Základy operačních systémů*

RNDr. Jakub Yaghob

Obsah

1 Historie, architektury **5**
1.1 Historie . . . 5
1.1.1 První generace . . . 5
1.1.2 Druhá generace . . . 5
1.1.3 Třetí generace . . . 5
1.1.4 Čtvrtá generace . . . 5
1.1.5 Čtyřapůltá generace . . . 6
1.2 Architektury . . . 6
1.2.1 Monolitická struktura . . . 6
1.2.2 Virtuální stroje . . . 6
1.2.3 Mikrojádro . . . 6
1.3 Používané architektury . . . 6
1.3.1 Architektura Windows NT . . . 6
1.3.2 Architektura Windows 2000 . . . 7
1.3.3 Architektura Linuxu . . . 7

2 Procesy, plánování **7**
2.1 Procesy, vlákna . . . 7
2.1.1 Proces . . . 7
2.1.2 Vlákno . . . 7
2.1.3 Stavy procesu . . . 8
2.2 Přerušení . . . 8
2.2.1 Obsluha přerušení . . . 8
2.3 Plánování . . . 8
2.3.1 Druhy plánování . . . 9
2.3.2 Cíle plánování . . . 9
2.3.3 Kritéria plánování . . . 9
2.3.4 Priority . . . 9
2.3.5 Plánovací algoritmy . . . 9
2.3.6 Plánování v SMP . . . 10
2.3.7 Real-time plánování . . . 10
2.4 Používané plánování . . . 10
2.4.1 Plánování ve Windows . . . 10
2.4.2 Plánování v Linuxu . . . 11

*sepsal a vypracoval Petr Hošek

3 Meziprocesová komunikace a synchronizace **11**
3.1 Klasické synchronizační problémy . . . 11
3.2 Metody dosažení vzájemného vyloučení . . . 12
3.2.1 Aktivní čekání . . . 12
3.2.2 Pasivní čekání/blokování . . . 13
3.3 Reálně používaná primitiva . . . 13
3.3.1 Synchronizační primitiva vyšší úrovně . . . 14
3.3.2 Synchronizační primitiva ve Windows . . . 14
3.3.3 Synchronizační primitiva v Unixových systémech . . . 14

4 Prostředky, zablokování **14**
4.1 Prostředky . . . 14
4.1.1 Práce s prostředky . . . 15
4.2 Zablokování . . . 15
4.2.1 Coffmanovy podmínky . . . 15
4.2.2 Modelování zablokování . . . 15
4.3 Řešení zablokování . . . 15
4.3.1 Pštrosí algoritmus . . . 15
4.3.2 Detekce a zotavení . . . 16
4.3.3 Vyhýbání se . . . 16
4.3.4 Prevence . . . 16

5 Správa paměti **17**
5.1 Hierarchie paměti . . . 17
5.1.1 Registry . . . 17
5.1.2 Cache . . . 17
5.1.3 Hlavní paměť . . . 17
5.1.4 Pomocná paměť . . . 17
5.1.5 Zálohovací paměť . . . 17
5.2 Správa paměti . . . 17
5.2.1 Přiřazení adresy . . . 18
5.2.2 Překlad adresy . . . 18
5.2.3 Přidělování paměti . . . 18
5.2.4 Fragmentace paměti . . . 20
5.3 Virtuální paměť . . . 20
5.3.1 Stránkování . . . 20
5.3.2 Segmentace . . . 21

6 Souborové systémy **22**
6.1 Soubory . . . 22
6.1.1 Pojmenování souborů . . . 22
6.1.2 Atributy souborů . . . 22
6.1.3 Struktura souborů . . . 22
6.1.4 Typy souborů . . . 22
6.1.5 Druhy přístupu k souborům . . . 22
6.1.6 Paměťově mapované soubory . . . 23
6.1.7 Operace se soubory . . . 23
6.1.8 Adresáře . . . 23
6.1.9 Operace s adresáři . . . 23
6.1.10 Cesty . . . 24
6.1.11 Linky . . . 24
6.1.12 Hierarchická struktura . . . 24
6.2 Implementace souborových systémů . . . 24
6.2.1 Uložení souborů . . . 24

6.2.2 Implementace adresářů . . . 25
6.2.3 Volné místo na disku . . . 25
6.3 Implementace . . . 25
6.3.1 NTFS . . . 25
6.3.2 EXT2/EXT3 . . . 26

7 Vstupně/výstupní subsystémy 26
7.1 Charakteristiky zařízení . . . 26
7.1.1 Druhy přístupu . . . 26
7.1.2 Druhy zařízení . . . 26
7.1.3 Přístup . . . 26
7.1.4 Synchronnost komunikace . . . 27
7.1.5 Sdílení . . . 27
7.1.6 Rychlost . . . 27
7.1.7 Směr dat . . . 27
7.2 Cíle vstupně/výstupních subsystémů . . . 27
7.3 Přenos dat . . . 27
7.4 Vstupně/výstupní software . . . 28
7.4.1 Obsluha přerušení . . . 28
7.4.2 Jednotné rozhraní pro zařízení . . . 28
7.4.3 Vyrovnávací paměti . . . 28
7.5 Zařízení . . . 28
7.5.1 Disky . . . 28
7.6 RAID (Redundant Array of Inexpensive Disks) . . . 29

8 Ochrana a bezpečnost 30
8.1 Ochrana a bezpečnost . . . 30
8.1.1 Domény ochrany . . . 30
8.1.2 Cíle útoků . . . 30
8.1.3 Autentikace . . . 30
8.1.4 Vnitřní útoky . . . 30
8.1.5 Vnější útoky . . . 31
8.2 Kryptografie . . . 31
8.2.1 Kryptografický systém . . . 31
8.2.2 Symetrické kryptografické systémy . . . 32
8.2.3 Asymetrické kryptografické systémy . . . 32
8.2.4 Útoky na protokoly . . . 32

1 Historie, architektury

Co je to operační systém?

- operační systém jako abstrakce pro zjednodušení přístupu a ovládání zařízení
- operační systém jako nástroj pro přidělování a správu prostředků

1.1 Historie

1.1.1 První generace

Operační systém jako takový neexistoval. Počítače se programovaly přímo, neexistovaly žádné standardy.

1.1.2 Druhá generace

Nástup prvního programovacího jazyka *Fortran*. Operační systém stále neexistoval ale objevovaly se první myšlenky.

1.1.3 Třetí generace

První počítač pro všeobecné použití *IBM 360*. Tento počítač bylo možné jako první přizpůsobit různému způsobu použití. Obsahoval také první operační systém *OS 360*. Tento operační systém měl otevřené zdrojové kódy a tyto se upravovaly způsobu použití daného počítače. Obsahoval také několik v té době převratných vlastností.

multitasking prostředky počítače může využívat "paralelně" více procesů

spooling umožňuje každému procesu používat zařízení bez ohledu na ostatní a zajišťuje správné rozvržení přístupu k zařízení

interaktivní uživatelé umožňuje více uživatelům paralelně používat operační systém

virtuální paměť zvětšení velikosti operační paměti

sítě podpora prvních sítí

real-time umožňuje vykonání procesu v zadaný čas, samotné plánování úloh je *NP-uplná* úloha

1.1.4 Čtvrtá generace

Zmenšuje se objem počítačů. Síťové operační systémy umožňují využívat síťové zdroje. Distribuované operační systémy využívají transparentně síťové zdroje.

