Přednáška 1

Struktura OS

- modul pro správu procesů (program, proces, vlákno, KS, synchr., deadlock, vyhladovění)
- modul pro správu paměti (virtuální paměť: stránkování, segmentace)
- modul pro správu I/O
- modul pro správu souborů
- síťování
- bezpečnost
- uživatelské rozhraní

OS je softwarová služba, jejíž úlohou je spravovat hardware a poskytovat k němu programům jednotné rozhraní

Jádro OS běží v privilegovaném režimu = všechny instrukce povoleny

Aplikace běží v uživatelském režimu = nemá přímý přístup k HW

Dělení - dle sdílení CPU = jednoprocesový, multiprocesový

- dle interakce = dávkový, interaktivní, OS reálného času = výsledek v omezeném čase
- velikost HW, míra distribuovanosti, počet uživatelů, podle fcí

Program = spustitelný kód, v binární podobě; **Proces** = instance běžícího programu

SW přerušení - instrukce **INT 0x80** nebo speciální (nová, rychlejší) instrukce: **sysenter**

Systémové volání - vyvolání služby, kterou sama nemůže vyvolat. Buď **přímo** (open, create), nebo přes **knihovní fce**. Př: do registru uložím č. služby; do dalšího parametry; provedu instrukci, která mě přepne do světa jádra; v jádře se zpracuje požadovaná služba; návrat

Druhy přerušení: - HW (vnější) = z I/O zařízení, **asynchronní** událost - např. stisk klávesy

- Vnitřní = vyvolá **CPU sám**! pokus o dělení nulou, výpadek stránky paměti!!
- SW = instrukce, **synchronní** = záměrně vyvolané, volání služeb OS z procesu

Vektor přerušení = vlastně **index do pole**, obsahující **adresu obslužné rutiny**, vykonané při daném typu přerušení.

Přijde-li přerušení: dokončena rozpracovaná strojová instrukce; na zásobník **uložena adresa následující instrukce**; z vektoru zjistí adresu podprogramu pro obsluhu přerušení; obsluha; instrukce návratu (RET, IRET) - vyzvedne za zásobníku návratovou adresu a na ní pokračuje; přerušení úloha nepozná, že proběhla obsluha přerušení;

Maskované přerušení - v době obsluhy přerušení lze zamaskovat méně důležitá místa; SW jsou nemaskovatelná

Architektura OS -> OS = jádro + systémové nástroje

Monolitické jádro - jádro je jeden funkční celek; jeden spustitelný soubor, uvnitř moduly, UNIX, Linux, MS DOS; Typickou součástí jádra je např. souborový systém.

Main procedure - vstupní bod jádra, na základě čísla služby zavolá servisní proceduru

Service procedure - odpovídá jednotlivým sys. voláním; volá pro splnění svých cílů pomocné

Utility procedures

Mikrojádro - malé jádro, oddělitelné části pracují jako samostatné procesy; **Model klient-server**; poskytuje pouze nejdůležitější nízkoúrovňové funkce(správa procesů, adresový prostor, komunikace mezi aresovými prostory); pouze mikrojádro běží v privilegovaném režimu(méně pádů systému)

Hybridní - kombinace(XP, Vista)

Přednáška 2 - koncepce, procesy, vlákna

IRQ = interrupt request - signál, kterým zařízení(časovač, klávesnice) žádá procesor o přerušení za účelem provedení důležitější akce; **IRQL** - priorita; **NMI** - nemaskovatelné přerušení, např. nezotavitelná chyba

Proces - instance běžícího programu, **Adresní prostor** - MMU, kód, data, zásobník; S procesem sdruženy registry = stavové informace; registry - **čítač instrukcí PC, ukazatel vrcholu zásobníku SP;**

Obecné registry - EAX, EBX,..., AX, BX, CX, ..., AL, AH; Uložení offsetu: **SP**, **BP**(práce se zásobníkem), **SI**(offset zdroje), **DI**(offset cíle); Segmentové: CS, DS, ES(extra), FS(volné), SS(zásobník)

Speciální reg: IP = offset vykonávané instrukce -> Proces (**CS:IP**), FLAG - IF, ZF, OF, DF, ...

Pseudoparalelní běh - v jednu chvíli aktivní **jeden** proces; po určité době pozastaven a spuštěn další

Proces nesmí mít vestavěné předpoklady o časování! Neběží nikdy stejně rychle.

