1. Anatomie OS

Co je to operační systém? Neexistuje nějaká přesná a jednotná definice. Je to software, který nám umožňuje používat hardware. Operační systém je takový software, který způsobuje, že další software se nám píše lépe. Operační systém je (téměř) nejvíce nízkoúrovňový software.

K operačnímu systému nepatří firmware.

Pod operačním systémem se tedy nachází základní firmware (BIOS, UEFI) a hardware, nad ním je poté aplikační vrstva. OS umožňuje komunikaci mezi těmito vrstvami, tedy umožňuje softwaru nad ním komunikoivat s a využívat hardware pod ním.

Operační systém komunikuje s uživatelem, spravuje data (pomocí souborového systému), zajišťuje bezpečný běh (oddělování a kontrola procesů atd.)

Jaké má OS komponenty? Jádro, systémové knihovny, systémoví démoni (spící programy, spouštěné událostí, obvykle programy, které nechcete mít přímo v jádře, obykle je kontroluje nějaký hlavní démon, základní funkce init, démoni se dělí do dvou základních typů: údržbové, třeba init; a démoni, kteří poskytují nějaké služby, od systémových knihoven se démoni liší tím, že jsou to běžící programy, které ale také poskytují služby jiným programům), uživatelské rozhraní, systémové utility (umožňují interakci s OS na úrovni samotného systému, textový editor, ls, dir, obecně příkazy shellu atd.). V jaké podobě jsou ovšem tyto komponenty v OS zastoupeny je odlišné v různých operačních systémech, nicméně v nějaké základní podobě by se měly vyskytovat ve všech OS.

Uživatelské rozhraní: Velmi různorodé, úkolem je zprostředkovat interakci člověka a počítače (hardwaru). Dříve komunikace probíhala pomocí děrných štítků, dnes pomocí myši a klávesnice skrz nějaké grafické rozhraní.

Přímo s OS často dodávány další aplikace a programy, které ale nejsou součástí OS, např přehrávač hudby, internetový prohlížeš atd.

Další volitelné komponenty, u kterých je těžké určit, zda patří do OS nebo ne, například interpret Pythonu v UNIXu by se dal považovat za OS, případně překladač C.

Rozhraní

Programátorské rozhraní: Existuje mezi systémovými knihovnami a utilitami. Směrem dolů zprostředkovává kernel systémová volání. Směrem nahoru se používají API, které zprostředkovávají systémové knihovny.

Meziprocesová komunikace: Poskytuje komunikaci mezi procesory, typicky komunikace po síti. Často obsahuje vlastní céčkové API. Z pohledu programátora často není poznat, jestli funkce komunikuje s jádrem, nebo s nějakým jiným programem.

Přenositelnost: Jedna z velmi důležitých vlastností OS. Bez OS bychom museli programovat aplikace pouze pro konkrétní hardware a měnit kód pokaždé, když dojde k jeho změně. Zde velmi pomáhá OS, který nám dává rozhraní, které nám umožní předstírat, že je hardware vždy stejný (dává nám pocit uniformního hardwaru). Nicméně i u samotného OS je dobré, aby byl přenositelný i na jiný hardware. Operační systémy jsou proto rozděny na části, které vyloženě nevyžadují pracovat se syrovým hardwarem a které jsou na něm tedy víceméně nezávislé, a na části, které je potřeba naprogramovat na různé typy hardwaru.

Rozlšujeme dva termíny: instrukční sada a platforma. Všechny klasické počítače, které vzešly z původního IBM, mají obecně stejnou platformu. Opačný situace je u ARM, například v mobilech, kde existuje velké množství platforem. Stejné procesory tak mají různé platformy, které vyžadují jiné operační systémy.

OS napsaný v nějakém vyšším programovacím jazyce jeobvykle možné přeložit pro množství procesorů a platforem. Na jedné straně OS, které sedají přeložit pro velkou spoustu procesorů a platforem, na druhé straně speciálně zaměřené systémy pro jedno konkrétní zařízení (satelity).

OS umožňuje vysokou abstrakci nad hardwarem, čím více abstrakce, tím pomalejší systém, nicméně se zrychlujícím se hardwarem to není takový problém, benefity převažují.

Klasifikace OS:

Obecně použitelné OS: Vhodné pro většinu situací, Flexibilní, komplexní a velké (miliony řádků kódu). Běží jak na serverech, tak na osobních PC. Zjednodušené verze běží na mobilech. Podpora velkého množství hardwaru.

Speciální OS: Embedded OS – OS pro specifická zařízení, často přímo součást tohoto konrétního zařízení. Nepřenostitelný. Malý, kompaktní, robustní, těžko, nebo zcela neupdatovatelný (OS v různých zařízeních, pračkách, mikrovlnkách, chytrých hodinkách atd.).

Real-time systémy. Musí okamžitě reagovat na události reálného světa. Musí být velmi bezpečné. Příklad – autonomní auta, roboti atd.

Další klasifikace podle typu jádra, viz kernel.

2. Systémové knihovny a API

Systémové knihovny tvoří jakýsi obal kolem kernelu. Poskytují rozšířenou a základní funkcionalitu systémových voláním dalším programům. Např. v čistém C by nebylo možné bez assembleru použít třeba výpis na standardní výstup. Tuto funkcionalitu (na UNIXu) poskytuje systémová knihovna libc .

Na UNIXu je v libc další množství funkcí, které jsou součástí POSIXu.

Systémové knihovny: Především C funkce a datové typy. Rozhraní definované hlavičkovými soubory. Definice zpřístupněné statickými (linkují se při překladu) nebo dynamickými (linkují se za běhu) knihovnami. Hlavní knihovny v UNIX – libc.a libc.so, na Windows msvcrt.lib msvcrt.dll.

Spousta dalších.

Knihovní soubory: /usr/lib na většině UNIXech. Mohou se míchat s apliakčními knihovnami.

