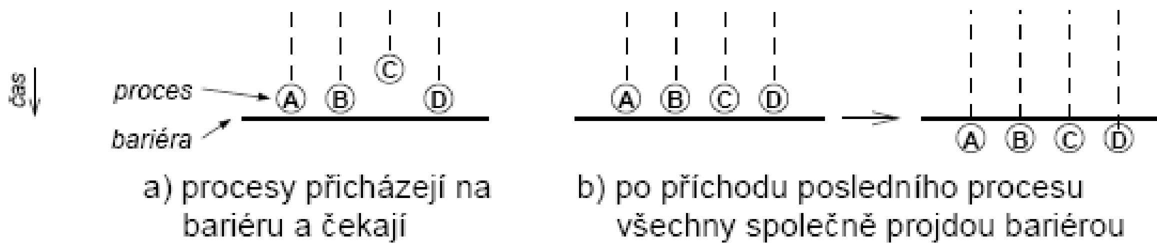
Ekvivalenty uvedených primitiv

Lze implementovat semafory pomocí zpráv a zprávy pomocí semaforů

Lze ukázat, že je možné implementovat

- Semafory pomocí monitoru
- Monitory pomocí semaforů

Bariéry



Producent-konzument

- Dva procesy sdílejí společnou paměť (buffer) pevné velikosti N položek
- Jeden proces je producent - generuje nové položky a ukládá je do vyrovnávací paměti
- Paralelně běží proces konzument, který data vyjímá a spotřebovává
- Procesy mohou běžet různými rychlostmi => musí být zabezpečeno, aby nedošlo k přetečení/podtečení:
    - Konzument musí být schopen čekat na producenta, nejsou-li data
    - Producent musí být schopen čekat na konzumenta, je-li buffer plný

Klasicky se zde sdílená paměť řeší pomocí pole, kde se z jedné strany položky přidávají a z druhé odebírají. Index kam zapisovat/odebírat počítáme modulo velikost pole. Kolizím zabráníme pomocí dvou semaforů, kde jeden obsahuje informaci o počtu plných prvků pole (f) a druhý o počtu prázdných prvků (e).

- přidání prvku: P(e), V(f)
- odebrání prvku: P(f), V(e)

Problém večeřících filosofů

- 5 filosofů sedí kolem kulatého stolu
- Každý filosof má před sebou talíř se špagetami
- Mezi každými dvěma talíři je vidlička
- Špagety jsou tak klouzavé, že filosof potřebuje 2 vidličky, aby mohl jíst
- Když filosof dostane hlad, pokusí se vzít 2 vidličky; pokud uspěje, nějakou dobu jí, pak položí vidličky a pokračuje v přemýšlení
- Problémy: uvíznutí(deadlock) (všichni filozofové zvednou levou vidličku, žádný z nich už nemůže pokračovat), vyhladovění (pokud by filozofové vzali najednou levou vidličku, budou běžet cyklicky - vidí, že pravá není volná, položí…)
- Řešení - Dijkstra (pomocí semaforů)

Problém čtenářů a písařů

- Představme si velkou databázi, množina procesů se pokouší souběžně číst a zapisovat
- Více požadavků čtení - akceptovatelné
- Při zápisu nesmí nikdo jiný přistupovat, ani žádní čtenáři
- První čtenář, který dostane přístup do databáze, provede P(w), další pouze zvětšují čítač
- Po skončení čtenáři zmenšují čítač, poslední provede V(w)
- Semafor w zabrání vstupu písaře, pokud jsou čtenáři
- Semafor w také zabrání vstupu čtenářům, je-li písař
- toto je s předností čtenářů

- nebo to jde udělat tak, že když chce vstoupit písař, tak už tam nepustí další čtenáře (s předností písařů)

Problémy

- Uvíznutí (Deadlock) - cyklické čekání dvou či více procesů na událost, kterou může vyvolat pouze některý z nich, nikdy k tomu však nedojde
- Vyhladovění (starvation) - proces se nedostane k požadovaným zdrojům

11
Plánování úloh a procesů v dávkových systémech

- Non-preemtivní plánování: proces si podrží kontrolu nad CPU dokud se jí nevzdá
- Preemtivní - proces lze přerušit během CPU burstu a naplánovat jiný

Bere se ohled na:

- průchodnost (počet úloh dokončených za časovou jednotku)
- průměrnou dobu obrátky (průměrná doba od zadání úlohy do systému do dokončení úlohy)
- využití CPU

Obsah

- 1 First-Come First-Served (FCFS)
- 2 Shortest Job First (SJF)
- 3 Shortest Remaining Time (SRT)
- 4 Multilevel Feedback

First-Come First-Served (FCFS)

- Někdy používán název FIFO
- Zpracování podle času příchodu úlohy, „kdo dřív přijde, ten dřív mele“
- Nepreemptivní FCFS je nejjednodušší plánovací algoritmus (úloha běží dokud neskončí)
- Je spravedlivý
- Během I/O operací se CPU nevyužívá, ale přesto je blokováno stávajícím procesem

Shortest Job First (SJF)

- Předpoklad, že je předem známá doba vykonávání úlohy
- Nepreemptivní, jedna fronta příchozích úloh, plánovač vybere vždy úlohu s nejkratší dobou běhu
- Nevýhodou je, že na dlouho trvající úlohy se nemusí vůbec dostat

Shortest Remaining Time (SRT)

- Preemptivní varianta SJF
- Plánovač vždy vybere úlohu, jejíž zbývající doba běhu je nejkratší
- Např. právě prováděné úloze to bude trvat 10 min., do systému přijde úloha trvající 1 minutu - systém prováděnou úlohu pozastaví a nechá běžet novou úlohu
- Problém: možnost vyhladovění úloh

