6. Požadavky OS na HW nutný pro jeho implementaci: zejména na procesor, správu a adresování paměti. Registry CPU.

1. Nutné vlastnosti CPU a paměti pro implementaci OS a jejich účel.2 + 3

- Pro efektivní využití CPU je nutný přerušovací systém – nutné pro DMA
- Časovač, který pravidelně generuje přerušení – umožní preempci
- CPU s podporou alespoň režimů user a kernel – ochrana paměti
- CPU s podporou virtualizace paměti
  - Umožnění relokace
    - Tedy zavedení procesu na libovolnou adresu
    - Logické adresování – MMU (Memory Management Unit)
      - Vytvoření virtuální adresy k fyzické adrese
- Přítomnost subsystémů

2. Nutné subsystémy HW pro implementaci OS (kromě CPU a paměti) a jejich účel.2 + 2

- Asi soubory a adresářů na paměťových médiích
- Správa V/V zařízení
- Správa procesů ??
-

3. Definice registrů CPU, jejich druhy.1 + 2

● program counter – obsahuje adresu aktuálně prováděné instrukce

● instruction register -obsahuje aktuálně vykonávanou instrukci. Obsahuje binární kód instrukce, který určuje, jakou operaci provést

● stack pointer - ukazuje na vrchol zásobníku (stack) v paměti

● PSW (program status word) – příznaky C, N, Z, V

● ostatní registry (obecné, datové, adresní, privátní, …)

4. Stavový registr CPU a zahrnuté příznaky (alespoň tři).3

- uchovává informace o stavu a prováděných operacích CPU. Zahrnuje několik příznaků, které poskytují informace o výsledku předchozích operací a o stavu procesoru
- PSW (program status word) – příznaky C, N, Z, V
- – C = carry, N = negative, Z = zero, V = overflow

Okruhy otázek z předmětu Operační systémy

1. Definice OS: typy OS; design OS: abstrakce a operace nad nimi – služby, systémová volání.

a) Základní definice (s vysvětlením).2

- OS je v informatice základní programové vybavení počítače, které je zavedeno do paměti počítače při jeho startu a zůstává v činnosti až do jeho vypnutí
- Rozšíření stroje – virtualizace, abstrakce
  - Má za úkol zjednodušit rozhraní, které má programátor k dispozici
  - Tvoří prostředí pro uživatele a programy
  - Příklad: čtení/zápis na disk
- Správce prostředků
  - Procesor, paměť, V/V zařízení (periferie)
  - Příklad: tisk na tiskárnu
  - multiplexing

b) Vlastnosti moderního OS.4

- Preemptivní plánování procesů (a efektivní) – procesy dostávají přidělen procesorový čas, tak jak určí OS
- Izolace procesů navzájem a mezi procesem a jádrem
  - Když proces zapisuje kam nemá, bude shozen jen on a ne celý OS nebo jiný proces
- Efektivní správa paměti
- Pokud je více uživatelů, tak Izolace uživatelů
  - Na úrovni procesů i na úrovni úložného prostoru
- Podpora IPC (Inter-Process communication – meziprocesová komunikace) – komunikace a synchronizace

c) Typy OS (alespoň šest).3

- Mainframe – sálové počítače
- Serverové OS – nabízejí síťové služby
- Osobní OS – PC
- Real-time OS – důležité je dodržení času
- Vestavěné OS – Zabudované OS v zařízení (PDA, Smartphone)
- Smart card OS, Sim card OS
  - Například: platební karty
- Distribuované OS (vícepočítačové) – clustery, paralelní počítače

d) Definice koncepcí (abstrakcí) a systémového volání.2

- Koncepce (abstrakce)
  - Obvykle je reprezentováno datovými strukturami a operacemi nad nimi
  - OS využívá různých koncepcí
    - Například: procesy, vlákna, správa operační paměti, správa vstupů a výstupů, správa úložišť, systémová volání
- Systémová volání
  - Nástroj využívaný aplikacemi pro volání systémových funkcí/služeb (v případě monolitických jader – Unix)
  - Volání služeb jádra OS

e) Popište průběh systémového volání.3

- Na Stack se uloží parametry
- Následně se zavolá funkce v knihovně, která odpovídá danému systémovému volání
- Ta zařídí skok do OS ale ještě předtím do registru nastaví typ volání
- Dále skočí do jádra OS na předem definovanou adresu pomocí instrukce TRAP
- Jádro provede dispatch, zavolá příslušný ovladač
- A navrátí se do knihovny a programu

f) Jmenujte příklady systémových volání a co obstarávají (alespoň tři).1

- Procesy – vznik či ukončení procesu

- Soubory – čtení či zápis
- Adresáře a souborové systémy – vytvoření čí zrušení
- Ostatní (práva, signály, …) – změna práv

2. Architektura jádra OS: monolitický systém, vrstvený systém, virtualizace na úrovni jádra OS, mikrojádro.

a) Monolitické jádro. (systém)2

- Vše je v jednom – jádro není vnitřně členěné
- Každá procedura může volat libovolnou jinou
- Procedury mají pevně definované rozhraní
- Jsou velké, nerozšiřitelné
- Výhodou je centralizace
- Může mít i strukturu
  - Hlavní program
  - Obslužné procedury
  - Užitkové procedury
- Příklad: BSD, Linux, Windows

b) Vrstvené jádro a čím se liší od monolitického s vnitřní strukturou vrstev.2

- Vrstva může volat procedury jen stejné nebo nejbližší nižší vrstvy – zajištěno HW
- Každá vrstva má jiný úkol
- 5 – operátor; 4 – uživatelské programy; 3 – správa V/V zařízení, buffering; 2 – komunikace mezi procesy a konzolí operátora; 1 – správa paměti; 0 – alokace CPU a multiprogramming
- Příklad: MULTICS

c) Mikrojádro: vlastnosti, funkce.2 + 3

- Obsahuje pouze základní funkce (důležité) – komunikace procesů, správa paměti, plánovač
  - Vše ostatní v knihovnách
- Spolehlivost, bezpečnost, rozšiřitelnost
- Méně výkonné
- Jen nejnutnější funkce
  - Mikroprogramování – alokace CPU
  - Zajištění ochrany paměti
  - Základní IPC (meziprocesová komunikace)
- Příklady: QNX, MINIX 3

d) Virtualizace na úrovni jádra.3

- Jádro, které je navrženo tak, že je schopné obsluhovat více skupin procesů, které jsou navzájem izolované a které tvoří samostatné celky a samostatně přístupné OS
- Příklady: Solaris 11, Free BSD Jails, Linux-Vserver
- Možno virtualizovat jen stejné OS najednou

e) Příklady OS různých architektur jádra.3

- Příklady výše

3. Návrh OS a jeho bezpečnost: důvody náročnosti implementace OS, principy jeho vývoje; zabezpečení systému, uživatelských dat a procesů (vyjma útoků na systém, to je jiný okruh).

a) Cíle návrhu OS.5

- Hlavní společné cíle – definovat abstrakce, spravovat HW, poskytnout základní operace, zajistit izolaci
- Uživatelský pohled – snadno použitelný a naučitelný, spolehlivý, rychlý a bezpečný
- Systémové hledisko – snadná implementace, bezchybnost, údržba

b) Důvody náročnosti.4

- Jedná se o komplexní, rozsáhlou, složitou aplikaci
- Konkurence procesů v přístupu k systémovým prostředkům
- Ochrana dat, sdílení dat a systémových prostředků
- Obecnost použití, přenositelnost, zpětná kompatibilita

c) Principy vývoje OS.3

- Jednoduchost

- Jasné chování, jedna funkce by měla dělat jednu věc
- Pokud je vlastnost potřeba jen „možná“, je lepší to dát do knihovny, ne do jádra

d) Zabezpečení systému (procesů, dat na médiu, přenášených dat).3

- Integrita dat
- Řízený přístup k systémovým prostředkům
- Autentizace, autorizace, zamezení přístupu a šifrování

---

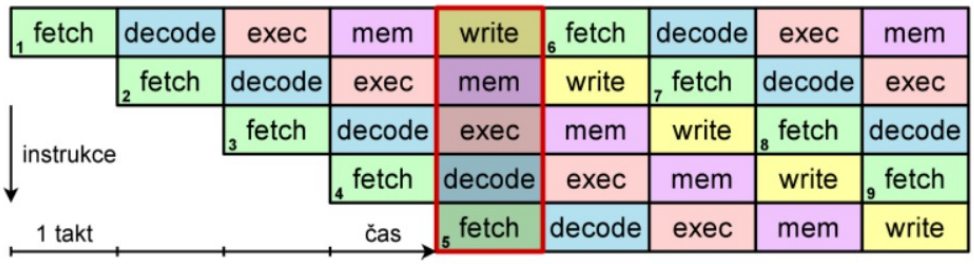
4. CPU: provádění instrukce, pipeline, atomicita a přerušitelnost procesu a průběhu zpracování instrukce, přerušovací systém, průběh zpracování přerušení, časovač, sdílení času.

a) Fáze provádění instrukce.4

- Fetch – načtení z paměti do registru
- Decode – dekódování (zjištění, jaká je to operace)
- Execute – provedení

b) Pipeline, zvyšování výkonu CPU.4

- Pipeline – vykonávání více instrukcí najednou



- Zvyšování výkonu
  - Superskalární CPU
    - Snaha určité paralelizace instrukcí
  - Spekulativní provádění instrukcí
    - Provádění instrukcí v pořadí, jak si CPU rozhodne
  - Hyperthreading
    - Sdílení subčástí CPU mezi vlákny
  - Více jader

c) Přerušitelnost procesu, průběhu zpracování instrukce.2

- Atomicita – činnost se provede najednou, nemůže být přerušena
- Preemptivní – přerušení
- Nepreemtivní

d) Přerušovací systém: účel, průběh přerušení.5

- Účel:
  - Používá se k ošetření výjimek a chyb procesů
    - Když chce proces přistoupit do zakázaní oblasti nebo se jedná o neplatnou instrukci, vykoná se přerušení a provede se osetření.
  - Chyby v HW:
    - Výpadek napájení
- Časovač
  - Generuje pravidelné přerušení
- Sdílení času
  - Běžící procesy se dělí o strojový čas CPU, dochází k přepínání mezi nimi bez zásahů a ovlivnění uživatele
- Průběh:
  - Pokud CPU vykonává instrukci a HW vygeneruje IRQ (přerušení)
  - CPU dokončí instrukci, pak uloží návratovou hodnotu na stack a skočí na obslužnou rutinu přerušení.
  - Vykoná se rutina
  - Plánovač rozhodne, který proces poběží
  - Návrat do procesu