Statické knihovny: sufix .a (UNIX) nebo .lib (Windows). Pokud chci zkompilovaný program pouze spustit, tak mě statické knihovny vůbec nemusí zajímat. Statické knihovny se linkují přímo při překladu. Dále už nejsou potřeba. Výsledná binárka je také statická. Tyto binárky jsou oproti binárkám s dynamickými knihovnami jednodušší. Nevýhoda je ve větší paměťové náročnosti. Třeba funkce printf je nakopírována téměř do většiny apliakcí, zbytenčě tedyzabírá místo.

Dynamické (sdílené) knihovny: Knihovna se linkuje až dynamicky za běhu. Výhoda je, že knihovna a funkce v ní jsou potřeba jen v jedné kopii, šetří se tedy paměť. Problém s udržováním kompatibility, více aplikací mohou vyžadovat více verzí vzájemně nekompatibilních knihoven. Pokud se ovšem objeví chyba v dynamických knihovnách, pak stačí aktualizovat tuto knihovnu a není potřeba nic dalšího řešit. Při chybě ve statické knihovně je potřeba znovu zkompilovat s novou verzí této knihovny všechny programy, které ji používají.

Každá knihovna je potřeba v paměti pouze jednou. Podobají se objektovým souborům, obsahují symbolická jména funkcí. OS obsahuje “mini-linker”, který musí vyřešit tyto symboly – přiřadit správnou adresu, viz linker. Sdílené knihovny můžou využívat jiné sdílené knihovny.

Hlavičkové soubory: usr/include na UNIXu. Obsahují prototypy funkcí a definice datových struktur. Většinou požadovány při kompilaci C a C++ souborů (pokud program k něčemu je, pravděpodobně používá nějakou knihovnu a hlavičkový soubor).

POSIX: Funkce specifikované standardem POSIX. Často céčkové programy, které umožňují systémová volání.

Systémová volání: Systémová volání jsou realizovány na strojové úrovni. Každé volání je reprezentováno číslem. Ke každému číslu máme céčkovou funkci s rozumným jménem.Rozdílné pro různé OS.

K čemu tedy knihovny jsou? Knihovny obsahují spoustu kódu, který nám usnadní práci. Dále nám umožňují další abstrakci nad jádrem. Např POSIXové funkce nám umožňují spouštět stejné funkce i na vnitřně odlišných systémech, pouze se vyměňují knihovny.

MacOS místo C používá z velké části jazyk Objective C. Tento jazyk je i přes podobný název od jazyka C velmi odlišný. Neplatí tedy, že všechny operační systémy mají svoje rozhraní v C, nicméně C je stále v této oblasti dominantní a rozumí si i s MacOS.

Dokumentace: V UNIX většinou kompletní dokumentace pro všechny systémové funkce, příkaz man.

Překladače a linkery

C Compiler: POSIXové systémy dodávané většinou se zabudovaným C překladačem. Překladač si vezme zdrojový soubor s c kódem (*.c) a vytvoří objektový soubor (*.o), tedy strojový kód, který ale ještě není možné přímo spustit. Neobsahují kompletní program. Objektový soubor ještě neobsahuje přilinkované funkce, pouze čísla na ně odkazující. Přilinkování funkcí poté zajišťuje linker podle tabulky symbolů.

Archivy: Statické knihovny v UNIX se jmenují archivy (odtud *.a). Jsou podobné zip souboru, obsahující objektové soubory a tabulku symbolů (funkčních jmen).

Linker: Běžně poslední fáze překladu. Na UNIX příkaz ld. Kompletuje objektové soubory do sebe, linkuje k nim knihovny a vyprodukuje finální spustitelný soubor. Nepracuje s dynamickými knihovnami. Ve strojovém kódu neexistují žádná jména funkcí, je potřeba nejdříve přeložit jména funkcí pomocí tabulky symbolů a dále na místo těchto symbolů vložit konrétní adresu v paměti, kde se bude vyskytovat daná funkce – práce linkeru. Linker tedy přiděluje adresy jednotlivým kusům programu, překládá symboly na adresy. Začíná s prázdným programem, na vstupu má objektové soubory, kterým postupně přiřazuje adresy a ukládá je. Do své tabulky symbolů si pak poznačuje, kam jednotlivé soubody přestěhoval. Poté vezme další soubor. Pokud tento soubor potřebuje volat funkce z už zpracovaných souborů, tak do kódu podle své tabulky vloží příslušnou adresu.

Ve výsledném programu pak tedy už všechny instrukce pracují s validními adresami.

API

Programové rozhraní, usnadňujícím programátorům práci. Obsahuje už napsané funkcionality, které programátor může využívat. Zajišťují další abstrakci nad hardwarem (třeba pro práci s grafikou použiji nějaké grafické API, místo abych přímo pro GPU programoval, co se má vykreslovat na obrazovku).

Proč je (v UNIX) všechno soubor? Soubory mají některé dobré vlastnosti, které se dají přenést i na ostatní typy dat. Jde o část základní UNIXové filosofie. Umožňuje používání stejného souborového API nad velkým množstvím různých souborů.

3. Kernel a bootování

Jádro je nejblíže hardwaru a běží v privilegovaném systému. Privilegovaný řežim je jeden z řežimů procesoru (obvykle má procesor uživatelský a privilegovaný řežim). Výjimkou bývají mikrokontrolery, které mohou mít pouze jeden režim. Architektura x86 nabízí 4 privilegované úrovně (většina systémů ale používá pouze 0. a 3. úroveň).

Privilegovaný režim

Privilegovaný režim: Pouze některé operace jsou povolené v uživatelském módu. Téměř veškeré aplikace a kód operačního systému, vyjma jádra, beží v uživatelském režimu. Software, běžící v privilegovaném režimu, je jádro. Jádro může provádět jakékoliv instrukce, především může programovat MMU, které zajišťuje překlad adres.

MMU: Memory Managment Unit, překládá logické adresy na fyzické. Fyzické adresy jsou reálný kus hardwaru, který má nějaké číslo. Logická/virtuální adresa je také číslo, které ale nemá přímou spojitost s číslem fyzické adresy. Virtuální adresy slouží jednak ke swapování, tak zajišťuje každému procesu adresovatelný prostor začínající od nuly.