Multilevel Feedback

- N rozdílných prioritních úrovní, každá úroveň má svou frontu úloh
- Úloha vstoupí do systému s nejvyšší prioritou
- Na každé prioritní úrovni je stanoveno maximu času CPU, které může úloha na této úrovni obdržet
  - Pokud úloha překročí limit, její priorita se sníží
  - Na nejnižší úrovni může proces běžet neustále nebo může být překročení považováno za chybu
- Procesor obsluhuje nejvyšší neprázdnou frontu
- Výhoda: rozlišuje mezi I/O a CPU úlohami (upřednostňuje I/O)

12
Plánování procesů v interaktivních systémech

Základní stavy procesu

- běžící
- připravený - čeká na CPU
- blokovaný - čeká na zdroj nebo zprávu
- novy (new) - proces byl právě vytvořen
- ukončený (terminated) - proces byl ukončen

Správce procesů - udržuje tabulku procesů. Záznam o konkretním procesu - PCB (Process Control Block) - souhrn dat potrebnych k řízení procesu.

Plánovač vs. dispatcher

- dispatcher předává řízení procesu vybranému short time planovacem
  - prepnuti kontextu
  - prepnuti do user modu
  - skok na vhodnou instrukci daneho programu
- vice pripravenych procesu k behu { planovac vybere, ktery spusti jako prvni
- planovac procesu (scheduler) - pouziva planovaci algoritmus (scheduling algorithm)
- Preemptivni vs. non-preemptive planovani

Obsah

- 1 Plánování v interaktivních systémech
- 2 Algoritmus cyklické obsluhy (Round RObin)
- 3 Prioritní plánování
- 4 Plánovač spravedlivého sdílení
- 5 Plánování pomocí loterie
- 6 Plánování vláken

Plánování v interaktivních systémech

- preemptivní

- Základní problém - každý proces je jedinečný a nepredikovatelný, nedá se říci, jak dlouho poběží, než se zablokuje
- Aby proces neběžel příliš dlouho, má počítač vestavěné hodiny, které provádějí pravidelné přerušení
- Při přerušení se vyvolá obslužný podprogram přerušení v jádře
- OS rozhodne, zda dovolí procesu pokračovat nebo zda dá CPU jinému procesu

Algoritmus cyklické obsluhy (Round RObin)

- Každému procesu je přiřazen časový interval = časové kvantum, po které může běžet
- Pokud běží ještě na konci kvanta, je provedena preempce a je naplánován a spuštěn další připravený proces
- Pokud proces skončil, nebo se zablokoval ještě před spotřebováním časového kvanta, je také naplánován a spuštěn další připravený proces
- Délka časového kvanta:
  - Krátké: vysoká režie
  - Dlouhé: vyšší efektivita, může zhoršovat dobu odpovědi
- znevýhodnění I/O vázaných úloh, protože zpravidla využijí pouze malou část kvanta a zablokují se (výpočetně vázané úlohy jsou zvýhodněné)

Prioritní plánování

- Interaktivní procesy mohou mít vyšší prioritu než procesy běžící na pozadí
  - Staticky - např. při startu procesu
  - Dynamicky - např. pokud mají I/O procesy vyšší prioritu, budou se moci po krátkém použití CPU opět zablokovat. S délkou běhu klesá dynamická priorita
- celková priorita = statická + dynamická
- rozdělení procesů do priotitních tříd, v nich je využíváno round robin

Plánovač spravedlivého sdílení

- Přidělovat čas každému uživateli proporciálně bez ohledu na to, kolik má procesů

Plánování pomocí loterie

- Procesy obdrží tikety (losy)
- Plánovač vybere náhodně jeden tiket
- Vítězný proces obdrží cenu - 1 kvantum času na CPU
- Důležitější procesy mohou obdržet více tiketů, aby se zvýšila jejich šance na výhru
- Spolupracující procesy si mohou předávat losy

Plánování vláken

Buď jsou vlákna plánována v OS nebo ve vlastním procesu (obvykle pomocí RR)

13
Správa hlavní paměti, metody přidělování paměti, virtuální paměť

Ideál programátora

- Pameť nekonecně velká, rychlá, levná
- Zároveň persistentní (uchovává obsah po vypnutí)
- Bohužel neexistuje

Reálný počítač - hierarchie paměti ("pyramida")

1. Registry CPU
2. Malé množství rychlé cache paměti
3. Stovky MB až gigabajty RAM paměti
4. GB na pomalých, levných, persistentních discích

Modul pro spravu pameti

- informace o prideleni pameti
- ktera cast je volna pridelena (a kteremu procesu)
- pridelovani pameti na zadost
- uvolneni pameti, zarazeni k volne pameti
- odebira pamet' procesum
- ochrana pameti
  - pristup k pameti jineho procesu
  - pristup k pameti OS

Funkce MMU (memory managment unit)

- Dostava adresu od CPU, prevadi na adresu do fyzicke pameti
- Nejprve zkontroluje, zda adresa neni vetsi nez limit Ano - vyjimka, Ne - k adrese pricte bazi
- Pokud baze 1000, limit 60
  - Adresa 55 - ok, vysledek 1055
  - Adresa 66 - není ok, vyjimka

Tri varianty rozdeleni pameti

- OS ve spodni casti adresniho prostoru v RAM (minipocitace)
- OS v horni casti adresniho prostoru v ROM (zapouzdrene 2. systemy)
- OS v RAM, ovladace v ROM (PC { MS DOS v RAM, BIOS v 3. ROM)

