### Operační systém

* Základní programové vybavení, které se zavádí do paměti po spuštění PC -> skládá se z kernelu (jádra) a pomocných systémových knihoven/funkcí/nástrojů
* Rozšíření stroje (virtualizace) – pohled shora
* Správce prostředků – pohled zdola

### Role OS

* Správa prostředků (efektivní sdílení prostředků)
* Prostředí pro uživatele – OS nabízí abstrakce pro snadnější ovládání prostředků

### Typy

* Mainframové
  + Sálové, servery, velká kapacita IO operací…
* Dávkové
  + Maximalizace využití CPU a propustnosti
  + Minimalizace obratu
* Realtimové
  + Předvídatelnost
  + Dodržení termínů
  + Rychlý filesystem spolu s rychlou meziprocesovou komunikací
  + Některé procesy musí být stále v paměti, protože jejich swapování by degradovalo odezvu
* Interaktivní
  + Pro uživatele snadno pochopitelné/na každou akci přiměřená reakce
  + Minimalizace odezvy (čas mezi zadáním a odezvou)
* Víceprocesorové (SMP)
  + Zátěž rozložena na více procesorů
* Osobní (PC)
  + Windows, Linux, Mac
* Vestavěné
  + Zabudované OS v zařízení (smartphone, tablet…)
* Smartcard
* Víceúlohové
  + Poskytuje nástroje pro současný běh několika úloh na jednom a více procesorů

Služby OS

* Prostředí pro programování aplikací
* Prostředí pro spouštění aplikací
* Prostředí umožňující přístup k I/O zařízením a souborům
* Správa systémových prostředků
* Sledování systémových prostředků
* Detekce chyb
* Reakce na chyby

Systémová volání

* Nástroj využívaný aplikacemi pro volání systémových funkcí/služeb -> v případě monolitických jader -> unix
* Př.: vznik procesu, uknčení, otevření souboru, zavření, změna práva, signál, zjištění času…

Moderní koncepce OS obsahuje obsloužení

* Procesů
* Správy paměti
* Správy I/O
* Správy úložišť
* Systémových volání

Jádro OS

* Základní stavební kámen
* Ovládá I/O, přístup k souborům
* Přiděluje paměť procesům

Monolitické systém

* Všechny funkce systému v jednom, linux, solaris, BSD
* Nevýhody: velké jádro, nepřenositelné, monolitické aplikace
* Výhody: centralizace

Vrstvený systém

* Každá vrstva systému řeší dílčí problém, využívá primitiv předchozí vrstvy
* 5-operátor, 4-uživ. aplikace, 3-správa I/O, 2-komunikace mezi konzolí a procesy,2-správa paměti,0-alokace CPU
* Výhody: rychlost, snadnější provádění změn

Mikrokernel

* Jádro obsahuje pouze základní/důležité funkce (komunikace procesů, správa paměti, plánovač), zbytek v knihovnách, Windows 7, QNX, Symbian
* Výhody: bezpečnost, spolehlivost, přenositelnost
* Nevýhody: Snížení výkonu v důsledku meziprocesové komunikace

Virtuální stroj

* Aplikace vytvářející virtualizované prostředí mezi OS a HW, stírá rozdíly mezi HW platformami definicí další abstrakce
* Aplikační VM: VM dostupný na více platformách->aplikaci lze provozovat na více platformách-Java Virtual Machine
* Hardwarový VM (virtuální stroj): virtuální prostředí s odlišným OS na jednom PC a OS

Cíle návrhu OS

* Definice abstrakce
* Obsluha základních operací
* Zajištění izolace
* Správa HW

Z pohledu uživatele

* Snadno ovladatelný a zapamatovatelné ovládání
* Spolehlivost
* Rychlost
* Bezpečnost