Stránkování: Fyzická pameť je rozdělena na rámce. Rámec neobsahuje data, ale je schopný je uložit (analogie poličky na knihy). Virtuální adresa je rozdělena na stránky, ty jsou skutečná data, která však nemají fyzické místo (analogie knih na polici). Rámec je tedy místo, stránka obsah. Překlad těchto adres zajišťují stránkové tabulky. Tuto tabulku vyrobí jádro pomocí MMU.

Swap: Historicky byla RAM paměť malá, bylo jí potřeba víc, než jí reálně bylo. Swap umožňuje ukládat data z RAM paměti na disk (analogie knižního skladu). Jakmile budou data potřeba, jsou data nahrána zpátky do RAM, samotná aplikace, kromě zdržení, nepozná rozdíl. Užitečné pro memory mapping files (vyžádáme si vyrobení virtuálních adres, které jsou mapované do souborů na disku (neobsahují paměť z RAM))

Kernel protection: Některé stránky mají nastavný příznak, že se k nim dá přistoupit pouze z privilegovaného režimu (pamě jádra).

Bootování

Start operačního systému: Po vypnutí se počítač dostane do defaultního stavu (především z důvodu energeticky závislých pamětí). Celou platformu je potřeba inicializovat.

Boot process: Proces nastartuje ze zabudovaného hardwarového inicializátoru. Jakmile je připraven, hardware předá řízení firmwaru (dříve BIOS, dnes EFI nebo UEFI). Firmware poté

načte bootloader a předá mu řízení, bootloader následně načte kernel (a předá mu řízení).

Kernel dále inicializuje ovladače zařízení, souborový systém a přenechá řízení procesu init (v UNIX-like systémech). Řízení přebírá user space. Kernel poslouchá a reaguje na události.

Init připojí zbylé souborové systémy a spustí uživatelský mód, poté spustí služby aplikací (démony) a nakonec login proces. Po loginu se načtou desktopové moduly, případně nějaký textový shell.

Bootloader: Lokalizuje kernel na disku, načte jej do paměti RAM a předá mu řízení. Z historických důvodů je velikost bootloaderu limitována na pár desítek Kib. Má tedy jen pouze omezené porozumění souborového systému.

Kernel

Typy architektur: Monolitické jádro (Linux, BSD), mikrojádro, hybridní jádro (MacOS)

Mikrojádro: Základním úkolem je ochrana paměti (MMU), hardwarová přerušení, plánovač procesů, komunikace se zbytkem systému (typicky posílání zpráv). Všechna ostatní funkcionalita je oddělené od jádra. Nicméně potřebujeme mnohem více funkcí, než poskytuje mikrojádro. "Pravá" mikrojádra jsou modulární. Další funkcionalita je v oddělených modulech. Každý ovladač pracuje v separátním procesu. Tradičně se těmto modulům říká servery. Velmi robustní, pokud spadne jeden server, není to takový problém, proběhne restart serveru.

Monolitické jádro: Vše, co dělá mikrojádro (vyjma posílání zpráv). Dále spravuje ovladače, souborové systémy, správa svazků (šifrování disků), implementace protokolů pro síťovou komunikaci. Málo robustní, pokud spadne jedna část jádra (nějaký ovladač), spadne všechno.

Nicméně monolit je efektivnější.

Hybridní jádro: Kombinace mikrojádra a monolitického jádra. V zásadě stejné jako mikrojádro, ale všechny servery z pravého mikrojádra spravuje jeden jediný server. Tedy spojíme mikrojádro a monolitické jádro bez mikrojádra. Není tedy potřeba přepínání mezi různými procesy pro správu různých částí systému. Méně robustní než mikrojádro, pokud spadne jeden ovladač, tak odejdou všechny, nicméně pořád je možné stav opravit bez restartu systému.

Mikro vs Mono: Mikrojádra nejrobustnější, monolitická jádra jsou naopak efektivní (není potřeba tak často přepínat kontext). Monolitická jádra je obecně snažší implementovar. Hybridní jádra jsou pak komrpomisem mezi těmito dvěma.

Exokernel: Ještě jednodušší než mirkojádr. Mnohem méně abstrakce (například neexistuje systém souborů, musí být vytvořen až jinde). Pro některé speciální systémy.

Type 1 Hypervisors: Typ exojádra. Obsahuje nejzákladnější ovladače pro komunikaci s hardwarem.

4. Systém souborů

Souborový systém je označení pro způsob organizace dat ve formě souborů a adresářu tak, aby k nim bylo možné snadno přistupovat.

Údaje v elektronické paměti jsou přístupné v podstatě jako vektor čísel. Avšak různé oblasti tohoto vektoru mohou být v závislosti na typu a stavu paměti dostupné za různou dobu.

Souborový systém zajišťuje ukládání a čtení dat paměťového média tak, aby s nimi uživatelé mohli pracovat ve formě souborů a adresářů. Základní ideou souborového systému je tedy zpřístupnění a ukládání dat pomocí hierarchicky organizovaného systému adresářů a souborů.

Souborový systém zaznamenává kromě jména souboru a jeho umístění v hierarchii adresářů další informace sloužící pro správu souborů. Především jsou to časové známky (nejdůležitější je čas poslední změny). Dále může souborový systém vést informace o vlastnících souborů a přístupových právech, což je důležité ve víceuživatelských systémech, nebo při zpřístupňování dat na disku pomocí počítačové sítě.

Souborový systém typicky existuje na nějakém blokovém zařízení. Souborový systém je sbírka souborů a adresářů. Tyto soubory a adresáře tvoří určitou "stromovou" hierarchii (ne přesně stromovou, ale téměř). Tento systém je přístupný a viditelný uživateli. Souborový systém je perzistentní (data se zachovají i po vypnutí počítače).

Soubor: Z pohledu operačního systému pouze sekvence bajtů + nějaká metadata. OS se nestará o obsah souborů. Soubory samotné nemají žádné jména, jména souboru jsou obsahem adresářů.V UNIX-like systémech je téměř vše reprezentováno jako soubor, jelikož soubor má spoustu dobrých vlastností. Je možné nad více různými typy dat používat stejné souborové API.

