PB152 – OPERAČNÍ SYSTÉMY – POZNÁMKY

SEMESTR JARO 2013

Jedná se o přepracované poznámky z fi.muny.cz
doplněné/opravené podle slajdů doc. Staudka

xvoda@mail.muni.cz

Lecture_01 Principy_HW

OS je program, který funguje jako spojka mezi uživatelem PC a hardwarem. Spravuje a řídí využívání všech zdrojů počítače.

Cíle OS:

- Efektivně využívat hardware počítače
- Řídit uživatelské (aplikační) programy – které pracují na vysoké úrovni a využívají jen volání OS (nemohou k HW přistupovat přímo)
- Učinit počítač jednodušším a bezpečněji použitelným

Architektura počítačového systému, součinnost I/O a CPU:

- I/O zařízení (periferie) a CPU mohou pracovat souběžně
- Řadič periferie je odpovědný za její chod
- Každý řadič má vyrovnávací paměť – buffer
- CPU (DMA) přesouvá data z bufferu do operační paměti a zpět
- Řadič periferie informuje CPU o své činnosti přerušením, nebo změnou v registrech
- Př. TSS – Time Sharing Systems (multitasking)

Procesor: získává instrukce z operační paměti, dekóduje je a sekvenčně vykonává

- Instrukce: operace nad hodnotami, skoky, řízení I/O, zápis/čtení z paměti
- Množina instrukcí je specifická pro jistý typ procesoru
- Je vybaven rychlými interními pamětmi s malou kapacitou – **registry**:
    - **Viditelné uživateli:**
        - Dostupné OS i uživatelským procesům
        - obsahují data, adresy, flagy, ukazatele…
    - **Stavové a řídíci:**
        - nedostupné uživatelským procesům
        - některé používá CPU pro řízení vlastních operací (IR)
        - některé používá OS pro říz. provádění programů (PC)

Hlavní paměť: = operační paměť, FAP …, energeticky závislá, vedle registrů jediná paměť dostupná procesoru přímo (sběrnicí)

Přerušení:

- potlačení provádění běžícího procesu v CPU tak, aby jej bylo možné později obnovit
- potřeba provést jinou posloupnost příkazů v reakci na nějakou událost
- cílem je zlepšení účinnosti překrývaní více operací v čase
- CPU testuje nutnost přerušení po každém provedení instrukce
- Přerušení předává řízení správci přerušení (interrupt handler) pomocí vektoru přerušení
- Mechanismus přerušení musí uchovat adresu instrukce prováděné jako příští

Systémová sběrnice: komunikační prostředek mezi CPU, pamětí a periferiemi pro přenos adres a dat

Metody obsluhy I/O:

- **Busy waiting** (činné čekání):
  - V systémech bez řízení IO pomocí OS
  - Žádné souběžné zpracování I/O s během programu
  - Nevyřešený zůstává nejvýše jeden požadavek
  - Program testuje konec IO operace opakovanými dotazy na příslušný stavový registr IO zařízení
- **Přerušením:**
  - V systémech s řízením IO pomocí OS
  - Souběžné zpracování I/O s během programu
  - Synchronní řešení IO – proces čeká na dokončení IO operace
  - Asynchronní řešení IO – proces nečeká na dokončení IO operace

Direct Memory Acess (DMA):

- Používá se pro velmi rychlá I/O zařízení pro přednost dat do/z pamětí
- Rychlost je blízka rychlosti vnitřní pamětí
- Řadič periferie přenáší bloky dat mezi vyrovnávací pamětí a periferií bez zásahu CPU – cycle stealing (kradení cyklů)
- Přerušení se generuje po přenesení celého bloku

Struktura pamětí:

- **Hlavní paměť (operační paměť):**
  - Adresovatelná matice buněk
  - CPU do ní může přistupovat přímo
  - Energeticky závislá
  - Kapacita v řádech desítek MB až GB
  - Rychlost přístupu v řádech desítek nanosekund
- **Sekundární paměť (magnetický disk):**
  - Energeticky nezávislá
  - Kapacita v řádech GB až TB
  - Rychlost přístupu v jednotkách ms
- **Terciální paměť (řízené sklady pásek):**
  - Archivní média
  - Maximální kapacita, nejpomalejší přístup

Caching:

- Mikroprogramem řízené kopírování informací z RAM do rychlejší paměti
- Rychlejší zpřístupňování aktuálních dat/instrukcí
- Problém udržení konzistence více kopií téže dat
- MAGNETIC DISK > MAIN MEMORY > CACHE > HW REGISTER

Bezpečnostní mechanismy OS:

- Nesprávný program nesmí negativně ovlivnit běh jiných programů nebo OS
- 2 režimy procesoru:
    - **User mode:**
        - má omezený instrukční repertoár, aby uživatelský program nezasahoval do paměti a zdrojů, které mu nepatří
        - procesor do něj přechází privilegovanou instrukcí
    - **Kernel mode:**
        - nemá omezený instrukční repertoár, může provádět i tzv. privilegované instrukce
        - procesor do něj přechází přijetím přerušení
- Privilegované instrukce se mohou provádět pouze v privilegovaném režimu
- CPU se po přijetí přerušení automaticky přepne do privilegovaného režimu
- Ochrana paměti – souvisí se správou paměti
- Ochrana dostupnosti CPU – časovač, souvisí s plánováním

Lecture_02 Operační systémy – Úvod

OS v roli rozhraní uživatel/počítač:

- Koncoví uživatelé vidí počítač jako sestavu aplikačních programů
- Pro složité ovládání mají k dispozici funkčnost součástí OS (knihovny, pomocné programy)
- Programy pro efektivní a pohodlné využívání počítače jsou soustředěny v jádře OS a jsou poskytovány službami OS

Služby poskytované OS:

- Vývoj programů – editory, ladící systémy … (pomocné programy)
- Provádění programů – plánování, zavádění do paměti …
- Přístup k IO zařízením – OS poskytuje jednotné API pro různá zařízení
- Přístup k souborům dat – na vnějších pamětech
- Přístup k systémovým zdrojům – bezpečnost, řešení konfliktů
- Chybové řízení – reakce na nestandartní stavy
- Protokolování – info o tom, co se dělo

OS v roli správce zdrojů:

- PC je sestavou zdrojů/prostředků pro přesun, uchování a zpracování dat
- OS je odpovědný za správu těchto zdrojů
- OS je suita programů prováděných procesorem

Problémy budování OS: jsou to obrovské a složité systémy, těžko předpovídatelé

Komponenty počítačového systému: hardware, OS, systémové programy, aplikační programy, uživatelé

Klasifikace počítačů:

- **Střediskový počítač:**
  - Řadil úlohy do dávek (batch processing)
  - Historický (v současnosti podnikové servery)
  - Řízen monitorem
  - První koncept multiprogramového režimu činnosti
- **Osobní počítač:**
  - pro jednoho uživatele, kompletní IO vybavení, multitasking
  - Windows, Linux, MacOS
- **Paralelní systémy:**
  - Více procesorů, společný FAP = úzce provázané
  - Symetrický multiprocessing (SMP) – více procesů běží zároveň na různých procesorech, běžné v moderních OS
  - Asymetrický multiprocessing (AMP) – každý procesor má svou úlohu (master/slaves)

- **Distribuované systémy:**
    - Více počítačů, nesdílí FAP
    - Komunikují předáváním zpráv = volně prováz.
    - Síťové infrastruktury LAN, WAN – model P2P nebo K/S
- **Shluky (clustering):**
    - Pod řízením OS jde shlukovat více procesů
    - Cíl: vysoká dostupnost řešené služby

Rysy potřebné k implementaci multiprogramování:

- **Ovládání I/O:**
    - Privilegované/neprivilegované instrukce
- **Správa paměti:**
    - Dynamické přidělování paměti
    - Fyzický a logický adresový prostor
    - Ochrana paměti před neautorizovaným přistupem
- **Mechanismus přerušení:**
    - Reakce na asynchronní události
- **Plánování CPU:**
    - Časovač
    - Po uplynutí časového intervalu se generuje přerušení
- **Přidělování zdrojů:**
    - Dynamicky

Jádro OS:

- logické rozšíření rysů hardware + poskytované služby
- vše mimo jádro je řešeno formou procesů
- může využívat speciální rysy hardware nedostupné procesům

Mikrojádro OS:

- Minimalistická varianta jádra OS používaná v některých OS
- Zabezpečuje: správu paměti, s. přerušení, s. procesorů, s. procesů, komunikaci mezi procesy předáváním zpráv – Interprocess communication (IPC)
- Ostatní funkce jádra se přesouvají do uživatelské oblasti (drivery, služby systému souboru, virtualizace paměti …) = minimum funkcí v privilegovaném režímu

Fynnova kategorizace výpočetních systémů:

- SISD = Single Instruction Single Data – klasický 1-procesorový systém, OS s multitaskingem
- SIMD = Single Instruction Multiple Data – vektorové procesory
- MISD = Multiple Instruction Single Data – každý procesor vykonává nad stejnými daty jiné operece, nikdy prakticky neimplementované
- MIMD = Multiple Instruction Multiple Data – těsně a volně provázané systémy

Time – Sharing System (TSS) – periferní zařízení i CPU mohou pracovat souběžně = multitasking

Přínos OS pro počítačové vědy – procesy, správa paměti, ochrana informací a bezpečnost, plánování a řešení zdrojů, strukturování systémů

Proces:

- Identifikovatelná jednotka činnosti charakterizovatelná sekvenčním prováděním operací
- Program je strukturovaný příkaz, proces je děj
- Program umístěný v paměti může řídit běh více procesů
- Souběžné řešení více procesů = multitasking
- Souběžnost řešení procesů může být příčinou časově závislých chyb

Vlákno:

- Jednotka plánování činností definovaná v programu
- Vlákna využívají zdroje přidělené procesu
    - Jednovláknový proces: jediný čítač instrukcí určující přiště prováděnou instrukci, sekvenční vykonávání instrukcí
    - Vícevláknové: jeden čítač instrukcí pro každé vlákno, souběžné vykonávání vláken, jejichž instrukce jsou vykonávány sekvenčně

Příčiny časově závislých chyb v procesech:

- **Nesprávná synchronizace:**
    - Jeden proces očekává událost, kterou generuje jiný proces
    - Proces, který generuje událost, signalizuje její vznik
    - Signály se nesmí ztrácet ani duplikovat
- **Chybné vzájemné vyloučení:**
    - Editovat sdílenou datovou strukturu smí v jednom okamžiku pouze jeden proces
    - Souběžné procesy se musí při takové editaci vzájemně vylučovat, změna stavu sdílené struktury musí proběhnout nedělitelně
- **Nedeterminismus operací:**
    - Činnost procesu má záviset pouze na jeho vstupech, ne na činnosti jiných procesů
    - Pokud procesy sdílí paměť, pak při prokládání jejich běhů nesmí výstupy ovlivňovat jejich pořadí, ve kterém je jejich běh plánován
- **Uváznutí:**
    - Dva nebo více procesů na sebe vzájemně čekají
    - První proces výlučně drží zdroj A, druhý zdroj B a první požaduje přístup k B a druhý k A.