Stavy procesu: Běžící, Připravený, Blokovaný

Tabulka procesů - každý proces v ní má záznam - **Process control block = PCB** -> obsahuje všechny informace potřebné pro opětovné spuštění přerušeného procesu; Pole správy **procesů**, správy **paměti**, správy **souboru**

Správa procesů - identifikátory - PID, UID; stavové inf. - registry; ukazatel na další instrukci - **PC**, **SP**, stav CPU - PSW; stav procesu; plánovací parametry; odkazy na rodiče a potomky; účtovací informace nastavení meziprocesové komunikace

Správa paměti - popis = ukazatel, velikost, práva; úsek s kódem, data - Halda, zásobník

Správa souborů - nastavení prostředí(aktuální adresář); otevřené soubory - čtení/zápis, pozice

Mezi CPU a pamětí je MMU: program pracuje s **virtuálními** adresami, MMU je převede na **fyzické** adresy;

fork() - vytvoří přesnou kopii rodičovského procesu - PID == 0;exit(), wait(); execve() - nahradí obsah paměti procesem spouštěným ze zadaného souboru

Vlákna v procesu sdílejí adresní prostor, vlákna mají soukromý čítač instrukcí, obsah registrů, soukromý zásobník

cobegin - coend … grafy pro znázornění paralelismu

Přednáška 3

Zombie = proces dokončil svůj kód, stále má záznam v PCB

Sirotek = jeho kód stále běží, ale skončil rodičovský proces

Plánování: **Krátkodobé**: CPU scheduling; **Střednědobé**: swap out; **Dlouhodobé**-job scheduling(dávky) Liší se frekvecní spuštění plánovače

Plánování Nepreemptivní = proces skončí, běžící -> blokovaný(I/O, semafor); **Preemptivní** = navíc přechod běžící->připravený(při uplynutí časového kvanta - přerušení i v nevhodnou chvíli)

Vlákna mohou být implementována - v jádře, v uživatelském režimu, kombinace

Vlákno má vlastní: zásobník(SP), registry, plánovací vlastnosti, množina pending a blokovaných signálů, data specifická pro vlákno. Všechna vlákna uvnitř procesu sdílejí stejný adresní prostor!

Synchronizace procesů: **časový souběh**(race condition) - sdílená paměť = čtení a zápis: např. dva procesy zvětšují asynchronně společnou proměnnou; **kritická sekce**: místo v programu, kde je prováděn přístup ke společným datům.

Pravidla: Vzájemné vyloučení - žádné dva procesy nesmějí být současně uvnitř své kritické sekce; Proces běžící mimo kritickou sekci nesmí blokovat jiné procesy; Žádný proces nesmí na vstup do své KS čekat nekonečně dlouho

Možnosti: **Zákaz přerušení**-vadí přeplánování procesů; **Aktivní čekání**: průběžné testování proměnné ve smyčce, snažíme se tomu vyhnout(plýtvání času CPU), použijeme, pokud předpokládáme krátké čekání(spin lock)**Spinlock** s **TSL**: HW podpora, otestuje hodnotu a nastaví paměťové místo v jedné neoddělitelné operaci - **Test and Set Lock**;

Semafor: primitivum, zjednodušuje komunikaci a synchronizaci procesů, **atomické** operace P(s) a V(s) - s = celočíselná hodnota;Struktura: hodnota, fronta; Pokud je zablokováno více procesů, vzbudí jen jeden - náhodně! **Mutex** = binární semafor

Producent-konzument 3 semafory - **e** - počet prázdných, **f** - počet plných, **m** - pro vzájemné vyloučení

Přednáška 4 -mutexy, monitory

Mutex: zajištění vzájemného vyloučení, chceme "spinlock" bez aktivního čekání, nepotřebujeme schopnost čítat; mutex = paměťový zámek; volání **yield** = dobrovolné vzdání se procesoru; Mutex s koncepcí vlastnictví **x** binární semafor -> odemknout mutex může jen stejné vlákno/proces, který ho zamkl!!

Monitor: na rozdíl od semaforu- jazyková konstrukce! Modul sdružuje data a procedury, který s nimi mohou manipulovat. V monitoru může být v jednu chvíli aktivní pouze **jeden** proces. Speciální typ proměnné nazývané **podmínka**-představují frontu procesů, které nad danou podmínkou čekají.

Operace: **wait:** volající bude pozastaven nad podm., **signal:** pokud existuje 1 a více pozastavených procesů nad podmínkou, reaktivuje jeden z pozastavených procesů.Pokud nikdo nespí, nedělá nic.

!Hoare: proces volající signal se pozastaví. **Hansen**: signal smí být uveden až jako poslední příkaz**!**

Java: čekající signál může běžet až poté, co proces volající signal opustí monitor; více podmínek + navíc **notify()**

Výhody monitoru: automaticky řeší vzájemné vyloučení, větší odolnost proti chybám programátora

Nevýhody monitoru: je to jen koncepce, překladač musí umět rozpoznat a implementovat