Část OS, která spravuje paměť, se nazývá správce paměti

- Udržuje informaci, které části paměti se používají a které jsou volné
- Alokuje paměť procesům podle potřeby
- Zařazuje paměť do volné paměti po uvolnění procesem

- Jednoprogramové systémy bez odkládání a stránkování
  - Nejjednodušší - spouštíme pouze jeden program v jednom čase
  - Dovoluje procesu použít veškerou paměť, kterou nepotřebuje OS
  - Po skončení procesu je možné spustit další proces
- Multiprogramování s pevným přidělením paměti
- Nejjednodušší schéma = rozdělit paměť na n oblastí (mohou být i různé velikosti)
  - V historických systémech se provádělo ručně při startu zdroje
  - Po načtení úlohy je obvykle část oblasti nevyužitá
  - Snaha umístit úlohu do nejmenší oblasti, do které se vejde
- Multiprogramování s proměnnou velikostí oblasti
- Každé úloze je přidělena paměť podle požadavku
- Obsazení paměti se postupně mění, jak úlohy přicházejí a končí
- Zlepšuje využití paměti
- Postupem času může vzniknout mnoho malých volných oblastí
- OS musí vědět, která paměť je volná a která alokovaná

Obsah

- 1 Nejpoužívanější způsoby správy paměti
- 2 Správa paměti pomocí bitových map
- 3 Správa paměti pomocí seznamů
- 4 Mechanismus „buddy system“
- 5 Virtuální paměť

Nejpoužívanější způsoby správy paměti

níže

Správa paměti pomocí bitových map

- Paměť rozdělena do alokačních jednotek stejné délky
- S každou alokační jednotkou sdružen jeden bit (0 = volno, 1 = obsazeno), tyto bity jsou pohromadě v bitové mapě
  - menší alokační jednotky = větší bitmapa
  - Větší alokační jednotky = více nevyužité paměti, protože velikost procesu nebude přesně násobek alokační jednotky
- Výhoda - konstantní velikost bitmapy
- Nevýhoda - pokud požadujeme úsek paměti velikosti N alokačních jednotek, musí se v bitmapě vyhledat N po sobě následujících nulových bitů (drahá operace)

Správa paměti pomocí seznamů

- Myšlenka udržovat seznam alokovaných a volných oblastí
- Každá položka seznamu obsahuje:
  - Informaci, zda se jedná o proces nebo díru (P nebo H)
  - Počáteční adresu oblasti
  - Délku oblasti

- Práce se seznamem:
  - Pokud proces skončí, nahradí se dírou
  - Pokud jsou vedle sebe dvě díry, sloučí se
- Seznam je dobré mít seřazený podle počáteční adresy oblasti
- zároveň je seznam obousměrný, takže se snadno vrací

Možnosti alokace:

- Algoritmus first fit
  - Prohledávání seznamu, dokud se nenajde dostatečně velká díra
  - Rychlý
  - díra se rozdělí na proces a zbytek díry

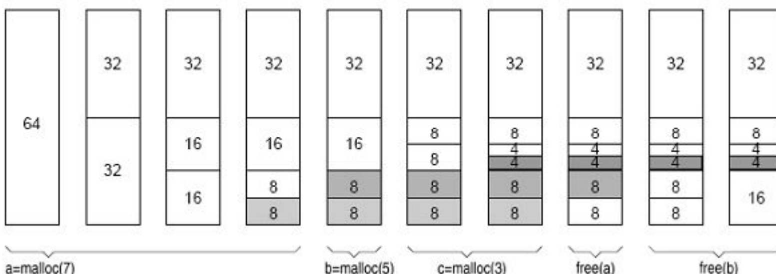
- Algoritmus next fit
  - Prohledávání začne tam, kde skončilo předchozí
  - o trochu pomalejší než first-fit

- Algoritmus best fit
  - Prohledá celý seznam, vezme nejmenší díru, do které se proces vejde
  - Pomalejší, protože prohledává celý seznam
  - zase ale najde nejlepší místo (byť na druhou stranu pak zbyte spousta malých nepoužitelných děr)

Možná vylepšení

- oddělěné seznamy děr a perocesů (rychlejší alokace, pomalejší dealokace
- seznam děr seřazený podle velikosti (rychlé best fit, opět náročná dealokace)
- místo samostatného seznamu děr lze využít samotné díry, které na sebe odkazují - úspora paměti

Mechanismus „buddy system“



- Mějme seznamy volných bloků velikosti 1,2,4,8,16 … alokačních jednotek až do seznamu bloků velikosti celé paměti
- Na začátku veškerá paměť volná, všechny seznamy jsou prázdné kromě seznamu obsahující 1 položku velikosti paměti
- Přijde-li požadavek, zaokrouhlí se nahoru na mocninu 2
- Blok se rozdělí na 2 bloky, pokud ještě moc velké, jeden z nich se zase rozdělí na 2 bloky atd.
- Uvolnění paměti: pokud jsou volné oba sousední bloky stejné velikosti, spojí se do jednoho
- Neefektivní ve využití paměti, ale rychlý

Příklady použití:

- Buddy system - jádro Linuxu, běží ve fyzické paměti
- First fit, Next fit - malloc v C

- Bitová mapa - SWAP v Linuxu

Virtuální paměť

- Problém, že programy jsou větší než dostupná fyzická paměť
- Chceme, aby ve skutečné paměti byla realizovaná pouze část adresového prostoru, zbytek může být odložen na disku
- Procesor používá tzv. virtuální adresy
- Pokud je požadovaná část virtuálního paměťového prostoru ve fyzické paměti, MMU převede virtuální adresu na fyzickou
- Pokud není ve fyzické paměti, OS jí musí přečíst z disku
- Většina systémů virtuální paměti používá techniku nazývanou stránkování

