

IOS Teorie

Poslední úprava: **02.07.2020**

Vytvořeno by **btk. & Lúčkou**

DEADLOCK

Zadání

Deadlock, 4 coffmanovy podmínky, princip prevence, popis používaných přístupů, základní princip vyhýbání se uváznutí, graf alokace zdrojů a jeho používání ve vyhýbání se, jaký další klasický mechanismus se využívá (ne verifikace).

Definice

Uváznutím (deadlockem) při přístupu ke zdrojům s výlučným (omezeným) přístupem rozumíme situaci, kdy každý proces z nějaké neprázdné množiny procesů je pozastaven a čeká na uvolnění nějakého zdroje s výlučným (omezeným) přístupem vlastněného nějakým procesem z dané množiny, který jediný může tento zdroj uvolnit a to až po dokončení práce s ním.

Coffmanovy podmínky

Aby deadlock mohl nastat, musí být splněny všechny Coffmanovy podmínky.

1. Vzájemné vyloučení při používání prostředků
2. Vlastnictví alespoň jednoho zdroje, pozastavení a čekání na další
3. Prostředky může uvolnit pouze ten proces, který je vlastní, a to po dokončení jejich využití
4. Cyklická závislost na sebe čekajících procesů (nesouvisí s aktivním čekáním tzn while (///) ;)

Řešení uváznutí

- Prevence uváznutí
- Vyhýbání se uváznutí
- Detekce a zotavení

Prevence uváznutí

Snaha **zrušit platnost alespoň jedné z podmínek uváznutí** - když se jedna z Coffmanových podmínek zneplatní, nemůže dojít k uváznutí.

1. Nepoužívat žádné sdílené prostředky, nebo užívat takové sdílené prostředky, které umožňují skutečně současný sdílený přístup a tedy není nutné vzájemné vyloučení procesů
2. Proces může požádat o prostředky tehdy, pokud žádné prostředky nevlastní a pokud chce více prostředků, je potřeba je zamknout všechny zároveň
3. Zdroje jsou přiděleny procesu pouze tehdy, pokud je možné jej opravdu získat. Pokud proces požádá o zdroj, který není možné získat, jsou mu odebrány všechny prostředky a proces je zabit nebo čeká až mu budou moct být prostředky přiřazeny.
4. Zdroje jsou očíslovány. Je možné tyto zdroje získávat pouze od určitého pořadí, které vylučuje vznik cyklické závislosti. Například od nejmenšího k největšímu

Vyhýbání se uváznutí

- Procesy musí oznamovat informace o tom, jaké zdroje a jak budou používat.
- V nejjednodušším případě se jedná o maximální počet současně požadovaných zdrojů jednotlivých typů.
- Systém přidělování zdrojů si vede informace co tyto procesy deklarovaly, o jejich možných požadavcích a o aktuálním stavu přidělování
- Tyto informace rozhodují o tom, které požadavky mohou být uspokojeny tak, aby nemohla vzniknout cyklická závislost na sebe čekajících procesů i v nejhorší možné situaci.

Graf alokace zdrojů

- Veden systémem, který zdroje přiděluje
- Průběžně si udržuje graf vztahů mezi procesy a zdroji
- Dva typy uzlů
 - Procesy a zdroje
- 3 typy hran
 - Hrana od zdroje k procesu - který zdroj je kým vlastněn
 - Hrany od procesu ke zdroji - jaký proces o který zdroj žádá a může požádat
- Zdroj je přidělen pouze tehdy, pokud systém posoudí, že v budoucnu nemůže nastat cyklická závislost
 - Cvičně se provede otočení žádostí na vlastnictví, pokud v grafu vznikne cyklus, znamená to, že v budoucnu by mohl vzniknout deadlock.
 - V takovém případě systém nedovolí přidělení zdroje

Detekce uváznutí

- Uváznutí může vzniknout, periodicky přitom se detekuje, jestli k uváznutí nedošlo.
Pokud ano, provede se zotavení
- Vedení grafu vlastnictví zdrojů
 - Dva typy uzlů
 - Procesy a zdroje
 - 2 typy hran
 - Žádosti o zdroj a vlastnictví zdroje
 - Pokud vznikne v grafu cyklus, víme, že uváznutí nastalo

Zotavení

- Některým procesům, které uvázly, odeberu zdroje
- Proces se buď zruší, nebo se pozastaví s tím, že může pokračovat, až bude moci získat všechny zdroje, které potřebuje
- Možný problém s rozpracovanými operacemi
 - Nutnost nechat systém uváznutý, nebo se spokojit s nekonzistencemi
 - Systém může být navržen tak, že při zabití procesu se nezabije ihned, ale prvně se provede zotavení (rollback- anulace operací)

VÝPADKY STRÁNEK

Zadání

Co musí udělat OS při výpadku stránky (více variant). Základní myšlenka FIFO, výhody nevýhody a podoba použití v praxi. Základní myšlenka LRU pro výběr victim page, základní výhoda a nevýhoda, princip dvou variant ve kterých se užívá v praxi.