1.1.5 Čtyřapůltá generace

Operační systémy pro jednoduchá zařízení, většinou jednodušší než klasické systémy uzpůsobené konkrétnímu použití.

1.2 Architektury

Základní pojmy

systémová volání volání operačního systému

proces vykonávaný program

soubor persistentní data

1.2.1 Monolitická struktura

Veškeré služby operačního systému běží v chráněném režimu, tedy v privilegované části procesoru. Procesy samotné běží na uživatelské úrovni a s operačním systémem komunikuje pomocí systémových volání. Všechny servisní a pomocné funkce jsou společné a běží v chráněném režimu.

1.2.2 Virtuální stroje

Poprvé se objevily v *OS 360*. Systém byl rozdělen na několik částí. Obsahoval *virtuální stroje* jejichž jediným úkolem byl *multitasking*. Každý *virtuální stroj* obsluhoval jednu *aplikaci* a vlastní *operační systém*.
V dnešní podobě existují abstraktní stroje, které vykonávají instrukce skutečného stroje. Nezáleží na skutečném hardware.
V současnosti se začíná využívat *paravirtualizace*. Operační systémy v tomto případě vědí o tom že jsou *virtualizovány* a koopoerují s *paravirtualizérem*.

1.2.3 Mikrojádro

Mikrojádro obsluhuje pouze základní prostředky, jako jediné běží v chráněném režimu. Zbývající zařízení obsluhují *servery*. Pro komunikaci mezi procesy a servery se používá architektura *klient/server*. Úkolem mikrojádra je zajistit tuto komunikaci.

1.3 Používané architektury

1.3.1 Architektura Windows NT

Velmi malé jádro nad kterým běží nezávislé subsystémy, tyto mohou podporovat různé jiné systémy.

1.3.2 Architektura Windows 2000

Velmi dobře organizované jádro, jedinou vyjímku tvoří grafický subsystém který je kvůli optimalizaci implementován přímo do jádra. Teoreticky nezávislé na hardwaru díky abstrakční vrstvě *HAL*, vyjímku opět tvoří grafický subsytém který přistupuje k hardwaru přímo.

1.3.3 Architektura Linuxu

Klasické monolitické jádro.

2 Procesy, plánování

Základní pojmy

proces vlastní běžící program, obálka sdružující prostředky pro jednotlivé vlákna; těžká entita

vlákno oddělení místa vykonávání instrukcí od adresního prostoru; lehká entita

přerušení přerušení způsobí že procesor přeruší proud vykonávaných instrukcí a začne vykonávat instrukce na jiné adrese obsluhující toto přerušení, po vykonání těchto instrukcí se vrací zpět k vykonávání původního proudu instrukcí

2.1 Procesy, vlákna

2.1.1 Proces

V různých operačních systémech má pojem *proces* mírně odlišný význam. Mají svou hierarchii. Typicky proces spravuje adresový prostor (paměť), prostředky, práva a některé signály.

2.1.2 Vlákno

Vlákno spravuje jediný prostředek, a to procesor. Dříve běželi vlákna pouze na uživatelské úrovni, jádro operačního systému o vláknech nevědělo. Postupně se koncept vláken dostal i do jader operačních systémů a dnes tyto jádra vlákna přímo podporují.
Dříve docházelo k problémům s omezeným počtem vláken na jaderné úrovni. Dnes existují hybridní vlákna, tedy jaderné vlákna které jsou na uživatelské úrovni rozděleny na další vlákna.
Každé vlákno musí mít vlastní zásobník. Typicky má také programový čítač a registry procesoru. Tyto prostředky se nazývají kontext.

2.1.3 Stavy procesu

Obecně stavy nejmenších entit které daný operační systém rozeznává. Pomocí plánovače a dostupných prostředků se snažíme vytvořit iluzi že na jednom procesoru běží více procesů zároveň.
Standardně procesy díky režii plánovače přechází ze stavu *připraven* do stavu *běžící* a zpět jakmile běží moc dlouho, případně dokud neskončí. Pokud přijde systémové volání, je nutné proces *blokovat*, jakmile tato situace vyřešena, přechází proces opět do stavu *připraveno*.

2.2 Přerušení

Druhy přerušení

synchronní speciální instrukce umožňující vstoupit do operačního systému díky speciálnímu druhu přerušení, takto fungují systémová volání; druhým případem jsou vyjímky které generuje procesor a které informují operační systém o nesprávném chování procesoru

asynchroní vznikají díky vnějším událostem

polling umožňuje operačnímu systému pravidelně kontrolovat stav zařízení pomocí přerušení, náhrada pro zařízení které generování přerušení při změně stavu nepodporují

2.2.1 Obsluha přerušení

Procesor obsahuje vektor přerušení který obsahuje informace o jednotlivých druzích přerušení a místa které obsahují instrukce k jejich obsluze. Operační systém typicky obsadí tyto místa vlastními instrukcemi.

Průběh obsluhy přerušení

1. operační systém převezme řízení
2. uloží se stav procesoru, kontexty procesů se uloží do tabulky procesů
3. analyzuje se druh přerušení; více zařízení typicky využívají jedno přerušení, operační systém musí zjistit které zařízení přerušení vyvolalo
4. vyvolá se příslušná obsluha
5. obslouží se přerušení
6. obnoví se stav procesoru; této situace se typicky využívá k přeplánování
7. aplikace pokračuje

2.3 Plánování

Plánování řídí plánovač. V operačním systému existují plánovací entity.

2.3.1 Druhy plánování

preemtivní plánování plánování při kterém je operační systém schopen vynutit si převzetí plánování od plánovací entity a přeplánovat procesy; operační systém má plnou kontrolu nad běžícími procesy

nepreemtivní plánování (kooperativní plánování) procesy předávají řízení dobrovolně, kooperují při řízení práce procesoru mezi sebou a operačním systémem

2.3.2 Cíle plánování

spravedlnost každý proces dostane jednou za čas příležitost využívat procesor

efektivnost snaha o maximální využití procesoru

doba odpovědi dostatečná doba odezvy procesu

průchodnost ukončit v čase maximální počet procesů

minimální režie systému algoritmus pro plánování by měl využívat co nejmíň prostředků

2.3.3 Kritéria plánování

- proces který se váže na procesor tento často využívá
- proces který se váže vstupně výstupní zařízení tyto často využívá
- dávkový a interaktivní proces se liší dobou odezvy
- priorita symbolizuje jak důležitý uvnitř systému proces je
- výpadky stránek
- skutečný čas procesoru

2.3.4 Priority

Statická priorita se nemění v průběhu času a je dána důležitosti, dynamická priorita se mění s časem a její účel je splnění cíle spravedlnosti. Celková priorita je dána součtem statické a dynamické priority. Čím větší má proces prioritu, tím větší je četnost jeho naplánování v čase.