---

5. Vstupně-výstupní zařízení: ovladače a techniky programování vstupu a výstupu, DMA. Paměť cache, procesorová cache a střední přístupová doba do paměti.

a) Metoda 1 komunikace se vstupně-výstupním zařízením.2
- Přístup (komunikace) pomocí instrukcí CPU
  - Přímo na port V/V zařízení zapíšeme/čteme data pomocí instrukcí CPU
  - V/V zařízení je mnohem pomalejší než CPU.
  - Neefektivní

b) Metoda 2 komunikace se vstupně-výstupním zařízením.2
- Přístup (komunikace) s využitím přerušení
  - Zapíše se operace na port, ale nečeká se na výsledek. CPU pokračuje v jiné práci.
  - Jakmile se operace dokončí, vyvolá se přerušení a pokračuje se v práci.
  - Lepší, ale neefektivní

c) Metoda 3 komunikace se vstupně-výstupním zařízením.3
- Použití DMA (Direct Memory Access) a IRQ
  - Povolení zařízení, aby si samo četlo data z operační paměti nezávisle na CPU. K tomu se využívá DMA kanál.
  - Po dokončení se vyvolá přerušení

d) Definice cache.2
- Obecně je to rychlá paměť mezi rychlejším a pomalejším zařízením (například mezi RAM a CPU)

e) Důvod existence konceptu paměti cache.1
- Účelem je urychlení práce s často používanými daty

f) Důvod efektivity používání cache.2
- Cache používá princip lokality odkazů v paměti
  - Pokud přistupujeme na nějaké místo v paměti, tak ze všech paměťových míst, které bychom mohli příště adresovat (potřebovat), mají největší pravděpodobnost ta místa, která jsou v okolí.

g) Výpočet střední přístupové doby do paměti.3
- střední přístupová doba: $T_s = T_c + (1 - HR) \cdot T_{OP}$ ($T_c << T_{OP}$)
  - HR blízko 1 → přístup je blízký přístupu do cache
- T_c je doba přístupu do paměti cache, T_{op} je doba přístupu do operační paměti (např. RAM)
- Pokud se přistoupilo jen do paměti Cache, tak $T_s = T_c$ a opačně, pokud se přistupuje jen do T_{op}, pak $T_s = T_{op}$
- HR je Hit ratio

6. Požadavky OS na HW nutné pro jeho implementaci: zejména na procesor, správu a adresování paměti. Požadavky na moderní OS.

a) Nutné vlastnosti komponent – CPU, paměti, přítomnost subsystémů, jejich účel.4× 2
- Pro efektivní využití CPU je nutný přerušovací systém
- Časovač, který pravidelně generuje přerušení
- CPU s podporou alespoň režimů user a kernel
- CPU s podporou virtualizace paměti
  - Umožnění relokace
    - Tedy zavedení procesu na libovolnou adresu
    - Logické adresování – MMU (Memory Management Unit)
      - Vytvoření virtuální adresy k fyzické adrese
- Přítomnost subsystémů

b) Navazující vlastnosti moderních OS: procesy, paměť, další vlastnosti.2 + 2 + 3
- Preemptivní plánování procesů (a efektivní) – procesy dostávají přidělen procesorový čas, tak jak určí OS
- Izolace procesů navzájem a mezi procesem a jádrem
  - Když proces zapisuje kam nemá, bude shozen jen on a ne celý OS nebo jiný proces

- Efektivní správa paměti
- Pokud je více uživatelů, tak Izolace uživatelů
  - Na úrovni procesů i na úrovni úložného prostoru
- Podpora IPC (Inter-Process communication – meziprocesová komunikace) – komunikace a synchronizace

7. Implementace procesu v OS: proces a program, tabulka procesů, přepínání kontextu, stav procesu, třístavový model, příčiny změn stavů.

a) Proces a program: definice.2

- Proces: vykonávání sekvence příkazů (programu)
- Program: sekvence příkazů napsaných programátorem

b) Metadata procesu (alespoň šest).3

- Tabulka procesů (PCB)
  - Identifikace procesu (PID), adresový prostor, stav, přidělené prostředky, práva, čas běhu

c) Průběh přepnutí kontextu.3

- Při plánování procesů dochází k tzv. přepínání kontextu (**context switch**). Procesu, kterému se odebírá CPU, je třeba uložit rozpracovaný stav výpočtu (kontext), aby později mohl korektně pokračovat v běhu. Tento kontext (stav) se ukládá obvykle do PCB. Proces (kód) je na procesoru vykonáván, dokud proces nezavolá službu systému nebo nedojde k (hardwarovému) přerušovacímu signálu, zde časovače. Signál přerušení je zpracován tak, že se skočí na obslužnou rutinu (do jádra OS), což je pro přerušení časovače plánovač. Plánovač uloží kontext procesu (stav registrů) do PCB, následně vybere dle plánovacího algoritmu další připravený proces, který má běžet. Stav tohoto procesu nahraje do procesoru a přenechá mu procesor.

d) Třístavový model: popis stavů.3

- Stavy:
  - Běžící (Running)
    - Proces, který je vykonáván na procesoru (CPU)
  - Připravený (Ready)
    - Je proces, který může běžet, ale čeká na přidělení prostředků od plánovače
  - Blokovaný (Blocking)
    - Proces, který nemůže běžet. Čeká na událost, třeba na data od uživatele.

e) Třístavový model: příčiny přechodů mezi stavy.4

- Blokovaný -> připravený
  - pokud nastala událost
- připravený -> běžící
  - plánovač přidělil procesu CPU
- běžící -> připravený
  - plánovač odebral procesu CPU
- běžící -> blokovaný
  - blokující systémové volání

8. Procesy v OS UNIX/Linux: vznik a zánik procesu, systémová volání fork(2), exec(3), exit(3), wait(2), kill(2); hierarchie procesů, stavy procesů v Linuxu (podle příkazu ps); posixové signály a jejich zpracování.

a) Příčiny vzniku a zániku procesu, související systémová volání, hierarchie.7

- Vznik:
  - Původcem je jádro
    - Při inicializaci systému
  - Původcem je jiný proces
    - Systémové volání
- Zánik:
  - Normální
  - Při detekci chyby – například chybný vstup
  - Fatální chyba – nepovolená instrukce, dělení nulou
    - Proces je ukončen jádrem

        - Ukončení jiným procesem – uživatelem
    - Systémová volání
        - Fork(2) – vytvoření nového procesu
        - Exec(3) – nahrazení kódu běžícího procesu
        - Exit(3) – ukončení procesu nebo vlákna
        - Wait(2) – čekání na změnu potomka (na ukončení, pozastavení nebo třeba obnovení běhu)
        - Kill(2) – poslání signálu/zabití procesu
    - Hierarchie
        - Každý proces má svého rodiče s číslem, které proces eviduje (PPID), tím vzniká strom, který lze zobrazit příkazem pstree

b) Stavy procesů v Linuxu.4

- R (Running/Runnable) – běžící/připravený
- S (Sleep) - blokovaný
- Z (Zombie, defunct) – ukončený
- T (Traced, stopped) – pozastavený, krokovaný
- D (uninterruptible sleep) – nepřerušitelný spánek

c) Posixové signály, možnosti jejich zpracování a příklady.4

- Posixové signály
    - Jednoduché zprávy v SW
    - Může je posílat OS nebo se posílají třeba příkazem kill
- Zpracování
    - Ukončení (SIGTERM)
    - Pozastavení (SIGSTOP)
    - Pokračování procesu (třeba pokud byl proces pozastaven)(SIGCONT)
    - Ignorování (SIGCHLD)
- Příklady:
    - Signál 15 – SIGTERM (ukončení) – core
    - Signál 9 – SIGKILL (ukončení, které nelze ignorovat)
    - Signál 14 – SIGALRM (alarm)

---

9. Vlákna: motivace zavedení vláken, proces × vlákno, možné implementace vláken, obecné problémy při implementaci a používání vláken; knihovna posixových vláken: mutex, bariéra, podmínková proměnná a jejich použití.

---

a) Motivace zavedení vláken.1

- Jednodušší a rychlejší správa než u procesů (vytvoření, ukončení,…)
- Kvaziparalelismus
- Zvýšení výkonu

b) Společné a samostatné položky metadat procesů/vláken, důvody.3 + 2

- Vlákna sdílí paměťové prostředky a struktury, adresní prostor
- Samostatně má každé vlákno stack (tam se umisťuje návratová hodnota podprogramu, lokální proměnné a parametry), plánovací položky: stav (připraveno/blokováno/běží), kontext (uložené registry CPU), priorita,…

c) Implementace s podporou OS a bez ní, výhody a nevýhody.4

- S podporou OS
    - Vzdálená volání – více režie (trvá déle)
    - Výhody:
        - Není potřeba neblokující volání
        - Page-fault způsobí zastavení jen vlákna, které to způsobilo
    - Nevýhody:
        - Přístup do jádra – náročnější režie (trvá déle)
        - Pevná strategie plánovače vláken
- Bez podpory OS
    - Pomocí knihovních funkcí – náročné
    - Problémy:

        - Blokující volání je třeba převést na neblokující
        - Je nutné vytvořit plánovač vláken – pracné
        - Page-fault – stránka není v operační paměti
    - Výhody:
        - Lepší režie – rychlejší vznik, přepnutí kontextu, …
        - Bez přechodu do jádra
        - Vlastní plánovač, dá se přizpůsobit aplikaci
    - Nevýhody:
        - Náročné na implementaci
        - Page-fault zastaví všechna vlákna

d) Komplikace při používání vláken.2

- Globální proměnné
    - Samostatné alokace pro každé vlákno
- Nereentrantní volání některých knihovních funkcí
    - Některé fce, pokud je opustíme, už se k nim nelze vrátit (malloc())
- Přístup ke sdíleným proměnným
    - Vznik kritických sekcí
- znalost implementace signálů a jejich obsluhy
    - které vlákno má dostat signál, které se má přerušit
- stack
    - problém při přetečení

e) Posixová knihovna vláken: vyjmenujte nástroje pro řešení problémů souběhu a uveďte jejich účel.3

- Posixová knihovná vláken
    - pthread.h
- Vytváření vláken
    - Pthread_create(pthread_t *restrict thread, const pthread_attr_t *restrict attr, void *(*start_routine) (void*) void *restrict arg)
    - Thread – identifikace vlákna
    - Attr – nastavení atributů
    - Start_routine(attr) – vykonávaná funkce
- Ukončení vláken
    - Pthread_exit(void *value_ptr)
- Čekání na ukončení vlákna
    - Pthread_join(pthread_t thread, void **value_ptr)