Lecture_03 Struktura OS

Typologie OS:

- **Mainframe OS:**
  - OS datových center
  - Spousta periferií (tisíce disků, terabajty dat)
  - Obrovské množství souběžně běžících procesů
  - Orientace na Linux
- **Server OS:**
  - Vysoký výkon, paměťová i komunikační kapacita
  - Obsluha a poskytování služeb mnoha vzdáleným uživatelů
  - Solarix, Linux, Windows Server, FreeBSD
- **Multiprocessor OS:**
  - Souběžnost více funkcí OS (někdy problém)
  - Pro počítače s vícejádrovým CPU
  - Linux, Windows
- **Personal Computer OS:**
  - Podpora multitaskingu, podpora jednomu uživateli
  - Linux, Windows Vista/7/8, MacOS
- **Handheld OS:**
  - OS pro Smartphone, PDA
  - Nemají vnější paměť
  - Běžně spouštění aplikace třetích stran (ne vždy bezpečné)
  - Android, Symbian OS
- **Embedded OS:**
  - V autech, mikrovlnkách, DVD přehrávačích
  - Vše je uložené v ROM = uživatel nemůže do OS instalovat
  - QNX, VxWorks
- **Sensor node OS:**
  - OS uzlů senzorových sítí
  - Malá paměť, nutná velká výdrž baterie, bezdrátová komunikace
  - Vše je nainstalováno předem
  - TinyOS
- **Smart card OS:**
  - Malý výkon, paměť, jednoduché, př. JVM
- **Realtime OS (striktní a tolerantní):**
  - Má definované časové limity pro zpracování instrukcí
  - Často nemá vnější paměť ani interpret příkazů
  - Jednoúčelové: řízení výrobních procesů, monitorovací sys.
  - Důraz na minimální latence
  - Prioritní plánování běhu procesů
  - Preemptivní plánování – prioritnější proces předbíhá méně prioritní

Generické komponenty OS:

- **Správa procesorů:**
    - Jejich krátkodobé plánování dispečerem
    - Vytváření, rušení, pozastavování a obnova běhu procesů (multitasking) a vláken
    - Plánování válen podle typu OS
- **Správa hlavní paměti:**
    - Její přidělování a uvolňování podle potřeby (své i procesů)
    - Vedení přehledu, který proces kterou část paměti využívá
    - **FAP =** pole samostatně adresovatelných jednotek, repozitář instrukcí a dat
    - **LAP =** uchovává programy, transformace LAP na FAP (přesun dat do fyzické paměti) až při vykonávání daní instrukce
    - **Struktura LAP:**
        - Lineární (pole stránek) – virtualizace stránkováním
        - Dvoudimenzionální (2D pole) segmentů – virtualizace segmentováním
    - Lineární LAP může být do FAP zobrazen identitou, nebo pomocí HW (Dynamic Address Translation DAT, Memory Managment Unit MMU)
    - Při odkázání na místo, které není ve FAP správa paměti naleze (vytvoří) ve FAP volný blok, kde přenese požadovaná data
- **Správa procesů a vláken:**
    - Vytváření a rušení uživatelských a systémových procesů
    - Potlačení a obnovování běhu procesů
    - Poskytnutí mechanismů pro synchronizaci, komunikaci, uváznutí
    - Výběr procesu běžícího na (dostupném) procesoru
    - Plánování vláken
- **Správa souborů:**
    - Vytváření a rušení souborů
    - Vytváření a rušení adresářů a organizace souborů do nich
    - Přidělování přístupových práv
    - Archivování na stabilní, energeticky nezávislá média
- **Správa I/O systému =** ovladače, cache paměti, spooling (překrývání výstupů jednoho procesoru a vstupu druhého)
- **Správa vnější paměti:**
    - Správa volné paměti na disku, její přidělování souborům
    - Plánování činností disku
- **Networking (síťování), distribuované systémy** = kooperace a sdílení zdrojů
- **Systém ochran:**
    - Řízení přístupu ke zdrojům
    - Rozlišování ne/autorizovaných přístupů

        - Ochranné opatření proti vnějším/vnitřním útokům
- **Interpret příkazů** – zadává požadavky na služby OS
- **Systémové programy**

Hlavní přístupy k rozvoji architektu OS:

- **Mikrojádrová arichtektura:**
    - Jádro OS: správa paměti, procesů, základní planování
    - Koncept zjednodušuje implementaci OS, pružnost, vhodná pro distribuované prostředí
- **Multi-vlákna:**
    - Proces lze řešit souběžnými sekvenčními toky operací – vlákny
    - Vlákno je jednotkou plánování, ne subjekt vlastnící zdroje
    - Vlákno je částí procesu, proces je subjekt vlastnící zdroje
    - Má svůj kontext a svou datovou oblast, může volat podprogramy
    - Programátorovi dává silnější nástroje pro modularitu aplikace a časového řízení událostí souvisejících s aplikací

Virtualizace:

- Metoda, která nám umožňuje nahlížet na jednu entitu jako na více logických entit, nebo slučovat více fyzických entit do jednoho virtuálního clelku
- **Systém** – kompletní výpočetní prostředí složené ze zdrojů (výpočetních, paměťových, I/O) a operací, které nad nimi můžeme vykonat
- **Virtuální stroj:**
        - Je výpočetní prostředí, do kterého můžeme instalovat běžný OS a v tomto OS provozovat běžné operace formou procesů
        - Nebere ohled na okolní virtuální systémy a předpokládá, že má exkluzivní přístup ke všem zdrojům hostitelského systému
        - Systémová virtualizace zavádí multitasking na úrovni počítačů řízených běžnými OS
        - Multitasking hostovaných serverů řeší manažer – hypervizor
        - Lze obtížně implementovat
- V každém virtuálním stroji lze provozovat jiný OS
- Izolaci a sdílení zdrojů zajišťuje hypervizor
- **Implementace:**
        - Virtuální stroj běží ve fyzickém uživatelském režimu hostujícího počítače
        - Musí se chovat jako samostatný systém, mít virtuální uživatelský i privilegovaný režim
        - Proces běžící ve virtuálním uživatelském režimu zavolá službu OS
        - Přerušení aktivuje virtualizační software běžící v privilegovaném režimu hostitele
        - Tento software změní registry příslušného virtuálního stroje, restartuje stroj a sdělí mu, že běží ve virtuálním privilegovaném režimu

- Např. VMWare, Xen
- **Služby Hypervizora:**
    - Musí zastat všechny bázové funkce OS – plánování, přidělování, účtování zdrojů
    - Zajišťovat izolaci virtuálních systému
    - Zprostředkovávat privilegované služby OS
    - Umožnit vytvoření/zrušení virtuálního systému
- Virtuální systémy spolu mohou komunikovat pouze jako vzdálené entity přes síťové rozhraní
- Pokud dojde k poškození nebo infiltraci některého virtuálního systému, tento defekt nesmí ovlivnit ostatní systémy

Lecture_04 Služby OS

Rozhraní OS:

- **Pro uživatele:**
    - Poskytuje většina OS
    - Command-Line Interface (CLI), klasický UNIX
    - Graphics User Interface (GUI), okna
    - Batch, dávka
- **Pro procesy:**
    - Poskytující všechny OS
    - Komunikační styk procesy – OS
    - Analogie volání podprogramů
    - Implementace na bázi synchronního přerušení
    - Důvody – procesy i OS běží ve svých adresových prostorech, OS musí běžet v privilegovaném režimu CPU

Command-Line Interface (CLI):

- Command interpreter, shell
- Umožňuje přímé zadnání přímé zadání příkazu od uživatele
- Řešení může být zabudované přímo v shellu nebo shell volá samostatné programy (vč. využívaní služeb OS)

Graphics user Interface (GUI):

- user-friendly rozhraní
- okna, myš, klávesnice, obrazovka …
- ikony – soubory, programy, akce …
- MacOS X, Windows, GNOME …

Mnohé systémy obsahují CLI i GUI (MS Windows, Apple MacOS X, Solaris …)

Volání služeb OS:

- Podporuje ho rozhraní mezi běžícími procesy a OS
- Aplikační programy si služby OS zpřístupňují spíše přes API (Application Program Interface) vysoké úrovně něž přímým voláním systému

Typy poskytovaných služeb:

- **Řízení procesů:**
    - Zavedení programů do hlavní paměti a start jeho řešení procesu, ukončení procesu (normálně, s indikací chyby)
    - Fork(), exec(), wait(), abort()
- **Správa souborů:**
    - Manipulace s daty ve správě systému souborů
    - Schopnost číst, zapisovat, vytvářet a rušit soubory dat

        - Open(), close(), chmod(), link(), stat(), creat(), get(), put() …
- **Správa IO zařízení:**
        - Provedení I/O operace
        - Uživatelský program nesmí provádět I/O operace provádět přímo, OS musí poskytovat prostředky vykonávající I/O
        - Select(), read(), write() …
- **Údržba informací** – time(), gettimeofday() …
- **Detekce chyb a chybové řízení** – záruka za správnost výpočtu detekcí chyb
- **Komunikace** – realizace výměny komunikací v rámci jednoho počítače nebo mezi více počítače v síti pomocí sdílené paměti, nebo předáváním zpráv (socket(), send(), wait() …)

Vnitřní služby OS: slouží pro zabezpečení provozu, ne pro pomoc uživateli

- Přidělování prostředků (Resource Allocation) mezi vícesouběžně operujících uživatelů (procesů)
- Účtování (Accounting) – udržování přehledu o tom, kolik kterých zdrojů systému který uživatel používá (sběr statistik, účtování za služby …)
- Ochrana a bezpečnost (Protection) – kontrola všech přístupů ke zdrojům

Metody předávání parametrů mezi běžícím procesem a OS:

- V registrech (dostupné procesu i OS)
- V tabulce uložené v hlavní paměti (adresa tabulky umístěná v registru)
- V zásobníku (dostupný procesu i OS), instrukce push (store), pull (load)

Dynamic Link Layer (DLL) – dynamicky zaváděné knihovny při běhu programu

User mode services (NT Services) – rozšiřují funkcionalitu systému, snadno napadnutelné, množství trvale běžících služeb, analogie služeb implementovaných v jádru