Běžný soubor: Obsahuje sekvenční data. Může mít vnitřní strukturu, o kterou se ale OS nestará. Ke každému souboru jsou přičleněná nějaká metadata o něm.

Adresář: Struktura, která jménům přiřazuje i-uzly. Chová se jako asociativní kontejner. V zásadě taky soubor, který obsahuje jména a odkazy na další soubory.

I-uzel: Soubor beze jména, základní stavební jednotka souborového systému v UNIXU.

Blokové a znakové soubory: Abstrakce připojených zařízení.

File Descriptor: Jádro udržuje tabulku otevřených souborů. File descriptor je potom index v této tabulce. Všechny práce se soubory jsou zařizovány pomocí file descriptorů. NeUNIXové systémy mají podobné koncepty – na Windows se jmenují handely.

Vlastnictví a přístupová práva: Uživatel, který vytvoří soubor, rozhoduje, co se s ním může dělat.

Diskové kvóty: Umožňuje každému uživateli dovolit používat pouze omezený adresnní prostor.

Disková zařízení: Disková zařízení umožňují blokový přístup, tedy četba a zápis probíhá po 512 (případně 4K) bajtových blocích. Interně chápáno jako velké pole bloků (ne vždy). Snaha ukládat víceblokové sekvence dat na bloky, které jsou fyzicky co nejblíž u sebe (jinak dochází k fragmentaci).

Cachování: Cache paměti slouži ke skrytí latence. Stejný princip jako cache mezi RAM a CPU. Implementováno v rámci OS.

Bufferování: Opačný směr jako cache (cache slouži pro zrychlení čtení, buffer pro zrychlení zápisu). Stejně jako při čtení, tak i při zápisu je nutnost zapisovat po celých blocích, což je nevýhodné, pokud chceme zapisovat hodně malých kusů paměti. Data jsou ukládána do bufferu a až časem se zapíší na disk.

RAID: Disky nejsou nejspolehlivější komponenty. Ve velkých úložištích je šance selhání disku vysoká. Způsob, jak toto řešit, je systém RAID (Redundant Array of Inexpensive Disks). Spojení více disků do jednoho diskového pole, které fungují jako jeden disk s redundantními daty. Jeden virtuální disk složený z mnoha fyzických disků. Ztráta jednoho disku tedy nepoškodí data. Více typů RAID. Typicky ho nemá na starosti souborový systém.

Soubor systémů řeší problémy blokových zápisů.

Co dalšího umí souborvý systém? Transparentní komprese dat, šifrování souborů (ačkoliv je přirozenější ji provádět už na úrovni bloků). Deduplikace dat (vlastně speciální forma komprese). Snapshoty (způsob, jak “zmrazit” v čase kopii souborového systému, ke které se můžu vrátit). Checksumy (nástroje, které umožní zjistit, jestli se soubor poškodil). Redundantní ukládání dat (RAID).

Komprese: Použití nějakého standardního kompresního alogritmu (neztrátového). Poměrně náročné na implementaci.

Deduplikace: Občas je ten stejný blok s daty uložený vícekrát. Některé souborové systémy jsou schopně roeznat tyto situace a případně interně namapují více těchto souborů do stejného bloku. Dedupliace je velmi důležitá u virtualizace. Implementována jako copy-on-write, tedy pokud z změním blok, o kterém se ví, že je duplicitní, tak je zkopírován, aby nebyl změněn ve všechy svých duplicitách.

Snapshoty: Fungují na podobném principu jako deduplikace. Implementace snapshotu je snažší než dedupliace. Snapshot je zamražený obraz souborového systému. Je levný, jelikož sdílí úložný prostor. Také implementován jako copy-on-write.

5. Procesy, vlákna, základní zdroje a multiplexing

Procesy a virtuální paměť

První počítače uměly spustit v jeden moment pouze jeden program, tedy nebylo nutné řešit procesy a sdílení jejich prostředků. Malé počítače byly schopné spouštět programyinteraktivně (vstup od uživatele). Většinou uživatel nevyužije počítač celý, je tedy vhodné, aby jej mohlo používat více lidí. Zde začíná být nutné nějak pracovat s procesya více spuštěnými programy.

Huistoricky byla definice procesu takováto:

Proces je spustitelný program. Může existovat více procesů, jsou mu přiděleny nějaké zdroje. Každý proces patří konkrétnímu uživateli.

Proces tedy ovládá jemu přidělené zdroje. Text obsahuje instrukce (název text z historických důvodů). Data, statická a dynamická. Paměť na zásobníku.

Klíčová vlasnost procesu je jeho adresní prostor. Stránkování a virtualizace paměti slouží k oddělení procesů od sebe, každý proces má tedy svůj vlastní adresní prostor a nevidí na adresní prostory ostatních procesů.

Přepínání procesů: Přepínání procesů je realizované přeppínáním stránkovací tabulky. Fyzická adresa se tedynemění, mění se pouze mapování virtuálních adres na adresy fyzické. Procesy tedy mohou mít stejně virtuální adresy (a obvykle mají), zatímco jejich fyzickéadresy jsou zcela jiné.

Stránkování a TLB: Překládání adres je pomalé. Procesor má tedy svou hardwarovou cache, která se jmenuje TLB (translation). Tato cache obsahuje několik položek už přeložených adres, vlastně slovník virtuálních a fyzických adres. Další překlad je už tedy podstatně rychlejší, paměť se jen najde v TLB. Pokud ovšem přepnu proces, musím tuto cache vylít a začít od začátku, po přepnutí procesu tedy nastává zpomalení, než se nově přeložené adresy zase uloží do TLB.

Sdílení procesorového času: Procesor umí spouštět pouze jeden program najednou. Procesorový čas je tedy rozdělen na malé úseky, ve kterých se jednotlivé procesy střídají, čímž vzniká iluze, že procesy běží najednou. Procesy se střídají tak rychle, že to nejsme schopni rozpoznat. Jednotlivé úseky procesorového času pak odpovídají rámcům paměti.