14

Algoritmy nahrazování stránek paměti

Obsah

- 1 Relokace při zavedení do paměti
- 2 Mechanismus báze a limitu (Dynamická relokace)
- 3 Pojmy
- 4 Výpadek stránky
- 5 Algoritmus FIFO
- 6 Algoritmus MIN/OPT
- 7 Algoritmus Least Recently Used
- 8 Algoritmus Not-Recently-Used
- 9 Algoritmus Second Chance
- 10 Algoritmus Clock
- 11 Algoritmus Aging

Relokace při zavedení do paměti

Program volá instrukci na adrese 66 ale sám v paměti běží až od adresy 1000. Tedy instrukce se nechází na adrese 1066. - Při zavedení programu do paměti provede Linker modifikaci, aby adresy souhlasily

Mechanismus báze a limitu (Dynamická relokace)

Jednotka správy paměti (MMU) mezi procesorem a pamětí obsahuje dva registry:

- báze - počáteční adresa oblasti
- limit - velikost oblasti

Dostává adresu od CPU a pževádí ji na adresu v paměti

Pojmy

- virtuální paměť - stránky (pages) stejné délky
- fyzická paměť - rámce (stejné délky)
- rámec může obsahovat právě jednu stránku
- na známém místě v paměti tabulka stránek, poskytuje mapování virtuálních stránek na rámce

Výpadek stránky

- Pokud všechny rámce obsazené a nastane výpadek stránky, je nutné některou stránku vyhodit a rámec uvolnit (dojde k přerušení, zavede se požadovaná stránka, program může pokračovat v běhu)
- Vyhození
  - Pokud byla stránka modifikována, zapíše se na disk
  - Pokud oproti kopii na disku nebyla modifikovaná, bude pouze uvolněna
- Otázka - kterou stránku vyhodit?

Tabulka stránek procesu - v MMU, obsahuje stránky daného procesu, mapuje číslo stránky na fyzickou adresu rámce, řeší realokaci a ochranu

Algoritmus FIFO

- Udržujeme seznam všech stránek v pořadí, ve kterém byly zavedeny
- Vyhazujeme nejstarší stránku (je možné, že se vyhodí stránka, která se bude brzy potřebovat)

Algoritmus MIN/OPT

- V paměti je množina stránek, každá stránka je označena počtem instrukcí, po který se k ní nebude přistupovat
- V okamžiku výpadku stránky se vybere stránka s nejvyšším označením
- Algoritmus je optimální, tj. vybere se stránka, která bude zapotřebí nejvzdáleněji v budoucnosti
- Není realizovatelný (neexistuje způsob, jakým by OS mohl zjistit, která stránka bude zapotřebí jako příští)

Algoritmus Least Recently Used

nejdele nepouzita (pohled do minulosti)

- princip lokality
  - stranky pouzivane v poslednich instrukcich se budou pravdepodobne pouzivat i v nasledujicich
  - pokud se stranka dlouho nepouzivala, pravdepodobne nebude brzy zapotrebi

Vyhazovat zbozi, na kterem je v prodejne nejvice prachu = nejdele nebylo pozadovano

- Obtizna implementace
- sw reseni (neni pouzitelne)
  - seznam stranek v poradi referenci
  - vypadek - vyhozeni stranky ze zacatku seznamu
  - zpomaleni cca 10x, nutna podpora hw
- hw reseni - citac
  - MMU obsahuje citac (64bit), pri kazdem pristupu do pameti zvetsen

- kazda polozka v tabulce stranek - pole pro ulozeni citace
- odkaz do pameti
- obsah citace se zapise do polozky pro odkazovanou stranku
- vypadek stranky - vyhodí se stranka s nejnizsim cislem

Realizece čítače pomocí matice

Algoritmus Not-Recently-Used

- Bit R - nastaven na 1 při čtení nebo zápisu do stránky
- Bit M - nastaven na 1 při zápisu do stránky; označuje, že se stránka má při vyhození zapsat na disk

Na začátku mají všechny stránky R=0, M=0, bit R je OS nastavován periodicky na 0

4 kategorie:

- Třída 0: R=0, M=0
- Třída 1: R=0, M=1
- Třída 2: R=1, M=0
- Třída 3: R=1, M=1

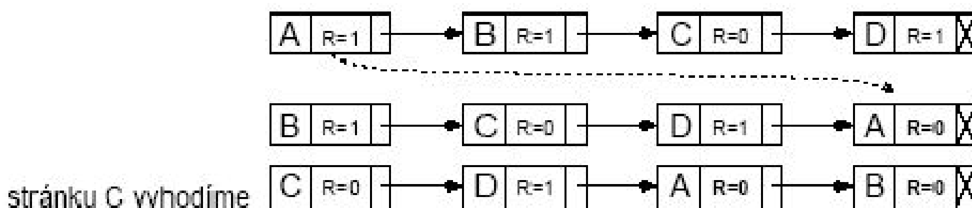
Algoritmus vyhodí stránku z nejnižší neprázdné třídy, výběr mezi stránkami ve stejné třídě je náhodný
Jednoduchý, efektivně implementovatelný, ale nevýkonný

(R = referenced)

Algoritmus Second Chance

- Vychází z algoritmu FIFO
- Vyhledává nejstarší stránku, která nebyla referencována v poslední době
- Pokud byly všechny referencovány => FIFO

Snaží se zabranit vyhozeni casto pouzivane - dle bitu R nejstarsi stranky R = 0 ... stranka je nejstarsi, nepouzivana, vyhodime. Pokud R = 1 ... nastavme R=0, presuneme na konec seznamu stranek (jako by byla nove zavedena) a vezmeme další v pořadí.