Z pohledu systému

* Snadná implementace, přizpůsobení a údržba
* Spolehlivost
* Bezporuchovost

Důvody náročnosti na implementaci

1. Jedná se o komplexní, rozsáhlou, složitou aplikaci
2. Ošetření konkurence procesů v přístupu k systémovým prostředkům
3. Ochrana uživ. Dat
4. Sdílení dat a systémových prostředků
5. Životnost systému
6. Obecnost použití
7. Přenositelnost
8. Zpětná kompatibilita se starší verzí

Princip vývoje OS

* OS by měl dělat jen to, co má, stejně jako každá funkce by měla dělat jednu konkrétní věc
* Pokud není tolik důležitá, měla by být v knihovně, ne v jádru

Bezpečnost a ochrana OS

* Důvěrnost a integrita dat
* Dostupnost systému
* Kontrola a řízení přístupu k systémových prostředkům (oprávnění uživatelů)
* Autentizace – ověření identity (heslo…)
* Autorizace - ověření oprávnění pro provádění operací

Typy útoků na OS

* Zevnitř – co si uživatel natáhne jeho „blbostí“ nebo dírou v systému/aplikaci

Prevence – přidělování práv uživatelům, ACLka, firewall, antivir, školení o používání internetu/aplikací

Cpu, čítač, instrukce, registr, přerušovač, plánovač

Provádění instrukce

* Čítač obsahuje adresu příští instrukce
* CPU načte instrukci->dekóduje->provede
* Čítač se inkrementuje

Typy provádění instrukcí

* Sekvenční – jedna za druhou
* Multiprogramming – schopnost systému provádět více úloh/procesů/vláken zároveň na jednom (pseudo-paralelní běh více úloh->přepínání mezi nimi)
* Atomicita operace: požadavek na operaci takový, že musí být provedena naráz a nelze její provádění přerušit -> mutual exclusion

Přerušení procesů

* Preemptivní – o délce přidělení CPU rozhoduje plánovač->požadavek na přerušovač a časovač
* Nepreemptivní – o délce přidělení CPU si rozhoduje úloha sama, využití zásobníku dle priority procesů

Přerušovač

* Umožňuje efektivní využití CPU, podílí se na meziprocesové komunikaci
* Arbitr – řadič přerušení, který rozhoduje o pořadí přerušení úloh
* Fáze přerušení: přepnutí kontextu, uložení kontextu, obsluha ovladače, rozhodnutí plánovače

Časovač

* Generuje pravidelné přerušení v T -10ms

Sdílení času

* Běžící procesy se dělí o strojový čas CPU, dochází k přepínání kontextu (mezi nimi) bez zásahů a ovlivnění uživatele

Techniky programování I/O

* I/O:
* Programovaný I/O přenos:
  + Nevyužívá přerušení, CPU čeká na dokončení I/O operace a testuje stav I/O modulu
* I/O přenos s přerušením:
  + Během provádění může CPU pracovat na jiné operaci
  + Po dokončení I/O operace dojde přerušení
  + I/O přenos spotřebuje pouze část výkonu (procedura pro přerušení musí obsloužit přenos každého slova dat)

### DMA

* Výhody: CPU se neúčastní přenosu dat (pouze inicializace), následně může pracovat na něčem jiném

### Cache

* Rychlá vyrovnávací paměť dostupná jen při napájení
* Cíl: urychlení přístupu k datům na pomalých médiích (typicky HDD->RAM)

### Střední přístupová doba

* Doba od příchodu požadavku do jeho provedení

Tstřední přístupová doba=Tcache + (1-Hit Ratio) \* Toperační paměti

### Požadavky OS na HW

* Přerušovač
* Časovač
* CPU s podporou user/kernel (TRAP)
* CPU s podporou ochrany a virtualizace paměti

### Implementace procesu v OS

* Tabulka procesů (PCB):
  + Každý proces má své PID, tvoří strom, informace o stromu: pstree
  + Uchovává: adresový prostor, přidělené prostředky, stav a atributy