Výkon procesoru se v čase velmi zrychlovat, nicméně teď už narážíme na samotné fyzické limity. Zvyšování výkonu se tedy musí provádět přidáváním více jader.

Vlákna: Vlákno je sekvence instrukcí. Jiná vlákna mohou počítat jiné instrukce (tedy MIMD architektura). Každé vlákno má pak svůj vlastní zásobník a více vláken může sdílet určitou část paměti (sdílejí jeden adresovatelný prostor jejich nadřazeného procesu).

Z moderního pohledu je jednotka správy paměti, vlákno je potom jednotka výpočtu. Vlákno je oučástí nejvýše jednoho procesu. Proces se skládá z jednoho nebo více vláken.

Fork: Způsob vytváření nových procesů. Fork je systémové volání, které rozdělí jeden proces na dva identické, lišící se pouze svým identifikačním číslem. Po rozdělení pokračují oba dva procesy od stejného místa. Díky návratové hodnotě volání fork (identifikační číslo procesu) od sebe pak tyto procey můžeme odlišit a tak jim zadat jinou práci. Fork není příliš drahý, neprovádí se kopie paměti procesů, pouze se zkopíruje stránkovací tabulka, která je poměrně malá. Tato paměť se poté označí jako pouze pro čtení, v okamžiku, kdy se jeden proces pokusí zapsat, se aplikuje metoda copy-on-write. Je vyvolána výjimka procesoru, OS poté skutečně zkopíruje paměť pro oba procesy zvlášť, tedy už vytvoří skutečné dva adresové prostory. Iluze, že proces má vlastní paměť, je tedy zachována.

Init: Fork musí být zavolán z nějakého procesu. Tedy všechny procesy jsou vytvořené z jednoho původního procesu init, které vyvolá jádro při bootování. Z tohoto procesu pak pomocí forku vzniknou všechny ostatní procesy.

Proces vs. spustitelný soubor: Proces je dynamická antita. Spustitelný soubor je statický soubor. Spustitelný soubor obsahuje obraz paměťového prostoru procesu. Při spuštění spustitelného souboru se zkopíruje jeho obraz do paměti, následně se předá kontrola procesu.

Exec: Exec vezme existující proces a vymění jeho obsah paměti. Kompletně přepíše paměť procesu.

Plánování vláken

Komponenta OS, která je zodpovědná za plánování, je plánovač. Má dvě základní funkce, rozhoduje, kdy spustit jaké vlákno. Dále zařídí samotné přepnutí vláken a procesů. Plánovač je vždy součástí jádra.

Přepínání vláken: Vlákna v rámci jednoho procesu používají stejnou virtuální paměť, stačí tedy jen částečné přepínání. Nemusím vylívat TLB a přepínat stránkovací tabulku, tedy procesor je brzděn mnohem méně, než pri přepínání procesů. Je potřeba jen částečné přepnutí kontextu, tedy obsah registrů. Přepnutí procesu vyžadují plné přepnutí kontextu.

Fixní a dynamické plánování: U fixního plánování předem vím, kdy má dojít k jakým přepnutím. Zkonstruuje se tedy nějaký seznam, tabulka, přepínání a podle ní se pak funguje. Většinou ale není možně vědět předem, kdy bude docházet k jakým přepnutím, tedy je potřeba dynamicky přepočítávat, která vlákna a procesy budou potřeba přepnout v blízké budoucnosti.

Preemptivní vs. kooperativní plánování: Preemptivní – procesy, případně vlákna, ani neví, že se budou muset vzdát procesoru. Vlákno nebo procesor myslí, že má procesor jen pro sebe a navždy. Nevýhoda je ta, že odebírání procesoru není úplně zadarmo. Výhoda je v robustnosti, plánovač všechno ovládá a má přehled o využití procesoru.

Kooperativní plánování – jednotlivá vlákna nebo procesy ví, že musí sdílet procesor a tak se ho ve vhodný okamžik sami vzdají. Tento způsob je časově efektivnější, vlákna samotná zajišťují předávání procesoru. Také je tento systém ale méně robustní, stačí jeden špatný program, který může rozbít plánování všem ostatním.

Dnes používáno téměř výhradně preemptivní plánování. V dřívějších verzí MacOS byly dva samostatné plánovače, jeden pro vlákna a jeden pro procesy, přičemž plánovač pro vlákna byl kooperativní a plánovač procesů byl preemptivní.

Přerušení a hodiny

Přerušení: Umožňuje procesoru přerušit aktuálně prováděnou operaci. Pokud procesor zaznamená přerušení, podívá se do své tabulky přerušení a vykoná požadovanou operaci (typicky v privilegovaném režimu). Přerušení může být systémové nebo hardwarové.

Hardwarové přerušení: Realizováno jako signál, který přijde procesoru po nějaké sběrnici. Existuje programovatelný interrupt controler (PIC), který spravuje kontrolu přerušení.

PIC – 8 vstupů, jeden výstup. APIC – rozšířená verze.

Časovač: PIT – programmable interval timer. Obykle krystalový oscilátor + dělič frekvence. Připojený přes IRQ k procesoru. V APIC je zabudovaný vlastní časovač, lepší než PIT. RTC – hodiny reálného času, napojené na baterii, na základní desce.

Přerušení časovače: Generované PIT nebo APIC. OS může nastavit frekcenci. Umožňujě apliakce preemptivního plánování.

6. Komunikace a zámky

Souběžnost: Události mohou nastat ve stejnou dobu. Není důležité, jestli se nějaké události skutečně staly najednou, ale že se mohly stát.

Vlákno: Vlákno je taková sekvence instrukcí, že každá další instrukce se stane před tou předchozí. Tedy je zaručeno, že instrukce se provádí jedna za druhou. Základní jednotka plánování.

Proces: Základní jednotka, která vlastní prostředky. Může obsahovat jedno nebo více vláken. Jednotlivé procesy jsou od sebe izolované (mají vlastní paměť).