Algoritmus Clock

Optimalizace datovych struktur algoritmu Second Chance

- Stranky udrzovany v kruhovem seznamu
- Ukazatel na nejstarsi stranku { "rucicka hodin\
- Vypadek stranky { najit stranku k vyhozeni
- Stranka kam ukazuje rucicka
  - ma-li R=0, stranku vyhodime a rucicku posuneme o jednu pozici
  - ma-li R=1, nastavime R na 0, rucicku posuneme o 1 pozici, opakovani,..
- Od SC se lisi pouze implementaci
- Varianty Clock pouzivaji napr. BSD UNIX

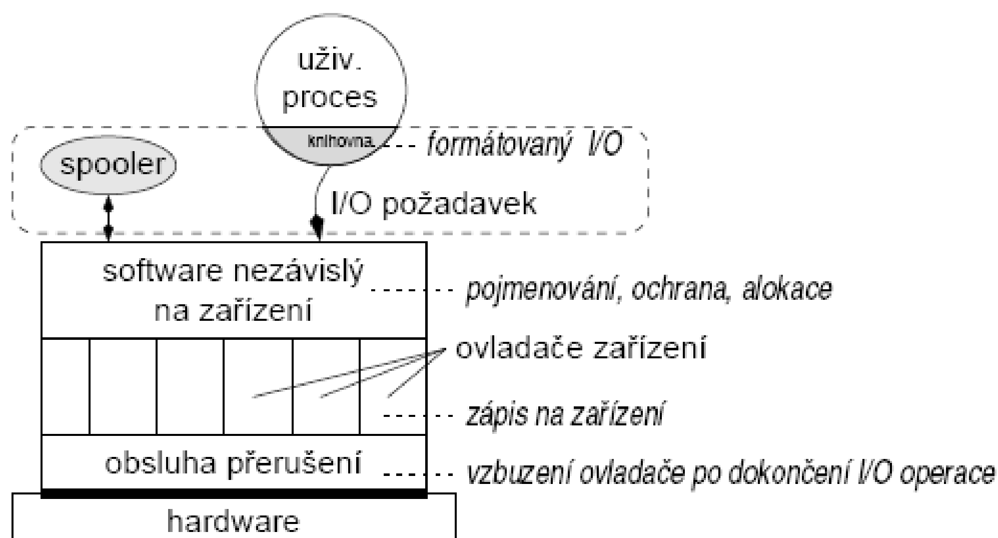
Algoritmus Aging

- Každá položka má pole „stáří“, na počátku = 0
- Při každém přerušení časovače - posun pole „stáří“ o 1 bit vpravo, zleva se přidá hodnota bitu R, nastavení R na 0
- Při výpadku stránky se vyhodí stránka, jejíž pole „stáří“ má nejnižší hodnotu

15
Ovládání periferních zařízení, RAID

Principy I/O software:

1. obsluha přerušení (nejnižší úroveň v OS)
2. ovladač zařízení
3. SW vrstva OS nezávislá na zařízení
4. uživatelský I/O SW



Obsah

- 1 Obsluha přerušení
- 2 Ovladače zařízení
- 3 SW vrstva OS nezávislá na zařízení
- 4 I/O SW v uživatelském režimu
- 5 Funkce ovladače zařízení
- 6 RAID 0
- 7 RAID 1 mirroring
- 8 RAID 5
- 9 RAID 6
- 10 RAID 10
- 11 Hot spare

Obsluha přerušení

- Řadič vyvolá přerušení ve chvíli dokončení I/O požadavku
- Ovladač zadá I/O požadavek, usne, po příchodu přerušení ho obsluha vzbudí

- Obsluha přerušení je časově kritická, musí být co nejkratší

Ovladače zařízení

- Ovladače obsahují veškerý kód závislý na I/O zařízení, jediný zná HW podrobnosti
- Ovladači předán příkaz vyšší vrstvou; pokud ještě obsluhuje předchozí požadavek, zařadí nový požadavek do fronty; zadá příkazy řadiči; zablokuje se do vykonání požadavku; po dokončení zkontroluje, zda nenastala chyba; předá výsledek vyšší vrstvě; spustí další požadavek

SW vrstva OS nezávislá na zařízení

- Poskytuje I/O funkce společně pro všechna zařízení daného druhu
- Definuje rozhraní s ovladači
- Poskytuje jednotné rozhraní uživatelskému SW

I/O SW v uživatelském režimu

- Programátor používá v programech I/O funkce nebo příkazy jazyka
- Spooling = způsob obsluhy vyhrazených I/O v multiprogramovém systému (požadavky se zařadí do fronty a počkají až na ně přijde řada)

Funkce ovladače zařízení

- ovladači předán příkaz vyšší vrstvou
- požadavek zařazen do fronty (může ještě obsluhovat předchozí)
- ovladač zadá příkazy řadiči (přijde na řadu), např. nastavení hlavy, přečtení sektoru
- zablokuje se do vykonání požadavku (neblokuje při rychlých operacích, např. zápis do registru)
- vzbuzení obsluhou přerušení (dokončení operace) - zkontroluje, zda nenastala chyba
- pokud OK, předá výsledek (status + data) vyšší vrstvě

RAID 0

- není redundantní
- ztrata 1 disku = ztrata celeho pole
- duvod pouziti - vykon napr. strih videa

RAID 1 mirroring

- zrcadleni na 2 disky stejnych kapacit
- totozne informace
- vypadek 1 disku - nevadi
- jednoducha implementace - casto ciste sw
- nevyhoda - vyuzijeme jen polovinu kapacity zapis - pomalejsi (2x) cteni - rychlejsi (radic - lze stridat pozadavky mezi disky)

RAID 5

- redundantni pole s distribuovanou paritou
- minimialne 3 disky
- rezie: 1 disk z pole n disku, pr. 5 disku 100GB, 400GB pro data
- vypadek 1 disku nevadi cteni - vykon ok zapis - pomalejsi 1 zapis - cteni starych dat, cteni stare parity, vypocet nove parity, zapis novych dat, zapis nove parity

RAID 6

- RAID 5 + navic dalsi paritni disk
- odolne proti vypadku dvou disku
- rekonstrukce pole pri vypadku - trva dlouho po dobu rekonstrukce neni pole chraneno proti vypadku dalsiho disku, narocna cinnost - muze se objevit dalsi chyba, radic disk odpoji a ...

RAID 10

- kombinace RAID 0 (stripe) a RAID 1 (zrcadlo)
- min. pocet disku 4 rezie 100% diskove kapacity navic
- nejvyssi vykon v bezpecnych typech polich podstatne rychlejsi nez RAID 5, pri zapisu odolnost proti ztrate az 50% disku x RAID 5

Hot spare

Disk který se aktivuje až při výpadku. Nahradí poškozený disk a pole je opět funkční (hot spare disk pak pouze vyměníme za nový)

Citováno z „http://www.512.cz/index.php?title=Ovl%C3%A1d%C3%A1n%C3%AD_perifern%C3%ADch_za%C5%99%C3%ADzen%C3%AD,_RAID“
Kategorie: Fav-kiv-bzinf

- Stránka byla naposledy editována 20. 2. 2014 v 06:48.
- Stránka byla zobrazena 1 177krát.

16
Systémy souborů

Téměř všechny aplikace potřebují trvale uchovávat data

Hlavní požadavky:

- Možnost uložit velké množství dat
- informace musí zůstat zachována i po ukončení procesu
- data musejí být přístupná více procesům

při přístupu k zařízení mají všechny procesy společné problémy

- alokace prostoru na disku
- pojmenování dat
- ochrana dat před neoprávněným přístupem
- zotavení po havárii

souborový systém:

- konvence pro ukládání a přístup k souborům
- část OS, která poskytuje mechanismus pro ukládání a přístup k datům

uživatelské rozhraní:

- konvence pro pojmenování souborů
- typy souborů
- způsob přístupu
- atributy a přístupová práva
- služby OS pro práci se soubory

Windows: NTFS, FAT12, FAT16, FAT32, ISO 9660 (CD-ROM)

Linux: ext2, ext3, ReiserFS, JFS, XFS, FAT12 až 32, ISO 9660, Minix, VxFS, OS/2 HPFS, SysV fs, UFS, NTFS read-only

- soubor – pojmenovaná množina záznamů, které lze zpracovávat jako celek
- záznam – strukturovaný datový objekt tvořený konečným počtem pojmenovaných položek

atributy - informace sdružené se souborem, liší se podle jednotlivých OS / FS

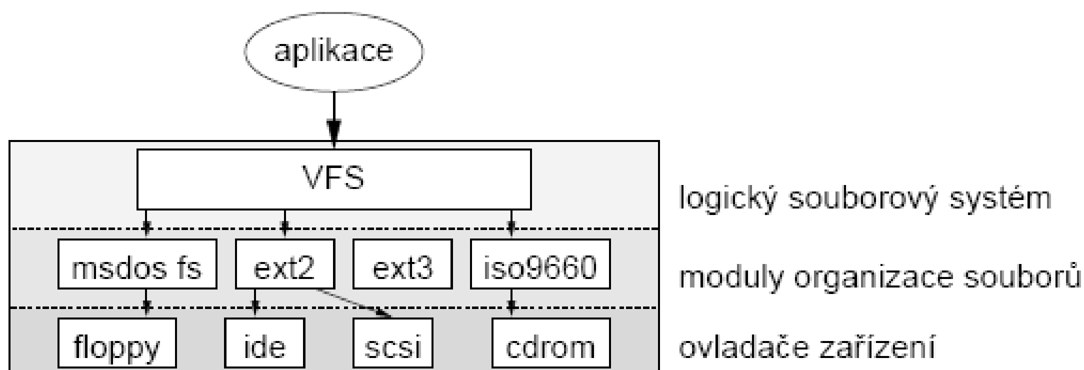
- příznaky (hidden, archive, temporary, read-only...)
- velikost, datum vytvoření, poslední modifikace...

Veškeré I/O jsou dnes (díky UNIXu) prováděny jako práce se soubory - interní struktura souboru OS nezajímá, adresář je také soubor

Základní operace pro práci se soubory

- otevření
- vytvoření
- čtení
- zápis
- nastavení pozice v souboru
- zavření souboru

Virtuální FS



Kód společný pro všechny typy FS, volán aplikacemi, udržuje informaci o otevřeném souboru, ověřuje přístupová práva.

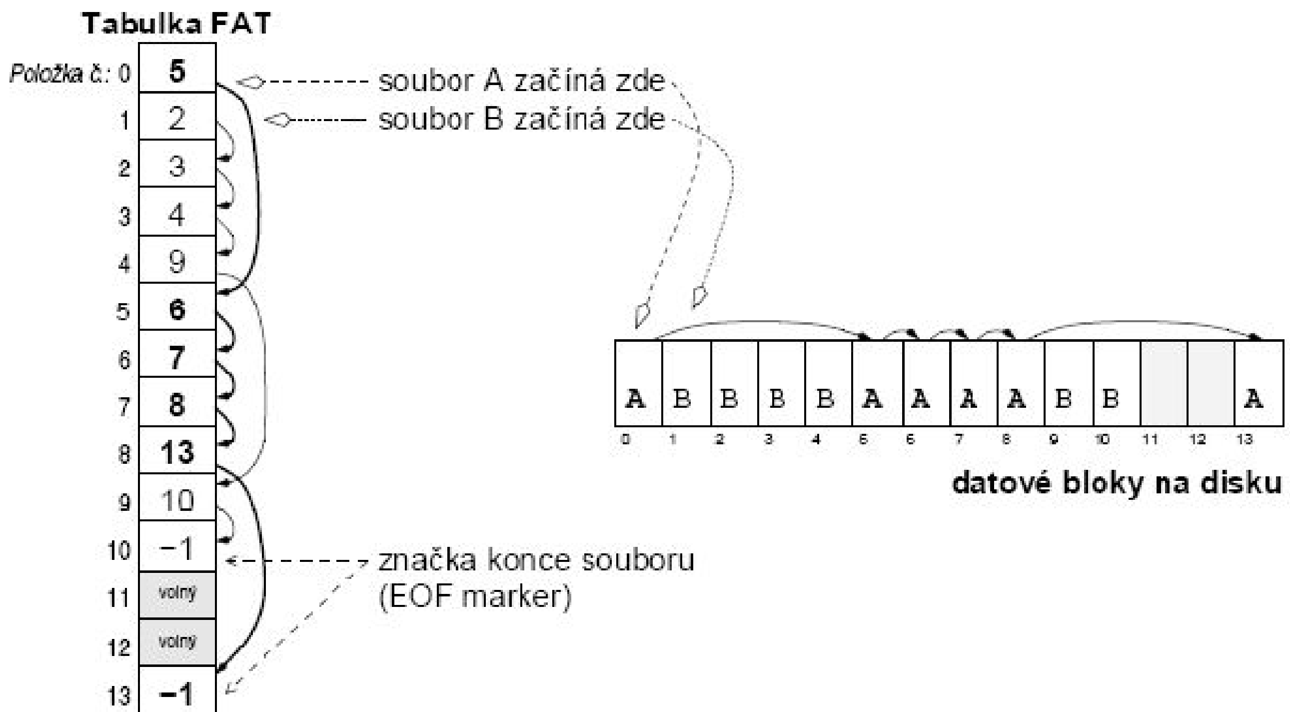
Typy FS

- kontunuální alokace (jen u read-only médií - např CD)
- seznam diskových bloků (blok obsahuje data a odkaz na další blok)
- FAT

FAT

Implementace:

- Každému bloku odpovídá jedna položka v tabulce FAT
- Položka FAT obsahuje číslo dalšího bloku souboru
- Řetězec odkazů je ukončen speciální značkou, která není platným číslem bloku (-1)

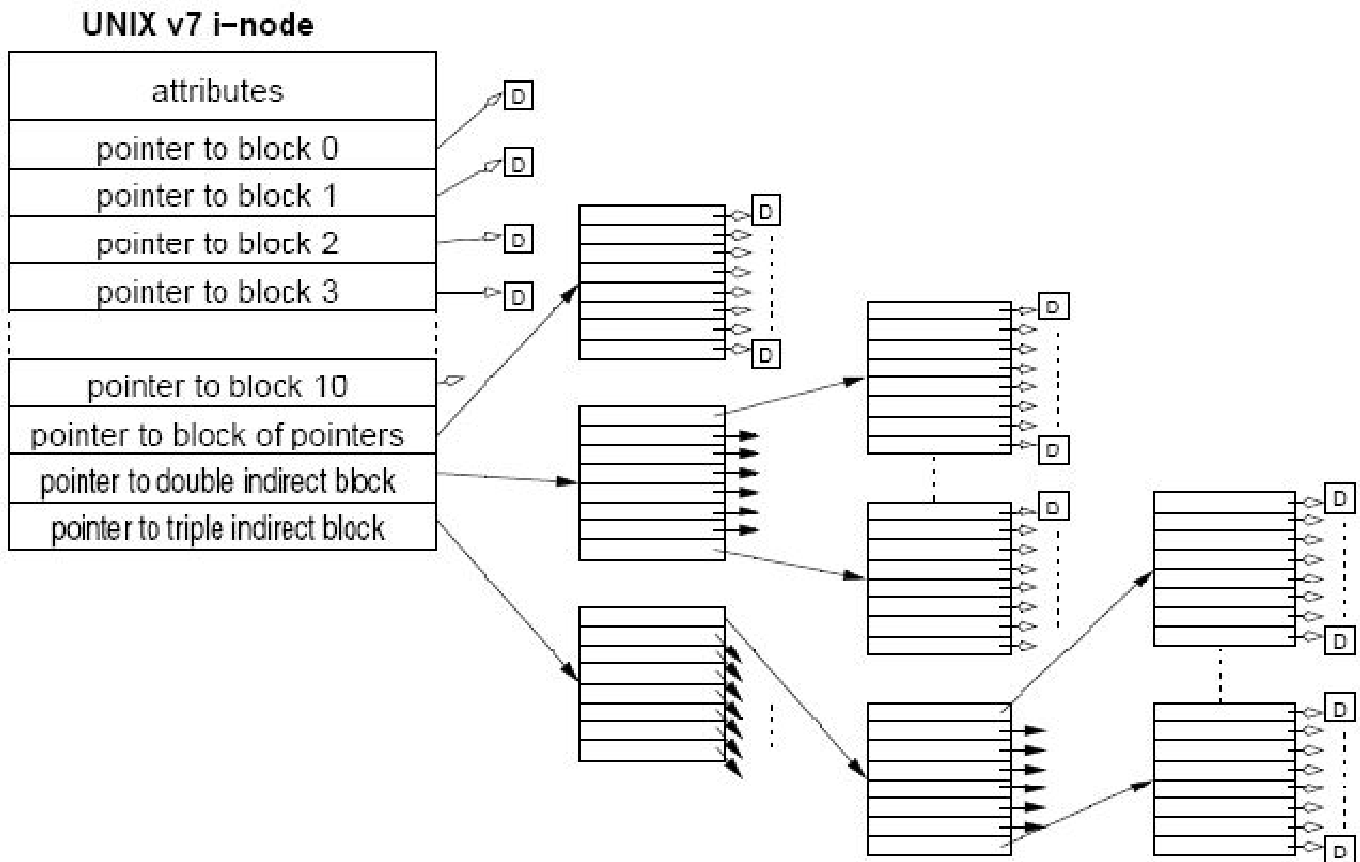


Nevýhodou FAT je velikost tabulky (80 GB disk, bloky 4 kB => 20 mil položek, každá položka alespoň 3 byty => 60 MB FAT)

- FAT 12 - 12 bitů, 2^{12} = 4096 bloků, diskety
- FAT 16 - 16 bitů, 2^{16} = 65536 bloků
- FAT 32 - 2^{28} bloků, blok 4-32KB, cca 8TB

I-uzly

S každým souborem sdružená datová struktura i-uzel. I-uzel obsahuje atributy souboru, diskové adresy prvních N bloků a jeden nebo více odkazů na diskové bloky obsahující další diskové adresy. Tradiční FS pro Linux a UNIX



Výhoda: Po otevření souboru můžeme zavést i-uzel a případný blok obsahující další adresy do paměti => urychlý přístup k souboru

Implementace adresářů: tabulka obsahující jméno souboru a číslo jeho i-uzlu.

Info o volných blocích = seznam, jeho začátek je v superbloku na začátku partition

Adresáře - většinou se jedná o speciální typ souboru

Info o volných blocích v paměti - bitová mapa

17

Kontrola konzistence souborového systému, mechanismy ochrany před neoprávněným přístupem

Obsah

- 1 Kontrola konzistence informace o diskových blocích souborů
- 2 Kontrola konzistence adresářové struktury
- 3 Žurnálování
- 4 Mechanismy ochrany

Kontrola konzistence informace o diskových blocích souborů

Program vytvoří 2 tabulky, obsahující čítač pro každý blok

- Tabulka počtu výskytů bloku v souboru
- Tabulka počtu výskytů bloku v seznamu volných bloků (seznam nebo bitová mapa)

Je-li FS konzistentní, bude mít každý blok 1 buď v první, nebo ve druhé tabulce

Číslo bloku	0	1	2	3	4	5	6	7	8
Výskyt v souborech	1	0	1	0	1	0	2	0	1
Volné bloky	0	1	0	0	1	2	0	1	0

Možné chyby: 0-0: blok se nevyskytuje v žádné tabulce, není zavažné pouze redukuje kapacitu - vložení do seznamu volných bloků 0-2: blok je dvakrát nebo vícekrát v seznamu volných (!) - smazání jednoho záznamu ze seznamu volných bloků 1-1: blok patří souboru a zároveň je na seznamu volných (!) - vyjmeme ze seznamu volných bloků 2-0: blok patří do dvou nebo více souborů, nejzávažnější problém, nejspíše už došlo ke ztrátě dat (!!!) - alokujeme nový blok, problémový soubor do něj umístíme a upravíme i-uzel druhého souboru. Informujeme uživatele o problému.

Kontrola konzistence adresářové struktury

Projdeme celý adresářový strom, kontrolujeme zda odpovídá počet odkazů v i-uzlu (i) s počtem výskytů v adresářích (a)

- i = a - vše je OK
- i > a - soubor by nebyl zrušen ani po zrušení všech odkazů v adresářích
- i < a - soubor by byl zrušen po zrušení i odkazů, ale v adresářích budou ještě jména
- a = 0 , i > 0 - ztracený soubor, na který není v adresáři odkaz

Žurnálování

Před každým zápisem na disk vytvoří na disku záznam popisující plánované operace, pak provede operace a záznam zruší. Výpadek – na disku najdeme žurnál o všech operacích, které mohly být v době havárie rozpracované, zjednodušuje kontrolu konzistence fs.

Mechanismy ochrany

- Soubor je třeba chránit před neoprávněným přístupem
- Systém musí uchovávat informace o přístupových právech subjektů k objektům
- Informace může být ve dvou různých podobách: ACL nebo CL

- Subjekt – entita schopná přistupovat k objektům (většinou proces)
- Objekt – cokoliv, k čemu je potřeba omezovat přístup pomocí přístupových práv (např. soubor)

Mechanismus ACL (Access Control Lists)

- S každým objektem sdružen seznam subjektů, které mohou k objektu přistupovat
- Pro každý uvedený subjekt je v ACL množina přístupových práv k objektu
- sbjekty se zpravidla sdružují do skupin (učitelé, studenti, admini...)
- ntfs

Mechanismus Capability Lists

- S každým subjektem sdružen seznam objektů, ke kterým může přistupovat a jakým způsobem
- Problém: zjištění, kdo všechno má k objektu přístup + zrušení přístupu velmi obtížné
- Řešení: odkaz neukazuje na objekt, ale na nepřímý objekt; systém může zrušit nepřímý objekt, tím neplatní odkazy na objekt ze všech C-seznamů.
- někter=é distribuované systémy

Mechanismus ACL se používá častěji.