---

10. Plánovač. Cíle plánování, režimy plánování, plánovací kriteria (cíle) pro plánovací algoritmy, plánovací algoritmy.

---

a) Úloha plánovače, režimy plánování procesů.3

- Plánovač rozhoduje, který proces, či vlákno dostane CPU
- Řídí se nějakým plánovacím algoritmem
- Režimy:
    - Preemptivní
        - Plánovač rozhoduje, který proces dostane CPU
        - Je to efektivní, pokud je k dispozici časovač a přerušení
        - Časovač tiká a aktivuje periodicky přerušovací signál
    - nepreemptivní
        - procesy si sami určují, kdy se vzdají CPU

b) Cíle plánování (obecně a dle určení OS).3

- Cíle obecně:
    - Spravedlivost – každý proces by měl dostat stejně času
    - Dodržování strategie – priorit
    - Efektivní využití zdrojů – využívání všech součástí systému současně
- Interaktivní systémy
    - Zde je nějaký uživatel

        - Minimalizace odezvy
            - Doba mezi zadáním příkazu a odezvou systému
        - Proporcionalita
            - Vyhovět očekávání uživatele
            - Pokud by to mělo chvíli trvat, dát uživateli odezvu
            - Uživatel klikne, čeká, že se něco stane. Nic se neděje, klikne znovu a najednou se to spustí několikrát
- Dávkové systémy
    - Maximalizovat propustnost
    - Minimalizovat obrat
    - Maximalizovat využití CPU
- Real-time systémy
    - Respektování (dodržování) lhůt – zabránění ztráty dat
        - Třeba při zpracování signálu
        - Není čas čekat, musí se zpracovat hned, jinak se data přepíší jinými daty
    - Předvídatelnost – zabránění ztrátě dat (například CD/DVD přehrávač si načítá data dopředu)

c) Popište plánovací algoritmy: historické.3 (stačí vyjmenovat)

- Dávkové systémy
- FIFO (First in First out)
    - Proces, který přišel první je také první vykonán
    - Úlohy se řadí do fronty
    - Nepreemptivní
    - Proces, který je ve frontě nejdéle, dostane CPU
- Podle odhadu doby běhu
    - SJF (Shortest Job First)
        - Nejkratší úloha jde první
        - Vždy se spustí proces, který má nejkratší dobu běhu
        - Hrozí vyhladovění procesu – je možné, že se proces s dlouhou dobou běhu nedostane k CPU
    - Shortest Remaining Time First
        - Preemptivní obdoba SJF
        - CPU dostane proces, který má nejkratší odhadovanou dobu do dokončení

d) Popište plánovací algoritmy: moderní a specifické.6

- Round-robin
    - Cyklická obsluha
    - Každý proces dostane stejný čas daný časovačem. Po uplynutí této doby jde další proces. A
    - Máme kruhovou frontu a procesu přidělujeme stejný čas podle toho kde je ve frontě
    - Postupně se střídají všechny procesy
    - Přepnutí je vykonáno při vypršení času, nebo při volání blokujícího přerušení
    - Musí se optimalizovat délka kvanta (času daného časovačem)
- Prioritní
    - K CPU se dostanou procesy, které mají nejvyšší prioritu
    - Kvůli nízké prioritě muže proces vyhladovět
        - Lze tomu zabránit, když se přizpůsobí priorita na době čekání a historii běhu procesu
- Uživatelsky férové (FS – Fair-Share)
    - Každý uživatel dostane stejný čas k CPU
    - Pokud je *n* uživatelů, každý dostane 1/n času CPU
- Loteriové
    - Každý proces dostane tiket, losuje se, který proces dostane CPU
    - Procesy mají možnost mít více tiketů
    - Kooperativní procesy si mohou měnit tikety
- Termínové (real-time)
    - Důležité dokončení v daném termínu
    - Periodické události se mohou plánovat staticky

- Aperiodické procesy (události) se plánují dynamicky

11. Požadavky na plánování v systémech reálného času. Možnosti plánování vláken na víceprocesorových systémech (SMP). Časové a periodické plánování úloh uživatelem – příkazy at a crontab.

a) Kritéria pro plánování v systémech reálného času, plánovatelnost.4

- Některé procesy musejí být stále v operační paměti
  - Kvůli potřebné krátké odezvě
- Práva a priority procesu se určují podle účelu
  - Procesy důležité pro správné chování a bezpečnost systému musejí mít přednost
- Minimalizace intervalů se zákazem přerušení
- Systém musí být plánovatelný

b) Možné optimalizace plánování vláken na systémech SMP.4× 2

- Sdílení zátěže
  - Procesy nejsou přiřazeny napevno k procesoru
  - Zátěž procesů se rozděluje rovnoměrně
  - Žádný procesor není nevyužitý
  - Není potřeba centralizovaný plánovač
- Skupinové plánování
  - Máme spolu související vlákna. Ta mohou být plánována tak, že poběží na různých procesorech současně, vzájemná synchronizace
- Pevné přiřazení procesoru
  - Procesům se napevno přiřadí procesor
  - Procesory mohou zůstat nevyužity
- Dynamické plánování
  - Během provádění se může měnit počet vláken
  - OS upravuje zátěž na procesory – snaha zlepšit využití systému

c) Příkazy pro nastavení spouštění úloh v daném čase a démony, které to obstarávají. Orientačně možnosti konfigurace.3

- Příkaz at
  - Jednou provede v určitý čas nějaký příkaz
  - Používá se pro plánování (odložení) spuštění programů
- Příkaz crontab
  - Provádí periodicky nějaký příkaz
  - Na více uživatelském systému muže mít každý uživatel vlastní crontab

12. Požadavky na paměť, alokace, adresování, pevné (statické) a proměnné (dynamické) dělení paměti (fixed partitioning, variable partitioning), fragmentace paměti, typy fragmentace, umisťovací algoritmy.

a) Požadavky na paměť (na její vlastnosti), předpoklady.3

- Relokace paměti
  - Paměť by měla umožnit přemístění procesu na jinou adresu
  - Musí se pracovat v logickou adresou, které se za běhu převedou na skutečné fyzické adresy
- Ochrana paměti
  - Procesy nesmějí přímo vstoupit do paměti jádra OS a do jiných procesů
  - Toto se musí řešit v reálném čase
  - Ochrana má různé stupně (režimy)
    - Čtení, zápis, provádění (execute)
- Možnost sdílení paměti
  - Nejrychlejší způsob výměny dat mezi procesy
    - Více procesů může přistupovat do jedné paměti
  - Snižuje využití paměti
- Logická a fyzická organizace paměti
  -
  - Fyzická paměť nemusí stačit
    - Pokud proces požaduje více paměti, než je dostupné, použijeme překrývání (overlaying) nebo swapping

- Překrývání
    - Moduly programu nejsou potřeba najednou, pak můžou použít stejnou fyzickou oblast paměti
- Swapping
    - Odložení procesu na sekundární paměť (například disk)

b) Pevné dělení paměti, nevýhody, umisťování procesů.4

- Paměť je napevno rozdělena na oblasti s pevnými hranicemi
- Oblasti jsou buď stejně velké, nebo různě velké
- Pokud jsou všechny oblasti plné, je možné proces odložit na disk
- Pokud je proces moc velký a nevejde se do oblasti, použije se překrývání
- Vzniká vnitřní fragmentace
    - Procesu je přidělena paměť, kterou nevyužije
    - Z venku je paměť alokována (je využita), ale z pohledu procesu je paměť nevyužita
- Stejná velikost oblasti – proces se uloží do libovolné oblasti
- Různá velikost – buď má každá velikost oblasti svoji frontu, pak proces čeká na konkrétní velikost, nebo je jedna fronta pro všechny oblasti a procesy se přidělují pomocí algoritmu best-fit

c) Dynamické dělení paměti, nevýhody.4

- proměnlivý počet oblastí i jejich velikost
- Paměť je alokována podle potřeb procesů
- Nevzniká vnitřní fragmentace
- Ale po ukončení procesu vznikne mezera (vnější fragmentace)
    - Ta se dá odstranit defragmentací (přemístění pamětí procesů tak, aby vznikla souvislá oblast)

d) Umisťovací algoritmy.4

- Best-fit – nejlépe padnoucí
    - Vybere se stejná nebo nejmenší možná volná oblast
    - Nejméně výkonná, protože se musejí projít všechny oblasti
    - Má nejmenší fragmentaci
- First-fit – první padnoucí
    - Vybere se první volná oblast, do které se proces vejde
    - Prochází se vždy od začátku
- Next-fit – následující padnoucí
    - Vybere se první volná oblast, do které se proces vejde
    - Prochází se od oblasti, do které se naposledy umisťovalo
    - Nejrychlejší metoda
- Worst-fit – nejhorší padnoucí
    - Vybere se stejná oblast, jako je velikost procesu
    - Pokud taková není, vybere se největší volná oblast
    - Dělí velké oblasti paměti – pak se může stát, že nebude místo pro velké procesy
    - Nejhorší využití paměti

---

13. Problém nedostatku operační paměti, odkládání obsahu paměti na disk, virtuální paměť, motivace k zavedení virtuální paměti, výhody virtualizace paměti, princip lokality odkazů, thrashing.