I/O vs komunikace: I/O bere standartní vstup a výstup. Komunikace probíhá v reálném čase. Často mezi procesy, probíhá automaticky. Dva typy komunikace podle směru, jednosměrná a obousměrná.

Komunikace skrz soubory: Komunikaci je možné provádět skrz soubory. Ne příliš optimální styl komunikace. Několik procesů otevře stejný soubor, jeden pak může psát data a ostatní

procesy mohou data zpracovávat. Používané, pokud se program používá jako modul. Případně tento způsob využívají překladače.

Komunikace srkz adresáře: Ve formě vytváření souborů nebo linků. Typické použití spool adresáře, adresář, kam může každý zapisovat. Jeden server pak kontroluje stav adresáře. Typicky emaily nebo komunikace s tiskárnou.

Roura: Zařízení přímo pro komunikace, pro posílaní proudu bajtů. Každý konec roury získá svůj file descriptor. Jeden proces zapisuje, druhý čte. Čtecí proces se zablokuje, když je roura prázdná, píšící proces se zablokuje, pokud je plný buffer. Roura může existovat i v systému souborů, tzv pojmenovaná roura.

Socket: podobný jako roura, ale schopnější. Umožňuje jednomu serveru komunikovat s více klienty. Každé spojení se chová jako obousměrná roura. Může být lokální nebo připojený skrze síť. Sockety jsou složitější než normální soubory, vytváření spojení je těžké, zprávy se mnohem více ztráci narozdíl od bžných souborových dat. Sockety mohou být internetové nebo unix doménové. Dále mohou být proudové (chovají se jako normální soubor) nebo datagramové (pro posílání individuálních zpráv/paketů).

Sdílená paměť: o paměti hovoříme jako o sdílené, pokud k ní může přistupovat několik vláken. Přirozeně se tak děje pro několik vláken jednoho procesu. Primární význam je intervláknová komunikace. I procesy samotné mohou sdílet jednu paměť, respektive paměťový prostor více procesů je mapován do stejné oblasti fyzické paměti.

Posílání zpráv: Komunikace pomocí zasílání zpráv. Může garantovat pořadí nebo ne. Můžeme se rozhodnout tolerovat ztrátu paketu. Používáno v sítích.

Synchronizace

Sdílená proměnná: Přístupná pomocí sdílené paměti. Typicky ve vícevláknovém programování. V zásadě každá globální proměnná programu. Případně pamě na haldě.

Race Condition: Pokud více vláken přistupuje ke stejné globální proměnné, její stav na konci programu nemůžeme zaručit. Je to způsobeno tím, že operace obecně nejsou atomické a tak jde o to, jaké vlákno přišlo dřív a jak s proměnou pracovalo. Místa, kde může k tomuto chování dojít, nazýváme kritická sekce.

Kritická sekce: Část programu, do které nesmí nikdo jiný vstoupit (nesmí být přerušena). X = X + 1 může být kritická sekce.

Mutex: Způsob, jak ochránit kritickou sekci. Mutex poskytuje dvě operace, zamknutí a odemknutí. Pokud proces vstoupí do kritické sekce, zamkne mutex a žádné další vlákno nemůže do kritické sekce vstoupit, dokud ji původní vlákno zase neodemkne.

Semafor: Podobné jako mutex, ale můžeme kontrolovat, kolik vláken vstupuje do kritické sekce. Mutex umožňuje pouze jednomu vláknu vstoupit do sekce, semafor to umožňuje více vláknům. Zjednodušeně atomický čítač.

Monitor: Zařízení programovacího jazyka (mutex a semafor obvykle zajišťován OS). Interně používá standardní mutex. Pouze jedno vlákno smí vstoupit do monitorované sekce.

Spinlock: Nejjednodušší forma mutexu. Lock metoda se opakovaně snaží vstoupit do uzamknuté sekce, tedy vyžaduje vyšší CPU time. Také označováno jako busy waiting. Dvě vlákna soutěžící o mutex na jednom CPU je velmi špatné. Na více CPU však může být velmi efektivní.

Bariéra: Pokud vlákno přijde k bariéře, musí počkat, než k baréře přijdou i ostatní vlákna, a teprve poté můžou všechna vlákna pokračovat dál.

Deadlock: Deadlock nastává, když jsou dvě nebo více jader zamknutá a čekají na sebe navzájem.

7. Ovladače zařízení

Ovladače jsou kus softwaru, který se stará o chod hardwarových zařízení, kromě RAM a CPU.

Většina zařízení, se kterými OS přijde do styku, umožňují číst, případně zapisovat, data.

Co je to ovladač? Kus softwaru, který komunikuje se zařízením. Součást OS. Obvykle specifický pro konrétní zařízení a operační systém. Nedá se tedy snadno, pokud vůbec, přenášet. Často součást samotného jádra operačního systému, ale není to pravidlo.

Kernelové ovladače: Jsou přímo součást kernelu. Běží s plnými právy jádra, což zahrnuje neomezený hardwarový přístup. Žádný nebo pouze minimální přepínání kontextu – rychlé, ale nebezpečné.

Mikrojádrové OS se snaží vysunout ovladače pryč z jádra. Nicméně ovladače stále potřebují přístup k hardwaru. Ovladač si od jádra vyžádá mapování na fyzické adresy konkrétního zařízení. Tento přístup je bezpečnější, ale dražší.

Spousta ovladačů může být přesunuta přímo do uživatelského prostoru. Tento způsob vylepšuje robustnost a efektivitu ovladačů. Chyby v ovladači nemůžou ohrozit funkci celého systému nebo jeho bezpečnost Menší rychlost, je potřeba přepínat procesy.

User-mode ovladače obvykle běží ve svých vlastních procesech. Tento přístup vyžaduje přepínání kontextu pokaždé, když zařízení vyžaduje pozornost (přerušení) nebo chce nějaký jiný proces využít konkrétní zařízení. Ovladače potřebují systémová volání, aby mohly komunikovat se zařízením, toto vyžaduje vyšší režii.

Dobré řešení je vytvoření velké části ovladače jako knihovny. Toto zkombinuje dobré vlastnosti z obou předchozích řešení. Není potřeba přepínat kontext, chyby a bezpečnostní problémy zůstávají izolované.