Win32 API:

- Veřejně dostupné funkční rozhraní pro tvorbu aplikací
- Knihovní podprogramy (buď problémy řeší přímo, nebo pomocí volání služeb)

.NET = virtuální stroj, pro který je možné psát programy nezávislé na OS počítače, který tento virtuální stroj hostuje. Programy napsané v C# se kompilují na tzv. assemblies, které jsou just-in-time přeloženy do nativního kódu hostujícího systému

Lecture_05 Architektury OS

Modely architektur OS:

- **Monolitický OS** – pro každou volanou službu existuje jedna procedura, která ji řeší (MS-DOS)
- **Hierarchicky vrstvený OS:**
  - Řeší problém přílišné složitosti velkého systému dekompozicí (rozdělení velkého problému na několik menších)
  - Každá úroveň řeší konzistentní podmnožinu funkcí
  - Nižší vrstva nabízí služby vyšší, nemůže však po ní požadovat provedení služby
  - Entity v jedné vrstvě komunikují pomocí protokolů
  - Vrstvu je možno modifikovat bez toho, že by to ovlivnilo jiné
  - Jednoduchá konstrukce, ladění
  - Obtížnost definice hranic vrstev, vznikají kolize
- **OS s mikrojádrem:**
  - Jádro monolitického OS řeší všechny služby v režimu jádra
    - Režim jádra = použitelnost privilegovaných instrukcí …
  - V režimu jádra však nemusí být řešená plná funkčnost služeb
  - Cíl konceptu mikrojádra – minimalizovat rozsah kódu řešeného v režimu jádra:
    - Zbytek funkčnosti OS je řešený v režimu uživatele
    - V privilegovaném režimu se plní pouze vydání privilegované IO instrukce
    - Zbytek driveru se řeší jako proces v user mode
    - Krach driveru nezpůsobí krach celého systému
    - Pružnější, snadněji rozšířitelné rozhraní
    - Podpora distribuovanosti, OO přístupu
    - Zvýšená režie – výměna zpráv mezi procesy

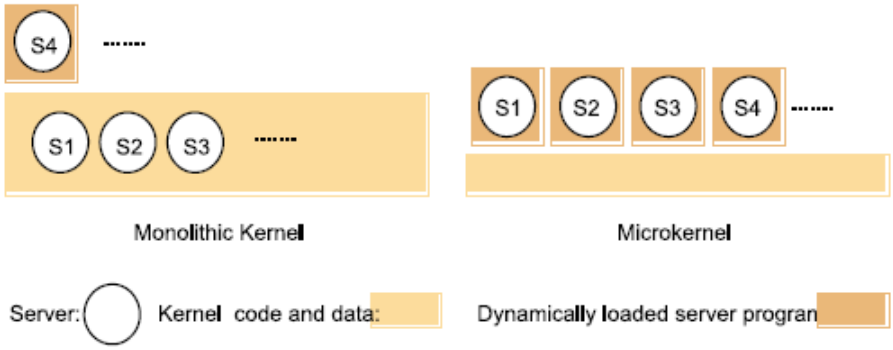
Monolitické jádro:

- Provádí všechny služby poskytované OS
- Mohutné, obtížně manipulovatelné a diferencovatelné
- Minimální modulace, vysoká provázanost funkcí
- Původní řešení OS Unix
- V rámci jádra mohou fungovat i serverovské procesy
- Služby jádra jsou řešeny sekvenčně

Mikrojádro:

- Zůstává v něm pouze práce s pamětí, s procesy a vlákny a zajišťuje meziprocesovou komunikaci (IPC)
- OS je kolekcí systémových procesů, které poskytují jednotlivé služby

- Minimum funkcí v privilegovaném režimu – jejich přesun do uživatelksé oblasti (drivery, služby souborového systému, virtualizace paměti …)
- Jeho funkcí je procesy separovat a umožnit jejich kooperaci
- Jádro se chová jako ústředna – systémové procesy separuje a propojuje
- Přináší větší spolehlivost, bezpečnost, rozšiřitelnost a přenositelnost
- Mezi uživatelskými moduly se komunikuje předáváním zpráv



- **Modulová architektura:**
  - Vytvoření jádra pomocí OOP
  - Komponenty jádra jsou samostatné jednotky – moduly
  - Moduly mezi sebou komunikují přes známá rozhraní
  - Každý modul se zavádí, když je třeba ho použít
  - **Jádro:**
    - **Mikrojádro Mach** – zodpovídá za volání vzdálených procedur (RPC, Remote Procedure Call), IPC, správu paměti, výměnu zpráv, dispečer
    - **Jádro BSD** – rozhraní na UNIXovské příkazy, podpora síťování, systému souborů
- **OS typu Klient-Server:**
  - **Thread-per-request** = pro řešení každého nového požadavku klienta se vytvoří nové vlákno – po splnění služby se samo zruší
  - **Thread-per-connection** = server vytvoří jedno vlákno pro každé spojení s jedním klientem a jeho požadavky řeší vždy sekvenčně
  - **Thread-per-object** = samostatné vlákno na každý spřístupňovaný objekt serveru

- **Virtuální stroje**
- **Síťový a distribuovaný OS**
    - **Síťový OS**
        - Unix, Windows
        - OS řídící 1 uzel schopným pracovat se vzdálenými zdroji sítě
    - **Distribuovaný OS**
        - Zatím komerční, ekonomicky efektivní rovině neexistuje
        - Celá síť se jeví jako jediný systém
        - Middleware = nadstavba síťového OS řešící neexistenci distribuovaných OS

Bootstraping: spuštění činnosti počítače zavedením jádra do operační paměti

- **Inicializace jádra:**
    - Zjištění skutečné velikosti operační paměti
    - Konfigurace HW komponent
    - Inicializace virtuální paměti (stránkováním), nastavení tabulek pro transformace LAP do FAP
    - Nastavení vektoru přerušení, povolení přerušování
    - Inicializace datových struktur jádra
    - Vytvoření iniciálního procesu pro jeho spuštění (initt, smss)

Vývoj OS Windows: MS-DOS, Windows 3.0, Windows NT 3.1, Windows 95, Windows 2000, Windows XP, Windows Vista, Windows 7

Lecture_06 Procesy a vlákna

Stavy procesu – nový, běžící, čekající, připravený, ukončený

Proces Control Block (PCB, TCB):

- Tabulka OS obsahující informace potřebné pro definici správy procesu
    - ID procesu
    - Stav procesu
    - Čítač instrukcí
    - Registry procesoru
    - Informace potřebné k plánování CPU
    - Informace potřebné pro správě paměti
    - Účtovací informace
    - Informace pro správu I/O
    - Seznam otevřených souborů
    - Vazební pole pro řazení PCB do front, seznamů

Context Switching (přepínání CPU mezi procesy):

- Procesor je přidělovaný procesu OS
- Akt přepínání mezi procesy = context switching
- OS po přepnutí musí uchovat stav procesu, kterému je procesor odebíraný a definovat nové výpočetní prostředí podle stavu přístě běžícího procesu
- Stavy procesů si procesor uchovává v tabulce procesů – seznam PCB
- Doba pro přepnutí procesu je režijní ztráta
- Přepínání procesů může kriticky ovlivňovat vákon systému
- Přepínání procesů je klíčový koncept multitaskingu
- **Asociované problémy:**
    - Proces čekající na signál od IO zařízení nesmí signál ztratit
    - Opakovaně použitelné zdroje musí být zpřístupňovány exkulzivně – problém synchronizace
    - Determinismus – výsledky procesů nesmí být závislé na frekvenci a okamžiky přepínání mezi procesy
    - Uváznutí …
- **Požadavky na OS z hlediska multitaskingu:**
    - Pravidla určující běžící procesy
    - V jádru existuje kód řídící přepínání – dispečer, plánovač
    - V jádru jsou implementovatelné mechanismy ochrany

Plánovače OS:

- **Krátkodobý plánovač (operační plánovač, dispečer)**
    - Vybírá proces, který poběží na uvolněném procesoru a přiděluje procesu procesor (CPU)
    - Vyvoláván velmi často, desítky – stovky ms, musí být rychlý
- **Střednědobý plánovač (taktický plánovač)**
    - Vybírá, který proces je nutné zařadit mezi odsunuté procesy (odebírá mu prostor ve FAP)

        - Vybírá, kterému odsunutému procesu lze opět přidělit prostor ve FAP
- **Dlouhodobý plánovač (strategický plánovač, job scheduler):**
    - Vybírá, který požadavek na výpočet lze zařadit mezi procesy
    - Je vyvoláván zřídkakdy
- **Plánovač CPU je aktivovaný a plánovací rozhodnutí vydává v okamžiku, kdy:**
    - běžící proces přechází do stavu čekající
    - běžící proces končí
    - uplynul čas, po který proces směl běžet (do stavu připravený)
    - nastala událost, na kterou proces čekal (do stavu připravený)
        - první dva případy se řeší voláním příslušné služby OS (synchronně, nepremtivní plánování)
        - poslední dva případy se iniciují na základě výskytu relevantního přerušení (asynchronně, premptivní plánování)
    - překládání procesoru z dispečera na proces představuje:
        - nastavení kontextu
        - přepnutí režimu procesoru z privilegovaného režimu na užívatelský
    - zpoždění způsobené dispečem

Přepnutí kontextů procesů: proces A >> OS >> proces B

- nejprve OS uchová stav původně běžícího procesu A
- v jádru OS se provede akce odpovídající příčině přerušení
- poté se nastaví stav nově běžícího procesu B

- přepnutí kontextu představuje režijní ztrátu
- doba přerušení závisí na rozsahu HW podpory v procesoru

Vytvoření procesu: každý OS musí mít mechanismy pro vytváření nových procesů

- **„na zelené louce“ :**
    - Do přidělené paměti se zavede text programu a data
    - Vytvoří se prázdná halda
    - Vytvoří se a inicializuje PCB
    - Proces se zpřístupní dispečerovi
- **Klon existujícího procesu:**
    - Běžící proces se pozastaví (vrátí mezi připravené)
    - Okopíruje se text, data, dynamické oblasti paměti, … a PCB
    - Nový proces se zpřístupní dispečerovi
    - Generující i generovaný proces je informován o své roli (rodič/potomek)
    - Rodičovský proces vytváří procesy potomky, které mohou vystupovat jako rodič a vytvářet své potomky
    - Tři způsoby sdílení zdrojů mezi rodiči a potomky (sdílení zdroje vlastněného rodičem, sdílet vyčleněnou podmnožinu zdrojů, jako samostatné procesy nesdílet)

- Voláním služby fork()