---

a) Možná řešení nedostatku RAM (s vysvětlením principu).2 + 2

- Překrývání (overlaying)
    - Moduly programu nejsou potřeba najednou, pak můžou použít stejnou fyzickou oblast paměti
- Swapping
    - Odložení procesu na sekundární paměť (například disk)

b) Virtualizace paměti, motivace, důsledky, pojmy (RS, S), princip fungování a HW podpora.6

- Virtuální paměť
    - Zahrnuje skutečnou paměť – fyzická operační paměť v počítači (RAM)
    - A virtuální paměť – CPU je schopné využít více paměti, než má nainstalováno
- Motivace
    - Chceme spouštět programy, které potřebují více paměti, než je fyzicky možná

        - Můžeme využit multitasking
    - Pojmy
        - Resident set (RS)
            - Je to část adresového prostoru procesu, která je v RAM
        - Swap (S)
            - Je to sekundární paměť (disk), kam se dá dočasně odložit nepoužívaná část adresového prostoru procesu
    - HW podpora
        - Procesor musí umět pracovat s virtuální adresou
            - MMU převádí virtuální adresu na fyzickou adresu
        - Pokud není adresovaná část v RAM, musí se vygenerovat přerušení a předat řízení OS
            - OS pak vydá příkaz nahrání této časti paměti z disku do RAM

c) Princip lokality odkazů.2

- Proces má tendenci přistupovat k okolí svého adresového prostoru, kam přistupoval dříve (nedávno)
- Tedy, lze odvodit, jakou část paměti bude proces v nejbližší době potřebovat a nahrát jí do RAM

d) Thrashing, příčiny.3

- Vzniká, když se špatně organizuje odkládání částí paměti procesu na disk
    -
- Může to být následek nedodržení principu lokality odkazů
    - Při neoptimalizovaném překladu kompilátorem nebo při neefektivním programování
- Nastává taky, když spouštíme mnoho procesů a máme malou operační paměť

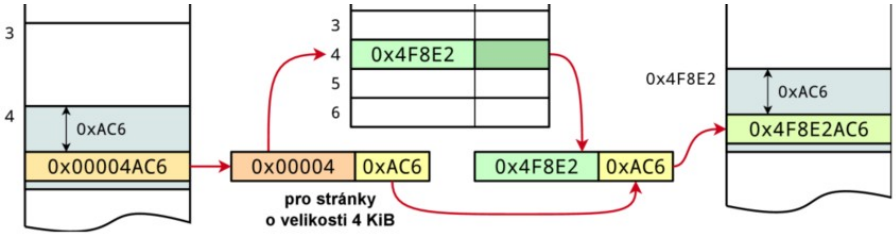
14. Stránkování paměti, převod adresy, vlastnosti stránkování, sdílení stránek, volba velikosti stránky, řešení problému rozsáhlých stránkových tabulek, TLB.

a) Princip, vlastnosti (spojitost, fragmentace, sdílení).5

- Fyzická operační paměť RAM je rozdělena na rámce (malé oblasti)
- Logický adresový prostor procesu je rozdělen na stejně velké části jako RAM – na stránky
- OS udržuje tabulku přiřazení stánek k rámcům
- Pak logická adresa se dá rozdělit na číslo stránky a offset
    - Offset je relativní adresa vzhledem k začátku stránky (zbytek po tom dělení)
    - Číslo stránky je logická adresa/velikost stránky (celočíselné dělení)
- Vlastnosti
    - Spojitost
        - Umístění stránek v RAM je nespojité (vloží se tam, kde je zrovna místo)
    - Fragmentace
        - Stránkování odstraňuje vnější fragmentaci z RAM
        - Vnitřní fragmentace je závislá na volbě velikosti stránky
            - Čím menší stránka, tím menší vnitřní fragmentace
    - Sdílení
        - Pokud mají procesy stejnou stránku, mohou sdílet jeden rámec
        - Není třeba mít dvě stejné stránky ve dvou rámcích
        - Sdílení třeba knihoven

b) Stránkové tabulky, převod adresy.4

- Obsahují čísla stránek a čísla rámců k nim přiřazeným a řídicí bity
- Převod adresy je má konstantní složitost
- Převod adresy
    - MMU převádí virtuální adresu na fyzickou adresu
        - MMU vezme lineární adresu procesu
        - Tu rozdělí na číslo stránky a offset
        - Číslo stránky je index do stránkovací tabulky. Tam najdeme číslo rámce a řídicí bity
        - V adrese vyměníme číslo stránky za číslo rámce, offset necháme
        - Tím dostaneme fyzickou adresu



- Offset jsou poslední 3 cifry, zbytek je číslo stránky (rámce)

c) Volba velikosti stránky a důsledky.3

- Stránky pevné velikosti
  - Vzniká vnitřní fragmentace
  - Menší stránky
    - Menší vnitřní fragmentace
    - Více stránek na proces
      - Větší stránková tabulka zabírá více místa
    - Více rámců v RAM
      - Dá se lépe hospodařit s RAM
  - Větší stránky
    - Větší vnitřní fragmentace
    - Více stránek na proces
      - Menší stránková tabulka zabírá více místa
    - Více rámců v RAM
      - Horší hospodaření s RAM
    - Disk je efektivnější, když přenáší data po větších blocích

d) Řešení rozsáhlosti stránkových tabulek, TLB.3

- Stránkové tabulky se umisťují do virtuální paměti
  - V RAM je jen část stránkové tabulky a zbytek je na disku
  - Nevýhoda: dlouhé čtení, když položka není v RAM
- Proto se používá TLB nebo Víceúrovňové stránkové tabulky
- TLB (Translation Lookaside Buffer)
  - Je to speciální cache pro položky stránkové tabulky
  - Obsahuje několik naposledy použitých položek ze stránkové tabulky
  - Když se hledá položka stránkové tabulky, kouká se nejdříve do TLB
  - Nalezení – okamžitě se adresa převede
  - Nenalezení – položka se přidá do TLB
- Víceúrovňové stránkové tabulky
  - Rozdělení stránkovací tabulky na více menších
  - Virtuální adresa se dělí na offset a na 2 indexy
    - první index určí položku stránkového adresáře, ze které se odvodí adresa stránkové tabulky druhé úrovně
    - druhý index určí položku stránkové tabulky, která obsahuje číslo rámce
  - tím se vytvoří stromová struktura tabulek
  - hlavní tabulka je v RAM
  - Ostatní tabulky jsou ve virtuální paměti
    - Mohou být odloženy na disk
    - Vůbec nemusí existovat (mohou být vytvořeny až při běhu procesu)

15. Řídicí bity ve stránkových tabulkách, strategie zavádění, umisťování, nahrazování a uklízení (čištění) stránek paměti, vliv velikosti resident-set na běh procesů.

a) Důležité řídicí bity a jejich význam.3

- stránková tabulka obsahuje příznaky (řídicí bity)
  - přítomnost stránky v RAM (present – P)
  - stránka je používaná (referenced, accessed – A)
  - modifikovaný obsah (modified, dirty – D)

        - nebyla-li stránka v RAM modifikovaná a je již na disku, není třeba ji znovu zapisovat při uvolňování rámce

b) Strategie zavádění, účel, algoritmy.3
- Určuje, která stránka a kdy se má zavést do RAM
- Demand paging
    - Stránky se zavádějí jen pokud jsou potřeba (pokud nastal page-fault)
- Lookahead paging
    - Stránky se zavádějí napřed
        - Podle principu lokality odkazů se zavádějí stránky z okolí té vyžadované

c) Strategie umisťování, účel, algoritmy.1
- Určuje, kam (do které části RAM) se má stránka zavést (do kterého rámce)
- OS to neřeší – vybírá se náhodně
    - Důvod: doba přístupu je pro všechny adresy v RAM stejná

d) Strategie nahrazování, účel, algoritmy.4
- Určuje, která stránka se odloží na disk, pokud je načíst jinou stránku
- Ideálně – odstranění stránky, která BUDE nejdéle nevyužitá
    - Doba se pouze odhaduje
- Jsou stránky, které se nesmí odstranit z RAM – pro ně se používá zamykání rámců (například: jádro OS, V/V buffery, klíčové řídicí struktury
- Používá se Clock policy
    - Rámce se dají do kruhového seznamu
    - Index nám určuje poslední položku
    - Pokud potřebujeme volný rámec, zvyšujeme index, dokud nenarazíme na stránku s bitem R=0 (referenced)
    - Ta stránka se pak nahradí novou stránkou
    - Při procházení seznamu se bit R nastavuje na 0

e) Strategie uklízení (čištění), účel, algoritmy.3
- Určuje, kdy se mají modifikované stránky uložit na disk
- Demand cleaning
    - Ukládání stránky z rámce při nahrazování jinou stránkou
        - Proces, který čeká po page-fault čeká na přenos dvou stránek
- Precleaning
    - Periodicky se ukládají stránky na disk
        - Mohou se provádět zbytečné zápisy (obsah se ještě může měnit)

f) Volba velikosti resident-set.1
- Pevná alokace
    - Procesu je při nahrávání vyhrazen pevný počet rámců
- Proměnná alokace
    - Počet rámců se může průběžně měnit
        - Třeba, když má proces často page-fault zvýší se mu počet rámců
        - Naopak, když má proces málo page-fault sníží se mu počet rámců
        - To vyžaduje režii OS

---

16. Požadavky na OS pro práci v reálném čase, rozdělení RT OS, pojmy latence a odezva (na úrovni přerušení); vestavěné systémy, OS pro ně a typické požadavky na ně.

---

a) Rozdělení systémů RT s popisem.3
- Požadavky
    - Zpracování dat v daném čase – zpoždění muže znamenat nefunkčnost
    - Minimalizace rizika selhání systému
    - Konstrukční a signálová unifikace
- Rozdělení
    - Hard RT
        - Jsou dány limity, kdy musí systém reagovat

        - Jejich překročení znamená selhání systému
    - Soft RT
        - Jsou dány přibližné časové limity
        - Jejich překročení znamená pouze snížení užitečnosti systému
    - Firm RT
        - Odezva po limitu je k ničemu, ale systém pár zameškání snese a neselže
- Jiné dělení
    - **mission critical system** – porucha může mít katastrofální důsledky,
    - **dependable system** – systém natolik spolehlivý a bezpečný, že na něm můžeme být zcela závislí,
    - **fault-tolerant system** – systém odolný proti poruchám, porucha může snížit výkonnost systému, ale nesmí ho vyřadit z funkce: přednost mají úlohy kritické pro funkci systému, úlohy s nižší prioritou se provádějí, jen když na ně zbývá čas.

b) Typické vlastnosti RTOS, příklady.5 + 1

- Rychlé přepínání kontextu -
- Prioritní plánování, preempce procesů i jádra
- Malé rozměry
- Rychlý souborový systém – rychlé čtení a zápis dat
- Podpora speciálních systémových služeb – alarm, timeout
- Spolehlivost
- Multitasking s komunikací procesů – semafory, signál, fronty zpráv

c) Definujte pojmy latence a odezva (na úrovni přerušovacího systému).2

- Latence
    - Je to doba mezi příchodem požadavku na přerušení a začátkem provádění odpovídajícího obslužného programu
- Odezva
    - Je to doba, za kterou OS zareaguje na požadavek (na přerušovací signál)
    - Skládá se z doby latence a doby obsluhy
    - Doba obsluhy
        - Doba, která je potřebná ke zpracování přerušení

d) Definice a typické vlastnosti (alespoň šest) vestavěných systémů.4

- Počítačový systém, který je součástí jiného systému
    - Tvoří jeho řídicí složku, nebo subsystém
- Vlastnosti
    - Malá spotřeba
    - Odolnost
    - Malé rozměry a váha
    - Schopnost reagovat v reálném čase
    - Reaktivita – reagují na externí události
    - Spolehlivost
    - Bezpečnost
    - Cenová citlivost