2.3.5 Plánovací algoritmy

FIFO proces běží tak dlouho dokud má procesor, přicházející procesy se řadí do fronty

Round Robin jakmile proces spotřebuje své časové kvantum, je mu pomocí preemce odebrán procesor což je zařízeno pomocí vnějšího přerušení časovače

více FIFO se zpětnou vazbou více front, každá má určené časové kvantum, nejnižší fronta je typu *Round Robin*; při plánování hledám první neprázdnou frontu, pokud proces spotřebuje své časové kvantum, je mu preemtivně odebrán procesor a je posunut do nižší fronty, pokud je proces zablokován, posunu ho po odblokování do vyšší fronty, tím upřednostňuji I/O procesy které jsou často blokovány

2.3.6 Plánování v SMP

Samostatná fronta procesů a procesorů. Pokud je proces i procesor připraven, spustí se tento proces na daném procesoru.
Při *aktivním* čekání plánovač aktivně využívá jeden procesor, toto čekání spotřebovává výkon a energii. Při *pasivním* čekání je využito speciálních instrukcí.
Problém při přeplánování nastává při přesunutí procesu na jiný procesor kdy je nutné provést synchronizaci cache, toto řeší *afinita* procesů k procesoru kdy se plánovač snaží aby daný proces běžel stále na stejném procesoru.

2.3.7 Real-time plánování

Každý úkol má určený realný čas na dokončení. *NP-úplný* problém, řešení pomocí heuristik. Obvykle předimenzování hardwarových prostředků.

2.4 Používané plánování

2.4.1 Plánování ve Windows

Plánování na úrovni vláken. Neexistuje samostatný plánovač, kód pro plánování rozprostřen po celém jádře. Samostatné fronty vláken pro jednotlivé priority.

Priority 32 úrovní rozdělených podle důležitosti. Priority se dynamicky mění v rozsahu $< -2; 2 >$ od základní přidělené.

Časová kvanta Abstraktní časové jednotky. Velikost kvanta se liší podle typu systému. Časové jednotky určuje *HAL* vrstva podle hardware.

Zvýšení priority Úprava priority pouze pro jedno časové kvantum. Zajištění interaktivity systému. Ochrana před nenaplánováním.

Plánovací scénáře Standardní situace které jsou rozeznávány a standardním způsobem řešeny.

Plánování na SMP Každé vlákno má dvě čísla určující ideální procesor a poslední procesor. Tyto čísla využívám k plánování. Systém Windows neobsahuje vyvažování zátěže, toto je řešeno pomocí generování čísel.

2.4.2 Plánování v Linuxu

Obsahuje vyvažování zátěže na procesorech.

Plánování Datová struktura *Runqueue* určující aktivní a neaktivní procesy pro jednotlivé procesory.

Priorita Statická a dynamická priorita podobně jako na systému Windows s větším škálováním.

Vyvažování front na SMP Vyvažuje se, pokud se vyprázdní *Runqueue*, jinak v pravidelných intervalech podle zátěže systému. Hledá se nejvíce zatížený procesor a algoritmus pro vyvažování se mu snaží ulehčit přesouváním procesů na méně zatížené procesory.

3 Meziprocesová komunikace a synchronizace

Základní pojmy

race conditions výsledek operace závisí na plánování

vzájemná vyloučení (mutual exclusion) kritickou sekci provádí nejvýše jeden proces

kritická sekce část programu, kde se provádí kritická operace

Potřebuji zajistit atomicitu procesů, to znamená že kritickou operaci provádí v jednom okamžiku pouze jeden proces.

Podmínky

1. žádné 2 procesy nemohou být najednou v jejich kritické sekci
2. nemohou být učiněny žádné předpoklady o rychlosti nebo počtu procesorů
3. žádný proces mimo kritickou sekci nesmí blokovat jiný proces
4. žádný proces nesmí čekat nekonečně dlouho v jeho kritické sekci

3.1 Klasické synchronizační problémy

Problém producent-konzument Producent produkuje prostředky, konzument je spotřebovává. Mezi nimi je sklad pevné velikosti. Konzument nemá co prodávat když je sklad prázdný, producent přestane vyrábět, když je sklad plný.

Problém obědvajících filosofů Pět filosofů sedí okolo kulatého stolu. Každý má před sebou talíř špaget a jednu vidličku. Špagety jsou bohužel slizké a je třeba je jíst dvěma vidličkami. Život filosofa sestává z období jídla a období přemýšlení. Když dostane hlad, pokusí se vzít dvě vidličky, když se mu to podaří, nají se a vidličky odloží.

Problém ospalého holiče Holič má ve své oficíně křeslo na zákazníka a pevný počet sedaček pro čekající zákazníky. Pokud v oficíne nikdo není, holič se posadí a spí. Pokud přijde první zákazník a holič spí, probudí se a posadí si zákazníka do křesla. Pokud přijde zákazník a holič už stříhá a je volné místo v čekárně, posadí se, jinak odejde.

3.2 Metody dosažení vzájemného vyloučení

3.2.1 Aktivní čekání

- spotřebovává čas procesor
- vhodnější pro předpokládané krátké doby čekání
- nespotřebovává prostředky operačního systému
- rychlejší

Zakázání přerušení Zakázání přerušení nelze použít v preemtivních operačních systémech, kde je instrukce pro zakázání přerušení procesům nedostupná. Taktéž nelze použít na více procesorech kde procesy na různých procesorech mohou přistupovat ke stejným prostředkům. Používá se v jádrech operačních systémů.

Zámky Nefungují protože samotný zámek je sdíleným prostředkem a problém s přístupem může nastat i na něm.

Důsledné střídání Fungují pokud výpočty v nekritických sekcích jsou vyvážené. V opačném případě porušuje podmínku 3.

Petersonovo řešení Inteligentně kombinuje *zámky* a *důsledné střídání*, tak aby fungovalo ve všech případech. Pro více než 2 procesy se jedná o poměrně komplikované řešení.

Instrukce TSL Vyžaduje podporu hardwaru. Instrukce pro *spin-lock* je atomická ale poměrně složitá, proto se kombinuje s vnější smyčkou aby se zajistilo, že se bude provádět co nejméně. Jedná se o funkční a používané řešení.

3.2.2 Pasivní čekání/blokování

- proces je ve stavu blokován
- vhodné pro delší doby čekání
- spotřebovává prostředky operačního systému
- pomalejší
- používá se také u bariér které slouží pro synchronizaci vláken

Jednotlivé synchronizační primitiva jsou navzájem ekvivalentní a každé z nich lze implementovat pomocí jiného.