- Rodič a potomek mohou běžet současně
- Rodič může čekat na ukončení potomka
- Rodič se může bezprostředně ukončit (pokud skončí dřív něž potomek, potomek po ukončení se stane zombie)

Vytvoření procesu stylem Copy on Write:

- Nově vytvářený proces požaduje vytvoření nového prostředí běhu
- Tradiční forma vytvoření procesu uničovského tipu
- Iniciálně nový proces sdílí stránky s původním procesem
- Při zápisu do stránky novým procesem se vytvoří nový proces samostatné kopie modifikované stránky

Ukončení procesu:

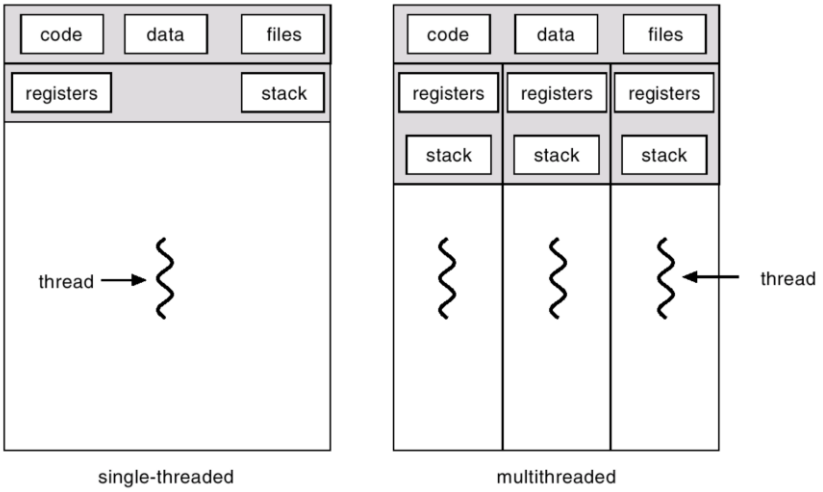
- Proces provede poslední příkaz programu a žádá OS o ukončení (exit)
    - Výstupní data procesu-potomka se mohou předat procesu-rodiči, který čeká v provádění služby wait
    - Zdroje končícího procesu se ukončují
- O ukončení procesu-potomka může požádat jeho rodič (abort), protože:
    - Potomek překročil jisté množství povolených zdrojů
    - Rodič ukončil svou existenci
- Proces může končit abnormálně (chybou programu, uplynutím doby běhu)

Formy kooperace procesů:

- Nezávislé procesy – nemohou se vzájemně ovlivnit
- Kooperující procesy – mohou ovlivňovat běh jiných procesů
- Přínosy koop. procesů – sdílení informací, urychlení výpočtů, modularita, pohdlí
- Paradigma kooperace: producent – konzument, klient – server …

Vlákno = sekvenční děj definovaný v procesu:

- Viditelné jen v rámci konkrétního procesu
- Může přistupovat ke zdrojům, které vlastní tento proces
- Vlákno se taky nachází ve stavech (running/ready/waiting), jeho kontext je uložen v TCB
- Všechna vlákna v procesu se řeší souběžně (př. textový editor)
- Umožňuje efektivní využití více jader procesoru
- Pro OS je jednotkou plánování, ne zdrojů – skupina vláken jednoho procesu sdílí proměnné
- Nemůžeme odkládat jedno vlákno – swapuje se celý proces
- **Výhody:**
    - Paralelismus, strukturalizace programu
    - Rychlejší vytvoření, ukončení i přepnutí kontextu jako při procesu
    - Sdílení zdrojů (pozor na komunikaci mezi vlákny)
    - Interakce (např. požadavky na server = vytvářejí se vlákna)



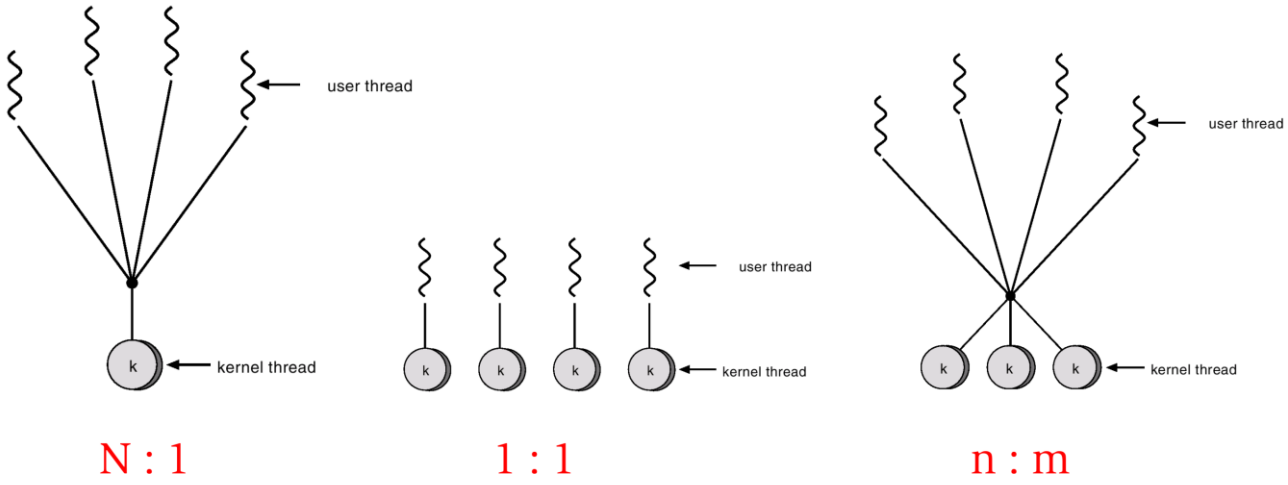
Vlákna na uživatelské úrovni (ULT – User-level Threads)

- Správa vláken se provádí prostřednictvím vláknové knihovny
- Obsahuje funkce pro vytváření/rušení vláken, předávání správ a dat mezi nimi, plánování jejich běhu a uchovávání a obnovu kontextu
- Výhody: jádro o jejich existenci neví – při přepínání vláken není nutné přerušovat, přepínat režim procesoru a volat služby jádra
- Aplikace si sama volí nejvhodnější algoritmus
- Nevýhody: při volání služeb se blokuje celý proces – jsou nedostupné všechny vlákna

Vlákna na úrovni jádra (KLT – Kernel-level Threads)

- Kompletně celou správu vláken podporuje AIP v jádru – nepoužívá se knihovna
- Výhody: jádro může současně plánovat běh více vláken jednoho procesu na více procesorech + k blokování dochází na úrovní vláken
- Prakticky se používá v kombinaci s ULT

Multivláknové modely



Lecture_07 Plánování

Klasifikace metod plánování:

- **Krátkodobé, operativní plánování**
  - Krátkodobý plánovač (operační paměť, dispečer)
  - Rozhodování, kterému procesu / vláknu OS přidělí CPU
  - Vyvoláván velmi často, desítky – stovky milisekund, musí být rychlý
  - Součást správy procesoru
- **Střednědobé, taktické plánování**
  - Střednědobý plánovač
  - Taktika využívání omezené kapacity FAP při multitaskingu, rozhodování, které procesy mohou využít prostor hlavní paměti
  - Náleží správě paměti
  - Řídící algoritmus techniky zvané swapping (odsunuté procesy)
- **Dlouhodobé, strategické plánování**
  - Dlouhodobý plánovač (job scheduler)
  - Zadání úlohy ke zpracování programu se stane procesem a předá se do fronty připravených procesů dispečerovi
  - V některých OS se mohou nové procesy zařazovat mezi potlačené

Kritéria kvality plánování:

- **Uživatelsky orientovaná kritéria**
  - **Doba obrátky (Turnaround time)**
    - Doba běhu + doba čekání na zdroje vč. CPU
    - Vhodná míra pro dávkové zpracování
    - Přirozená je snaha o minimalizaci doby obrátky
    - Normalizovaná doba obrátky = doba obrátky / doba běhu
  - **Doba reakce (Response time)**
    - Míra pro interaktivní systémy
    - Doba od zadání požadavku do doby očekávané reakce
    - Snaha o minimalizaci pro co největší komunitu uživatelů
  - **Časové limity (Deadlines)**
    - Pokud jsou zadané, plnění ostatních cílů se musí upozornit
    - Vhodné pro real-time systémy
  - **Časová proporcionalita**
- **Systémově orientovaná kritéria**
  - **Prostupnost (Throughput)**
    - Počet procesů dokončených za jednotku času
    - Snaha o maximalizaci propustnosti
  - **Využití CPU** – maximalizace ve víceuživatelských systémech
  - **Spravedlivost (Fairness)** – porovnatelné procesy musí získat porovnatelnou část procesoru
  - Prosazování priorit – prioritní procesy jsou přednostní
  - Vyrovnávání zátěže – udržování využitelnosti systémových zdrojů

Váha kritérií podle konkrétního systému:

- **Interaktivní:** minimalizace doby čekání, rychlá interakce, proporcionalita
  - Algoritmy RR (Round Robin), FSS (Fair Share Scheduler), + prioritní plánovaní
- **Dávkové systémy:** maximální propustnost, maximalizace využití CPU
  - Algoritmy FCFS, SJF (Shortest Job First), SRTF (Shortest Remaining Time First)

First-Come – First-Served:

- Nepremtivní, jednoduchá implementace
- Když se jako první dostaví velký proces a spousta malých za ním čeká – konvojový efekt

Shortest Job First:

- K definici procesu se doplní délka jeho následující CPU dávky
- **Nepreemptívní varianta:** když se proces dostane na řadu, nemůže být předběhnutý žádným jiným procesem, pokud svou dávku CPU nedokončí
- **Premptivní varianta:** když se ve fronte procesů objeví ready proces s délkou dávky CPU kratší než je doba zbývající k dokončení dávky právě běžícího procesu, nový proces předběhne právě běžící proces – tato varianta se nazývá SRTF
  - Optimální je, když je při plánovaní rozhoduje minimální průměrná doba čekání
- Běžící proces ale nemusí vždy využít celé svoje přidělené kvantum času (např. vyvolá IO operaci a skončí dřív)
- Délku jeho následující dávky je možné odhadnout z historie chování procesu
  - Musí být zaznamenané předešlé odhady délek dávek CPU o A taky jak je proces ve skutečnosti využil
  - **Použije se exponenciální průměrování:**
    - t_n … skutečná délka n-té dávky CPU
    - T_n+1 … odhad délky následující n+1 dávky
    - α … vliv historie, α je z intervalu [0, 1]
      - čím je menší, tím má historie na odhad menší vliv
      - čím je větší, tím více se respektuje krátká historie
    - $T_n+1 = α * t_n + (1 – α) * t_n$
    - Formuli můžeme rozvinout pomocí sumy až do T_0, které se zvolí jako konstanta
    - Když 1-α, α <= 1, každý term má na T_n+1 menší vplyv jako jeho předchůdce