---

17. Víceprocesorové systémy, rozdělení dle vazby a dle symetrie, granulovatelnost úlohy, souvislost s vazbou (víceprocesorových systémů) a stupně paralelismu, distribuované (rozptýlené, clusterové) OS.

a) Kategorie počítačových systémů z hlediska paralelizace zpracování dat.4

- SISD (Single Instruction Single Data)
    - Jeden procesor zpracovává jednu množinu dat jedním proudem instrukcí
- SIMD (Single Instruction Multiple Data)
    - Jedním proudem instrukcí se na více procesorech zpracovává více množin dat
- MISD (Multiple Instruction Single Data)
    - Více procesorů provádí různé operace na jedné množině dat
- MIMD (Multiple Instruction Multiple Data)
    - Více procesorů zpracovává různými instrukcemi různé množiny dat

b) Rozdělení víceprocesorových systémů dle vazby.2

- S volnou vazbou
  - Každý procesor má vlastní operační paměť a V/V subsystém
  - Různé typy vazeb
    - Společná sběrnice
    - Společný disk
    - Nic společného
- S těsnou vazbou
  - Procesory sdílejí jednu operační paměť
  - Ten je řízen jedním OS

c) Rozdělení víceprocesorových systémů dle symetrie.2

- Symetrický (SMP)
  - Stejné procesory
  - Jádro OS může být prováděno libovolným procesorem
  - Procesy a vlákna může provádět libovolný procesor
- Asymetrický
  - Procesory jsou specializované
  - Třeba grafický procesor, V/V procesor, …
  - Systém je řízen centrálním procesorem

d) Granularita úlohy: účel, jak se pozná, že úloha je/není granulovatelná.4

- Každá úloha může být rozčleněna na úseky, které lze provádět samostatně
- Tyto úseky mohou být prováděny paralelně na více procesorech
- Zrychlení vykonání procesů
-

e) Granularita a vhodné stupně vazby.3

- Hrubě granulovatelná úloha
  - Má méně sektorů
  - Je vhodné pro kooperující procesy
  - Pro tyto úlohy stačí víceprocesorový systém s volnou vazbou
- Jemně granulovatelná úloha
  - Více menších sektorů
  - Častější požadavky na synchronizaci

- Vyžaduje víceprocesorový systém s těsnou vazbou

1. Paralelismus
   - nezávislý paralelismus 10
   - velmi hrubý paralelismus 10
   - hrubý paralelismus 11
   - střední paralelismus 11

18. Soutěžení procesů (o prostředky), obecné problémy souběhu, vzájemné vylučování, kritická sekce, předpoklady pro řešení KS, požadované vlastnosti řešení KS, typy řešení.

- obecné problémy souběhu
  - vzájemné vylučování (mutual exclusion)
    - v každém okamžiku smí ke sdílenému prostředku přistupovat jen jeden proces (vlákno)
  - Synchronizace
    - Stav, kdy proces čeká na dokončení operace jiného prostředku
  - Stav uváznutí (deadlock)
    - Procesy vzájemně čekají na nějakou událost, která nemůže nastat
    - Třeba jeden proces čeká na druhý a opačně
  - Vyhladovění
    - Stav, kdy nekončí čekání na získání nějakého prostředku
    - Může nastat, pokud má proces nízkou prioritu

a) Definice kritické sekce v programu.4

- Je to část kódu, kde se pracuje se sdíleným prostředkem (paměť, CPU, soubory,…)
- Provádění této části kódu musí být vzájemně výlučné
  - Do kritické sekce smí vždy přistupovat jen jeden proces
  - Procesy žádají o přístup do kritické sekce

b) Předpoklady pro řešení přístupu do kritické sekce. (Co se předpokládá, že platí.)4

- Procesy se provádějí nenulovou rychlostí (proces musí běžet)
- Žádné předpoklady o relativní rychlosti procesů
  - Nepředpokládáme, že je jeden proces rychlejší, než druhý
- Uvažujeme i víceprocesorové systémy
  - Kritickou sekci zpřístupňuje jen jeden procesor
- Žádné předpoklady o prokládaném provádění
  - Nepředpokládáme, že se procesy pravidelně střídají
- Stačí specifikovat jen vstupní a výstupní sekci

c) Požadované vlastnosti řešení přístupu do kritické sekce (s vysvětlením).3

- Musí platit vzájemná výlučnost
- Progress – pokrok v přidělování
  - ZS (zbytková sekce)
  - Pokud je KS volná, musí v konečném čase proběhnout rozhodnutí o přidělení KS
- Omezené čekání
  - Pokud je prosem ve vstupní sekci a chce vstoupit do KS, musí se požadavku vyhovět v konečném čase (do této KS pak před ním smí vstoupit jen omezený počet procesů)

d) Typy řešení přístupu do kritické sekce (s příklady).4

- SW řešení (použití algoritmu pro vstupní a výstupní sekci)
  - Aktivní čekání
  - K doktorovy vždy smí jen 1 pacient. Sestřička volá do ordinace jednotlivé pacienty.??
  - K přepážce na poště smí jen 1 člověk, jeho číslo je napsané na tabuli. Pokud je tam jiné číslo, musí počkat.
- HW řešení (použití specializovaných instrukcí procesoru)
  - Použití instrukcí CPU
    - Například instrukci na zákaz přerušení
- Řešení OS (nabízí prostředky k řešení KS)
  -

- Řešení programovacího jazyka (konkurenční/souběžné programování)
  -

---

19. Řízení přístupu do kritické sekce pomocí SW metod, příklady nevhodných (obecně nefunkčních) SW řešení, SW algoritmy, vlastnosti (nedostatky) SW metod.

---

a) SW algoritmus s proměnnou locked, důvod nefunkčnosti.1 + 1

- Sdílená proměnná *locked*
  - Říká, jestli je KS volná, nebo ne
- Proces P_i čeká, dokud je KS obsazena. Jakmile se uvolní, proces vstoupí do KS a nastaví lockes na 1
- Problém: 2 procesy chtějí vstoupit do KS najednou. Pokud je locked = 0 (KS je volná), oba procesy vstoupí do KS. Neplatí princip vzájemného vylučování

b) SW algoritmus s proměnnou turn, důvod nefunkčnosti.1 + 1

- Vytváří se sdílená proměnná *turn*
- Turn se nastavuje na indexy jednotlivých procesů
-
- Máme 2 procesy, jeden má mnohem delší ZS než ten druhý
- Ten první pak bude čekat na vstup do kritické sekce i když bude KS volná
- Nesplňuje požadavek na pokrok

c) SW algoritmus s proměnnými flag[*i*], důvod nefunkčnosti.1 + 1

- Vytváří se sdílená proměnná *turn*
- Může se stát, že dva procesy nastaví flag[i] na true, pak bude první čekat na druhý a opačně
- Není splněn požadavek pokroku
  - Může vzniknout Deadlock

d) Funkční SW algoritmy pro řízení přístupu do KS, krátká charakteristika.2 + 2

- Petersonův algoritmus
  - Vytvoříme sdílené proměnné flag a turn
  - Pokud budou chtít 2 procesy vstoupit do KS, tak jen jeden z nich bude mít správně nastavenou proměnnou turn a vstoupí do KS, ten druhý musí počkat
  - Flag[i] říká, že proces je připraven vstoupit do KS
- Algoritmus bakery
  - Prý stačí vědět jen to, že se používá pro více procesů (pro *n* procesů)

proces P_i

```
repeat
  flag[i] = true;
  turn = j;
  while (flag[j]
      && turn == j);
  KS;
  flag[i] = false;
  ZS;
forever
```

e) Nevýhody SW algoritmů (včetně vysvětlení).5

- SW řešení nejsou odolné proti chybám v KS (muže se stát, že proces je ukončen (zabit) v KS)
  - Pokud nějaký proces havaruje v KS, tak to pro ostatní procesy vypadá tak, že je tak stále a už se do této KS nedostanou
- Aktivní čekání (stále se zjišťuje, jestli je KS volná)
  - Je vhodné jen pro velmi krátké KS
    - Krátká vzhledem k délce časového kvanta běhu na CPU
  - Pokud je KS dlouhá, tak procesy, které čekají na vstup do KS spotřebovávají čas procesoru (zbytečně a neproduktivně)
  - Lepší by bylo tyto čekající procesy blokovat

---

20. Řízení přístupu do kritické sekce pomocí HW metod, výchozí předpoklady pro HW řešení, algoritmy využívající HW instrukce, vlastnosti (nedostatky) HW metod.

---

a) Hardwarové předpoklady pro řízení přístupu do KS.3

- Procesy se provádějí v procesoru nepřetržitě, dokud nevyvolají službu OS nebo nejsou ukončeny
- K přerušení procesu může dojít pouze na hranicích instrukcí (mezi dvěma instrukcemi)
- Přístup k paměti je zpravidla výlučný (pouze jeden proces přistupuje)

b) HW podpora pro řízení přístupu do KS, možná řešení.2 + 3

- Použití instrukce, která zakáže přerušení -> obecně nevhodné řešení
  - Když se zakáže přerušení, tak proces nemůže být přerušen a jiný proces tedy nemůže vstoupit do KS

- Nevýhody
  - Funguje pouze na jednoprocesorových systémech
    - Na SMP není zajištěno vzájemné vylučování
  - Zvyšuje latenci systému, jelikož během KS nesmí být vykonána jiný událost
  - Jádro OS může aktivovat plánovač, ten může přepnout na jiný proces, který by mohl vstoupit do KS
  - Zákaz přerušení je privilegovaná instrukce
    - Normální procesy, která neběží v privilegovaném režimu to zakázat nemohou.