Často se tímto způsobem realizují GPU ovladače.

Port-mapped IO: Většina starých zařízení fungovala pomocí speciálního protokolu, který fungoval tak, že procesor měl kromě svého adresního prostoru pro paměť ještě jeden fyzický adresní prostor pro zařízení. Pro komuniaci se zařízeními, které nejsou fyzická paměť, se bude používat separátní sada 16bitových adres.

Memory-mapped IO: Dnes používáno veškerým moderním hardwarem. Zvýšila se paměť a začalo být lepší paměť mezi zařízeními sdílet. Procesor tak nemusí používat na zápis do zařízení speciální instrukce. Komunikace se zařízením probíhá jako zápis do paměti, I když zařízení nemusí přímo obsahovat paměť. Méně ovladačů, které je potřeba mít přímo v jádře.

Programmed IO: Vstup a výstup byl řízen procesorem, zařízení samo o sobě hloupé. CPU musí čekat, až je zařízení připraveno. Obvykle rychlé jako sběrnice. Jakmile zařízení začaly být chytřejší, stal se tento způsob zastaralý.

Interrupt-driven IO: Periferie jsou mnohem pomalejší než CPU, polling je velice drahý. Periferi mohou místo toho signalizovat dostupnost svých dat pomocí hardwarového přerušení. Toto umožňuje procesoru dělat jinou práci místo neustálého pomalého pollingu zařízení.

Direct Memory Access: Dovoluje zařízením přímo psát do nebo číst z paměti. Nejpoužívanější metoda komunikace se zařízením. Jedná se o veliké vylepšení oproti programmed IO. Zařízení nemusí vůbec komunikovat s procesorem, komunikuje přímo s pamětí, která je vysoce paralelní. Procesor se vůbec neúčastní přenosu dat. Zařízení zapíše data a poté vyvolá přerušení, které jen dá vědět, že data jsou už v paměti. Nevýhoda je v nižší bezpečnosti. Dříve prakticky žádná bezpečnost, bylo jen na “dobré vůli” zařízení, že pracovalo jen s přidělenou pamětí, nebylo to jak vynutit. Dnes už existují mechanismy, které vynucují používat pouze přidělenou paměť (IO MMU).

Sběrnice

Sběrnice je v podstatě něco, co nám zprostředkovává komunikaci. Hardwarově jdeo dráty, které dovolují procesoru komunikovat. Může být paralelní nebo sériová. U paralelních sběrnic problém se synchronizací, není technicky možné, aby všechny vodiče byly přesně dlouhé, což má za následek, že při vysokých frekvencích se signály mohou snadno rozcházet, dnes jsou sběrnice především sériové (právě I z tohoto důvodu).

PCI: Používá se dnes. PCI je ten skutečný nástupce ISA. Zásadní navýšení pracovní frekvence, až 133 MHz.

Ovladače síťových zařízení

Networking: Sít nám dovoluje sdílet data mezi počítači. Existují drátové a bezdrátové sítě. NIC - Network Interface Card.

Ethernet: Nejzákladnější modelový typ sítě je ethernet. Je to jedna samotné fyzické médium, tak protokol, jakým komunikují síťové karty. Moderní ethernet vždy point-to-point. Ethernet dokáže vzít nějaká data fixní velikosti a poslat je nějakému adresátovi, tomuto balíčku dat říkáme rámec. OS musí mít ponětí o těchto rámcích.

8. Sítě

Síť je nástroj, díky kterému můžeme sdílet data mezi počítači.
Hostname: Lidsky čitelný zápis číselné adresy. Zprava hiearchický (little endian).
Adresy: Strojově čitelná a numerická. IPv4 adresy mají 4 bajty, dnes už nedostačující. IPv6 má 16 bytů. Ethernetová (MAC) adresa - 6 bajtů. Big endian.
Typy sítí: Dva základní typy, LAN a WAN.
LAN: Local Aread Network. LAN typicky používá ethernet a wifi.
WAN: Wide Area Network. Samotný internet.

ISO/OSI model. Důležité 2., 3., 4. a 7. vrstva. 2, 3, 4 běžně součást OS.

Síťová vrstva je standardní součástí OS. U monolitických jader žije velká část této vrstvy přímo v jádru, u mikrokernelů je přesunuta do aplikační vrstvy.
Další části jsou v systémových knihovnách.
V jádru především běží ovladače síťových zařízení, síťové a transportní protokoly, směrování a paketové filtrování (firewall), síťová systémová volání a síťový systém souborů.

Systémové utility: Umožňují nastavení nebo konfiguraci sítě, případně o ní zjistit informace. Shellové příkazy ping, traceroute, config atd.
Aspekty sítí: Formát paketů, adresování a doručování paketů.

Protocol Nesting: Protokoly běží nad sebou. Proto se nazývá network stack, je tvořen vrstvami protokolů.
Pakcet Nesting: Pakety nižší úrovně berou ty vyšší úrovně jako nspecifikovaná data. Nezajímá je, o jaká data přesně jde.
Doručování dat probíhá point-to-point. Směrování je obvykle skryto před vyššími vrstvami. Fragmentace paketů.
Vrstvy vs. Adresování: Ne tak přímočaré jako packet nesting. Existují speciální protokoly pro překlad adres, především překlad lidksy čitelných adres na adresy ve forme IPv4 nebo IPv6. Případně ARP.
ARP: Address Resolution Protocol. Z IP adresy nemůžeme doručit rámec na konrétní síťovou kartu. Přiřazuje ip adresám správne MAC adresy. Vyžadován pro správné doručování paketů. OS systém má tabulku, která překládá IP na MAC adresy.
Bridge: Zařízení druhé vrstvy, v podstatě kus hardwaru. Spojuje dva fyzické segmenty sítě, přeposílá mezi nimi rámce. Lze udělat i softwarově. Most je pro protokoly vyšších vrstev transparentní (neviditelný), odděluje provoz různých segmentů sítě a tím zmenšuje i zatížení sítě.
ICMP: Internet Control Message Protocol, jeden z nejdůležitějších protokolů počítačových sítí založených na rodině protokolů TCP/IP. Protokol ICMP používají operační systémy v síti pro odesílání služebních informací, například chybových zpráv pro oznámení, že požadovaná služba není dostupná nebo že potřebný počítač nebo router není dosažitelný. ICMP se svým účelem liší od TCP a UDP protokolů tím, že obvykle není používán síťovými aplikacemi přímo, nýbrž je vygenerován na základě nějaké události.