Sleep/Wakeup Implementuje operační systém. Atomické operace *Sleep* a *Wakeup*. Nejedná se o funkční řešení! V případě problému *producent-konzument* řešení nebude fungovat, protože čítač je v tomto případě sdílený prostředek v kritické sekci.

Semafory Vylepšení metody *Sleep/Wakeup* zajišťující atomicitu operací. Struktura obsahující čítač a frontu uspaných procesů, atomické operace *Up* a *Down* pracující s touto strukturou. Pokud je čítač nezáporný dojde při operaci *Down* ke snížení čítače, v opačném případě dojde k zablokování a proces je přidán do frony čekajících. Pokud je fronta neprázdná, probudí operace *Up* libovolný proces z fronty, jinak zvětší čítač. *Binární semafor* se používá na vynucení vzájemného vyloučení. Díky atomicitě operací bude toto řešení fungovat.

Monitory Synchronizační primitivum implementované překladačem. Vzájemné vyloučení v jedné instanci zajištěné synchronizací na vstupu a výstupu veřejných funkcí. Překladač zajistí synchronizaci pomocí některého z blokovacích primitiv poskytovaných operačním systémem. Uvnitř monitoru existují proměnné sloužící k zablokování uvnitř monitoru. Pro práci s těmito proměnnými slouží operace *Wait* a *Signal*. Může nastat problém při probouzení procesu. Jedná se o funkční teoretické řešení, ale v praxi neexistuje.

Zprávy *Zpráva* je množina dat a informací přenášených mezi odesilatelem a příjemcem. Atomické operace *Send* a *Receive*. Při odeslání zprávy operací *Send* nedochází k zablokovaní, pokud na zprávu čeká příjemce operací *Receive*, odblokuje se. Pokud není žádná zpráva dostupná, operace *Receive* zablokuje proces. Princip klient/server. Adresace pomocí identifikace procesu nebo schránky. *Randezvous* je speciální situace kdy mám schránku nulové velikosti a synchronizuji dva procesy v čase.

3.3 Reálně používaná primitiva

- *spin-lock*, *kritické sekce*, *semafor*
- závislé na operačním systému

3.3.1 Synchronizační primitiva vyšší úrovně

Read-Write Lock vícenásobně/paralelně operace *Read*, exkluzivně *Write* operace

Bariéry hlavní vlákno vypustí pomocná vlákna a čeká na bariéře na dokončení úkolu všech vláken

Reentrantní zámky několik funkcí které používají stejnou kritickou sekci, některé z těchto funkcí volají jiné z těchto funkcí, zdánlivě zjednodušuje použití, reálně přináší spíše potíže, to lze odstranit vyrobením blokovacích a neblokovacích variant funkcí

Uzamčené operace atomicky prováděné jednoduché operace

3.3.2 Synchronizační primitiva ve Windows

- jednotné funkce pro pasivní čekání
- *semafor*, *event*, *kritické sekce*, *jednosměrný spojový seznam*, *uzamčené operace*

3.3.3 Synchronizační primitiva v Unixových systémech

- operační systém implementuje *semafor*
- dnes obvykle knihovna *pthread*, ta implementuje *podmíněné proměnné*, *mutexy*, *Read-Write Lock*, *spin-lock*, nemusí být vždy dostupná

4 Prostředky, zablokování

4.1 Prostředky

Prostředek je cokoliv, k čemu je třeba hlídat přístup.

- operační systém jako správce prostředků
- hardwarové zařízení (procesor, tiskárna, ...)
- informace (záznam v databázi, ...)

Druhy prostředků

odnímatelné lze odejmout bez následků

neodnímatelné nelze odejmout bez nebezpečí selhání výpočtu

4.1.1 Práce s prostředky

1. žádost o prostředek
   - blokující
   - právě zde dochází k zablokování
2. používání prostředku
3. odevzdání prostředku
   - dobrovolné
   - při skončení procesu

4.2 Zablokování

Množina procesů je zablokována, jestliže každý proces z této množiny čeká na událost, kterou může způsobit pouze jiný proces z této množiny.

4.2.1 Coffmanovy podmínky

vzájemné vyloučení každý prostředek je buď vlastněn právě jedním procesem nebo je volný

drž a čekej procesy aktuálně vlastnící nějaké prostředky mohou žádat o další

neodnímatelnost přidělené prostředky nemohou být procesům odebrány

čekání do kruhu existuje kruhový řetěz procesů, kde každý z nich čeká na prostředek vlastněný dalším článkem řetězu

4.2.2 Modelování zablokování

Lze modelovat jako orientovaný graf se dvěma typy uzlů - prostředky a procesy.

4.3 Řešení zablokování

- pštrosí algoritmus
- detekce a zotavení
- vyhýbání se
- předcházení (prevence)

4.3.1 Pštrosí algoritmus

Zablokování se ani nedetekuje, ani se mu nezabraňuje, ani se neodstraňuje. Uživatel sám rozhodne o řešení. Nespotřebovává prostředky operačního systému, používané na systémech Windows, Unix.

4.3.2 Detekce a zotavení

Operační systém modeluje zablokování orientovaným grafem, hledá v něm kružnici, pokud ji nalezne, došlo k zablokování a je třeba ho řešit.

odebrání prostředku pod dohledem operátora, pouze na přechodnou dobu

zabíjení procesů z cyklu, mimo cyklus vlastnící identický prostředek

rollback operační systém ukládá stav procesů, při zablokování se některé procesy vrátí do předchozího stavu, dojde ke ztracení práce

4.3.3 Vyhýbání se

Spotřebovává prostředky operačního systému. Procesy musí dopředu hlásit jaké prostředky budou požadovat.

bezpečný stav stav procesů a prostředků, nejsou zablokovány a existuje cesta jak uspokojit všechny požadavky na prostředky

nebezpečný stav nesplňuje podmínky bezpečného stavu

Bankéřův algoritmus Bankéř má klienty a těm slíbil jistou výšku úvěru. Bankéř ví, že ne všichni klienti potřebují plnou výši úvěru najednou. Klienti občas navštíví banku a žádají postupně o prostředky do maximální výše úvěru. Až klient skončí s obchodem, vrátí bance vypůjčené peníze. Bankéř peníze půjčí pouze tehdy, zůstane-li banka v bezpečném stavu.

4.3.4 Prevence

vzájemné vyloučení spooling

drž a čekej žádost o všechny prostředky před startem procesu; nejprve je potřeba vše uvolnit a poté žádat o vše najednou

neodnímatelnost vede k chaosu

čekání do kruhu nejvýše jeden prostředek; všechny prostředky jednoznačně očíslovány, procesy mohou žádat o prostředky ve vzestupném pořadí, musím vrátit prostředky, pokud chci prostředek s nižším číslem

Dvoufázové zamykání Nejdříve se vše zamyká, pokud je něco zamčeno, je třeba vše uvolnit a začít znovu. Poté lze může pracovat se zamčenými prostředky. Na závěr pouze odemyká, nesmí se nic zamykat.

5 Správa paměti

5.1 Hierarchie paměti

5.1.1 Registry

Extremní rychlost ale malá kapacita v řádu desítek až stovek bajtů. Klíčové pro překladače.

5.1.2 Cache

Slouží jako nárazník pro vyrovnání rozdílů rychlostí mezi registry a hlavní pamětí. Dělí se do více úrovní podle rychlosti a velikosti v řádu desítek kilobajtů až jednotek megabajtů podle úrovně.
V současných procesorech velmi efektivně využívány, *cache-hit* se pohybuje okolo 95-97%, *cache-miss* se pohybuje okolo 3-5%. Obrazy paměti se v cache drží po řádcích označované jako *cache-line*, její velikost je v řádu bajtů.
Na vícejaderných procesorech se využívá *koherence* cache paměti, cache paměť by měla být co nejvíce transparentní, pokud je nutné neustále synchronizovat cache paměť mezi jednotlivými jádry dochází k jevu nazývanému *cache-trashing*.

5.1.3 Hlavní paměť

Poslední úroveň paměti kterou je procesor přímo schopen adresovat. Velikost v řádu gigabajtů. Několik úrovní rychlosti podle typu přístupu.

5.1.4 Pomocná paměť

Pomocná paměť hraje významnou úlohu ve spojitosti s *virtuální pamětí*. Slouží k uchování *persistentních dat*, tedy dat které zůstávají v paměti i po vypnutí počítače. Důležitým údajem u tohoto typu paměti je *seek*, tedy doba nutná k adresování bloku paměti.

5.1.5 Zálohovací paměť

Zařízení jejichž úlohou bylo archivovat data. Dnes je ale kapacita disků tak vysoká, že neexistuje vhodná zálohovací technologie. Přitom jejich cena neustále klesá. Používaným řešením je tedy v současnosti vytvoření diskových polí s vysokou mírou *redundance dat*.
Tyto zařízení používají typicky sekvenční přístup.

5.2 Správa paměti

Důležitá součást operačního systému jejíž součástí je správa paměťové hierarchie.

Úkoly memory managementu

- informace o volné/plné části paměti, její rozdělení
- přidělování paměti
- výměna paměti s diskem

5.2.1 Přiřazení adresy

při překladu je již známo umístění procesu, generuje se absolutní kód; používá se pro systémové knihovny

při zavádění operační systém rozhodne o umístění, generuje se kód s relokacemi; používá se pro knihovny

za běhu proces se může stěhovat i za běhu, používá se *relokační registr*

Overlay Proces potřebuje více paměti než je skutečně k dispozici. Vyžaduje spolupráci programátora, ten rozdělí program na nezávislé části a část nezbytnou pro všechny části, ta je neustále v paměti, ostatní části jsou v oblasti překrývání a jejich paměťový prostor se navzájem překrývá.

Výměna (swapping) Proces musí být v hlavní paměti aby jeho instrukce mohly být vykonávány procesorem.

5.2.2 Překlad adresy

Proces pracuje v logickém (virtuálním) adresovém prostoru. Hardware pracuje s fyzickým adresovým prostorem. Mezi těmito adresovými prostory je potřeba provádět *překlad adresy*, ten zajišťuje hardware.

5.2.3 Přidělování paměti

Spojité přidělování Část přidělené paměti je fyzicky spojitá.

spojité přidělování s jedním pamětovým oddílem jednoduchý překlad adres, tato technika byla často používána při *swappingu*

spojité přidělování s více paměťovými oddíly pamětový prostor je rozdělen do několika pevných oddílů podle velikosti, procesy jsou přiřazovány do jednotlivých front podle paměťových požadavků

spojité přidělování s více volnými pamětovými oddíly dnes používaný způsob přidělování paměti, často při správě *haldy*; bloky jsou alokovány a přidělovány podle potřeby, při přidělování a uvolňování paměti dochází k jevu nazývanému *interní fragmentace*

Uložení informací o volném místě

bitmapa bloky pevné velikosti

- s velikostí bloku roste *interní fragmentace* ale klesá režie nutná ke správě paměti
- některé procesory obsahují specialní instrukce pro práci s bitmapou

spojový seznam seznam volných bloků

- se snižující se velikostí bloků roste délka seznamu a zvyšuje se režie nutná ke správě seznamu
- při uvolňování je nutné spojit uvolněné bloky

Algoritmy pro přidělování paměti

first-fit použiji první volný blok dostatečné velikosti

- rychlý
- občas rozdělí velký blok

next-fit použiji první volný blok dostatečené velikosti, hledání začíná na poslední prohledávané pozici

- ještě rychlejší než *first-fit*
- rovnoměrnější rozdělení paměti
- větší tendence k rozdělování velkých bloků

best-fit použiji nejmenší volný blok dostatečné velikosti

- pomalý vzhledem k nutnosti procházení celého seznamu
- nechává velké bloky vcelku
- zanechává velmi malé bloky do kterých se nic nevejde, tento jev se nazývá *externí fragmentace*

worst-fit použiji největší volný blok

- pomalý vzhledem k nutnosti procházení celého seznamu
- rozděluje velké bloky
- teoretický algoritmus vytvořený k zabránění *externí fragmentace*, v praxi nepoužívaný

buddy systém bloky o velikosti 2^n, bloky stejné velikosti v seznamu, zarovnání adres

- při přidělování se velikost bloku zaokrouhluje na nejbližší 2^n
- pokud neexistuje volný blok, rozštěpí se větší bloky na příslušné menší velikosti
- při uvolnění paměti se slučují sousední bloky stejné velikosti
- přidělování paměti lze provádět s konstatní časovou složitostí

5.2.4 Fragmentace paměti

externí fragmentace volný paměťový prostor je rozdělen na malé části, v některých případech může dosáhnout až 50%

Interní fragmentace nevyužití celého přiděleného prostoru

Sesypání Snaha o odstranění externí fragmentace. Lze použít pouze při přiřazení adresy za běhu nebo segmentaci. Velká časová náročnost, z toho důvodu se v praxi nepoužívá.

5.3 Virtuální paměť

Procesy nepracují s fyzickou adresou ale virtuální adresou paměti, tato je hardwarově převáděna na fyzickou adresu. Zobrazení virtuální adresy do fyzické nemusí existovat, v tom případě dochází k vyjímce a je nutné změnit obsah paměti tak aby toto zobrazení existovalo. Obraz virtuální paměti je částečně v hlavní a částečně v pomocné paměti. Použitím virtuální paměti dochází ke zvětšení adresního prostoru. Jednotlivé procesy mají vlastní adresní prostor čímž je zajištěna ochrana paměťového prostoru jednotlivých procesů.

Na 32-bitových procesorech je fyzický adresový prostor větší než virtuální adresový prostor. Každý proces má vlastní adresový prostor.

Na 64-bitových procesorech virtuální adresový prostor větší než fyzický adresový prostor. Používá se technika nazývaná *Single Address Space* umožňující vytvořit spojitý adresový prostor velikosti 2^{64} bitů ve kterém běží všechny procesy, ochrana je v tomto případě zajištěna pomocí klíčů.

5.3.1 Stránkování

Virtuální adresový prostor je rozdělen na úseky stené délky nazývané *stránky*. Fyzický adresový prostor je rozdělen na úseky stejné délky jako *stránky* nazývané *rámce*.
Převod adres je zajištěn pomocí stránkovací tabulky. Každý virtuální adresový prostor má vlastní stránkovací tabulku. První část adresy stránky použiji jako index stránkovací tabulky, obsahuje příznak existence mapování; druhá část funguje jako offset. Pokud chci dosáhnout sdílení paměti mezi procesy, sdílím rámce mezi více stránkovacími tabulkami.
Virtuální i fyzický adresní prostor je jednorozměrný. Podporován většinou procesorů a operačních systémů.

Stránkovací tabulky Jsou umístěny v hlavní paměti. Problémem je jejich velikost a pomalost.

víceúrovňové stránkování stránkovací tabulky jsou rozděleny do více úrovní

TLB asociativní paměť, lokalita chování programů

nulaúrovňové stránkování využívá poze *TLB*, ta je řízena i operačním systémem

inverzní stránkování fyzický adresový prostor je menší než virtuální adresní prostor, použití hašovací tabulky o velikosti fyzického adresového prostoru

Algoritmy pro výměnu stránek

Optimal Page teoreticky perfektní algoritmus který se nedá implementovat

Not Recently Used dělí stránky do kategorií podle frekvence přístupu

First In First Out může vykazovat anomálie

Second Chance úprava algoritmu *First In First Out* která nevykazuje anomálie

Clock modifikace algoritmu *Second Chance*

Least Recently Used stránky často používané v předchozím krátkém časovém úseku budou v nejbližších okamžicích znovu použity, lze implementovat v hardwaru

Not Frequently Used softwarová simulace agoritmu *Least Recently Used* která používá softwarový čítač ke každé stránce, modifikací je stárnutí čítače

Random

Alokační strategie Alokace paměti pro jednotlivé procesy.

lokální každý proces má svoji množinu rámců a v ní provádí algoritmy pro výměnu, pracovní množina

globální procesy sdílí všechny rámce

Implementace Je nutné zajistit znovuspuštění instrukce po výpadku stránky. Často se využívá sdílení stránek, při použití *Single Address Space* není nutné přepočítávat adresy. Po zrušení mapování je nutné zrušit položku *TLB*, při použití více procesorů je nutné toto zařídit na všech procesorech.

5.3.2 Segmentace

Program je rozdělen na logické celky se stejnými vlastnostmi nazývané *segmenty*. Ty mohou mít různou velikost a ta se může za běhu měnit. Dvojrozměrný virtuální adresní prostor, jednorozměrný fyzický adresní prostor.
Segment vytváří souvislou oblast ve fyzické paměti, fyzický adresní prostor tedy vypadá stejně jako při spojitém přidělování. Lze implementovat sesypání ve fyzickém adresním prostoru. Umožňuje rozdělit paměť procesu na různé části podle použití.
Při výpadku *segmentu* je nutné vyměnit celý *segment* a ten může být velký. Nelze mít *segment* větší než fyzický adresní prostor.
Lze kombinovat se stránkováním, toto řešení odstraňuje nevýhody segmentace protože se neprovádí výpadky *segmentů*. Dnes implementován pouze na architektuře *Intel IA-32*.

6 Souborové systémy

6.1 Soubory

Soubor je pojmenovaná množina souvisejících informací která leží v pomocné paměti. Jde o abstrakci umožňující uložit informaci na disk a později ji přečíst. Soubory mají jméno, atributy, strukturu, typ. Na souborech jsou definovány druhy přístupu a operace.

6.1.1 Pojmenování souborů

Umožňuje i prostému uživateli přístup k jeho datům. Existují pravidla pro pojmenování souborů které určuje operační systém.

6.1.2 Atributy souborů

Definují vlastnosti a uchovávají informace o souboru. Atributy opět určuje operační systém.

6.1.3 Struktura souborů

Sekvence bajtů, záznamů; strom. Struktura se běžně používá, navenek ale není vidět.

6.1.4 Typy souborů

běžné soubory obsahují informace

adresáře systémové soubory vytvářející strukturu souborového systému

speciální soubory znakové/blokové, soft linky

6.1.5 Druhy přístupu k souborům

sekvenční umožňují pohyb pouze jedním směrem, možné převynutí, umožňují operačnímu souboru přednačítání

přímý umožňuje měnit aktuální pozici

paměťově mapované využití stránkování

6.1.6 Paměťově mapované soubory

Pojmenovaná virtuální paměť. Výhodou je že program přistupuje k souboru instrukcemi pro práci s pamětí, tím se ušetří kopírování paměti.
Problémem je přesná velikost souboru, stránkování má *granularitu* která znemožňuje přesně namapovat soubor do paměti.
Dalším problémem je zvětšování souboru při jejich mapování do virtuálního adresového prostoru. Maximální velikost souboru může být větší než velikost virtuálního adresového prostoru, k řešení tohoto problému se používá *okno* které určí mapovanou část souboru. Při použití *Single Address Space* tento problém odpadá.

6.1.7 Operace se soubory

- vytvoření souboru
- smazání souboru
- otevření souboru
- zavření souboru
- čtení souboru
- zápis souboru
- posun aktuální pozice v souboru

6.1.8 Adresáře

Jde o zvláštní typ souboru který slouží pro udržení organizační struktury souborů, uchovávání atributů souboru. Dnes typicky hierarchický systém který má definovaný kořen.

6.1.9 Operace s adresáři

- hledání souboru
- vypsání adresáře
- přejmenování adresáře
- vytvoření adresáře
- smazání adresáře

6.1.10 Cesty

Pojmenování souboru v hierarchickém uspořádání. Aktuální adresář je vlastnost procesu, jména souborů která nezačínají kořenem se hledají vzhledem k aktuálnímu adresáři.

- absolutní cesta
- relativní cesta

6.1.11 Linky

hard link na jedna data souboru zse odkazuje z ruzných položek v adresářích

soft link speciální soubor, který obsahuje jméno souboru

6.1.12 Hierarchická struktura

strom jednoznačné pojmenování

Direct Acyclic Graph víceznačné pojmenování, neexistují zde cykly

obecný graf víceznačné pojmenování, existují zde cykly, ty mohou být problematické při vyhledávání

6.2 Implementace souborových systémů

Základní pojmy

správa souborů kde jsou umístěna data uložená v souboru

správa adresářů mapování jména na jeho binární identifikaci, uložení atributů

správa volného místa informace o volných blocích

Velikost bloku Disky jsou organizovány po sektorech. Pro zjednodušení správy sektorů se sousedící sektory spojují do bloků, jejich velikost je daná souborovým systémem. Operační systém pracuje s těmito bloky. Soubory se ukládájí na disk po blocích.

velké bloky rychlejší práce s diskem, nebezpečí velké vnitřní fragmentace (průměrná velikost souboru je v řádu jednotek kB)

malé bloky pomalejší práce s diskem, větší režie na udržení informace o volných blocích

6.2.1 Uložení souborů

Souvislá alokace Souvislý sled bloků. Informace o uložení souboru sestává pouze z čísla prvního bloku. Výhodou je lepší práce s diskem. Problém nastává při hledání volného místa, při zvětšování souborů. Z tohoto důvodu se nepoužívá.

Spojovaná alokace Pospojování bloků použitých pro soubor. Pro udržení informace o uložení souboru je potřeba pouze číslo prvního bloku. Nevýhodou je nutnost většího místa pro blok pro uložení informace o následníkovi. Operace posunu je díky nutnosti procházet spojový seznam velmi pomalá. V případě jednosměrného seznamu je problémem posun směrem dozadu. V praxi se používá modifikace která spočívá v přemístění spojového seznamu do speciální oblasti disku. Tato struktura také udržuje informace o volném místě na disku. Tento typ alokace používají všechny souborové systému typu *FAT*.

Indexová alokace Bloky jsou rozděleny na dva navzájem různé typy. *i-node* bloky obsahují informace o bloku a také indexy datových bloků souboru. Předposlední dva indexy se používají pro jednoúrovňovou indexaci, poslední index se používá pro dvouúrovňovou indexaci.

6.2.2 Implementace adresářů

- záznamy pevné velikosti
- spojový seznam
- B-stromy

6.2.3 Volné místo na disku

Obdoba udržování informací o volném místě v paměti.

- bitmapa
- spojový seznam volných bloků

6.3 Implementace

6.3.1 NTFS

V podstatě neomezená velikost i počet souborů. Jména v *UCS2* délky maximálně 255 znaků. Umožňuje libovolné atributy včetně přístupových práv. Dovolují vícenásobné streamy a *Sparse file* tedy díry ve streamech. Podpora šifrování a komprese. Automatická *fault tolerance* pomocí žurnálování (transakce). Dovolují *hard linky* i *soft linky*. Adresáře jsou soubory s *B-stromy* a odkazy na soubory. Řídicí struktury obsaženy ve speciálním souboru *Master File Table*.

Master File Table Rozdělena na soubory velikosti 1kB. Prvních 16 souborů je speciálních (metasoubory), ty obsahují informace o souborovém systému, o žurnálu, o velikosti, o atributech. Základní záznam obsahuje počet odkazů, příznaky a seznam atributů (včetně streamů). Malé soubory jsou uloženy přímo v tomto záznamu. Pokud pro popis dat nebo seznamu atributů nestačí základní záznam, připojí se další záznamy.

Run list Funkční implementace souvislé alokace pomocí spojového seznamu souvislých částí, prvek je *run* který obsahuje velikost, *virtual cluster number*, *logical cluster number* a odkaz na další *run*.

6.3.2 EXT2/EXT3

Velikost systému 4TB. Jména délky maximálně 255 znaků. Rezervace volného místa pro *roota*. Dovoluje *hard linky* i *soft linky*. Prealokace po 8 souvislých blocích při zápisu, zbytek vrácen při zavření. EXT3 implementuje *žurnálování* jako rozšíření EXT2, souborový systém rozumí transakci ale nedělá žurnálování, několik transakcí je cachováno, a pak uloženo najednou do žurnálu jako složená transakce.

Struktura Disk je rozdělen na části. Na počátku je uložen *boot block*, zbylá část je rozdělena do skupin. Každý blok je rozdělen na část která obsahuje superblok, popis bloku, bitmapy pro uložení informací o volném místě, *i-node* bloky a datové bloky.

i-node Předdefinované soubory odkazující na metasoubory. Několik pevných atributů UID, GID, velikost, různé časy, počet odkazů, 12 přímých odkazů na bloky, jeden nepřímý, jeden dvojitě a jeden trojitě nepřímý soubor. Žurnál je uložen jako rezerovaný *i-node*.

7 Vstupně/výstupní subsystémy

Operační systém je abstraktní stroj. Pro přístup k zařízení slouží řadič zařízení, adaptér jako část I/O jednotky.

7.1 Charakteristiky zařízení

7.1.1 Druhy přístupu

port speciální adresový prostor procesoru

paměťove mapované mapované do fyzické paměti

7.1.2 Druhy zařízení

- blokové
- znakové

7.1.3 Přístup

- sekvenční
- náhodný

7.1.4 Synchronnost komunikace

synchronní zařízení pracuje s daty na žádost

asynchronní zařízení poskytuje i "nevyžádaná" data, v operačním systému musí být dále zpracována

7.1.5 Sdílení

- sdílené, preemtivní
- vyhrazené, nepreemtivní, dedikovaná

7.1.6 Rychlost

Rozsah od několika B/s do GB/s.

7.1.7 Směr dat

- read/write
- read/only
- write/only

7.2 Cíle vstupně/výstupních subsystémů

nezávislost zařízení programy nemusí dopředu vědět, s jakým přesně zařízením budou pracovat

jednotné pojmenování

připojení časté u vyměnitelných zařízení, možné i u pevných zařízení, nutné pro správnou funkci cache operačního systému

obsluha chyb v mnoha případech oprava bez vědomí uživatele, velmi často způsobeno právě uživatelem

7.3 Přenos dat

polling ovladač aktivně čeká na změnu stavu zařízení, přenos dat je úkolem procesoru

přerušení zařízení hlásí připravenost nebo dokončení činnosti, přenos dat je úkolem procesoru

DMA (direct memory access) zařízení si samo řídí přístup na sběrnici a přenáší data z/do paměti, po skončení přenosu dojde k vyvolání přerušení; *scatter-gather DMA* kdy popíši celou sérii bloků které chci přenést

7.4 Vstupně/výstupní software

Typicky ve 4 vrstvách.

1. uživatelský I/O software
2. I/O nezávislý subsystém
3. ovladače zařízení
4. obsluha přerušení

7.4.1 Obsluha přerušení

1. uložení stavu procesoru (kontext vlákna)
2. potvrzení přerušení řadiči přerušení
3. obsluha přerušení ovladačem
4. přeplánování
5. načtení stavu procesoru

7.4.2 Jednotné rozhraní pro zařízení

Poskytuje uživatelským programům jednotné rozhraní. Skrývá rozdíly mezi jednotlivými zařízeními. Provádí pomocné úkoly pro ovladače.

7.4.3 Vyrovnávací paměti

Vyrovnání rychlosti aplikace a zařízení, cachování. Některá zařízení pracují po blocích.

7.5 Zařízení

7.5.1 Disky

Čas na provedení operace se skládá je rozdělen na tři části.

seek čas nutný k přesunu hlaviček

latency zpoždění způsobené rotací

transfer doba přenosu dat z disku do paměti

Prostor disku je dnes lineární.

Plánování pohybu hlav disků

FCFS (First-Come, First-Served) "kdo dřív přijde, ten dřív mele", vhodné pro malé zátěže

SSTF (Shortest Seek Time First) hledám nejmenší vzdálenost od aktuální pozice, v případě velké zátěže se může stát že na krajní pozice se nedostane, vhodný pro malé zátěže

LOOK algoritmus výtahu, hlavy se vždy pohybují od kraje ke kraji jedním směrem, příznak směru pohybu, v průběhu zpracováváme příchozí požadavky ve směru pohybu, vhodný pro velké zátěže

C-LOOK (Circular LOOK) směr provádění je pouze jeden, jakmile hlavy dorazí na konec, vrací se zpět a začínají znovu

7.6 RAID (Redundant Array of Inexpensive Disks)

Pole paralelně zapojených disků. Cílem je vytvořit iluzi jednoho disku použitím více disků.

Důvody

- zvětšení spolehlivosti

  MTBF (Mean Time Between Failure) průměrný čas mezi poruchami

  MTTR (Mean Time To Repair) průměrný nutný k rekonstrukci

- zrychlení

JBOD (Just Bunch Of Disks) spojení disků za účelem zvýšení kapacity

RAID 0 (Striping) spojení disků za účelem zvýšení rychlosti, operace se provádějí paralelně na všechny disky najednou

RAID 1 (Mirroring) spojení disků za účelem zvýšení spolehlivosti, zápisy se provádějí paralelně, ke čtení se používá pouze jeden disk

RAID 0+1 spojení *RAID 0* a *RAID 1*

RAID 2 použití 7-bitového *Hammingova paritního kódu*, data se ukládají po bitech na jednotlivé disky

RAID 3 použití jednoho paritního disku, data se ukládají po bitech na disky

RAID 4 použití jednoho paritního disku a *stripingu*, paritní disk se v tomto případě stává úzkým hrdlem

RAID 5 distribuovaná parita a *striping*, paritní *stripy* jsou na všech discích

RAID 50 spojení více *RAID 5* jako *RAID 0* pomocí *stripingu*

RAID 6 distribuovaná parita P+Q a *striping*

RAID 60 spojení více *RAID 6* jako *RAID 0* pomocí *stripingu*

8 Ochrana a bezpečnost

8.1 Ochrana a bezpečnost

ochrana s prostředky operačního systému mohou pracovat pouze autorizované procesy

bezpečnost zabraňuje neautorizovaným přístup do systému

8.1.1 Domény ochrany

Právo je povolení/zakázání vykonávat nějakou operaci. *Doména* je množina párů objekt a právo. Spojením dostáváme *matici ochrany*.

ACL (Access Control List) ke každému objektu seznam práv pro uživatele/skupiny

C-list (Capability List) ke každému uživateli/skupině seznam práv pro objekty

8.1.2 Cíle útoků

důvěrnost dat uživatel může určit kdo má data vidět, a systém skutečně dovolí pracovat s daty pouze povoleným uživatelům

celistvost dat možnost podstrčení falešných dat

dostupnost systému *DoS (Denial of Service)*

8.1.3 Autentikace

Identifikace něčím, co uživatel ví, má nebo je.

heslo dá se napadnout slovníkovým útokem, hrubou sílou, řešením může být vynucování délky, složitosti hesla

model otázka/odpověď často v kombinaci s heslem

fyzický objekt použití fyzického zařízení

biometrika rozeznání papilárních čar, rohovek, hlasu

8.1.4 Vnitřní útoky

trójský kůň zdánlivě neškodný program obsahuje "zlý" kód

trap door, back door kód při nějaké podmínce přeskočí normální kontroly

buffer overflow ve velkém množství kódu nejsou dělány kontroly na přetečení polí pevné velikosti, při přetečení se typuicky přepíše část zásobníku a lze tam umístit adresu kódu i samostatný kód, který se vykoná při návratu z funkce

8.1.5 Vnější útoky

virus vytvoření nakaženého souboru, po spuštění infikuje další soubory, přenos pomocí médií, sítí

worm samoreplikující se program využívající chyby systému

8.2 Kryptografie

Cíle kryptografie

- důvěrnost
- celistvost
- autentikace
- nemožnost popření

Vrstvy

- kryptografické algoritmy
- kryptografické protokoly

8.2.1 Kryptografický systém

Prostor otevřených zpráv M, šifrovaných zpráv C, šifrovacích a dešifrovacích klíčů K a K'. Efektivní generování klíčů $G : N \to K \times K'$, šifrování $E : M \times K \to C$, dešifrování $D : C \times K' \to M$. Dobrý kryptografický systém dodržuje *Kerchoffovy principy* dobrého kryptografického systému.

Kerchoffovy principy dobrého kryptografického systému

- algoritmy neobsahují žádnou tajnou část
- šifrovací algoritmus distribuuje rozumné zprávy rovnoměrně po celém prostoru šifrovaných zpráv
- se správným klíčem jsou šifrovací a dešifrovací algoritmy ekvivalentní
- bez správného klíče je nalezení otevřené zprávy ze šifrované zprávy NP-úplný problém

Model útočníka podle Doleva a Yao

- může získat libovolnou zprávu putující po síti
- je právoplatným uživatelem sítě a tudíž může zahájit komunikaci s jiným uživatelem
- může se stát příjemcem zpráv kohokoliv
- může zasílat zprávy komukoliv zosobněním se za jiného uživatele
- nemůže odhadnout náhodné číslo z dostatečně velkého prostoru
- bez správného klíče nemůže nalézt zprávu k šifrované zprávě a nemůže vytvořit platnou šifrovanou zprávu z dané zprávy, vše vzhledem k nějakému šifrovacímu algoritmu

8.2.2 Symetrické kryptografické systémy

Šifrovací klíč je shodný s dešifrovacím. Mají vysokou datovou propustnost. Klíče mají malou délku. Slučováním více šifer získáváme silnější šifru.
Klíče na obou koncích musí zůstat utajeny, ty je také třeba často měnit. Potřeba ověřené *Trusted Third Party.*

8.2.3 Asymetrické kryptografické systémy

Šifrovací klíč se liší od dešifrovacího, rozlišuje se veřejný a soukromý klíč. Pouze soukromý klíč je tajný. Klíče není nutné často měnit, není zapotřebý *Trusted Third Party.*
Jsou mnohem pomalejší než symetrické šifry. Používají dlouhé klíče. O žádném schématu veřejného klíče nebylo dokázáno, že je bezpečné.

8.2.4 Útoky na protokoly

- přehrání zpráv
- *man-in-the-middle*
- paralelní spojení
- odražení
- prokládání
- chyba typu
- vypuštění jména
- chybné použití šifrovací služby