Round Robin – cyklické plánovaní:

- Preemptivní plánovaní typu FCFS založené na sledovaní časových intervalů
- Každý proces dostává CPU cyklicky na malou jednotku času (100 ms) – **časové kvantum q**
- Po uplynutí doby q:
  - je běžící proces předběhnutý nejstarším procesem ve frontě
  - sám se zařadí na konec tento fronty
- jak je ve fronte n-procesů, každý získává 1/n-tinu doby (výkonu) CPU
- Žádný proces nečeká déle než (n-1) * q časových jednotek
  - Když je q velmi velké, plánovaní se blíží principu FCFS
  - Když je q velmi malé, CPU se věnuje převážně přepínaní kontextu procesů
    - Nech doba přepnutí kontextu je 0.01
    - Režijní straty při q = 12, 6 a 1 sú rovné 0.08%. 0.16% a 1%

Prioritní plánovaní:

- S každým procesem je spojené prioritní číslo – integer
- CPU se přiděluje procesoru s nejvyšší prioritou (typicky nejnižší číslo)
- Preemptivní varianta: když se ve fronte objeví proces s vyšší prioritou, než je priorita právě běžícího procesu – předběhne ho
- Procesy s velmi malou prioritou se ale **nikdy nemusí dostat na řadu – problém stárnuti**
- Řešeni: zřeni procesů – jejich priorita se po čase zvyšuje
- Např. Linux:
  - každému procesu přidělí jistý počet kreditů
  - při každém přerušení proces strácí 1 kredit – proces s 0 kredity se vzdává CPU
  - jak neexistuje žádný ready proces s kredity, všem procesům se nastaví kredity na hodnotu: kredity/2 + priorita
  - + real time algoritmus pro úlohy, kde je důležitější jejich priorita před spravedlivostí

Plánovaní s víceúrovňovými frontami:

- Přednostní fronta (foreground) – interaktivní – používá např. RR
- Fronta procesů na pozadí (background) – dávková – používá např. FCFS
- Musí se uplatnit vhodná politika, z které fronty se bude právě vybírat
  - prioritní výběr – např. dávková se obsluhuje, jen když je interaktivně prázdna
  - časové rezy – např. 80% času CPU pro interaktivní úlohy, 20% pro dávkové
- Možné i jiné rozděleni front – na více úrovní – procesy mezi nimi přesouvá plánovač

- Každá má svůj plánovací algoritmus a je definováno, kdy přeložit proces do fronty s vyšší / nižší preferencí

Fair Share Scheduler:

- Procesy se dělí do skupin, které dohromady využívají díl výkonu procesoru
- Plánovaní je prioritní a spravedlivé – priorita procesu klesá s růstem používaní procesoru daným procesem nebo skupinou, do které patří
- Skupině, které je přidělený větší díl výkonu procesoru klesá váha pomaleji
- Při exponenciálním průměrování se používá $\alpha = ½$

Plánovaní homogenního multiprocesoru (HMP):

- Jedna společná fronta pro všechny procesy nebo více prioritně uspořádaných front
- Plánovací politiky při multiprocesoru nemají až takový význam - HMP umožňuje realizovat **sdílení zátěže** (load sharing) – každý procesor může realizovat kterýkoli sled – proces
- **SMP** – jedna centrála fronta připravených sledů
    - každý procesor si sám vyhledává následující sled
    - předběhnuté sledy pravděpodobně nebudou pokračovat na stejném procesoru, proto není možné používat cache procesoru
    - sledy se nemůžou spustit ze společné fronty paralelně
- **Gangy** – technika plánovaní pro současný běh více sledů 1 procesu na vícerých procesorech
    - vhodné pro aplikace, jejichž výkon klesá, když se neřeší paralelně
    - sledy se musí vzájemně synchronizovat

Lecture_08 Synchornizace

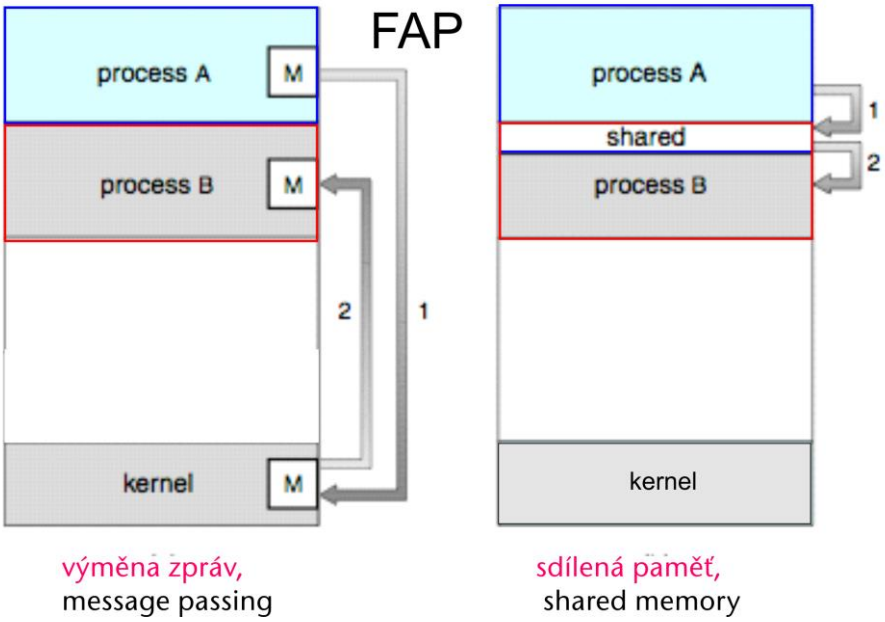
Potřeba IPC při aktivitách, které probíhají souběžně – multitasking, multithreading, multiprocessing, distribuované výpočty …

Souběžné aktivity mohou mezi sebou:

- **Soupeřit:**
  - Souběžné procesy se uchází o zdroje – procesor, FAP, periferie
  - Zdroje soupeřícím procesům přiděluje OS
  - OS efektivně řeší soupeřící procesy, aby se chybně neovlivňovaly
  - Soupeřící procesy se vzájemně neznají
  - Realizace musí být deterministická (bez bočních efektů)
- **Kooperovat**
  - Kooperující procesy sdílí jistou množinu zdrojů, vzájemně se znají
  - Kooperace se dosahuje implicitním sdílením zdrojů, nebo explicitní komunikací kooperujících procesů
  - Vlákna jednoho procesu obvykle kooperují
  - Procesy mohou kooperovat i soupeřit
  - Vlákna/procesy kooperují, aby mohly sdílet jisté zdroje, aby se mohly nezávislé akce řešit souběžně, podpora modulárnosti
- Souběžný přístup se mnohdy musí provádět neatomickými operacemi – udržování konzistence dat vyžaduje používání mechanismů zajišťujících deterministické provedení akcí kooperujících procesů

Úlohy související se souběžností:

- **Synchronizace běhu procesů** – čekání na událost
- **Komunikace mezi procesy** – výměna zpráv
  - Komunikace – způsob synchronizace, koordinace aktivit
  - Může dojít k uváznutí – každý proces čeká na zprávu od některého jiného procesu systému
  - Může dojít ke stárnutí – dva procesy si opakovaně posílají zprávy, zatímco třetí proces čeká na zprávu nekonečně dlouho
- **Sdílení prostředků** – problém synchronizace
  - Procesy používají a modifikují sdílená data
  - Operace musí být vzájemně výlučné
  - Operace musí být vzájemně výlučné s operacemi čtení
  - Operace čtení mohou být realizovány souběžně
  - Pro zabezpečení integrity dat se musí použít kritické sekce



Problém kritické sekce:

- Proces přistupuje ke sdílenému zdroji vzájemně výlučně (např. jednu proměnnou v daném okamžiku modifikuje maximálně jeden proces)
- v každém procesu se nachází kódu programu nazývaný kritická sekce, ve kterém proces vzájemně výlučně přistupuje ke sdíleným zdrojům
- je potřeba zajistit, že v kritické sekci sdružené s jistým zdrojem, se bude nacházet nejvýše jeden proces
- modelové prostředí pro hledání problémů kritické sekce
- proces v ní může setrvat konečnou dobu
- Podmínky vstupu do kritické sekce:
    - **Podmínka bezpečnosti (dosažení vzájemného vyloučení):**
        - Jestliže jeden proces provádí svoji kritickou sekci, žádný jiný proces nemůže provádět svojí kritickou sekci sdruženou se stejným zdrojem
    - **Podmínka živosti (trvalost postupu):**
        - Pokud žádný proces neprovádí svoji kritickou sekci s jistým zdrojem a existuje aspoň jeden proces, který si do ní přeje vstoupit, pak výběr procesu, který do takové kritické sekce vstoupí, se nesmí odkládat
    - **Podmínka spravedlivosti (Konečnost doby čekání):**
        - Po vydání žádosti procesu o vstup do kritické sekce a před uspokojením tohoto požadavku může být povolen vstup nejvýše n jiným procesů do sdružené kritické sekce

Možné řešení problému KS:

- **Softwarové řešení (přímé):**
  - Algoritmy, jejichž správnost se nespoléhá na žádné další služby
  - Používají standartní instrukční repertoár
  - S aktivním čekáním, busy waiting
  - Petersonovo řešení
  - **Nevýhody:**
    - nesplnění podmínky spravedlivosti
    - procesory žádající o vstup do KS metodou busy waiting spotřebovávají čas procesoru
- **HW řešení:**
  - Pomocí speciální instrukce procesoru
  - S aktivním čekáním
  - Používá se instrukce Test-and-Set Lock (TSL)
  - Na jiných procesorech XCHG
  - Výhody: vhodné i pro multiprocesory
  - **Nevýhody:**
    - Aktivní čekání
    - Možnost stárnutí
    - Možnost uváznutí
- **SW řešení zprostředkované (operačním systémem):**
  - Potřebné služby a datové struktury OS
  - S pasivním čekáním
  - Podpora volání služeb v programových systémech/jazycích
  - Semafory, monitory, volání zpráv

Semafory: široko použitelný synchronizační nástroj

- **Obecně:** hodnota semaforu indikuje počet jednotek zdroje, které jsou k dispozici
- Je to proměnná typu integer + 2 operace
  - **Operace čekej na událost (wait, P)** – čekej na zvednutí semaforu a zvednutý shoď = detekcí událostí se informace o události ztrácí shozením (acquire(P))
  - **Operace oznam událost (signal, V)** – zvednutí semaforu
- Operace jsou z hlediska přístupu atomické (release(V))