Program – seznam operací (recept na dort)

proces – provedení operací (pečení dortu), instance programu

* Vznik: při inicializaci systému, systémovým voláním, požadavkem uživatele, dávkou
* Ukončení: normální, kill, pád, error

### Stavy procesů

Třístavový model

* Připravený
* Běžící
* ukončený

Sedmistavový model

* Nový
* Připravený
* Běžící
* Odložený blokovaný
* Odložený připravený
* Blokovaný
* Ukončený

Pozn: z blokovaného se nemůže stát běžící a z připraveného blokovaný

(Linux)

ś stavy proces. . jak je vypisuje ps(1):

. R . p.ipraveny nebo b..ici (runnable/running)

. S . blokovany (sleep)

. Z . ukon.eny (defunct, zombie)

. T . pozastaveny nebo krokovany (stopped, traced)

. D . nep.eru.itelny spanek (uninterruptible sleep)

. X . mrtvy (dead) . nem.lo by se nikdy vyskytovat

. W . paging . jen p.ed jadrem 2.6

Fork() – nástroj pro tvorbu procesů v unixových systémech, vytváří strom, po zavolání se aktuální proces rozdvojí, PID rodiči > 0, PID potomka = 0, PID = -1 chyba,

Exec() – systémové volání, po jehož zavolání je proces nahrazen programem a předán dál jako parametr. PID zachováno včetně PCB. Využívá se při volání fork()

Exit() – slouží pro ukončení procesu a vrácení všech přidělených systémových zdrojů

Ps – vypíše seznam běžících procesů, má další parametry, -f full access, -e all processes

Adresář /proc – virtuální filesystém pro meziprocesovou komunikaci

Posixové signály

* Komunikační meziprocesový standard, který měl zajistit přenositelnost a nezávislost na platformě
* Definuje API pro programátory, ale i pro uživatele (seznam příkazů do cmd)
* Posix. Signál: jednoduchá zpráva ((číslo)) poslaná všem procesům
* Implicitně odesílány OS
* Explicitně při kill
* SW obdoba HW přerušení
* Reakce na signál: implicitní akce, ignorování, obsluha signálu na základě uživ. Def. funkce

### Vlákna

* Odlehčený proces (skupina vláken tvoří proces)
* Snižuje režii OS při změně kontextu při multitaskingu
* mají vlastní zásobník, nesdílejí čítač, registry, stav
* S procesem a ostatními vlákny sdílí paměť a struktury->zjednodušení mezivláknové komunikace->nutné řešit vzájemný běh (race condition)
* Systémové zdroje přidělovány procesu, on sám si je obhospodařuje
* Multithreading: současný běh několik vláken
* Výhody: snadná správa (vznik/zánik), užitečné na SMP, kde pomocí vláken dosáhneme rovnoměrného zařízení systému
* Nevýhody: globální proměnné nutné alokovat pro každé vlákno, řešit obsluhu signálů vláken

Posixová vlákna

* Vlákna vyhovující standardu posix (pthreads.h)
* 2 typy – starší (tzv. linux threads), kdy se vlákno tváří jako proces  
  nativ posix threads – implementace vlákna na úrovni jádra

Implementace vláken

* Bez podpory OS:
  + Výhody: lepší režie, není nutný přechod do kernelu, lze přizpůsobit plánovač aplikaci
  + Nevýhody: nutnost převádět blokovaná volání na neblokovaná a ošetřovat page-fault
* Implementace v OS:
  + Výhody: není nutné převádět volání na neblokovaná, lze provádět proces, jehož vlákno způsobilo page-fault (přístup do nenamalované oblasti paměti)
  + Nevýhody: horší režie (přechod do kernelu), pevná strategie plánovače

Mutex

* Zámek procesu/vlákna, který nám zajišťuje atomicitu operace a vzájemné vyloučení
* Fast mutex (zamkutí jednou)
* Recurzive mutex (zamktnuí „na více západů“)

Podmínkové proměnná

* Nezaručuje samo o sobě vzájemné vyloučení, nutné přistupovat pomocí mutexu
* Místo v paměti, které zadržuje vlákna do splnění určité podmínky (bariéra čeká na počet vláken)

Plánovač

* Systémový proces, který rozhoduje o přidělení času CPU dalším procesům
* Založen na preemptivním přístupu
* Cíl plánování:
  + Minimalizace odezvy
  + Maximalizace propustnosti systému a vytížení CPU
  + Spravedlnost
  + Dodržení politik
* Typy plánování:
  + Krátkodobé – řeší spuštění/ukončení úloh
  + Středně dobé – řešení uložení/vyndání úloh ze swapu
  + Dlouhodobé – řeší, který proces dostane CPU
* Kritéria plánování:
  + Ohled na uživatele – minimalizace odezvy
  + Ohled na požadavky systému – efektivní využití zdrojů
  + Ohled na výkon

Plánovací algoritmy

* FCFS – FIFO, kdo dřív přijde, ten dřív mele
* SJF – nejkratší úloha první
* SRTN – nejkratší zbývající úloha první
* RR – použití časově, procesu přiděleno časové kvantum
* Prioritní plánování – prioritní fronta – hrozí vyhladovění procesů s nízkou prioritou
* LS – přidělení tiketů a následné losování

Příkaz at – provedení úlohy jednou v naplánovaný čas

Příkaz cron – provádění úlohy v pravidelně naplánovaný čas za pomoci cron démona

Plánování na SMP

* Plánování
  + sdílení zátěže mezi jednotlivé procesory tak, aby nezůstal žádný CPU nevyužit, není potřeba centrální plánovač
    - nevýhody: globální fronta vyžaduje výlučný přístup, nepravděpodobné, že vlákno bude pokračovat ve stejném CPU
  + Skupinové plánování – související vlákna jsou zpracovávána na rozdílných procesorech – důležité pro kritické aplikace, kde by mohl výkon klesnout, pokud by jedna část neběžela
  + Proces pevně přiřazen procesu – vlákno má definovaný CPU, binární přepínání procesů, nevyužití všech CPU
  + dynamické přiřazování – počet vláken se v průběhu mění a OS dynamicky upravuje zatížení

Alokace paměti

* vymezení prostoru paměti pro běh procesu
* realokace: přemístění procesu v paměti na jinou adresu (proto se nesmí pracovat s absolutními adresami)->využití logického afresování
* typy alokace
  + statická – alokace před spuštěním aplikace na základě informací z překladače, globální proměnné pouze staticky
  + dynamická – alokace za běhu paměti  
    na stacku-automaticky  
    na heapu-řídí uživatel pomocí run-time pocedur(operace za běhu)

Logické adresování

* adresace vzhledem k počátku oblasti
* nutnost převádět logické adresy na fyzické

Dělení paměti

* stejná velikost – každý proces dostane stejnou velikost paměti, dochází k vnitřní fragmentaci (proces nevyužije celou přidělenou velikost)
* dynamická velikost – proces dostane jen tolik, kolik potřebuje, redukuje vnitřní fragmentaci

Fragmentace

* nesouvislé uložení dat způsobuje ztrátu výkonu
* vnitřní fragmentace: plýtvání místem uvnitř alokovaných bloků. Viz. Dělení paměti na stejné bloky
* vnější fragmentace: plýtvání místem mezi alokovanými oblastmi. Odstraníme defragmnteací

SWAP

* při nedostatku fyzické paměti dochází k odkládání obsahu (virtualizaci paměti) na pomalou paměť (HDD) - swap
* trashing-odložení obsahu, který vzápětí potřebuji (dochází ke snížení výkonu)  
  - zabránění: zvětšení fyzické paměti, snížení počtu běžících procesů a přizpůsobení velikosti SWAPU
* motivace virtualizace: obejití limitu fyzické operační paměti->možnost spouštět náročnější aplikace, možnost využití multitaskingu
* lokalita odkazu: proces má tendence přistupovat k okolí svého adresového prostoru, kam přistupoval nedávno->lze tak předpokládat, kam budu přistupovat v budoucnu->zvýšení efektivity virtualizace paměti a zabránění trashingu

Fyzická adresa – adresa běžící po sběrnici

Logická adresa – adresa, kterou má k dispozici proces

Efektivní adresa – adresa vznikající modifikací operandu segmentovanými registry

Postup překladu adres – operand->efektivní->logická->fyzická

Stránkování paměti

* FOP se dělí na rámce o velikosti 512-64kB
* logická adresa se skládá z čísla stránky a offsetu

Volba velikosti stránky

* pevná alokace – při alokaci je zvolen pevný počet rámců, rovnoměrně rozdělen mezi procesy
* proměnná alokace – počet rámců se dynamicky mění v závislosti na page-fault

Sdílení paměti

* není potřeba a je nežádoucí mít v paměti stejná data->stránky různých procesů mohou sdílet rámec
* lze sdílet i stránky obsahující stránkové tabulky

TLB – speciální cache, která uchovává poslední položky ve stránkové tabulce

Segmentace paměti

* přidělená paměť procesu je rozdělena jednotlivé oblasti (pro program, knihovny, stack...)
* procesu je přidělen blok, který si obhospodařuje proces sám
* výhoda oproti stránkování je minimalizace počtu výpadků
* zjednodušená práce s datovými strukturami a sdílení mezi procesy

Kombinace stránkování a segmentace

* rozdělíme segmenty na stránky
* odstraňuje vnější fragmentaci
* proces má jednu segmentovou tabulku a stránkovou pro každý segment tabulky

Registry CPU

* nastavují se při zavádění procesu
* bázový – počáteční adresa
* mezní – koncová adresa
* adresní – logická adresa místa v rámci procesu

Latence – doba od příchodu požadavku na přerušení po okamžik, kdy se začne aplikace provádět

Doba odezvy – čas, ve kterém systém musí reagovat na událost

Vazba SMP systémů

* volná – každý CPU má vlastní RAM, I/O subsystém atp
* těsná – sdílejí RAM, řízeno jedním OS

Symetrie SMP systémů

* symetrické – CPU jsou shodné a proces může běžet na jakémkoli z nich
* asymetrické – specializované CPU, řízeno jedním centrálním CPU

Granulita úlohy

* úlohu lze rozdělit na několik dílčích úseků, a proces lze provádět na více CPU
* hrubě granulovaná úloha – vhodné pro kooperující CPU
* jemně granulovaná úloha – častá vzájemná komunikace a synchronizace
* čím je vazba volnější, tím větší časové ztráty nastávají při synchronizaci

Paralelismus

* nezávislý – na každém CPU běží jiné procesy
* velmi hrubý – distribuované zpracování představují jedno výpočetní prostředí
* hrubý – provádění více procesů na jednom procesoru(multiprocessing)
* střední – paralelní zpracování nebo multitasking v rámci aplikace->jedna aplikace tvořena několika vlákny

Kritická sekce

* část kódu, kde se manipuluje se sdíleným prostředkem
* procesy spolu bojují o systémové prostředky
* komunikují mezi s sebou pomocí zpráv
* synchronizace procesů probíhá pomocí semaforu

Problémy konkurence

* vzájemné vyloučení (mutual exclusion) – v každém okamžiku může mít přístup ke sdílenému prostředku pouze jeden proces
* smrtící objetí – jeden proces čeká na uvolnění prostředku druhého procesu a opačně
* vyhladovění (starvation) – nekončící čekání procesu na získání prostředků

Struktura KS

* vstupní sekce
* kritická sekce
* výstupní sekce
* zbytková sekce

Typy řešení

* SW – proměnná free, vlajky (flag), petersnův algoritmus
* HW – využívání instrukcí CPU
* Řešení na úrovni OS
* Řešení pomocí nástrojů programovacího jazyka (konkurenční programování)

Semafor

* Synchronizační nástroj procesů
* Prostředek OS, datová struktura
* Nevyžaduje aktivní čekání
* Dokud je ks blokován, je proces zařazen do fronty->následně je vybrán další proces
* Operace: init, wait, signal
* Obědvající filozofové: 5filozofů u oběda, 5 vidliček, pro jídlo nutné 2. Když zvedne každý vidličku, nikdo se nenají.

Předávání zpráv

* Komunikační prostředek procesů, skrz mailbox a porty
* Mailbox – soukromý pro dva komunikující procesy nebo pro více procesů
* Port- schránka vlastněná jedním příjemcem
* Systémové volání (send, recieve)

Monitor

* Konstrukce vyšších programovacích jazyků, obdoba semaforu OS
* V monitoru může být rovněž jen jeden proces
* Typický problém: producenti, konzumenti (pro vyšší efektivitu vhodné zavést mezisklad)
* Kapacitu skladu můžeš ošetřit buď pomocí posílání zpráv nebo semaforu.   
  Pokud zvolíš posílání zpráv, tak se to provádí tak, že naposíláš prázdné zprávy do mailbox, které producent postupně vybírá ... dokud tam nějaké zprávy jsou - sklad má místo, pokud tam žádné nejsou, kapacita byla vyčerpána. Konzument pak zase musí po vybrání položky ze skladu posílat prázdné zprávy do mailboxu, aby dal vědět, že se uvolnilo místo. Tenhle příklad je zrovna i v přednáškách u zpráv (někde u konce, jak je řešení producent/konzument pomocí zpráv).   
  Dále to můžeš řešit i pomocí semaforu, nainicializuješ ho na kapacitu skladu.

Bariéra

* Zarážka v programu
* Čeká na daný počet vláken před vykonáním další akce

Podmínkové proměnné

* Obdoba bariéry, ale čeká na splnění podmínky

Roura

* Prostředek pro komunikaci procesů
* Pracuje je s nimi jako se soubory
* Jednosměrná komunikace, FIFO (fronta)
* Pojmenovaná roura: využíváno pro příbuzné procesy, trvalá, nutné dealokovat
* Nepojmenovaná: není nutné dealokovat, zanikají automaticky při ukončení procesu

Socket

* Obousměrná komunikace
* Nejen pro lokální, ale také vzdálenou komunikaci
* Klient/server

Dělení disku

* Instalace více OS
* Obcházení omezení filesystému
* Více filesystémů na jednom disku

MBR – hlavní spouštěcí záznam, obsahuje zavaděč, prvních 512 bajtů

MPT – tabulka oddílů

Filesystém – určuje způsob, jakým jsou data na disku uložena  
souvislá alokace, spojovaná alokace, indexová alokace

Soubor

* Posloupnost uložených bajtů
* Má atributy, vlastníka, timestamp, práva pro přístup
* Hierarchická organizace: kořenový adresář obsahuje další adresáře, podadresáře (omezení počtu úrovní)  
  - cesta k souboru relativní vzhledem k adresáři nebo absolutní  
  -adresář: organizační jednotka/struktura seskupující soubory a adresáře

Vzdálený OS

* Běží někde na síti a přistupujeme k němu pomocí klienta
* Spojení zajištěno: autentizací, autorizací a VPNkou
* Typy připojení: dial-ul  
  on-demand  
  SSL VPN – SSL připojení, pro přístup k webovým aplikacím, kde nelze použít IPSEC nebo klasického VPN klienta  
  VPN – šifrovaný tunel uvnitř stávajícího spojení
* Telnet: nezabezpečené spojení ke vzdálenému PC/routeru atp. (port 23)
* SSH: zabezpečené kanál, přenos po SFTP, (port 22)
* Tvorba hesla: nemělo by obsahovat jméno uživatele, posloupnost znaků/čísel, alespoň 8 znaků, používat velká písmena, speciální znaky, volit jiná hesla pro každou službu