- Speciální instrukce
  - Musíme navrhnout instrukci, která provede čtení i zápis najednou (atomicky)
  - Provedení instrukce nelze přerušit
    - To nám zajistí vzájemné vylučování i na víceprocesorových systémech
  - Instrukce test-and-set
    - Jedna instrukce přečte příznak a současně ho i nastaví
    - Pokud byl už příznak nastaven, pak se nic nemění (KS už je obsazena)
      - Nevýhody (nedostatky)
    - Při čtení příznaku se používá Aktivní čekaní (stále se zjišťuje, jestli je KS volná)
    - Může dojít k vyhladovění
      - Pokud chce více procesů do KS po jejím uvolnění, vždy se tam dostane jen jeden a ostatní můžou vyhladovět
    - Může vzniknout Deadlock
      - Proces s nižší prioritou je přerušen v KS, proces v vyšší prioritou chce do KS (zabírá procesor pro sebe), ale proces s nižší prioritou nemůže KS opustit
    - Vstupní sekce využívá funkci testAndSet()
      - Instrukce xchg
    - Vymění obsah dvou proměnných
    - Pokud je KS volná, tak proměnná locked je 0, pak se do *s* uloží 0 a do locked 1
    - A skočí se do KS

inicializace
```
int locked = 0;
```
proces P_i
```
repeat
  while
    (testAndSet(&locked));
  KS;
  locked = 0;
  ZS;
forever
```

proces P_i
```
int s;
repeat
  s = 1;
  while (s)
    xchg(s,locked);
  KS;
  locked = 0;
  ZS;
forever
```

instrukce test-and-set
```
int testAndSet(int *lck)
{
  if (*lck == 0) {
    *lck = 1;
    return 0; // KS
  } else {
    return 1; // čekání
  }
}
```

c) Vlastnosti (nedostatky) HW řešení.4 + 3
- Snad už jsem to zmínil

---

21. Nástroj OS: semafor, jeho popis včetně systémových volání, použití semaforu pro řízení přístupu do KS a pro synchronizaci, problém obědvajících filozofů.

---

a) Popište nástroj OS: semafor.2
- Synchronizační nástroj a nástroj pro řešení KS
- Nevyžaduje aktivní čekání
  - Dokud je KS obsazená, tak jsou čekající procesy blokovány a ukládány do fronty procesů čekajících na uvolnění KS
  - Jakmile se KS uvolní, je vybrán další proces z fronty
- Obecně je to datová struktura, která obsahuje celočíselný čítač, frontu procesů a atomické operace *init, wait, signal*

b) Vysvětlete systémová volání semaforu.1 + 2 + 2
- Init – inicializuje čítač semaforu
- Wait – snižuje čítač semaforu
  - Pokud se tím dostane čítač do záporu, tak zařadí vlákno do fronty a blokuje je
- Signal – zvyšuje čítač semaforu
  - Je-li fronta neprázdná, tak:
    - Vybere vlákno z fronty a zařadí ho do seznamu připravených vláken

semafor – implementace
```
void sem_init(sem_t *s, int v){
  s->count = v;
}
void sem_wait(sem_t *s) {
  s->count--;
  if (s->count < 0) {
    fifo_put(s->q, self);
    block calling thread self;
  }
}
void sem_post(sem_t *s) {
  s->count++;
  if ((t = fifo_get(s->q)))
    activate thread t;
}
```

c) Popište řešení KS používající semafor.3

jeden z procesů P_i
```
sem_init(&s, 1);
```
proces P_i
```
repeat
  sem_wait(&s);
  KS;
  sem_post(&s);
  ZS;
forever
```

- Máme sdílený semafor, který se inicializuje na počet procesů, které mohou vstoupit do KS (bývá to jeden)
- Vstupní sekce procesu volá wait
- Výstupní sekce procesu pak zavolá signal

d) Popište řešení synchronizace vláken pomocí semaforu.3

- Mějme dvě vlákna P_0 a P_1
- P_0 počítá hodnotu x a P_1 pak nové x používá
- Aby se nestalo, že P_1 použije starou hodnotu x, musí počkat na dokončení P_0
  - K tomu se použije semafor
- Vlákno P_1 posílá operaci wait a čeká, dokud P_0 nepošle operaci signal
  - Tím se synchronizují obě vlákna a P_1 použije aktuální hodnotu x

inicializace

```
sem_init(&sync, 0);
```

proces P_0

```
vypočti x;  // S1
sem_post(&sync);
```

proces P_1

```
sem_wait(&sync);
použij x;   // S2
```

e) Definujte problém obědvajících filozofů a vyřešte jej pomocí semaforů. 2

- U stolu sedí 5 filosofů (každý buď přemýšlí, nebo jí
- Na stole je 5 vidliček
- K jídlu potřebuje každý filosof 2 vidličky
- Filosof = proces, jedení = kritická sekce, přemýšlení = zbytková sekce, vidličky = sdílené prostředky
- Řešení:
  - Dovolíme nevýše n-1 filosofům zvedat vidličky
  - Pak vždy alespoň 1 filosof může jíst (ostatní čekají)
  - Pro omezení zvedání vidliček použijeme semafor
  -

inicializace

```
sem_t waiter;
sem_init(&waiter, n - 1);
```

proces P_i

```
repeat
  think;   // ZS
  sem_wait(&waiter);
  sem_wait(&fork[i]);
  sem_wait(&fork[(i+1)%n]);
  eat;     // KS
  sem_post(&fork[(i+1)%n]);
  sem_post(&fork[i]);
  sem_post(&waiter);
forever
```

inicializace

```
for (i = 0; i < n; ++i) sem_init(&fork[i], 1); // vidličky
sem_init(&waiter, n - 1);  // n = 5, pouze čtyři smějí zvedat vidličku
```

proces P_i	P_0	P_1	P_2	P_3	P_4	f_0	f_1	f_2	f_3	f_4	w
think; // ZS	ZS	ZS	ZS	ZS	ZS	1	1	1	1	1	4
sem_wait(&waiter);	R	R	R	R	B_w	0	0	0	0		-1
sem_wait(&fork[i]);	R	R	R	R	R		-1	-1	-1	0	
sem_wait(&fork[(i+1)%n]);	B_{f_1}	B_{f_2}	B_{f_3}	R	B_{f_0}				0	1	0
eat; // KS	KS	KS	KS	KS	KS	-1		0	1	0	
sem_post(&fork[(i+1)%n]);	R	R	R	R			0	1			
sem_post(&fork[i]);	R	R	R	R		0	1				
sem_post(&waiter);				R							

22. Nástroj OS: předávání zpráv, popis systémových volání a možností blokování, použití fronty zpráv pro řízení přístupu do KS a pro synchronizaci, problém svázaných producentů a konzumentů.

a) Popište nástroj OS: předávání zpráv.3

- Používá se pro komunikaci mezi procesy – Je to prostředek k předávání dat mezi procesy
  - K tomu se používají 2 systémová volání
    - Send – pošli zprávu
    - Receive – přijmi zprávu
- Terminologie
  - Mailbox
    - Do mailboxu může zapisovat více procesů a více jich i může číst
    - Zprávu může poslat kdokoliv a já ji můžu číst odkudkoliv
  - Port
    - Na port může posílat zprávy více procesů, ale zpráva se posílá jen do jednoho místa (třeba na server)

b) Popište typickou implementaci blokování systémových volání pro předávání zpráv.3

- Obvykle se používá
  - neblokující send
    - jen pokud není fronta plná, pokud je, použije se blokující send
  - blokující receive
    - pokud není ve frontě žádná zpráva, pak je příjemce blokován
    - jakmile se do fronty pošle zpráva, blokování končí

c) Popište řešení KS používající frontu zpráv.3

- Na frontu zpráv se pošle 1 inicializační zpráva (nebo více, záleží na tom, kolika procesům najednou chceme povolit přístup to KS)
- Vstupní sekce procesu zavolá receive, tím získá tu zprávu a ostatní procesy budou blokovány
- Výstupní sekce pak zavolá send, tím tam tu zprávu opět pošle a do KS může přistoupit jiný proces
- Exkluzivitu přístupu zařídí OS

d) Popište řešení synchronizace vláken pomocí fronty zpráv.3

- Třeba proces čeká na událost jiného procesu – použití proměnné, výpočet proměnné
- Na začátku musí být fronta zpráv prázdná.
- Pokud se receive zavolá jako první, tak receive blokuje (proces čeká)
- Jakmile druhý proces zavolá send, znamená to, že výpočet je hotový a receive se odblokuje první proces a použije vypočítanou proměnnou
- Pokud se send pošle dříve než receive, nic se neblokuje

e) Definujte problém svázaných producentů a konzumentů a vyřešte jej pomocí fronty zpráv.3

- Definice jinde
- Pomocí front zpráv se tento problém asi řeší nejsnadněji
- Řešení:
  - Producenti posílají pomocí send položky do fronty zpráv (storage)
  - Tato fronta má danou velikost, pokud je plná, producenti jsou blokováni (send bude blokovat)
  - Konzumenti vybírají položky voláním receive, pokud je fronta prázdná, pak receive blokuje
  - Toto funguje i pro více producentů i konzumentů

23. Nástroj programovacích jazyků: koncept monitoru, problém producentů a konzumentů a jeho řešení pomocí monitoru.

a) Popište koncept monitoru: jeho účel, strukturu a základní vlastnosti.2 + 2

- SEMAFOR A MONITOR JE NĚCO JINÉHO
- Účel
  - Koncept návrhu řešení vzájemného vylučování, tedy kritických sekcí a synchronizací
  - Nástroj pro řešení problémů souběhu
- Konstrukce ve vyšším programovacím jazyce. Je to prostředek programovacího jazyka, který umožňuje řešit vzájemné vylučování a synchronizaci
- Dají se snadněji ovládat

- Mohou být implementovány pomocí semaforů
- Důležité!
  - V monitoru (tedy v jeho funkci) smí v jednu chvíli být maximálně jedno vlákno
  - Tím se zaručuje vzájemné vylučování
- Synchronizaci monitor zaručuje pomocí podmínkových proměnných
- Struktura:
  -

b) Popište řešení KS pomocí monitoru.2

- KS řešíme tak, že jí dáme do funkce monitoru. (do metody monitoru)
- Kód KS vložíme do funkce monitoru

c) Popište řešení synchronizace vláken pomocí monitoru.3

- Platí to samé jako pro podmínkové proměnné u posixových vláken
- Podmínkové proměnné jsou lokální proměnné uvnitř monitoru
- Ale jednodušší než u posixových knihoven, nemusí se použít mutex (exkluzivitu přístupu zařídí monitor)
- Používá se cwait(cv) (condition wait) - blokuje vlákno, dokud není zavoláno csignal(cv) - obnoví provádění vlákna blokovaného podmínkovou proměnnou cv
  - Pokud je vláken více, provádí se postupně (ostatní se uloží do fronty
  - Pokud takové vlákno není, signal se ztratí
  - Tyto funkce se používají pouze pro synchronizaci

d) Definujte problém svázaných producentů a konzumentů a vyřešte jej pomocí monitoru, zaměřte se též na podmínky pro synchronizaci a zdůvodněte její použití.6

- Máme jeden nebo více procesů generujících data (producenti) a několik procesů zpracovávajících vygenerovaná data (konzumenti). Pro snížení prodlev (zvýšení efektivity) je zaveden mezisklad, kam budou producenti data ukládat a odkud si je konzumenti budou vybírat.
- Přístup do skladu musí být řízen tak, aby producenti ukládali data pouze na volná místa a konzumenti vybírali každý jiná data z obsazených míst. Konzumenti musejí čekat, pokud je sklad prázdný, producenti musejí čekat, pokud je zaplněný.
- Do skladu může vždy přistoupit jen jeden
- Řešení
  - Vše, co je u tohoto problému v KS se vloží do monitoru
  - Producent vygeneruje data a zavolá metodu monitoru vlož data do skladu
  - Konzument vybere ze skladu a zpracuje
  - Jednoduchá implementace

Producent

```
repeat
  item = produce_item();
  PCmon.append(item);
forever
```

Konzument

```
repeat
  PCmon.take(&item);
  consume_item(item);
forever
```

Monitor: proměnné, init

```
monitor PCmon {
  item_t buffer[BUF_SIZE];
  cond not_full, not_empty;
  int in=0, out=0, count=0;
  void append(item_t item) {
    while (count == BUF_SIZE)
      cwait(not_full);
    buffer[in] = item;
    in = (in+1)%BUF_SIZE;
    count++;
    csignal(not_empty);
  }
  void take(item_t *item) {
    while (count == 0)
      cwait(not_empty);
    *item = buffer[out];
    out = (out+1)%BUF_SIZE;
    count--;
    csignal(not_full);
  }
}
```

24. Nástroje knihovny posixových vláken: mutex, bariéra, podmínková proměnná.

a) Popište mutex, jeho účel, způsob použití a vlastnosti.2 + 3

- Mutex je zámek (binární semafor, True-false)
- Zaručuje:
  - Atomicitu a vzájemné vylučování (mutex se dá zamknout jen jedním vláknem)
  - Neaktivní čekání tohoto vlákna (způsobuje)

- Typy:
  - Fast mutex (binární semafor) – dá se zamknout jen jednou
  - Recursive mutex – zámek, který lze zamknout na více západů
  - Error checking mutex – dá je zamknout jednou, pokud se pokusíme vícekrát, tak selže
- Inicializace:
  - Pthread_mutex_init
  - Int pthread_mutex_init(pthread_mutex_t *mutex)
  - Nebo pomocí makra
    - Pthread_mutex_t a_mutex = PTHREAD_MUTEX_INITIALIZER
- Uzamčení
  - Int pthread_mutex_lock(pthread_mutex_t *mutex)
- Pokus o zamčení
  - Int pthread_mutex_trylock(pthread_mutex_t *m)
- Odemčení
  - Int pthread_mutex_unlock(pthread_mutex_t *mutex)
  - Nelze odemknout vláknem, které nevlastní tento mutex
- Zrušení
  - Int pthread_mutex_destroy(pthread_mutex_t *mutex)
  - Zamčený mutex nelze zrušit

b) Popište bariéru, její účel, způsob použití a vlastnosti (včetně návratové hodnoty funkce čekání). 2 + 3

- Používá se pro synchronizaci vláken
  - Funguje jako běžecká dráha, jakmile přijdou všechna vlákna, tak se vystartuje
  - Umožňuje vláknu neaktivně čekat na ostatní vlákna
  - Nutno definovat
    - Třeba #define _XOPEN_SOURCE 600
    - Nebo #define _POSIX_C_SOURCE 200112L
- Inicializace
  - int pthread_barrier_init (pthread_barrier_t *restrict barrier, const pthread_barrierattr_t *restrict attr, unsigned count);
  - count udává, kolik vláken musí zavolat pthread_barrier_wait aby se odblokovala
- čekání na bariéře
  - int pthread_barrier_wait (pthread_barrier_t *barrier);
  - při úspěchu vrátí pro jedno vlákno PTHREAD_BARRIER_SERIAL_THREAD a pro ostatní nulu
    - právě toto vlákno uvolní prostředky
  - nenulová hodnota znamená číslo chyby
- Zrušení
  - int pthread_barrier_destroy (pthread_barrier_t *barrier);
  - 0 = úspěch, nenulová hodnota = číslo chyby

c) Popište podmínkovou proměnnou, její účel, způsob použití a vlastnosti. 2 + 3

- Slouží k synchronizaci vláken
- Umožňuje vláknu neaktivně čekat na událost
- Ale nezaručí exkluzivitu přístupu (musí se použít s mutexem!)
- Na událost může čekat několik vláken a může být probuzeno jen jedno nebo klidně všechna
- Událost oznamuje některé vlákno signálem
  - Pokud na tento signál nečeká žádné vlákno, pak se ztracen

25. Stav uváznutí (deadlock) a vyhladovění (definice a rozdíl), nutné podmínky pro vznik stavu uváznutí, předcházení a řešení problému stavu uváznutí, algoritmus bankéře.

a) Definujte stav uváznutí (deadlock). 2

- Vzájemné zablokování dvou či více procesů procesů
- Deadlock skupiny procesů nastane, pokud každý proces ve skupině čeká na událost, kterou může vyvolat jen jiný proces ze skupiny

b) Definujte stav vyhladovění (starvation), uveďte příklad. 1

- Nekončící čekaní procesu na prostředky

- Například, pokud má proces nízkou prioritu, může se stát, že je stále předbíhán procesy s vyšší prioritou, a tak vyhladoví.
- Dá se říct, že důsledkem stavu uváznutí je vyhladovění

c) Vysvětlete podmínky pro vznik stavu uváznutí.4

- Coffmanovy podmínky (všechny jsou nutné, pokud je jedna porušena, deadlock nenastane)
  - Vzájemné vylučování
    - Prostředek, který se účastní deadlocku, smí vlastnit jen jeden proces
  - Alokace a čekání
    - Proces vlastní nějaký prostředek a vyžaduje (čeká) další
  - Neodnímatelné prostředky
    - Jsou to prostředky, které mohou být uvolněny pouze vlastnícím procesem (ani OS je nemůže odebrat)
  - Cyklické čekání
    - Procesy si vzájemně čekají na prostředky tak, že se to uzavře do cyklu (kruhu)

d) Navrhněte způsoby řešení stavu uváznutí.3

- Lze to ignorovat
- Detekce a obnovení
  - Vrátíme se do stavu před deadlockem (obnovení systému)
  - Můžeme násilně odebrat prostředky (třeba odebrání periferie)
  - Můžeme najít a ukončit proces, který způsobuje deadlock
- Prevence vzniku
  - Tím, že vyloučíme alespoň jednu z podmínek deadlocku

e) Navrhněte způsoby prevence vzniku stavu uváznutí.3

- Negace vzájemného vylučování
  - Spooling
    - Odložené zpracování požadavků
    - Prostředek bude vlastněn jedním procesem, který bude řídit přístup k tomuto procesu
- Negace alokace a čekání
  - Zajistíme, aby se všechny prostředky alokovaly najednou
    - Zablokujeme proces, dokud nejsou všechny prostředky dostupné
- Negace neodnímatelných prostředků
  - Pokud to jde, zavedeme možnost násilného odebrání prostředků
- Negace cyklického čekání
  - Zakážeme alokaci prostředku, pokud by vznikl cyklus
  - Definujeme lineární uspořádání na prostředcích (očíslujeme je)
    - Dovolíme procesu alokovat pouze prostředky s vyšším pořadovým číslem, než má jakýkoli jemu přidělený prostředek

f) Popište bankéřův algoritmus aplikovaný na procesy.2

- Je založen na odmítnutí nebo odložení požadavků, které by uvedly systém do nebezpečného stavu. Vychází z jednání bankéře, který má finance a půjčuje je postupně klientům. Přitom se nesmí dostat do situace, že by podle dohody měl vyplatit klientovi sumu, kterou by neměl právě k dispozici.
- Výchozí předpoklady
  - Pevný počet prostředků
  - Každý proces musí deklarovat své maximální požadavky na prostředky, tak aby nepřesáhli maximální prostředky systému
  - Lze postupně alokovat prostředky, ale mohou být zamítnuty (takové, které by vedly na nebezpečné stavy)

26. IPC: komunikace procesů a vláken, možné prostředky komunikace.

a) Vyjmenujte možné prostředky pro komunikaci procesů / vláken.5

- Soubor, databáze (nejméně efektivní)
- Roura
  - Je to jednosměrný komunikační nástroj pro dva procesy

        - Tedy jeden proces zapisuje (posílá) a druhý čte (přijímá)
        - Dokud druhý proces rouru neotevře, blokuje operce na ní
        - Nevýhoda
            - Je pouze jednosměrná
        - Vytvoření
            - Int pipe (int filedes[2])
            - To vytvoří pár deskriptorů propojených rourou
    - Socket
        - Je to obousměrný komunikační nástroj pro dva procesy
        - Domény (rodiny) socketů: například IPv4, IPv6
        - Funguje mezi různými systémy
        - Typy:
            - STREAM
            - DGRAM
        - Vytvoření
            - Int socketpair (int d, int type, int protocol, int sv[2]
            - d je doména (třeba **AF_UNIX**, **AF_INET**, **AF_INET6)**, type je typ (například **SOCK_STREAM**, **SOCK_DGRAM),** protocol: 0 znamená výchozí protokol pro daný typ socketu
    - Fronty zpráv
        - Je to jednosměrná komunikace pro více procesů
    - Sdílená paměť (nejvíce efektivní)

b) Krátce charakterizujte jmenované prostředky.5

- ↑

c) Popište funkce pro sokety.5

- Klientské funkce:
    - getaddrinfo – překládá adresu
    - socket – alokuje soket
    - bind – spojí soket s lokálním portem
    - connect – naváže spojení
    - send/write a read/recv – používají se pro komunikaci
    - close – zavření soketu

---

27. Dělení disku na oddíly, zavaděč OS, důvody dělení, MBR, GPT, swap.

a) Popište důvody pro rozdělení disku na oddíly.4

- Na jednom disku můžeme mít více OS
- Chceme oddělit systém od dat – snadněji se spravuje
- Snadnější obnova dat při chybě – může stačit obnovit jeden oddíl
- Vytvoření oddílu pro swap – oddíl nemá datovou fragmentaci, pak je rychlejší přístup

b) Popište MBR a způsob dělení disku na oddíly.3

- MBR – Master Boot Record
- Je to hned první sektor disku
- Je v něm umístěn zavaděč OS a tabulka rozdělení disku na oddíly
- Dokáže rozdělit disk na 4 primární a 1 rozšířený oddíl
- Disk se dělí na:
    - Primární oddíly, ze kterého lze pomocí sekundárního zavaděče zavést OS
    - Rozšířený oddíl, ten lze rozdělit na libovolný počet logických oddílů (disků)
        - Rozšířený oddíl obsahuje EBR (Extended Boot Record), který obsahuje informaci o následujícím logickém oddíle a jeho EBR

c) Popište typy oddílů (pro dělení MBR).3

- ↑

d) Charakterizujte GPT.3

- Je nástupcem MBR

- Je součástí standardu UEFI
- MBR dokáže adresovat jen disky do 2 TiB, tento limit GPT odstraňuje
- Ukládá si zálohu na konci disku
- Umožňuje rozdělit disk až na 128 oddílů

e) Vysvětlete rozdíly mezi odkládacím prostorem (swap) na samostatném diskovém oddíle a v souboru.2

- V souboru má swap datovou fragmentaci, ale na samostatném oddíle ne
- Díky tomu je přistup ke swapu na samostatném oddíle rychlejší

28. Souborový systém, metadata, speciální soubory.

a) Popište obecnou strukturu souborových systémů (metadata souborových systémů a souborů).2 + 3

- Umožňuje přístup uživatelům a procesům k souborům na paměťovém mediu (podle jejich jména)
- Vytváří adresářovou strukturu souborů
- Eviduje metadata souborů (jejich atributy, oprávnění, …)
- Sjednocuje přístup v datům různého typu
- Metadata
  - Data o datech
  - Metadata FS obsahují
    - Velkost FS, vlastnosti FS, seznam volných bloků a odkaz na evidenci metadat souborů
  - Metadata souborů obsahují
    - Typ, velikost, čas změny, umístění, oprávnění

b) Uveďte atributy souborů (alespoň šest).3

- Hidden, read-only, systém, archive, compressed, encrypted, temporary, not content indexed

c) Uveďte možná oprávnění na soubor (alespoň šest).3

- Žádná – nesmíme se souborem dělat nic
- Provádění (execute) – soubor můžeme spustit
- Čtení – smíme zjistit obsah souboru
- Mazání – smíme smazat obsah souboru
- Vytvoření – smíme vytvořit nový soubor
- Změna oprávnění – smíme měnit oprávnění jiným uživatelům

d) Popište zvláštní typy souborů (alespoň čtyři).2

- Odkazy – jen symbolický odkaz
  -
- Pojmenovaná roura
  - Jednosměrný komunikační nástroj pro dva procesy
  - Jakmile se data přečtou, odstraní se
- Pojmenovaný socket
  - Obousměrná Síťová komunikace pro dva procesy
- Soubor Zařízení
  - Slouží ke komunikaci s periferiemi
  - Jsou bloková a znaková zařízení
    - Bloková – s náhodným přístupem po blocích (disky)
    - Znaková – proudový přístup po bajtech (tiskárny)

e) Charakterizujte typy odkazů a jejich reprezentaci na souborovém systému.1 + 1

- Pevný odkaz
  - Je to soubor, který se odkazuje na stejná data a metadata jako nějaký existující soubor
  - Je to tedy nové jméno pro existující soubor
  - Jakmile se odstraní poslední odkaz, dojde i k smazání dat
- Symbolický odkaz
  - Textová záměna za jiné jméno
  - Pro procesy je tato záměna transparentní
  - NENÍ TO ZÁSTUPCE (SHORTCUT)

29. Konzistence metadat souborových systémů: příčiny vzniku nekonzistencí, metody zachování konzistence, vlastnosti metod, příklady souborových systémů.

a) Popište příčiny vzniku nekonzistence metadat souborového systému.3

- Zápis dat do souboru znamená provedení operací na různých místech
  - Metadata FS – alokace nových bloků, aktualizace volných bloků
  - Data FS – zápis nových dat souboru
  - Metadata souboru – aktualizace přidělených bloků, velikosti a času změny
- Důvod nekonzistence je, že tyto změny probíhají postupně

b) Popište princip žurnálování a činnost obnovy konzistence po pádu.5

- Zabraňuje nekonzistenci dat při pádu systému
- Změny metadat se zapisují do transakčního logu (žurnálu), jakmile se to dokončí, začne se zapisovat do FS
- Žurnál je kruhový buffer
- Když spadne systém, prochází se žurnál a kontroluje se konzistence na místech posledních změn
- Žurnál nezabrání ztrátě dat

c) Popište princip metody copy-on-write a činnost obnovy konzistence po pádu.4

- Nemodifikují se data na místě
- Při zápisu nových dat se vytvoří kopie bloku a upravuje se ta
- Po dokončení se atomicky provede zneplatnění starých dat a metadat a potvrzení (platnost) nových
- Nevzniká žádná nekonzistence
- Vzniká větší datová fragmentace

d) Uveďte příklady souborových systémů ve vztahu k metodě zachování konzistence (alespoň dva pro každou metodu, pro žurnálování dvojí způsob).3

- FS bez konzistence po pádu
  - FAT, exFAT, HFS (apple)
- Žurnálovací FS
  - Úplné
    - Ext3, ext4, ReiserFS
  - Pouze metadata
    - NTFS, VMFS (VMware)
- FS s podporou copy-on-write
  - ZFS, ReFS

30. Typy úložišť, RAID, způsob alokace dat souborů.

a) Popište typy úložišť (DAS, NAS, SAN).3

- DAS (directly attached storage)
  - To je lokální paměťové uložiště (blokové zařízení)
  - Například SCSI, SATA, IDE
- NAS (Network attached storage)
  - Souborový systém, který je zpřístupněný síťovým protokolem
  - CIFS, NFS
- SAN (Storage area network)
  - Blokové zařízení, které je zpřístupněno síťovým protokolem
  - iSCSI, Fibre Channel

b) Popište vícedisková úložiště (RAID), účel.3

- RAID (Redundant Array of Independent Disks)
- Je to zapojení více disků do jednoho virtuálního zařízení
- Jsou uspořádány tak, aby se prokládáním a ukládáním dat vytvořily redundantní (rezervní) data
- Pokud selže jeden disk, data jsou ještě uložena na jiném disku

c) Krátce charakterizujte úrovně RAID 0, 1 a 5 (způsob zapojení, ukládání dat, vlastnosti).3

- Raid 0
  - Zapojení dvou a více disků
  - Data se prokládají bit po bitu mezi více disky
  - Pokud disk selže, pak jsou všechna data ztracena, jelikož tu není žádná záloha
  - Výhoda je rychlost čtení a zápisu

- RAID 1
  - Je tvořeno dvěma disky, mezi kterými se veškerá data zrcadlí
  - Pokud jeden disk selže, máme ještě všechna data na druhém disku
  - Nevýhoda je, že využijeme pro data kapacitu jen jednoho disku za cenu dvou
- RAID 5
  - Je tvořeno třemi disky
  - Data mezi dvěma disky se prokládají
  - Třetí disk se používá pro obnovu dat, pokud jeden z disků selže
  - Rychlejší čtení, ale pomalejší zápis
  - Výhoda: záloha dat

d) Uveďte a charakterizujte možné způsoby alokace dat pro soubory.3× 2

- Soubory alokují paměť po alokačních blocích (cluster)
- Bloky nemusejí být plně využity – vzniká vnitřní fragmentace
- Metody alokace
  - Souvislá alokace
    - Souboru se alokují po sobě jdoucí bloky v paměti
    - V metadatech souboru se eviduje první blok a velikost v blocích
    - Při alokaci prostoru pro soubory vzniká vnější alokace
    - Pokud soubor nemá více místa pro rozšíření, musí se celý přestěhovat na jiné místo v paměti
  - Řetězená alokace
    - Souboru se alokují jednotlivé bloky, které mohou být na různých místech v paměti
    - Tvoří se alokační tabulka
      - Každý blok alokační tabulky odkazuje na následující blok
    - V metadatech se eviduje první blok
    - Nevzniká vnější fragmentace, ale tím, že jsou data souboru na různých místech v paměti vzniká datová fragmentace
    -
  - Indexová alokace
    - Metadata obsahují index, který je tvořen seznamem alokovaných bloků souboru
    - Indexy se mohou odkazovat i na bloky s indexem – nepřímá index
    - Není vnější fragmentace
    - Bloky mohou být různě velké, takže se snižuje vnitřní fragmentace

---

31. Vzdálený přístup k OS, telnet, SSH, autentizace, autorizace, zásady tvorby hesla, typy útoků na systém a prevence.

---

a) Definujte pojmy autentizace a autorizace.2

- Autentizace – ověření identity
  - Identitu lze ověřovat více způsoby:
    - Jméno a heslo
    - Heslo na jedno použití
    - Soukromí klíč (SSH)
    - Otisky prstů, oční sítnice
- Autorizace – ověření oprávnění
  - Co smí uživatel v systému dělat a kam smí přistoupit

b) Uveďte možné metody autentizace (alespoň čtyři).2

- ↑

c) Popište princip zabezpečení přihlašování pomocí protokolu SSH.3

- Vzdálený přístup k OS
  - vzdálený přístup do sítě či k aplikaci je metoda zajišťující přímou nebo nepřímou komunikaci klienta se vzdáleným systémem
- Telnet
  - Protokol používaný v počítačových sítích
  - Umožňuje uživateli připojení ke vzdálenému počítači

    - Komunikace se posílá jako normální text (není šifrováno)
- SSH
    - Stejné jako telnet, ale komunikace je šifrována pomocí veřejného a privátního klíče
    - Uživatel si tyto dva klíče vygeneruje a svůj privátní klíč si uschová u sebe.
    - K první komunikaci mezi počítači se používají certifikační autority

d) Uveďte riziko přihlašování pomocí protokolu SSH a jak mu předcházet. 2 + 2

- Někdo může sledovat první komunikaci mezi námi a počítačem na druhé straně

e) Uveďte zásady tvorby hesla (čtyři). 2

- nemělo by obsahovat jméno uživatele
- posloupnost znaků/čísel
- alespoň 8 znaků
- používat velká písmena, speciální znaky,
- volit jiná hesla pro každou službu

f) Popište alespoň čtyři typy útoku na systém. 2

- Trojský kůň
    - Nevinně vypadající programy a funkcí navíc
- Virus, červ
- Login spoofing
    - Imitace přihlašovací obrazovky OS
- Logická bomba
    - Škodlivý kód se spouští v určitý čas
- Stack overflow
    - Neošetření maximální délky vstupu