Co musí OS udělat

- Kontrola, zda se proces neodkazuje mimo přidělený adresový prostor (v RAM)
- Alokace rámce
 - Pokud existuje volný rámec, použijeme jej
 - Pokud není
 - Vybereme vhodnou stránku s přiděleným rámcem (victim page).
Tato stránka je odložena na swap a použijeme její rámec
- Inicializace stránky po alokaci je závislá na předchozím stavu stránky
 - První odkaz na stránku
 - kód a inicializovaná data se načtou z programu
 - vše ostatní z bezpeč. důvodů vynulujeme
 - Stránky byla v minulosti uvolněna z FAP
 - kód a inicializovaná data se znova načtou z programu
 - Pouze pokud nejdou přepisovat kódové stránky
 - Pokud byla stránka modifikována, je ve swapu a musí se načíst zpět do FAP, jinak je obsah vynulován
- Úprava tabulky stránek (Namapování zpřístupňované stránky na přidělený rámec)
- Proces je připraven k opakování instrukce, která výpadek způsobila (je ve stavu připravený)

FIFO

Algoritmus FIFO (First In First Out) odstraňuje stránku, která byla zavedena do paměti před nejdelší dobou a doposud nebyla odstraněna.

- **Výhody**
 - Jednoduchá implementace
- **Nevýhody**
 - Může odstranit "starou", ale stále často používanou stránkou
 - Trpí tzv. Beladyho anomálií - více výpadků při zvětšení paměti

LRU

Algoritmus LRU odkládá nejdéle nepoužitou stránku

- **Výhody**
 - Velmi dobrá approximace hypotetického ideálního algoritmu
- **Nevýhody**
 - Někdy se uvádějí problémy s cyklickými průchody rozsáhlými poli
 - Problematická implementace vyžadující výraznou HW podporu
 - Označování stránek časovým razítkem posledního přístupu
 - Udržování zásobníku stránek, vrchol=poslední použitá stránka
- Používají se approximace LRU
 - **Aproximace pomocí omezené historie**
 - Referenční bit stránky je HW nastaven při každém přístupu a jádro si vede omezenou historii tohoto bitu pro jednotlivé stránky.
 - Periodicky posouvá obsah historie doprava a na nejlevější pozici uloží aktuální hodnotu referenčního bitu a vynuluje ho.
 - Oběť je vybrána jako stránka s nejnižší číselnou hodnotou historie
 - **Aproximace "druhá šance" (aka clock algorithm)**
 - Stránky jsou v kruhovém seznamu, postupujeme a nulujeme referenční bit, odstraníme první stránku, která již nulový referenční bit má.

-
- **Sdílené kritické sekce**
 - Úseky kódu takové, že provádění 1 z nich jedním procesem vylučuje provádění libovolného z nich jiným procesem.
 - **Časově závislá chyba nad data (data race)**
 - Přístup ke zdroji ze dvou procesů bez synchronizace, alespoň jednou pro zápis.
 - **Stárnutí či hladovění**
 - Čekání na podmínu, která nemusí nastat. V případě KS je touto podmínkou vstup do kritické sekce.
 - **Diskový sektor**
 - Nejmenší jednotka, kterou disk umožňuje načíst/zapsat
 - Dříve velikost 512B, Nyní 4096B (možnost emulace na 512B). 2048B u CD/DVD/Blu-ray.
 - **Alokační Blok**
 - Skupina pevného počtu sektorů, typicky 2^n pro nějaké n, následujících logicky i fyzicky za sebou.
 - Nejmenší jednotka diskového prostoru, kterou OS čte či zapisuje při běžných operacích
 - **Extent**
 - Posloupnost proměnného počtu bloků jdoucích za sebou logicky i fyzicky.
 - Zrychluje práci s velkými soubory-menší objem metadat, lepší index. struk.

OPERAČNÍ SYSTÉM, JÁDRO

Zadání

Co je to OS, cíle, role, jádra, mikrojádro

Definice OS

Operační systém je program, resp. kolekce programů, která vytváří spojující mezi výpočetního systému (fyzický nebo virtualizovaný) a uživateli a jejich aplikacemi.

Cíle OS

- Maximální využití zdrojů počítače
 - Dříve, kdy byly drahé počítače a levnější pracovní síla
- Jednoduchost použití počítačů
 - Dnes, kdy je drahá pracovní síla a levné počítače

Role OS

- Správce prostředků
 - Dovoluje sdílet prostředky efektivně a bezpečně
 - Více procesů sdílí procesor, paměť
- Tvůrce prostředí pro uživatele a jejich aplikace
 - Poskytuje standardní rozhraní a poskytuje abstrakce (Windows okénka)

Tyto role jsou dost často kontraproduktivní, protože využíváme výkon na abstrakci a ne na úlohy.

Skládá se z:

- Jádra
- Systémových knihoven a utilit
- Textové nebo grafické rozhraní

Jádro

- Nejnižší a nejzákladnější část OS
- Zavádí se první a běží po celou dobu běhu systému
- Navazuje přímo na hardware a běží v privilegovaném režimu
 - Tento režim může provádět libovolné operace nad HW
- Zajišťuje základní správu prostředků a tvorbu prostředí
-

Typy jáder

- Monolitická jádra
 - Vytváří vysokoúrovňové komplexní rozhraní s řadou služeb a abstrakcí nabízených vyšším vrstvám
 - Všechny subsystémy běží v privilegovaném režimu a jsou těsně provázané
 - Problém s bezpečností
 - **Mikrojádra**
 - Minimalizují rozsah jádra, jednoduché rozhraní,
 - Malý počet služeb
 - většina z nich implementována mimo jádro v tzv. serverech, jež neběží v privilegovaném režimu
 - **Výhody**
 - Bezpečnost - útok na server neovládne celé jádro
 - Flexibilita - spouštění a zastavování serverů
 - **Nevýhody**
 - Vyšší režie - nižší efektivita
 - Příklady mikrojader
 - L4, seL4, Mach, ProvenCore
 - Hybirdní jádra
 - Mikrojádra rozšířená o kód, který by mohl být implementován ve formě serveru, ale je za účelem menší režie těsněji provázán s mikrojádrem
-

CFS - Completely Fair Scheduler

Zadání

Základní myšlenka a princip plánovače CFS implementovaného v Linuxu 2.6.

Princip a základní myšlenka

- Snaží se explicitně každému procesu poskytnout odpovídající procento strojového času dle jejich priorit
 - U každého procesu si vede údaj o tom, kolik procesorového času spotřeboval
 - Vede si údaj o minimálním stráveném procesorovém čase, který dává nově připraveným procesům
 - Procesy udržuje ve vyhledávací stromové struktuře (red-black tree) podle využitého procesorového času
 - Vybírá proces s nejmenším stráveným časem
 - Procesy nechává běžet po dobu časového kvanta spočítaného na základě priorit a pak je zařadí zpět do plánovacího stromu
 - Podporuje skupinové plánování
 - Rozděluje čas spravedlivě pro procesy spuštěn z různých terminálů
-

PLÁNOVÁNÍ PROCESŮ, DISPEČER

Zadání

Dispečer, Plánovač. Nepreemptivní plánování, preemptivní, princip FCFS, round – robin, obecný princip víceúrovňového plánování a víceúrovňové plánování se zpětnou vazbou (základní i běžná varianta+motivace)

Dispečer

Na základě rozhodnutí plánovače přepíná mezi procesem A a B.

Definice plánovače

Plánovač rozhoduje o tom, který proces poběží a případně jak dlouho poběží.
Přepíná kontext -- fyzicky přiděluje a odděluje procesům procesor a paměť
Umožnuje mezi procesorovou komunikaci (IPC)

Preemptivní plánování

Ke změně běžícího procesu může dojít na základě přerušení i bez nápomoci přepnutí kontextu. Přerušení typicky od časovače.

Nepreemptivní plánování

Ke změně běžícího procesu může dojít pouze tehdy, pokud to běžící proces umožní, předáním řízení jádru. Předání řízení tím, že požádá o službu. (typicky I/O operace, konec)

Plánovací algoritmy

FCFS (First Come, First Served)

- Procesy čekají na přidělení procesoru ve FIFO (first in, first out) frontě
- Procesor se přiděluje procesu na začátku fronty
- Nepreemptivní algoritmus - proces se procesoru musí vzdát sám

Round-Robin

- Preemptivní obdoba FCFS
- Každý procesor má uděleno časové kvantum, po kterém mu bude procesor odebrán a jde na konec fronty

Víceúrovňové plánování

- Procesy jsou rozděleny do různých skupin
 - Typicky podle priority nebo typu procesu
- V rámci každé skupiny může být použit jiný dílčí plánovací algoritmus
- Hrozí hladovění procesů s nízkou prioritou

Víceúrovňové plánování se zpětnou vazbou

- Víceúrovňové plánování se skupinami procesů rozdelenými dle priorit
- Proces nově připravený běžet je zařazen do fronty s nejvyšší prioritou, postupně klesá do nižších prioritních front. Nakonec je plánován pomocí Round-Robin v nejnižší úrovni.
- Používají se varianty, kdy je proces zařazen do počáteční fronty na základě své statické priority. Dynamická priorita se následně může snižovat na základě spotřeby procesorového času / zvyšovat, pokud čeká hodně I/O operací.
 - Cílem je zajistit rychlou reakci interaktivních procesů.

Další algoritmy

- SJF (Shortest job first)
 - Přiděluje procesor procesu, který požaduje nejkratší dobu pro své další provádění procesu
 - Minimalizuje průměrnou dobu čekání, Hrozí hladovění
 - SRT (Shortest remaining time)
 - Preemptivní podoba SJF
-

PLÁNOVÁNÍ DISKOVÝCH OPERACÍ

Zadání

Funkce plánovače diskových operací + motivace. Výtahový algoritmus. 2 heuristiky v plánovačích diskových operací a modifikující základní výtahový algoritmus

Funkce plánovače diskových operací

- Shromažďuje požadavky od FS (načtení, zapsání z/do disku)
- Ukládá si požadavky do svých plánovacích front
 - Případně si tyto požadavky přeuspořádává
- Snaží se minimalizovat režii disku

Výtahový algoritmus

- Snaha, aby se hlavička disku plynule pohybovala od středu ke kraji a zpět a vyřizovat požadavky dle pohybu hlavičky

Modifikace výtah. algoritmu

- Circular SCAN
 - Požadavky se vyřizují pouze při jednom směru
 - LOOK/C-LOOK
 - Hlavíčka se nepohybuje od středu ke kraji , ale pouze v tom rozsahu, kde je potřeba provádět operace
-

PROBLEMATIKA SSD

Zadání

Proč je operace přepisu dat u SSD problematická a to i v případě, že neuvažujeme omezení na počtu přepisů, tři způsoby které ssd používá k minimalizaci uvedeného problému

Problém

- Pokud chci přepisovat jednu stránku, je nutné načíst celý blok do paměti, vymazat a v paměti upravený blok načíst zpět
 - Mnohonásobné zpomalení - SSD se několikrát přesouvají po disku
- Zápis do prázdné stránky mohu ale jednotlivě

Minimalizace problému

- SSD může mít více stránek, než je oficiální kapacita
- Příkaz TRIM - sdělí systému, které stránky nejsou používány a následně umožňuje vymazat bloky tvořené takovými stránkami
- Řadič SSD přesouvá dlouho nezměněné stránky - minimalizace počtu přepisů

Zabezpečení disků

EDC (Error detection and correction) -- k určitým datům si vytváří redundantní data pro opravu

S.M.A.R.T. (Self-monitoring analysis and reporting technology) -- vytváří statistiky a rozpoznává vadné bloky (na úrovni OS)

RAID -- 0-6, 10, 0+1

ŽURNÁLOVÁNÍ

Zadání

Co zajišťuje žurnálování s ohledem na jím pokryté operace (jakým přívlastkem lze označit operace pokryté žurnálováním?). Jaké dva pozitivní dopady má tato skutečnost Popište základní princip REDO používané při implementaci žurnálování.

Funkce

- Žurnál slouží pro záznam modifikovaných metadat před jejich zápisem na disk.
- Žurnálování umožňuje spolehlivější a rychlejší návrat dat do konzistentního stavu.
- Operace pokryté žurnálováním jsou **atomické**
- Cyklicky přepisovatelný buffer -> žurnál

Dopady

- Pozitivní dopady
 - Konzistentnost dat
 - Pokryté operace jsou atomické
 - Operace uspěje celá nebo neuspěje vůbec
- Negativní dopady
 - Velká režie
 - Obvykle data kvůli velké režii nejsou žurnálována

REDO

- Implementace na základě dokočení transakcí
 - Sekvence dílčích operací se uloží do žurnálu. Jestli uspějí, transakce se ze žurnálu uvolní. Jestli ne, dokončí se všechny transakce, které jsou v žurnálu zapsány celé.
 - ext3, ext4

Existuje ještě UNDO a REDO+UNDO

Copy on Write

- Alternativa k žurnálování
- Nejprve zapisuje nová data či metadata na disk, pak je zpřístupněno
- Obsah disku je popsán hierarchickou stromovou strukturou
 - Vyhledávací strom popisující uložení dat a metadata na disku, nikoliv adresářový strom
- Nabízí implementaci snímků a klonů

Dalšími alternativami je logování změn (Log-structured) a nebo Soft updates, který sleduje závislosti a uzpůsobuje disk aby byl vždy konzistentní.

Základní princip

- Vytvoří kopii měněného uzlu a upraví ho
 - Vytvoří kopii nadřazeného uzlu a upraví ho tak, aby odkazoval na uzel vytvořený v předchozím kroku
 - Tento krok se opakuje až ke kořenu
 - Pokud dojde ke krachu systému, stačí si načíst všechny kořeny, zkontrolovat kontrolní součty, vybrat si všechny kde sedí kontrolní součet a tyto použít
-

HW Přerušení

- Mechanismus, kterým HW oznamuje jádru asynchronně vznik události, které je potřeba obsloužit. -- IRQ
- Řeší to čip na základní desce
- Tabulku přerušení definuje jádro (Seznam přerušení od procesů...)
- Příjem nebo obsluhu přerušení lze zakázat
 - Na procesoru
 - Čistě programově v jádře
- NMI (non-maskable interrupt)
 - HW přerušení, které nelze maskovat na řadiči ani zakázat jeho příjem na procesoru (typicky chyby HW)
- Přerušení vznikající na procesoru (výjimky)
 - Trap
 - po obsluze se pokračuje další instrukcí
 - Fault
 - po obsluze se opakuje instrukce, která výjimku vyvolala
 - Abort
 - nelze určit jak pokračovat, provádění se ukončí

Spinlock

Nežádoucí účinky

Při dlouhém čekání může blokovat prostředky jiným procesům, které pak nemohou pokračovat. Zatěžuje procesor, protože proces stále běží ve smyčce.

Např. `while ();` -- proces pak cyklí ve smyčce místo toho aby byl pozastaven

INVERZE PRIORITY

Zadání

Problém inverze priorit a jeho 3 řešení

Problém

- Nízko prioritní proces si naalokuje nějaký zdroj, více prioritní procesy ho předbíhají a nemůže dokončit práci s tímto zdrojem.
- Časem tento zdroj mohou potřebovat více prioritní procesy, jsou nutně zablokovány a musí čekat na nízko prioritní proces
- Pokud v systému v tomto okamžiku jsou středně prioritní procesy, které nepotřebují daný zdroj, budou předbíhat nízko prioritní proces
- Tímto způsobem uvedené středně a nízko prioritní procesy získávají efektivně vyšší prioritu

Řešení

- **Priority ceiling**
 - procesy v kritické sekci získávají nejvyšší prioritu
 - **Priority inheritance**
 - proces v kritické sekci, který blokuje výše prioritní procesy, dědí prioritu čekajícího procesu s největší prioritou
 - **Zákaz přerušení po dobu běhu v kritické sekci**
 - proces v podstatě získává nejvyšší prioritu
-

AKCE OS

Zadání

Akce OS při operaci close(). Kroky systému při volání open().

open()

při prvním otevření

1. Vyhodnotí se cesta, zjistí se zda soubor existuje, má potřebné informace a jaké jsou k němu práva. Pokud jsou správné práva, načte se i-uzel.
2. V tabulce v-uzlů se vyhradí nová položka a do ní se načte i-uzel a doplní se dalšími informacemi, mimo jiné počtem odkazů
3. Alokuje se nová položka v tabulce otevřených souborů. Naplní se odkazem na novou položku tabulky v-uzlů a doplní se další údaje, a to:
 - a. pozice v souboru
 - b. počet odkazů
 - c. počet otevření
 - d. režim otevření
4. Alokuje se nová položka v tabulce popisovačů a naplní se odkazem na nově vytvořenou položku v tabulce otevřených souborů
5. Vrátí se index na nově alokovanou položku v tabulce popisovačů
 - a. Případně -1 při chybě

open() již otevřeného souboru

1. Vyhodnotí cestu a získá číslo i-uzlu
2. V tabulce v-uzlů nalezne již načtený i-uzel
3. Zvýší počítadlo odkazů na v-uzel o 1
4. Další beze změny

close()

1. Kontrola platnosti file descriptoru
2. Uvolní se odpovídající položka v tabulce deskriptorů, sníží se počítadlo položky v tabulce otevřených souborů
3. Pokud je počítadlo nulové, uvolní se odpovídající položka v tabulce otevřených souborů a sníží se počítadlo odkazů ve v-uzlu
4. Pokud je počítadlo odkazů nulové, i-uzel se z v-uzlu okopíruje do VP a uvolní
5. Funkce vrací nulu
 - a. -1 při chybě

Start systému

1. Firmware PC (UEFI/BIOS)
 2. Načtení a spuštění zavaděče OS
 3. Načtení inicializační funkce jádra
 4. Vytvoření jádra O a procesu init
 5. Proces init pokračuje v inicializaci systému
-

NTFS

Zadání

Charakterizujte jak je popisováno rozložení dat na disku v systémech Windows používající NTFS a to včetně odlišností reprezentace různě dlouhých souborů. Definujte pojem extent.

Extent

- Posloupnost proměnného počtu alokačních bloků logicky jdoucích za sebou v souboru a uložených fyzicky na disku

Rozložení dat

MFT (Master File Table)

- Základní datovou strukturou popisující disk je MFT
 - Pro každý soubor má alespoň 1 řádek
 - Na řádku 0 popisuje samo sebe, na řádku 1 je kopie MFT, případně metadata, poté obsahy souborů
 - Obsah souborů může být reprezentován
 - Řádkem, pokud jde o krátký soubor
 - Soubor je rozdělen na extentu, v řádku jsou informace jako číslo logického a fyzického bloku, vyhledávání jako v B+ stromu
 - Pokud je extentů více, než se vleze na jeden řádek, alokují se pomocné řádky
-

PŘEKLAD LA NA FA

Zadání

S maximální přesností popište algoritmus převodu logické adresy na adresu fyzickou v systémech s dvouúrovňovou hierarchickou tabulkou stránek a TLB. Algoritmus převodu popište ve formě posloupnosti číslovaných kroků. Můžete se opřít o schematický obrázek, ale důležitý je zejména k němu uvedený popis algoritmu převodu (obrázek bez popisu je za Ob). U každého dílčího přístupu do paměti uvedte (a) s jakou pamětí se pracuje a (b) jakým způsobem se daná paměť adresuje. Uveďte, jak se detekuje, že dojde k výpadku stránky, ovšem samotná obsluha výpadku již není předmětem otázky. Kontroly práv pro čtení/zápis ignorujte.

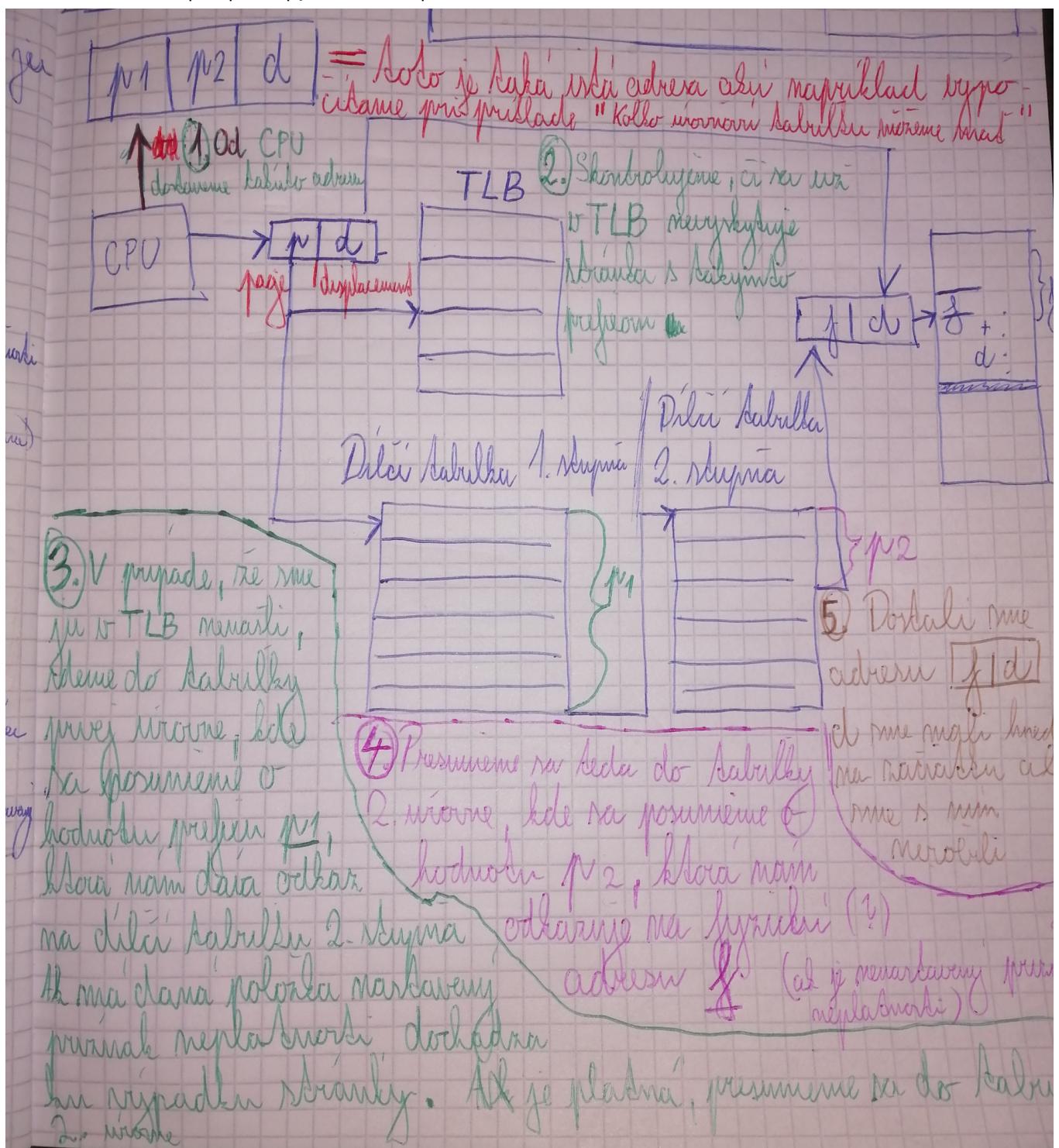
Upozornění: Výroky typu "v paměti se najde" či "paměť se zaindexuje" jsou bez dalšího vysvětlení zcela nedostatečné! Kdykoliv se pracuje s nějakou tabulkou, uveďte, v které paměti se nachází (disk, RAM, či jiný typ paměti), jak

Algoritmus

1. LA se rozloží na prefix **P** sestávající z odkazu p1 do adresáře stránek a z odkazu p2 do konkrétní tabulky stránek. Tedy P=p1.p2
2. Test, zda TLB obsahuje dvojici (p,f), vyhledává se asociativně dle p
 - a. Je li položka (p,f) nalezena, je výsledná adresa (f,d)
3. Přístup k položce na indexu p1 v adresáři stránek, který je v RAM na adrese uložené ve speciálním registru MMU
4. Má-li výše uvedená položka nastaven příznak neplatnosti, nastává výpadek v adresáři stránek
 - a. Není tomu tak, pak se z uvedené položky získá bázová adresa dílčí tabulky stránek rovněž uložené v RAM. Použije se položka s indexem p2 této tabulky.
5. Obsahuje-li položka na indexu p2 v dílčí tabulce stránek příznak neplatnosti, nastává výpadek stránky. Jinak se z položky zmíněné výše získá číslo rámce f a výsledná adresa je (f,d)

Obrázek provided by Lyred

Check komentáře pro postupy v textové podobě



Poznámky z přednášek

Pravděpodobně nad rámec testu ale pořád důležité

Definice pojmu

- Process -- činnost řízená programem (běžící aktivita)
- Program -- je návod pro nějakou činnost (process), pasivní
- Soubor -- kolekce záznamů, základní jednotka pro ukládání dat (abstrakce)
- Adresář -- kolekce souborů
- Multitasking -- současný běh více aplikací najednou (rychlé přepínání procesů plánovačem tak, že to vypadá že běží současně)
- IPC -- roury, signály, semafory, sdílené paměti

Pevný disk

Diskový sektor: nejmenší jednotka, kterou disk může zapsat

Adresace sektorů:

- Cylinder, Head (1-6 hlav), Sektor
- LBA: Linear Block Address (0... N)

Fragmentace

Jev vznikající v pamětech postupným obsazováním a uvolňováním paměti, kdy v paměti vznikají oblasti, které jsou volné ale nemohou být použity.

Jde o příliš malé kousky souborů, které disk nedokáže zpracovat napří pod 512B.

Externí

Přidělování a uvolňování dat na disku má za následek rozkouskované soubory různě po disku s volnými oblastmi mezi nimi

Dopady

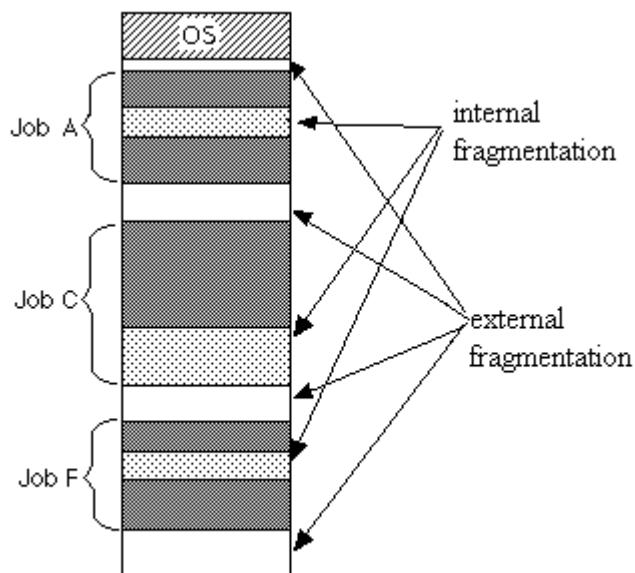
1. Nevyužité oblasti se nedají použít v daném okamžiku nebo vůbec (spojitý zápis)
2. Malá nevyužitá část disku (pod 512B) nejde využít vůbec
3. Fyzické zpomalení kde disk musí procházet celý disk aby dal dohromady 1 soubor

Řešení

1. Rozložení souborů na disku s rozestupy pro rozširování
2. Alokace většího místa než je potřeba (většinou uživatelem, napří steam při instalaci her)
3. Odložení alokace -- bude čekat nějakou dobu a pak vše zapíše na disk

Interní

Paměťový blok je větší než data která do něj budou nahrána. Část bloku je nevyužitá.

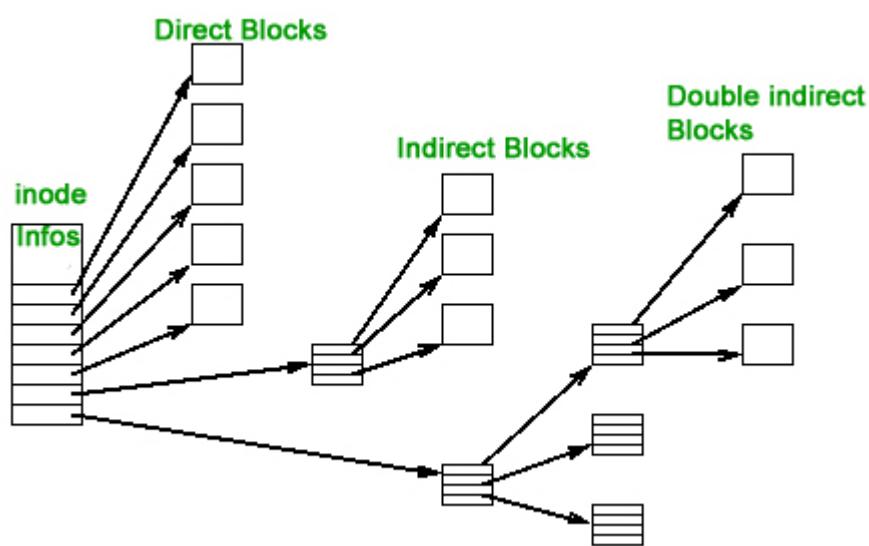


I-Uzel

Je to základní datová struktura které obsahuje informace o jednom souboru.

Obsahuje: typ souboru, stav, mtime, ctime, atime, UID, GID, práva a odkazy na data

V ext2/3 se používají direct (10 blocků), indirect bloky prvního, druhého a třetího stupně na odkazy dat.



B+ stromy

- Databázové systémy
- Často se kombinují s extenty
- 2 typy uzlů -- vnitřní a listové
- Musí být vyvážené
- Vkládají se nové uzly pouze na listové úrovni
- Neplést z extenty v ext4, podobná struktura, ale trochu jiné limity
- Minimální vytíženosť (zaokrouhuje se nahoru), n = počet odkazů uzlu
 - Vnitřních uzlů = $[n/2]$
 - Listů = $[n/2]-1$