TCP: Transmission Control Protocol, proudově orientovaný protokol na vrcholu IP. Funguje jako roura (přenáší sekvence bajtů). Musí respektovat pořadí a opakované zasílaní ztracených paketů.
IP: Internet Protocol, jde o základní protokol pracující na síťové vrstvě. Protokol IP poskytuje datagramovou službu rodině protokolů TCP/IP. Sám o sobě neposkytuje záruky na přenos dat a rozlišuje pomocí IP adresy pouze jednotlivá síťová rozhraní (doplňující služby jsou poskytovány na vyšších vrstvách, viz referenční model ISO/OSI). V současné době je stále ještě používána starší verze protokolu IPv4, nově se přechází na IPv6.
UDP: User Datagram Protocol, TCP přichází s netriviální režií, UDP protokol je v tomto mnohem jednodušší. Minimální režie. O protokolu UDP říkáme, že nedává záruky na datagramy, které přenáší mezi počítači v síti. Někdy je označován jako nespolehlivý, ale správněji by mělo být bez záruky doručení, což je hlavní rozdíl proti protokolu TCP.
DNS: Domain Name Service, hierarchický protokol pro překlad jmen. Doménová jména jsou rozdělena na části oddělené tečkami. Pak se provede rekurzivní překlad jména na adresu.

9. Příkazové interprety a uživatelské rozhraní

Shell: Základní netypovaný programovací jazyk navržený pro ovládání systému. Interaktivní režim – příkazy se píší za sebou a samostatně se ihned vyhodnocují. Téměř všechny shelly mají interaktivní mód. Můžeme psát i shellové scripty, tedy píšeme příkazy "do zásoby". Shell se velmi hodí na zautomatizování nějakého chování systému. Shellových jazyků je víc, ale moc se od sebe neliší. Různé dialekty shellové rodiny, nejznámější asi bash.

POSIX vyžaduje /bin/sh, který umí bash.

C Shell: přebírá základní syntaxi z C, ale stále docela odlišný od C. Vylepšený interaktivní mód. Stále se používá. Nekompatibilní s bashem.

Korn Shell: Také známý jako ksh. Něco mezi C Shellem a bashem. Základní požadavek POSIX.2.

Příkazy: Typicky názvy spustitelných příkazů. Spustitelný soubor je vyhledán v systému a shell provede fork + exec. Tento způsob není moc efektivní, na každý příkaz je potřeba vyrobit nový proces.

Některé příkazy, které buď není možné provést skrz fork nebo jsou velmi často používané, jsou v shellu přímo zabudované, například cd, export, echo atd.

Proměnné: Proměnné v shellu jsou všechny brány jako řetězec. Proměnné se používají pomocí $. Všechny proměnné jsou globální. Používání aritmetiky je problém, spousta shellů aritmetiku vůbec nepodporuje.

Příkazová substituce: Zapisuje se buď jako `command` nebo $(command). Příkaz se vyhodnotí a nahradí se v textu.

Prostředí: Proměnné prostředí se podobají shellovým prostředím, ale ne úplně. Proměnné prostředí se propagují do všech vykonávaných programů. Žádný program ale nemůže měnit proměnné prostředí svého rodiče. Běžné shellové proměnné se mění na proměnné prostředo pomocí příkazu *export*. Typické proměnné prostředí jsou: $PATH – říká systému, kde má hledat spustitelné programy, $HOME je cesta k domovskému adresáři, $PWD obsahuje cestu k aktuálnímu adresáři, $PS1 obsahuje primární prompt shellu atd.

Shell dovoluje * a ? expanzi (velmi základní regulární výrazy). Silná vlastnost je možnost použití globingu i v cestách, tedy například ls src/*/*.c

Shell dále obsahuje podmínky i smyčky. Shell nemá výrazy, vyhodnocování podmínek se tedy provádí pomocí příkazu *test.*

Grafické rozhraní

Okenní systém: Každá aplikace běží ve svém vlastním okně. Je možno spouštět více programů najednou, tedy je potřeba komunikovat s více programy najednou. Pomocí oken můžeme zobrazit na obrazovce více aplikací.

Ne u všech systémů je vhodné okna používat. Například u zařízení s malými obrazovkami (mobily) je vhodnější, aby apliakce zabírala celou obrazovku, ačkoliv jich může běžet více zároveň.

Terminál.

GUI vrstvy: Ovladač grafické karty. Vykreslování. Widgety. Layout.

10. Správa přístupu

Většina OS počítá s tím, že na nich běží víc než jeden program a případně víc než jeden uživatel.

Uživatel tvoří základní jednotku systému práv, bývá vlastníkem objektů. Uživatel je základní jednotkou vlastnictví.
Snaha použít počítač efektivně, tedy pokud jej nevyužívá jeden uživatel, může ho využívat uživatel jiný. Je tedy potřebovat uživatelům specifikovat nějaká práva a omezení.

Možno vyřešit virtualizací a rozsekání jednoho systému na více menších systémů pro jednotlivé uživatele, tento způsob ale přestává fungovat v okamžiku potřeby sdílet data. Sdílení dat vyžaduje nějaký systém kontroly, kdo k nim přistupuje.
Vlastnictví: Spousta objektů v OS může být někým vlastněna. Primárně soubory a procesy. Vlastník je typicky někdo, kdo objekt vytvoří, ale vlastnictví se může měnit, být přesouváno mezi uživateli. O tomto obecně rozhoduje původní vlastník.

Vlastnictví procesů: Každý proces patří nějakému uživateli. