Základy operačních systémů

1 Rozdělení OS, architektura a komponenty OS, základní funkce OS

Operační systém je v informatice základní programové vybavení počítače (tj. software), které je zavedeno do paměti počítače při jeho startu a zůstává v činnosti až do jeho vypnutí. Skládá se z jádra (kernel) a pomocných systémových nástrojů. Hlavním úkolem operačního systému je zajistit uživateli možnost ovládat počítač, vytvořit pro procesy stabilní aplikační rozhraní (API) a přidělovat jim systémové zdroje. Operační systém je velmi komplexní software, jehož vývoj je mnohem složitější a náročnější, než vývoj obyčejných programů.

1.1 Rozdělení OS

Dle úrovně sdílení CPU

- Jednoprocesový – MS DOS, v daném čase v paměti aktivní 1 program
- Multiprocesový – Efektivnost využití zdrojů – Práce více uživatelů

Dle typu interakce

- Dávkový systém – Sekvenční dávky, není interakce
- Interaktivní – Interakce uživatel – úloha – Víceprocesové – interakce max. do několika sekund (Win, Linux, ..)

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

1.2 Architektura a komponenty OS

Podle architektury operačního systému se typicky rozlišuje mikrokernel (mikrojádro, jádro je velice jednoduché a obsahuje pouze zcela základní funkce, zbytek operačního systému je mimo toto jádro v aplikacích) a makrokernel (monolitické jádro, jádro je rozsáhlé, obsahuje velké množství funkcí pro všechny aspekty činnosti operačního systému včetně například souborového systému). Jakýmsi kompromisem je modulární jádro, které je fakticky makrojádrem (celé běží v privilegovaném režimu) ovšem jeho značná část je tvořena takzvanými moduly, které je možné přidávat a odebírat za běhu systému.

1.2.1 Komponenty OS

- procesy
- správa hlavní paměti
- soubory
- I/O subsystém
- síť (networking)
- ochrana a bezpečnost
- uživatelské rozhraní

1.3 Funkce

- Operační systém plní tři základní funkce:
- ovládání počítače – umožňuje uživateli spouštět programy, předávat jim vstupy a získávat jejich výstupy s výsledky
- abstrakce hardware – vytváří rozhraní pro programy, které abstrahuje ovládání hardware a dalších funkcí do snadno použitelných funkcí (API)
- správa prostředků – přiděluje a odebírá procesům systémové prostředky počítače

1.3.1 Ovládání počítače

Při definici operačního systému se obvykle omezuje ovládání počítače na schopnost spustit program, předat mu vstupní data a umožnit výstup výsledků na výstupní zařízení. Někdy je však pojem operační systém rozšířen i na grafické uživatelské rozhraní, což může být z důvodů marketingových, ale i problému nejasné hranice mezi operačním systémem a aplikacemi.

U systémů, které disponují jediným grafickým rozhraním (Microsoft Windows, Symbian OS, ...) je často grafické rozhraní zahrnováno do operačního systému. U systémů, kde je uživatelské rozhraní možné vytvořit několika nezávislými způsoby nebo různými aplikacemi, je běžné nepovažovat ho za součást systému (unixové systémy).

1.3.2 Abstrakce hardware

Operační systém skrývá detaily ovládání jednotlivých zařízení v počítači (tzv. hardware) a definuje standardní rozhraní pro volání systémových služeb tak, že vytváří abstraktní vrstvu s jednoduchými funkcemi (tzv. API), které využívají programátoři aplikací. Tím nejen zjednodušuje programátorům vytváření programů, ale umožňuje programům pracovat i se zařízeními, které v době vzniku programu neexistovaly (například z hlediska programátora není rozdíl mezi otevřením souboru na pevném disku, CD, DVD, flash, síťovém disku nebo Blu-ray). Někdy je uvnitř operačního systému vytvářena podobná abstraktní mezivrstva, která usnadňuje programování ovladačů jednotlivých zařízení (tzv. HAL, anglicky Hardware Abstraction Layer).

1.3.3 Správa zdrojů

Operační systém přiděluje spuštěným programům systémové prostředky (operační paměť, procesor, pevný disk, vstupně-výstupní zařízení). V případě potřeby může operační systém procesům přidělené prostředky násilně odebrat (preempce). Operační systém využívá schopnosti procesoru k ochraně sebe samého, ale i k oddělení pracovního prostoru jednotlivých procesů.

2 Vyvolání služeb OS, zpracování přerušení

Systém nastaví časovač – pravidelně přerušení

Na předem definovaném místě – adresa obslužného programu přerušení

CPU po příchodu přerušení provede:

- Uloží čítač instrukcí PC do zásobníku
- Načte do PC adresu obsluž. programu přerušení
- Přepne do režimu jádra
- Vyvolána obsluha přerušení
  - Uloží obsah registrů do zásobníku
  - Nastaví nový zásobník
- Plánovač nastaví proces jako ready, vybere nový proces pro spuštění
- Přepnutí kontextu
  - Nastaví mapu paměti nového procesu

  - Nastaví zásobník, načte obsah registrů
  - Provede návrat z přerušení – RET (do PC adresa ze zásobníku, přepne do uživatelského režimu)

3 Proces, implementace procesu, konstrukce pro vytváření procesů

3.1 Proces

- instance běžícího programu
- Adresní prostor procesu
  - MMU zajišťuje soukromí
  - kód spustitelného programu, data, zásobník
- S procesem sdruženy registry a další info potřebné k běhu procesu = stavové informace
  - registry – čítač instrukcí PC, ukazatel zásobníku SP, univerzální registry

3.2 Implementace procesu

- Vytvoření nového procesu
  - fork v UNIXu, CreateProcess ve Win32
- Ukončení procesu
  - exit v UNIXu, ExitProcess ve Win32
- Čekání na dokončení potomka
  - wait (waitpid) v UNIXu, WaitForSingleObject ve Win32
- Alokace a uvolnění paměti procesu
- Komunikace mezi procesy (IPC)
- Identifikace ve víceuživatel. systémech
  - identifikátor uživatele (UID)
  - skupina uživatele (GID)
  - proces běží s UID toho, kdo ho spustil
  - v UNIXu – UID, GID – celá čísla
- Problém uvíznutí procesu

4 Paralelní procesy, prostředky pro popis paralelních procesů, vlákna

- Běžící SW – organizován jako množina sekvenčních procesů
- Proces – běžící program včetně obsahu čítače instrukcí, registrů, proměnných; běží ve vlastní paměti
- Koncepčně každý proces – vlastní virtuální CPU
- Reálný procesor – přepíná mezi procesy (multiprogramování)
- Představa množiny procesů běžících (pseudo)paralelně

Základní stavy procesu

- Běžící (running) – skutečně využívá CPU, vykonává instrukce

- Připraven (ready, runnable) – dočasně pozastaven, aby mohl jiný proces pokračovat
- Blokován (blocked, waiting) – neschopný běhu, dokud nenastane externí událost

Vlákna

- Vlákna v procesu sdílejí adresní prostor, otevřené soubory (atributy procesu)
- Vlákna mají soukromý čítač instrukcí, obsah registrů, soukromý zásobník
- Mohou mít soukromé lokální proměnné
- Původně využívána zejména pro VT výpočty na multiprocesorech (každé vlákno vlastní CPU, společná data)

5 Problém kritické sekce

- Sekvenční procesy komunikace přes společnou datovou oblast
- kritická sekce (critical section, region) místo v programu, kde je prováděn přístup ke společným datům
- úloha – jak implementovat, aby byl v kritické sekci v daný okamžik pouze 1 proces

Řešení časového souběhu - pravidla

- Vzájemné vyloučení - žádné dva procesy nesmějí být současně uvnitř své kritické sekce
- Proces běžící mimo kritickou sekci nesmí blokovat jiné procesy (např. jim bránit ve vstupu do kritické sekce)
- Žádný proces nesmí na vstup do své kritické sekce čekat nekonečně dlouho (jiný vstupuje opakovaně, neumí se dohodnout v konečném čase, kdo vstoupí první)

Možnosti řešení

- Zákaz přerušení
- Aktivní čekání
- Zablokování procesu

6 Prosředky pro synchronizaci procesů

6.1 Aktivní čekání

- Průběžné testování proměnné ve smyčce, dokud nenabude očekávanou hodnotu
- Většinou se snažíme vyhnout plýtvá časem CPU
- Používá se, pokud předpokládáme krátké čekání spin lock

6.2 Spin lock s instrukcí TSL

- hw podpora
- většina počítačů – instrukci, která otestuje hodnotu a nastaví paměťové místo v jedné nedělitelné operaci
- operace Test and Set Lock – TSL, TS:
- TSL R, lock
  - R je registr CPU
  - lock – buňka paměti, 0 false nebo 1 true; boolean; provádí:
  - LD R, lock
  - LD lock, 1
- Problém inverze priorit, proces s nižší prioritou blokuje procesu s vyšší vstup

7 Semafory, jejich použití a implementace

Semafor je synchronizační primitivum obsahující celočíselný čítač, který si lze představit například jako počítadlo volných prostředků. Poskytuje atomické operace „up“ a „down“. Operace „down“ sníží čítač o jedničku, v případě, že už je nulový (nedostává se prostředků), se proces zablokuje a přidá do fronty procesů čekajících na daný semafor. Operace „up“ zkontroluje frontu, a v případě, že je neprázdná, vybere jeden proces čekající ve frontě a odblokuje jej (ten pak pokračuje za svou operací „down“); je-li fronta prázdná, zvýší hodnotu čítače o jedničku.

Příklad implementace v pseudokódu:

```
P(Semaphore s)
{
čekej dokud není s > 0 pak s = s-1; /* musí být atomické jakmile je zjištěno, že s > 0 */
}
V(Semaphore s)
{
s = s+1; /* musí být atomické */
}
Init(Semaphore s, Integer v)
{
s = v;
}
```

Názvy operací P a V jsou tradiční a pochází od Edsgera Dijkstry, který semafory vymyslel. Jsou odvozeny od holandských slov „prolaag“ (složeno z probeer te verlagen, zkus a sniž) a „verhoog“, zvyš.

8 Monitory

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

Monitor se skládá z dat, ke kterým je potřeba řídit přístup, a množiny funkcí, které nad těmito daty operují.

Příklad v Javě:

```
class Kanál {
private int data;
private boolean naplněn = false;
public synchronized int přijmout() {
while (!naplněn) {
try { wait(); } catch (InterruptedException e) {}
}
naplněn = false;
return data; }

public synchronized void poslat(int hodnota) {
data = hodnota;
naplněn = true;
notify(); }
}
```

9 Problém uvíznutí procesů, graf alokace zdrojů

Uvíznutí - definice

- Obecný termín zdroj – zařízení, záznam, ...

- V množině procesů nastalo uvíznutí, jestliže každý proces množiny čeká na událost, kterou může způsobit jiný proces množiny
- Všichni čekají – nikdo událost nevygeneruje, nevzbudí jiný proces

Jak se vypořádat s uvíznutím

- Problém uvíznutí je zcela ignorován
- Detekce a zotavení
- Dynamické zabránění pomocí pečlivé alokace zdrojů
- Prevence, pomocí strukturální negace jedné z dříve uvedených nutných podmínek pro vznik uvíznutí

Podmínky vzniku uvíznutí

- **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**
  Jednou přiřazené zdroje nemohou být procesu násilně odejmuty (proces je musí sám uvolnit)
- **cyklické čekání**
  Musí být cyklický řetězec 2 nebo více procesů, kde každý z nich čeká na zdroj držený dalším členem

Pro vznik uvíznutí – musejí být splněny všechny 4 podmínky 1. až 3. předpoklady, za nich je definována 4. podmínka. Pokud jedna z podmínek není splněna, uvíznutí nenastane.

Detekce cyklu

Při žádostech o zdroj OS konstruuje **graf alokace zdrojů**. Detekce cyklu pozná, zda nastalo uvíznutí. Různé algoritmy (teorie grafů) Např. prohledávání do hloubky z každého uzlu, dojdeme-li do uzlu, který jsme již prošli - cyklus.

Vyhladovění

Procesy požadují zdroje – pravidlo pro jejich přiřazení. Může se stát, že některý proces zdroj nikdy neobdrží, když nenastalo uvíznutí !

10 Klasické problémy meziprocesové komunikace, producent konzument aj.

Meziprocesová komunikace:

- Předávání zpráv
- Primitiva send, receive
- Mailbox, port
- RPC
- Ekvivalence semaforů, zpráv, ...
- Bariéra, problém večeřících filozofů

Problém sdílené paměti

- Uvedené mechanismy

  - umístění objektu ve sdílené paměti
- Někdy není vhodné
  - Bezpečnost – globální data přístupná kterémukoliv procesu bez ohledu na semafor
- Někdy není možné
  - Procesy běží na různých strojích, komunikují spolu po síti
- Řešení – předávaní zpráv

Předávání zpráv – send, receive

- Zavedeme 2 primitiva
  - send (adresát, zpráva) - odeslání zprávy
  - receive(odesilatel, zpráva) - příjem zprávy
  - Send Zpráva (lib. datový objekt) bude zaslána adresátovi
  - Receive Příjem zprávy od určeného odesilatele Přijatá zpráva se uloží do proměnné „zpráva“

Producent-konzument pomocí předávání zpráv

```
cobegin
while true do { producent }
begin
produkuj záznam;
receive(konzument, m); // čeká na prázdnou položku
m := záznam; // vytvoří zprávu
send(konzument, m); // pošle položku konzumentovi
end {while}
||
for i:=1 to N
do { inicializace }
send(producent, e); // pošleme N prázdných položek
while true do { konzument }
begin
receive(producent, m); // přijme zprávu obsahující data
záznam := m;
send(producent, e); // prázdnou položku pošleme zpět zpracuj záznam;
end {while}
coend
```

11 Plánování úloh a procesů v dávkových systémech

- průchodnost (throughput) – počet úloh dokončených za časovou jednotku
- průměrná doba obrátky (turnaround time) – průměrná doba od zadání úlohy do systému do dokončení úlohy
- využití CPU
- maximalizace průchodnosti nemusí nutně minimalizovat dobu obrátky
- dlouhé úlohy následované krátkými
- upřednostňování krátkých
- dobrá průchodnost
- dlouhé úlohy se nevykonají
  - doba obrátky bude nekonečná

12 Plánování procesů v interaktivních systémech

Základní stavy procesu

- běžící
- připraven -čeká na CPU
- blokován – čeká na zdroj nebo zprávu
- nový (new) – proces byl právě vytvořen
- ukončený (terminated) – proces byl ukončen

Správce procesů – udržuje tabulku procesů. Záznam o konkrétním procesu – PCB (Process Control Block) – souhrn dat potřebných k řízení procesů.

plánovač vs. dispatcher

- dispatcher předává řízení procesu vybranému short time plánovačem
  - přepnutí kontextu
  - přepnutí do user modu
  - skok na vhodnou instrukci daného programu
- více připravených procesů k běhu – plánovač vybere, který spustí jako první
- plánovač procesů (scheduler) - používá plánovací algoritmus (scheduling algorithm)
- **Preemptivní vs. non-preemptive plánování**

13 Správa hlavní paměti, metody přidělování paměti, virtuální paměť

- Ideál programátora
  - Paměť nekonečně velká, rychlá, levná
  - Zároveň persistentní (uchovává obsah po vypnutí)
  - Bohužel neexistuje
- Reálný počítač
  - hierarchie pamětí („pyramida“)
  - Registry CPU
  - Malé množství rychlé cache paměti
  - Stovky MB až gigabajty RAM paměti
  - GB na pomalých, levných, persistentních discích
- **Jednoprogramové systémy**
  - Spouštíme pouze jeden program v jednom čase
  - Uživatel – příkaz , OS zavede program do paměti
  - Dovoluje použít veškerou paměť, kterou nepotřebuje OS
  - Po skončení procesu lze spustit další proces
- **Tři varianty rozdělení paměti**
  - OS ve spodní části adresního prostoru v RAM (minipočítače)
  - OS v horní části adresního prostoru v ROM (zapouzdřené 2. systémy)

  - OS v RAM, ovladače v ROM (PC – MS DOS v RAM, BIOS v 3. ROM)

- **Modul pro správu paměti**
  - informace o přidělení paměti
  - která část je volná přidělená (a kterému procesu)
- přidělování paměti na žádost
- uvolnění paměti, zařazení k volné paměti
- odebírá paměť procesům
- ochrana paměti
  - přístup k paměti jiného procesu
  - přístup k paměti OS
- **Funkce MMU**
- Dostává adresu od CPU, převádí na adresu do fyzické paměti
- Nejprve zkontroluje, zda adresa není větší než limit Ano – výjimka, Ne – k adrese přičte bázi
- Pokud báze 1000, limit 60
  - Adresa 55 – ok, výsledek 1055
  - Adresa 66 – není ok, výjimka
- **Virtuální paměť**
  - program větší než dostupná fyzická paměť
  - mechanismus překrývání (overlays)

Virtuální adresy

fyzická paměť slouží jako cache virtuálního adresního prostoru procesů. Procesor používá virtuální adresy.P ožadovaná část VA prostoru JE ve fyzické paměti -> MMU převede VA=>FA, přístup k paměti. Požadovaná část NENÍ ve fyzické paměti -> OS ji musí přečíst z disku I/O operace – přidělení CPU jinému procesu většina systémů virtuální paměti používá stránkování.

14 Algoritmy nahrazování stránek v paměti

Algoritmus MIN / OPT

- optimální – nejmenší možný výpadek stránek
- Vyhodíme zboží, které nejdelší dobu nikdo nebude požadovat.
- stránka označena počtem instrukcí, po který se k ní nebude přistupovat
- p[0] = 5, p[1] = 20, p[3] = 100
- výpadek stránky – vybere s nejvyšším označením
- vybere se stránka, která bude zapotřebí nejpozději v budoucnosti
- není realizovatelný (křišťálová kole) jak zjistit dopředu která stránka bude potřeba?
- algoritmus pro srovnání s realizovatelnými běh programu v simulátoru uchovávají se odkazy na stránky spočte se počet výpadků pro MIN/OPT srovnání

Least Recently Used (LRU, LUR)

- nejdéle nepoužitá (pohled do minulosti)
- princip lokality
  - stránky používané v posledních instrukcích se budou pravděpodobně používat i v následujících
  - pokud se stránka dlouho nepoužívala, pravděpodobně nebude brzy zapotřebí
- Vyhazovat zboží, na kterém je v prodejně nejvíce prachu = nejdéle nebylo požadováno
- Obtížná implementace
- sw řešení (není použitelné)
  - seznam stránek v pořadí referencí
  - výpadek – vyhození stránky ze začátku seznamu
  - zpomalení cca 10x, nutná podpora hw
- **hw řešení – čítač**
- MMU obsahuje čítač (64bit), při každém přístupu do paměti zvětšen
- každá položka v tabulce stránek – pole pro uložení čítače
- odkaz do paměti
  - obsah čítače se zapíše do položky pro odkazovanou stránku
- výpadek stránky
  - vyhodí se stránka s nejnižším číslem

Not-Recently-Used (NRU, NUR)

- snaha vyhazovat nepoužívané stránky
- HW podpora u systémů s VM
  - stavové bity Referenced (R) a Dirty (M = modified)
  - v tabulce stránek
- bity nastavované HW dle způsobu přístupu ke stránce
- bit R – nastaven na 1 při čtení nebo zápisu do stránky
- bit M – na 1 při zápisu do stránky stránku je třeba při vyhození zapsat na disk
- bit zůstane na 1, dokud ho SW nenastaví zpět na 0
- NRU předpokládá – lepší je vyhodit modifikovanou stránku, která nebyla použita 1 tik, než nemodifikovanou stránku, která se právě používá
- **výhody** jednoduchost, srozumitelnost efektivně implementovaný
- **nevýhody** výkonnost (jsou i lepší algoritmy)

Algoritmy Second Chance a Clock

- vycházejí z FIFO
- **FIFO** – obchod vyhazuje zboží zavedené před nejdelší dobou, ať už ho někdo chce nebo ne
- **Second Chance**
  - evidovat, jestli zboží v poslední době někdo koupil (ano – prohlásíme za čerstvé zboží) modifikace FIFO – zabránit vyhození často používané
  - dle bitu R nejstarší stránky R = 0 ... stránka je nejstarší, nepoužívaná – vyhodíme R = 1 ... nastavíme R=0, přesuneme na konec seznamu stránek (jako by byla nově zavedena)

Algoritmus Clock

- Optimalizace datových struktur algoritmu Second Chance
- Stránky udržovány v kruhovém seznamu
- Ukazatel na nejstarší stránku – „ručička hodin“
- Výpadek stránky – najít stránku k vyhození
- Stránka kam ukazuje ručička
  - má-li R=0, stránku vyhodíme a ručičku posuneme o jednu pozici
  - má-li R=1, nastavíme R na 0, ručičku posuneme o 1 pozici, opakování,..
- Od SC se liší pouze implementací
- Varianty Clock používají např. BSD UNIX

15 Ovládání periferních zařízení, RAID

- **RAID 0**
  - není redundantní
  - ztráta 1 disku – ztráta celého pole
  - důvod použití – výkon např. střih videa
- **RAID 1 mirroring**
  - zrcadlení na 2 disky stejných kapacit
  - totožné informace
  - výpadek 1 disku – nevadí
  - jednoduchá implementace – často čistě sw
  - nevýhoda – využijeme jen polovinu kapacity zápis – pomalejší (2x) čtení – rychlejší (řadič - lze střídat požadavky mezi disky)
- **RAID 5**
  - redundantní pole s distribuovanou paritou
  - minimálně 3 disky
  - režie: 1 disk z pole n disků 5 disků 100GB, 400GB pro data
  - výpadek 1 disku nevadí čtení – výkon ok zápis – pomalejší 1 zápis – čtení starých dat, čtení staré parity, výpočet nové parity, zápis nových dat, zápis nové parity
- **RAID 6**

  - RAID 5 + navíc další paritní disk
  - odolné proti výpadku dvou disků
  - rekonstrukce pole při výpadku – trvá dlouho po dobu rekonstrukce není pole chráněno proti výpadku dalšího disku náročná činnost – může se objevit další chyba, řadič disk odpojí a ...

- **RAID 10**
  - kombinace RAID 0 (stripe) a RAID 1 (zrcadlo)
  - min. počet disků 4 režie 100% diskové kapacity navíc
  - nejvyšší výkon v bezpečných typech polích podstatně rychlejší než RAID 5, při zápisu odolnost proti ztrátě až 50% disků x RAID 5

- Principy I/O software
- typicky strukturován do 4 úrovní
  - obsluha přerušení (nejnižší úroveň v OS)
  - ovladač zařízení
  - SW vrstva OS nezávislá na zařízení
  - uživatelský I/O SW 4.

- **Funkce ovladače zařízení**
- ovladači předán příkaz vyšší vrstvou např. zapiš data do bloku n
- nový požadavek zařazen do fronty
  - může ještě obsluhovat předchozí
- ovladač zadá příkazy řadiči (požadavek 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 status – datová struktura pro hlášení chyb další požadavky ve frontě – jeden vybere a spustí

16 Systémy souborů

potřeba aplikací trvale uchovávat data

- hlavní požadavky
  - možnost uložit velké množství dat
  - informace zachována i po ukončení procesu
  - data přístupná více procesům
- společné problémy při přístupu k zařízení
  - alokace prostoru na disku
  - pojmenování dat
  - ochrana dat před neoprávněným přístupem

– zotavení po havárii (výpadek napájení)

Souborový systém (anglicky filesystem) je označení pro způsob organizace informací (ve formě souborů) tak, aby bylo možné je snadno najít a přistupovat k nim. Souborové systémy mohou používat paměťová média jako pevný disk nebo CD, mohou poskytovat přístup k datům uloženým na serveru (síťové souborové systémy, např. NFS, SMB nebo 9P) nebo mohou poskytovat přístup k čistě virtuálním datům (např. procfs v Linuxu). Souborový systém umožňuje ukládat data do souborů, které jsou označeny názvy. Obvykle také umožňuje vytvářet adresáře, pomocí kterých lze soubory organizovat do stromové struktury.

17 Kontrola konzistence souborového systému, mechanismy ochrany před neoprávněným přístupem

OS precte blok souboru, zmenı ho, zapıše.Co kdyz nastane havárie pred tım, nez jsou všechny modifikované bloky zapsány? => fs muze být v nekonzistentnım stavu.S vetšinou OS se dodává systémový program, který kontroluje konzistenci fs (UNIX: fsck, Windows: scandisk resp. chkdsk). Program je automaticky spušten pri startu po havárii systému. Program muze provádet následujıcı testy konzistence fs: - konzistence informace o diskových blocıch souboru (ve vetšine fs muze blok patrit pouze jednomu souboru nebo být volný) - konzistence adresárové struktury (jsou všechny adresáre a soubory dostupné? apod.)

kontrola konzistence adresárové struktury - tabulka cıtacu, jedna polozka pro kazdý soubor - program prochází rekurzivne celý adresárový strom - polozku pro soubor program zvýši pro kazdý výskyt souboru v adresári - pak zkontroluje zda odpovídá pocet odkazu v i-uzlu ("i") s poctem výskytu v adresárıch ("a").

Kontrola konzistence fs muze pro rozsáhlé disky trvat minuty, prıpadne i hodiny. Jako cástecné rešenı tohoto problému vznikly modernı tzv. zurnálujıcı fs (v Linuxu jsou to napr. souborové systémy ext3 (coz je zurnálujıcı rozšırenı ext2), ReiserFS a dalši). Zurnálujıcı fs pred kazdým zápisem na disk vytvorı na disku záznam popisujıcı plánované operace; pak operace provede a záznam zrušı. V prıpade výpadku napájenı lze na disku najıt informaci (zurnál) o všech operacıch které mohly být v dobe havárie rozpracované, coz znacne zjednodušuje kontrolu konzistence fs.

Mechanismy ochrany

- soubory je treba chránit pred neoprávneným prıstupem
- krome souboru existujı v OS dalši objekty, které je treba chránit - muze být HW (segmenty pameti, I/O zarızenı), - SW objekty (procesy, semafory...)
- systém musı uchovávat informace o prıstupových právech subjektu k objektum - subjekt = entita schopná pristupovat k objektum (vetšinou proces) - prıstupové právo je právo vykonat jednu z mozných operacı, naprıklad Read, Write, Execute, Delete... - objekt = cokoli k cemu je treba omezovat prıstup pomocı prıstupových práv
- informace o prıstupových právech muze být ve dvou ruzných podobách: ACL nebo capability list - ACL = s objektem sdruzen seznam subjektu a jejich prıstupových práv - capability list (cti [kejpa-]) = se subjektem je sdruzen seznam objektu a prıstupových práv k nim Mechanismus ACL (Access Control Lists)
- s kazdým objektem sdruzen seznam subjektu které mohou k objektu pristupovat (zde subjekt = uzivatel, resp. jeho procesy), - pro kazdý uvedený subjekt je v ACL mnozina prıstupových práv k objektu
- Mechanismus capability lists (C-seznamy)
- s kazdým subjektem (procesem) sdruzen seznam objektu ke kterým muze pristupovat a jakým zpusobem (tj. prıstupová práva)
- seznam nazýván "capability list" (C-list = C-seznam) a jednotlivé polozky "capabilities"
- struktura "capability" má prvky: - typ objektu - práva - obvykle bitová mapa popisujıcı dovolené operace nad objektem - odkaz na objekt, napr. cıslo i-uzlu, cıslo segmentu apod.
- problém - zjištenı, kdo všechno má k objektu prıstup + zrušenı prıstupu velmi obtızné - znamenalo by to najıt pro objekt všechny capability + odejmout práva

- rešenı - odkaz neukazuje na objekt, ale na neprımý objekt; systém muze zrušit neprımý objekt, tım zneplatnı odkazy na objekt ze všech C-seznamu
- C-seznamy zacnou být zajımavé pokud jsou jediný zpusob odkazu na objekt (capability chápána jako bezpecný ukazatel; název "capability-based addressing) - rušı rozdıl mezi objekty na disku, v pameti (segmenty) nebo na jiném stroji (objekty by se dokonce mohly presouvat za behu) - proto mechanismus C-seznamu najdeme v nekterých distribuovaných systémech (napr. Hydra, Mach apod.)