Semafor coby synchronizační nástroj:

- Obecný semafor – celočíselná hodnota z neomezovaného intervalu
- Binární semafor – celočíselná hodnota z intervalu < 0,1 >
- Implementovatelnost – binární semafor lze snadněji implementovat, obecný semafor lze implementovat semaforem binárním
- Vzájemné vyloučení pomocí semaforu

Implementace semaforu:

- Implementace jako služba z rozhraní služeb OS
- Dvě pomocné operace:
    - Block – běžící proces dá mezi procesy čekající na semafor
    - Wakeup(P) – přiřazuje čekající proces P mezi připravené procesy
- Implementace musí zaručit, že žádné dva procesy nemohou provádět operace acquire() a/nebo release() se stejným semaforem současně
- Splnění podmínek bezpečnosti, živosti a spravedlivosti je řešen v jádru OS
- Implementace semaforu zůstává problémem kritické sekce
- Na multiprocesoru se musí použít buďto SW přímé řešení, nebo se využijí speciální instrukce
- Chybné použití semaforu hroutí souhru všech kooperujících procesorů

Semafor coby synchronizátor:

- Signalizuje vykonání té samé události
- **Může dojít k uváznutí:**
    - Dva nebo více procesů neomezeně dlouho čekají na událost, kterou může generovat jen jeden z čekajících procesů
- **Může dojít ke stárnutí:**
    - Proces čekající ve frontě se nemůže dostat ven, protože jej předbíhají jiné procesy

Typy semaforů:

- Obecný semafor, semafor (čísla z neomezeného intervalu, musí se implementovat jako služba jádra
- Binární semafor, mutex – (celočíselná hodnota z intervalu < 0,1 >, snadno implementovatelná instrukcemi TSL/XCHG), Petersonovým algoritmem)
- Implementovatelnost

Klasické synchronizační úlohy:

- Producent-Konzument (předávání zpráv mezi 2 procesy)
- Čtenáři a písaři (souběžnost čtení a modifikace dat)
- Úloha o večeřících filozofech (zajímavý problém pro řešení uváznutí)

Producent – Konzument:

- Producent vyrábí určitá data a dává je do zásobníku o omezené velikosti
- Spotřebitel je zase ze zásobníku odebírá
- Producent nemůže přidávat data, pokud je zásobník plný, a spotřebitel nemůže odebírat data, pokud v zásobníku žádná data nejsou
- Problém se řeší tak, že se producent uspí, jakmile se zásobník naplní, spotřebitel ho pak probudí, jakmile odebere data ze zásobníku
- Stejně tak se postupuje u spotřebitele, když je zásobník prázdný. Při nevhodném řešení může nastat stav, kdy oba usnou a už se nikdy nevzbudí. Dojde tak k uváznutí

- Potřebujeme semafor pro vzájemné vyloučení operací s bankem bufferů – mutex
- Další dva semafory potřebujeme pro synchronizaci producenta a konzumenta počtem zpráv banku N bufferů – semafory pro indikaci naplněných a vyprázdněných bufferů

Čtenáři a písaři:

- **S prioritou čtenářů:**
  - První čtenář přistupující ke zdroji zablokuje písaře
  - Poslední ze čtenářů končící čtení zdroji uvolní přístup ke zdroji případně čekajícímu některému písaři
  - Písaři mohou stárnout
- **S prioritou písařů:**
  - První čtenář přistupující ke zdroji zablokuje všechny písaře
  - Poslední ze čtenářů končící čtení zdroje uvolní přístup ke zdroji, případně čekajícímu některému písaři
  - První písař žádající o vstup do kritické sekce dalším čtenářům zakáže přístup ke zdroji
  - Čtenáři mohou stárnout
- Operace zápisu do sdíleného zdroje musí být exklusivní, vzájemně vyloučené s jinou operací
- Operace čtení mohou čtený zdroj sdílet
- V jednom okamžiku může daný zdroj modifikovat pouze jeden proces-písař
- Jestliže písař modifikuje zdroj, nesmí ho současně číst žádný čtenář

Monitory:

- Synchronizační nástroj vysoké úrovně
- Umožňuje bezpečné sdílení abstraktního datového typu souběžnými procesy
- Provádění procedur P1, P2 … se implicitně vzájemně vylučují
- **Aby proces mohl čekat uvnitř monitoru, deklaruje se proměnná typu condition x:**
  - x.wait(); - proces, který vyvolá tuto operaci je potlačen do doby, kdy jiný proces provede operaci x.signal
  - x.signal(); - aktivuje právě jeden proces, který čeká na splnění podmínky x (pokud žádný nečeká, je její provedení prázdnou operací)

Mailbox: schránka pro předávání zpráv

- může být privátní pro dvojici komunikujících procesů
- může být sdílená procesy
- OS může dělat typovou kontrolu zpráv
- OS vytvoří mailbox na pokyn procesu
- Proces je vlastník schránky, může jí zrušit
- Schránka zaniká, když její vlastník končí

Port:

- Mailbox patřící jednomu přijímajícímu procesu a více procesům zasílajících zprávy
- Port vytváří přijímající proces
- V modelu K/S je přijímajícím - procesem server
- Port se ruší ukončením přijímajícího procesu
- Procesy, které se chtějí vzájemně vylučovat, si budou posílat zprávu ,,go“
    - Do KS vstupuje proces, který dostane zprávu ,,go“ jako první
- Producent-Konzument pomocí mailboxu
    - Producent umísťuje data do mailboxu myconsume, pokud získal zprávu z mailboxu mayproduce
    - Konzument získává data z mailboxu myconsume a když skončí, pošle zprávu do mailboxu myproduce

Úloha o 5 večeřících filozofech:

- 5 filozofů sedících u kulatého buď myslí, nebo jí
- Jí špagety jen 2 vidličkami
- Co se stane, když se všech 5 filozofů chopí např. levé vidličky?
- „No přece zemřou hladem, děti“
- Hledáme řešení zajišťující ochranu proti uváznutí

Řešení:

- Zrušení symetrie
    - Jeden filozof je levák, ostatní jsou praváci
    - Levák se liší pořadím získávání vidliček
- Strava se podává n filozofům v jídelně s n-1 židlemi
    - Vstup do jídelny hledá obecný semafor počátečně nastavený na kapacitu n-1
    - Řešení chránící jak před uváznutím, tak před stárnutím
- Filozof se smí uchopit vidličky, jen když jsou obě volné
    - Řešení monitorem
    - Uváznutí nehrozí, v monitoru je pouze jeden filozof

Lecture_09 Uváznutí

Charakteristika:

K uváznutí dojde, když současně platí 4 podmínky:

- Vzájemné vyloučení:
  - Sdílený zdroj může v jednom okamžiku používat více než jeden proces
- Požadavky se uplatňují postupně:
  - Proces vlastnící alespoň jeden zdroj čeká na získání dalšího zdroje, dosud vlastněného jiným procesem
- Nepřipouští se předbíhání:
  - Zdroj lze uvolnit pouze procesem, který ho v dané době vlastní
- Došlo k zacyklení požadavku:
  - Existuje taková posloupnost n čekajících procesů (P_0, P_1, P_2 ..) takových, že P_0 čeká na (uvolnění drženého zdroje) P_1, P_1 na P_2 … (P_{n-1}) čeká na P_0

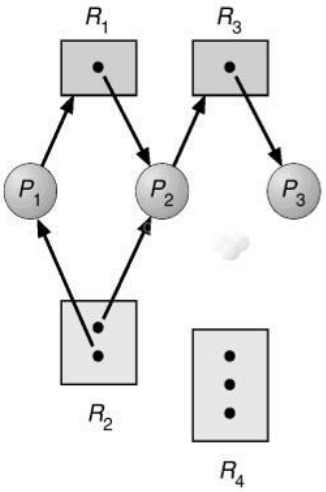
Použitý model: každý proces používá zdroj podle následujícího schématu

- Žádost o přidělení zdroje, Request (Get)
- Použití zdroje (po konečnou dobu)
- Uvolnění zdroje, Release

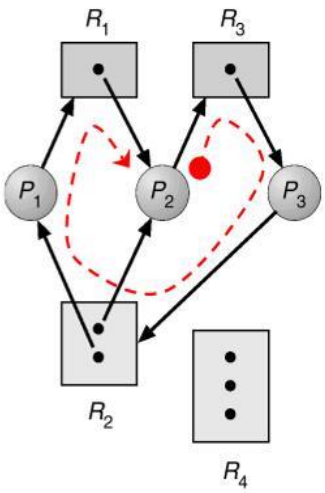
Graf přidělení zdrojů – Resource-Allocation Graph (RAG):

- Množina uzlů V a množina hran E
- V se dělí na množinu procesů a množinu zdrojů existujících v systému

RAG bez uváznutí



RAG s uváznutím



Cyklus RAG:

- Jestliže se v RAG nevyskytuje cyklus => nedošlo k uváznutí
- Jestliže se v RAG vyskytuje cyklus:
  - a existuje pouze jedna instance daného typu => došlo k uváznutí
  - a existuje více instancí zdroje daného typu => k uváznutí může dojít

Metody zvládnutí uváznutí:

- **Ochrana před uváznutím prevencí:**
  - Systém se nikdy nedostane do stavu uváznutí, ruší se platnost některé nutné podmínky
  - a prioritní eliminace některé nutné podmínky návrhovým rozhodnutím
- **Obcházení uváznutí:**
  - Detekce potenciální možnosti vzniku uváznutí a nepřipuštění takového stavu
  - Zamezuje se současné platnosti nutných podmínek rozhodnutími vydávaných za běhu
  - Prostředek se nepřidělí, hrozí-li uváznutí (hrozí stárnutí)
- **Detekce uváznutí a obnova po uváznutí:**
  - Detekce uváznutí a obnova stavu před uváznutím
- **Ignorováním hrozby uváznutí**
  - Řešení akceptovatelné většinou OS vč. Unixu

Ochrana před uváznutím prevencí:

- **Konzervativní politika** – omezuje se způsob provedení požadavku
- **Možnosti:**
  - Nepřímé metody – zneplatnění některé nutné podmínky
  - Přímá metoda – nepřipuštění platnosti postačující podmínky
- **Nepřímé metody:**
  - Ne vzájemná výlučnost
  - Ne postupné uplatňování požadavků
  - Připouští se předbíhání (preemtion)
- **Přímé metody:**
  - Zabránění zacyklení pořadí

Prevence uváznutí:

- **Ne vzájemné vyloučení**
  - Aplikovatelné pro procesy užívající zdroje, které lze sdílet
  - Neaplikovatelné pro používání opakovatelně dostupných zdrojů
  - Mnohdy se řeší virtualizací prostředků
  - Obecně neuplatitelné řešení
- **Ne postupná alokace**
  - Proces žádající nějaký zdroj nesmí nevlastnit žádný jiný zdroj
  - Proces musí požádat o všechny požadované zdroje a obdržet je dříve, než se spustí jeho běh, nebo se o ně smí žádat tehdy když žádné zdroje nevlastní

- **Ne předbíhání**
    - Jestliže proces držící nějaké zdroje požaduje přidělení dalšího zdroje, který mu nelze přidělit okamžitě (není volný), pak správce odebere procesu všechny jím držené zdroje v daném v tomto okamžiku
    - „odebrané zdroje“ zdroj zapíše do seznamu zdrojů, na které proces čeká
    - Proces bude obnoven tehdy, když bude moci získat jak původně držené zdroje, tak jím nově požadované zdroje
- **Zabránění zacyklení pořadí**
    - Zavede se úplné uspořádání typů zdrojů a každý proces požaduje prostředky v pořadí daném vzrůstajícím pořadím výčtů

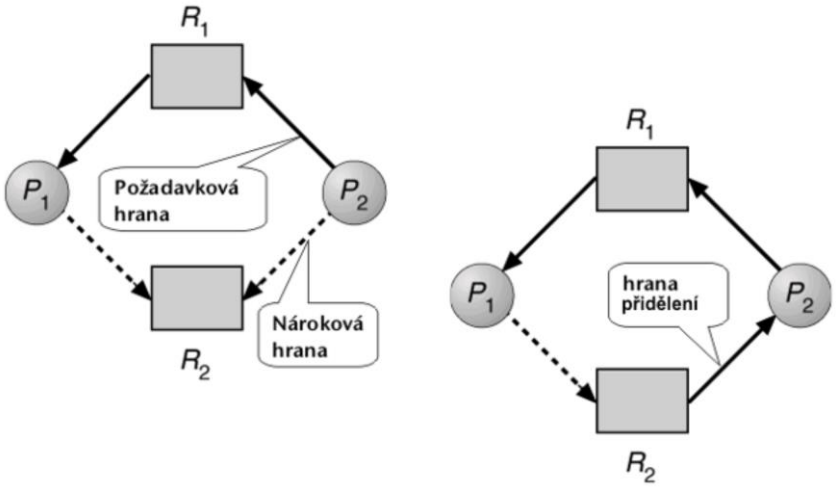
Obcházení uváznutí:

- Procesy mohou žádat o zdroje dynamicky
- Povoluje se existence všech tří nutných podmínek, zabraňuje se ale existenci postačující podmínky
- Dynamicky se testuje, zda splnění požadavku přidělení zdroje by mohlo vést k pozdějšímu uváznutí
- Požaduje se znalost budoucího chování procesů
- Řešením je tzv. Bankéřův algoritmus – řešení dynamickým rozhodováním
- Algoritmus řešící obcházení zkouší, zda přidělení zdroje nezpůsobí uváznutí = zda ponechá systém v bezpečném stavu (safe state). Když by přešel do unsafe state, přechod do stavu uváznutí je hrozbou.
- Posloupnost procesů je bezpečná, když požadavky každého procesu je možné uspokojit právě volnými zdroji a zdroji drženými ostatními procesy (čeká se)

Ilustrace obcházení pomocí RAG:

- nároková hrana $P_i \rightarrow R_j$ indikuje, že proces P_i může někdy v budoucnosti požadovat zdroj R_j (vede stejným směrem jako požadavková hrana, ale je zobrazená čárkovaně)
- konvertuje se na požadavkovou hranu v okamžiku, kdy proces P_i požádá o zdroj Rj
- když proces zdroj získá, požadavková hrana se mění na hranu přidělení
- když proces zdroj uvolní, hrana přidělení se mění zpátky na nárokovou hranu
- nesmí vzniknout cyklus

-



Bankéřův algoritmus:

- Procesy
  - Zákazníci, kteří si chtějí půjčovat peníze z banky, předem deklarují maximální výši úvěru
  - Úvěry v konečném čase zákazníci splácí
- Bankéř nesmí poskytnout úvěr, pokud si není jistý, že může uspokojit všechny požadavky všech svých zákazníku alespoň v jednom pořadí uspokojení
- Algoritmus obcházení uváznutí aplikovatelný i při násobnosti instancí zdrojů
- Každý proces musí maxima používaných zdrojů deklarovat ihned při svém vytvoření
- Proces požadující přidělení může být dát do stavu čekání
- Proces získané zdroje v konečném čase uvolní
- **Nikdy nedojde k uváznutí:**
  - Proces vznikne pouze tehdy, bude-li možno uspokojit všechny požadavky
  - Požadavek na přidělení se uspokojí jen tehdy, bude-li možno uspokojit všechny požadavky všech procesů

Detekce uváznutí:

- Umožňuje se, aby se systém uváznul
- Provede se algoritmus detekce uváznutí – hledání cyklů
- Aplikuje se plán obnovy

Kombinovaná ochrana před uváznutím, příklad:

- Kategorie zdrojů z hlediska pořadí přidělování
- Oblast na disku pro odkládání procesu
- Zdroje procesů – periferie typu disky, pásky, soubory
- Oblast hlavní / operační paměti
- I/O kanály

Lecture_10 Správa paměti

Program řídící běh procesu musí být umístěn v operační paměti – na jaké místo, o tom rozhoduje správa paměti. Správa paměti je předmětem činnosti OS.

Musí zajistit:

- **Možnost relokace programů** – jejich přemisťování v paměti
    - Programátor nemůže vědět, kde bude jeho program umístěný
    - Po swap-out a následném swap-in se pozice programu v paměti může změnit
- **Ochranu:** proces může zasahovat jen do míst, které mu patří
- **Logickou organizaci:** programy jsou moduly se vzájemně odlišnými vlastnostmi
        - Moduly s instrukcemi jsou většinou execute-only
        - Moduly s daty jsou buď read-only nebo read/write
        - Moduly mohou být privátní a veřejné
- **Možnost sdílení:** více procesů může sdílet společnou část paměti
- **Efektivitu**

Adresový prostor:

- Generické chápání – vymezení adres cílových objektů
- Je škála adres buněk hlavní paměti
- Procesy jsou řízeny programy a v ideálním případě se každý proces vč. OS realizuje svým vlastním AP
- Cílem je umožnit každé entitě typu proces používat svojí vlastní abstrakci adresového prostoru počítače (LAP)

Fyzická adresový prostor (FAP)

- Reálný adresový prostor
- Je dán škálou hlavní paměti, je lineární
- Adresa ve FAP = fyzická adresa (reálná adresa)
- Kapacita FAP je dána šířkou adresové sběrnice hlavní paměti

Logický adresový prostor (LAP)

- Virtuální adresový prostor
- Je vymezen šířkou a formou adresy ve strojovém jazyku
- Adresa v LAP = logická adresa (virtuální adresa)
- Kapacita a struktura LAP je dána šířkou a strukturou adresy v instrukci

Vázání LAP na FAP (= je bázový koncept správy paměti):

- **Statické**
  - **Při kompilaci** – vázání LAP na FAP entitou
    - Umístění programu ve FAP známe dopředu, před předkladem (vestavěné systémy)
    - Při kompilaci vzniká absolutní program – nepředpokládá se výměna dat v paměti
    - Při změně umístění programu ve FAP se musí překlad opakovat
  - **Při sestavování** – opět LAP = FAP
    - Umístění programu ve FAP je známo při sestavování programu
    - Překladač generuje sestavovatelný modul, při linkování se zobrazuje z LAP do FAP
    - Zaváděný modul může být absolutní, nebo přemístitelný
- **Dynamické**
  - Cílovým prostorem programování a sestavení je LAP
  - Program se zavede do FAP ve tvaru připraveném pro LAP
  - Vázání adres LAP na FAP se odkládá na běh instrukce
  - Musí být HW podpora – Memory Managment Unit, Data Adress Translation
  - **Nejjednodušší řešení:** práce s relokačním registrem CPU
    - Nastaví se počáteční hodnota, o kterou se bude posouvat logická adresa
    - Když je tato adresa použita jako ukazatel do hlavní paměti, obsah relokačního registru se připočítává k adresám interpretovaných instrukcí
    - Relokační registr je privilegovaný, dostupný jen OS
  - Podprogram – routine – se zavádí, až když je vyvolán
  - Dosahuje se lepšího využití prostoru ve FAP

Základní techniky správy paměti:

- Souvislé oblasti ve FAP pro běh procesů
- Formy – fixní oblastni, proměnné oblasti, překryvy
- Vesměs používají pro zobrazování LAP na FAP dynamickou relokaci bázovým registrem
- Pro zvýšení efektivity umožňují swapping
- Další možné formy – stránkování/segmentování

Přidělování oblasti procesům:

- Pro ochranu procesů uživatelů mezi sebou a OS lze použít schéma s relokačním mezním registrem
- Relokační registr = nejnižší (bázová) adresa ve FAP
- Mezní registr = nejvyšší dostupná adresa

Multitasking s více pevnými souvislými oblastmi FAP:

- Obvykle dávkové systémy, dnes už vesměs historie
- FAP sdílí OS a n procesů
- Pro vazbu LAP-FAP se použije relokace využívající bázový registr

Přidělováni souvislých oblastí různé délky: procesu se přidělí sekce volné paměti ve FAP, která uspokojí jeho požadavky. Při přidělení sekce – různé techniky: first-fit, best-fit, worst-fit. Vznikají ale úseky volného místa roztroušené ve FAP => nastává fragmentace.

Fragmentace:

- Vnější – souhrn volné paměti je dostatečný, ale není to souvislý celek
- Vnitřní – přidělená oblast paměti je větší než požadovaná, přebytek se nevyužije
- Přesun bloků – dodatečná režie, nutná relokace (MMU)

Rozptylování LAP do FAP:

- Přidělování jedné oblasti FAP procesu trpí fragmetnací
- Řešením je rozptylování LAP do FAP po více částech
- LAP procesu se zobrazuje do volných oblastí FAP po více částech

Stránkování:

- FAP se dělí na rámce pevné délky
- LAP se dělí na tzv. stránky (pevné délky, shodné s délkou rámců)
- Program délky n stránek se umístí do n rámců
- OS si udržuje seznam volných rámců
- Pořadí rámců ve FAP nesouvisí s pořadím stránek v LAP

Segmentace:

- Prostor ve FAP se přiděluje po oblastech proměnné délky
- LAP se dělí do dvou dimenzí – na oblasti zvané segmenty, je definován maximální počet segmentů a maximální délka segmentu
- Segmentům se přidělují oblasti ve FAP, skutečná délka segmentu být delší než maximální délka
- OS si udržuje seznam volných oblastí ve FAP
- Pořadí přidělených oblastí ve FAP nesouvisí s pořadím segmentů v LAP

Lecture_11 Virtuální paměť

Princip:

- Separace LAP a FAP
  - LAP – virtuální paměť, manipulovaná po stránkách
  - FAP – reálná paměť, spravovaná po rámcích
- ve FAP se musí nacházet alespoň stránky programu potřebné pro bezprostřední řízení procesu
- stránky procesu umístěné ve FAP nemusí být umístěné v sousedních rámcích FAP
- soudobé architektury – LAP je podstatně větší než FAP
- adresové prostory FAP lze sdílet
- **techniky implementace virtuální paměti:**
  - stránkování na žádost
  - segmentování na žádost

Důvody pro virtualizaci paměti stránkováním/segmentováním:

- překlad adres LAP do FAP je třeba dělat co nejpozději, dynamicky
- přidělování souvislých oblastí FAP procesu způsobuje vnější fragmentaci, stránkování tuto fragmentaci eliminuje
- stupeň multitaskingu je třeba udržovat co nejvyšší (ale rostou nároky procesů na FAP)
- některé procesy mají nároky přesahující dostupnou velikost FAP
- při běhu procesu se nemůže ve FAP najednou uchovat celý program a všechna potřebná data

Běh procesů ve virtuální paměti:

- respektuje se separace LAP a FAP
- velikost LAP je mnohem větší než velikost FAP
- OS při startu procesu zavede do FAP malou část programu (LAP) s místem, kam se předává řízení
- Poté dochází k dynamickým výměnám částí LAP na FAP
- Část procesů umístěnou ve FAP nazýváme rezidentní množinou
- Odkaz mimo rezidentní množinu způsobuje přerušení (výpadek stránky/semgentu)

Fetch policy – kdy stránku zavádět do FAP?

- Stránkování na žádost – stránka se zobrazuje do FAP při odkazu na ni, pokud již není ve FAP zobrazena
- Předstránkování – když sousední stránky v LAP se pravděpodobně budou používat čase po sobě, zavedeme do FAP dopředu několik sousedních stránek – i když je možná nevyužijeme, zabráníme tím výpadkům
- Vhodné pro inicializaci procesu

Placement Policy – kam stránku umístit ve FAP?

- Stránkování: libovolný rámec
- Segmentace: first-fit, best-fit, worst-fit …
- Stránkování segmenty: libovolný volný rámec

Replacement Policy – kterou stránku nahradit?

- FAP je plná a není dostupný volný rámec
- OS neví nic o běžících procesech – musí vybrat stránku, kterou odebere kterému procesu
- Lokální výběr, into = oběti se hledají mezi stránkami procesu, který vyvolal page fault
- Globální výběr, extra = oběti se hledají mezi stránkami všech procesů
- Pokud se do stráky nezapsalo, můžeme jí rychle vyhodit, její kopie je na disku
- Pokud se do stránky zapsalo, před vyhozením jí musíme zapsat na disk pomaleji

Algoritmy výběru obětí:

- Různé
- Ideální, optimální algoritmus = vyhodí se stránka, která bude nejdříve odkazovaní
- **OPT (optimální nahrzení)**
  - Oběť, musí platit = příští odkaz na oběť je nejpodstatnější odkaz ze všech následných odkazů ze stránky
  - Generuje nejmenší počet výpadků stránek
  - Obecně neimplementovatelný
- **FIFO (First In First Out)**
  - Oběť = stránka nejdéle zobrazená ve FAP
  - Často používané stránky jsou mnohdy nejstaršími
  - Jednoduchá implementace
- **LRU (Least Recently Used)**
  - Oběť = ve FAP nejdéle neodkazovaná stránka
  - Princip: pravděpodobnost odkazu na ni je malá
  - Kvalita je sice blízka optimální strategii, ale používá se zřídka, neefektivní implementace
- **„Druhá šance“ (aproximující LRU)**
  - Každá stránka má nejvýše jeden bit, když je označena za oběť, bit se nuluje a hledá se jiná oběť – vyhodí se až stránka s nulovým bitem
  - Vylepšená LRU – přidává se bit modifikace – dirty bit ,
- **LFU/MFU** – nejméně/nejvíce často referovaná stránky

Přidělování počtu rámců procesům:

- **Hlavní principy přidělování rámců procesům**
  - **Pevné přidělování** – stejný počet nebo proporciální počty
    - Pevný počet rámců, lokální náhrady:
      - Alokace počtu rámců podle aplikace
      - Důsledek – podhodnocení/nadhodnocení
    - Pevný počet rámců, globální náhrady
      - Nerealizovatelné, nesmyslné
    - Proměnný počet rámců, globální náhrady
      - Nejsnazší implementace
      - Nebezpečí výprasku Trashing
    - Proměnný počet rámců, lokální náhrady
      - Pracovní množiny
  - **Prioritní přidělování** – počet rámců přidělených procesu se v čase dynamicky mění
- **Hlavní politiky výběru oběti**
  - **Lokální výběr** – ze stránek procesu, který způsobil výpadek stránky
  - **Globální výběr** – ze stránek kteréhokoli procesu

Výprask, Trashing:

- Jestliže proces nemá v paměti dost stránek, generuje výsledky stránek rychle, tzn:
  - Dochází k nízkému využívání CPU aplikacemi
  - OS má dojem, že má zvýšit stupeň multitaskingu
- Trasing = procesor nedělá nic jiného, než výměny stránek

Model pracovní množiny (Working-Set):

- d = okno pracovní množiny (zvolen počet referencí stránek, např. 10 000)
- WSi = pracovní množina procesu Pi
- Pokud suma všech WSi je větší než kapacita reálné paměti, vzniká výprask
- Ochrana před vznikem výprasku (např. jeden proces se pozastaví)
- Příklad:
  - Nechť okno pracovní množiny d = 10 000 referencí
  - Nechť časovač přerušuje po 5000 časových jednotkách
  - V paměti se v PT udržují každé 2 bity (buffer a reference)
  - Kdykoli časovač přeruší, okopíruje se bit reference do bufferu a nastaví se na 0
  - Kdykoli se referencuje daná stránka, bit reference se mikroprogramově nastaví na 1
  - Jestli alespoň jeden z bitů = 1, stránka je WS

Výkon stránkování na žádost:

- p – pravděpodobnost výpadku stránky, pak
  EAT = (1 – p) * doba přístupu do paměti + p * (režie přerušení výpadkem stránky + doby výpisu stránky + doba načtení stránky + režie instrukce startu
- když p = 0, k výpadkům nedochází
- když p = 1, každá reference způsobí výpadek
- 0 <= p <= 1.0
- Př. EAT = (1 – p) * 200 + p * (8 000 000) = 7 999 800 * p + 200

Problém velikostí stránek:

- Velmi malý rozměr = ve stránkách nic není
- Čím je stránka menší = tím je menší vnitřní fragmentace a delší PT
- Pokud PT není ve FAP, způsobuje velké čerpání prostoru
- Umístění PT ve virtuální paměti způsobuje dvojnásobný počet přerušení
- Čím je dálka strany větší

Windows – správa paměti

- Jakmile vlákno stránkováním na žádost umístí stránku do volného rámce FAP, rámec se stává pohotovostním (standby)
- Jakmile vlákno stránku modifikuje v rámci FAP, rámec se stává modifikovaným rámcem
- Udržuje 5 seznamá rámců – volné, čisté, pohotovostní, modifikované, chybové

Techny umožněné virtualizaci

- Mapování LAP do FAP
- Princip tvorby procesů Copy-on-Write
- Vytváření potomka pomocí virtual memory fork (bfork())
- Odstraňování stránek z FAP

Memory-Mapped Files (MMF)

- Zobrazování diskových dat do FAP
- Se soubory se zachází stejně jako s texty programů
- Soubor se načítá pomocí stránkování na žádost

Lecture_12 Podsystém I/O, rozhraní souborových systémů

Čím se liší IO zařízení:

- Rychlostí přenosu dat
- Aplikací (disk, terminál)
- Složitost obsluhy
- Jednotkou přenosu
- Zobrazením dat
- Chybovostí

I/O instrukce:

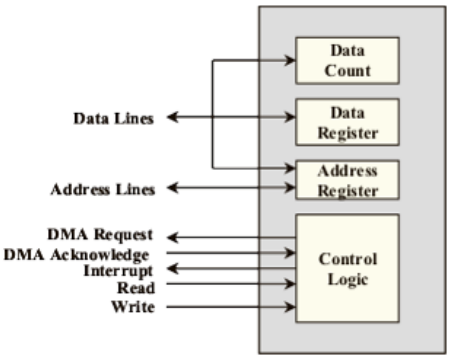
- Funkce čtení/zápis
- Zadávání příkazů do IO zařízení
- Odebíraní stavových informací od IO zařízení
- I/O adresy jsou mapované do FAP
- Samostatný IO adresový prostor

Techniky provádění I/O

- Programovaný (polling, busy waiting)
  - Vydání instrukce
  - Cyklické dotazování na stav přenosu až do zjištění konce, synchroní
- Programovaný I/O, řízený přerušením
  - Vydání instrukce
  - Paralelní běh s procesorem
  - I/O model oznamuje konec procesu přerušením, asynchronní

Základní koncepce ovládání:

- Direct Memory Access (DMA)
  - Zadání definice IO operace
  - Výměna bloků mezi FAP a I/O zařízením (kradení cyklů)
  - Přerušení generované DMA po ukončení přenosu bloku
  - Náhrada I/O při velkých přenosech dat
  - Požaduje se speciální DMA řadič
  - Při přenosu dat se obchází procesor, přímý tok zařízení – paměť



Soubor:

- Je pojmenovaná kolekce souvisejících informací na sekundárních/terciálních pamětí
- Je to logická paměťová jednotka zobrazovaná OS do fyzického paměťového zařízení

Systém souborů – cíle:

- Poskytovat služby správy dat požadované uživateli
- Zajišťovat validitu dat v souboru
- Optimalizovat výkon z hlediska systému i uživatele
- Poskytovat jednotnou podporu IO
- Řeší vhodné pojmenovávání souborů
- Umožňovat souběžnost operací

Soubor systémů:

- Je součástí OS
- Řeší správu sdílených zdrojů na vnějších pamětech
- Přemosťuje
- Nemusí být plně implementovatelný v jádru OS

Atributy souboru – jméno, typ, umístění, rozměr, ochrana, čas, datum, id vlastníka

Adresář – kolekce dat, je sám souborem