

Materiály na SZZ pro obor Webové inženýrství

Matěj Douša, Vojtěch Jedlička

Červen 2024

Obsah

1 Společné okruhy	3
1.1 SP-1 (AAG)	3
1.2 SP-2 (AAG)	7
1.3 SP-3 (AG1)	9
1.4 SP-4 (AG1)	12
1.5 SP-5 (DBS)	15
1.6 SP-6 (DBS)	17
1.7 SP-7 (DML)	19
1.8 SP-8 (DML)	21
1.9 SP-9 (KAB)	24
1.10 SP-10 (KAB)	29
1.11 SP-11 (LA1)	32
1.12 SP-12 (LA1)	35
1.13 SP-13 (MA1)	36
1.14 SP-14 (MA1)	39
1.15 SP-15 (MA2)	41
1.16 SP-16 (MA2)	44
1.17 SP-17 (OSY)	47
1.18 SP-18 (OSY)	51
1.19 SP-19 (PA1)	54
1.20 SP-20 (AG1 + PA1)	56
1.21 SP-21 (AG1 + PA1)	58
1.22 SP-22 (PA2)	60
1.23 SP-23 (PA2)	62
1.24 SP-24 (PSI)	64
1.25 SP-25 (PSI)	68
1.26 SP-26 (PST)	71
1.27 SP-27 (PST)	75
1.28 SP-28 (SAP)	77
1.29 SP-29 (SAP)	79
1.30 SP-30 (SAP)	82
2 Web	83
2.1 W-1 (AWD)	83
2.1.1 Architektura databázového serveru	83
2.1.2 Fyzické soubory	84
2.2 Systémová paměť	84
2.2.1 Úloha databázového administrátora	84
3 Bezpečnost - kvůli referencím	85
3.1 OB-1 (ADU)	85
3.2 OB-2 (ADU)	87
3.3 OB-3 (ADU)	91
3.4 OB-4 (APS)	93
3.5 OB-5 (APS)	95
3.6 OB-6 (APS)	98
3.7 OB-7 (BEK)	101
3.8 OB-8 (BEK)	103

3.9	OB-9 (BEK)	104
3.10	OB-10 (BEK)	106
3.11	OB-11 (ASB)	107
3.12	OB-12 (ASB)	110
3.13	OB-13 (ASB)	113
3.14	OB-14 (EHA)	117
3.15	OB-15 (EHA)	120
3.16	OB-16 (EHA)	122
3.17	OB-17 (HWB)	124
3.18	OB-18 (HWB)	126
3.19	OB-19 (HWB)	128
3.20	OB-20 (UKB)	129
3.21	OB-21 (UKB)	131
3.22	OB-22 (UKB)	132
3.23	OB-23 (ZSB)	133
3.24	OB-24 (ZSB)	135
3.25	OB-25 (ZSB)	136
3.26	OB-26 (ZSB)	138

1 Společné okruhy

1.1 SP-1 (AAG)

Regulární jazyky: Deterministické a nedeterministické konečné automaty. Determinizace konečného automatu. Minimalizace deterministického konečného automatu. Operace s konečnými automaty. Regulární gramatiky, regulární výrazy, regulární rovnice.

Regulární jazyky

- jsou podmnožinou bezkontextových, kontextových i rekurzivně spočetných jazyků
- jazyk je regulární právě tehdy, kdy lze generovat regulární gramatikou
- regulární jazyk lze přijmout (ne)deterministickým konečným automatem
- regulární jazyk lze popsat regulárním výrazem
- regulární jazyky jsou uzavřené pro všechny operace: sjednocení, průnik, doplněk, rozdíl, zřetězení, iterace
- konečný jazyk je vždy regulární

Gramatiky

- prostředek pro popis jazyků
- uspořádaná čtveřice: $G = (N, \Sigma, P, S)$, kde:
 - N je konečná neprázdná množina neterminálních symbolů
 - Σ je abeceda symbolů, které nazýváme terminální symboly — abeceda generovaného jazyka
 - $N \cap \Sigma = \emptyset$
 - P je konečná množina přepisovacích pravidel ve tvaru $\alpha A \beta \rightarrow \gamma$ ($\alpha, \beta, \gamma \in (N \cup \Sigma)^*$, $A \in N$)
 - S je počáteční symbol gramatiky
- regulární gramatika má pravidla ve tvaru $A \rightarrow a$ nebo $A \rightarrow aB$ ($a \in \Sigma, A, B \in N$)
- jediná povolená výjimka je pravidlo $S \rightarrow \epsilon$, které může být přítomno pouze pokud symbol S není na pravé straně žádného pravidla
- regulární gramatika je tedy nezkracující

Konečné automaty

- výpočetní model skládající se z řídící jednotky (stavy a přechodová funkce), read-only vstupní pásky a čtecí hlavy, která čte jednotlivé symboly na pásce
- na počátku je automat v počátečním stavu, a hlava ukazuje na první symbol na pásce
- v každém kroku se přečte symbol, na základě něj se může změnit stav automatu, a hlava se přesune na další znak
- tento proces se opakuje, dokud existuje na vstupu nějaký symbol
- formálně: konečný automat je uspořádaná pětice: $M = (Q, \Sigma, \delta, q_0, F)$, kde:
 - Q je konečná neprázdná množina stavů
 - Σ je konečná vstupní abeceda
 - δ je přechodová funkce
 - * pro deterministický KA je to zobrazení z množiny $Q \times \Sigma$ do množiny stavů Q (tedy $\delta : Q \times \Sigma \rightarrow Q$)
 - * pro nedeterministický KA je to zobrazení z $Q \times \Sigma$ do všech podmnožin Q
 - * NKA se dá navíc rozšířit o ϵ -přechody, přechodová funkce tedy je z množiny $Q \times (\Sigma \cup \epsilon)$
 - * NKA se také dá rozšířit tak, že má více počátečních stavů
 - q_0 je počáteční stav
 - $F \subset Q$ je množina koncových stavů

- NKA i s ε -přechody a více počátečními stavy lze vždy převést na jednoduší variantu, tedy až na DKA
- KA lze zapsat více různými způsoby, liší se v zápisu přechodové funkce (formálně, orientovaný ohodnocený graf, tabulka,...)
- homogenní KA — do konkrétního stavu se lze vždy dostat za pomocí právě jednoho symbolu
- výsledkem ”práce“ automatu je přijmutí/nepřijmutí řetězce (rozhodnutí, zda patří do jazyka)
- řetězec se přijme, pokud se přečte celý, a automat skončí v některém z koncových stavů

Úpravy konečných automatů

- nedosažitelný stav: stav, do kterého se nedá dostat z počátečního stavu žádnou posloupností přechodů
- zbytečný stav: stav, ze kterého se nedá dostat do žádného koncového stavu
- nedosažitelné a zbytečné stavy lze odstranit, a nemá to vliv na jazyk přijímaný automatem
- odstranění ε -přechodů
 - vypočteme funkci ε -closure pro všechny stavy (pro daný stav vrací množinu stavů, kam se lze dostat bez přečtení znaku ze vstupu, tedy kam se dá dostat ε -přechody, obsahuje i sebe sama)
 - z automatu odstraníme ε -přechody, a pomocí ε -closure upravíme přechodovou funkci automatu (pokud pro daný stav S je ε -closure $\{S, A\}$, přidáme přechody ze stavu S do stavů, kam vedou přechody z A)
 - upravíme množinu koncových stavů — pokud stav v ε -closure měl původní koncový stav, je teď nově také koncový
- odstranění více počátečních stavů — přidáme nový, jediný počáteční, a z něj uděláme ε -přechody do původních počátečních
- determinizace (převod NKA na DKA)
 - na vstupu předpokládáme NDA s jedním počátečním stavem a bez ε -přechodů
 - tvoříme nová (D)KA tak, že procházíme původní NKA po množinách jeho stavů (stav nového DKA — množina všech původních stavů, kde NKA v tu chvíli může být)
 - pro každou takto objevenou množinu dále tvoříme přechodovou funkci
 - ve chvíli, když již máme všechny objevené množiny ”zpracované“, máme hotovo
- minimalizace
 - ekvivalentní stavy — stavy, po jejichž sloučení se nezmění přijímaný jazyk
 - minimální DKA — takový automat pro daný jazyk L , který má co nejméně stavů, tedy nemá ani nedosažitelné, zbytečné ani ekvivalentní stavy
 - nejprve odstraníme nedosažitelné a zbytečné stavy (zjistíme jednoduše BFS z počátečního/koncových stavů)
 - rozdělíme stavy na koncové a nekoncové, stavy přejmenujeme podle toho, do které skupiny patří
 - pro tyto přejmenované stavy vyplníme přechodovou funkci podle tohoto nového pojmenování
 - projdeme, zda všechny stavy v rámci jedné skupiny mají shodné přechody
 - pokud najdeme ve skupině odlišný stav, vytvoříme pro něj novou skupinu, případně pokud už jsme v aktuálním průchodu narazili na ”stejně odlišný“ stav, tak je dáme do stejné nové skupiny
 - předchozí 3 kroky opakujeme, dokud se nějakým způsobem mění složení skupin a pojmenování stavů
- ekvivalentní KA — převedeme na minimální DKA, a nalezneme isomorfismus

Skládání KA

- sjednocení
 - pokud jsou vstupem NKA, NKA s ε -přechody nebo DKA — bud' prostě více počátečních stavů, nebo nový 1 počáteční stav a ε -přechody z něj
 - pokud vstupem jsou úplně určené DKA nebo NKA, lze využít paralelní činnosti
 - * podobné jako determinizace
 - * začne se ve stavu, který představuje počáteční stavy obou vstupních automatů
 - * postupuje se podle přechodových funkcí vstupních automatů, každý nový stav je kombinací stavů z obou původních automatů
- průnik
 - pomocí paralelní činnosti, podobně jako sjednocení, ale automaty nemusí být úplně určené
 - tam kde v jednom automatu není přechod, nebude existovat ani ve výsledném automatu
 - koncové stavy jsou navíc jen ty, které ”obsahují” pouze koncové stavy obou vstupních automatů
- doplněk jazyka — vstup je úplně určený DKA, prohodíme koncové a nekoncové stavy
- rozdíl — $L(M_1) \setminus L(M_2) = L(M_1) \cap \overline{L(M_2)}$
- součin (zřetězení) — ze všech koncových stavů M_1 ε -přechody do počátečního stavu M_2 , koncové stavy budou jen M_2 a počáteční jen M_1
- iterace — nový počáteční stav (zároveň koncový), ε -přechod do původního počátečního, ze všech koncových vedeme ε -přechody do nového počátečního

Regulární výrazy

- $\emptyset, \varepsilon, a$ jsou regulární výrazy pro všechna $a \in \Sigma$
- pokud jsou x, y regulární výrazy nad Σ , pak $(x + y)$, $(x \cdot y)$ a $(x)^*$ jsou regulární výrazy nad Σ
- hodnota regulárního výrazu je regulární jazyk, který daný výraz reprezentuje
 - $h(\emptyset) = \emptyset$ — prázdný výraz reprezentuje prázdný jazyk
 - $h(\varepsilon) = \{\varepsilon\}$
 - $h(a) = \{a\}$
 - $h(x + y) = h(x) \cup h(y)$
 - $h(x \cdot y) = h(x) \cdot h(y)$
 - $h(x^*) = h(x)^*$
- lze podle různých pravidel zjednodušovat (např \emptyset se chová jako 0 při násobení i sčítání, ε se chová jako 1 při násobení)
- regulární rovnice
 - levá regulární rovnice $x = x\alpha + \beta$
 - pravá regulární rovnice $x = \alpha x + \beta$
 - α, β jsou známé výrazy, x neznámý
 - potřebujeme dostat neznámý výraz ”ven” z podvýrazů, aby se dala aplikovat levá/pravá rovnice
 - soustavu řešíme tak, že co jde dosadit rovnou do dalších rovnic dosadíme, jinak jednu vyřešíme klasickým způsobem a pak dosadíme
- derivace regulárních výrazů — odebírání předpony
- pokud nejde odebrat konkrétní předponu, výsledek je \emptyset
- pravidla:
 - $\frac{dV}{d\varepsilon} = V$
 - $\frac{d\varepsilon}{da} = \emptyset$

- $\frac{d\emptyset}{da} = \emptyset$
- $\frac{db}{da} = \emptyset$
- $\frac{da}{da} = \varepsilon$
- $\frac{d(U+V)}{da} = \frac{dU}{da} + \frac{dV}{da}$
- $\frac{d(UV)}{da} = \frac{dU}{da}V + \left\{ \frac{dV}{da} : \varepsilon \in h(U) \right\}$
- $\frac{d(V^*)}{da} = \frac{dV}{da} \cdot V^*$

Převody

- RG \leftrightarrow KA — stavy jsou většinou stejné jako neterminály, jen se musí pořešit koncové stavy
- RV \rightarrow KA
 - metoda sousedů
 - * každý znak se ”očíslouje” — tedy každý znak se považuje za jiný
 - * vytvoří se množiny znaků co můžou být na začátku, na konci, a množina možných sousedních znaků
 - * každý znak má v automatu vlastní stav + 1 nový počáteční stav, koncové jsou koncové, z nového počátečního (pokud $\varepsilon \in L$ tak zároveň koncového) hrany do počátečních znaků, pak se doplní hrany podle množiny sousedů
 - metoda derivací — postupně derivujeme všemi znaky abecedy, každý nový regulární výraz je nový stav
 - metoda postupné konstrukce — sestavujeme automat od elementárních výrazů
- KA \rightarrow RV
 - metoda regulárních rovnic — odchozí přechody — sestavíme soustavu rovnic, každý stav je neznámá, pro odchozí přechody stavíme pravé rovnice, ke koncovým stavům přidáme $+\varepsilon$ a výsledkem je výraz pro počáteční stav
 - RR — příchozí přechody — podobně jako odchozí, ale staví se levé rovnice, $+\varepsilon$ se přidá k počátečnímu stavu, a výsledkem je sjednocení (+) výrazů koncových stavů
 - eliminace stavů — přidáme nový počáteční stav (a také nový koncový, pokud původních koncových je více) a postupně odebíráme stavy s tím, že postupně sestavujeme regulární výraz
- RG \rightarrow RV
 - metoda regulárních rovnic — vezmeme pravidla, předěláme na reg. rovnice a vyřešíme pro počáteční neterminál
 - eliminace neterminálních symbolů — jako eliminace stavů
- RV \rightarrow RG
 - metoda derivací
 - postupná konstrukce

Pumping lemma

- regulární jazyk \Rightarrow pumpující vlastnost
- používá se pro důkaz neregularity jazyka
- najdeme nějakou větu z jazyka, rozdělíme ji na 3 části xyz , $|xy| \leq p$, $|y| \geq 1$, a najdeme k takové, aby $xy^kz \notin L$

1.2 SP-2 (AAG)

Bezkontextové jazyky: Bezkontextové gramatiky, zásobníkové automaty a jejich varianty. Modely syntaktické analýzy bezkontextových jazyků.

Bezkontextové jazyky

- formální jazyk je bezkontextový právě tehdy, kdy jej lze generovat bezkontextovou gramatikou
- lze přijímat zásobníkovým automatem
- BJ jsou uzavřené vůči sjednocení, zřetězení a iteraci
- nejsou uzavřené pro průnik, doplněk a rozdíl

Bezkontextové gramatiky

- uspořádaná čtverice: $G = (N, \Sigma, P, S)$, kde pravidla jsou ve tvaru: $A \rightarrow \alpha \quad (\alpha \in (N \cup \Sigma)^* A \in N)$
- bezkontextová gramatika gramatika může být zkracující
- nejednoznačná BG — lze pro jeden řetězec z jazyka sestavit alespoň 2 různé derivační stromy
- jednoznačná BG — pro všechny řetězce existuje právě jeden derivační strom
- nejednoznačný bezkontextový jazyk — neexistuje pro něj jednoznačná gramatika
- jednoznačný BJ — lze generovat jednoznačnou gramatikou
- ϵ -pravidlo — pravidlo tvaru $A \rightarrow \epsilon$
- jednoduché pravidlo — pravidlo tvaru $A \rightarrow B$
- gramatika je bez cyklů ($A \Rightarrow^+ A$), pokud neobsahuje jednoduchá a ϵ pravidla

Zásobníkové automaty

- můžeme si představit jako konečný automat se zásobníkem
- (nedeterministický) ZA formálně: $R = (Q, \Sigma, G, \delta, q_0, Z_0, F)$
 - Q je konečná neprázdná množina stavů
 - Σ je konečná vstupní abeceda
 - G je konečná neprázdná abeceda zásobníku
 - δ je přechodová funkce, definována jako zobrazení z $Q \times (\Sigma \cup \{\epsilon\}) \times G^*$ do množiny konečných podmnožin množiny $Q \times G^*$
 - $q_0 \in Q$ je počáteční stav
 - $Z_0 \in G$ je počáteční symbol na zásobníku
 - $F \subset Q$ je množina koncových stavů
- nedeterministický ZA je výpočetně silnější než deterministický
- DZA přijme řetězec, pokud přečte celý řetězec, a skončí buď v koncovém stavu, nebo s prázdným zásobníkem (výpočetně ekvivalentní způsoby, jdou na sebe převádět)

Syntaktická analýza

- levá derivace — v každém kroku generování věty bezkontextovou gramatikou se nahradí neterminál, který je nejvíce vlevo
- pravá derivace — v každém kroku generování věty bezkontextovou gramatikou se nahradí neterminál, který je nejvíce vpravo
- levý rozklad věty — posloupnost čísel pravidel použitých v levé derivaci
- pravý rozklad věty — posloupnost čísel pravidel použitých v pravé derivaci

- syntaktická analýza je proces, který pro danou bezkontextovou gramatiku G a řetězec ω určí, zda $\omega \in L(G)$
- v kladném případě také získáme syntaktickou strukturu řetězce v podobě levého nebo pravého rozkladu
- metody syntaktické analýzy:
 - shora dolů (top-down), která nalezne levý rozklad
 - zdola nahoru (bottom-up), která nalezne pravý rozklad
- jak pro danou gramatiku vytvořit ZA jako model syntaktického analyzátoru metodou shora dolů?
 - 1 stav, přijímá prázdným zásobníkem
 - provádí 2 operace — expanzi a srovnání
 - * expanze: pokud mám na vrcholu zásobníku neterminál, vyndám ho a nahradím pravidlem gramatiky, ze vstupu nečtu nic
 - * srovnání: pokud mám na vrcholu zásobníku terminál, vyndám ho a zároveň přečtu ze vstupu
 - na začátku je na zásobníku počáteční neterminál
- jak pro danou gramatiku vytvořit ZA jako model syntaktického analyzátoru metodou zdola nahoru?
 - 2 stavy, přijímá přechodem do koncového stavu
 - provádí 3 operace — redukci, přesun a přijetí
 - * redukce: zůstávám ve stejném stavu, nečtu nic ze vstupu, pokud je na vrcholu zásobníku pravá strana pravidla, vyndám a nahradím příslušným neterminálem
 - * přesun: přečtu terminál ze vstupu a vložím na zásobník, zůstávám ve stejném stavu
 - * přijetí: na zásobníku je počáteční neterminál — vyndám spolu s počátečním symbolem a přejdu do koncového stavu

1.3 SP-3 (AG1)

Základní pojmy teorie grafů. Grafové algoritmy: procházení grafu do šířky, určení souvislých komponent, topologické uspořádání, vzdálenosti v grafech, konstrukce minimální kostry a nejkratších cest v ohodnoceném grafu.

Základní pojmy

- neorientovaný graf — usporádaná dvojice (V, E) , kde V je neprázdná konečná množina vrcholů a E je množina hran
- hrana je dvouprvková podmnožina V , značíme $\{u, v\}$
- množina všech možných hran je $\binom{V}{2}$
- nechť $e = \{u, v\}$ je hrana grafu G , pak jsou u, v koncové vrcholy / incidentní s e , a u je sousedem v a naopak
- sled v grafu: sekvence vrcholů a hran v grafu dané délky (délka dána počtem hran), hrany musí být mezi navazujícími vrcholy (prostě jak a přes co se někam dostat)
- cesta: sled, ve kterém se neopakují vrcholy (a tedy ani hrany)
- orientovaný graf — dvojice (V, E) , kde E je množina orientovaných hran
- stupeň vrcholu — počet hran obsahujících vrchol (nebo také počet sousedů, pokud nepovolíme více hran mezi stejnými vrcholy)
- okolí vrcholu: množina sousedů
- regulární graf — všechny vrcholy stejný stupeň
- úplný graf (K_n) — množina vrcholů $E = \binom{V}{2}$
- úplný bipartitní graf $(K_{m,n})$ — tvořen 2 partitami, hrany vedou pouze z jedné do druhé
- cesta P_m
- kružnice C_n
- doplněk grafu \overline{G} — stejné vrcholy, ale doplněk hran (kde byly nejsou, kde nebyly jsou)
- podgraf — množiny V a E jsou podmnožinami původního grafu
- indukovaný podgraf — podgraf, jehož množina hran je maximální možná
- princip sudosti — součet stupňů vrcholů je 2krát počet hran
- u orientovaných grafů je vstupní a výstupní stupeň vrcholu
- vstupní stupeň 0 — zdroj
- výstupní stupeň 0 — stok
- graf je souvislý, pokud mezi každými 2 vrcholy existuje cesta, jinak je nesouvislý
- souvislá komponenta — maximální souvislý podgraf
- paměťová reprezentace grafu — seznam sousedů ($O(|V| + |E|)$) nebo matice sousednosti ($O(|V|^2)$)
- symetrizace orientovaného grafu: pokud mezi vrcholy vede jakákoli orientovaná hrana, nahradíme neorientovanou
- slabá souvislost — symetrizace je souvislá
- silná souvislost — pro každé dva vrcholy musí existovat cesta tam a zase zpátky
- izomorfismus grafů — $f(G)$ bijekce vrcholů na vrcholy, hrany musí být pořád "mezi stejnými vrcholy"
- strom — graf, který je souvislý a neobsahuje kružnice
- kostra grafu — podgraf, který obsahuje všechny vrcholy původního grafu a je to strom
- topologické uspořádání orientovaného grafu — seřazení vrcholů tak, aby hrany vždy vedly "zleva doprava"

Grafové algoritmy

- BFS
 - prohledávání grafu do šírky
 - začne v daném vrcholu, postupně nově nalezené vrcholy přidává do fronty a z fronty postupně zpracovává a rozšiřuje dál
 - lze použít pro hledání vzdálenosti mezi vrcholy (délka nejkratší cesty)
 - také lze využít např. pro hledání souvislých komponent nebo kostry grafu
 - při reprezentaci seznamem sousedů je časová i paměťová složitost $O(n + m)$
- DFS
 - prohledávání grafu do hloubky
 - rozšiřuje v nejnověji nalezených vrcholech, lze použít zásobník místo fronty, nebo implementovat rekurzí
 - lze použít pro hledání kostry
 - také lze využít např. pro hledání souvislých komponent
 - při reprezentaci seznamem sousedů je časová i paměťová složitost $O(n + m)$
- TopSort
 - výsledkem je topologické uspořádání vrcholů
 - spočítá vstupní stupně vrcholů, najde zdroje (vstupní stupeň je nula) a postupně odebírá zdroje, tím vznikají vždy další (pokud b grafu není orientovaný cyklus)
 - časová i paměťová složitost $O(n + m)$
- Jarník
 - výsledkem je minimální kostra ohodnoceného grafu
 - začne v jednom vrcholu, postupně přidává další vrcholy podle toho, jaká je v aktuálním řezu grafu (mezi již vybranými vrcholy a ostatními) hrana s nejmenší hodnotou
 - paměťová složitost $O(n + m)$, naivní implementace má časovou složitost $O(mn)$, implementace s použitím binárních minimových hal $O(m \log n)$
- Kruskal
 - výsledkem je minimální kostra ohodnoceného grafu
 - začne se všemi vrcholy, ale bez hran
 - postupně přidává hrany od nejlevnějších, zahazuje ty, které by vytvořily cyklus
 - implementace s keříky (struktura Union-find) má složitost $O(m \log n)$
- Dijkstra
 - prohledávání ohodnoceného grafu (omezení na kladné ohodnocení hran)
 - hledá nejkratší cesty
 - podobně jako BFS prochází vrcholy, jde vždy ale do vrcholu, který je aktuálně nejblíže startovnímu vrcholu
 - otevře nový vrchol (nejbližší startu), podívá se na hrany z něj a přepočítá případně rychlejší cesty do dosud neuuzavřených sousedů
 - pojem relaxace — přepočítání sousedů vrcholu
 - složitost $O(m \log n)$ při implementaci s binární haldou
- Bellman-Ford
 - další relaxační algoritmus prohledávání ohodnoceného grafu (obecné ohodnocení hran, jen nesmí existovat záporné cykly)
 - hledá nejkratší cesty

- podobně jako dijkstra, ale prochází vrcholy ”náhodně” (ne podle vzdálenosti od startu, ale podle toho který byl dříve nalezený, podobně jako BFS)
- počítá se s tím, že uzavřené vrcholy lze znovu otevřít
- otevře nový vrchol, podívá se na hrany z něj a přepočítá případné rychlejší cesty do sousedních vrcholů, které mohou být i uzavřené, tedy je znovu otevře
- složitost $O(m \cdot n)$

1.4 SP-4 (AG1)

Binární haldy, binomiální haldy. Binární vyhledávací stromy a jejich vyvažování. Hešovací tabulky.

Stromy

- zakořeněný strom — jeden vrchol prohlášen za kořen, vzniká vazba mezi vrcholy předek-potomek
- binární strom — zakořeněný strom, kdy každý vrchol může mít nejvýše 2 syny, a rozlišujeme, který je levý/pravý
- binomiální strom
 - binomiální strom má vždy nějaký řád k
 - obsahuje právě 2^k vrcholů
 - nultý řád je jeden izolovaný vrchol
 - další řády získáme tak, že vezmeme stromy všech předchozích řádů, a nad ně přidáme kořen
 - to samé získáme tak, že vezmeme 2 binomiální stromy řádku $k - 1$ a jeden připojíme pod kořen druhého

Binární haldy

- datová struktura tvaru binárního stromu
- v každém vrcholu je uložen klíč
- haldový tvar — všechny hladiny kromě poslední musí být obsazeny, poslední je obsazena zleva
- haldové uspořádání — pro každou dvojici předek-potomek platí, že klíč v předu je menší nebo roven potomku
- základní operace:
 - insert() — dle tvaru haldy vloží na konec, pak probublá nahoru ($O(\log n)$)
 - findMin() — vrátí minimum, které je vždy v kořenu ($O(1)$)
 - extractMin() — odstraní z haldy minimum tak, že ho prohodí s posledním prvkem, smaže poslední prvek a zabublá kořen dolů ($O(\log n)$)
- heapBuild() — postavení haldy odspodu, každý list je korektní haldy, postupně jdeme do vyšších hladin a nový kořen vždy probubláme dolů ($O(n)$)
- typicky se implementuje pomocí pole

Binomiální halda

- uspořádaná množina binomiálních stromů
- stromy jsou uspořádány vzestupně dle svých řádů
- celkový počet prvků v haldě je součet velikostí stromů
- pro každý řád k se v haldě vyskytuje maximálně jeden binomiální strom
- vrcholy (podobně jako v binární haldě) obsahují klíče
- stromy si drží haldové uspořádání
- množina stromů se implementuje spojovým seznamem
- n -prvková halda má až $O(\log n)$ binomiálních stromů
- základní operace:
 - findMin() — vrátí minimum, které musí být v kořeni některého ze stromů (složitost $O(\log n)$, v případě implementace ukazatele na minimum je složitost $O(1)$)
 - merge() — sloučení 2 binomiálních hal

- * postupuje se jako při binárním scítání — stromy se slučují od nejnižších rádu
- * sloučení dvou binomálních stromů je $O(1)$ — stačí porovnat kořeny a zařadit větší pod menší
- * složitost celého spojení je úměrná počtu stromů v haldách, tedy $O(\log n)$
- `insert()` — vytvoření jednoprvkové nové haldy, následně merge ($O(\log n)$)
- `extractMin()` — naleze se strom, kde se minimum nachází, tomuto stromu se utrhne kořen, a ze vzniklých stromů se vytvoří nová halda a provede se merge s původní ($O(\log n)$)

Binární vyhledávací stromy

- binární stromy, kde v každém vrcholu je uložený unikátní klíč, a kde klíče v levém podstromu jsou vždy menší a v pravém větší než v aktuálním vrcholu
- $h(T)$ — hloubka stromu
- operace:
 - `BVSShow()` — vypsání klíčů ve vzestupném pořadí ($O(n)$)
 - `BVSMIn()` — nalezení nejmenšího prvku, je to ten nejvíc vlevo ($O(h(T))$)
 - `BVSMMax()` — podobně jako `Min()`
 - `BVSPred()` — vrátí předchůdce prvku (ve vzestupném výpisu by byl levým sousedem) $O(h(T))$
 - * pokud má prvek levý podstrom, pak je předchůdce maximum z podstromu
 - * jinak je to první prvek, do kterého vstoupíme zprava při průchodu stromem nahoru
 - `BVSSucc()` — následník, podobně jako `Pred()`
 - `BVSFind()` — hledá prvek s konkrétním klíčem ($O(h(T))$)
 - `BVIInsert()` — vloží prvek na správné místo ($O(h(T))$)
 - `BVSDDelete()` — odstraní prvek ($O(h(T))$)
 - * pokud je prvek list, můžeme odtrhnout
 - * pokud má jednoho syna, nahradíme tímto synem
 - * pokud má 2 syny, prohodíme s následníkem/předchůdcem a smažeme potom (následník/předchůdce existují a je v pravém/levém podstromu aktuálního vrcholu, a mají maximálně 1 syna)
- rychlosť velké časti operací závisí na hloubce stromu $h(T)$
- v nejhorším případě může být $h(T) == |T|$, tedy složitost operací by byla $O(n)$
- hloubku budeme regulovat vyvažováním
- lze vyvažovat ”dokonale” — tedy rozdíl počtu prvků v levém a pravém podstromu každého prvku je maximálně 1, to ale není efektivní — při jakékoliv implementaci bude složitost `insert()` nebo `delete()` $O(n)$
- proto využíváme hloubkově vyvážené stromy jako AVL
- AVL:
 - rozdíl hloubek podstromů každého vrcholu může být maximálně 1
 - hloubka celého BVS je pak $\Theta(\log n)$
 - operace `insert()` a `delete()` musí udržovat hloubkové vyvážení (jiné ho nemění)
 - v každém vrcholu se udržuje znaménko vyvážení
 - pokud je více než (minus) jedna, je potřeba rotovat
 - rotace se provede ve směru podle znaménka (podle nevyvážení stromu)
 - vyvážení se opravuje jednoduchými nebo dvojitými rotacemi

Hashovací tabulky

- skloubení nízkých paměťových nároků s efektivitou operací (pouze v průměrném případě)
- implementace slovníku — klíč a hodnota, klíč je unikátní
- zvolíme konečné pole přihrádek a hashovací funkci h , která každému klíči univerza přiřadí jednu přihrádku
- pokud chceme vložit prvek s klíčem k do tabulky, vložíme ho do přihrádky $h(k)$
- pokud budeme hledat prvek s klíčem k , víme že musí být v přihrádce $h(k)$
- protože ale univerzum klíčů bývá zásadně větší než počet přihrádek, stávají se kolize — více klíčů se přiřadí do stejné přihrádky
- ideální hashovací funkce vypočte číslo přihrádky v konstantním čase a rozprostře zadanou n -prvkovou množinu rovnoměrně do všech m přihrádek, tedy každá přihrádka má $\lceil n/m \rceil$ prvků
- příklady dobře fungujících hashovacích funkcí:
 - lineární kongruence ($k \rightarrow a \cdot k \bmod m$) — m je typicky prvočíslo, a je vysoká konstanta nesoudělná s m
 - vyšší byty součinu ($k \rightarrow \lfloor (ak \bmod 2^\omega) / 2^{\omega-\mathcal{L}} \rfloor$) — hashujeme ω -bitové klíče do $m = 2^\mathcal{L}$ přihrádek, vybereme ω -bitovou lichou konstantu a , pak spočítáme ak , ořízneme na ω bitů a z nich vezmeme nejvyšších \mathcal{L}
 - pro posloupnosti — skalární součin nebo polynom
- řešení kolizí
 - chaining (řetězení) — prvky se při kolizi vloží do stejné přihrádky do spojáku
 - otevřená adresace — klíč se při kolizi vloží na jiné, volné místo dle určitého pravidla:
 - * lineární přidávání — prostě se posune o 1 vedle
 - * dvojité hashování — znova, novou hashovací funkcí se klíč přehashuje a uloží jinam ($h(k, i) = (f(k) + i \cdot g(k)) \bmod m$) — f, g jsou 2 různé hash funkce, i značí kolikátý pokus to je na umístění
 - * při mazání musíme označit, že tam nějaký prvek byl (aby se našly ty další co se daly jinam), říká se tomu náhrobek
- tabulka se může moc naplnit, pak ji chceme zvětšit, a klíče musíme přehashovat a umístit do rozšířené tabulky

1.5 SP-5 (DBS)

Relační databáze, dotazování v relační algebře, základní koncepce jazyka SQL (SELECT, DDL, DML, DCL, TCL), vyjádření integritních omezení v DDL.

Relační databáze

- databáze je soubor záznamů, jejichž systematická struktura umožnuje, aby tyto zprávy mohly být vyhledávány pomocí počítače
- existence dat v DB je nezávislá na aplikačních programech
- zabývá se řízením velkého množství, perzistentních, spolehlivých a sdílených dat
 - velké množství — nestačí operační paměť
 - perzistentní — data přetrvávají od zpracování ke zpracování
 - spolehlivá — lze rekonstruovat po chybě
 - sdílená — přístupná více uživatelům
- hlavní přínosy DB technologií:
 - nezávislost dat na aplikaci
 - efektivní přístup k datům
 - urychlení vývoje aplikací
 - integrita a ochrana dat
 - správa a zálohování dat
 - transakční zpracování
 - paralelní přístup k datům
 - zotavení po chybě
- DBS — Database system
- DBMS — Database Management system
- relace — dvouozměrná struktura (tabulka)
 - atributy (sloupce) — jména a domény atributů
 - n-tice (řádky) — prvky relace (unikátní)
- schéma relační databáze: množina relací R a množina integritních omezení I

Relační algebra

- pouze dotazovací formalismus (vyhledávání dat)
- specifikujeme co chceme, ne jak to získat
- výsledkem dotazu je relace (lze použít pro další dotaz)
- jakýkoliv jazyk realizující relační algebru se nazývá relačně úplný
- dotazování:
 - nazev_relace("atribut" = hodnota) — vyhledání záznamů v dané relaci, které vyhovují podmínce (podmínka není nutná) — tento proces se nazývá selekce
 - nazev_relace(podminka)[atribut1, atribut2] — hranaté závorky jsou výběr atributů, které se vloží do výsledné relace — nazýváme projekce
 - relace1 * relace2 — přirozené spojení relací na základě stejně se jmenujících atributů
 - nazev_relace[atribut1 → jine_jmeno]
 - množinové operace jako sjednocení, průnik, rozdíl a kartézský součin (\cup , \cap , \setminus , \times)
 - polospojení (left/right join) relace1 *> relace2

SQL

- Structured Query Language
- relačně úplný (jakýkoliv dotaz v RA lze převést do SQL)
- podobně jako u RA řešíme jaký chceme výsledek, ne jakým způsobem ho získat
- části SQL:
 - DDL (Data Definition Language) — create, alter, drop (tvorba relací samotných, také řeší integritní omezení)
 - DML (Data Modification Language) — insert, update, delete, merge, transakční zpracování (úprava záznamů v relacích / tabulkách)
 - DCL (Data Control Language) — grant, revoke (úprava práv uživatelů, přístupy)
 - TCL (Transaction Control Language) — commit, rollback, savepoint

Integritní omezení

- realizované pomocí DDL
- můžou se kontrolovat periodicky či při každé úpravě
- 2 typy omezení:
 - deklarativní (pro domény atributů) — kontrola hodnoty samotných atributů, např. nesmí být nějaké hodnoty (NOT NULL, CHECK) nebo další omezení (UNIQUE, PRIMARY KEY, FOREIGN KEY / REFERENCES)
 - procedurální (složitější) — nad celými tabulkami, trigger
- vyjádření integritních omezení v příkazu CREATE:

```
DROP TABLE KINA CASCADE CONSTRAINTS;
CREATE TABLE KINA ...
...
CREATE TABLE PREDSTAVENI
(NAZEV_K Char Varying(20) NOT NULL,
NAZEV_F Character Varying(20) NOT NULL,
DATUM date NOT NULL,
CONSTRAINT PREDSTAVENI_PK
PRIMARY KEY (NAZEV_K, NAZEV_F),
CONSTRAINT PREDSTVENI_KINA_FK
FOREIGN KEY (NAZEV_K) REFERENCES KINA,
CONSTRAINT PREDSTAVENI_FILMY_FK
FOREIGN KEY (NAZEV_F) REFERENCES FILMY);
```

1.6 SP-6 (DBS)

Transakce a jejich vlastnosti - ACID.

Požadavky na databázi

- ochrana dat před chybami, havárií serveru — řeší recovery modul (při systémových/hw chybách se DB vždy vrátí do konzistentního stavu)
- korektní, rychlý a asynchronní přístup pro větší počet uživatelů — řeší concurrency control modul (každý uživatel vidí konzistentní stav)

Transakce

- sekvence souvisejících akcí, která dostane DB z jednoho konzistentního stavu do druhého
- v průběhu může existovat nekonzistentní stav, ale vždy se musí dostat opět do konzistentního stavu
- tedy se provede buď celá, nebo vůbec
- začátek transakce — vznik session nebo konec předchozí transakce
- konec transakce — ukončení session, nebo klíčová slova COMMIT/ROLLBACK
- stavy transakce:
 - aktivní (A) — probíhají DML příkazy
 - částečně potvrzená (PC) — po provedení poslední transakce
 - potvrzená (C — Committed) — po úspěšném zakončení (COMMIT)
 - chybná (F) — nelze pokračovat v normálním průběhu transakce
 - zrušená (AB — aborted) — po skončení operace ROLLBACK

ACID vlastnosti transakce

- Atomicity — proběhne celá nebo vůbec
- Consistency — transformuje DB z konzistentního stavu do konzistentního stavu
- Independence — dílčí efekty jedné transakce nejsou viditelné jiným transakcím
- Durability — uložené efekty jsou trvale uloženy

Recovery

- využívá se žurnál (log) obsahující změnové vektory
- různé typy chyb: globální (pád serveru, ztráta spojení, incident na disku) a lokální (logické chyby v konkrétní transakci)
- nedokončené transakce se odvolávají (ROLLBACK)
- potvrzené transakce, jejichž efekt se nestihl např. zapsat na disk se zopakují

Rozvrhování transakcí

- transakce a jejich dílčí operace se musí nějak rozplánovat
- toto rozvržení musí zajistit korektní asynchronní přístup pro více uživatelů
- pro snazší realizaci se používá zamykání a uzamykací protokoly
- legální rozvrh (nezaručuje uspořádatelnost):
 - transakce musí mít objekt zamknutý, aby s ním mohla pracovat
 - transakce nemůže zamykat již zamknuté objekty
- tam se používají takzvané dobře formované transakce

- transakce zamyká objekt, chce-li k němu přistupovat
 - transakce nezamyká objekt, pokud ho již zamkla
 - transakce neodemyká objekt, který nezamkla
 - na konci nezůstane žádný objekt zamčený
- pro zamezení deadlocku se využívá dvoufázový uzamykací protokol
 - 1. fáze — pouze se uzamyká co je potřeba (klidně postupně spolu s dílčími operacemi transakcí, ale nesmí se nic odemykat)
 - 2. fáze — pouze se odemyká, po prvním odemčení se již nesmí nic zamykat
 - pokud jsou všechny transakce dobře formované a dvoufázové, lze pro ně vytvořit uspořádatelný rozvrh (aby na sebe nečekaly a nezamkly se)
 - používá se rozvrhování i jiných operací, pak v případě deadlocku se u jedné operace prostě provede rollback

1.7 SP-7 (DML)

Výroková logika: splnitelnost formulí, logická ekvivalence a důsledek, universální systém logických spojek, disjunktivní a konjunktivní normální tvary, úplné normální tvary.

- **Prvotní výrok:** jednoduchá oznamovací věta, u které má smysl se ptát zda je či není pravdivá. Prvotní výroky označujeme velkými písmeny, říkáme jim **prvotní formule**.
- **Pravdivostní ohodnocení:** ohodnocení množiny prvotních výroků je přiřazení v , které každé prvotní formuli přiřadí 0 nebo 1.
 - Je-li $v(A) = 1$, říkáme že A je pravdivý při ohodnocení v .
 - Je-li $v(A) = 0$, říkáme že A je nepravdivý při ohodnocení v .
- **Negace:** $\neg A$ výroku A je pravdivá pro všechna ohodnocení, při kterých je A nepravdivý. Pro ostatní je nepravdivá.
- **Konjunkce:** $A \wedge B$ výroků A a B je pravdivá pro všechna ohodnocení, při kterých jsou A i B současně pravdivé. Pro ostatní ohodnocení je nepravdivá.
- **Disjunkce:** $A \vee B$ výroků A a B je pravdivá pro všechna ohodnocení, při kterých je alespoň jeden z výroků A a B pravdivý. Pro ostatní ohodnocení je nepravdivá.
- **Implikace:** $A \Rightarrow B$ mezi výroky A a B je nepravdivá pro všechna ohodnocení, kdy **předpoklad** A platí a **závěr** B neplatí. Pro ostatní ohodnocení je pravdivá.
- **Ekvivalence:** $A \Leftrightarrow B$ mezi výroky A a B je pravdivá pro všechna ohodnocení, při kterých mají výroky A a B stejnou pravdivostní hodnotu. Pro ostatní je nepravdivá.
- **Tautologie (\top):** Formule, která je pro každé ohodnocení pravdivá.
- **Kontradikce (\perp):** Formule, která je pro každé ohodnocení nepravdivá.
- **Splnitelná formule:** Formule, která je alespoň pro jedno ohodnocení pravdivá.
- Nechť E a F jsou výroky. Pokud platí $E \Rightarrow F$, pak E je **postačující podmínka** pro F . Na druhou stranu, F je **nutná podmínka** pro E . Pokud platí $E \Leftrightarrow F$, pak je E **nutná a postačující podmínka** pro F a obráceně.
- Nechť E a F jsou výrokové formule. E a F jsou logicky ekvivalentní, právě když pro každé ohodnocení v je $v(E) = v(F)$. Píšeme $E \models F$.
- Nechť E a F jsou výrokové formule. F je logickým důsledkem E , právě když pro každé ohodnocení v , pro které $v(E) = 1$, je i $v(F) = 1$. Píšeme $E \models F$.
- **Základní principy logiky:**
 - Zákon vyloučení sporu: $A \wedge \neg A \models \perp$
 - Zákon vyloučení třetího: $A \vee \neg A \models \top$
 - Zákon dvojí negace: $\neg\neg A \Leftrightarrow A \models \top$
- **Obměněná implikace:** $(E \Rightarrow F) \models (\neg F \Rightarrow \neg E)$
- Množina logických spojek tvoří **universální systém**, právě když ke každé formuli existuje logicky ekvivalentní formule, která obsahuje pouze tyto spojky.
- Např. dvouprvkové systémy: $\{\neg, \vee\}$, $\{\neg, \wedge\}$, $\{\neg, \Rightarrow\}$
- Existují i jednoprvkové systémy pouze z NAND (\uparrow) či NOR (\downarrow)
- **Literál:** Výroková formule, která je prvotní formulí, nebo negací prvotní formule.
- **Implikant:** Literál, či konjunkce několika literálů.
- **Výroková formule v disjunktivním normálním tvaru (DNT),** pokud je implikantem, či disjunkcí několika implikantů.
- **Klausule:** Literál, či disjunkce několika literálů.

- **Výroková formule v konjunktivním normálním tvaru (KNT)**, pokud je klausulí, či konjunkcí několika klausulí.
- Každá výroková formule lze převést do logicky ekvivalentního KNT i DNT.
- **Minterm**: minterm formule F je takový její implikant, který obsahuje všechny prvotní formule vyskytující se v F a každou právě jednou.
- **Výroková formule v úplném disjunktivním normálním tvaru (ÚDNT)**, je-li mintermem nebo disjunkcí různých (logicky neekvivalentních) mintermů.
- **Maxterm**: minterm formule F je taková její klausule, která obsahuje všechny prvotní formule vyskytující se v F a každou právě jednou.
- **Výroková formule v úplném konjunktivním normálním tvaru (ÚDNT)**, je-li maxtermem nebo konjunkcí různých (logicky neekvivalentních) maxtermů.
- Každá výroková formule lze převést do logicky ekvivalentního ÚKNT i ÚDNT.

1.8 SP-8 (DML)

Základy teorie čísel: dělitelnost, REA a diofantické rovnice, prvočísla, modulární aritmetika, Malá Fermatova a Eulerova věta, lineární kongruence, Čínská věta o zbytcích.

- **Dělitelnost:** Nechť $a, b \in \mathbb{Z}$. Řekneme, že a dělí b , značíme $a | b$, jestliže existuje $k \in \mathbb{Z}$ takové, že $a \cdot k = b$. V takovém případě říkáme, že a je (celočíselný) dělitel b a b je (celočíselný) násobek a , případně také, že b je dělitelné a . Pokud a nedělí b , píšeme $a \nmid b$. Samotné $|$ nazýváme relací dělitelnosti.
- **Dělení se zbytkem:** Nechť $a \in \mathbb{Z}, d \in \mathbb{N}$. Pak existují jednoznačně určená čísla $q, r \in \mathbb{Z}$ taková, že $a = qd + r \wedge 0 \leq r < d$. Číslo q nazýváme celočíselný podíl (po dělení a číslem d). Číslo $r \in \{0, 1, \dots, d-1\}$ nazveme zbytkem po (celočíselném) dělení a číslem d a značíme jej $r = a \bmod d$.
- **Společný dělitel:** Nechť $a, b \in \mathbb{Z}$. Číslo $n \in \mathbb{N}_0$ je společný dělitel čísel a, b , jestliže $n | a \wedge n | b$.
- **Největší společný dělitel:** Nechť $a, b \in \mathbb{Z}$. Číslo $n \in \mathbb{N}_0$ je největší společný dělitel čísel a, b (značíme $n = \gcd(a, b)$), pokud je jejich společný dělitel a současně je (celočíselným) násobkem každého jejich dalšího společného dělitele, tedy pokud $(n | a \wedge n | b \wedge (\forall d \in \mathbb{N}_0)((d | a \wedge d | b) \Rightarrow d | n))$.
- **Soudělnost:** Nechť $a, b \in \mathbb{Z}$. Nazveme je nesoudělná, pokud $\gcd(a, b) = 1$. Pokud $\gcd(a, b) > 1$, nazveme tato čísla soudělná.
- **Společný násobek:** Nechť $a, b \in \mathbb{Z}$. Číslo $n \in \mathbb{N}_0$ je společný násobek čísel a, b , jestliže $a | n \wedge b | n$.
- **Nejmenší společný násobek:** Nechť $a, b \in \mathbb{Z}$. Číslo $n \in \mathbb{N}_0$ je nejmenší společný násobek čísel a, b (značíme $n = \text{lcm}(a, b)$), pokud je jejich společný násobek a současně dělí každý dalsí jejich společný násobek, tedy pokud $(a | n \wedge b | n \wedge (\forall m \in \mathbb{N}_0)((a | m \wedge b | m) \Rightarrow n | m))$.
- **REA — Rozšířený Euklidův Algoritmus**

$$\begin{array}{rcl}
 254 & = & 1 \cdot 254 + 0 \cdot 158 \\
 158 & = & 0 \cdot 254 + 1 \cdot 158 \quad (\lfloor \frac{254}{158} \rfloor = 1) \\
 \hline
 254 - 1 \cdot 158 = 96 & = & 1 \cdot 254 + (-1) \cdot 158 \quad (\lfloor \frac{158}{96} \rfloor = 1) \\
 158 - 1 \cdot 96 = 62 & = & (-1) \cdot 254 + 2 \cdot 158 \quad (\lfloor \frac{96}{62} \rfloor = 1) \\
 96 - 1 \cdot 62 = 34 & = & 2 \cdot 254 + (-3) \cdot 158 \quad (\lfloor \frac{62}{34} \rfloor = 1) \\
 62 - 1 \cdot 34 = 28 & = & (-3) \cdot 254 + 5 \cdot 158 \quad (\lfloor \frac{34}{28} \rfloor = 1) \\
 34 - 1 \cdot 28 = 6 & = & 5 \cdot 254 + (-8) \cdot 158 \quad (\lfloor \frac{28}{6} \rfloor = 4) \\
 28 - 4 \cdot 6 = 4 & = & (-23) \cdot 254 + 37 \cdot 158 \quad (\lfloor \frac{6}{4} \rfloor = 1) \\
 6 - 1 \cdot 4 = 2 & = & 28 \cdot 254 + (-45) \cdot 158 \quad (\lfloor \frac{4}{2} \rfloor = 2) \\
 \hline
 4 - 2 \cdot 2 = 0 & &
 \end{array}$$

- **LDR — Lineární diofantická rovnice:** libovolná rovnice typu $ax + by = c$, kde $a, b, c \in \mathbb{Z}$ pro 2 neznámé $x, y \in \mathbb{Z}$.

– LDR $ax + by = c$ má alespoň jedno řešení právě tehdy když $\gcd(a, b) | c$.

Nechť $a, b \in \mathbb{Z} \setminus \{0\}$ a dvojice $(x_0, y_0) \in \mathbb{Z}^2$ je řešením rovnice $ax + by = c$. Potom množina všech celočíselných řešení této rovnice je

$$\left\{ \left(x_0 + \frac{b}{\gcd(a, b)} \cdot k, y_0 - \frac{a}{\gcd(a, b)} \cdot k \right) \mid k \in \mathbb{Z} \right\},$$

což lze ekvivalentně zapsat také jako

$$\left\{ (x_0, y_0) + k \cdot \left(\frac{b}{\gcd(a, b)}, \frac{-a}{\gcd(a, b)} \right) \mid k \in \mathbb{Z} \right\}.$$

- Přirozená čísla \mathbb{N} dělíme podle počtu dělitelů do následujících 3 kategorií:

- číslo 1 (1 dělitel — 1)
- **prvočísla** — 2 dělitelé, samo číslo a 1
- **složená čísla** — 3 a více dělitelů

- **Kongruence modulo m :** Nechť $a, b \in \mathbb{Z}, m \in \mathbb{N}$. Pokud $m \mid (a - b)$, říkáme že a je kongruentní s b modulo m a píšeme $a \equiv b \pmod{m}$. V opačném případě a není kongruentní s b modulo m a píšeme $a \not\equiv b \pmod{m}$.
- **Množina zbytků:** Nechť $m \geq 2$. Jako \mathbb{Z}_m označíme množinu všech zbytků modulo m , $\mathbb{Z}_m = \{0, 1, 2, \dots, m - 1\}$. Operace sčítání: součet a modulo. Násobení totéž.
- **Inverze v \mathbb{Z}_m :** Aditivní inverze (opačný prvek) — součet modulo je 0. Multiplikativní inverze (inverzní prvek) — násobek modulo je 1.
- **Existence multiplikativní inverze:** Nechť $m \geq 2$ a $a \in \mathbb{Z}_m$. V \mathbb{Z}_m existuje multiplikativní inverze k a právě tehdy, když $\gcd(a, m) = 1$. Pokud existuje, je jediná.
- **Krácení v modulu:** Nechť $a, b, c \in \mathbb{Z}$ a $m \in \mathbb{N}, m \geq 2$, označme $d = \gcd(m, c)$. Pak platí ekvivalence: $ac \equiv bc \pmod{m} \Leftrightarrow a \equiv b \pmod{\frac{m}{d}}$
- **Malá Fermatova věta:** Budě p prvočíslo a $a \in \mathbb{N}$ takové přirozené číslo, které není násobkem p (tedy $\gcd(a, p) = 1$). Potom platí kongruence $a^{p-1} \equiv 1 \pmod{p}$.
- Pokud je p prvočíslo a $a \in \mathbb{Z}_p \wedge a \neq 0$, pak a^{p-2} je multiplikativní inverzí čísla a mod p .
- **Eulerova funkce:** $\varphi : \mathbb{N} \rightarrow \mathbb{N}$ je zobrazení, které každému $n \in \mathbb{N}$ přiřafí počet přirozených čísel menších nebo rovných n , která jsou s n nesoudělná. Tedy ($\forall n \in \mathbb{N} (\varphi(n) := |\{k \in \mathbb{N} \mid k \leq n \wedge \gcd(k, n) = 1\}|)$).
- **Eulerova věta:** Nechť $m \in \mathbb{N}, m \geq 2$ a $a \in \mathbb{N}$ je číslo nesoudělné s m . potom platí kongruence $a^{\varphi(m)} \equiv 1 \pmod{m}$.
 - p je prvočíslo, tedy $\varphi(p) = p - 1$
 - p je prvočíslo, $\alpha \in \mathbb{N}$, pak $\varphi(p^\alpha) = p^\alpha - p^{\alpha-1}$
 - $m, n \in \mathbb{N}, \gcd(m, n) = 1$. Potom $\varphi(m \cdot n) = \varphi(m) \cdot \varphi(n)$.
- Nechť $m \in \mathbb{N}, m \geq 2$ a $a \in \mathbb{Z}_{>}$ je číslo nesoudělné s m . potom $a^{\varphi(m)-1}$ je multiplikativní inverzí čísla a mod m .
- **Lineární kongruence:** řešením lineární kongruence rozumíme nalezení všech celých čísel x splňujících kongruenci $ax \equiv b \pmod{m}$, kde $a, b, m \in \mathbb{Z}$ a $m \geq 2$.
- **ČVOZ — Čínská věta o zbytcích**

Uvažujme soustavu lineárních kongruencí

$$\begin{aligned} x &\equiv a_1 \pmod{m_1}, \\ x &\equiv a_2 \pmod{m_2}, \\ &\vdots \\ x &\equiv a_N \pmod{m_N}, \end{aligned}$$

kde $m_1, m_2, \dots, m_N \geq 2$ jsou navzájem nesoudělná, tedy $\gcd(m_i, m_j) = 1$ pro každá $i \neq j$.

Řešení této soustavy vždy existuje a všechna řešení jsou kongruentní modulo M (tedy v \mathbb{Z}_M je řešení určeno jednoznačně), kde

$$M = \prod_{i=1}^N m_i.$$

- Výsledek je: $x = a_1 \cdot M_1 \cdot X_1 + \dots + a_N \cdot M_N \cdot X_N \pmod{M}$

Vyřešíme soustavu

$$\begin{aligned}x &\equiv 1 \pmod{2}, \\x &\equiv 2 \pmod{3}, \\x &\equiv 3 \pmod{5}.\end{aligned}$$

$M = 2 \cdot 3 \cdot 5 = 30$. Předpoklady ČVOZ jsou splněny, modula jsou dokonce různá prvočísla. Tedy řešení soustavy existuje a to jednoznačné v \mathbb{Z}_{30} .

Dále určíme

$$M_1 = 3 \cdot 5 = 15, \quad M_2 = 2 \cdot 5 = 10, \quad M_3 = 2 \cdot 3 = 6.$$

Nalezneme „inverze“ k M_i :

- $M_1 X_1 = 15 X_1 \equiv 1 \pmod{2}$ má za řešení $X_1 = 1$,
- $M_2 X_2 = 10 X_2 \equiv 1 \pmod{3}$ má za řešení $X_2 = 1$,
- $M_3 X_3 = 6 X_3 \equiv 1 \pmod{5}$ má za řešení $X_3 = 1$.

Konečně,

$$x = \underbrace{1 \cdot 1 \cdot 15}_{a_1 X_1 M_1} + \underbrace{2 \cdot 1 \cdot 10}_{a_2 X_2 M_2} + \underbrace{3 \cdot 1 \cdot 6}_{a_3 X_3 M_3} = 53 \equiv 23 \pmod{30}.$$

• ZČVOZ — Zobecněná čínská věta o zbytcích

Uvažujme soustavu lineárních kongruencí

$$\begin{aligned}x &\equiv a_1 \pmod{m_1}, \\x &\equiv a_2 \pmod{m_2}, \\&\vdots \\x &\equiv a_N \pmod{m_N},\end{aligned}$$

kde $m_1, m_2, \dots, m_N \geq 2$. Tato soustava má řešení právě tehdy, když $\gcd(m_i, m_j)$ dělí $a_i - a_j$ pro všechna $i \neq j$. Pokud řešení existuje, je určeno jednoznačně modulo $\text{lcm}(m_1, m_2, \dots, m_N)$.

1.9 SP-9 (KAB)

Asymetrické kryptosystémy (šifra RSA, Diffie-Hellman, RSA digitální podpis), hešovací funkce (SHA-2, HMAC). Digitální podpis. Certifikáty, certifikační autority.

Asymetrické šifry

- pro šifrování a dešifrování se používají různé klíče
- pomocí veřejného klíče šifrujeme, pomocí privátního klíče dešifrujeme
- privátní klíč nelze odvodit z veřejného klíče v rozumném čase
- každý subjekt má svůj vlastní pár veřejný-privátní klíč

princip RSA

- šifrovací systém založený na modulárním umocňování
- dvojice (e, n) je veřejný klíč (e — veřejný exponent, n — modul)
- n je součinem dvou velkých prvočísel p a q , tedy $n = pq$ a $\gcd(e, \varphi(n)) = 1$
- zašifrování plaintextu — písmena se převedou na numerické ekvivalenty, vytvoří se bloky s největší možnou velikostí — $E(m) = c = |m^e|_n$, $0 < c < n$ (m je vytvořený blok plaintextu, c je výsledný blok ciphertextu)
- k dešifrování je nutná znalost inverze d čísla e modulo $\varphi(n)$ — $\gcd(e, \varphi(n)) = 1$, inverze tedy existuje — $D(c) = |c^d|_n = |m^{ed}|_n$, e a d jsou inverzní modulo $\varphi(n)$, tedy $ed \equiv 1 \pmod{\varphi(n)}$, tedy $ed = 1 + k \cdot \varphi(n)$, z toho $\Rightarrow |m^{ed}|_n = |m^{1+k \cdot \varphi(n)}|_n = |m \cdot (m^{\varphi(n)})^k|_n$, a dle Eulerovy věty $m^{\varphi(n)} \equiv 1 \pmod{n}$, tedy celkově $D(c) = |m|_n$
- dvojice (d, n) tvoří soukromý klíč
- existuje možnost, že m a n jsou soudělné, ale je extrémně malá

Generování RSA klíčů

- jak hledat p a q ?
- subjekt nalezne 2 velká náhodná čísla p a q s 340 dekadickými číslicemi
- z věty o prvočíslech plyne, že pravděpodobnost toho, že takto vybraná čísla jsou prvočísla, je cca $2 / \log(10^3 40)$
- pro nalezení prvočísla je potřeba v průměru $1 / (2 / \log(10^3 40)) \approx 400$ testů takových čísel
- exponent e by následně měl být zvolen jako číslo větší než p a q — 2^e by mělo být větší než n , aby šifrování i dešifrování muselo používat redukci modulo n a nešlo pouze odmocnit

Bezpečnost RSA

- modulární umocňování potřebné k šifrování zprávy s použitím RSA může být provedeno při velikosti modulu a bloku cca 680 dekadických číslic v řádech milisekund
- dešifrovací exponent d nelze z dvojice (e, n) snadno odvodit, protože je potřeba znát hodnota $\varphi(n)$ — pro rychlé spočítání je třeba znát faktorizaci $n = pq$
- i v případě cca 100-číslicových p a q (tedy 200-číslicového modulu n) trvá nejrychlejším známým algoritmem faktorizace kolem 250 let
- náročnost prolomení je tím větší, čím větší je modul
- ochrana proti speciálním rychlým technikám faktorizace n : např. obě hodnoty $p - 1$ a $q - 1$ by měly mít velký prvočíselný faktor, tedy malé $\gcd(p - 1, q - 1)$ a rozdíl $p - q$ by měl být dostatečně velký

Digitální podpis s použitím RSA

- uvažujme 2 subjekty, každý má svůj pár privátní-veřejný klíč $(PK_1, VK_1), (PK_2, VK_2)$
- subjekt 1 chce poslat podepsanou zprávu

- subjekt 1 ji podepíše: $S = D_{PK_1}(m) = |m^{d_1}|_{n_1}$
- subjekt 1 zašifruje pro subjekt 2: $c = E_{VK_2}(S) = |S^{e_2}|_{n_2}$ (pokud $n_2 > n_1$, jinak je nutné rozdělit S do bloků o velikosti menší než n_2 před šifrováním)
- subjekt 2 nejprve dešifruje svým privátním klíčem, následně použije šifrovací transformaci veřejným klíčem subjektu 1 pro odkrytí původního obsahu
- subjekt 2 si je tedy jistý, že zpráva přišla od subjektu 1, a zároveň subjekt 1 nemůže poprít, že danou zprávu poslal

Urychlení RSA výpočtu

- urychlení šifrování — doporučená množina exponentů e s nízkou Hammingovou váhou
- urychlení dešifrování — použití Čínské věty o zbytcích (RSA-CRT) (rozklad čísel, počítá se s čísly poloviční délky)

Diffie-Hellmann

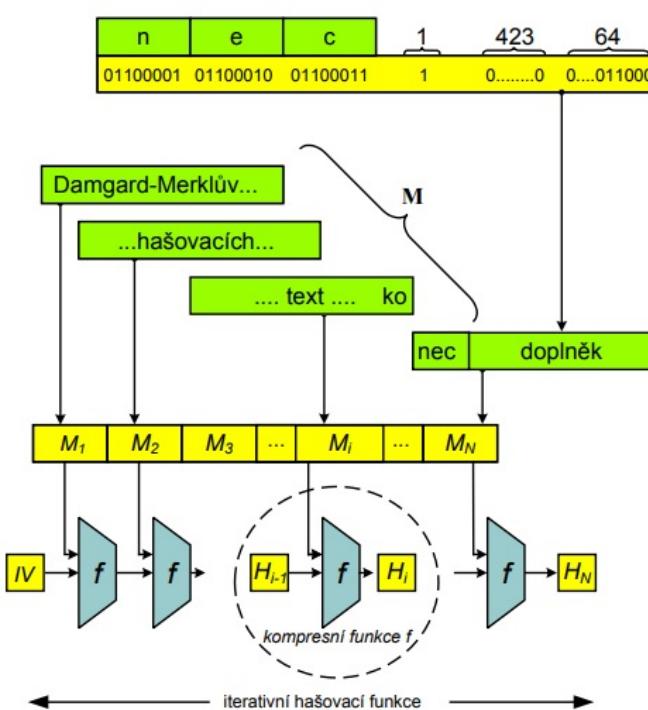
- algoritmus pro ustanovení společného klíče
- subjekt A a B si veřejně dohodnou prvočíslo m a bázi a ($1 < a < m$), přesněji grupu řádu $m - 1$
- subjekt A si náhodně zvolí číslo k_A takové, že $0 < k_A < m$ a $\gcd(k_A, m - 1) = 1$, spočítá $y_A = |a^{k_A}|_m$ a odešle ho B
- subjekt B si náhodně zvolí k_B takové, že $0 < k_B < m$ a $\gcd(k_B, m - 1) = 1$, spočítá $y_B = |a^{k_B}|_m$ a odešle ho A
- A i B spočítají sdílený klíč $K = |y_B|_m = |(a^{k_B})^{k_A}|_m = |a^{k_A \cdot k_B}|_m = |(a^{k_A})^{k_B}|_m = |y_A|_m$
- útočník nemůže z $|a^{k_A}|_m$ ani z $|a^{k_B}|_m$ spočítat $K = |a^{k_A \cdot k_B}|_m$ — tzv. Diffie-Hellmanův problém, DHP
- není složitější než DLP (problém diskrétního logaritmu), a zdá se že ale není ani jednodušší

El Gamal

- vzniká úpravou Diffie-Hellman
- A zvolí číslo g a prvočíslo m , $1 < g < m$, a g je generátor odpovídající grupy řádu $m - 1$
- A si náhodně zvolí soukromý klíč k_A tak, že $0 < k_A < m$, spočítá $y_A = |g^{k_A}|_m$
- A zveřejní uspořádanou trojici (m, q, y_A) jako veřejný klíč, k_A je soukromým klíčem
- B chce odeslat A zprávu p
- B si náhodně zvolí číslo k_B ($0 < k_B < m$), spočítá $y_B = |g_B^k|_m$
- B spočítá sdílený klíč $K = |y_A|_m = |g^{k_A \cdot k_B}|_m$
- B zašifruje zprávu p pomocí vztahu $c = |p \cdot K|_m$
- B odešle A uspořádanou dvojici (y_B, c)
- A si spočítá $K = |y_B|_m = |g^{k_A \cdot k_B}|_m$
- A si spočítá $|k^{-1}|_m$ (přes REA)
- A dešifruje zprávu: $p = |c \cdot K^{-1}|_m = |p \cdot K \cdot K^{-1}|_m = |p|_m$

Hash funkce

- základní pojmy: jednosměrnost, bezkoliznost
- **Jednosměrnost:** $f : X \rightarrow Y$, je snadné z jakékoli hodnoty $x \in X$ vypočítat $y = f(x)$, ale je výpočetně nemožné pro náhodný obraz $y \in f(X)$ najít vzor $x \in X$, aby $y = f(x)$
- **Hashovací funkce** je jednosměrná (1. typu — neexistují zadní vrátka pro zpětný výpočet) a bezkolizní, zároveň pro různé vstupy vrací stejně dlouhý výstup (hash)
- **Orákulum:** libovolná stroj, který na základě vstupu odpoví nějakým výstupem, na stejný vstup odpovídá stejným výstupem
- **Náhodné orákulum:** na nový vstup odpovídá náhodným výběrem z množiny vstupů
- Hashovací funkce se má chovat jako náhodné orákulum (bezpečnostní vlastnosti)
- **Bezkoliznost 1. řádu:** Hash fce h je odolná proti kolizím 1. řádu, jestliže je výpočetně nezvládnutelné nalezení libovolných dvou různých (byť nesmyslných) zpráv M a M' tak, že $h(M) = h(M')$
- bezkoliznost se využívá k digitálním podpisům
- nepodepisuje se zpráva (moc dlouhá) ale její hash
- bezkoliznost zaručuje, že je složité nalézt 2 dokumenty se stejným hashem — proto lze podepisovat hash
- **Bezkoliznost 2. řádu:** Hash funkce h je odolná proti kolizi 2. řádu, jestliže pro jakýkoliv vzor x je výpočetně nezvládnutelné nalézt 2. vzor $y \neq x$ tak, že $h(x) = h(y)$
- **Odolnost pro nalezení kolize 1. řádu:** pokud se hash funkce s hashem délky n bude chovat jako náhodné orákulum, složitost nalezení kolize s 50 % pravděpodobností je $\approx 2^{\frac{n}{2}}$ (narozeninový paradox)
- **Odolnost proti nalezení kolize 2. řádu:** pokud se hash funkce s hashem délky n bude chovat jako náhodné orákulum, složitost nalezení 2. vzoru je $\approx 2^n$
- pokud pro konkrétní hash funkci lze nalézat vzory/kolize rychleji, hovoříme o prolomení hashovací funkce
- při hashování se typicky zpráva rozdělí na bloky, poslední blok se doplní (např. jednou 1 a pak samé 0, aby otisky zpráv co by na konci měly např. jen víc nul byly jiné)
- pro konkrétní bloky se využívá kompresní funkce f , která z předchozího kontextu H_{i-1} a bloku M_i vytvoří kontext H_i
- při konstrukci hash funkce dle následujícího obrázku bezkoliznost kompresní funkce zaručuje bezkoliznost celé hashovací funkce



- jako kompresní funkce se typicky používá bloková šifra — kontext H_{i-1} je vstup, blok M_i je klíč
- dle Davies-Meyerovy konstrukce se ještě po zašifrování přičte (XOR) původní kontext

HMAC

- nepadělatelný integrální kód zprávy
- pro zprávu se spočítá za použití tajného klíče K
- detekuje chyby při přenosu, zabraňuje neoprávněné změně zprávy

Algoritmus HMAC

- Definujeme konstantní řetězce $ipad$ jako řetězec $b/8$ bajtů s hodnotou $0x36$ (0011 0110) a $opad$ jako řetězec $b/8$ bajtů s hodnotou $0x5C$ (0101 1100), kde b je velikost bloku v bitech.
- Klíč K v případě, že $\log_2 K < b$, doplníme byty 0 vlevo od MSB bitu klíče do délky b -bitů a označíme ho K^+ .
- Definujeme hodnotu $HMAC_K(M)$ jako

$$HMAC_K(M) = H((K^+ \oplus opad) \parallel H((K^+ \oplus ipad) \parallel M)),$$

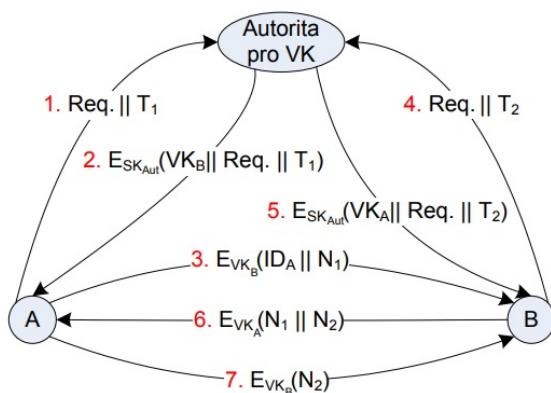
kde \parallel označuje zřetězení.

Public Key Infrastructure

- jak distribuovat tajné symetrické klíče? distribuujeme veřejné klíče, následně si tajné klíče předáme za použití asymetrického šifrování
- jak distribuovat veřejné klíče? jak zabránit možnosti podvrhnutí?
- **Zveřejnění veřejných klíčů**
 - zasílání veřejných klíčů přímo
 - rychlé a jednoduché
 - není odolné proti podvržení
- **Veřejně dostupný adresář**
 - vyšší úroveň bezpečnosti
 - distribuci zajišťuje důvěryhodná autorita/správce



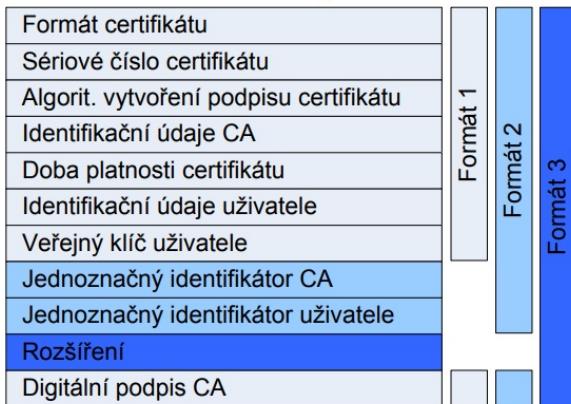
- **Autorita pro veřejné klíče**
 - přísnější dohled na distribuci veřejného klíče z adresáře
 - autorita má svůj pár veřejný-privátní klíč, každý účastník musí znát její veřejný klíč



- Certifikace veřejných klíčů

- distribuce veřejného klíče bez kontaktu se třetí stranou
- Certifikát: struktura obsahující veřejný klíč držitele, identifikační údaje držitele, dobu platnosti, další údaje vytvořené CA a zejména podpis CA
- Certifikační Autorita (CA): důvěryhodná třetí strana, která na základě žádosti vydává a aktualizuje certifikáty — certifikáty vytvořené CA lze ověřit jejím veřejným klíčem

X.509 - formáty certifikátů



- certifikáty mohou být podepsané ve stromové struktuře — certifikát držitele podepsaný CA₁, její certifikát podepsaný CA₂...
- kořenové certifikáty musí být distribuovány jinak (typicky např. s operačním systémem)

1.10 SP-10 (KAB)

Symetrické šifry blokové a proudové, základní parametry, operační módy blokových šifer, jejich základní popis a slabiny.

Symetrické šifry používají pro operaci šifrování a dešifrování stejný klíč (případně snadno převeditelný).

Proudové šifry

- proudová šifra zpracovává každý znak zvlášť
- na každý znak je použita jiná šifrovací transformace E_{h_i} , která je posunem v abecedě o h_i znaků
- posloupnosti h_1, h_2, \dots se říká key-stream, a je vygenerována z klíče
- synchronní šifry: key-stream je závislý pouze na klíci (vypadne znak — zbytek textu už nerozšifrujeme)
- asynchronní šifry: key-stream je závislý jak na klíci, tak na předchozích zpracovaných datech (takže na OT a/nebo ŠT) (vypadne znak — po n znacích jsme schopni zase dešifrovat správně)
- příklad šifer: RC4, Salsa20, ChaCha, A5/1

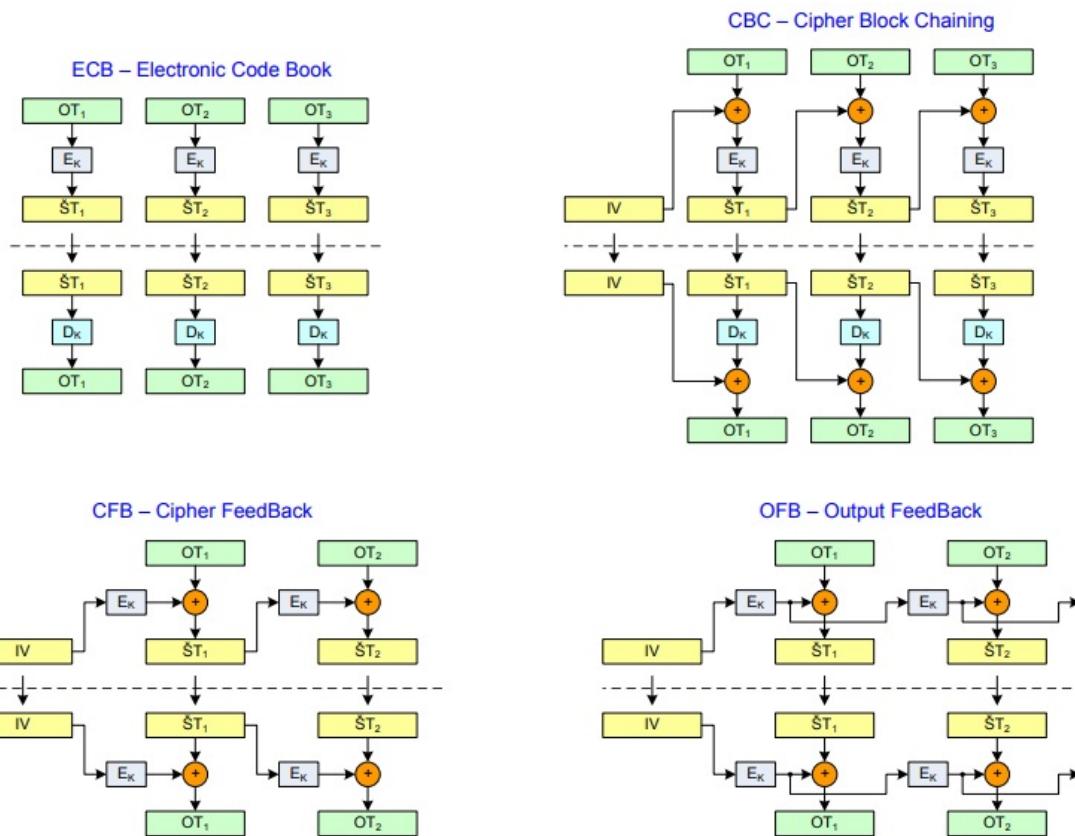
Blokové šifry

- šifrují t znaků najednou (bloky délky t)
- všechny bloky jsou šifrovány tou samou transformací
- DES, 3DES — 64b blok, DES 56b klíč, 3DES 112b/168b klíč
- AES — 128b blok, 128b/192b/256b klíč
- šifra Feistelova typu — postupná aplikace relativně jednoduchých operací, vytvoří poměrně složitý algoritmus
- iterativní bloková šifra — základem je jednoduchá funkce provádějící šifrování jednoho bloku, která je několikrát zopakována na tom samém bloku — jedna iterace se jmenuje 1 runda
- DES — 16 rund, z 56b klíče se expandují rundovní klíče, v praxi nevhoda krátkého klíče
- 3DES — kombinace 3 operací DES za sebou, s použitím 2 nebo 3 různých DES klíčů ($E_{K_1}, D_{K_2}, E_{K_{1/3}}$)
- AES — 10/12/14 rund v závislosti na délce klíče — není Feistelova typu, nemá slabé klíče, odolná proti různým útokům
 - stav = 1 blok (16B) v matici 4x4
 - Expanze klíče — z klíče se odvodí rundovní klíče
 - AddRoundKey (xor rundovního klíče se stavem)
 - Iterace/runda: SubBytes (náhrada bytů dle tabulky), ShiftRows (posunutí bytů cyklicky v řádcích), MixColumns (vynásobení matice maticí), AddRoundKey
 - v poslední rundě je vynechána MixColumns

Operační módy

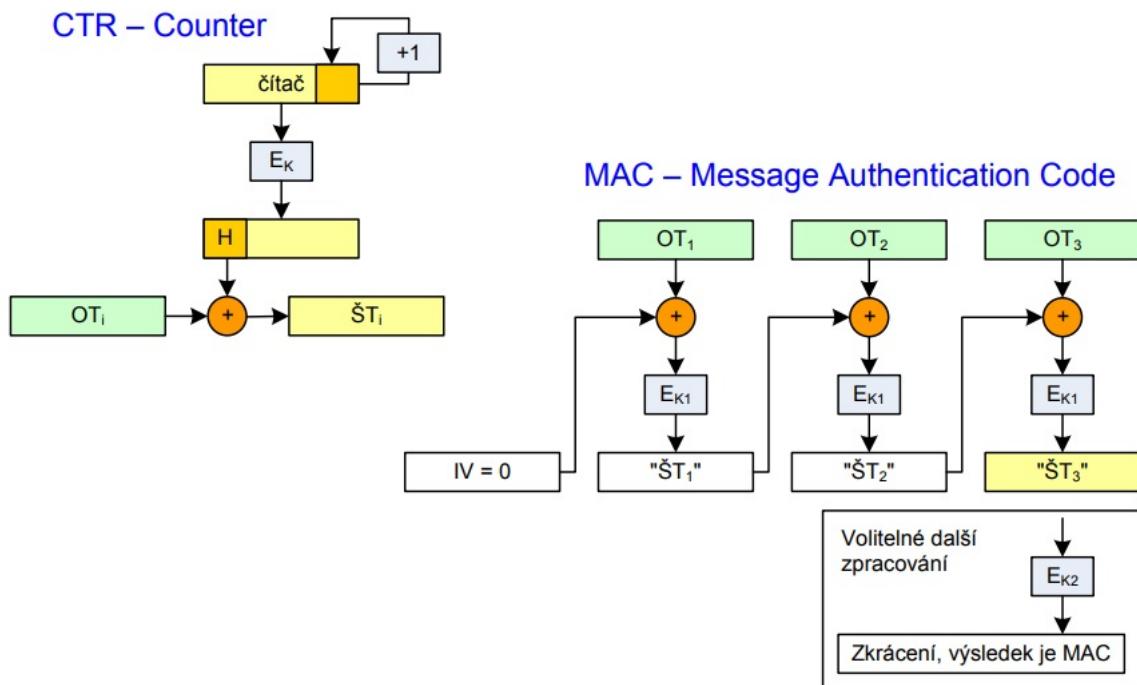
- operační módy blokových šifer nám mohou zlepšit vlastnosti blokových šifer
- ECB (Electronic Codebook)
 - bloky jsou šifrovány i dešifrovány nezávisle na sobě
 - stejný blok OT má stejný obraz v ŠT
 - lze zasahovat do dat — odebírat bloky a přehazovat
 - 1 bit chyby v ŠT poškodí celý blok OT
- CBC (Cipher Block Chaining)
 - je potřeba IV
 - každý blok OT se nejprve seče (xor) s předchozím blokem ŠT (první IV) a následně zašifruje

- 1 bit chyby v ŠT poškodí celý aktuální blok OT a 1 bit z následujícího
- CFB (Cipher Feedback)
 - je potřeba IV
 - dělá z blokové šifry proudovou (asynchronní/samosynchronní)
 - každý blok OT se sečte (xor) s přechozím blokem ŠT, který byl nejprve zašifrován klíčem (IV pro první blok)
 - 1 bit chyby v ŠT poškodí celý příští blok OT a 1 bit z aktuálního
- OFB (Output Feedback)
 - je potřeba IV
 - dělá z blokové šifry proudovou (synchronní)
 - každý blok OT se sečte (xor) s aktuálním heslem z key-streamu, key-stream je nezávislý na OT i ŠT, postupně se generuje z IV operací šifrování
 - 1 bit chyby v ŠT poškodí 1 bit v aktuálním bloku OT



- CTR (čítačový mód)
 - dělá z blokové šifry proudovou (synchronní)
 - každý blok OT se sečte (xor) s aktuálním heslem z key-streamu, key-stream je nezávislý na OT i ŠT, postupně se generuje použitím čítače (načte se IV, poté se něco přičítá)
 - 1 bit chyby v ŠT poškodí 1 bit v aktuálním bloku OT
 - má zaručit maximální periodu hesla
 - v žádných zprávách šifrovaných tímtož klíčem nesmí dojít k vygenerování stejného bloku hesla vícekrát — obsah čítače nesmí být stejný
- MAC (message authentication code)
 - proudové i blokové šifry zajišťují důvěrnost, ne integritu zpráv
 - MAC zajišťuje integritu a původ zprávy

- použije se jiný klíč než k šifrování
- funguje jako CBC s nulovým IV, průběžný ŠT se neodesílá
- MAC je tvořen posledním blokem ŠT_n



1.11 SP-11 (LA1)

Soustavy lineárních rovnic: Frobeniova věta a související pojmy, vlastnosti a popis množiny řešení, Gaussova eliminační metoda.

Základní pojmy

- **Grupa** — Nechť M je neprázdná množina a $\circ : M \times M \rightarrow M$ binární operace. Platí li:
 - **asociativní zákon:** $(\forall a, b, c \in M)(a \circ (b \circ c) = (a \circ b) \circ c)$
 - existence **neutrálního prvku:** existuje $e \in M$ tak, že $(\forall a \in M)(a \circ e = e \circ a = a)$
 - existence **inverzních prvků:** $(\forall a \in M)(\exists a^{-1} \in M)(a \circ a^{-1} = a^{-1} \circ a = e)$
- Říkáme, že uspořádaná dvojice $G = (M, \circ)$ je **grupa**.
- Pokud navíc platí **komutativní zákon:** $(\forall a, b \in M)(a \circ b = b \circ a)$, mluvíme o **abelovské grupě**.
- **Těleso** — Nechť M je neprázdná množina a $+ : M \times M \rightarrow M$, $\cdot : M \times M \rightarrow M$ dvě binární operace. Platí-li, že
 - $(M, +)$ je *abelovská grupa* (neutrální prvek 0 — nulový prvek)
 - $(M \setminus \{0\}, \cdot)$ je grupa (neutrální prvek značíme 1 — jednotkový prvek)
 - platí levý a pravý **distributivní zákon:** $(\forall a, b, c \in M)(a(b + c) = ab + ac \wedge (b + c)a = ba + ca)$
- nazýváme uspořádanou trojici $T = (M, +, \cdot)$ tělesem.
- Je-li navíc $(M \setminus \{0\}, \cdot)$ abelovská grupa, je T **komutativní těleso**.

Maticový zápis SLR

- nechť $m, n \in \mathbb{N}$, $\mathbf{A} \in T^{m,n}$, $\mathbf{b} \in T^m$
- rovnici $\mathbf{Ax} = \mathbf{b}$ nazýváme soustavou m lineárních rovnic pro n neznámých x_1, x_2, \dots, x_n
- vektor $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ nazýváme **vektorem neznámých**
- vektor $\mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$ nazýváme vektorem **pravých stran**
- matici \mathbf{A} nazýváme **maticí soustavy**
- matici $(\mathbf{A}|\mathbf{b})$ **rozšířenou maticí soustavy**
- je-li $\mathbf{b} = \theta \in T^m$, mluvíme o **homogenní soustavě**
- soustavu $\mathbf{Ax} = \theta$ nazýváme **přidruženou homogenní soustavou lineárních rovnic** k soustavě $\mathbf{Ax} = \mathbf{b}$
- množinu všech řešení soustavy $\mathbf{Ax} = \mathbf{b}$ značíme S a množinu řešení přidružené homogenní soustavy S_0

Horní stupňovitý tvar

- o matici $\mathbf{D} \in T^{m,n}$ řekneme, že je v **horním stupňovitém tvaru**, jestliže jsou všechny řádky nulové ($\mathbf{D} = \Theta$), nebo jsou splněny obě následující podmínky:
 - případné nulové řádky jsou pouze v dolní části matice: existuje $k \in \hat{m}$ tak, že řádky 1 až k matice \mathbf{D} jsou nenulové a ostatní řádky jsou nulové
 - v nenulových řádcích je vždy první nenulový prvek až za prvním nenulovým prvkem z předchozího řádku: máme-li k z předchozího bodu a označíme-li pro každé $i \in \hat{k}$ index nejlevějšího nenulového prvku v i -tému řádku jako j_i , tj. $j_i = \min\{l \in \hat{n} | \mathbf{D}_{il} \neq 0\}$ — potom platí $j_1 < j_2 < \dots < j_k$

- je-li matice v HST, potom sloupcům , ve kterých se vyskytuje první nenulový prvek řádku, říkáme **hlavní sloupce**, ostatním říkáme **vedlejší sloupce**
- soustava $\mathbf{Ax} = \mathbf{b}$ je v HST, pokud je v HST rozšířená matice soustavy $(\mathbf{A}|\mathbf{b})$

Operace GEM

Pro matici $\mathbf{A} \in T^{m,n}$ s prvky a_{ij} definujeme tyto operace:

- **(G1)** — prohození dvou řádků
- **(G2)** — vynásobení jednoho řádku nenulovým číslem
- **(G3)** — přičtení libovolného násobku jednoho řádku k jinému řádku

Vektorový prostor

Nechť T je libovolné těleso a $n \in \mathbb{N}$. Množinu n -tic $\{(x_1, \dots, x_n) | x_i \in T \text{ pro každé } i \in \hat{n}\}$ spolu s operacemi $+, \cdot$ definovaných po složkách takto: pro každé $\alpha \in T$ a pro každé $\mathbf{x}, \mathbf{y} \in T^n$ položíme

- $\mathbf{x} + \mathbf{y} = (x_1, \dots, x_n) + (y_1, \dots, y_n) := (x_1 + y_1, \dots, x_n + y_n)$
- $\alpha \cdot \mathbf{x} = \alpha \cdot (x_1, \dots, x_n) := (\alpha x_1, \dots, \alpha x_n)$

nazýváme **vektorovým prostorem T^n**

Podprostor

- Nechť P je podmnožina T^n . Řekneme, že P je **podprostor** vektorového prostoru T^n , právě když platí:
 - množina P je neprázdná, tedy $P \neq \emptyset$
 - množina P je uzavřená vůči scítání vektorů v ní, tedy $(\forall \mathbf{x}, \mathbf{y} \in P)(\mathbf{x} + \mathbf{y} \in P)$
 - množina P je uzavřená vůči násobení vektorů v ní libovolným skalárem, tedy $(\forall \alpha \in T)(\forall \mathbf{x} \in P)(\alpha \mathbf{x} \in P)$
- vztah "být podprostorem" pak značíme $P \subset \subset T^n$
- podprostory $\{\theta\}$ a T^n vektorového prostoru T^n nazýváme **triviálními podprostory**
- každý podprostor $P \subset \subset T^n$, pro který současně platí $P \neq T^n$ nazýváme **vlastním podprostorem**

Lineární kombinace

- Nechť $\mathbf{x} \in T^n$ a $(\mathbf{x}_1, \dots, \mathbf{x}_m)$ je soubor vektorů z T^n . Říkáme, že vektor \mathbf{x} je lineární kombinací souboru $(\mathbf{x}_1, \dots, \mathbf{x}_m)$, právě když existují čísla $\alpha_1, \dots, \alpha_m \in T$ taková, že

$$\mathbf{x} = \sum_{i=1}^m \alpha_i \mathbf{x}_i$$

- čísla α_i . $i \in \hat{m}$ nazýváme koeficienty lineární kombinace
- jestliže jsou všechny koeficienty nulové, nazýváme takovou kombinaci triviální
- v opačném případě se jedná o netriviální lineární kombinaci

Další pojmy

- Lineární (ne)závislost
- Nechť $(\mathbf{x}_1, \dots, \mathbf{x}_m)$ je soubor vektorů z T^n . Řekneme, že soubor je **lineárně nezávislý (LN)**, právě když pouze triviální lineární kombinace tohoto souboru je rovna nulovému vektoru θ . V opačném případě je soubor nazýván **lineárně závislým (LZ)**.
- **lineární obal** souboru vektorů — množina všech lineárních kombinací, značí se $\langle \mathbf{x}_1, \dots, \mathbf{x}_m \rangle$
- soubor **generuje** podprostor P právě když jeho lineární obal je roven podprostoru
- LN soubor který generuje podprostor je jeho **báze**
- **dimenze podprostoru** — délka nejdélšího možného LN souboru z daného podprostoru

Hodnost matice

Nechť $\mathbf{A} \in T^{m,n}$. **Hodností matice \mathbf{A}** nazýváme dimenzi lieárního obalu souboru řádků matice \mathbf{A} a značíme ji $h(\mathbf{A})$.

$$h(\mathbf{A}) = \dim \langle (\mathbf{A}_{1:})^T, \dots, (\mathbf{A}_{m:})^T \rangle$$

Frobeniova věta

Nechť $A \in T^{m,n}$ a $b \in T^{m,1}$.

- soustava m lineárních rovnic pro n neznámých $\mathbf{Ax} = \mathbf{b}$ je řešitelná ($S \neq \emptyset$) právě tehdy, když

$$h(\mathbf{A}) = h(\mathbf{A}|b)$$

Pokud platí, pak $S = \tilde{\mathbf{x}} + S_0$, kde $\tilde{\mathbf{x}}$ je **partikulární řešení**, tj. $\mathbf{A}\tilde{\mathbf{x}} = \mathbf{b}$.

- množina řešení homogenní soustavy $\mathbf{Ax} = \theta$ je podprostor dimenze $n - h(\mathbf{A})$, neboli:

- pokud $h(\mathbf{A}) = n$, pak $S_0 = \{\theta\}$
- pokud $h(\mathbf{A}) < n$, pak existuje LN soubor $(\mathbf{z}_1, \dots, \mathbf{z}_{n-h})$ vektorů z T^n tak, že $S_0 = \langle \mathbf{z}_1, \dots, \mathbf{z}_{n-h} \rangle$

1.12 SP-12 (LA1)

Matice: součin matic, regulární matice, inverzní matice a její výpočet, vlastní čísla matice a jejich výpočet, diagonalizace matice.

- Jednotková matice

Jednotkovou maticí řádu n rozumíme čtvercovou matici $\mathbf{E} \in T^{n,n}$ splňující: E_{ij} je 1 pokud $j = i$, jinak je to 0.

- Diagonální matice

Diagonální maticí řádu n nazveme libovolnou čtvercovou matici $\mathbf{A} \in T^{n,n}$ splňující A_{ij} je 0 pokud $j \neq i$.

- Inverzní a regulární matice

Bud' $\mathbf{A} \in T^{n,n}$. Matici $\mathbf{B} \in T^{n,n}$ nazveme **inverzní maticí** k matici \mathbf{A} , pokud platí $\mathbf{AB} = \mathbf{BA} = \mathbf{E}$. Značíme $\mathbf{B} = \mathbf{A}^{-1}$. Matici \mathbf{A} nazveme **regulární**, pokud existuje matice, která je k ní inverzní. Pokud matice není regulární, je singulární.

Vlastní číslo a vektor

- číslo $\lambda \in \mathbb{C}$ nazýváme vlastním číslem matice $\mathbf{A} \in \mathbb{C}^{n,n}$, právě když existuje nenulový vektor $\mathbf{x} \in \mathbb{C}^n$ splňující $\mathbf{A} \cdot \mathbf{x} = \lambda \mathbf{x}$
- takový vektor nazýváme vlastním vektorem matice \mathbf{A} příslušejícím vlastnímu číslu λ
- spektrem matice je množina všech vlastních čísel matice (ozn. $\sigma(\mathbf{A})$)
- výpočet vlastních vektorů ke konkrétnímu vlastnímu číslu λ — řešení homogenní soustavy $(\mathbf{A} - \lambda \mathbf{E})$

Charakteristický polynom

Charakteristický polynom matice $\mathbf{A} \in \mathbb{C}^{n,n}$ ($p_{\mathbf{A}}$) je zobrazení definované jako: $p_{\mathbf{A}}(z) := \det(\mathbf{A} - z\mathbf{E})$, $z \in \mathbb{C}$

Vlastní čísla matice jsou kořeny tohoto polynomu.

- algebraická násobnost vlastního čísla — násobnost jako kořene charakteristického polynomu
- geometrická násobnost — dimenze vlastního podprostoru příslušejícího číslu

Podobnost matic

Matice \mathbf{A} a \mathbf{B} jsou podobné, pokud $\mathbf{A} = \mathbf{P}^{-1} \cdot \mathbf{B} \cdot \mathbf{P}$.

Diagonalizovatelnost matice

- (čtvercová) matice je diagonalizovatelná, pokud podobná nějaké diagonální matici
- matice je diagonalizovatelná, pokud součet geometrických násobností vlastních čísel je roven rozměru matice

1.13 SP-13 (MA1)

Limita posloupnosti, limita a spojitost reálné funkce jedné reálné proměnné, nástroje pro výpočet limit, asymptotické chování funkcí a posloupností (horní, dolní a těsné asymptotické meze, asymptotická ekvivalence).

Reálná čísla

Množinu reálných čísel \mathbb{R} chápeme jako číselné těleso $(\mathbb{R}, +, \cdot)$, které je vybavené úplným usporádáním $<$ a které splňuje axiom úplnosti.

Rozšířená reálná osa

Množinu reálných řísel spolu se symboly $+\infty$ a $-\infty$, tedy množinu $\mathbb{R} \cup \{+\infty, -\infty\}$, nazýváme rozšířenou množinou reálných čísel (případně rozšířenou reálnou osou) a značíme $\overline{\mathbb{R}}$.

Okolí bodu

- Nechť $a \in \mathbb{R}$ a $\epsilon \in \mathbb{R} \wedge \epsilon > 0$. Otevřený interval $(a - \epsilon, a + \epsilon)$ nazýváme **okolím bodu a o poloměru ϵ** a značíme $U_a(\epsilon)$. Někdy též o této množině mluvíme jako o **ϵ -okolí bodu a** .
- Nechť $a \in \mathbb{R}$ a $\epsilon \in \mathbb{R} \wedge \epsilon > 0$. Polouzavřený interval $[a, a + \epsilon)$, respektive $(a - \epsilon, a]$, nazýváme **pravým**, respektive **levým** ϵ -okolí bodu a a značíme ho $U_a^+(\epsilon)$, resp. $U_a^-(\epsilon)$.
- Nechť $c \in \mathbb{R}$. Otevřený interval $(c, +\infty)$, resp. $(-\infty, c)$ nazýváme **okolím bodu $\pm\infty$ v \mathbb{R}** , a značíme $U_{\pm\infty}(c)$.
- Pod okolím bodu $a \in \overline{\mathbb{R}}$ (značíme U_a) máme na mysli buď klasicky okolí $U_a(\epsilon)$ pro nějaké $a \in \mathbb{R}$ a $\epsilon > 0$, nebo okolí $U_{\pm\infty}(c)$ pro nějaké $c \in \mathbb{R}$.

Hromadný bod množiny

Bod $\alpha \in \overline{\mathbb{R}}$ nazýváme **hromadným bodem množiny** $M \subset \mathbb{R}$, právě když v každém okolí U_α bodu α leží nějaký prvek množiny M různý od α .

Reálná funkce a vlastnosti

- mějme $n \in \mathbb{N}$ a $A \subset \mathbb{R}^n$ neprázdnou množinu — zobrazení f neprázdné množiny A do množiny \mathbb{R} ($f : A \rightarrow \mathbb{R}$) nazýváme **reálnou funkcí**
- množina A je definiční obor funkce f , značíme D_f
- obor hodnot funkce f (H_f) — $H_f := \{y \in \mathbb{R} \mid (\exists x \in A)(f(x) = y)\}$
- zobrazení $f : D_f \rightarrow \mathbb{R}$, kde $D_f \subset \mathbb{R}$ je neprázdná množina reálných čísel, nazýváme **reálnou funkcí reálné proměnné**
- omezená funkce — funkce s omezeným definičním oborem (existuje konstanta $K \in \mathbb{R}$ tak, že $|f(x)| \leq K$)
- konstantní funkce — funkce, kde pro všechna $x \in D_f$ platí $f(x) = c$, $c \in \mathbb{R}$
- monotónní funkce — klesající nebo rostoucí
- ryze monotónní funkce — ostře rostoucí či ostře klesající
- sudá funkce — funkce se symetrickým D_f vůči počátku, pro kterou platí $f(-x) = f(x)$ (zrcadlově dle osy y)
- lichá funkce — funkce se symetrickým D_f vůči počátku, pro kterou platí $f(-x) = -f(x)$ (zrcadlově dle osy 1. a 3. kvadrantu)
- periodická funkce — perioda T (konstanta), pro každé $x \in D_f$ platí $f(x \pm T) = f(x)$

Asymptotické meze

- mějme 2 funkce f, g a bod $a \in \overline{\mathbb{R}}$ takový, že a je hromadným bodem množiny $D_f \cap D_g$ a existuje okolí V_a splňující $V_a \cap D_f = V_a \cap D_g$
- řekneme, že funkce f je **asymptoticky shora omezená funkcí g pro x jdoucí k a**, symbolicky $f(x) = \mathcal{O}(g(x))$ pro $x \rightarrow a$, právě když existuje kladná konstanta $c \in \mathbb{R}$ a okolí U_a bodu a tak, že pro všechna $x \in (U_a \cap D_f \cap D_g) \setminus \{a\}$ platí $|f(x)| \leq c \cdot |g(x)|$
- řekneme, že funkce f je **asymptoticky shora striktně omezená funkcí g pro x jdoucí k a**, symbolicky $f(x) = o(g(x))$ pro $x \rightarrow a$, právě když pro každé kladné $c \in \mathbb{R}$ existuje okolí U_a bodu a tak, že pro všechna $x \in (U_a \cap D_f \cap D_g) \setminus \{a\}$ platí $|f(x)| < c \cdot |g(x)|$

Posloupnost

Zobrazení množiny přirozených čísel \mathbb{N} do množiny reálných čísel \mathbb{R} nazýváme reálná číselná posloupnost. Podobné vlastnosti jako u funkcí:

- (ostře) stoupající/klesající, (ryze) monotónní, konstantní, omezená
- hromadný bod má v každém jeho okolí nekonečně mnoho členů dané posloupnosti

Podposloupnost (vybraná posloupnost)

Nechť $(a_n)_{n=1}^{\infty}$ je libovolná posloupnost a $(k_n)_{n=1}^{\infty}$ je ostře rostoucí posloupnost přirozených čísel. Pak posloupnost $(a_{k_n})_{n=1}^{\infty}$ nazýváme posloupností vybranou z posloupnosti $(a_n)_{n=1}^{\infty}$, nebo také podposloupností.

Limita číselné posloupnosti

Reálná posloupnost $(a_n)_{n=1}^{\infty}$ má limitu $\alpha \in \overline{\mathbb{R}}$, právě když pro každé okolí bodu α lze nalézt $N \in \mathbb{N}$ takové, že pro všechna $n \in \mathbb{N}$ větší nebo rovno N platí $a_n \in U_{\alpha}$. V symbolech:

$$(\forall U_{\alpha})(\exists N \in \mathbb{N})(\forall n \in \mathbb{N})(n \geq N \Rightarrow a_n \in U_{\alpha})$$

Tuto skutečnost můžeme zapsat několika ekvivalentními způsoby, a to:

$$\lim_{n \rightarrow \infty} a_n = \alpha \text{ nebo } \lim a_n = \alpha \text{ nebo } a_n \rightarrow \alpha$$

Limita funkce

Mějme funkci $f : A \rightarrow \mathbb{R}$, hromadný bod $a \in \overline{\mathbb{R}}$ množiny A a bod $b \in \overline{\mathbb{R}}$. Funkce f má v bodě a limitu rovnou b , právě když pro každé okolí U_b bodu b existuje okolí U_a bodu a takové, že pokud $x \in U_a \cap A$ a $x \neq a$ pak $f(x) \in U_b$.

$$(\forall U_b)(\exists U_a)(\forall x \in (A \cap U_a) \setminus \{a\})(f(x) \in U_b)$$

Asymptotická ekvivalence

Mějme dvě funkce f, g a bod $a \in \overline{\mathbb{R}}$ takový, že a je hromadným bodem množiny $D_f \cap D_g$ a existuje okolí V_a splňující $V_a \cap D_d = V_a \cap D_g$. Řekneme, že funkce f je asymptoticky ekvivalentní funkci g pro x jdoucí k a ($f(x) \sim g(x)$ pro $x \rightarrow a$) právě když existuje okolí bodu a a funkce u definovaná na U_a pro jejíž limitu v bodě a platí $\lim_{x \rightarrow a} u(x) = 1$ tak, že pro všechna $x \in U_a \cap D_f \cap D_g$ platí $f(x) = u(x)g(x)$.

Konvergentní posloupnost

Posloupnost je konvergentní, pokud $\lim_{n \rightarrow \infty} a_n \in \mathbb{R}$, jinak je divergentní.

Podílové kritérium pro posloupnosti

Bud' $(a_n)_{n=1}^{\infty}$ posloupnost kladných čísel a nechť existuje limita

$$q := \lim_{n \rightarrow \infty} \frac{a_{n+1}}{a_n}$$

Potom platí následující 2 implikace:

- pokud $q < 1$, pak limita posloupnosti $(a_n)_{n=1}^{\infty}$ je rovna nule
- pokud $q > 1$, pak je limita posloupnosti rovna $+\infty$

Spojitost

- nechť f je reálná funkce reálné proměnné a nechť bod $a \in D_f$
- řekneme, že funkce f je spojitá v bodě a , právě když pro její limitu v bode a platí $\lim_{x \rightarrow a} f(x) = f(a)$
- funkce f je spojitá v bodě a zprava, právě když pro jednostrannou limitu $\lim_{x \rightarrow a+} f(x) = f(a)$
- funkce f je spojitá v bodě a zleva, právě když pro jednostrannou limitu $\lim_{x \rightarrow a-} f(x) = f(a)$
- funkce f je spojitá na intervalu J , právě když $f|_J$ (f zúženo na J) je spojitá v každém bodě intervalu J
- funkce f je spojitá, právě když je f spojitá v každém bodě svého definičního oboru

Nástroje na počítání limit

- limita součtu funkcí je součtem limit
- limita součinu funkcí je součinem limit
- limita podílu funkcí je podíl limit
- vytlačení do nekonečna (2 funkce, v okolí bodu a mají vztah \leq , pokud menší má limitu $+\infty$ tak i větší a obráceně)
- o 2 poličajtech (limita sevřené funkce)
- limita složené funkce
- podílové kritérium (jen posloupnosti)

1.14 SP-14 (MA1)

Diferenciální počet reálné funkce jedné reálné proměnné: derivace a její geometrický význam, vztah derivací funkce s její monotonii a konvexitou/konkavitou, lokální extrémy funkce jedné proměnné a metody jejich hledání, asymptoty funkce.

Derivace funkce v bodě

Nechť f je funkce definovaná na okolí bodu $a \in \mathbb{R}$. Pokud existuje limita

$$\lim_{x \rightarrow a} \frac{f(x) - f(a)}{x - a}$$

nazveme její hodnotu **derivací funkce f v bodě a** a označíme $f'(a)$. Pokud je tato limita konečná, řekneme, že f je **diferencovatelná v bodě a** .

Derivace

Budť f funkce s definičním oborem D_f . Nechť M označuje množinu všech $a \in D_f$ takových, že f má konečnou derivaci v bodě a . **Derivací funkce f** nazýváme funkci s definičním oborem M , která každému $x \in M$ přiřadí $f'(x)$. Tuto funkci značíme f' .

Tečna

Mějme funkci f a bod $a \in D_f$ a nechť existuje $f'(a)$. Tečnou funkce f v bodě a nazýváme:

- přímku s rovinicí $x = a$, je-li funkce f spojitá v bodě a a $f'(a) = \pm\infty$
- přímku s rovinicí $y = f(a) + f'(a)(x - a)$ je-li $f'(a) \in \mathbb{R}$ (tedy pokud je diferencovatelná v a)

Věty

- je-li funkce f diferencovatelná v bodě a , pak je i spojitá v bodě a
- derivace součtu funkcí je součet derivací
- derivace součinu: $(f \cdot g)'(a) = f'(a)g(a) + f(a)g'(a)$
- derivace podílu: $(\frac{f}{g})'(a) = \frac{f'(a)g(a) - f(a)g'(a)}{g(a)^2}$, pokud $g(a) \neq 0$
- derivace složené funkce je součin derivace vnější (vnitřní(x)) a derivace vnitřní(x)
- derivace inverzní funkce: $f'(c) = \frac{1}{(f^{-1})'(f(c))}$

Analýza průběhu funkce

- minimum, maximum, infimum, supremum
- lokální maximum, lokální minimum, globální...
- Rolleova věta — funkce je spojitá na $\langle a, b \rangle$, má derivaci v každém bodě (a, b) a $f(a) = f(b)$ — pak existuje bod $c \in (a, b)$, tak že $f'(c) = 0$
- Lagrangeova věta (o přírůstku funkce) — f je spojitá na $\langle a, b \rangle$, má derivaci v každém bodě intervalu (a, b) — potom existuje bod $c \in (a, b)$ tak, že $f'(c) = \frac{f(b) - f(a)}{b - a}$

l'Hospitalovo pravidlo

Nechť pro funkce f a g a bod $a \in \overline{\mathbb{R}}$ platí:

- $\lim_a f = \lim_a g$ nebo $\lim_a |g| = +\infty$
- existuje okolí U_a bodu a splňující $U_a \setminus \{a\} \subset D_{f/g} \cap d_{f'/g'}$
- existuje limita podílu derivací $\lim_a \frac{f'}{g'}$

Potom existuje $\lim_a \frac{f}{g} = \lim_a \frac{f'}{g'}$.

Inflexní bod

Nechť f je spojitá v bodě c . Bod c nazýváme inflexním bodem funkce f , právě když existuje $\delta > 0$ takové, že f je ryze konvexní na intervalu $(c - \delta, c)$ a ryze konkávní na intervalu $(c, c + \delta)$, nebo naopak.

Asymptoty funkce

Řekneme, že funkce f má v bodě $a \in \mathbb{R}$ asymptotu $x = a$, právě když limita $\lim_{x \rightarrow a \pm}$ je rovna $\pm\infty$. Řekneme, že přímka $y = kx + q$ je asymptotou funkce f v $+\infty$ nebo $-\infty$, pokud

$$\lim_{x \rightarrow \pm\infty} (f(x) - kx - q) = 0$$

$$k = \lim_{x \rightarrow \pm\infty} \frac{f(x)}{x}$$

$$q = \lim_{x \rightarrow \pm\infty} f(x) - kx$$

1.15 SP-15 (MA2)

Integrální počet funkce jedné proměnné (neurčitý integrál a Riemannův určitý integrál, metody integrace pomocí substituce a per partes), číselné řady a kritéria jejich konvergence, asymptotické odhadování chování posloupností částečných součtů řad pomocí integrálu.

Primitivní funkce

Nechť f je definovaná na intervalu (a, b) , kde $-\infty \leq a < b \leq +\infty$. Funkci F splňující podmíinku $F'(x) = f(x)$ pro každé $x \in (a, b)$ nazýváme primitivní funkci k funkci f na intervalu (a, b) .

Neurčitý integrál

Nechť k funkci f existuje primitivní funkce na intervalu (a, b) . Množinu všech primitivních funkcí k funkci f na (a, b) nazýváme neurčitým integrálem a značíme jej $\int f$ nebo $\int f(x)$.

Věty

- Nechť funkce f je spojitá na intervalu (a, b) . Pak má funkce f na tomto intervalu primitivní funkci.
- $F + G$ je primitivní funkci k $f + g$
- αF je primitivní funkci k αf

Per Partes

Nechť funkce f je diferencovatelná na intervalu (a, b) a G je primitivní funkce k funkci g na intervalu (a, b) . Nechť existuje primitivní funkce k fG' . Potom existuje primitivní funkce k fg a platí

$$\int fg = fG - \int f'G$$

Substituce

Nechť pro funkce f a φ platí:

- f má primitivní funkci F na intervalu (a, b)
- φ je na intervalu (α, β) diferencovatelná
- $\varphi((\alpha, \beta)) \subset (a, b)$

pak funkce $f(\varphi(x)) \cdot \varphi'(x)$ má primitivní funkci na intervalu (α, β) a platí

$$\int f(\varphi(x)) \cdot \varphi'(x) dx = F(\varphi(x)) + C$$

kde $C \in \mathbb{R}$ je integrační konstanta.

Riemannův určitý integrál

Mějme funkci definovanou na uzavřeném intervalu.

- dělení intervalu — interval hraničními body rozdelen na úseky
- ekvidistantní dělení — úseky jsou stejně dlouhé
- dolní/horní součet funkce (na intervalu) — dle dělení intervalu se pro každý úsek ručí infimum/supremum, a z toho se spočítá dolní/horní obdélník
- dolní/horní integrál funkce na intervalu — supremum dolních / infimum horních součtů
- Riemannův integrál funkce na intervalu — pokud se rovná dolní a horní integrál, tak je to jejich hodnota
- pokud je na intervalu funkce spojitá, existuje Riemannův integrál

Zobecněný Riemannův integrál

Nechť f je funkce definovaná na intervalu $\langle a, b \rangle$ pro nějaké $a \in \mathbb{R}$ a $b \in (a, +\infty) \cup \{+\infty\}$, která má Riemannův integrál na intervalu (a, c) pro každé $c \in (a, b)$. Pokud existuje konečná limita

$$\lim_{c \rightarrow b^-} \int_a^c f(x) dx$$

pak její hodnotu značíme

$$\int_a^b f(x) dx$$

a nazýváme ji zobecněným Riemannovým integrálem funkce f na intervalu $\langle a, b \rangle$ a říkáme že integrál $\int_a^b f(x) dx$ konverguje.

Číselná řada

Formální výraz tvaru

$$\sum_{k=n_0}^{\infty} a_k = a_{n_0} + a_{n_1} + \dots$$

kde $(a_k)_{k=n_0}^{\infty}$ je zadaná číselná posloupnost, nazýváme číselnou řadou. Pokud je posloupnost částečných součtů $(s_n)_{n=n_0}^{\infty}$ definovaná předpisem

$$s_n := \sum_{k=n_0}^n a_k, \quad n \in \mathbb{N}_0, n \geq n_0$$

konvergentní, nazýváme příslušnou řadu také konvergentní. V opačném případě o ní mluvíme jako o divergentní (číselné) řadě. Součtem konvergentní řady $\sum_{k=n_0}^{\infty} a_k$ a nazýváme hodnotu limity $\lim_{n \rightarrow \infty} s_n$.

Kritéria konvergence číselných řad

- Nutná podmínka konvergence

Pokud řada $\sum_{k=0}^{\infty} a_k$ konverguje, potom pro limitu jejích sčítanců platí $\lim_{k \rightarrow \infty} a_k = 0$.

- Bolzano-Cauchy

Řada $\sum_{k=0}^{\infty} a_k$ konverguje právě tehdy, když pro každé $\epsilon > 0$ existuje $n_0 \in \mathbb{R}$ tak, že pro každé přirozené $n \geq n_0$ a $p \in \mathbb{N}$ platí $|a_n + a_{n+1} + \dots + a_{n+p}| < \epsilon$.

- Absolutní konvergence

Řadu $\sum_{k=0}^{\infty} a_k$ nazýváme absolutně konvergentní, pokud číselná řada $\sum_{k=0}^{\infty} |a_k|$ konverguje. Pokud řada absolutně konverguje, potom konverguje.

- Leibnizovo kritérium

Bud' $(a_k)_{k=0}^{\infty}$ monotonné posloipnost konvergující k nule. Potom je řada $\sum_{k=0}^{\infty} (-1)^k a_k$ konvergentní.

- Srovnávací kritérium

Bud'te $\sum_{k=0}^{\infty} a_k$ a $\sum_{k=0}^{\infty} b_k$ číselné řady. Potom platí následující 2 tvrzení:

1. Nechť existuje $k_0 \in \mathbb{N}$ takové, že pro každé $k \in \mathbb{N}$ větší než k_0 platí nerovnosti $0 \leq |a_k| \leq b_k$ a nechť řada $\sum_{k=0}^{\infty} b_k$ konverguje. Potom řada $\sum_{k=0}^{\infty} a_k$ absolutně konverguje.
2. Nechť existuje $k_0 \in \mathbb{N}$ takové, že pro každé $k \in \mathbb{N}$ větší nebo rovno než k_0 platí nerovnosti $0 \leq a_k \leq b_k$ a $\sum_{k=0}^{\infty} a_k$ diverguje. potom i řada $\sum_{k=0}^{\infty} b_k$ diverguje.

- d'Alembertovo kritérium

Nechť $a_k > 0$ pro každé $k \in \mathbb{N}_0$. Pokud $\lim_{k \rightarrow \infty} \frac{a_{k+1}}{a_k} > 1$, potom řada $\sum_{k=0}^{\infty} a_k$ diverguje. Pokud ovšem $\lim_{k \rightarrow \infty} \frac{a_{k+1}}{a_k} < 1$, potom $\sum_{k=0}^{\infty} a_k$ konverguje.

Odhad posloupnosti částečných součtů

Nechť f je spojitá funkce na $(1, +\infty)$ a $n \in \mathbb{N}$. Je-li f klesající, pak platí

$$f(n) + \int_1^n f(x)dx \leq \sum_{k=1}^n f(k) \leq f(1) + \int_1^n f(x)dx$$

Je-li f rostoucí, pak platí

$$f(1) + \int_1^n f(x)dx \leq \sum_{k=1}^n f(k) \leq f(n) + \int_1^n f(x)dx$$

Integrální kritérium

Budě $\sum_{k=0}^{\infty} a_k$ číselná řada s kladnými členy taková, že existuje spojitá funkce definovaná na $(1, +\infty)$ taková, že $f(n) = a_n$ pro každé n . Potom:

- pokud zobecněný Riemannův integrál $\int_1^{\infty} f(x)dx$ konverguje, pak číselná řada $\sum_{k=0}^{\infty} a_k$ konverguje
- pokud zobecněný Riemannův integrál $\int_1^{\infty} f(x)dx$ diverguje, pak číselná řada $\sum_{k=0}^{\infty} a_k$ diverguje

1.16 SP-16 (MA2)

Funkce více proměnných: diferenciální počet funkcí více proměnných (limita, parciální derivace, derivace, gradient, Hessova matice), kvadratické formy a jejich definitnosti, analytická metoda hledání lokálních extrémů funkcí více proměnných (bez omezení).

Euklidovská norma a vzdálenost

Euklidovskou normu vektoru $\mathbf{x} \in \mathbb{R}^n$ definujeme předpisem

$$\|\mathbf{x}\| := \sqrt{\sum_{j=1}^n x_j^2}$$

Euklidovskou vzdálenost dvou bodů $\mathbf{x} \in \mathbb{R}^n$ a $\mathbf{y} \in \mathbb{R}^n$ pak představuje číslo

$$d(\mathbf{x}, \mathbf{y}) := \|\mathbf{x} - \mathbf{y}\| = \sqrt{\sum_{j=1}^n (x_j - y_j)^2}$$

Nerovnosti

- $\langle \mathbf{x} | \mathbf{y} \rangle$ je skalární součin vektorů (součet součinů čísel na stejně pozici)
- Schwarzova nerovnost — pro každé $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ platí nerovnost $|\langle \mathbf{x} | \mathbf{y} \rangle| \leq \|\mathbf{x}\| \cdot \|\mathbf{y}\|$, navíc rovnost nastává právě tehdy, když jeden z vektorů je násobkem druhého
- trojúhelníková nerovnost — pro každé $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ platí $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$

Okolí bodu

Mějme bod $\mathbf{a} \in \mathbb{R}^n$ a poloměr $\epsilon > 0$. Potom okolím bodu \mathbf{a} o poloměru ϵ nazýváme množinu všech bodů $\mathbf{x} \in \mathbb{R}^n$, jejichž vzdálenost od bodu \mathbf{a} je menší než ϵ a značíme ho $U_{\mathbf{a}}(\epsilon)$. Tedy:

$$U_{\mathbf{a}}(\epsilon) := \{\mathbf{x} \in \mathbb{R}^n \mid d(\mathbf{x}, \mathbf{a}) < \epsilon\} \subset \mathbb{R}^n$$

Hromadný bod

Bod $\mathbf{a} \in \mathbb{R}^n$ nazýváme hromadným bodem množiny $M \subset \mathbb{R}^n$, právě když v každém okolí bodu \mathbf{a} leží bod množiny M různý od \mathbf{a} .

Vnitřní bod množiny

O bodu $\mathbf{a} \in M \subset \mathbb{R}^n$ řekneme, že je vnitřním bodem množiny M , právě když existuje okolí $U_{\mathbf{a}}$ bodu \mathbf{a} takové, že $U_{\mathbf{a}} \subset M$.

Otevřená množina

O množině $M \subset \mathbb{R}^n$ řekneme, že je otevřená, právě když pro každý bod $\mathbf{a} \in M$ existuje okolí $U_{\mathbf{a}}$ takové, že $U_{\mathbf{a}} \subset M$.

Limita vektorové posloupnosti

Řekneme, že posloupnost $(\mathbf{x}_k)_{k=1}^{\infty}$ vektorů $\mathbf{x}_k \in \mathbb{R}^n$ má limitu (případně konverguje k) $\mathbf{a} \in \mathbb{R}^n$, právě když pro každé okolí bodu $U_{\mathbf{a}}$ bodu \mathbf{a} existuje $N \in \mathbb{N}$ takové, že pro každé přirozené $k > N$ platí $\mathbf{x}_k \in U_{\mathbf{a}}$. Tento fakt značíme $\lim_{k \rightarrow \infty} \mathbf{x}_k = \mathbf{a}$. Vektorovou posloupnost mající limitu, která je dle definice nutně prvkem \mathbb{R}^n , nazýváme konvergentní. Všechny ostatní posloupnosti jsou divergentní.

Limita funkce více proměnných

Mějme funkci n reálných proměnných $F : D_F \rightarrow \mathbb{R}^m$, $D_F \subset \mathbb{R}^n$ a hromadný bod \mathbf{a} množiny D_F . Potom funkce F má v bodě \mathbf{a} limitu $\mathbf{b} \in \mathbb{R}^m$, právě když pro každé okolí $U_{\mathbf{b}}$ existuje okolí $U_{\mathbf{a}}$ takové, že když když $\mathbf{x} \in (U_{\mathbf{a}} \cap D_F) \setminus \{\mathbf{a}\}$ pak platí $F(\mathbf{x}) \in U_{\mathbf{b}}$. Symbolicky zapisujeme jako $\lim_{\mathbf{x} \rightarrow \mathbf{a}} F(\mathbf{x}) = \mathbf{b}$. Pokud $m = 1$, pak lze výsledek vybírat z rozšířené reálné osy, tedy navíc s prvky $+\infty$ a $-\infty$.

Parciální derivace

- Mějme reálnou funkci n reálných proměnných $f : D_f \rightarrow \mathbb{R}$, $D_f \subset \mathbb{R}^n$, definovanou na okolí bodu $\mathbf{a} \in D_f$ a $j \in \hat{n}$. Existuje-li limita

$$\lim_{h \rightarrow 0} \frac{f(\mathbf{a} + h\mathbf{e}_j) - f(\mathbf{a})}{h}$$

pak její hodnotu nazýváme parciální derivací funkce f v bodě \mathbf{a} podle j -té proměnné a značíme ji $\frac{\partial f}{\partial x_j}(\mathbf{a})$, případně $\partial_{x_j} f(\mathbf{a})$.

- označme M jako množinu všech vnitřních bodů \mathbf{a} množiny D_f , v kterých existuje parciální derivace. Potom funkci přiřazující hodnotu $\frac{\partial f}{\partial x_j}(\mathbf{a})$ každému $\mathbf{a} \in M \subset \mathbb{R}^n$ nazýváme parciální derivací funkce f podle j -té proměnné a značíme ji $\frac{\partial f}{\partial x_j}$. případně $\partial_{x_j} f$.

Gradient

Mějme reálnou funkci n reálných proměnných $f : D_f \rightarrow \mathbb{R}$, $D_f \subset \mathbb{R}^n$ mající všechny parciální derivace v bodě $\mathbf{a} \in D_f$. Potom řádkový vektor

$$(\frac{\partial f}{\partial x_1}(\mathbf{a}), \frac{\partial f}{\partial x_2}(\mathbf{a}), \dots, \frac{\partial f}{\partial x_n}(\mathbf{a})) \in \mathbb{R}^{1,n}$$

nazýváme gradientem funkce f v bodě \mathbf{a} a používáme pro něj značení $\nabla f(\mathbf{a})$ nebo $\text{grad } f(\mathbf{a})$.

Derivace vektorové funkce

Mějme zobrazení $F : D_F \rightarrow \mathbb{R}^m$, $D_F \subset \mathbb{R}^n$, definované na okolí bodu \mathbf{a} . Derivací zobrazení F v bodě \mathbf{a} nazýváme matici $DF(\mathbf{a}) \in \mathbb{R}^{m,n}$ splňující:

$$\lim_{\mathbf{x} \rightarrow \mathbf{a}} \frac{\|F(\mathbf{x}) - F(\mathbf{a}) - DF(\mathbf{a}) \cdot (\mathbf{x} - \mathbf{a})\|}{\|\mathbf{x} - \mathbf{a}\|} = 0$$

Složky matice $DF(\mathbf{a})$

Pokud má zobrazení $F : D_f \rightarrow \mathbb{R}^m$, $D_f \subset \mathbb{R}^n$, definované na okolí bodu \mathbf{a} , derivaci $DF(\mathbf{a}) \in \mathbb{R}^{m,n}$ v bodě \mathbf{a} , potom

$$DF(\mathbf{a}) = \begin{pmatrix} \frac{\partial F_1}{\partial x_1}(\mathbf{a}) & \cdots & \frac{\partial F_1}{\partial x_n}(\mathbf{a}) \\ \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial x_1}(\mathbf{a}) & \cdots & \frac{\partial F_n}{\partial x_n}(\mathbf{a}) \end{pmatrix}$$

Hessova matice

Na derivaci, resp. gradient, funkce $f : D_f \rightarrow \mathbb{R}$, $D_f \subset \mathbb{R}^n$, lze nahlížet jako na zobrazení $Df : A \rightarrow \mathbb{R}^n$, $A \subset D_f$, jeho derivací v bodě $\mathbf{a} \in A$ je pak matice typu $\mathbb{R}^{n,n}$, kterou nazýváme Hessovou maticí a značíme $\nabla^2 f(\mathbf{a})$. Pokud existuje, pak platí:

$$\nabla^2 f(\mathbf{a}) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2}(\mathbf{a}) & \frac{\partial^2 f}{\partial x_2 \partial x_1}(\mathbf{a}) & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_1}(\mathbf{a}) \\ \frac{\partial^2 f}{\partial x_1 \partial x_2}(\mathbf{a}) & \frac{\partial^2 f}{\partial x_2^2}(\mathbf{a}) & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_2}(\mathbf{a}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n}(\mathbf{a}) & \frac{\partial^2 f}{\partial x_2 \partial x_n}(\mathbf{a}) & \cdots & \frac{\partial^2 f}{\partial x_n^2}(\mathbf{a}) \end{pmatrix}$$

Kvadratická forma

Funkci $q : \mathbb{R}^n \rightarrow \mathbb{R}$ nazýváme kvadratickou formou, právě když existuje symetrická matice $M \in \mathbb{R}^{n,n}$ splňující

$$q(\mathbf{x}) = \sum_{j,k=1}^n M_{j,k} x_j x_k, \quad \text{pro každé } \mathbf{x} = (x_1, \dots, x_n)^T \in \mathbb{R}^n$$

Typy definitnosti kvadratických forem

Kvadratickou formu $q : \mathbb{R}^n \rightarrow \mathbb{R}$ nazveme:

- pozitivně definitní (PD), právě když $q(\mathbf{x}) > 0$ pro každé nenulové $\mathbf{x} \in \mathbb{R}^n$
- pozitivně semidefinitní (PSD), právě když $q(\mathbf{x}) \geq 0$ pro každé $\mathbf{x} \in \mathbb{R}^n$
- indefinitní (ID), právě když existují vektory $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ splňující $q(\mathbf{x}) > 0$ a $q(\mathbf{y}) < 0$
- negativně semidefinitní (NSD), právě když $q(\mathbf{x}) \leq 0$ pro každé $\mathbf{x} \in \mathbb{R}^n$
- negativně definitní (ND), právě když $q(\mathbf{x}) < 0$ pro každé nenulové $\mathbf{x} \in \mathbb{R}^n$

Určování definitnosti KF

- pokud má symetrická matice M na diagonále prvky s různým znaménkem, potom je ID
- podle vlastních čísel (pouze kladná \Rightarrow PD, nezáporná \Rightarrow PSD, různá znaménka \Rightarrow ID,...)
- podle úpravy na čtverce
- Sylvesterovo kritérium — determinanty postupně pro podmatice z levého horního rohu, pokud vždy kladné číslo \Rightarrow PD, pokud se střídají \Rightarrow ND

Lokální extrémy

- pokud existuje okolí bodu, kde je bod největší/nejmenší atd tak je to (ostré) lokální maximum/minimum
- pokud je v bodě extrém, parciální derivace podle j -té proměnné bud' je nula nebo neexistuje
- pokud je v bodě extrém a existují všechny parciální derivace, gradient je nulový
- jak hledat lokální extrémy? najdeme stacionární body (kde je gradient nula) a prozkoumáme
- ve zkoumaných bodech vytvoříme Hessovu matici — musí být aspoň PSD nebo NSD, aby mohl existovat extrém
- pokud je Hessova matice PD nebo ND, je to minimum/maximum (postačující podmínka)

1.17 SP-17 (OSY)

Procesy a vlákna, jejich implementace, nástroje pro synchronizaci vláken. Klasické synchronizační úlohy. Uvážnutí (deadlock) vláken (alokace prostředků, Coffmanovy podmínky, strategie pro řešení uváznutí).

- **Program:**

Program je v systému reprezentován spustitelným binárním programem, který je uložený v sekundární paměti (např. disk).

- **Proces:**

Instance spuštěného programu/aplikace. Entita, v rámci které jsou alokovány prostředky (paměť, vlákna, otevřené soubory, zámky, semafory, sokety,...).

- **Vlákno:**

Výpočetní entita (proud instrukcí), které je přidělováno jádro CPU. Vlákna vytvořená v rámci procesu sdílí většinu prostředků alokovaných v tomto procesu.

Vytvoření procesu:

Nový proces lze vytvořit jako kopii/klon původního procesu, či jako úplně nový proces. V Unixu fork() / exec(), ve Windows CreateProcessA().

- **fork():**

Vytvoří nový proces, který je kopií toho procesu, ze kterého byla tato funkce zavolána. V případě chyby vrací -1, v potomkovi vrací 0, v rodiči vrací PID potomka.

- **exec():**

Adresový prostor aktuálního procesu je přepsán obsahem souboru, který se začne vykonávat od začátku.

- **wait():**

Zablokuje rodičovský proces, ve kterém je zavolána, dokud se konkrétní/jeden potomek neukončí.

Ukončení procesu:

- jádro se pokusí předat návratový kód rodiči
- ukončí se všechna vlákna pod procesem
- uvolní se adresový prostor procesu a příslušné struktury OS
- proces se může ukončit sám (buď normální konec programu jako *return*, nebo chyba, kvůli které se sám ukončí), nebo může být ukončen jádrem (fatální chyba nebo signál od jiného procesu)

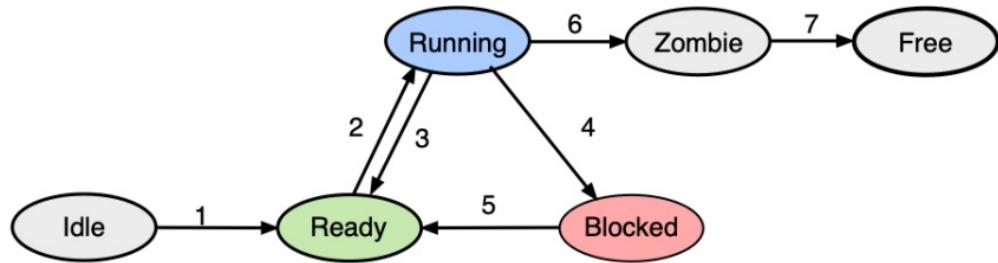
Vlákna:

- proces se implicitně vytváří s jedním "main"vlákнем
- další vlákna lze vytvořit z hlavního (volání OS)

Plánování:

- vláken je typicky zásadně více než logických jader procesoru
- jedno vlákno je zpracováváno max 1 logickým jádrem
- aby se vlákna na jádrech vystřídala, používá se typicky preemptivní plánování
 - vlákno je na základě plánovacích kritérií vybráno, a je mu přiděleno volné jádro CPU
 - vláknu je přiděleno množství času na CPU
 - vláknu je jádro odebráno, pokud uplyne přidělený čas, vlákno provede systémové volání nebo dojde k přerušení
- přepínání kontextu (vystřídání vláken na jádře CPU)
 - kontext = všechny nezbytné informace pro pozdější spuštění přerušeného vlákna od okamžiku přerušení
 - kontext se uloží do paměti, naplánuje se další vlákno a jeho kontext se nahraje do CPU jádra

Stavy vláken:



- **Časově závislé chyby:**

Situace, kdy více vláken používá společné sdílené prostředky a výsledek deterministického algoritmu je závislý na rychlosti jednotlivých vláken, které používají tyto prostředky. Špatně se detekují — lze předcházet správným návrhem paralelního algoritmu.

- **Kritická sekce:**

Část programu, kde vlákna používají sdílené prostředky.

- **Sdružené kritické sekce:**

Kritické sekce více vláken, které se týkají stejného sdíleného prostředku.

- **Vzájemné vyloučení:**

Vláknům není dovoleno sdílet stejný prostředek ve stejném čase, tedy se nenachází ve stejné sdružené sekci současně.

- **Korektní paralelní program:**

Nesmí klást předpoklady na rychlosť vláken a počet jader. Musí zajistit výlučný přístup ke sdíleným prostředkům. Mimo kritické sekce by vlákno nemělo být zpomalováno ostatními vlákny.

Problémy s použitím synchronizace vláken:

- deadlock (x vláken čeká na událost, kterou může vyvolat jen jedno z čekajících vláken)
- livelock (několik vláken vykonává neužitečný výpočet, ale nemohou dokončit)
- starvation (vlákno ve stavu "ready" předbíháno a dlouho se nedostane na řadu)

Zamykání kritických sekcí:

- zámek značí, zda je ve sdružené kritické sekci už jiné vlákno — musí se k němu přistupovat atomicky (např. TSL — Test and Set Lock)
- aktivní vs blokující čekání

- **Zámek (mutex):**

Pamatuje si svůj stav (zamčený/odemčený) a množinu vláken blokovaných na něm. Jsou nad ním definovány atomické operace lock() a unlock().

- **Podmíněná proměnná (conditional variable):**

Pamatuje si, která vlákna jsou na ní blokována. Jsou definovány operace cond_wait(mutex) a cond_signal(). cond_wait(mutex) — mutex musí být zamčen volajícím vláknam, funkce odblokuje mutex a zablokuje vlákno dokud nepřijde cond_signal().

- **Semafor:**

Obsahuje čítač a seznam blokovaných vláken. Jsou definovány operace sem_init(int) (nastaví čítač na 0), sem_wait() (vstup do sekce — pokud čítač je větší než 0 tak vlákno vstoupí a dekrementuje čítač, jinak se zablokuje) a sem_post() (uvolní nějaké čekající vlákno, nebo inkrementuje čítač).

- **Bariéry:**

Obsahuje čítač (síla bariéry — kolik vláken musí čekat, aby byla odblokována) a blokovaná vlákna. Operace: barrier_init(int) (nastaví sílu bariéry) a barrier_wait() (pokud čítač je více než 1, vlákno čeká a čítač je dekrementován, jinak jsou všechna vlákna probuzena)

Synchronizační úlohy:

- Večeřící filosofové
 - N filosofů u kulatého stolu
 - každý má před sebou jídlo a mezi sousedními talíři je vždy 1 vidlička (celkem tedy N vidliček)
 - pokud chce filosof jíst, musí získat obě vidličky vedle jeho talíře
 - stavý filosofa: přemýslí (nechce a nemá vidličky), má hlad (pokouší se získat obě vidličky), jí (má obě vidličky)
 - optimální řešení: může jíst až $\lfloor N/2 \rfloor$ filosofů, nevznikají časově závislé chyby ani synchronizační problémy
 - řešení: pokud mam hlad zamknu mutex, kouknu jestli jsou volny vidlicky, pokud ne spim (odemykam mutex), pokud jo, beru, odemykam mutex a jim. Az dojím, zamknu mutex, vratiš vidlicky, probudim sousedy a odemknu mutex.
- Čtenáři — písáři
 - v systému je 1 sdílený prostředek
 - písáři mohou modifikovat, čtenáři pouze číst
 - chceme, aby pokud není modifikováno (nepřistupuje písář) mohlo číst více čtenářů
 - zároveň by nikdo neměl být předbíhán
 - řešení: písáři i čtenáři se řadí do fronty, ale po skupinách — pokud přijdu na konec fronty a je tam už stejný typ, přidám se do skupiny — na začátku fronty je probuzena celá skupina a bud' písáři postupně zapíšou, nebo čtenáři společně přečtou
- Spící holiči
 - v holičství je N holičů a křesel k holení, a M křesel k čekání
 - pokud nejsou zákazníci, holič sedne do holícího křesla a usne
 - pokud přijde zákazník, bud' probudí holiče (pokud je volný), nebo si sedne do čekárny (pokud je místo) jinak odejde

Obecně alokace prostředku:

- vlákno žádá o prostředek pomocí alokační funkce
- pokud je prostředek volný, je přidělen
- pokud je již alokovaný, vlákno může být blokováno (v závislosti na alokační funkci — `mutex_lock` blokuje, `mutex_try_lock` blokuje jen na určitý čas, `fork()` a `malloc()` neblokují)
- v případu 2 a 3 se vlákno pak samo rozhodne jak pokračovat

Coffmanovy podmínky

Uváznutí (deadlock) nastane pouze pokud jsou splněny všechny následující podmínky:

- Vzájemné vyloučení — každý prostředek nemůže být sdílen více vláknů
- Podmínka neodnímatelnosti — již přidělený prostředek nemůže být odebrán násilím
- Podmínka ”drž a čekej” — vlákno s již přiděleným prostředkem může žádat o další
- Podmínka kruhového čekání — musí existovat smyčka více vláken, ve které každé vlákno čeká na prostředek držený dalším vláknem ve smyčce

Řešení uváznutí:

- Pštrosí strategie — ignorování
- Prevence uváznutí — nesplnění alespoň jedné z Coffmanových podmínek
- Předcházení vzniku uváznutí — pečlivá alokace prostředků
- Detekce uváznutí a zotavení — uváznutí je detekováno a odstraněno

Implementace procesů:

- jádro OS si udržuje zřetězený seznam struktur — tabulku procesů
- jedna položka tabulky obsahuje vše nezbytné, co si OS musí o procesu pamatovat (PCB — Process Control Block)
 - identifikace procesu (id procesu, id rodiče, číslo úlohy/seance/projektu, jméno procesu...)
 - identita/bezpečnost (vlastník, skupiny, práva procesu)
 - informace o alokovaných prostředcích (paměť, soubory, prostředky pro meziprocesovou komunikaci)

Implementace vláken:

- Thread Control Block (TCB) — identifikace vlákna, info o přepínání kontextu (registry), informace pro plánování vláken
- Implementace v uživatelském prostoru (zastaralé)
 - OS přistupuje k procesům jako by měly jedno vlákno
 - proces si svá vlákna spravuje sám
 - kooperativní plánování pro vlákna v procesu
- Implementace v jádře OS
 - OS plánuje samotná vlákna, ne procesy
 - OS udržuje jede PCB pro každý proces, jeden TCB pro každé vlákno
 - preemptivní plánování pro vlákna v procesu

Plánování v dnešních OS: Prioritní Round Robin

X front s různou prioritou. Na základě předchozího běhu vlákna se buď zvýší časové kvantum a sníží priorita (pokud vlákno využilo všechn čas) nebo zvýší prioritu a sníží čas (pokud nevyužilo celé časové kvantum).

1.18 SP-18 (OSY)

Virtualizace hlavní paměti stránkováním, principy překladu virtuálních adres na fyzické, struktura tabulek stránek, algoritmy pro nahrazování stránek.

Princip virtuální paměti se stránkováním:

- Proces používá virtuální/logické adresy, ty adresují virtuální adresní prostor
- VAS (virtual address space) je rozdělen na stejně velké stránky — typicky 4KB nebo 8KB
- na stejně velké úseky (rámce) je rozdělena fyzická paměť
- aktuálně používané stránky musí být aktuálně v hlavní paměti
- virtuální adresa = číslo stránky + offset

Možnosti překladu adres:

- jednoúrovňová tabulka stránek
- víceúrovňová tabulka stránek
- invertovaná tabulka stránek

Překlad adres zajišťuje MMU s TLB (viz 3.6).

Jednoúrovňová TS:

- pro každou stránku VAS daného procesu obsahuje jeden řádek obsahující číslo rámce a kontrolní byty (Present bit (P) — je stránka v hlavní paměti?, Reference bit (R) — přistupovalo se ke stránce?, Modify bit (M) — byl obsah modifikován?, Přístupová práva, Cache disabled/enabled, R/W, User/Supervisor (U/S) - lze přistupovat v uživatelském módu?)
- číslo stránky = index do této tabulky
- pro každý proces jedna tabulka

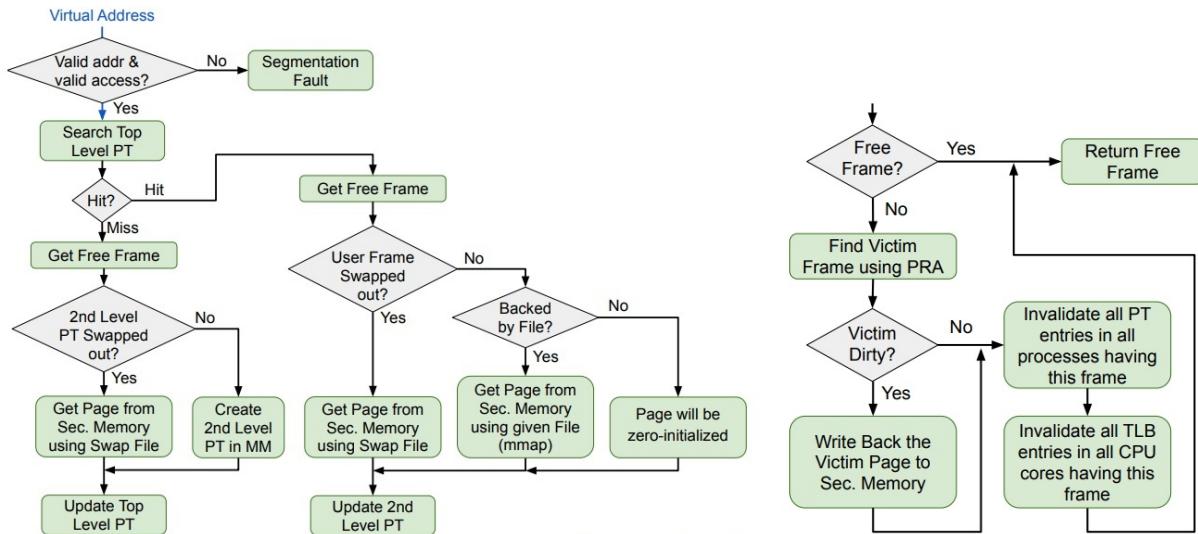
Víceúrovňová TS:

- virtuální adresa se skládá z n indexů, které ukazují do tabulek jednotlivých úrovní + offset
- tabulky stránek úrovní $1, \dots, n-1$ obsahují číslo rámce kde je následující tabulka + present bit
- tabulka úrovně n obsahuje present bit + číslo rámce hledané stránky
- hlavní/první tabulka je v paměti vždy

Invertovaná TS:

- obsahuje pro každý rámec fyzické paměti jeden řádek, kde je uloženo: číslo stránky nahrané do rámce, číslo procesu, kterému stránka patří, kontrolní byty, index zřetězení (stejně velký jako index do tabulky)
- existuje 1 tabulka pro celý systém
- číslo stránky se hashovací funkcí převede na index do tabulky
- více stránek se může namapovat na stejný rámec — proto index zřetězení

Řešení výpadku stránek:



Algoritmy pro náhradu stránek

V okamžiku kdy většina/všechny rámce fyzické (hlavní) paměti jsou obsazené, je úkolem OS najít vhodný rámeček, jehož obsah (stránka) se uvolní. K tomu slouží algoritmy pro náhradu stránek.

Co je od takových algoritmů požadováno?

- minimalizace počtu výpadků stránek
- rychlosť
- jednoduchá implementace

Tyto algoritmy využívají principů prostorové a časové lokality.

Optimální algoritmus:

- nahradí se stránka, která má čas příštího přístupu nejdelší
- generuje minimální počet výpadků stránek
- sice nelze udělat, ale slouží pro porovnání kvality reálných algoritmů

NRU (Not Recently Used)

- pro každou stránku se pamatuje reference bit (R) a modified bit (M) (viz výše)
- reference bit se periodicky nastavuje na 0
- stránky jsou rozděleny do 4 tříd (RM == 00, RM == 01, RM == 10, RM == 11)
- nahradí se nějaká stránka z co nejnižší třídy (tedy 00 → 01 → 10 → 11)

Jednoduchý na pochopení i implementaci, poměrně nízký počet výpadků stránek.

FIFO (First In First Out)

- je udržován seznam stránek nahraných v paměti
- nově nahraná stránka je zaznamenána na konec seznamu
- nahrazena je první stránka ze seznamu

Jednoduchý na pochopení i implementaci, ale generuje poměrně vysoký počet výpadků stránek.

Clock algoritmus

- modifikovaný FIFO algoritmus
- seznam stránek jako kruhová fronta
- na počátku ručička ukazuje na první položku seznamu

- pro každou položku je zaznamenán reference bit, který je nastaven na 1 při pridání stránky do seznamu a při přístupu k ní
- při potřebě nahradby stránky ručička u položky, na kterou ukazuje, zjistí stav R bitu — pokud 1, vynuluje a jde na další položku — pokud 0, tato stránka se nahradí a ručička se posune

Jednoduchý na implementaci, generuje poměrně nízký počet výpadků stránek. Existují varianty s více ručičkami, kde podle rychlosti posunu ručiček a jejich rozevření je definováno časové okno, dle kterého zjišťujeme, zda byla stránka nedávno použita.

LRU (Least Recently Used)

- vybere se stránka, která je nejdelší dobou bez přístupu
- pro každou položku je navíc zapamatován čas použití, který se aktualizuje při každém použití (existuje globální čítač, který se zvýší při každém přístupu do paměti, jeho hodnota je pak zanesena k právě použité stránce)
- kandidát je taková stránka, která má nejnižší čas posledního přístupu (nutno porovnat všechny)

Generuje poměrně nízký počet výpadků stránek, dobrá approximace optimálního algoritmu. Složitější implementace (čítač s časem a porovnání všech stránek)

Aging algoritmus

- simulace LRU algoritmu
- pro každou stránku je zapamatováno: R bit (nastaví se na 1 při každém přístupu), n -bitový čítač C , který má po načtení stránky do paměti všechny bity na 1
- periodicky se pro každou stránku C posune o 1 doprava, jeho nejvýznamější bit se nastaví na R, a R se nastaví na 0
- vhodným kandidátem je stránka s nejnižší hodnotou C

Menší režie než LRU, ale není tak přesný (nepamatuje se přesný čas, ale jen interval, kdy se naposledy přistupovalo — omezená historie).

1.19 SP-19 (PA1)

Datové typy v programovacích jazycích. Staticky a dynamicky alokované proměnné, spojové seznamy. Modulární programování, procedury a funkce, vstupní a výstupní parametry. Překladač, linker, debugger.

Datové typy

V programovacích jazycích používáme proměnné, tedy něco, co uchovává datovou hodnotu s nějakou vnitřní strukturou. Proměnné jsou identifikovány svými jmény — identifikátory. Datový typ proměnné definuje vnitřní strukturu/reprezentaci dat a jejich význam. Tím určuje jakých hodnot může proměnná nabývat a také jaké operace lze s proměnnou (její hodnotou) vykonávat.

Jednoduché datové typy:

- celočíselné
 - existují různé délky — short, int, long (byte, long long, ...)
 - signed (znaménkové) — umí uložit i záporné hodnoty, používá se doplňkový kód
 - unsigned (neznaménkové) — ukládá jen kladné hodnoty, přímý kód
- s pohyblivou řádovou čárkou
 - existují různé délky — float, double, long double
 - znaménko (1 bit) + mantisa (velikost=přesnost) + exponent (velikost=rozsah)
- znakové
 - Znaky jsou kódovány jako čísla, používá se ASCII / extended ASCII / UNICODE.
- logická hodnota
 - Není v C, ale často se v jazycích vyskytuje (boolean — true/false).

Další datové typy:

- ukazatel (pointer)
 - Adresy paměti, kde je uložen datový typ pointeru (pointer vždy ukazuje na konkrétní typ/funkci, případně void).
- výčtový typ (enum)
- struktura
 - Je složena z dalších datových typů, klidně dalších struktur.
- union
 - Ukládá více různých datových typů na stejné místo.
- třída (ve vyšších jazycích)

Statická a dynamická alokace

Staticky alokované proměnné:

- vzniknou běžnou deklarací
- ukládají se na zásobník (lokální proměnné) či do části .BSS (neinicializované globální proměnné) a .DATA (inicializované globální proměnné)
- v případě pole je nutno znát v době komplikace velikost (statická velikost)

Dynamicky alokované proměnné

- vzniknou použitím speciální funkcí/operátorem
- ukládají se na haldě (heap)
- přistupujeme přes pointer
- je možné alokovat paměť podle hodnot spočítaných za běhu programu

Spojové seznamy

- oproti poli nejsou položky seřazeny v paměti, ale každý prvek seznamu obsahuje ukazatel na další prvek.
- podobně jako v dynamicky alokovaném poli lze ukládat předem neznámý objem dat
- nelze jednoduše indexovat, ale lze libovolně přidávat či ubírat prvky z jakékoli pozice v seznamu

Modulární programování

- složitější programy mohou být rozděleny do modulů
- tyto moduly lze použít v různých dalsích částech programu
- modul má svou specifikační část (deklarace poskytovaných prostředků/rozhraní) a implementační část (definice/implementace poskytovaných prostředků)
- v C/C++ typicky hlavičkový soubor (.h/.hpp) a implementační soubor (.c/.cpp)

Procedury, funkce a parametry

- procedura/funkce je posloupnost příkazů uložených v paměti programu
 - procedura — bez návratové hodnoty (typ void)
 - funkce — s návratovou hodnotou
- použijeme ji zavoláním přes její jméno
- deklarace je specifikace jejího rozhraní — parametrů a typu návratové hodnoty
- definice je samotný kód funkce
- vstupní parametry jsou informace, které využije kód funkce
- výstupní parametry jsou výsledkem běhu funkce — typicky se nějak změní a tím nám dají výsledek

Překladač

- překládá vyšší programovací jazyky do nižších
- ze zdrojového kódu vzniká objektový soubor — modul se strojovým kódem
- front-end přeloží konkrétní jazyk do vnitřní reprezentace (abstrakce nezávislá ani na platformě ani na jazyku)
- back-end přeloží vnitřní reprezentaci do strojového kódu konkrétní platformy

Linker

- spojuje přeložené moduly do výsledného celku — programu
- výstupem je spustitelný soubor

Debugger

- usnadňuje hledání chyb v kódu, také usnadňuje pochopení programu
- je vhodné kompilovat s informacemi pro ladění
- je možné si na nějakém místě běhu programu zastavit a např. sledovat obsah proměnných, pouštět každý krok programu postupně...

1.20 SP-20 (AG1 + PA1)

Časová a paměťová složitost algoritmů. Algoritmy vyhledávání (sekvenční, půlením intervalu), slučování a řazení (BubbleSort, SelectSort, InsertSort, MergeSort, QuickSort). Dolní mez složitosti řazení v porovnávacím modelu. Řazení v lineárním čase.

Složitost

- vyjadřuje závislost potřebného času / paměti pro vykonání algoritmu v závislosti na velikosti vstupních dat
- popisuje se asymptotickým horním odhadem
- typicky se řeší nejhorší možný průběh algoritmu
- asymptotický odhad nám zjednodušuje vyjádření složitosti — zanedbávají se multiplikativní konstanty, v případě složeného výrazu se uvažuje pouze nejvýraznější člen

Vyhledávání

- sekvenční
 - procházení prvků popořadě, dokud se nenajde hledaný nebo neprojdou všechny
 - neklade žádné nároky na seřazení prvků
 - časová složitost lineární vzhledem k datům, tedy $O(n)$
- půlení intervalu
 - binární vyhledávání
 - vyžaduje seřazené prvky
 - vždy zkонтroluje prvek uprostřed aktuálního intervalu, podle porovnání s hledaným prvkem pokračuje dále do levé nebo pravé poloviny (zanoří se)
 - časová složitost logaritmická, tedy $O(\log n)$

Slučování a řazení

- bubble sort
 - postupně projde všechny prvky, vždy porovná sousední 2, pokud jsou seřazené špatně tak je prohodí
 - tyto průchody všemi prvky jsou opakovány dokud se alespoň jednou za průchod něco prohodí
 - in-place algoritmus (nevýžaduje paměť navíc)
 - složitost $O(n^2)$
 - stabilní algoritmus (pořadí prvku se stejnými klíči zůstane stejné)
 - citlivý na vstupní data (doba běhu ovlivněna pořadím dat na vstupu)
- select sort
 - rozděluje prvky na seřazené a neseřazené, z neseřazených vybere nejmenší a prohodí ho s prvním okresem v neseřazené části, tím rozšíří seřazenou část o 1
 - in-place algoritmus (nevýžaduje paměť navíc)
 - složitost $O(n^2)$
 - údajně nestabilní, asi je tím myšleno že závisí na implementaci vybírání prvku z neseřazené části
 - necitlivý na vstupní data (doba běhu není ovlivněna pořadím dat na vstupu)
- insert sort
 - opačný postup než u select sortu — vezme první prvek neseřazené části, a vloží ho na správné místo v seřazené části (na počátku seřazená část = první prvek)
 - in-place algoritmus (nevýžaduje paměť navíc)
 - složitost $O(n^2)$
 - stabilní (řekl bych ale že také závisí na implementaci)

- citlivý na vstupní data
- merge sort
 - rekurzivní algoritmus — rozdělí prvky napůl, seřadí poloviny svým rekurzivním zavoláním, následně seřazené poloviny sloučí
 - out-of-place algoritmus (vyžaduje paměť navíc kvůli slučování do nového pole, s rozumnou implementací stačí $O(n)$ paměti navíc)
 - složitost $O(n \cdot \log n)$ (ale celkem vysoká multiplikativní konstanta)
 - necitlivý na vstupní data
- quick sort
 - vybere se prvek zvaný pivot (snaha je, aby byl co nejblíže mediánu všech prvků, závisí na tom efektivita celého algoritmu)
 - nalevo od pivota se dají prvky menší jak on, napravo větší
 - na levou i pravou část se rekurzivně použije znova stejný proces
 - takto se postupuje, dokud se nedojde na velikost samostatných prvků
 - složitost $O(n \cdot \log n)$, teoreticky v nejhorském případě $O(n^2)$ (citlivý na vstupní data)

Dolní mez složitosti řazení

- uvažujeme řazení v porovnávacím modelu
 - prvky se mezi sebou mohou pouze porovnávat a přesouvat, není např. žádné omezení na rozsah hodnot
 - prohození 2 prvků i porovnání je $O(1)$
- požadujeme deterministický algoritmus, tedy musí postupovat na základě dat a předchozích akcí, ne náhodně
- při takových požadavcích je řazení ekvivalentní rozpoznání, o kterou permutaci z $n!$ možných jde — hloubka nejméně $\log(n!)$, DML magí převedeno na $\Omega(n \log n)$
- pro dolní mez vyhledávání podobný postup — (ternární) strom možností s hloubkou max $\log_3 n$

Řazení v lineárním čase

- je možné dosáhnout, pokud nepracujeme v klasickém porovnávacím modelu
- algoritmus counting sort
 - dosahuje lineární složitosti díky předpokladu omezené vstupní množiny hodnot
 - řadí n celých čísel z množiny např. $\{1, \dots, r\}$
 - pro každé číslo se spočte histogram (kolikrát je ve vstupu obsaženo)
 - z histogramu se spočítají pozice prvků ve výsledném poli, a v druhém průchodu se přesunou na správná místa
 - složitost $\Theta(n + r)$ (jak paměťová, tak časová)
 - necitlivý (datově) a není in-place
 - stabilní

1.21 SP-21 (AG1 + PA1)

Rekurzivní rozklad problému na podproblémy metodou Rozděl-a-panuj. Rekurze vs iterace. Dynamické programování.

Rozděl a panuj

- rekurzivní algoritmus je postup řešení problému, při kterém:
 - se stejný postup aplikuje na jednu nebo více částí vstupních dat
 - se poskytne přímé řešení triviální instance problému
 - řešení celého problému se sestaví z podproblémů
- rozděl a panuj je metoda řešení problému rozložením na podproblémy řešené stejným způsobem
- rekurze se typicky používá na rekurzivně definovaných datových strukturách jako zakořeněné stromy
- konkrétní problémy:
 - merge sort
 - karacubův algoritmus rychlého násobení čísel
 - quickselect

Rekurze vs iterace

- rekurze je takový způsob programování, kde funkce volá sebe sama
- rekurzivní volání typicky řeší menší instanci problému
- je nutné si ohlídat ukončovací podmínky, aby se zanořování někdy zastavilo
- iterace oproti tomu provádí kód v cyklu, nezanořuje se, nezabírá tolik místo na zásobníku
- rekurze bývá kratší, přehlednější, líp umí napodobit okolní svět (opakování problémů)
- rekurze lze vždy převést na iteraci

Dynamické programování

- další technika návrhu algoritmů založená na rekurzivním rozkladu problému na podproblémy
- využívá opakování řešení podproblémů
- podmínkou pro využití je právě opakovaný výskyt stejných podproblémů při rekurzivním rozkladu
- výsledky vyřešených podproblémů se ukládají, při příštém výskytu se použije již hotový výsledek (memoizace)
- typický postup:
 - máme rekurzivní algoritmus, typicky s exponenciální složitostí
 - odhalíme opakované podproblémy
 - vytvoření tabulky pro zápis řešení podproblémů
 - v rekurzivních voláních nejprve zkонтrolujeme zda již problém není vyřešen, a pokud ne tak vypočítáme, jinak vrátíme již hotovou hodnotu
 - tabulka často lze plnit i iterativně, to může být rychlejší než rekurze
- příklady využití:
 - fibonacciho čísla
 - * začíná se rekurzivně od n -tého fibonacciho čísla (to chceme spočítat)
 - * rekurzivně zavoláme výpočet $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$
 - * do tabulky ukládáme postupné výsledky odspoda ($F(1), F(2), \dots$)
 - nejdělsí rostoucí podposloupnost

- * máme pole čísel, chceme podposloupnost takovou aby byla rostoucí a zároveň co nejdelší
 - * zavoláme na každý prvek funkci, která nám vrátí nejdelší podposloupnost z tohoto prvku
 - * v daném prvku se postouně vyzkouší všechny následující, a vždy pokud je konkrétní další prvek větší, opět se pro něj zavolá stejná funkce
 - * výsledky postupně ukládáme pro každý prvek v poli (je zbytečné to pro stejný prvek počítat víckrát, výsledek bude stejný)
- editační vzdálenost 2 řetězců
- * zavoláme funkci na začátek, která vyřeší aktuální znak a zavolá sama sebe na zbytek řetězce
 - * postupně ukládáme výsledky pro různé varianty změny v konkrétním znaku

1.22 SP-22 (PA2)

Objektově orientované programování v C++, zapouzdření, dědičnost, atributy a metody, statické atributy a metody, virtuální metody, polymorfismus, abstraktní třídy, výjimky, šablony, přetěžování operátorů, mělká a hluboká kopie.

Objektové programování obecně

- základem je objekt
- objekt je obdoba proměnné z neobjektového programování
- obdobou datového typu je třída
- objekt má nějaký vnitřní stav, reprezentovaný hodnotami atributů (členských proměnných)
- operace nad objekty se nazývají metody (členské funkce) a jsou stejně jako atributy definované ve třídě
- metody se dají chápát jako služby/rozhraní poskytované uživateli
- OOP vzniklo snahou popsat a modelovat okolní svět v počítači
- OOP urychluje vývoj a zjednoduší znovupoužitelnost odladěného kódu

Zapouzdření

- lze specifikovat přístup k atributům a metodám
- public — přístupné zvenčí třídy
- private — přístupné pouze zevnitř konkrétní třídy (a z friend funkcí)
- protected — podobné private, ale dostupné navíc potomků třídy

Dědičnost

- mezi třídami může být vztah rodič — potomek
- potomek od rodiče může dědit atributy a metody
- stejně tak jako každá třída si může vytvořit svoje atributy a metody
- navíc může přepsat metodu předka

Statické metody a atributy

- nejsou součástí objektu, ale třídy jako takové (instanční vs třídní)
- musí se uložit mimo objekty
- statické metody nemohou používat nestatické položky

Virtuální metody

- pro každou třídu existuje tabulka virtuálních metod
- podle VMT (virtual method table) se za běhu dle typu objektu vybere, jaká metoda se má použít (z aktuálního typu, nebo podtřídy)
- aby šlo této dynamické vazby využít, musí být objekt referencován ukazatelem, přímo to nejde

Polymorfismus

- ukazatel na nadtřídu může ukazovat i na podtřídu
- objekt se chová podle toho, které třídy je instancí
- tedy se volají správně metody podtřídy a ne nadtřídy
- volaná metoda se určí až v okamžiku volání

Abstraktní třídy

- obsahují alespoň jednu abstraktní metodu
- nemůže mít instance
- abstraktní metoda je pouze deklarovaná, není definovaná (až v podtrídě)

Výjimky

- způsob ošetření chyb za běhu programu v C++
- klíčové slovo "throw"
- pro efekt musí být "testovaný" kód v "try" bloku, v catch bloku se následně případná výjimka zpracuje
- při vyhození výjimky se vykonávání kódu zastaví, a postupně se skáče nahoru a čistí zásobník (a volají destruktory objektů) k nejbližšímu "ovladači" výjimek

Šablony

- parametrizovaná deklarace (a definice) funkce/třídy, ze které komplilátor dosazením za parametry vytvoří instanci šablony
- parametr T šablony může být typ argumentů, typ proměnných, návratových hodnot...

Přetěžování operátorů

- operátory je možné přetížit podobně jako funkce/metody
- lze přetížit (friend) funkcemi, nebo metodami na objektech/třídách
- klíčové slovo "operator"
- nemění se priorita ani asociativita operátorů, ale mění se typy operandů a sémantika operací

Mělká a hluboká kopie

- mělká kopie
 - kopie objektu obsahuje ukazatele/reference na stejná data
 - je nutné vyřešit destruktor, aby data uvolnil až je nic nebude používat
 - lze řešit počítáním referencí
- hluboká kopie
 - kopie objektu si zkopíruje všechna dynamicky alokovaná data na nové místo
 - při destrukci objektu se data vždy uvolní
 - data se v kopiích po čase mohou lišit
- často, když objekt obsahuje přímo dynamicky alokovaná data (ne v dalším objektu, který to má vyřešené jako vector), tak se hodí vytvořit vlastní operátor přiřazení, kopírující konstruktor a destruktor

1.23 SP-23 (PA2)

Abstraktní datový typ, jeho specifikace a implementace. Zásobník, fronta, pole, seznam, tabulka, množina. Implementace pomocí spojových struktur, stromů a pole.

Klasické datové typy

- specifikují množinu hodnot
- specifikují množinu operací s hodnotami
- mají jasně stanovenou implementaci, tedy vnitřní reprezentaci hodnot a realizaci operací
- dělíme na:
 - jednoduché/primitivní — nedělitelné/atomické hodnoty
 - strukturované — skládají se ze složek
 - ukazatele — adresy do paměti, závislé na typu na který ukazují
 - jiné — např. funkce

Abstraktní Datový Typ (ADT)

- specifikuje množinu hodnot a množinu operací nezávisle na implementaci
- syntax je popsána signaturou operace — popis operace jako např. " $_ - + _ : \text{int}, \text{int} \rightarrow \text{int}$ "
- sémantika je popsána pomocí axiomů, slovně nebo jinak

Zásobník (stack)

- kontejner, kde nejnovější prvky jsou odebrány první (LIFO)
- rozhraní: $\text{push}(s, x)$, $\text{pop}(s)$, $\text{top}(s)$, $\text{isEmpty}(s)$
- implementace:
 - pomocí pole — má omezení, bud' musí být konstantní velikost, nebo v případě nafukovacího může být $\text{push}()$ lineární
 - pomocí spojového seznamu

Fronta (queue)

- kontejner, kde nejstarší prvky jsou odebrána první (FIFO)
- rozhraní: $\text{push}(q, x)$, $\text{pop}(q)$, $\text{front}(q)$, $\text{isEmpty}(q)$
- implementace:
 - pomocí pole — má omezení, kromě problémy s velikostí je nutné pole "protočít" při operaci $\text{pop}()$, což je opět lineární
 - pomocí spojového seznamu

Pole

- kontejner, kde lze ke všem prvkům přistupovat přímo v čase $O(1)$
- může být vícerozměrné (pak ale problémy s nafukováním)
- implementace:
 - pomocí jednorozměrného pole (lze i více dimenzí, např. ukládat řádky za sebe)
 - pomocí vícerozměrného pole

Seznam (list)

- kontejner, kde lze přistupovat ke všem prvkům
- implementace:
 - (obousměrný) spojový seznam (pozor, přístup a vkládání jinam než na konec/konce seznamu je lineární)

Množina (set)

- kontejner, který obsahuje prvky bez duplikátů
- implementace:
 - seřazené/neseřazené pole — fuj, bylo by to pomalý i na seřazeném
 - pomocí spojového seznamu (seřazeného) — fuj
 - binární vyhledávací strom
 - hash table

Tabulka (asi mapa?)

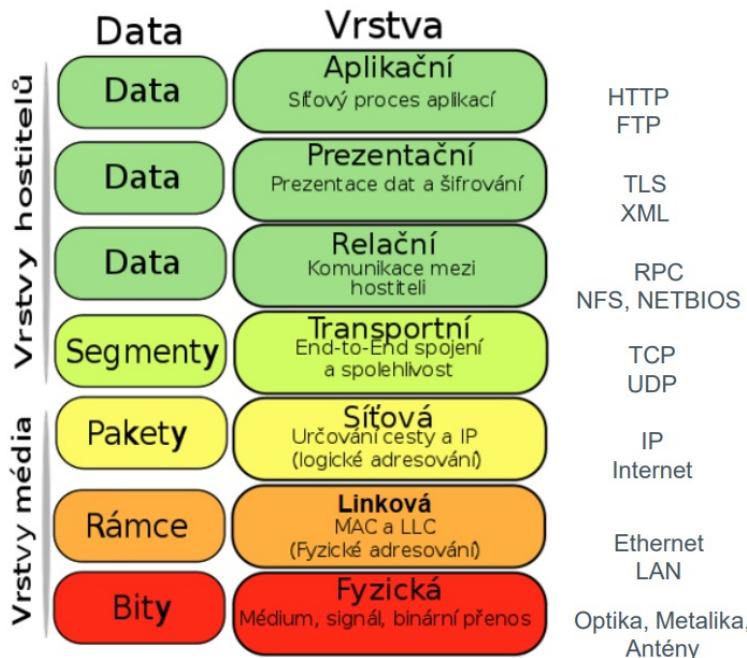
- kontejner, který obsahuje data identifikovaná klíčem
- chová se jako množina, která má pod klíči navíc data
- implementace:
 - pole (klíče == indexy v poli)
 - pole (seřazené dle klíčů, obsahuje páry key-value)
 - spoják (seřazený dle klíčů, obsahuje páry key-value)
 - hash table
 - binární vyhledávací strom

1.24 SP-24 (PSI)

ISO/OSI model, enkapsulace a dekapsulace posílaných dat, princip IP adresace. Linková vrstva, podvrstvy LLC, MAC, síťová zařízení, princip přepínání. Virtuální sítě (VLAN). Síťová vrstva, směrovače, princip směrování, protokoly IPv4 a IPv6, statické a dynamické směrování.

OSI Model

- Open System Interconnection



- Aplikační vrstva — protokoly pro komunikaci mezi aplikacemi, přenáší se data a zajímá nás význam
- Prezentační vrstva — formátování a prezentace dat, šifrování, přenáší se data a zajímá nás struktura
- Relační vrstva — logické rozhraní pro aplikace, RPC, sdílení file systému, přenáší se data
- Transportní vrstva — data jsou rozložena na segmenty, typicky protokol transportní vrstvy přidá nějakou hlavičku, posílá se z portu na port, řeší se E2E spojení a spolehlivost
- Síťová vrstva — segmenty vyšších vrstev rozděleny na pakety, posílají se na síťovou adresu (IP)
- Linková vrstva — pakety rozděleny na rámce (frames), opět přidána hlavička, posílá se na linkovou adresu (MAC)
- Fyzická vrstva — převod rámců na bity a posílání přes médium (kabel, vzduch)

TCP/IP model

- pouze 4 vrstvy: aplikační (první 3 OSI), transportní, internetová a síťová (spodní 2 OSI)

IP adresace

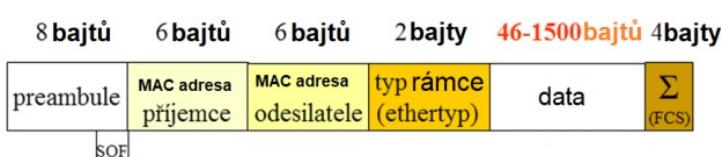
- IPv4 adresa — 4 byty
- stanice v IPv4 síti se sdružují do podsítí
 - adresní rozsah sítě — skupina všech IP adres, které patří do stejné sítě
 - maska — určuje rozsah, je stejně dlouhá jako IP adresa, používá se prefixová notace (aaa.bbb.ccc.ddd /mm — mm je počet bitů masky od začátku, které jsou 1)
 - adresa sítě — získá se z adresy nějakého zařízení v síti operací AND s maskou — nejnižší adresa v daném rozsahu
 - broadcast — nejvyšší adresa z rozsahu

Linková vrstva

- přenášení dat v rámci lokální (LAN) sítě
- základní přenášená jednotka je rámc (frame)
- skládá se z 2 podvrstev: MAC a LLC
- MAC:
 - Medium Access Control
 - zajišťuje přístup k médiu (fyzické vrstvě)
 - řeší fyzickou adresaci pomocí MAC adres
 - filtrování MAC adres
 - plánování rámců do front a jejich odesílání
 - VLAN
 - svázána s konkrétní technologií fyzické vrstvy
 - řeší sdílený přístup k médiu (multiplex — řešení kolizí — časový, frekvenční, kódový, prostorový)
 - další metody přístupu k společnému médiu — Carrier Sense Multiple Access (CSMA (-/CD/CA))



- LLC:
 - Logical Link Control
 - umožňuje existenci různých protokolů nad společnou MAC vrstvou
 - řízení toku a kontrola chyb
 - rozdelení toku dat z vyšších vrstev do rámců, určení velikosti rámce a jeho zakončení
 - zajištění doručení dat — potvrzovací schémata
 - * jednotlivé potvrzování (stop & wait)
 - * selective repeat
 - * Go-Back-N
 - * klouzavé okénko
- Zařízení na linkové vrstvě:
 - pracují s rámcí
 - obecný formát: hlavička, data, konec rámce (obsahuje např kontrolní součet)
 - switch — porty, které mají buffery — přepínací tabulka, aby věděl kam posílat dál, switch se učí — různé režimy přepravy (store and forward, cut through, fragment free)
 - bridge
- broadcastová doména — množina stanic dané sítě, kterým je doručen rámc s broadcastovou adresou
- v síti nesmí být smyčky, lze odstranit přes SPT (spanning tree protocol), udělá kostru
- MAC adresa: 6 bytů, první 3 identifikují výrobce



VLAN

- každý port na switchi se dá zařadit do VLAN
- trunk port podporuje více (všechny) VLAN
- trunkem chodí označené rámce (tagged), k workstations chodí neoznačené
- pro přeposílání dat mezi VLAN se musí chodit přes router
- VLAN může být dán jak portem, tagem, tak MAC adresou nebo protokolem vyšší úrovně

Síťová vrstva

- doručuje data jak v lokální síti, tak mezi sítěmi
- schémata IP adresace: prefixová a dle třídy

Class	HOB	NET ID Bits	Host ID Bits	No of Networks	Host Per Network	Start Address	End Address	Mask
Class A	0	8	24	$2^7=128$	$2^{24-8}=16,777,216$	0.0.0.0	127.255.255.255	255.0.0.0
Class B	10	16	16	$2^{14}=16,384$	$2^{16-16}=65,536$	128.0.0.0	191.255.255.255	255.255.0.0
Class C	110	24	8	$2^{21}=2,097,152$	$2^{8-24}=256$	192.0.0.0	223.255.255.255	255.255.255.0
Class D	1110	-	-	-	-	224.0.0.0	239.255.255.255	
Class E	1111	-	-	-	-	240.0.0.0	255.255.255.255	

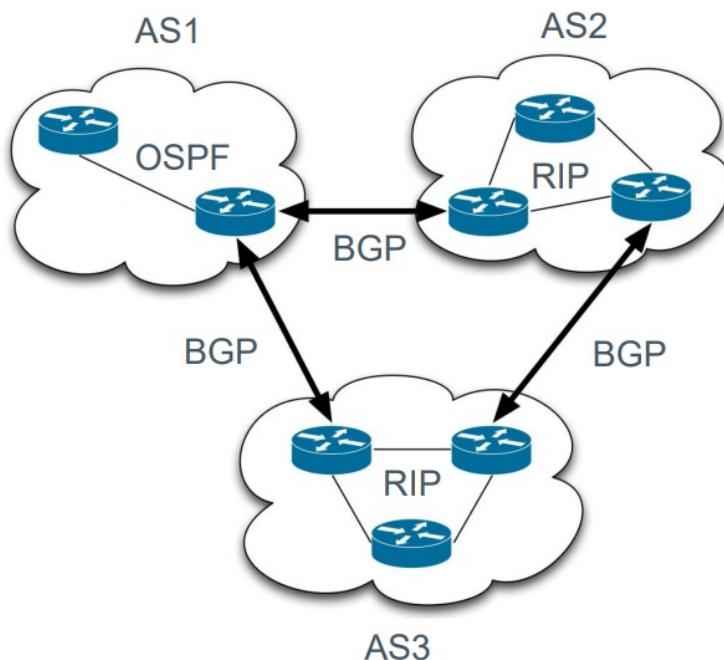
- speciální rozsahy:
 - privátní (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16)
 - link-local (169.254.0.0/16)
 - loop-back (127.0.0.0/8)
 - multicast (224.0.0.0/4)
- hlavička IP paketu:

Bajty	0	1	2	3
Bajt 0 až 3	verze	IHL	typ služby	celková délka
Bajt 4 až 7	identifikace		příznaky (3 bity)	offset fragmentu (13 bitů)
Bajt 8 až 11	TTL	číslo protokolu		kontrolní součet hlavičky
Bajt 12 až 15			zdrojová adresa	
Bajt 16 až 19			cílová adresa	
Bajt 20 až ((IHL × 4) - 1)		rozšířená nepovinná nastavení		
...			data	

- princip směrování:
 - směrování (routing) — odeslání konkrétního paketu prostřednictvím zvoleného síťového rozhraní na základě cílové IP adresy paketu
 - provádí router
 - podle toho, zda cílová adresa patří do sítě routeru se paket bud' pošle cílové stanici, nebo dalšímu routeru podle směrovací tabulky
 - směrovací tabulka má záznamy o sítích, a jakým způsobem se k nim dostat (kam a jak poslat paket)
 - ruční zadání informací do tabulky = statické směrování
- NAT (Network Address Translation) — překlad adres na routeru z veřejné (adresa routeru) na privátní (za routerem, adresy v soukromé síti)
- ICMP (Internet Control Message Protocol) — utility ping a tracert/traceroute
- ARP (Address Resolution protocol) — chceme poslat paket na IP adresu, ale neznáme MAC adresu, tak se zeptáme přes MAC broadcast "kdo má tuto IP" ... cíl odpoví už ne přes broadcast že on má tuto adresu, tedy uložíme do ARP table
- DHCP (Dynamic Host Control Protocol) — dynamické nastavení IP adresy a nastavení sítě pro stanici (4 zprávy, vše broadcast)

Směrování

- typy směrování:
 - Redundantní — existuje více cest do cíle
 - symetrické — cesta tam je stejná jako zpět
 - asymetrické — cesty tam a zpět se liší
 - proaktivní — používá směrovací tabulky, předpočítá/ví cestu do cíle (běžně v počítačových sítích)
 - reaktivní — zjišťuje cestu až na základě žádosti (např. v mobilních sítích)
- Dynamické směrování — algoritmy:
 - Distance Vector algoritmy (např. RIP — routing information protocol) — routery si vyměňují navzájem info o vzdálenosti všech sousedů, dělají to i za běhu, tedy + router i - router, vzdálenost je pouze počet routerů v cestě
 - Link-State algoritmy (např. OSPF) — sestaví graf sítě i s ohodnocením hran (linkstate udaná správcem sítě)
 - Path Vector algoritmy (např. BGP) — routery si vyměňují celé cesty ke konkrétním cílům



IPv6

- umožňuje zřetězit hlavičky, hlavička je jednodušší
- 128b adresy (16B)
- není potřeba NAT
- umožňuje objevování sousedů, efektivnější seměrování, ...

1.25 SP-25 (PSI)

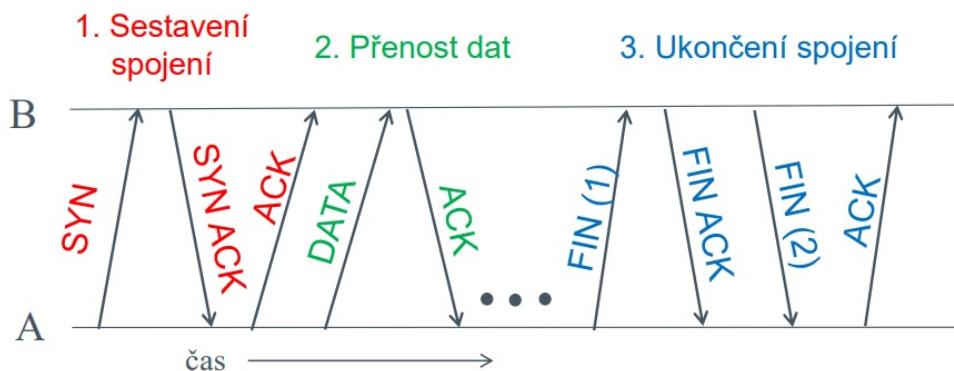
Transportní vrstva, protokol TCP, spolehlivost doručení paketů, zahlcení, srovnání s protokolem UDP. Překlad síťových adres (NAT) na síťové a transportní vrstvě. Systém doménových jmen (DNS).

Transportní vrstva

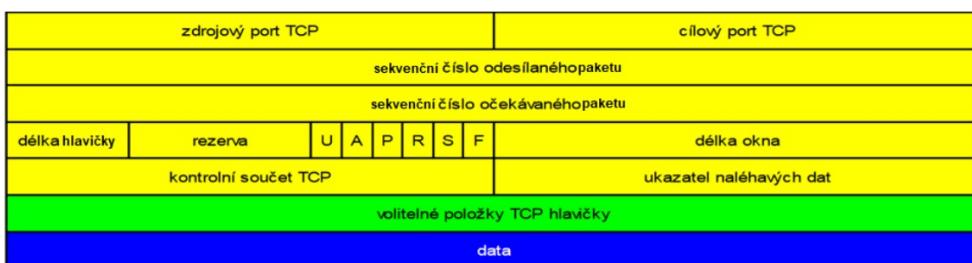
- zajišťuje doručení dat konkrétním aplikacím (přidává k IP adrese ještě port)
- řeší spolehlivost dodání dat
 - možnost stop and wait — jednoduchá, ale neefektivní, doručuje data přesně v pořadí jak jdou za sebou
 - možnost stop and go — nezajistí, že data dorazí do cíle, přijímač posílá vysílači stop a go signály
 - sliding window — okénko o nějaké velikosti, první se odesílá, na poslední se čeká na ACK

TCP

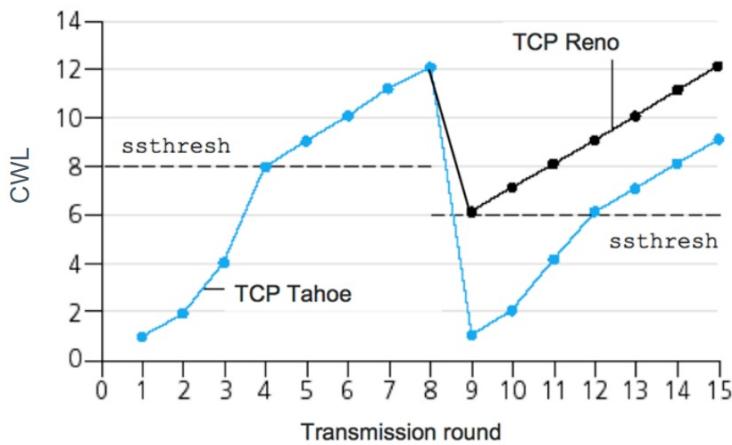
- nejznámější protokol transportní vrstvy, který zajišťuje spolehlivé doručení dat
- garantuje jak doručení, tak pořadí paketů
- je duplexní, obě strany jsou vysílačem i přijímačem
- navazování a ukončování spojení:



- struktura TCP paketu



- pro řízení toku se používá mechanismus klouzavého okénka (sliding window) — přijímač nastavuje délku okénka dle svých možností
- vysílač ještě musí nějak zajistit, aby se nepřetížila linka, to je složitější a liší se to dle varianty TCP — nastavuje se CWL (congestion window length)
- některé druhy obnovují rychlosť (velikost CWL) po výpadku rychleji, jiné pomaleji



UDP

- nižší režie — vyšší propustnost než TCP
- běžně se používá na přenos hlasu či videa
- struktura UDP paketu:

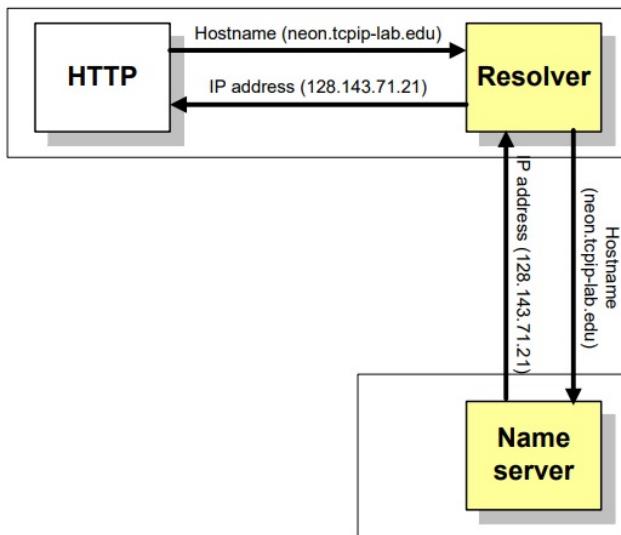
binární nuly	protokol vyšší vrstvy	celková délka IP paketu
	zdrojový port UDP	cílový port UDP
délka dat		kontrolní součet UDP hlavičky
	data	

NAT

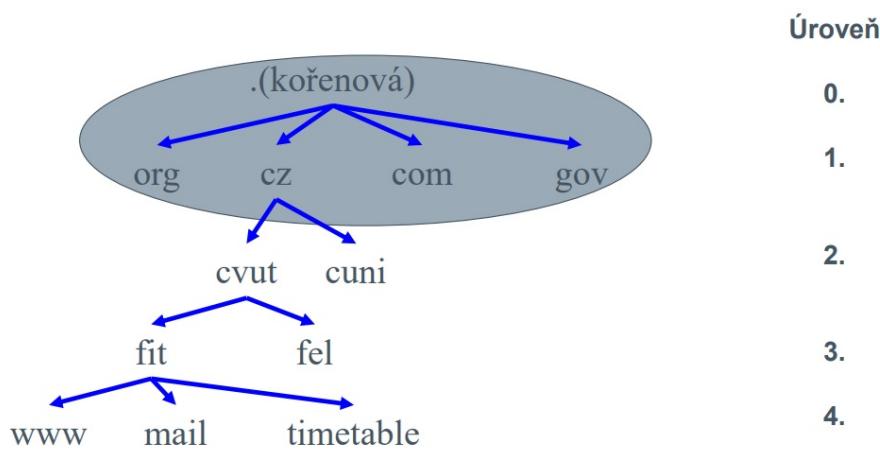
- díky transportní vrstvě a portům je možné na routeru překládat IP adresy pro více spojení a stanic za routerem
- směrem ven se přeloží IP adresa, pokud je to nutné tak i port
- tím se vytvoří mapování na routeru — tento výstupní port je mapován ke konkrétní IP a portu za routerem
- při odpovědi se dle portu rozhodne na jakou IP (a port) data poslat

DNS

- IP adresy jsou špatně zapamatovatelné
- DNS — Domain Name System je primárně určen k překladu doménových jmen na IP adresy a naopak
- DNS je hierarchický a nezabezpečený
- DNS je zásadní i pro další služby jako email
- běží typicky na portu 53 a většina implementací využívá UDP



- hierarchie domén:



- 4 typy Top Level Domains (TLD) (1. úrovni)
 - Generic TLD (gTLD) — 3 znaky označující funkci organizace (gov, mil, edu, org, com, net)
 - Country Code TLD — 2 znaky označující zemi (cz, us, de, eu)
 - New Generic TLD — libovolný řetězec (max 63 znaků)
 - reverzní domény — in-addr.arpa
- dříve centralizovaný systém všech domén v souboru hosts.txt
- existuje iterativní způsob dotazování (dotaz lokálnímu DNS serveru, ten se postupně od root zeptá všech nutných dalších DNS serverů) nebo rekurzivní (zeptají se v řadě za sebou, lokální rootu, root TLD atd)
- existuje DNSSEC — zabezpečený DNS (přes asymetrickou kryptografií)
- existuje dynamické DNS (DynDNS) — pro případ zařízení, která často mění IP adresu

1.26 SP-26 (PST)

Pravidla pro výpočty pravděpodobností, Bayesův vzorec. Náhodné veličiny, příklady rozdělení, distribuční funkce, hustota, momenty. Nezávislost náhodných jevů a veličin. Centrální limitní věta, zákony velkých čísel.

Náhodný experiment

- je reprezentován pravděpodobnostním prostorem (Ω, \mathcal{F}, P)
- Ω je množina možných výsledků (elementárních jevů)
 - vzájemně exklusivní — mělo by být vždy jednoznačné, který elementární jev nastal
 - v souhrnu vyčerpávající — vše co se může stát je popsáno elementárními jevy
- \mathcal{F} je systém podmnožin Ω s rozumnými vlastnostmi (σ -algebra)
 - \mathcal{F} obsahuje nemožný jev
 - pokud $A \in \mathcal{F}$, tak \mathcal{F} obsahuje i opačný jev (značeno A^c jako doplněk)
 - obsahuje spočetné sjednocení jevů
- prvky $A \in \mathcal{F}$ jsou náhodné jevy
- pravděpodobnostní míra P je funkce, která přiřazuje náhodným jevům reálné číslo z intervalu $\langle 0, 1 \rangle$, reprezentující ideální podíl případů, kdy jev nastává

Podmíněná pravděpodobnost

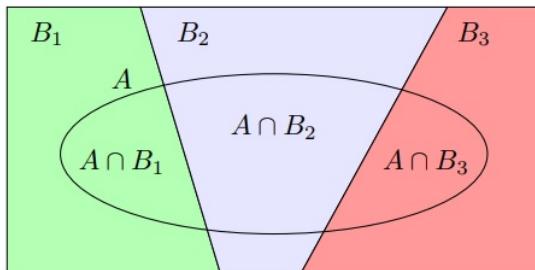
Pravděpodobnost jevu za předpokladu, že nastal jiný jev.

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Bayesova věta (prohození jevu a podmínky)

Pozorujeme jev A a ptáme se, jaká je pravděpodobnost, že nastala možnost B_j .

$$\Omega = B_1 \cup B_2 \cup B_3 \text{ (disjunktní rozklad)}$$



Připomenutí:

$$P(A \cap B_j) = P(A|B_j) P(B_j)$$

$$P(A) = P(A|B_1) P(B_1) + P(A|B_2) P(B_2) + P(A|B_3) P(B_3)$$

$$P(B_j|A) = \frac{P(A \cap B_j)}{P(A)}$$

$$P(B_j|A) = \frac{P(A|B_j) P(B_j)}{P(A|B_1) P(B_1) + P(A|B_2) P(B_2) + P(A|B_3) P(B_3)}$$

Nezávislé jevy

- Jevy A a B jsou nezávislé, pokud $P(A \cap B) = P(A) \cdot P(B)$.
- Nezávislé jevy se neovlivňují — $P(A|B) = P(A)$, $P(B|A) = P(B)$

Náhodné veličiny

Pro matematické zpracování se každému výsledku $\omega \in \Omega$ přiřadí číslo. Přiřazení čísla vybíráme dle potřeby, podle toho co zrovna zkoumáme.

- Náhodná veličina X na pravděpodobnostním prostoru (Ω, \mathcal{F}, P) je funkce, která každému výsledku experimentu $\omega \in \Omega$ přiřadí hodnotu $X(\omega) \in \mathbb{R}$ a pro kterou platí podmínka měřitelnosti $\{X \leq x\} \in \mathcal{F}, \forall x \in \mathbb{R}$.
- Distribuční funkce náhodné veličiny X je definovaná vztahem $F_X(x) = P(X \leq x)$.
- Diskrétní náhodné veličiny mají pravděpodobnostní funkci, spojité hustotu (vždy derivace distribuční funkce).

Střední hodnota

- střední hodnota diskrétních veličin:

$$EX = \sum_k x_k P(X = x_k)$$

- střední hodnota spojitých veličin:

$$EX = \int_{-\infty}^{+\infty} xf(x)dx$$

Rozptyl

- Rozptyl (variance):

$$\text{var}X = E[(X - EX)^2]$$

- nejlépe se vypočítá jako $\text{var}X = EX^2 - (EX)^2$

- směrodatná odchylka:

$$\text{sd}X = \sqrt{\text{var}X}$$

Momenty

Pro $k \in \mathbb{N}$ definujeme:

- k -tý moment náhodné veličiny X :

$$\begin{aligned}\mu_k &= E(X^k) = \sum_{\text{all } x_i} x_i^k P(X = x_i) \text{ pro } X \text{ diskrétní} \\ \mu_k &= E(X^k) = \int_{-\infty}^{+\infty} x^k f_X(x)dx \text{ pro } X \text{ diskrétní}\end{aligned}$$

- k -tý centrální moment veličiny X :

$$\begin{aligned}\sigma_k &= E(X - EX)^k = \sum_{\text{all } x_i} (x_i - \mu_1)^k P(X = x_i) \text{ pro } X \text{ diskrétní} \\ \sigma_k &= E(X - EX)^k = \int_{-\infty}^{+\infty} (x - \mu_1)^k f_X(x)dx \text{ pro } X \text{ diskrétní}\end{aligned}$$

Diskrétní rozdělení

- Bernoulliho (Alternativní) rozdělení

Definice

Náhodná veličina X má **Bernoulliho** (alternativní) **rozdělení** s parametrem $p \in [0, 1]$, jestliže platí:

$$P(X = 1) = p, \quad P(X = 0) = 1 - p.$$

Značení: $X \sim \text{Be}(p)$ či $X \sim \text{Bernoulli}(p)$ nebo $X \sim \text{Alt}(p)$.

$$- EX = p$$

- $\text{var}X = p(1 - p)$
- Binomické rozdelení — opakované hody, zajímá nás pravděpodobnost že právě n -krát padne např. hlava — výpočet $P(X = k) = \binom{n}{k} p^k (1 - p)^{n-k}$
 - $EX = np$
 - $\text{var}X = np(1 - p)$
- Geometrické rozdelení — opakované hody, zajímá nás pravděpodobnost že právě na k -tý pokus padne poprvé např. hlava — výpočet $P(X = k) = (1 - p)^{k-1} p$
 - $EX = \frac{1}{p}$
 - $\text{var}X = \frac{1-p}{p^2}$
- Poissonovo rozdelení — zajímá nás např počet jevů za určitý čas, je to approximace, limita binomického do nekonečna

Spojitá rozdelení

- rovnoměrné rozdelení
- exponenciální rozdelení
- normální rozdelení

Nezávislost náhodných veličin

Náhodné veličiny X a Y nazýváme nezávislé, pokud pro všechna $x, y \in \mathbb{R}$ jsou jevy $\{X \leq x\}$ a $\{Y \leq y\}$ nezávislé. Tedy pokud platí:

$$P(X \leq x \cap Y \leq y) = P(X \leq x) \cdot P(Y \leq y)$$

Lze také vyjádřit pomocí hustot/pravděpodobnostních funkcí:

$$P(X = x \cap Y = y) = P(X = x) \cdot P(Y = y)$$

Limitní věty

Místo samotných náhodných veličin řešíme jejich posloupnosti, které vznikly opakováním experimentu. Co nás typicky zajímá:

- průměr (aritmetický):

$$\bar{X}_n = \frac{1}{n} \sum_{i=1}^n X_i$$

- součet:

$$S_n = \sum_{i=1}^n X_i$$

kde X_1, \dots, X_n jsou nezávislé náhodné veličiny se stejným rozdelením (I.I.D.).

Zákony velkých čísel

- Slabý zákon velkých čísel: X_1, \dots, X_n jsou i.i.d. s konečnou střední hodnotou $EX_i = \mu$ a konečným rozptylem σ^2 . Potom \bar{X}_n konverguje k μ v pravděpodobnosti:

$$\bar{X}_n \xrightarrow{P} \mu \quad \text{pro} \quad n \rightarrow \infty$$

Jinak řečeno, pro $n \rightarrow \infty$ je pravděpodobnost nenulové vzdálenosti průměru od μ nulová.

- Silný zákon velkých čísel: X_1, \dots, X_n jsou i.i.d. se střední hodnotou $EX_i = \mu$. Potom \bar{X}_n konverguje k μ skoro jistě (s pravděpodobností 1).

CLV — Centrální limitní věta

Nechť X_1, X_2, \dots je posloupnost i.i.d. náhodných veličin s konečnou střední hodnotou $\text{E}X_i = \mu$ a konečným rozptylem $\text{var}X_i = \sigma^2 > 0$. Potom

$$\frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}} \xrightarrow{\mathcal{D}} N(0, 1) \quad \text{pro } n \rightarrow \infty$$

Podobně

$$\frac{S_n - n\mu}{\sigma\sqrt{n}} \xrightarrow{\mathcal{D}} N(0, 1) \quad \text{pro } n \rightarrow \infty$$

1.27 SP-27 (PST)

Základy matematické statistiky, náhodný výběr, bodové odhady pro střední hodnotu a rozptyl, intervalové odhady pro střední hodnotu, testování statistických hypotéz o střední hodnotě.

Úvod

- na základě skutečných výsledků/dat vybíráme pravděpodobnostní model (tedy opačný přístup než u pravděpodobnosti)
- nejprve odhadneme tvar rozdělení dle kontextu a zkušenosti
- odhadneme parametry rozdělení
 - bodové odhady — určení nejpravděpodobnější hodnoty parametru
 - intervalové odhady — určení intervalu, kde parametr leží s vysokou pravděpodobností
- ověření správnosti modelu — testování hypotéz

Bodové odhady

Chceme, aby odhady byly nestranné (střední hodnota odhadu byla rovna reálné hodnotě) a konzistentní (pro nekonečno pokusů se odhad musí rovnat reálu) Nejčastější bodové odhady:

- výběrový průměr — bodový odhad střední hodnoty
- výběrový rozptyl — bodový odhad rozptylu
- výběrová směrodatná odchylka
- k -tý výběrový moment
- výběrová kovariance
- výběrový korelační koeficient

Metoda momentů:

- máme rozdělení určené d -rozměrným parametrem
- musíme si tedy vypočítat prvních d teoretických momentů
- vyjádříme parametry jako funkce momentů
- odhadneme teoretické momenty pomocí výběrových momentů
- odhad rozptylu a střední hodnoty:

- první 2 momenty jsou:

$$EX_i = \mu, \quad EX_i^2 = \text{var}X_i + (EX_i)^2 = \sigma^2 + \mu^2$$

- parametry lze vyjádřit jako funkce momentů: $\mu = EX_i, \sigma^2 = EX_i^2 - (EX_i)^2$
- odhadneme EX_i pomocí m_1 a EX_i^2 pomocí m_2

Metoda maximální věrohodnosti:

- vytvoříme věrohodnostní funkci
- jako hodnotu parametru použijeme takovou hodnotu, která maximalizuje věrohodnostní funkci

Intervalové odhady

- Konfidenční intervaly / intervaly spolehlivosti pro konkrétní parametr θ zkoumaného rozdělení jsou takové meze L a U , pro které

$$P(L < \theta < U) = 1 - \alpha$$

- oboustranný konfidenční interval pro střední hodnotu, pokud známe rozptyl, lze nalézt jako:

$$(\bar{X}_n - Z_{\alpha/2} \frac{\sigma}{\sqrt{n}}, \bar{X}_n + Z_{\alpha/2} \frac{\sigma}{\sqrt{n}})$$

kde Z značí kritickou hodnotu standardního normálního rozdělení

- pokud neznáme rozptyl, dosadíme výběrovou směrodatnou odchylku a místo normálního rozdělení použijeme Studentovo t -rozdělení t_{n-1}

Testování hypotéz

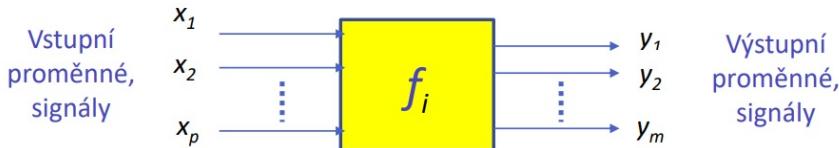
- máme k dispozici náhodný výběr, chceme ověřit nějaké tvrzení o rozdělení dat
- tvrzení = hypotéza
- testujeme nulovou hypotézu H_0 oproti alternativní hypotéze H_A , výsledkem je potvrzení/zamítnutí H_0
- chyba prvního druhu — zamítneme, přestože platí
- chyba druhého druhu — nezamítneme, přestože neplatí
- typicky máme pod kontrolou pouze jednu z předchozích chyb
- chceme, aby pravděpodobnost chyby prvního druhu byla nejvýše hladině významnosti testu α
- síla testu je $1 - P(\text{chyba druhého druhu})$
- za nulovou hypotézu volíme tu, jejíž neoprávněné zamítnutí je závažnější
- zamítnutí H_0 je silný výsledek
- postup:
 - typicky chceme prozkoumat, zda nějaký parametr rozdělení θ odpovídá nějaké udávané/zkoumané hodnotě
 - $H_0 : \theta = \theta_0$
 - $H_A : \theta \neq \theta_0$
 - sestavíme konfidenční interval pro θ na základě náhodného výběru
 - otestujeme, zda $\theta_0 \in (L, U)$
 - pokud ne, H_0 zamítáme, jinak nezamítáme

1.28 SP-28 (SAP)

Kombinační a sekvenční logické obvody (Mealy, Moore), popis a možnosti implementace na úrovni hradel. Minimalizace vyjádření logické funkce s využitím map.

Kombinační obvody

- popsány kombinační funkcí
- hodnoty všech výstupů (výstupních proměnných) jsou v každém časovém okamžiku určeny pouze vstupem (hodnotami vstupních proměnných) ve stejném okamžiku
- mohou být popsány např. Booleovskou (logickou) formulí
Příklad: $f = x_1 \cdot \bar{x}_2 + \bar{x}_1 \cdot x_2 = (x_1 + x_2) \cdot (\bar{x}_1 + \bar{x}_2)$
- obecně kombinační obvod vypadá následovně:



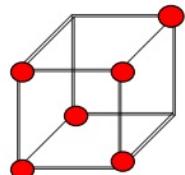
Logická funkce $y_k = f(x_1, x_2, x_3, \dots, x_p)$ existuje pro každý výstup y.

- možnosti reprezentace logických funkcí:

– tabulka

ab	f
00	0
01	1
10	1
11	0

– n-rozměrná krychle



– Booleovský výraz

Viz výše

– mapa (Karnaughova)

		b		
		a		
c		X	1	
		1	1	X

- možnosti realizace obvodů:

- na úrovni hradel
- mapování na technologii (FPGA, ASIC)
- popis v jazyku (VHDL, Verilog)

Sekvenční obvody

- výstup závisí na posloupnosti/sekvenci hodnot na vstupu
- zapamatování se realizuje zpětnou vazbou
- popsány konečným stavovým automatem

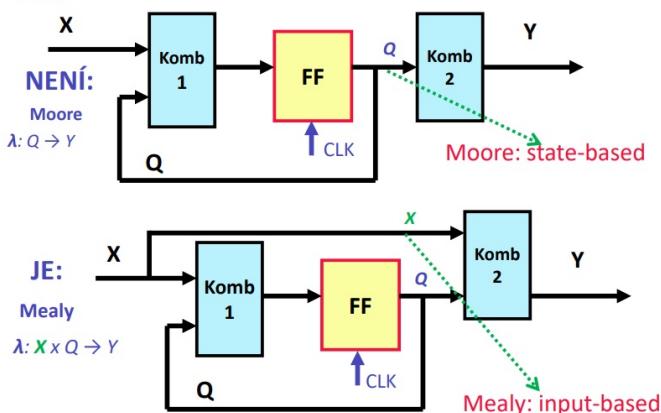
- typy sekvenčních obvodů:
 - Moore

Obvod, jehož výstup závisí pouze na vnitřním stavu.
 - Mealy

Obvod, jehož výstup závisí také na aktuálním vstupu (kromě stavu).

Mealy, Moore

Rozdíl je v definičním oboru výstupní funkce (přímá vazba ze vstupu na výstup buď není nebo je)

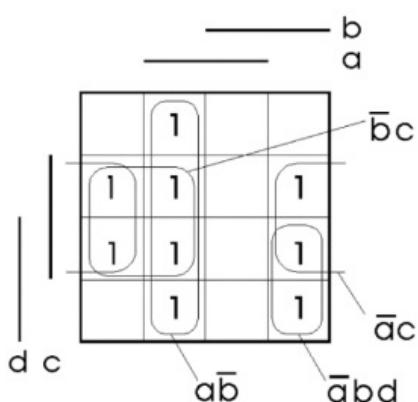


Implementace na úrovni hradel

- nejprve minimalizace logické funkce a zápis výsledku např. v Booleovském výrazu
- následně nakreslení/vytvoření funkce pomocí základních hradel — NOT, AND, OR, NAND, NOR, XOR

Minimalizace logické funkce

- smysl — zjednodušit a zkrátit zápis, snížit potřebný materiál pro výrobu
- minimalizace pomocí map — založenana hledání co největších skupin sousedních stavů.
- postup při minimalizaci:
 - vytvoření Karnaughovy mapy pro funkci
 - nalezení všech přímých implikantů (maximální skupiny jedniček či "dont care")
 - určení všech podstatných implikantů (obsahující jedničku, kterou jiný implikant neobsahuje)
 - pokud nejsou pokryty všechny vrcholy s "1", nutno vybrat další přímé implikanty (takové, kde je nejméně negací)



$$a\bar{b} + \bar{b}c + \bar{a}c + \bar{a}bd$$

- lze také kroužkovat nuly — pak je ale nutné funkci sestavit jinak.
 $(\bar{a} + \bar{b})(a + c + d)(a + b + c)$

1.29 SP-29 (SAP)

Architektura číslicového počítače, instrukční cyklus počítače, základní třídy souborů instrukcí (ISA). Paměťový substituční systém počítače, paměťová hierarchie, skrytá paměť (cache).

Architektura číslicového počítače

- architektura se zabývá strukturou a chováním počítače
- řeší specifikaci různých funkčních modulů jako procesor a paměť a nebo např. instrukční sadu
- podsměry:
 - ISA — Instruction Set Architecture — architektura souboru instrukcí
 - Mikroarchitektura — konkrétní sestavení a složení procesoru
 - Systémový design — řeší další HW komponenty
- každý počítač je složen z následujících částí:
 - datová část procesoru — ALU, registry
 - řadič — řídící jednotka procesoru
 - paměťový substituční systém
 - vstupní zařízení
 - výstupní zařízení
- typy architektury:
 - Von Neumannova architektura
Data i instrukce jsou uložena spolu, nejsou explicitně označena/ny.
 - Harvardská architektura
Data a instrukce jsou rozdělené.

Instrukční cyklus počítače

- čtení instrukce (IF — Instruction Fetch)
- dekódování instrukce (ID — Instruction Decode)
- načtení operandů (OF — Operand Fetch)
- provedení instrukce (IE — Instruction Execution)
- zapsání/uložení výsledku (WB — Write Back / Result Store)
- přerušení?

Co je instrukce? Obsahuje informace:

- co se má provést
- s čím se to má provést (operandy)
- kam se má uložit výsledek
- kde se má pokračovat

Tyto informace mohou být zadány explicitně, nebo mohou být dány typem instrukce, tedy architekturou počítače — tedy implicitně.

ISA — Architektura souboru instrukcí Co je potřeba určit:

- typy a formáty instrukcí, instrukční soubor
- datové typy, kódování a reprezentace, způsob uložení dat v paměti
- módy adresování paměti a přístup do paměti dat a instrukcí
- mimořádné stavy

Výhody:

- abstrakce — možnost různě implementovat stejnou architekturu instrukcí
- definice rozhraní mezi nízkoúrovňovým AW a HW
- standardizuje instrukce, bitové vzory strojového jazyka

Třídy souborů instrukcí (ISA)

- Střadačově (akumulátorově) orientovaná ISA

Akumulátor je registr pro mezivýpočty, používá se implicitně jako zdroj pro výpočty i jako cíl pro výsledky. Používají se instrukce s jedním operandem. Nejstarší ISA (1949-60) — vyvinula se z kalkulaček.

Výhody:

- jednoduchý HW
- minimální vnitřní stav procesoru — rychlé přepínání kontextu
- krátké instrukce
- jednoduché dekódování instrukcí

Nevýhody:

- častá komunikace s pamětí
- omezený paralelismus mezi instrukcemi

Populární v 50. — 70. letech, HW byl drahý, paměť byla rychlejší než CPU.

- Zásobníkově orientovaná ISA

Pracovní registry jsou uspořádány do struktury zásobníku. Přistupuje se k vrcholu tohoto zásobníku. Využití pro výhodnocení výrazů a vnořená volání podprogramů. Většina instrukcí nemá operand (použije se implicitně např. vrchní 2 registry zásobníku).

Výhody:

- jednoduchá a efektivní adresace operandů
- krátké instrukce
- krátké programy
- jednoduché dekódování instrukcí
- snadno lze napsat neoptimalizující překladač

Nevýhody:

- nelze náhodně přistupovat k lokálním datům
- omezený paralelismus — zásobník je sekvenční
- přístupy do paměti je těžké minimalizovat

- ISA orientovaná na registry pro všeobecné použití

Dnes převládá. GPR — General Purpose Registers. Typicky 2 nebo 3 operandy.

Výhody:

- registry (a cache) jsou rychlejší než paměť
- k registrům lze přistupovat náhodně
- registry mohou obsahovat mezivýsledky a lokální proměnné
- méně častý přístup do paměti

Nevýhody:

- složitější překladač (optimalizace pro použití registrů)
- přepnutí kontextu trvá déle

Paměťový subsystém počítače

- cache (skrytá paměť) — rychlá, drahá, umístěna blíž k procesoru
- hlavní paměť — pomalejší, levnější, větší
- vnější paměť — pomalá, velká
- záložní paměť (CD, DVD, flash, magnetické pásky)
- RAM — random access memory (přístup adresou)
- CAM — content addressable memory (přístup klíčem)

Paměťová hierarchie

- registry
- L1 cache (SRAM)
- L2 cache (SRAM)
- L3 cache (SRAM)
- hlavní paměť (DRAM)
- HDD, SSD
- Mass storage (optical disks, tapes)
- Remote storage (cloud)

Cache

Kopie často používaných dat z hlavní paměti

- časová lokalita

Data, ke kterým bylo právě přistupováno, budou pravděpodobně brzy potřeba znova.

- prostorová lokalita

Po přístupu k nějakým datům se pravděpodobně budou používat i vedlejší data.

1.30 SP-30 (SAP)

Kódy pro zobrazení čísel se znaménkem a realizace aritmetických operací (paralelní sčítáčka/odčítáčka, realizace aritmetických posuvů, dekodér, multiplexor, čítač). Reprezentace čísel v pohyblivé řádové čárce.

Čísla se znaménkem

Existuje několik možností, jak v počítači ukládat celá čísla:

- Přímý kód
 - první bit je znaménkový — určuje tedy, zda je hodnota za ním kladná či záporná
 - ostatní byty představují absolutní hodnotu čísla
 - existuj zde kladná i záporná nula
- Doplňkový kód
 - dle prvního bitu lze poznat znaménko čísla
 - převod kladné ↔ záporné lze vysvětlit jako inverze bitů a následné přičtení jedničky
 - 0111 (7) → 1001 (-7)
 - není zde záporná nula
- Aditivní kód
 - uložené číslo je posunuto o nějakou konstantu, typicky polovina rozsahu
 - pro 4 bity určeme nulu jako 1000 — pak 1111 je 7, 0000 je -8
 - nula není zobrazena jako nula
 - není zde záporná nula

Čísla v pohyblivé řádové čárce

Reprezentace pohyblivé řádové čárky vychází ze zobrazení $A = M * z^e$ používaném např. ve fyzice, kde z je základ soustavy (zde 2), e je exponent jako celé číslo, M je mantisa.

- používá se normalizovaný tvar, tedy mantisa je zapsána tak, že ji nelze "posunout" více doleva.
- v přímém kódu mantisy je vlevo vždy jednička, která se skrývá (zvýšení přesnosti)
- pro mantisu se typicky používá přímý kód, pro exponent aditivní
- float (32b) typicky vypadá jako 1b znaménko, pak 8b exponent a nakonec 23b mantisa (tedy přesnost 24b)

Realizace aritmetických operací

- Paralelní sčítáčka

Tvořena více jednobitovými sčítáckami. Jednobitová sčítáčka má 3 vstupy: A, B (sčítané byty) a vstupní přenos (carry — např. ze sčítácky nižšího řádu). Výstupy jsou S (výsledek) a výstupní přenos.

- Aritmetické posuvy

Posun čísla vlevo/vpravo. Realizuje posuvný registr. Existuje více různých posuvů:

- logický posuv — doplňuje nuly
- cyklický posuv — doplňuje co vylezlo na druhé straně
- aritmetický posuv — doplňuje 1 nebo 0 podle znaménka čísla

- Dekodér

Kombinační logický obvod, který má méně bitů na vstupu než na výstupu, a podle tabulky převádí. Kodér má opačnou funkci.

- Multiplexor

Na základě řídícího signálu vybere, který ze vstupů pošle na výstup (má několik vstupů + řídící vstup, a jeden výstup).

- Čítač

Registr s funkcí inkrementu/dekrementu, může čítat nahoru a/nebo dolů. Existují úplné čítače (do mocnin 2) či neúplné (do jiných čísel). Typicky čítají v binárním kódu, lze i např. v Grayově kódu.

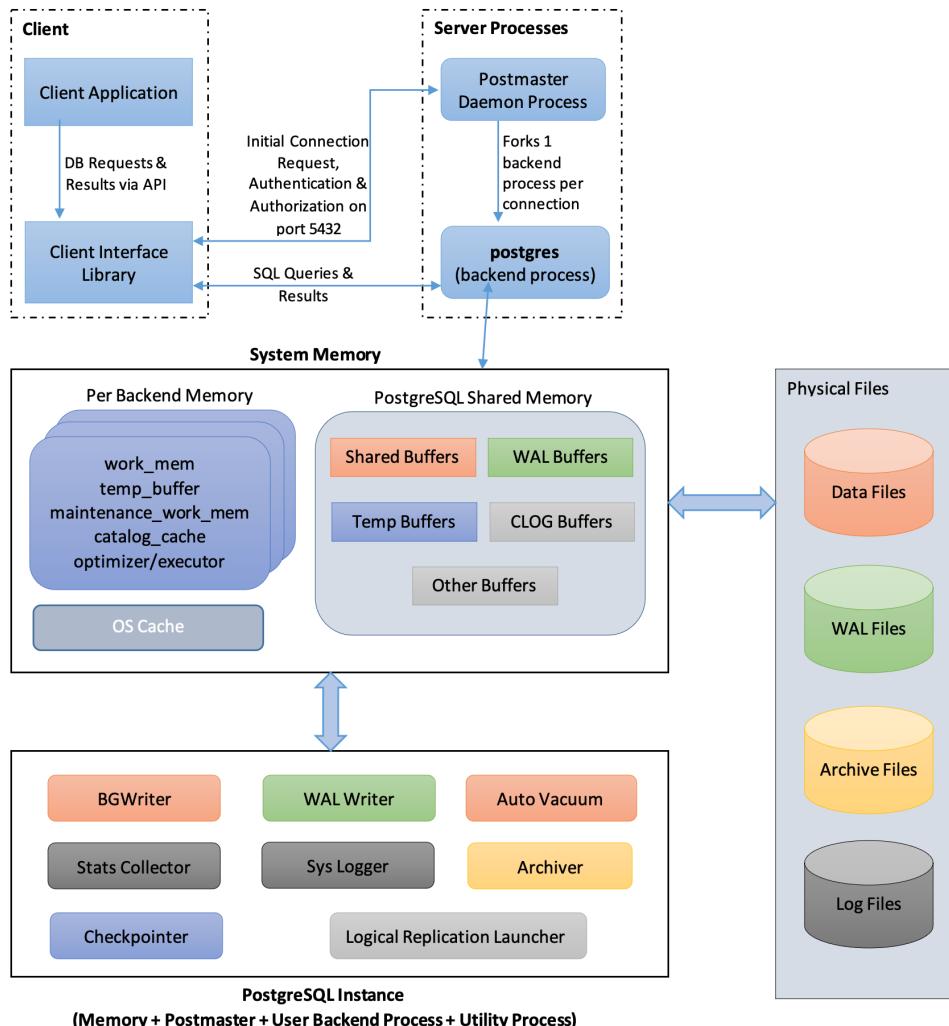
2 Web

2.1 W-1 (AWD)

Architektura databázového serveru, její podstatné komponenty a úloha databázového administrátora při jejich správě.

2.1.1 Architektura databázového serveru

Základním a výchozím procesem databázového serveru Postgres je tzv. Postmaster. Je to hlavní démon který naslouchá na síťových portech a přijímá nová spojení od klientů. Každému nové příchozímu klientovi vytvoří nový proces - svůj fork, který s ním poté komunikuje, odbavuje dotazy a podobně. Díky oddeleným procesům je zajištěna lepší bezpečnost a stabilita - když jeden proces spadne, ostatní pokračují v běhu.



Obrázek 1: Architektura databázového serveru Postgresql

Databáze ještě obsahuje další podpůrné procesy:

- Background writer - periodicky zapisuje změněné bloky z paměťového bufferu (shared buffer pool) na disk.
- WAL writer - zapisuje změny do WAL bufferu a periodicky je flushuje na disk. Je také důležitý pro zotavení aplikace po pádu a umožňuje point-in-time recovery.
- Checkpointer - vytváří tzv. checkpointy - body pro zotavení databáze. Checkpoint je snapshot, do kterého databázový server zapíše všechny změny v shared bufferu na disk a zároveň tento bod zapíše do WAL jako bod pro zotavení. Jinými slovy jedná se o snapshoter.
- Autovacuum Daemon - čistí tabulky od mrtvých řádků vzniklých během transakcí, automaticky spouští VACUUM a ANALYZE příkazy. Mrtvé řádky vznikají při UPDATE nebo DELETE operacích, kdy se stará

verze řádku fyzicky nepřepíše, ale zůstává v tabulce jako mrtvý řádek. Toto je důsledek MVCC (Multi-Version Concurrency Control). VACUUM odstraňuje mrtvé řádky, ANALYZE aktualizuje statistiky a ještě se provádí Anti-wraparound VACUUM, který zabraňuje přetečení 32bitových identifikátorů transakce.

- Archiver - archivuje WAL soubory pro účely zálohování nebo replikace.
- Logical replication launcher - spouští procesy replikace pro logickou replikaci.
- Stats collector - shromažďuje statistiky o činnosti databáze a ukládá je do systémových tabulek. Tyto statistiky jsou užitečné pro ladění výkonu a monitorování databáze.

2.1.2 Fyzické soubory

Databáze využívá následující druhy fyzických souborů:

- Data files - obsahují data tabulek a indexů. Každá tabulka a index je reprezentována jedním nebo více soubory na disku. Tyto soubory mají obvykle příponu .dat a jsou umístěny v adresáři pg_data/base/oid_db/, kde oid_db je identifikátor databáze.
- WAL soubory - obsahují záznamy o změnách provedených v databázi. Tyto soubory jsou důležité pro zotavení databáze po pádu.
- Konfigurační soubory - obsahují nastavení databázového serveru, jako je pg_hba.conf (nastavení přístupu) a postgresql.conf (hlavní konfigurační soubor).
- Archive files - archivované WAL soubory, které jsou uloženy na jiném místě pro účely zálohování nebo replikace.
- Log files - obsahují záznamy o činnosti databázového serveru, jako jsou chybové hlášení, varování a informace o výkonu. Tyto soubory jsou užitečné pro ladění a monitorování databáze.

2.2 Systémová paměť

Databáze využívá následující druhy paměti:

- Shared memory - sdílená paměť, která je používána všemi procesy databázového serveru. Obsahuje shared buffers, WAL buffers, Temp buffers, CLOG buffers (commit log) a další.
- Per backend memory - paměť, která je alokována pro každý proces databázového serveru. Obsahuje lokální proměnné, pracovní paměť a další dočasné struktury.

2.2.1 Úloha databázového administrátora

Databázový administrátor (DBA) je zodpovědný za správu databázového serveru a jeho komponentů. Mezi jeho hlavní úkoly patří:

- instalace a konfigurace databázového serveru
- správa uživatelských účtů a oprávnění
- monitorování výkonu databáze a ladění výkonu
- zálohování a obnova databáze
- správa replikace a clusteringu
- správa bezpečnosti databáze
- správa aktualizací a patchů databázového serveru

3 Bezpečnost - kvůli referencím

3.1 OB-1 (ADU)

Identita uživatelů v unixových operačních systémech (identita, práva administrátora, sudo, su, PAM moduly, role, privilegia, identita a přístupová práva, ACL, suid programy).

Administrace uživatelů

- konfigurační soubory
- uživatelé a jejich primární skupiny v souboru /etc/passwd
- hesla uživatelů v /etc/shadow
- (sekundární) skupiny uživatelů v /etc/group
- přihlašování na jiného uživatele (změna identit): příkaz su [-] [username [argument]] ("-" rozhoduje zda se použijí přihlašovací skripty cílového uživatele, tedy jestli se změní prostředí)

Identita procesů

- vnější identita: username, groupname
- vnitřní identita: UID, GID
- 3 druhy vnitřní identity:
 - Reálná (RUID, RGID) — využívána při akceptaci signálů
 - Efektivní (EUID, EGID) — využívána při práci se soubory (vlastnictví, práva,...)
 - Uložená/saved (SUID, SGID) — uchování identity při změně EUID/EGID
- potomci procesů dědí identity od rodiče, s výjimkou SUID/SGID programů. kdy potomek získá EUID a SUID od vlastníka programu (respektive EGID a SGID) — např. "su"

Administrátor

- nazýván root
- UID = 0
- může všechno — měnit runlevel systému, vypínat/zapínat systémové služby, používat/přidávat zařízení...
- v bezpečnějších systémech pouze jako role — nelze se přihlásit přímo

sudo

- super user do
- konfigurace v /etc/sudoers (může měnit jen root)
- spuštění konkrétního příkazu jako jiný uživatel (typicky root)
- lze nakonfigurovat konkrétní práva pro konkrétní uživatele, lze logovat použití
- náchylné k chybám při konfiguraci
- běžný účet může díky právům něco zničit
- záznam v konfiguraci: who where=(as who) what
- možné distribuovat na další zařízení, tedy řídit více systémů
- použití většinou vyžaduje heslo aktuálního uživatele (ne toho za koho je příkaz prováděn)

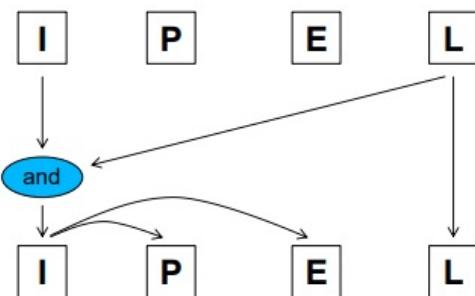
ACL

- klasická oprávnění (rwx pro vlastníka, skupinu a ostatní) nebyla dostatečná
- ACL (Access Control List) jsou dodatečné informace o právech přístupu k souborům (pouze rozšíření, ne nahrazení)
- umožňují specifikovat rwx práva pro každého konkrétního uživatele a skupinu

- lze použít masky
- příkaz setfacl a getfacl
- není standardem, některé FS/OS neimplementují

Práva a role

- klasický UNIXový systém řeší práva přes UID (kontroluje, zda $UID == 0$)
- administrátorská práva jsou nedělitelná, problém v případě více administrátorů
- systém, který podporuje práva:
 - cca 80 práv pokrývajících všechny operace
 - UID nemá efekt na práva, procesy si s sebou přenáší množinu práv
 - root má typicky všechna práva, běžný uživatel podmnožinu
 - 4 množiny práv v procesu: Effective, Inheritable, Permitted, Limit



- systém kontroluje, zda je odpovídající právo obsaženo v množině E
- RBAC:
 - Role Based Access Control
 - uživatelé mají přiřazené role
 - role mají vlastnosti jako uživatelé, ale nelze se přihlásit přímo, pouze přes "su"— např role root
 - uživatel se pro použití práv role musí přihlásit (musí tedy mít roli přiřazenou, a zároveň znát její heslo)
 - role mají přiřazené profily
 - profily mají přiřazené množiny příkazů, spustitelné pod určitými identitami a se specifickými právy
- PAM
 - Pluggable Authentication Modules
 - moduly pro autentizaci, které se dají připojit ke každému programu zvlášť
 - možnost vytvářet programy nezávisle na konkrétních autentizačních postupech
 - konfigurace pro konkrétní program: typ modulu — typ použití — modul

/etc/pam.d/login

```

auth      required  pam_securetty.so
auth      required  pam_shells.so
auth      required  pam_nologin.so
auth      include   system-auth
account  include   system-auth
password include   system-auth
session  required  pam_env.so
session  include   system-auth
  
```

3.2 OB-2 (ADU)

Správa disků a souborových systémů (zařízení, souborové systémy UFS (EXT) a ZFS, disková pole RAID, diskové kvóty), síťové souborové systémy (NFS, CIFS), swap v unixových operačních systémech.

Zařízení:

- popsána speciálními soubory v adresáři /dev
- znakové/blokové
- zápis/čtení souboru znamená stejnou operaci nad zařízením
- pseudozařízení — /dev/null, /dev/random, /dev/zero,...
- vše potřebné uloženo v i-node (jako odkaz na driver)
- většinou může operace nad těmito soubory provádět jen root
- lze vytvořit přes mknod

Disky:

- rozdelení disku závisí na HW i na OS
- na x86: partitions (partition tabulka)
- na solaris: slices (pokud na x86, jsou v rámci partitions)
- Partitions:
 - rozdělují fyzický disk na více logických celků
 - popsány partition tabulkou
 - MBR — Master Boot Record — v prvním sektoru disku, obsahuje kód zavaděče (GRUB) a partition tabulku se 4 záznamy (max 4 partitions)
 - GPT — GUID Partition Table — partitions mají globálně unikátní ID, je součástí UEFI (nástupce BIOS)

Filesystem:

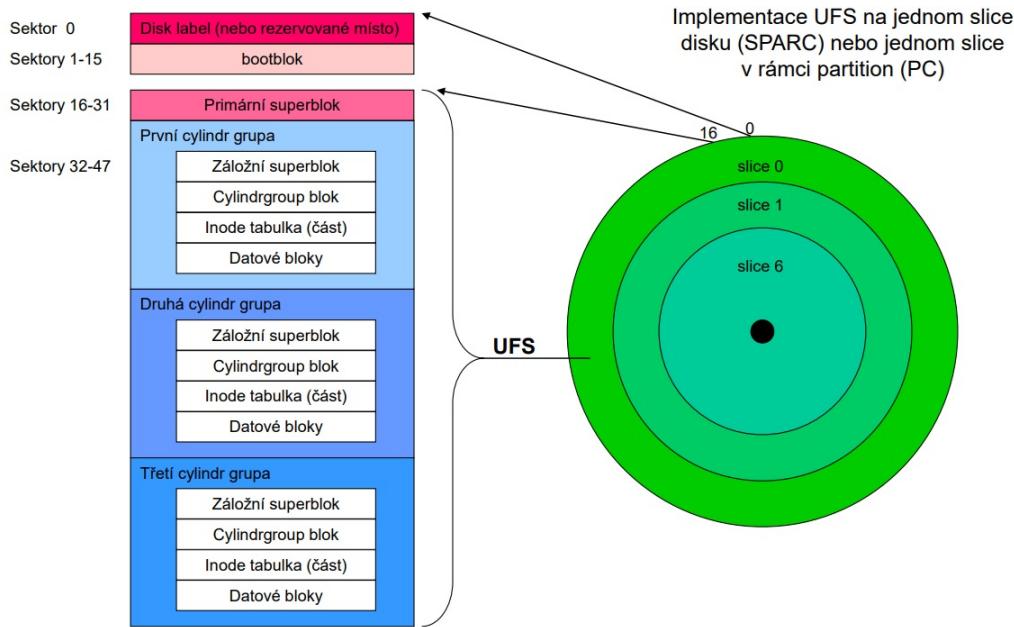
- logický — adresářový strom
- fyzický — filesystem na disku, připojuje se k mountovacímu bodu
- diskový, síťový, ...

SWAP:

- odkládací prostor na disku pro nepoužívané stránky paměti
- většinou se řeší buď pomocí swap partition, nebo swapovacími soubory

UFS

- efektivní pro menší soubory (max desítky/stovky MB)
- nevzniká fragmentace
- bootblock (na začátku disku), superblock primární (po bootblocku) a záložní (na začátku každé cylindr grupy)
- i-node
 - struktura obsahující metadata
 - typ, přístupová práva, vlastník, skupina, velikost, čas vytvoření, přístupu a modifikace, čítač linků
 - odkazy na datové bloky: 12 přímých, 3 nepřímé (první, druhé a třetí úrovně)
- příkazy: mkfs (vytvoření filesystemu), mount (připojení FS k mountovacímu bodu), fsck (kontrola a oprava disku)
- snapshot — zapamatování stavu FS, používá se k archivaci



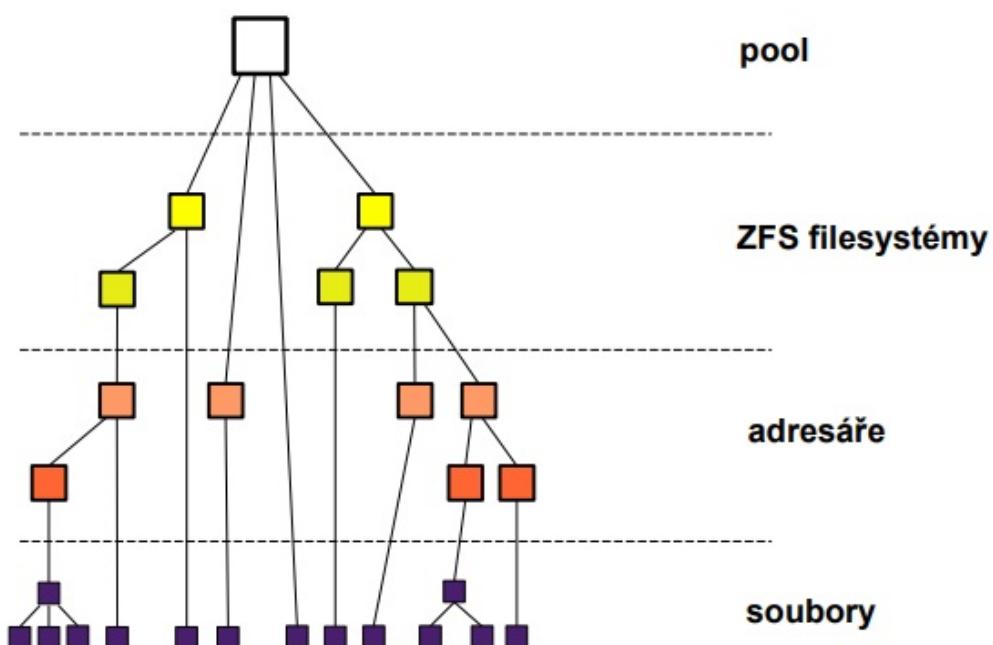
RAID

- Redundant Array of Independent Disks
- možné řešit jak HW tak SW
- složení jednoho logického disku z více fyzických
- může zvýšit kapacitu, zvýšit výkon a zvýšit bezpečnost (obrana proti výpadku)
- RAID 0 - zřetězení (JBOD - just a bunch of disks)
 - data se ukládají postupně na disky za sebe — když se zaplní jeden, začne se ukládat na druhý
 - redundancy 0 %
 - výpadek jednoho disku způsobí ztrátu dat
 - výkon se nemění
- RAID 0 - prokládání
 - data jsou ukládána cyklicky po blocích na jednotlivé fyzické disky
 - redundancy 0 %
 - výpadek jednoho disku způsobí ztrátu všech dat
 - výkon zvýší m krát (m je počet disků)
- RAID 1 - zrcadlení
 - stejná data jsou uložena na všech discích
 - redundancy je $100 \times (m - 1)/m$ %
 - data přežijí výpadek $m - 1$ disků a výkon nebude degradován
 - write operace stejně rychlé, read se může zrychlit až m krát
- RAID 1+0 / 0+1
 - nejvyšší logický disk je rozdělen podle RAID 0/1, nižší logické disky jsou opačně (RAID 1/0)
 - redundancy je podle počtu zrcadlení v RAID 1 části — při 2 kopiích dat je to 50 %
 - data přežijí výpadek od 1 do $m/2$ disků (při 2 kopiích)
 - write operace zrychleny až $m/2$ krát, read až m krát
- RAID 2, 3, 4 se nepoužívají (2: prokládání po bitech + hammingův kód, 3: prokládání po bytech a zabezpečení pomocí uložení parity uložené na jednom fyzickém disku, 4: prokládání po blocích, uládání parity na jednom fyzickém disku)

- RAID 5
 - prokládání po blocích na m fyzických discích + ukládání parity cyklicky ukládané na jednotlivých discích
 - redundancy je $100/m$
 - data přežijí výpadek jednoho disku, ale bude degradován výkon
 - read se zrychlí, write pomalejší
- RAID 6
 - prokládání po blocích na m fyzických discích + ukládání dvojí parity cyklicky ukládané na jednotlivých discích
 - redundancy je $200/m$
 - data přežijí výpadek dvou disků, ale bude degradován výkon
 - read se zrychlí, write pomalejší

ZFS

- snaží se řešit problémy stávajících FS
 - nekonzistence při nekorektním vypnutí
 - nemožnost/komplikované zvětšování kapacity FS
 - RAID, snapshot, zálohování jsou řešené mimo vlastní FS
 - FS nejsou hierarchické
 - komplikovaná administrace
 - chybějící klonování, cache, kryptování, deduplikace...
- metadata řešena podobně jako v UFS
- datový prostor tvoří virtuální datová oblast — pool
- pool je tvořen zdroji dat — disky, partitions, soubory (speciální — zařízení)
- z poolu se alokují datové bloky — různá velikost, tvořené nad strukturami RAID
- transakční systém "copy on write" — data se nepřepisují, pouze se zkopirují, změní a pak je změna přijata či zamítnuta
- pool sám je (ZFS) FS
- každý blok má kontrolní součet



NFS

- daemons na serveru:
 - mountd — vyřizuje požadavky na mount, vrací file handle
 - nfsd — vyřizuje požadavky na operace se soubory, vrací data
 - statd — spolupracuje s lockd, znovunastavuje spojení po výpadku
 - lockd — zamykání NFS souborů, požadavek se posílá z klienta na server
 - nfslogd — logování přístupů
 - daemons na klientu — statd, lockd
 - NFS 2/3 nestavové, NFS 4 stavový
 - NFS 4 — sjednocený daemon, well known port 2049, delegace cache, možnost mountovat pseudo filesystem (vyšší adresář, ale s přístupem jen do nižšího)
- CIFS/SMB** (Samba) je novější síťový FS, snadnější vazba unix-windows.

3.3 OB-3 (ADU)

Procesy a systémové služby v unixových operačních systémech: hierarchie a vzájemné vazby, limity, zapínání a vypínání systému, logování aktivit systému.

Procesy:

- vykonávané programy
- mají svoje ID (PID), id uživatele (UID), id rodiče (PPID) ...
- každý proces někdo vytvořil, tedy každý proces má nějaké PPID — init proces je první proces spuštěn při bootu, má PID = 1 a PPID = 0
- rodič typicky čeká, až mu dítě předá návratový kód
- daemon = systémový proces (běží na pozadí bez nutnosti vstupu ... jako u windows služby)
- orphan = proces, jehož rodič už neexistuje — adoptuje se pod proces init
- zombie = proces, který již skončil, ale ještě si rodič nepřevzal návratovou hodnotu
- procesy si navzájem mohou posílat signály

Limity:

- soft limit — limit mezi 0 a hard limitem, lze uživatelsky přenastavit
- hard limit — může změnit jen root, nelze překročit
- lze nastavit max počet procesů pro uživatele a další omezení využití zdrojů (paměť, velikost souborů, počet file descriptorů...)
- příkaz ulimit

BOOT

- Firmware fáze
 - POST (Power On Self Test)
 - inicializace HW a driverů
 - výběr bootovacího zařízení
 - načtení a spuštění bootovacího kódu
- Boot-loader fáze
 - nalezení a načtení boot programu
 - boot program se nahrává do pevně daných adres v paměti
 - kontrola řízení se předá programu
- GRUB (Grand Unified Boot Loader) fáze
 - načte se GRUB konfigurace
 - vybere se odkud/co bootovat
 - kernel se načte a spustí
- Kernel fáze
 - inicializace datových struktur jádra
 - načtení driverů, inicializace HW
 - mount kořenového filesystému
 - načtení konfigurace jádra
 - načtení modulů
 - vytvoření init procesu

- Init fáze
 - dříve spouštění start/stop scriptů dle úrovní běhu systému (run levels / milestones)
 - * 0 — systém vypnutý
 - * 1, s, S — single user mode
 - * 2 — multi user mode
 - * 3 — multi user mode + síť
 - * 4 — nepoužívaný (občas GUI)
 - * 5 — vypnutí
 - * 6 — restart
 - v dnešní době spouštění služeb
 - konfigurace procesu init — /etc/inittab
 - init se přes fork() a exec() naklonuje a spouští další procesy

Vypínání systému:

- shutdown *level timeout -y* (přívětivý jak k uživatelům a aplikacím, tak k systému)
- init *level* (přívětivý k aplikacím a systému)
- poweroff, restart (přívětivé k systému)
- vypnout napájení (fuj)

Logování:

- servrová logovací služba s dlouhou historií
- umožňuje oddělit SW generující zprávy, SW který je ukládá a SW který je analyzuje a nahlašuje
- konfigurace: /etc/syslog.conf
- logovací složky: /var/log a /var/adm
- mnoho variant
- záznamy v logu mají info o službě a závažnosti logu (alert, critical, error, warning, notice, info, debug)

3.4 OB-4 (APS)

Instrukční cyklus počítače a zřetězené zpracování instrukcí. Mikroarchitektura skalárního procesoru se zřetězeným zpracováním instrukcí, datové a řídicí hazardy při zřetězeném zpracování instrukcí a způsoby jejich ošetření.

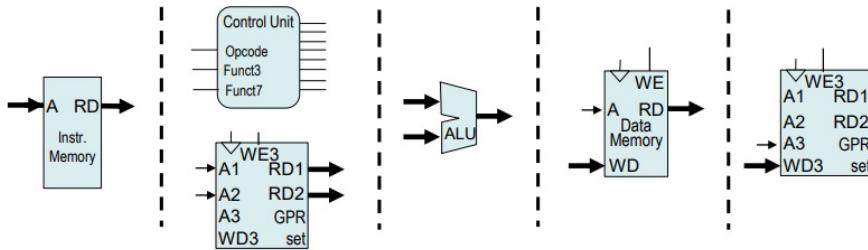
Definice 1: ISA (Instruction Set Architecture) je abstraktní rozhraní mezi HW a nízkoúrovňovým SW, které zahrnuje vše nezbytné pro psaní korektních programů ve strojovém jazyce. Zahrnuje instrukční sadu, registry, organizaci paměti, vstupy a výstupy,...

Definice 2: ISA je kompletní instrukční sada procesoru, včetně adresních módů.

Mikroarchitektura: Mikroarchitektura je detailní interní organizace procesoru, včetně hlavních funkčních jednotek, jejich propojení a řízení.

Instrukční cyklus počítače

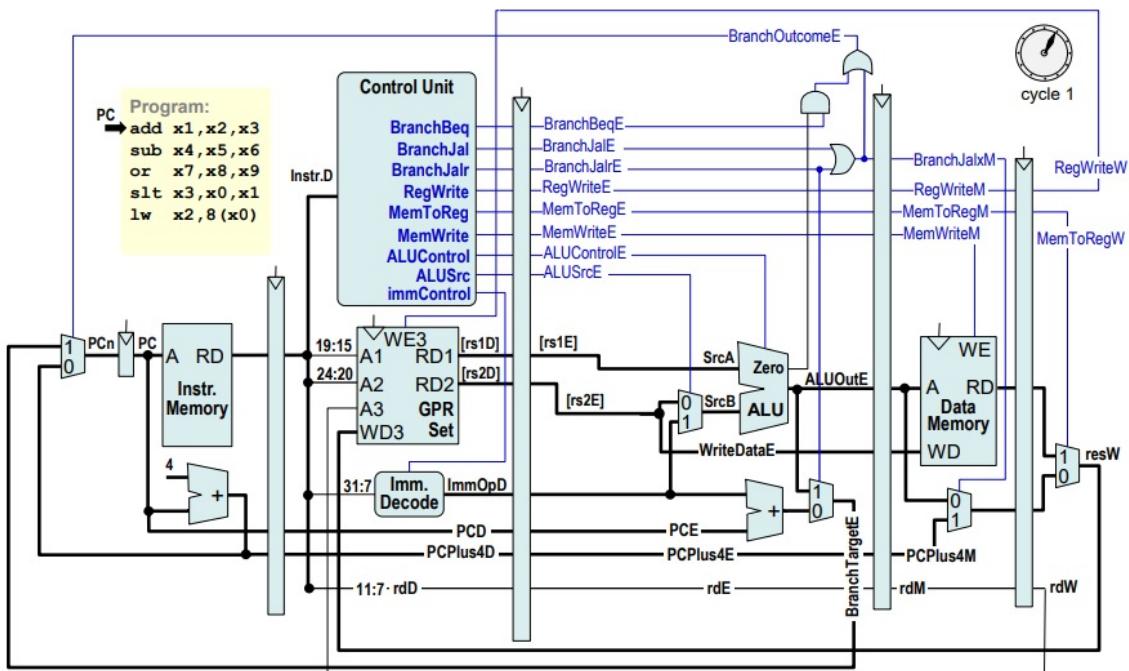
- IF — Instruction Fetch
- ID/OF — Instruction Decode and Operand Fetch
- EX — Execute
- MEM — Memory access
- WB — Write Back



Zřetězené zpracování instrukcí

Instrukce se dle svých fází rozdělí a v procesoru vykonává postupně. Procesor je rozdělen dělícími registry. Instrukce se zpracovává postupně v rozdělených částech procesoru, vykonává se zároveň více instrukcí naráz (v různých fázích).

Mikroarchitektura skalárního procesoru se zřetězeným zpracováním instrukcí



Hazardy

Protože je rozpracováno více instrukcí najednou, mohou vznikat konflikty při přístupu ke sdíleným prostředkům počítače. Tomu se říká hazardy. Sdíleným prostředkem je prostředek, který je opakovaně použit v různých stupních instrukčního zřetězení.

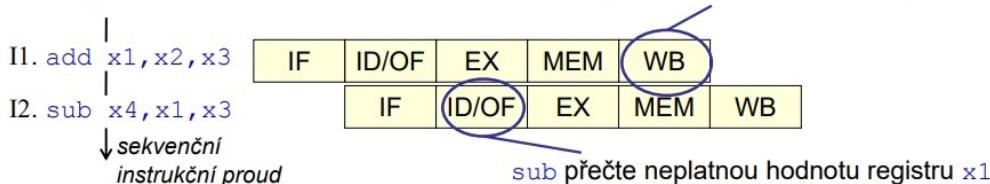
Typy hazardů:

- datové (důsledek datových závislostí: RAW, WAR, WAW)
- řídící (instrukce měnící PC, tedy obsah fronty instrukcí)
- strukturální (počet současných požadavků na daný prostředek převyšuje počet jeho instancí)

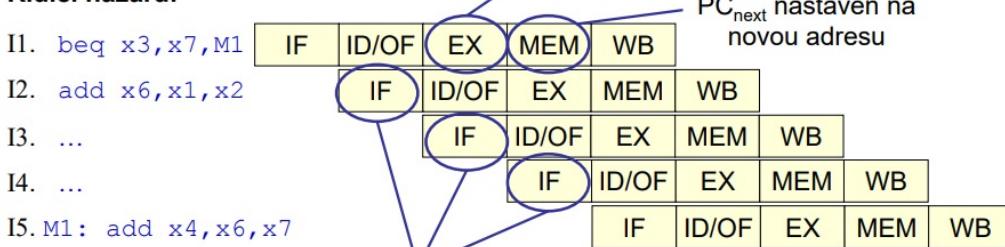
Hazardy mohou způsobovat pozastavení instrukčního zřetězení (stall) nebo vyprázdnění (flush).

Datový hazard:

add zapíše novou hodnotu do registru x1



Řídicí hazard:



Možnosti řešení hazardů:

- přeposílání (forwarding)

Lze použít v případě datového hazardu, kdy výsledek předchozí instrukce požadovaný instrukcí nasledující vznikne dříve nebo ve stejném cyklu, kdy má být následující instrukcí použit. Výsledek je přeposlán tam, kde je potřeba (část EX).

- pozastavení (stall)

Lze použít v případě datového hazardu, kdy je výsledek předchozí instrukce potřeba před jeho vznikem. Zpracovávání následujících instrukcí se pozastaví a vyplní se instrukce nop (bublina). V dalších instrukcích se pokračuje, když výsledek existuje — lze jej tedy přeposlat.

Také může řešit strukturální hazardy.

- vyprázdnění části pipeline (flush)

Lze použít v řídících hazardech, kdy se PC nastaví na neočekávanou adresu nějakou skokovou instrukcí. Všechny již načtené a zpracovávané instrukce, které následují po skoku, se musí z fronty odstranit, a následuje zpracovávání instrukcí z chtěné adresy (kam skok skočil).

Hazardy řeší nová jednotka v procesoru — HMU (Hazard Management Unit).

3.5 OB-5 (APS)

Paměťová hierarchie se skrytou pamětí (cache memory), principy lokality a fungování skryté paměti. Architektura přímé, částečně asociativní, plně asociativní skryté paměti.

Paměťová hierarchie

Rozdíl mezi rychlostí procesoru a rychlostí odpovědi paměti je velký (procesor vs DRAM — 100x, procesor vs HDD — 10 milionkrát). Tento rozdíl se překlene paměťovou hierarchií.

- L1 cache
 - SRAM
 - nejmenší, nejbliže jádru, díky tomu nejrychlejší
 - velikost v řádu jednotek či desítek KB
 - pro každé jádro zvlášť, bývá rozdělena na instrukční a datovou cache
 - obsahuje právě nejpoužívanější data a instrukce
- L2 cache
 - SRAM
 - větší, blízko jádra, latence větší než L1
 - velikost v řádu stovek KB
 - pro každé jádro zvlášť, společná pro instrukce a data
 - obsahuje vše co je v L1 + druhá nejpoužívanější data a instrukce
- L3 cache
 - SRAM
 - ještě větší, stále poměrně blízko jádrům, latence větší než L2
 - velikost v řádu jednotek MB
 - typicky sdílena více jádry, společná pro instrukce i data
 - obsahuje vše co je v L2 + třetí nejpoužívanější data a instrukce
- Hlavní paměť
 - DRAM
 - velká, mimo procesor, tedy latence zásadně vyšší než u cache
 - velikost typicky v řádu jednotek či desítek GB, může být i větší/menší
 - společná pro celý procesor(y), obsahuje data i instrukce
 - obsahuje vše co je v L3 + naprostou většinu potřebných dat i instrukcí
- Sekundární paměť
 - HDD, SSD
 - největší, nejpomalejší
 - společná pro celý počítač, obsahuje vše

Principy lokality

Programy typicky přistupují v daném okamžiku jen k malé části instrukčního a datového adresního prostoru.

Časová lokalita:

- položky, ke kterým se přistupovalo nedávno, budou brzy zapotřebí znovu
- příklad: opakované procházení dat v cyklu, opakované čtení instrukcí v rekurzivních algoritmech

Prostorová lokalita:

- položky poblíž právě používaných budou brzy zapotřebí také
- příklad: sekvenční přístup k instrukcím programu, sekvenční přístup k datovým polím nebo lokálním proměnným umístěným poblíž sebe

Principy fungování skryté paměti

- Cache block

Souvislý, nedělitelný úsek hlavní paměti, který lze přenést do cache během jedné paměťové transakce.

- Cache hit

Úspěšný přístup ke cache block — tedy úspěšný přístup k datům/instrukcím, které jsou obsaženy v cache a jsou platné.

- Cache miss

Opak cache hit — neúspěšný přístup k datům/instrukcím v cache.

- Hit rate

Úspěšnost přístupů do cache (poměr).

- Miss rate

Neúspěšnost přístupů do cache (poměr).

- Hit time

Latence cache, čas na získání dat z dané úrovni cache.

- Miss penalty

Celkový čas pro získání dat při výpadku (cache miss) dané úrovni (počítá se od dané úrovni dál).

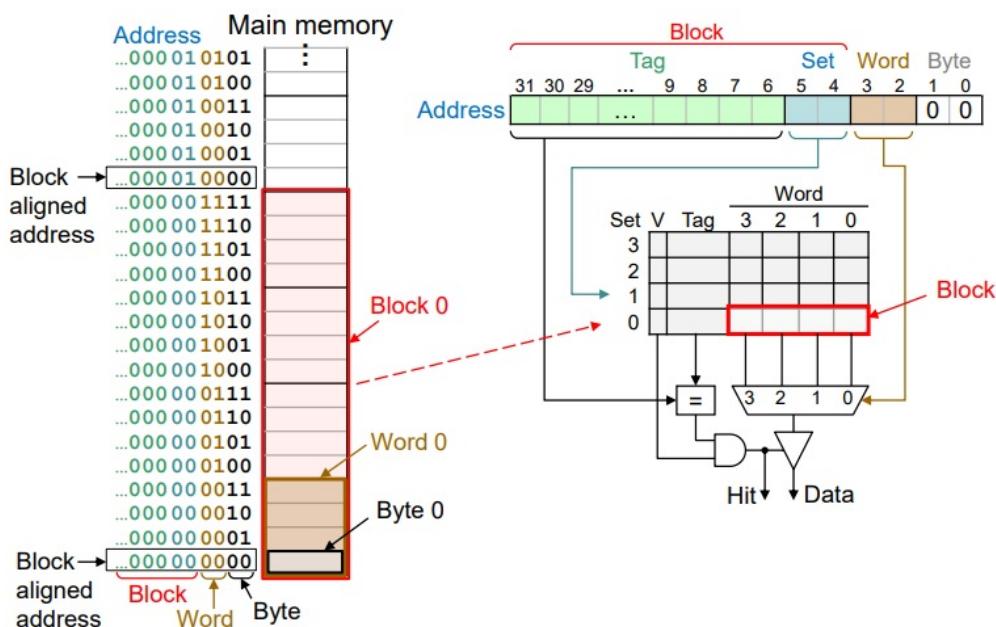
Průměrný čas přístupu do paměti: Hit time + (Miss rate * Miss penalty)

Adresace skryté paměti

Kusy hlavní paměti se do cache ukládají po blocích. Blok může být různě veliký, typicky obsahuje více bytů (nejmenších adresovatelných kusů paměti).

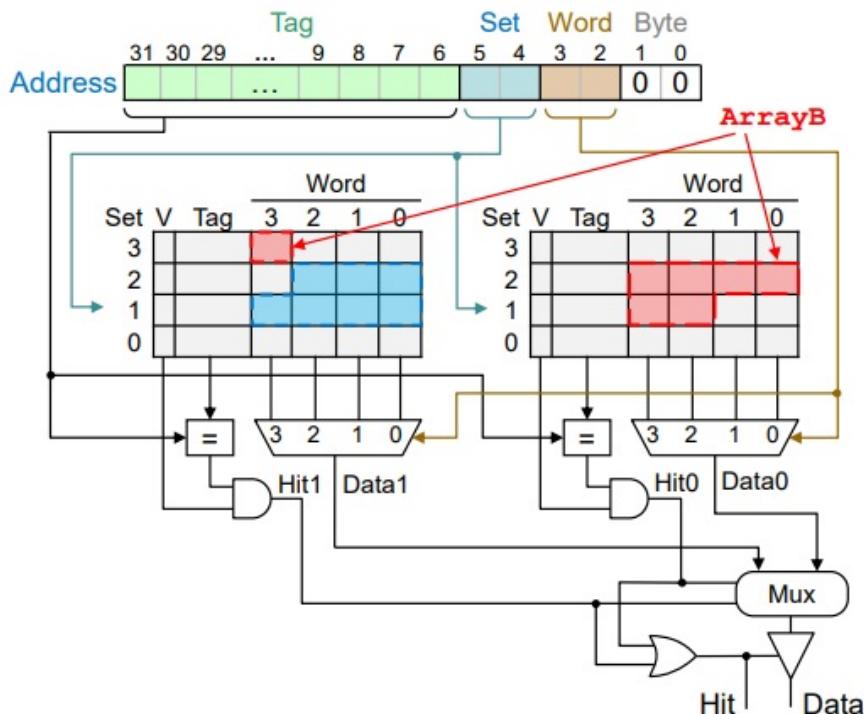
- Přímo mapovaná cache

- každý set (rádek) cache obsahuje právě 1 blok.
- za sebou jdoucí bloky hlavní paměti se mapují do za sebou jdoucích bloků cache
- rádek = (Adresa v hlavní paměti / velikost bloku) % počet rámek
- každý blok hlavní paměti má tedy vždy stejný blok cache kam se uloží.
- na stejný blok cache lze uložit více různých bloků paměti, musíme tedy uložit navíc číslo bloku hlavní paměti (část původní adresy) = Tag



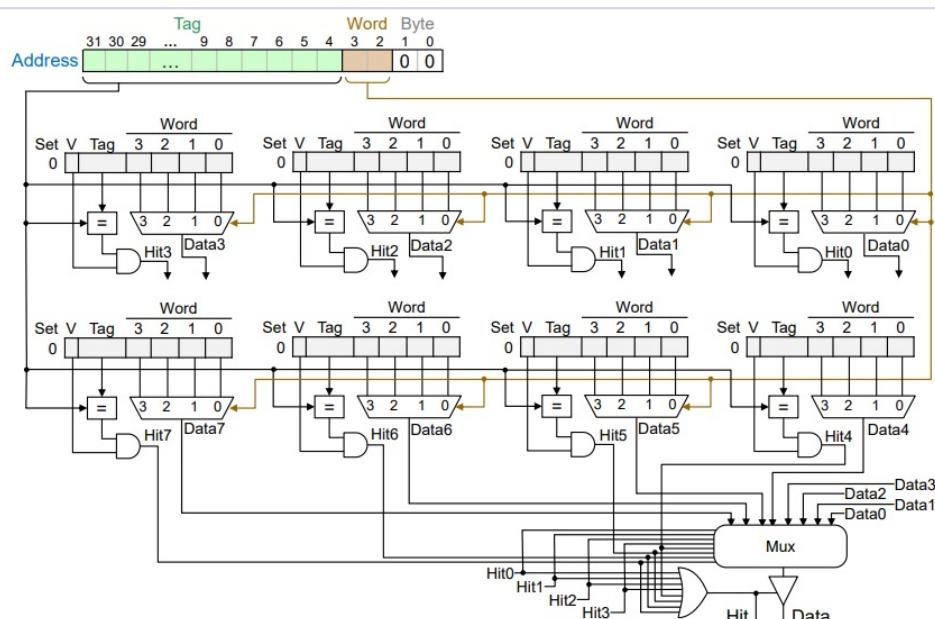
V takovémto cache ale často dochází ke kolizím, když chceme např. přistupovat do více různých částí hlavní paměti. Je ale snazší ji implementovat.

- Cache s částečným stupněm asociativity
 - replikace instancí přímo mapované cache
 - stupeň asociativity je počet takových instancí, neboli také počet cest v cache
 - bloky hlavní paměti lze uložit na více různých míst (přesně na kolik je stupeň asociativity)



Vyšší složitost implementace, ale zásadně nižší miss rate.

- Plně asociativní cache
 - počet cest je roven počtu bloků cache
 - instance přímo mapované cache mají tedy jen jeden řádek (set)
 - u každého bloku se ukládá celá jeho adresa v hlavní paměti



Nejnáročnější na HW prostředky, ale musí řešit maximální počet kolizí.

3.6 OB-6 (APS)

HW podpora virtualizace hlavní paměti stránkováním, funkce MMU (Memory Management Unit) a překlad virtuálních adres na fyzické adresy pomocí TLB (Translation Lookaside Buffer), ošetření výpadku stránky.

Problémy bez virtualizace (motivace pro virtuální paměť):

- Velikost paměti: paměť nemusí být dost velká pro spuštění procesu.
- Fragmentace: Při ukončování procesů vzniknou v hlavní paměti volá místa různých velikostí.
- Dynamická alokace: Jak alokovat další paměť?
- Bezpečnost: Jak zařídit, aby si procesy nemohly číst paměť navzájem?

Jak funguje stránkování?

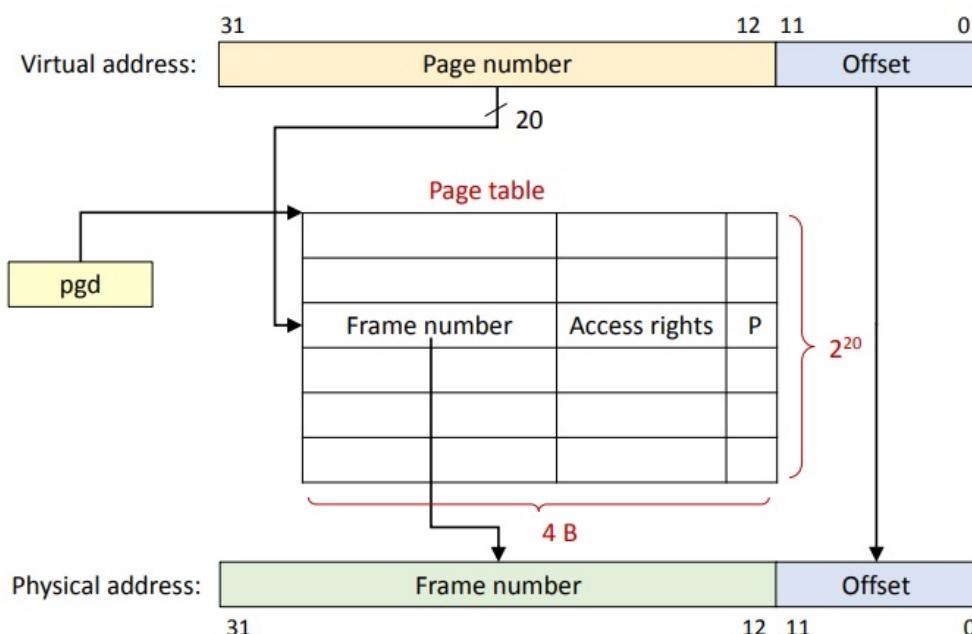
- uživatelským procesům je nabídnut Virtuální Adresní Prostor (VAP)
- každý proces má svůj VAP
- VAP je rozdělen na stejně velké *stránky*, hlavní paměť (HP) je rozdělena na stejně velké *rámce*.
- běžící procesy do rámci HP umisťují momentálně potřebné stránky svého VAP (pracovní množina)
- nepoužívané stránky se při nedostatku paměti odloží na disk
- mapování VAP do HP a přenos stránek mezi HP a diskem zajišťuje OS
- virtuální adresa se skládá z čísla stránky a offsetu
- fyzická adresa se skládá z čísla rámce a offsetu

Možnosti překladu virtuálních adres na fyzické:

- Konvenční stránkovací tabulka
Každý proces má svou stránkovací tabulkou (většinou strom stránkovacích tabulek).
- Inverzní stránkovací tabulka
Všechny procesy sdílejí jedinou, inverzní stránkovací tabulkou.

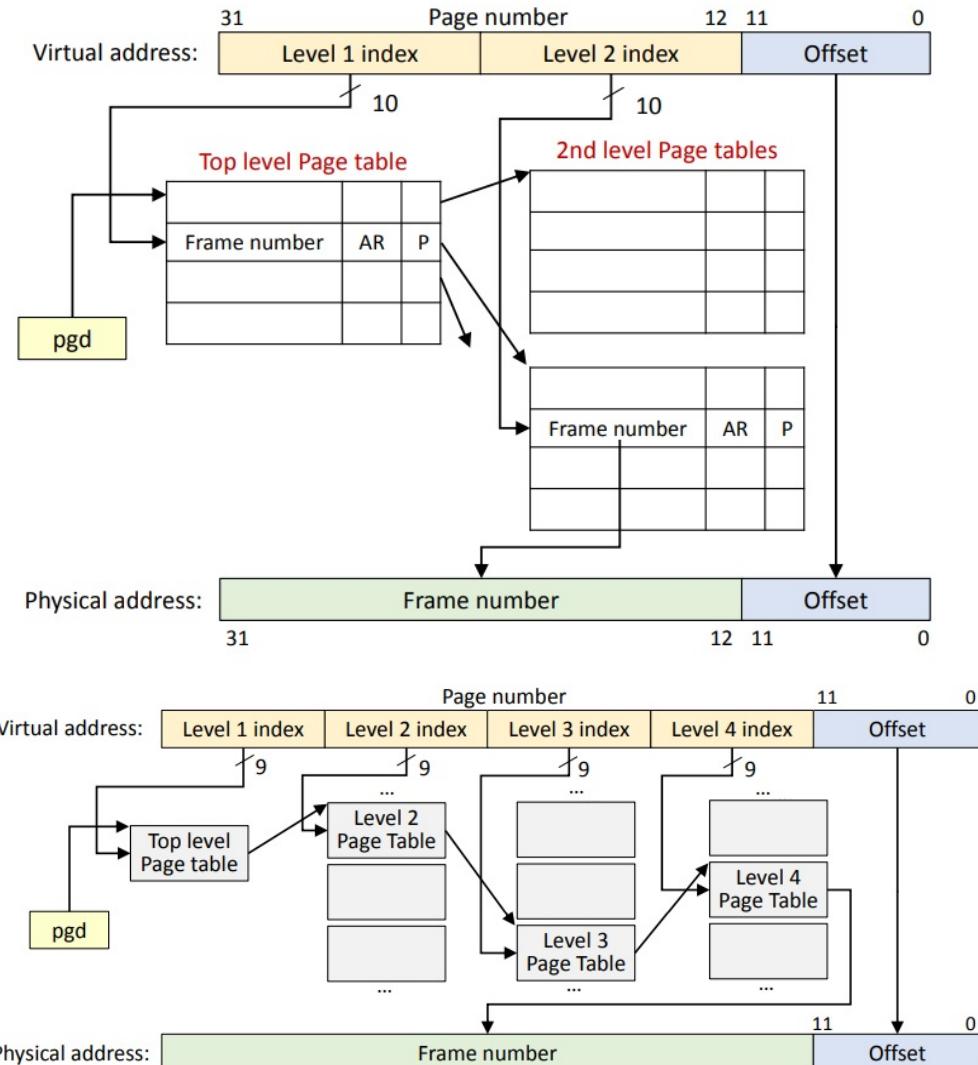
Jednoúrovňová stránkovací tabulka:

- pro překlad se použije číslo stránky jako index do tabulky, a tam se najde číslo rámce
- jednoúrovňová tabulka je ale i pro 32bitové systémy velká, a když ji má každý proces, zabere to hodně místa



Víceúrovňové stránkovací tabulky:

- pro překlad se použije číslo stránky, které je složené z indexů do různých úrovní stránkovacích tabulek
- jednotlivé tabulky jsou zásadně menší, na počátku stačí jedna tabulka v každé úrovni, další OS může podle potřeby přidat



Při překladu VA na FA se musí procházet několik úrovní tabulek stránek (page walk). Ty mohou být uloženy mimo paměť a může dojít k výpadku stránky (page fault).

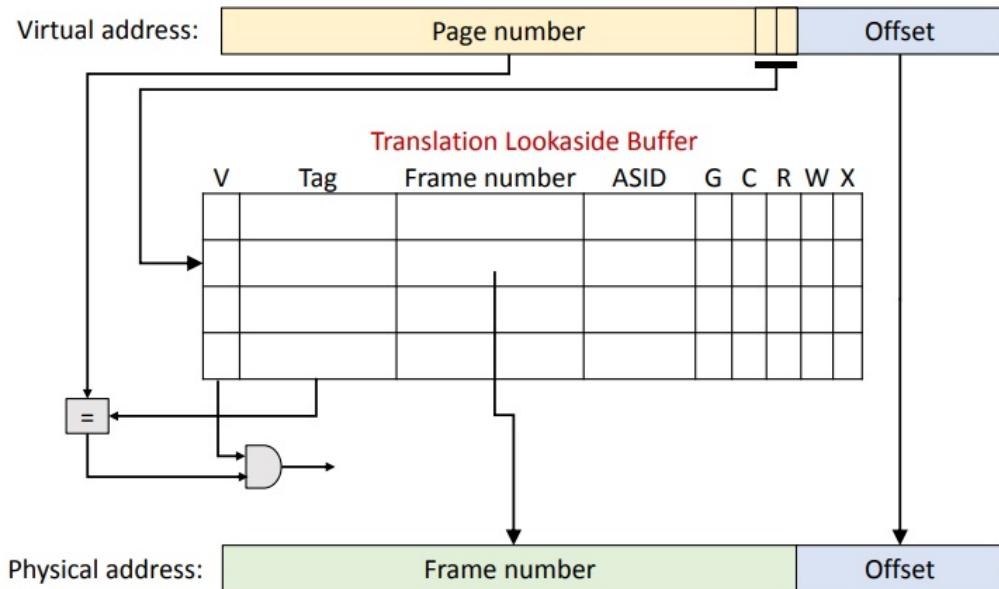
MMU — Memory Management Unit

- vykonává page walk
- dostane adresu stránkovací tabulky první úrovně přes domluvený registr
- pokud nedokáže přeložit adresu, nastává page fault, a procesor generuje výjimku
 - Invalid page fault — adresa není součástí adresního prostoru procesu — obvykle porces zastaven se segmentation fault
 - Valid page fault — adresa je součástí VAP, ale nezle přeložit (nenachází se v MMU, tedy musí page walk vykonat OS / překlad neexistuje — stránka není v HP ale na disku — OS vymění stránky v HP)

TLB — Translation Lookaside Buffer

Vykonávání page walk je časově náročný proces. Aby nebylo nutné pokaždé page walk vykonávat, každá MMU používá speciální HW překladovou tabulkou — TLB.

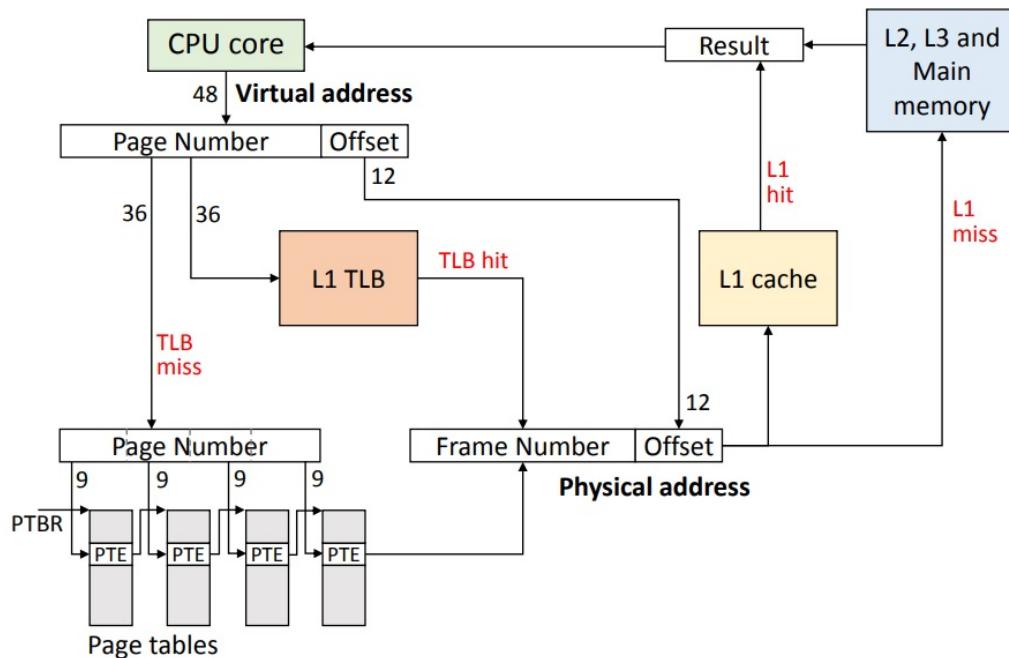
TLB jako přímo mapovaná cache:



Typicky se používá stupeň asociativity 4-64. Podobně jako u cache se používá SRAM. Každá položka TLB typicky obsahuje:

- V: validity bit
- Tag: číslo stránky (část)
- Frame number: číslo rámce
- ASID: Identifikátor adresního prostoru (pro oddělení procesů)
- G: global flag (pokud $G = 1$, ASID se ignoruje)
- C: cache policy (informace, jestli jsou data na adrese "cachovatelná"— pro I/O zařízení, kam se změny dat musí posílat rovnou)
- Access permissions: R, W, X

Sumarizace HW podpory:

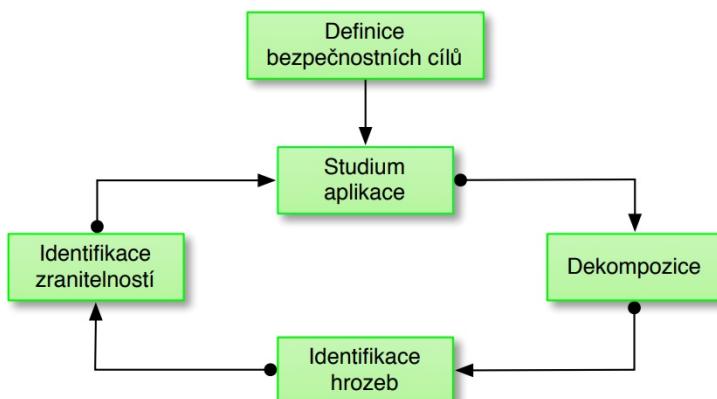


3.7 OB-7 (BEK)

Základní bezpečnostní principy. Modelování bezpečnostních rizik, metodiky STRIDE a DREAD.

- Hrozba — potenciální příčina bezpečnostního incidentu
- Zranitelnost — slabé místo aktiva/skupiny aktiv, které lze využít hrozbou/hrozbami
- bezpečný software — chrání CIA (důvěrnost, integritu, dostupnost)
- spolehlivý software — dělá jen to k čemu byl určen
- kvalitní software — spolehlivý a bezpečný
- potřeba definovat bezpečnostní cíle pro aplikaci (kdo bude používat, co znamená pro uživatele bezpečnost, kde bude app provozována, co když...)
- secure by design — bezpečnost se musí zahrnout již do návrhové fáze
- secure by default — základní nastavení bude to nejvíce používané, musí být tedy bezpečné
- decure in deployment — jak zařídit, aby aplikace byla bezpečná i po vydání (aktivní údržba, možnosti bezpečného nastavení)
- myslíte jako útočník
- minimalizujte plochu pro útok
- defense in depth — víceúrovňová bezpečnost
- princip nejnižších oprávnění
- zpětná kompatibilita je problém
- vstup je zlo
- plánujte selhání

Modelování hrozeb



- Studium aplikace — analýza designu aplikace s cílem určit komponenty, tok dat a hranice důvěry
- Dekompozice — seskládání modelu z předchozích zjištění (z kroku 1), od vyšší úrovně níže, 2-3 opakování do hloubky
- Identifikace hrozob — popis komponent, uživatelů, administrátorů, autentizaci, a následně pro každou komponentu zjišťovat, co může vzniknout za problémy, např. použít metodiku STRIDE — výsledky zapsat do stromu / strukturovaného seznamu hrozob — každou hrozbu ohodnotíme, např podle metodiky DREAD
- identifikace zranitelností — seřazení hrozob dle skóre, vybereme ty které se reálně můžou stát, a použijeme jeden z následujících postupů:
 - nechat hrozbu v programu
 - informovat uživatele
 - odstranit problém
 - zmenšit/mitigovat hrozbu

STRIDE

- Spoofing of Identity — podvržení identity uživatele
- Tampering with Data — neautorizované pozměnění dat
- Repudiation — popření transakce uživatelem
- Information Disclosure — únik informací
- Denial of Service — odepření služby
- Elevation of Privilege — neautrizované zvýšení oprávnění

DREAD

- Damage Potential — potenciální rozsah způsobených škod
- Reproducibility — jak snadné je hrozbu zopakovat
- Exploitability — jak snadné je provedení útoku samotné
- Affected Users — množství zasažených uživatelů
- Discoverability — odhad obtížnosti nalezení

3.8 OB-8 (BEK)

Zranitelnosti desktopových aplikací: Přetečení bufferu, DLL hijacking, chyby v C.

Buffer overflow

- přetečení bufferu nastává, přepíšeme-li jeden či více bytů za koncem bufferu
- přetečení může být jak na zásobníku, tak na haldě
- tím se mohou pozměnit důležitá data, např. návratová adresa, zálogy registru...
- obrany:
 - DEP (řeší OS) — Data Execution Prevention — stránky zásobníku jsou nespustitelné
 - ASLR (řeší OS) — Address Space Layout Randomization — náhodné rozvržení adresního prostoru, takže adresy haldy, zásobníku i instrukcí, při každém spuštění se mění
 - kanárci (řeší si program sám) — náhodné hodnoty hlídající zásobník, pokud se přepíšou, program zjistí že má problém

DLL Hijacking

- jedná se o podvržení dynamické knihovny
- útočník vytvoří dynamickou knihovnu se stejným názvem jako knihovna, kterou hledá software, a dle algoritmu vyhledávání dll se může stát, že útočníkova verze bude upřednostněna
- existuje indirect varianta, kdy útočník cílí na použití své dll jinou dll, která je aplikací vyžadována
- bránit se lze nastavením konkrétního místa, kde se DLL má hledat (a nestahovat blbosti)

Chyby v C

- v C existuje mnoho různých funkcí pro práci se "stringy" (pole znaků)
- základní varianty těchto funkcí vůbec neřeší délku bufferu
- existují bezpečnější varianty které řeší velikost bufferu — i ty ale musí být použity správně
- pozor na ukončovací nulu, na různých platformách se k ní přistupuje různě
- knihovna StrSafe — Windows knihovna pro bezpečnou práci se stringy, systematické a jasné pojmenování
- pozor na přetečení datového typu
- pozor na vstup, vstup je zlo
- jak mít bezpečnější kód (v C)?
 - nepoužívat některé funkce
 - důsledně používat ochranu zásobníku
 - na každou skupinu funkcí/třídu napsat testy
 - nechávat kód kontrolovat další osobou
 - statická a dynamická analýza kódu

3.9 OB-9 (BEK)

Řízení přístupových oprávnění. Běh programu při nízkých oprávněních.

ACL

- ACL — Access Control List, mocný nástroj zabezpečení, seznam ACE
- ACE — Access Control Entry (jeden bezpečnostní záznam)
- SO — Securable Object — objekt, který může mít SD — security descriptor
- DACL (Discretionary ACL) — seznam ACE, které povolují/zakazují přístup k SO
- SACL — seznam ACE, které definují typy přístupu k SO, které se logují
- Určení ACL — zajistit prostředky, zjištění požadavků na zabezpečení, určení ACL technologie a převedení požadavků do ACL

Přístupové tokeny

- pro práci s oprávněními se ve windows používají přístupové tokeny
- popisují aktuální bezpečnostní kontext — pravidla a atributy daého procesu/vlákna
- obsahují SID vlastníka, skupiny, uživatelského účtu, výchozí DACL, seznam прав uživatele, ...
- příklad bezpečnostního kontextu — přihlášený uživatel, zadané heslo...

Běh s nejnižším oprávněním

- každý program má dělat pouze k čemu byl určen a nic navíc
- má k tomu používat pouze nutná práva, ostatních se vzdát
- proč programy často potřebují zvýšit práva na administrátorská?
 - ACL — zápis do administrátorských složek
 - problémy s právy — pokud je právo opravdu nutné, holt s tím nic neuděláme, ale pokud ne, nesmíme ho vyžadovat
 - přístup k LSA secrets
- určení nezbytné úrovně práv:
 - zjistit všechny prostředky používané programem
 - zjistit všechny privilegovaná API, které program používá
 - posoudit uživatelský účet, pod kterým má program běžet
 - zjistit všechny práva tokenu
 - určit nezbytná práva pro aplikaci
 - změnit token — zbavit se nepotřebných práv

Windows UAC

- uživatel dostane po oprávnění token definující oprávnění
- administrátor dostane token s vysokými právy
- pro spuštění explorer.exe (a jeho potomků) se použije ořezaný token (ne admin práva)
- admin může svůj plný token využít pro spuštění jiných programů
- UAC kontroluje přístup, případně vyžádá administrátorské povolení
- mody UAC:
 - Elevate without prompting
 - prompt for consent

- prompt for credentials
- UAC komponenty:
 - Application Information Service (AIS) — zajišťuje spouštění dalších aplikací s právy (vyššími)
 - Consent.exe — prezentuje uživateli UAC okno, výsledky předá AIS (je spouštěno AIS)
 - virtuální ovladač (pro nekompatibilní aplikace)
- Manifest — vystavení, jaká práva program chce

3.10 OB-10 (BEK)

Zranitelnosti typu injection.

- Injekce je souhrnný název pro všechny situace, kdy v důsledku nesprávného ošetření vstupu dojde k tomu, že uživatelský vstup (což by z principu měla vždy být data) získá charakter kódu. Tedy jde o spuštění kódu (svého) útočníkem.
- Injekce SQL
 - běžný, snadno exploitovatelný útok s rozsáhlými následky (zvlášť, pokud je použit účet s vysokou úrovňí oprávnění)
 - Obrana: vyhnout se dynamicky generovaným dotazům kde to jde, a důsledně kontrolovat vstup
 - nejlepším řešením jsou parametrizované dotazy
- Injekce souborů
 - jde o útok na include souborů, ježiž jméno je dynamické
 - server načte buď vzdálený útočníkův soubor, nebo nějaký nechtěný lokální soubor
 - do vstupu se místo přímého jména souboru zadá cesta k útočníkovu souboru
 - i přes local file inclusion lze spustit vlastní kód, jen je potřeba ho na server dostat jinak
 - obrana: kontrola includů, nepoužívání přímých odkazů, implementace chroot jailu (virtuální prostředí), vypnout v nastavení PHP include a otevírání cizích souborů
- Cross Site Scripting
 - XSS nastává, když se útočníkův vstup dostane do serverem generovaných stránek a chová se tam jako spustitelný skript
 - tímto skriptem může útočník např. ukrást cookie uživatelů
 - obrana: escapování znaků, whitelist pro znaky, používat hlavičku Content Security Policy (definuje jaký externí obsah smí být načítán)

3.11 OB-11 (ASB)

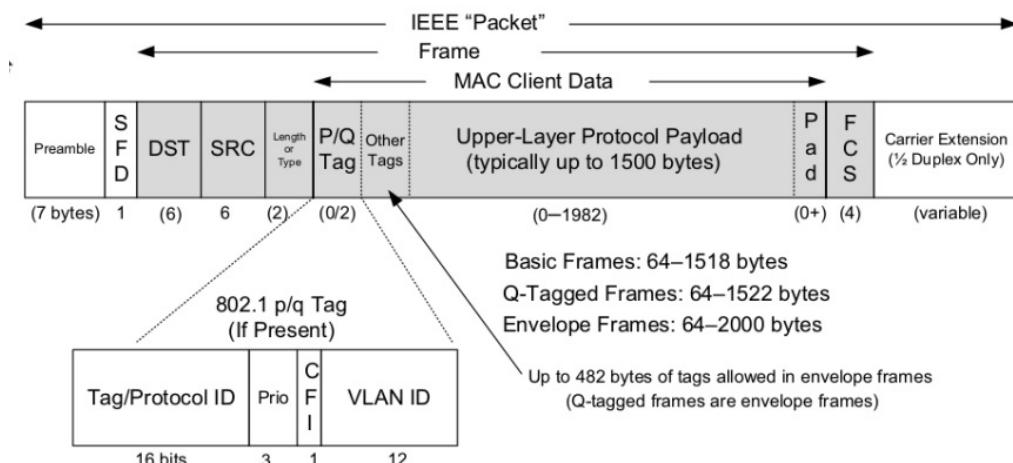
LAN z hlediska kybernetické bezpečnosti. Zranitelnosti protokolů rodiny TCP/IP. Zabezpečení sítí LAN na úrovni síťových zařízení (switches, routers, firewalls). Využití technologie VLAN.

model CIA

- Confidentiality (Důvěrnost)
 - porušení důvěrnosti: odposlech síťového provozu
 - různá úroveň informací v závislosti na vrstvě: aplikační (čtení uživatelských dat), transportní (informace o protistranách a službě), síťová (protistrany, typ transportního protokolu, velikost), ...
 - nemusí jít vždy jen o uživatelská data, ale např informace směrovacích protokolů
- Integrity (Integrita)
 - data jsou nezměněna v průběhu přenosu
 - porušení integrity: úprava přenášených dat
- Availability (Dostupnost)
 - běžící služba, přenos dat sítí (prostě to funguje)
 - porušení dostupnosti: DoS (Denial of Service)

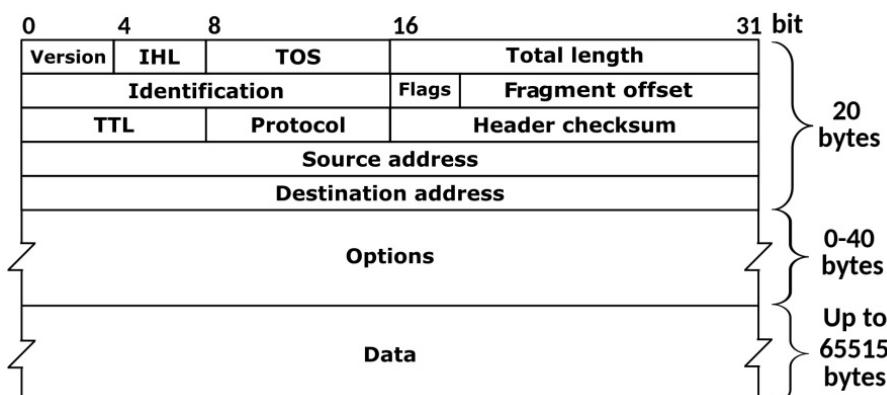
Bezpečnost síťových protokolů

- Ethernet



- otevřená (plaintext) komunikace
- zajišťuje doručení dat (kontrolní součty), ale data lze jak odposlouchávat tak měnit
- u protokolu CSMA/CD lze udělat DoS (místo náhodné doby čekání prostě vysílat hned)
- lze použít VLAN — k packetům se přidají tagy (do jaké VLAN packet patří) — max 4094 VLAN (12 bitů, 2 rezervované hodnoty) — rozdělení broadcastové domény

- IP



- otevřená (plaintext) komunikace
- pouze zapouzdření a přenos dat, neřeší ani správné doručení
- kontrola integrity musí být zajištěna vyšší vrstvou
- IP adresu lze podvrhnout

- UDP

- v hlavičce prakticky jen zdroj, cíl a checksum
- slouží pouze k zapouzdření dat, není garantováno pořadí
- integritu musí řešit vyšší vrstva
- nenáročný na správu spojení (lze využít k DDoS)

- TCP

Bit offset	0–3	4–7	8–15	16–31
0			Source address	
32			Destination address	
64	Zeros		Protocol	TCP length
96		Source port		Destination port
128			Sequence number	
160			Acknowledgement number	
192	Data offset	Reserved	Flags	Window
224			Checksum	Urgent pointer
256			Options (optional)	
256/288+			Data	

- slouží pro doručení v pořadí, neřeší integritu
- integritu musí řešit vyšší vrstva
- komunikující uzly si musí uchovávat informaci o stavu spojení — lze využít k SYN Flood DoS útoku

- ARP

- ARP cache — každý uzel si zapisuje příchozí odpovědi (jakékoliv)
- ARP Spoofing — podvržení linkové adresy
- ARP Cache Poisoning — zneužití ARP Cache oběti pomocí spoofingu — výsledkem je zaslání dat na špatnou linkovou adresu, umožňuje získat pozici MitM (Man in the Middle)
- obrana: static ARP, kontrola na portu switchu

- DHCP

- DHCP server přiděluje dynamicky konfiguraci sítě zařízením
- rogue DHCP server — DHCP server navíc, může vzniknout špatnou konfigurací, nebo útokem
- obrana: DHCP Snooping na switch portu, filtrování DHCP komunikace

- ICMP

- nepřenáší data vyšší vrstvy, ale může přímo i nepřímo sloužit k útokům

Bezpečnost síťových zařízení

- hub

- pracuje na fyzické vrstvě, přeposílá vstup
- dnes už se moc nesetkáváme, překonané zařízení
- bezpečnost 0, rozšiřuje kolizní doménu, každá stanice „slyší“ vše co se v dané kolizní doméně děje, může způsobovat kolize úmyslně

- switch

- pracuje na linkové vrstvě

- rámce jsou přeposílány na konkrétní port na základě paměti switche
- broadcast přeposílán všude
- odděluje kolizní domény, nedochází ke kolizím
- možnosti útoku typu ARP spoofing / MAC flooding
- široké možnosti bezpečné konfigurace switche
 - * vypnutí nepoužitých portů / zásuvek (aby se do volné zásuvky nemohl připojit kdokoliv)
 - * limit počtu MAC adres na portu (obrana proti MAC flooding)
 - * VLAN — logické členění síťových segmentů, rozdělení broadcastových domén (útok VLAN hopping — switch spoofing, pc se tváří jako switch a chce nastavit trunk spojení / double tagging, přidání extra vlan tagu)
 - * spanning tree protocol — potřeba nastavit, aby se nemohl připojit nový (útočníkův) switch
 - * lze nastavit ochranu proti DHCP spoofingu
- router
 - možnost nastavení ACL
- firewall
 - možnost nastavení ACL
- IDS (Intrusion detection system)
- IPS (Intrusion prevention system)

3.12 OB-12 (ASB)

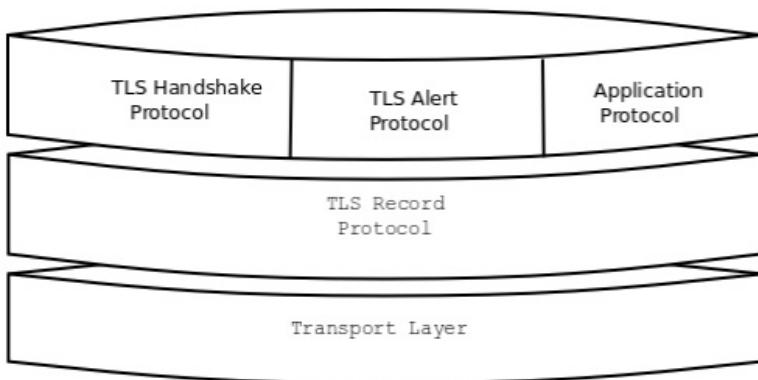
Kryptografické síťové protokoly, využití šifrování a algoritmu Diffie-Hellman. Protokoly TLS a SSH.

Šifrování na různých vrstvách

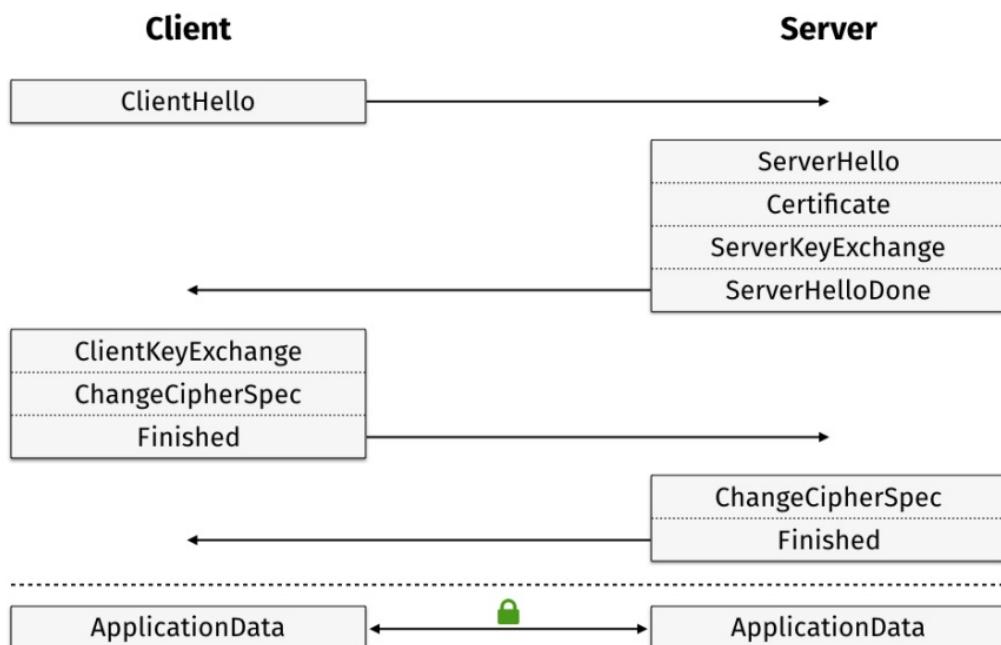
- šifrování na vrstvě x znamená ukrytí dat vrstvy $x + 1$
- aplikační vrstva — šifrování řeší aplikace sama, šifrování jsou různá, závislá na aplikaci
- transportní vrstva — SSL/TLS, ukrývá data a protokol aplikace
- síťová vrstva — některé VPN, IPsec — ukrývá data transportní vrstvy
- linková vrstva — Wi-Fi

TLS

- šifrování na transportní vrstvě
- TLS se skládá z následujících vrstev:



- Record Layer — vrstva zodpovědná za posílání TLS zpráv, skládá se z dalších vrstev
- Handshake Protocol — výběr šifrovací sady, parametrů spojení a ustanovení společných klíčů
- Alert Protocol — přenos chybových zpráv, např. neplatný certifikát
- Application Protocol — šifrovaný protokol aplikační vrstvy
- TLS handshake:



- Client Hello — obsahuje aktuální (UNIXový) čas, náhodné číslo (28B), podporované šifrovací sady a rozšíření
- Server Hello — obsahuje aktuální (UNIXový) čas, náhodné číslo (28B) a podporované šifrovací sady
- Certifikát serveru
- ServerKeyExchange — parametry serveru pro ustanovení klíčů (Diffie-Hellman)
 - * dle Diffie-Hellmann server stanoví hodnoty p (prvočíslo) a G (generátor)
 - * server vygeneruje náhodné číslo a a vypočítá $Y_a = |G^a|_p$
 - * výsledné číslo podepří svým soukromým klíčem (hash(client_random + server_random + server_params)) a odešle
- ServerHelloDone — konec serverKeyExchange
- ClientKeyExchange — klient spočítá svoje $Y_b = |G^b|_p$ a odešle
- v tuto chvíli obě strany mají společnou hodnotu $Y = |G^{a \cdot b}|_p$, která se nazývá pre-master secret
- z pre-master secret a z náhodných čísel se pomocí pseudo-random function vygeneruje master secret (či blok klíčů dle potřeby vybraných šifrovacích funkcí)
- client ChangeCipherSpec — poslední nešifrovaná zpráva od klienta
- client Finished — obsahuje hash všech předchozích zpráv + řetězec "client finished" — slouží pro ověření funkčnosti šifrování a jako ochrana proti replay a tampering útokům
- server ChangeCipherSpec a Finished — podobně jako client
- pro Diffie-Hellmann je možné použít vždy stejné hodnoty a/a/nebo b (static-static, ephemeral-static), ale po kompromitaci kteréhokoliv private-key by šly dešifrovat i zpětně veškerou komunikaci, proto se od TLS 1.3 musí vždy generovat nové (ephemeral-ephemeral) — (perfect) forward secrecy

- TLS 1.3

- povinné použití perfect forward secrecy
- navíc povinné některé šifrové sady
- navíc AEAD šifry (Authenticated encryption with associated data)
- odebraly se nepoužívané funkce jako komprese
- odebrala se statická výměna klíčů
- odebral se operační mód CBC, šifry RC4, DES, 3DES, hashe MD5, SHA-1, SHA-224
- odebraly se slabé a nepoužívané elliptické křížky
- zjednodušení TLS Handshake



SSH

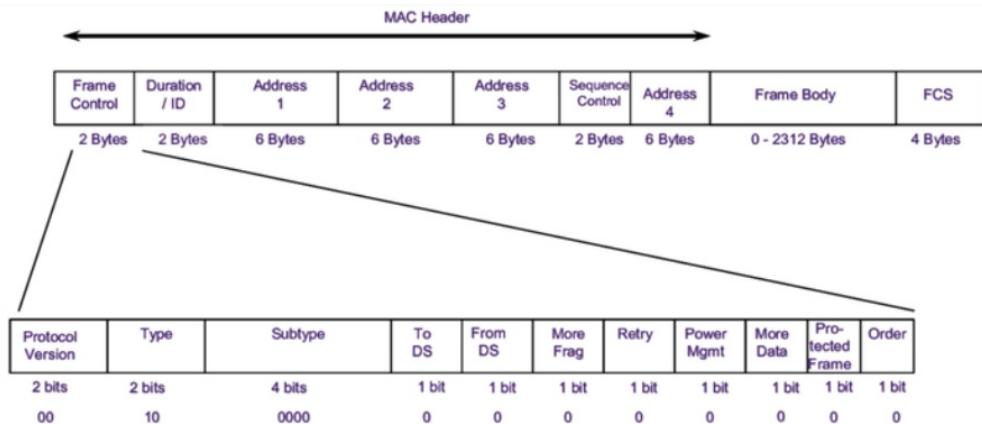
- Secure Shell Protocol
- TCP port 22
- transportní vrstva — typicky běží přes TCP
- autentikační vrstva — řeší autentikaci uživatele a šifrovací algoritmy, řeší autentizaci serveru

- spojovací vrstva — koncept kanálů a služeb , více současných obousměrných kanálů
- typy kanálů — shell, direct-tcpip (přeposílání od klienta na server), forwarded-tcpip (přeposílání ze serveru klientovi)
- SSH Handshake
 - Diffie-Hellman výměna klíčů
 - server klientovi posílá jak jeho vygenerované číslo Y_b , tak certifikát/klíč a podepsaný hash zprávy
 - následně se vypočítá 6 kryptografických hodnot: (IV, šifrovací klíč, integritní klíč) jak pro klienta tak pro server

3.13 OB-13 (ASB)

Bezpečnost bezdrátových sítí technologie Wi-Fi. Bezpečnostní standardy WEP, WPA, WPA2 a WPA3.

- médium přenosu je prostor, tedy je to sdílené médium
- přenos elektromagnetickým vlněním v radiových frekvencích (20 kHz — 300 GHz)
- využívá se CSMA/CA (carrier sense medium access / collision avoidance)
- 13 standardních pásem + 1 nestandardní cca kolem 2,4 GHz (šířka pásem cca 22 MHz, jsou odsunuté od sebe cca o 5MHz, tedy se překrývají částečně)
- struktura WiFi rámce:



- 4 různé MAC adresy s různými rolemi podle typu rámce:

Protocol Version	Type	Subtype	To DS	From DS	More Frag	Retry	Power Mgmt	More Data	Prot. Frame	Order
Frame Control field										
0	0	RA = DA	TA = SA	BSSID	N/A					
0	1	RA = DA	TA = BSSID	SA	N/A					
1	0	RA = BSSID	TA = SA	DA	N/A					
1	1	RA	TA	DA	SA					

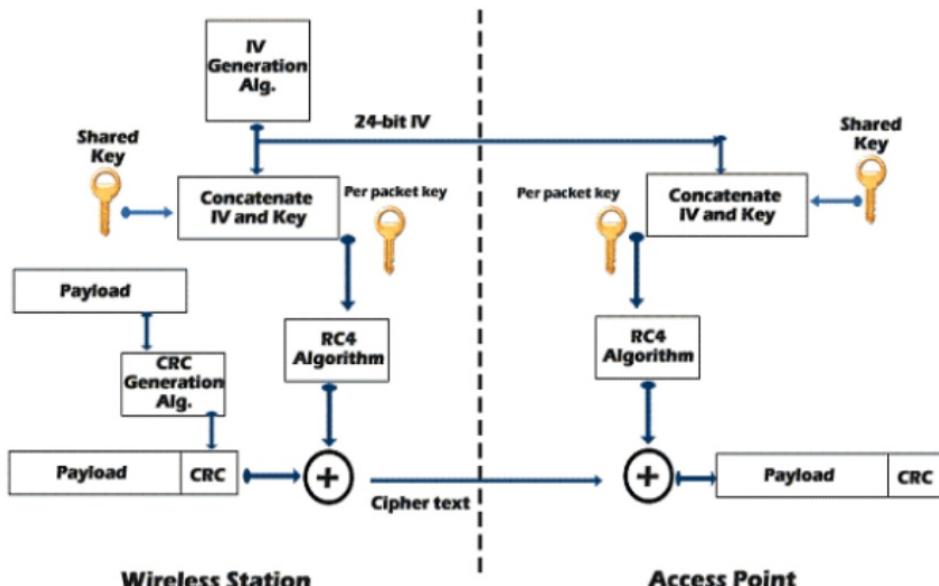
- SA = MAC address of the original sender (wired or wireless)
- DA = MAC address of the final destination (wired or wireless)
- TA = MAC address of the transmitting 802.11 radio
- RA = MAC address of the receiving 802.11 radio
- BSSID = L2 identifier of the basic service set (BSS)

- druhý rámců: management, control (RTS/CTS — request to send, clear to send — pro kontrolu toku dat), data
- management rámce
 - Association request/response (žádost o připojení / odpověď)
 - Reassociation request/response (znovupřipojení)
 - Probe request/response (vyhledávání sítí, buď konkrétního SSID nebo všech)
 - Beacon (ohlašování se — vysílá se v pravidelných intervalech z AP, oznamuje jaké SSID jsou dostupná, spolu s konfigurací sítě)

- Authentication (proces pro ověření, zda je stanice schopna se připojit)
- Disassociation (bez odpovědi — oznámení o odpojení)
- Deauthentication (resetuje stav připojení stanice)
- Action (vyvolají nějakou akci, lze pod to schovat nějaké nové typy rámčů)
- Timing Advertisement (dnes se nepoužívá, předpokládané využití v komunikaci se zařízeními co nemají vlastní čas)

WEP

- Wired Equivalent Privacy
- zastaralý bezpečnostní standard, vydán v roce 1999
- autentizace — stanice pošle request, AP pošle v plaintextu challenge, stanice zašifruje a pošle reponse, dostane zpět potvrzení
- RC4 pro šifrování, CRC32 pro integritu

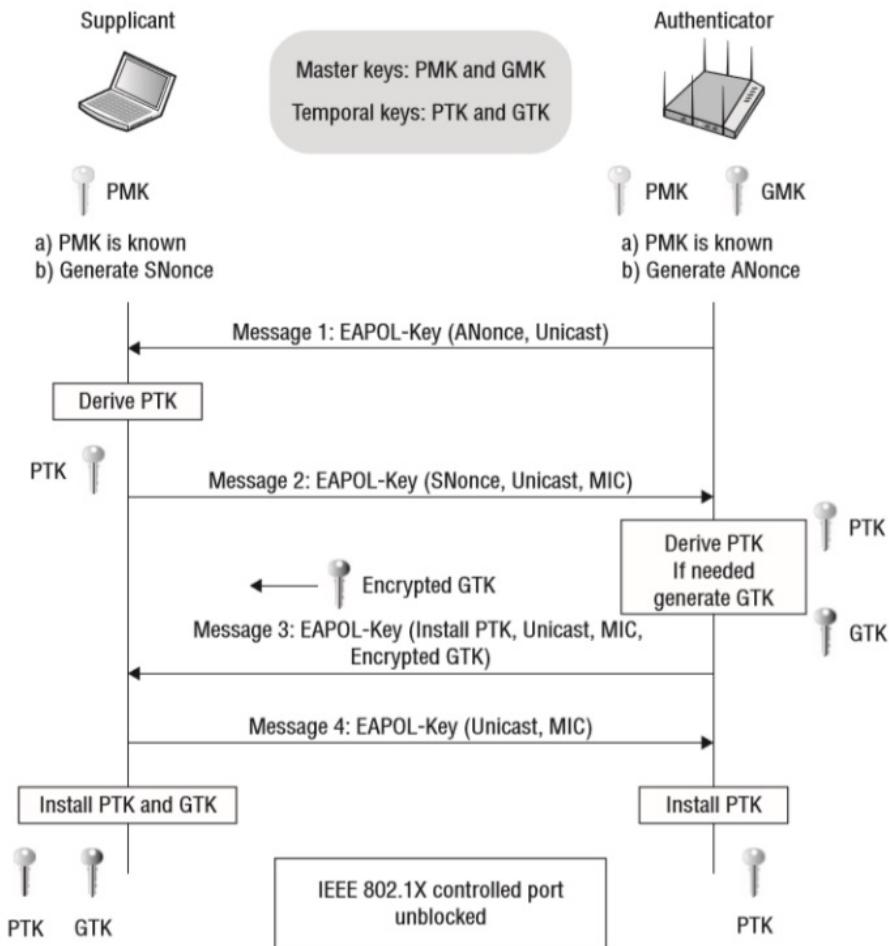


- problémy:

- sdílený klíč je používán pro autentizaci i pro šifrování dat
- znovupoužití keystreamu
- sdílený klíč se nemění (nebo jen málo často)
- IV se sdílí veřejně, je možné zachytit opakující se IV
- teoreticky 24 bitů IV, prakticky kolize po cca 5000 paketech

WPA/WPA2

- Wi-Fi Protected Access
- různé módy autentizace — personal (WPA-PSK, pre-shared key) nebo enterprise
- 4-way handshake (WPA/WPA2)

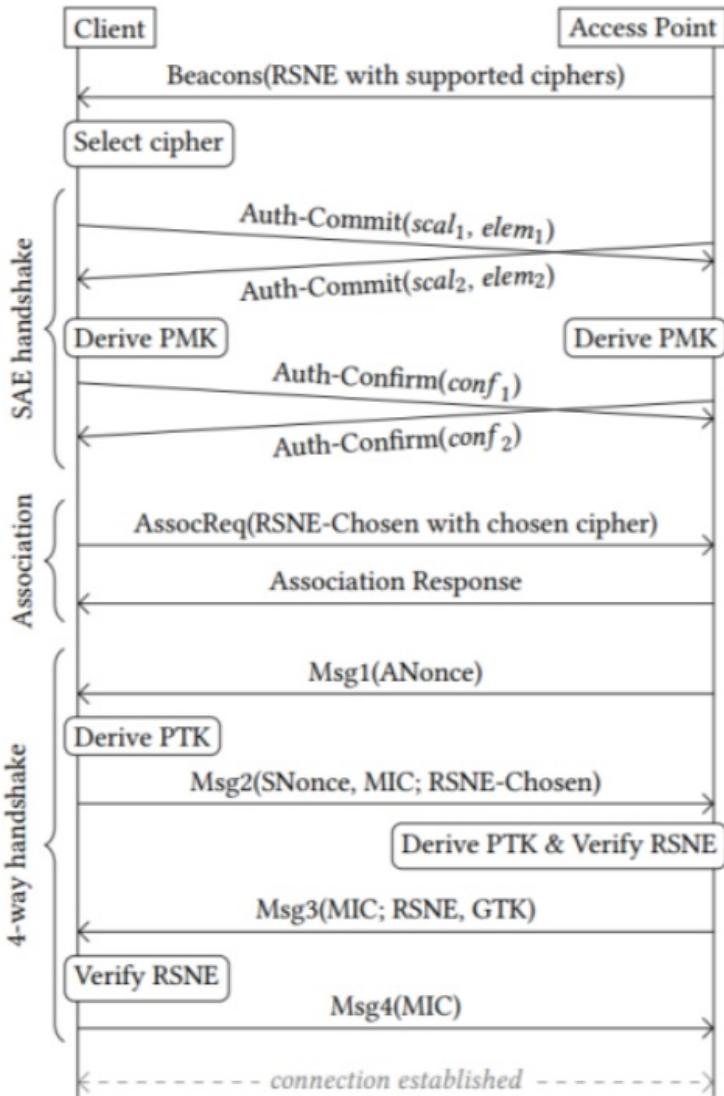


- PSK — pre-shared key
- PMK — pairwise master key = PBKDF2(HMAC-SHA1, PSK, SSID, 4096, 256) — Password-Based Key Derivation Function 2
- PTK — pairwise transit key = PRF(PMK, "Pairwise Key Expansion", MAC1, MAC2, nonce1, nonce2) — Pseudo Random Function
- ANonce — náhodné číslo AP
- SNonce — náhodné číslo stanice
- případně GMK, GTK — Group Master/Temporal Key

- problém s handshake — lze zachytit a offline hádat klíč díky MIC — Message Integrity Code spočítaný pomocí PTK
- šifrování v WPA — TKIP (Temporal Key Integrity Protocol), založený na RC4
- šifrování v WPA2 — CCMP (Ctr mode with CBC-MAC Protocol), založený na AES

WPA3

- obsahuje forward secrecy
- PMK se nevytváří z PSK
- vyřešený problém s handshake:



- SPEKE — Simple Password Exponential Key Exchange
 - A a B si dohodnou prvočíslo p , hash funkci H a sdílené heslo P (může být PSK u personal wifi)
 - oba spočítají generátor $G = H(P)$
 - následuje Diffie-Hellman výměna, spočítají společný klíč $K = |G^{a \cdot b}|_p$
- existuje upravená verze "Dragonfly" využívající eliptických křivek
- stále lze na Wi-Fi útočit, lámat slabá hesla, zkoušat rainbow-tables pro nejznámější SSID, downgrade attack...

3.14 OB-14 (EHA)

Etické hackování a penetrační testování. Metodologie provádění penetračních testů PTES a OWASP.

Penetrační testování (pentest(ing))

- definované, úzké zaměření
- typický cíl: počítačový systém, síťová služba, aplikace...
- identifikace bezpečnostních zranitelností, rizik a nespolehlivých prvků a prostředí
- výsledek je report/zpráva o výsledcích testu
- penetrační tester bývá často zaměřený na jednu oblast

Etické hackování

- široké zaměření
- všemožné hackerské metody objevování zranitelností
- pentest je podmnožina těchto činností
- Etický hacker má široké a hluboké znalosti, typicky nebývá specializovaný

Pojmy:

- Black Hat Hacker
 - různé, typicky škodlivé úmysly
 - primární motivace bývá osobní/finanční zisk
 - aktivity považovány za kyber-zločin
- White Hat Hacker
 - stejné metody jako Black Hat
 - dobré úmysly, s povolením vlastníka systému — tedy legální, zranitelnosti nahlášeny vlastníkovi
 - motivace hlavně finanční odměna
- Grey Hat aproach
 - někdy legální, někdy ilegální aktivity
 - přístup typicky závisí na cílovém systému
 - nemívá škodlivé úmysly, ale občas porušuje zákony či etické standardy
- Hrozba (Threat) — událost/podmínka, která může způsobit škody nebo další následky
- Attack vector — cesta/metoda umožňující škodlivou činnost či odhaluje zranitelnosti
- Attack Surface — množina vektorů útoku
- Zranitelnost (Vulnerability) — slabina, která může být zneužita (použitím attack vectoru), vystavuje systém hrozbě
- Zero-Day vulnerability — nedávno nalezená zranitelnost, dříve neznámá
- Penetrační test — povolená simulace útoku, výsledek je report
- různé metody testování:
 - Black box — tester má stejně informace jako běžný uživatel, žádná předchozí znalost systému
 - White box — tester má stejně informace a přístupy jako vývojový tým aplikace/systému
 - Grey box — podobné white box, ale informace jsou limitované
- existují různé metodologie pro penetrační testování, jako PTES a OWASP

PTES metodologie

- Penetration Test Execution Standard
- návody, jak strukturovat pentest
- 7 fází pentestu
- Pre-engagement interactions
 - domluví se rozsah/zaměření testu (jaké konkrétní aplikace/stroje/systémy/služby budou testovány)
 - domluví se cena
 - domluví se časový rozsah — od kdy do kdy
 - přijatelné praktiky sociálního inženýrství
 - kontakty
 - cíle klienta
 - poskytnuté zdroje klientem
 - pravidla vztahu — oprávnění, testovaná verze, kdy se může testovat, odkud,...
- Intelligence Gathering
 - identifikace a pojmenování cíle
 - získávání informací o cíli — infrastruktura cíle, mapa služeb a systémů, chování zaměstnanců... jednoduše vše co se dá zjistit
- Threat Modeling
 - vysokoúrovňové modelování hrozob
 - * analýza podnikového vlastnictví — intelektuální vlastnictví, data zákazníků, kritičtí zaměstnanci
 - * analýza procesů — technická i lidská infrastruktura
 - * analýza hrozob ze strany lidí
 - * analýza možností hrozob
 - model infrastruktury
 - * síťová topologie
 - * identifikace služeb
 - * mapování možných hrozob a zranitelností
 - je potřebná kooperace klienta
- Vulnerability analysis
 - nejdůležitější (a nejcennější) jsou znalosti a zkušenosti testera
 - hledání využitelných zranitelností
- Exploitation
 - na základě všech předchozích informací se ověřují zranitelnosti, tedy snaha využít domělé zranitelnosti
 - jak univerzální nástroje, tak přímo nástroje ke zranitelnostem
 - analýza zdrojového kódu/reverzní inženýrství
 - fuzzing
- Post Exploitation
 - již jsme získali přístup
 - ochránit data klienta
 - ochránit sebe — neprovést nic co není dohodnuté
 - další analýza, co vše se dá zjistit / ukrást
- Reporting
 - vytvoření reportu, nejlépe na následující 2 části
 - Executive summary — vysokoúrovňová zpráva pro vedení, dopad na byznys, krátké shrnutí vážných nálezů
 - Technical report — podrobný popis celého procesu a postupu, popis nálezů, kroky pro replikaci a návrhy mitigace

OWASP

- seznamy top 10 nejčastějších bezpečnostních problémů v různých kategoriích, např, webové aplikace
- lze použít pro vyzkoušení na testovaném cíli, a jako důkaz že se na nic důležitého nezapomnělo

3.15 OB-15 (EHA)

Standardy pro hodnocení závažnosti bezpečnostních zranitelností. Standard CVSS. Databáze zranitelností.

Hodnocení zranitelností

- existuje hodně ožností jak hodnotit, některé standardizované
- vhodný způsob hodnocení závisí na povaze zranitelnosti
- technická závažnost vs dopad na byznys
- Naivní přístup:
 - závažnosti Low, Medium, High
 - jednoduché na provedení
 - obtížné udržet konzistentní, není objektivní
- DREAD
 - hodnotící model Microsoftu
 - Damage potential
 - Reproducibility
 - Exploitability
 - Affected Users
 - Discoverability
 - každá z kategorií se ohodnotí (1 až 3) a podle součtu se zvolí celková závažnost
 - Low (5-7), Medium (8-11), High (12-15)
- OWASP — Risk Rating Methodology
 - Risk = Likelihood * Impact
 - Step 1: Identifying a Risk
 - Step 2: Factors for Estimating Likelihood
 - Step 3: Factors for Estimating Impact
 - Step 4: Determining Severity of the Risk
 - Step 5: Deciding What to Fix
 - Step 6: Customizing Your Risk Rating Model
- CVSS (Common Vulnerability Scoring System)
 - verze: v2.0, v3.1, v4
 - vzorce pro hodnocení zranitelnosti v závislosti na ohodnocení různých kategorií
 - 3 metriky (spíše skupiny metrik): základní metrika (charakteristika zranitelnosti samotné), temporální metrika (charakteristiky vyvíjející se v čase) a metrika prostředí (zranitelnosti závisející na konkrétní implementaci/prostředí) — používá se vždy jedna z nich
 - v2.0
 - * "základní metrika" je nejčastěji využívaná
 - * obsahuje "Exploitability metrics" a "Impact metrics"
 - * Exploitability — attack vector, access complexity, authentication
 - * impact — confidentiality impact, integrity impact, availability impact
 - * tato verze je ale moc jednoduchá pro některé případy, neobsahuje např. fyzický přístup
 - v3.1
 - * opět je popsána základní metrika:
 - * exploitability — attack vector, access complexity, privileges required, user interaction, scope
 - * impact — confidentiality impact, integrity impact, availability impact
 - v4.0

- * z listopadu 2023 (první povinný běh to neměl vůbec v předmětu)
- * přibyla úplně nová metrika — Supplemental metrics group
- * základní skupina zůstala podobná, jen v impact metrics se CIA triáda rozdělila pro 1) zranitelný systém a 2) následující systém (je to tam celý 2x)

CVSS v2.0.0 Ratings		CVSS v3.1 and v4.0 Ratings	
Severity	Base Score Range	Severity	Base Score Range
Low	0.0-3.9	None	0.0
Medium	4.0-6.9	Low	0.1-3.9
High	7.0-10.0	Medium	4.0-6.9
		High	7.0-8.9
		Critical	9.0-10.0

Databáze zranitelností

- CVE — Common Vulnerabilities and Exposures (udržováno organizací MITRE)
- NVD — the National Vulnerability Database (udržováno NISTem, synchronizováno s CVE, dodatečné informace)
- označení CVE-YYYY-NNNN — CVE-YYYY-NNNNNNNN (4-7 číslic pořadového čísla)

3.16 OB-16 (EHA)

Základní typy zranitelností webových aplikací, jejich testování a náprava. Bezpečnostní mechanismy webových prohlížečů.

Webové aplikace

- struktura: klient (browser), server (webový server)
- používané protokoly: DNS, TCP, TLS, DTLS, HTTP, WebSocket, ...
- dále mnoho různých technologií, komponent jak na serveru tak na klientu — mnoho míst pro zranitelnosti

Základní zranitelnosti

- Cross Site Request Forgery (CSRF/XSRF)
 - Útočník ze své stránky (tedy z klienta oběti) pošle request na jinou stránku/službu, kde pokud je oběť přihlášena, může tento request uspět a tím se provede nějaká akce, např. se pošlou útočníkovi peníze
 - možnosti testování moc nejsou, obrana je na prohlížeči (můžem otestovat max ten)
 - prohlížeč se může (musí) bránit tak, že nedovolí posílání requestů na jinou stránku/zdroj (cross origin)
- Cross Site Scripting (XSS)
 - stored XSS
 - * v databázi je uložený kus javascript kódu, který se spustí při načtení a zobrazení klientovi (např. jméno uživatele, v nějakém příspěvku...)
 - * možnosti řešení: odebrat problematické sekvence znaků (může narušovat funkčnost, špatné řešení), vyescapovat nebezpečné znaky
 - * jak testovat? prostě to zkusím několika způsoby vložit kde to jde (kde se něco ukládá a zobrazuje)
 - reflected XSS
 - * javascript není uložený, ale dostane se do nějaké zobrazované proměnné, která může být ovládnuta, jako např. hodnota z URL adresy
 - * jak řešit? z klientského pohledu — pozor na to co může přijít zvenčí za vstup — jako správce je lepší takové proměnné navázat jinak, např. do payloadu POST requestu, a/nebo kontrolovat jejich hodnotu (VSTUP JE ZLO)
 - * jak testovat? najít možná zranitelná místa (vstup z url) a vyzkoušet
 - DOM-based XSS
 - * založené na dynamicky generovaném obsahu, tedy jiný ”typ” proměnné ovlivňující vzhled/obsah stránky, která může být ovládnuta uživatelem/útočníkem
 - * podobně jako výše — kontrola vstupů pro obranu, identifikace slabých míst a vyzkoušení pro test
- SQL Injection
 - vložení SQL kódu do nějakého vstupu, výsledkem je ovlivnění výsledku / nechtěné změny v DB / vyzrazení informací
 - jak se bránit? ošetřovat vstupy! escapování kritických znaků, prepared statements (nejlepší)
 - jak testovat? najdu vstup co možná jde do databáze, vyzkouším
- Command Injection
 - uživatel ovlivňuje nějakým vstupem příkaz na serveru aplikace, může provádět nechtěné příkazy
 - jak se bránit? blacklist/whitelist znaků, specifická kontrola vstupu pro konkrétní data ... případně upustit od této funkcionality
 - jak hledat? podobně jako předchozí...
- File uploads
 - ovlivňování jména nahrávaného souboru, možné nahrát někam kam autor nechtěl, něco přepsat...

- obrana: kontrola vstupu, blacklist/whitelist znaků
- jak testovat? pokud se něco tváří jako jméno souboru, vyzkoušet různé varianty vyšších složek a podobně
- File inclusion
 - uživatel může do aplikace přidat kód, může to být způsobeno nebezpečnými přímými odkazy
 - může přidat serverový kód, klientský kód, nebo např. /dev/random (způsobit DoS) nebo /etc/passwd (information disclosure)
 - jak bránit? nenechat uživatele volit soubory přímo, ale přes nepřímý odkaz, např. ID
- Server side request forgery (SSRF)
 - útočník může donutit aplikaci k vykonávání požadavků na další místa, např. v lokální síti serveru
 - lze díky tomu např. scanovat vnitřní síť nebo získat citlivá data
 - řešení: nepřímé odkazy, validace vstupů
- XML external entity (XXE)
 - při parsování XML souboru lze přidat odkaz na externí XML entitu, lze tím vykonat kód, přidat soubor...
 - obrana: správně nakonfigurovat parser, nebo zakázat externí entity úplně
- server-side template injection (SSTI)
 - do šablony (template) se dá zadat něco, co způsobí vykonávání kódu na serveru
 - obrana: použití logic-less template engine
- open redirection
 - neověřený uživatel může změnit, kam stránka jiného uživatele po přihlášení přesměruje
 - obrana: vyhnout se možnosti uživatelského nastavení přesměrování
- formula injection
 - týká se aplikací exportujících tabulky, .csv nebo .xlsx — pokud se vyexportuje neověřená data od útočníka, můžou se pak na PC oběti spustit nebezpečná makra
 - prostě ověřovat vstup lol

Obrazy prohlížeče

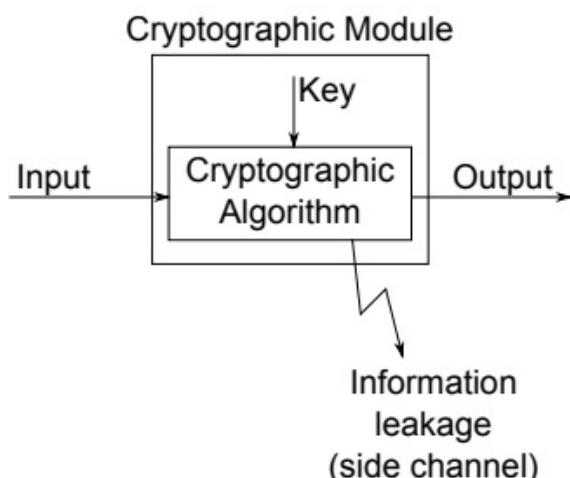
- security headers
- Strict-Transport-Security — řeší povinnost HTTPS
- X-Frame-Options — povolení vykreslování částí stránek v rámečcích (jako iframe)
- X-Content-Type-Options — definování typu obsahu, browser podle toho k obsahu přistupuje
- Content-Security-Policy — kontroluje povolený obsah k načtení

3.17 OB-17 (HWB)

Princip útoků postranními kanály. Typy postranních kanálů, časový útok na porovnání polí, útok jednoduchou odběrovou analýzou (SPA) na šifru RSA.

Princip

- typy útoků:
 - invazivní / semi-invazivní / neinvazivní (z hlediska fyzického průniku)
 - pasivní / aktivní (jen odposlech, nebo i manipulace s rozhraním)
- útoky postranními kanály jsou typicky pasivní a neinvazivní
- postranní kanál je výměna informace mezi kryptografickým modulem a jeho okolím, není to součást jeho normální funkce ale spíš vedlejší příznak způsobený slabinou fyzické či softwarové implementace
- postranním kanálem lze obejít matematické principy, na kterých je založena bezpečnost kryptografických operací
- často dovolí odhadovat klíč po částech



Typy postranních kanálů

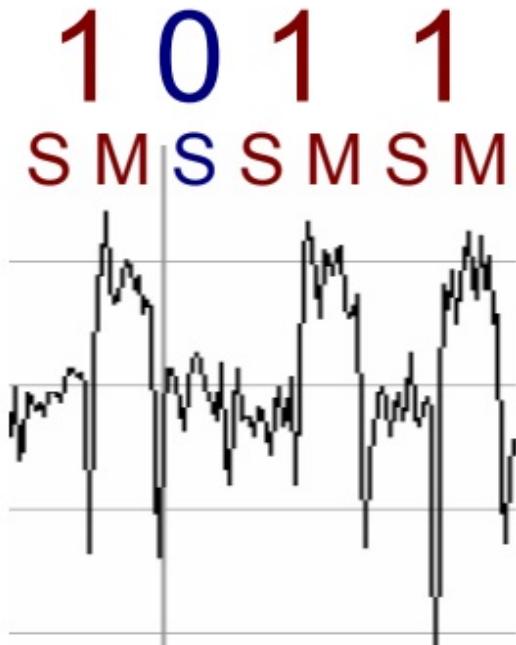
- časový postranní kanál — doba vykonání operace závisí na tajemství, měřením času lze tajemství odhalit
- chybový postranní kanál — chybový kód závisí na utajovaných datech
- odběrový postranní kanál (proudový / výkonový) — proudová spotřeba obvodu závisí na vnitřních datech v průběhu výpočtu šifry (simple/differential power analysis)
- elektromagnetický postranní kanál
- sociální kanál — vyžaduje spolupráci uživatele

Časový útok na porovnání polí

- uvažujme 2 stejně velká pole, která chceme porovnat, dejme tomu zadané heslo vs správné heslo
- typicky porovnáváme po znacích/bytech, a když najdeme neshodu, končíme
- to je ale časový postranní kanál — čím později porovnání skončí, tím více znaků od začátku je správně
- takto by správné heslo šlo hádat po znacích — značně jednodušší než celé heslo naráz
- řešení: musí se vždy nějakým způsobem porovnat celé pole, tedy i když už víme že zadané heslo je špatné

Odběrový útok na RSA

- útok SPA — Simple Power Analysis
- při dešifrování RSA dle vzorce $x = |c^d|_n$ se používá algoritmus Square and Multiply
- špatná implementace: square se provádí vždy (jednoduché, nízký odběr), a multiply se provádí jen v případě, že aktuální bit je 1 (multiply je náročnější, vyšší odběr)
- v případě špatné implementace lze z odběru odhadnout, kdy se operace multiply prováděla a kdy ne, tedy sestavit bity tajné informace



- správná implementace: multiply se provede vždy, pokud ale výsledek není potřeba tak se zahodí

DPA

- Differential Power Analysis
- mnoho průběhů, provádí se analýza signálu ve stejných okamžicích napříč všemi průběhy
- dále se předgenerují všechny možné hodnoty klíče, na základě toho také hypotetické spotřeby průběhů
- z hypotetických průběhů se najde ten, co nejvíce odpovídá reálným průběhům — klíč odhalen

3.18 OB-18 (HWB)

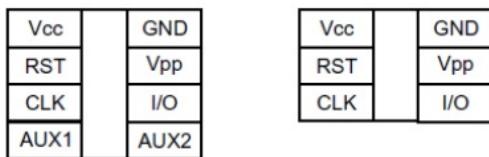
Kontaktní a bezkontaktní čipové karty, princip činnosti a použití. Radiofrekvenční identifikace (RFID) a komunikace v blízkém poli (NFC).

Čipové karty obecně

- obvykle plastové kartičky obsahující integrovaný obvod
- obecněji — nosič informace a bezpečnostních funkcí
- odolnost proti padělání
- lze použít např. pro vícefaktorovou autentizaci
- obsahuje vlastně počítač, který je: konfigurovatelný pomocí API, programovatelný při výrobě nebo i v téřenou (java karty)
- z čeho se skládá: výpočetní jednotky (CPU, kryptografický koprocessor), paměti (RAM, ROM, EEPROM), komunikační zařízení, obvody napájení, senzory

Kontaktní karty

- rozhraní / konektor má typicky 6 nebo 8 pinů



- Inicializace připojení
 - indikace přítomnosti karty
 - zapnutí napájení
 - signál hodin, reset
 - karta pošle odpověď na reset (ATR — Answer to Reset)
 - dekódování ATR
 - volba parametrů protokolu
- přenos dat APDU (Application Protocol Data Unit — něco jako aplikační vrstva) pomocí TPDU (Transmission Protocol Data Unit — něco jako transportní vrstva)
- T=0 — byte oriented protocol
- T=1 — block oriented protocol

Bezkontaktní karty

- zajímá nás hlavně norma ISO 14443 (vysoká frekvence — 13.56MHz, komunikace na blízko), je používána často, jako v platebních kartách, lítačka, pas, průkaz ČVUT...
- RFID (RadioFrekvenční Identifikace)
 - systém, který používá elektromagnetické pole k automatické identifikaci a sledování transponderů připevněných k objektům
 - můžou se přenášet jak data tak napájení
 - transponder — nosič dat připevněný k objektu — anténa + mikročip
 - čtečka — napájí a komunikuje s transpondery — RF modul + řídící jednotka + anténa
- NFC (Near Field Communication)
 - rozšiřuje technologii RFID na bezdrátové datové rozhraní mezi zařízeními
 - sada komunikačních protokolů pro přenos dat mezi dvěma zařízeními na krátkou vzdálenost (do cca 4 cm)

- pasivní transponder — nemá baterii/zdroj napájení, všechna energie čerpána z čtečky
- aktivní transponder — obsahuje baterii
- využití: bezkontaktní platební systémy, sdílení dat, elektronický průkaz, vstupní karta...

3.19 OB-19 (HWB)

Generátory skutečně náhodných čísel (TRNG), příklad konstrukce, základní vlastnosti. Srovnání s pseudonáhodnými generátory (PRNG).

PRNG

- Pseudo Random Number Generator
- deterministické / algoritmické RNG
- má nějaký vnitřní stav a logiku přechodu do dalšího stavu, na základě stavu dává výstup
- výhoda je rychlosť
- jeden z nejznámějších PRNG je lineární kongruenční generátor, definován rekurentním vztahem $X_{n+1} = (a \cdot X_n + c) \bmod m$, X_0 je počáteční hodnota zvaná seed
- X se opakuje nejpozději po m iteracích
- kvalita generátoru závisí na vybraných hodnotách m, a, c
- entropie musí být dodána dobrým seedem
- požadované bezpečnostní vlastnosti:
 - musí projít statistickými testy náhodnosti
 - next-bit test — na základě předchozích bitů nesmí být následující odhadnutelný
 - "state compromise"/ "backtracking resistance" — z aktuálního vnitřního stavu generátoru nesmí jít zpětně rekonstruovat předchozí výstupy

TRNG

- True Random Number Generator
- nedeterministický, nepředvídatelný
- typicky pomalejší a složitější než PRNG
- využívá jako zdroje entropie náhodné fyzikální či jiné procesy, které jsou složité předvídat
- horší statistické vlastnosti — vhodné použít výstup jako seed do PRNG = hybridní RNG
- příklady: radioaktivní rozpad, teplotní šum, kvantové jevy, nestabilita oscilátoru, chování uživatele, parametry prostředí...
- nestabilita oscilátoru: 2 čítače se inkrementují podle 2 oscilátorů, a když jeden čítač dojde na požadovanou hodnotu, z hodnoty druhého se vezmou spodní bity — náhodné
- TRNG založený na SRAM: SRAM po zapnutí napájení má náhodný obsah
- pro více bitů entropie lze pustit vícekrát
- jako postprocessing lze využít hashovací funkci, nebo např. Von Neumannův dekorelátor:

Vstup	Výstup
00, 11	— (vstup se zahodí)
01	0
10	1

3.20 OB-20 (UKB)

Řízení rizik v kybernetické bezpečnosti, proces řízení rizik, základní pojmy (zranitelnost, hrozby, rizika, aktiva) a související aktivity (analýza rizik, hodnocení primárních aktiv, reakci na rizika).

- proč řídíme rizika?

Nikdy není možné zabezpečit vše na 100 %, je potřeba se připravit na možnost napadení a udržovat zabezpečení aktuální a silné.

- řízení rizik:

Neustálý proces, který má za úkol neustále dokola identifikovat rizika v organizaci a navrhovat opatření, aby případné škody v případě působení rizika byly co nejmenší.

- analýza rizik:

Identifikace rizik — provádí se pravidelně a dle potřeby např. při implementaci nového systému. Riziko je hrozba primárním aktivum. Postup:

- určení rozsahu analýzy a míry detailu
- identifikace primárních aktiv
- identifikace podpůrných aktiv
- identifikace rizik, tedy hrozeb primárním aktivum

- primární aktivum:

Informace či procesy/služby, které organizace potřebuje pro svoje fungování. Příklady informace: seznam zákazníků, receptura, výsledky výzkumu. Příklady procesu: vytvoření objednávky, zápis studentů do studia, čištění odpadních vod.

- podpůrné aktivum:

Jsou to např. informační systémy, servery, datová centra, síťové prvky... obecně komponenty, bez kterých nemohou fungovat primární aktiva. Vazba mezi primárními aktivy a podpůrnými je M:N.

- PDCA cyklus řízení rizik (Plan-Do-Check-Act)

Začíná se na malém detailu, postupně se úroveň detailu zvětšuje. Postupně se tím zdokonaluje systém a bezpečnost.

- CIA triáda (Confidentiality, Integrity, Availability)

Aktiva zabezpečujeme od nejdůležitějšího, každé nejprve ohodnotíme dle CIA. Každé kategorii přiřadí garant aktiva nějaké ohodnocení (jak velký problém bude, když se poruší C, I a A)

- Hrozba

Událost nebo aktivita, která hrozí nějakému aktivu, a pokud nastane, tak způsobí nějakou škodu.

- Zranitelnost

Nedostatek v návrhu nebo ve vlastnostech aktiva, který může být zneužit hrozbou (či hrozbami).

- hodnocení hrozeb a zranitelností

Při tvorbě rizikových scénářů provádíme také hodnocení hrozeb a zranitelnosti vůči danému primárnímu aktivu. U hrozby řešíme s jakou pravděpodobností může nastat, u zranitelnosti jak snadné a pravděpodobné je její zneužití. Pro taková hodnocení existují různé stupnice.

- Riziko

Kombinace aktivum-zranitelnost-hrozba. Je to jev/událost, kdy hrozba zneužije zranitelnost, může s nějakou pravděpodobností nastat a nebo taky ne, a vznikne incident, který způsobí nějakou škodu. Riziko = dopad (hodnota aktiva) * hrozba * zranitelnost.

- reakce na rizika

Máme více možností reakce na identifikovaná rizika:

- Akceptovat — přijmout možnost případného incidentu — používá se, když řešení je dražší než dopad
- Přenesení — přesměrovat škodu jinam, např. pojištění
- Mitigace (snížení) — nejčastější způsob, riziko se sníží

- Vyhnutí — pokud je riziko příliš velké a nelze použít předchozí možnosti, projekt (s tímto rizikem) se nerealizuje

Mitigace se typicky provádí přijetím nějakých nápravných opatření, technických a/nebo organizačních.

- dokumenty řízení rizik

- Jmenování garantů primárních aktiv
- Prohlášení o aplikovatelnosti — aktuální stav řízení rizik
- Definování rozsahu analýzy rizik
- Plán zvládání rizik — organizační proces, co s jakým rizikem udělat

3.21 OB-21 (UKB)

Hrozby síťové bezpečnosti, základní kategorie síťových útoků, principy DoS útoků (konkrétní příklady/techniky). Principy obrany proti síťovým útokům v moderních sítích.

- je nutné zabezpečit všechny vrstvy (ISO/OSI model)
- základní rozdělení útoků:
 - pasivní — odposlech, analýza provozu
 - aktivní — MAC/IP spoofing, modifikace/přerušení komunikace
- příklady útoků:
 - ping of death — DoS, poslání pingu (ICMP) tak, že sestavený síťový packet měl více než 64 KiB, OS si s tím dříve neuměl poradit ⇒ buffer overflow
 - ping flood — DoS, zaplavení oběti pingy (ICMP Echo request), využíval pomalých linek připojení
 - smurf attack — DoS, útočník zaslal ping requesty s podvrženou zdrojovou adresou na různá zařízení (klidně broadcast), která pak polala reply na oběť — amplification attack, malý payload útočníka, velké zahlcení oběti
 - Man in the Middle — útočník se vetře do komunikace mezi 2 uzly, díky tomu vidí a může měnit jejich komunikace (docílí se např. ARP poisoning, vystavení falešného AP, podvržení domény, napadení webových stránek...)
 - SYN flood attack — oběť má omezený počet TCP spojení, která může otevřít, útočník všechny vyčerpá tím, že naváže hodně spojení (počátkem TCP handshake — zprávou SYN), oběť je ve stavu DoS
- DoS (Denial of Service) obecně – snaha vyřadit službu z provozu, lze provést např. zahlcením linky, vyčerpáním HW zdrojů oběti zneužitím špatně nastavené aplikace poskytující službu
- DDoS (Distributed DoS) — DoS provedený z více míst najednou
- možnosti obrany:
 - zamezení fyzického přístupu k síťovým prvkům, zamezení připojení ”kamkoliv”
 - segmentace sítě — síťové oddělení např. různých oddělení firmy, každý vidí jen to co potřebuje
 - firewall — pravidla pro provoz, filtrování provozu, hlídání provozu, inspekce paketů
 - používání šifrované komunikace
 - proxy server — kontroluje obsah komunikace
 - DMZ (demilitarizovaná zóna) — jediná část sítě přístupná přímo z internetu
 - logy
 - IDS/IPS
 - omezení přístupu, autentizace a autorizace, nejmenší oprávnění...

3.22 OB-22 (UKB)

Bezpečnost kyber-fyzikálních systémů a Internetu věcí, specifické hrozby a specifika zabezpečení (ve srovnání s tradiční IT bezpečností). Možnosti detekce útoků vedených proti kyber-fyzikálních systémů. Purdue model pro informační a komunikační systémy (ICS) a specifika jejich bezpečnosti.

- Internet of Things — síť chytrých zařízení (chytrá domácnost, chytrá města, kamery, senzory, chytrá auta, ...)
- CPS (Cyber-Physical Systems) — propojuje fyzická zařízení s IT, může ovládat/monitorovat fyzické procesy v reálném čase (průmyslová zařízení, zdravotní vybavení, dopravní systémy...)
- rozdíly IoT, CPS oproti tradiční IT bezpečnosti:
 - selhání může způsobit fyzické škody/smrt
 - cíle útočníků typicky bývají způsobit škody
 - IoT zařízení mají většinou omezené zdroje (pomalé CPU, malou paměť...), tedy nelze vždy aplikovat stejná bezpečnostní řešení
 - zařízení mohou být provozována lidmi bez znalosti technologie/bezpečnosti
 - výrobní náklady jsou tlačeny na minimum
 - doba na návrh, vývoj, implementaci, vývoj a vyřazení z provozu jsou v řádu desetiletí (např. elektrárny)
- možnosti prevence útoků:
 - lze i tradičním přístupem (firewall atd), ale je to nepraktické pro IoT
 - navrhovat systémy, ve kterých lze zabezpečení průběžně aktualizovat
 - zajistit dodatečná bezpečnostní řešení pro stávající starší systémy
 - přidání extra zařízení ”na drát”, které kontroluje pakety
 - extra zařízení blokující bezdrátové připojení neautorizovaných zařízení
 - ”lehké”(lightweight) kryptografické algoritmy
 - bezpečná mikrojádra (bezpečný malý OS)
- detekce útoků na CPS
 - důvěryhodný ověřovatel vnitřního stavu IoT zařízení, např. kontrola obsahu RAM
 - sledování interakcí CPS (něco jako IDS v klasických sítích, ale jednodušší, protože komunikace je méně pestrá a stabilnější)
 - sledování fyzických stavů / konfigurací efektorů — např konfigurace co jsme zatím neviděli, nebo fyzikálně nesmyslná data senzoru
 - aktivní detekce — pravidelné dotazování zařízení, detekce anomalií v odpovědích
- mitigace
 - konzervativní kontrola — provozování systému s dostatečnými bezpečnostními rezervami
 - kontrola senzorů navzájem — zda data dávají smysl
 - omezení aktivace — v případě útočníka v systému chvíli trvá než může nějak zasáhnout a něco změnit
 - kontrola akcí systému — referenční monitor, nepovede akce k nebezpečnému chování?

3.23 OB-23 (ZSB)

Digitální forenzní analýza, základní principy a procesy forenzní analýzy, digitální důkaz a digitální stopa, proces akvizice dat.

- **Digitální forenzní věda** se zabývá zkoumáním a obnovením materiálu nalezeného na digitálních zařízeních, často v kontextu kyberzločinů
- **forenzní analýza** je proces analyzování důkazů za účelem identifikace nebo rekonstrukce událostí, které způsobily tyto stopy
- **digitální forenzní analýza** je proces analyzování digitálních důkazů za účelem identifikace nebo rekonstrukce událostí, které způsobily tyto digitální stopy
- jednodušeji DFA je proces odhalování a interpretace elektronických dat
- DFA se snaží zachovat důkazy v jejich nejoriginálnější podobě

Základní principy DFA

- Legalita
- Integrita
- Opakovatelnost (analýza nesmí mít vliv na kvalitu stopy)
- Možnost znovuprohlédnutí
- Nezávislost
- Expertíza
- Dokumentabilita

Procesy DFA

- role investigátora — zjistit Kdo, Co, Kdy, Jak, Čím a Proč
- digitální důkaz — jakákoli důkazní informace přenášená/uložená v digitální podobě
- využití DFA — dokazování kriminální aktivity, zkoumání bezpečnostních incidentů, ne-kriminální procesy

Vlastnosti digitálních důkazů

- nemateriálnost — data jako taková jsou nehmotná, potřebují hmotné médium, které je sice součástí evidence, ale forenzní image dat se považují za originál
- latence digitální stopy — digitální informace nelze spatřit pouhým okem, zároveň mohou navíc ukryty nebo zašifrovány
- časová identifikovatelnost — digitální stopy často mívají značku, kdy byly vytvořeny
- obsahují mnoho informací
- problematická odolnost
- nízká hustota informací
- vysoká dynamika vývoje — různorodé typy médií
- systémová dynamika — data se často mění a přepisují
- složitá identifikace jedince — je obtížné/nemožné s jistotou přiřadit člověka ke konkrétním akcím
- různé typy dat a platform
- problém s určením fyzického úložiště dat — např. cloud
- obrana přístupu — šifrování (zašifrovaná data nemají význam)
- obnovitelnost — některá zničená data lze obnovit

DFA — obrazy disků

- obecně image/obraz reprezentuje obsah a strukturu logického disku či úložného zařízení
- image disků se využívají v DFA kvůli zachování dat — speciální formáty ukládající metadata a kontrolní součty/hashe obsahu
- nejpoužívanější formáty jsou E01 a RAW/DD

Akvizice dat

- získání dat z fyzického přenašeče — vytvoření forenzního image
- u image kontrola kontrolních součtů (kontrola integrity důkazu)

3.24 OB-24 (ZSB)

Forenzní analýza souborových systémů, principy, možnosti obnovy smazaných dat.

Soubory

- pojmenovaná množina dat přesně definovaného formátu, uložená na datovém médiu
- na uživatelské úrovni lze typ souboru rozeznat dle přípony
- strojově lze typ rozeznat dle přípony a podle hlavičky uvnitř souboru
- existují otevřeně specifikované formáty a uzavřené formáty
- typy souborů:
 - textové/dokumentové soubory (txt, doc, docx, pdf, xls, pages, html,...)
 - obrázkové soubory (rastr — jpg, png, bmp,...) (vektor — psd, cdr, ai,...)
 - zvukové soubory (wav, mp3, midi (není zvukovej lol), wma,...)
 - video (avi, mpg, mov, mp4,...)
 - archivy (zip, rar, gz,...)
 - spustitelné soubory (exe, dmg,...)
 - speciální...

Souborové systémy

- organizuje data (soubory do složek)
- obsahuje metadata popisující strukturu a některé vlastnosti souborů
- příklady: FAT32, NTFS, EXT3, EXFAT...

Obnova smazaných dat

- co jsou to smazaná data?
 - nechťěné smazání souborů či složek
 - reformátování filesystému
 - poškození superbloku/FAT/MFT
 - poškození záznamů o oddílech (partitions)
 - fyzické poškození hard disku
- informace na discích se nemažou, když se smaže soubor
- segmenty/clustery se pouze označí jako volné
- obnovit se dá 2 způsoby — pomocí metadat filesystému, a pomocí hledání hlaviček známých typů souborů
- obnovení pomoví metadat vydá strukturu složek, názvy složek a souborů, metadata souborů a složek
- obnovení pomocí file carvingu (hledání pomocí hlaviček) vydá surové bloky dat bez metadat a názvů složek a souborů

3.25 OB-25 (ZSB)

Řízení přístupu v operačních systémech, obecný model řízení přístupu - Trusted Computing Base, vícestupňové (multilevel) a multilaterální modely, přístupy Discretionary Access Control a Mandatory Access Control, konkrétní příklady implementace v OS.

Model hrozeb OS

- důvěrnost — ztráta dat procesu nebo uživatelské přihlašovací údaje
- integrita — úprava přístupových práv, úprava dat jiného procesu, instalace malware
- dostupnost — pád systému, využití všech systémových zdrojů, způsobení nedostupnosti dat
- možné útoky:
 - škodlivý software — boot kits, škodlivá rozšíření (drivery, moduly)
 - chyby a bugy — paměťové poškození, paměťové chyby, leaknutí dat
 - postranní kanály — hardwarové, spekulativní, softwarové
 - DoS — vyčerpání zdrojů, deadlock
- vektory útoku:
 - kód běžící v user space
 - rozšíření OS
 - kód z internetu (např. javascript)
 - škodlivé periferie
 - vzdálený systém
- různé (historicky) návrhy OS
 - single domain — žádná izolace, vše běží přímo na HW
 - monolitické OS — moderní univerzální OS, každá aplikace ve vlastní bezpečnostní doméně, OS jako celek je pod všemi aplikacemi
 - microkernel based multi-server OS — všechny součásti OS mají vlastní doménu
 - unikernel/library OS — aplikace spolu s OS knihovnou ve svých doménách, pod nimi samotný plánovač

Řízení přístupu v OS

- principy:
 - izolace — oddělení bezpečnostních domén
 - domény by neměly mít mnoho společných mechanismů
 - zprostředkování komunikace mezi domény by mělo být fail-secure
 - pouze povolené domény by měly mít přístup ke zdroji
- primitiva pro izolaci a zprostředkování — autentizace, autorizace, auditování, accountability (účtovatelnost?)
- příklady izolace
 - android — každá aplikace má svoje user id, nemohou spolu komunikovat a mají omezený přístup k OS
 - iOS — sice stejné user-id, ale všechny aplikace běží v chroot jailu — každá má svoje virtuální prostředí a nemůže přistupovat k souborům mimo
- obecný model řízení přístupu — Trusted Computing Base (TCB)
 - TCB komponenty jsou zodpovědné za vynucování bezpečnostních politik
 - Reference monitor — komponenta TCB validující přístup ke zdrojům, vynucuje bezpečnostní politiky

- TCB musí být kompletní (vše musí být vyalidováno přes TCB), izolované (chráněné před nedovolenou změnou) a ověřitelné (TCB splňuje návrhové specifikace)
- multilevel security model — různé úrovně důvěrnosti informací (open , confidential, secret, top secret) — pochází z armády
- multi-lateral security model — odděluje různé kategorie informací
- bezpečnostní politiky — kdo nastavuje?
 - Discretionary Access Control (DAC) — nastavuje vlastník — typický příklad: ACL
 - Mandatory Access Control (MAC) — v celém systému stejné politiky — většinou víceúrovňový model
- v SELinux je implementováno jak DAC tak MAC
- SELinux — více kategorií, v každé různé úrovně důvěrnosti
- existuje i např. ve Windows, kontrola úrovně integrity (integrity level — od untrusted (0) po installer (5))

3.26 OB-26 (ZSB)

Řízení zranitelností (vulnerability management) a záplatování (patch management), základní pojmy a proces řízení zranitelností.

- proces řešení technických zranitelností jako klíč k zajištění bezpečnosti
- zranitelnost — slabina v systému nebo jeho součástech
- technická zranitelnost — slabina v hardware, software nebo firmware či chyba v designu a konfiguraci systému, které umožňuje hrozbám způsobit bezpečnostní událost/incident
- řízení zranitelností je důležitá činnost k zabezpečení systému, který často obsahuje technické zranitelnosti (kolem 20k dokumentovaných zranitelností ročně)

Proces řízení zranitelností

- asset inventory — identifikace součástí systému/sítě
- asset prioritization — určení, v jakém pořadí/prioritě budeme součásti/zařízení/systémy řešit
- vulnerability assesment — identifikace zranitelností na součástech/zařízeních/systémech, typicky se používají automatizované nástroje — ohodnocení zranitelností a jejich dopadu na byznys
- remediation (náprava) — začíná se od nejzávažnějších zranitelností, čím závažnější, tím rychleji se musí vyřešit a nasadit náprava
 - součástí je Patch Management (PM) — řízení záplatování
 - kontrolují se systémy a jejich případné existující záplaty
 - vytvoří se záplata (či se použije záplata od výrobce, pokud se jedná o cizí systém)
 - prioritizace záplat
 - otestování
 - nasazení
 - dokumentace
- v případě kdy záplata neexistuje, jsou použita jiná nápravná opatření, či je systém vyřazen z provozu
- verification — ověření že byla zranitelnost odstraněna

Asset Business Value	Vulnerability severity				
	5 (Urgent)	4 (Critical)	3 (Serious)	2 (Medium)	1 (Minimal)
Criticality 5 (Very High)	ASAP remediation in hours, highest priority, decision about server shutdown must be given	ASAP remediation in hours / days, decision about server shutdown must be given	7 days	4 weeks / next release / maintenance window / ...	next release / maintenance window / ...
Criticality 4 (High)	ASAP remediation in hours, highest priority, decision about server shutdown must be given	7 days	7 days	4 weeks / next release / maintenance window / ...	next release / maintenance window / ...
Criticality 3 (Medium)	7 days	14 days	2 weeks	next release / maintenance window / ...	next release / maintenance window / ...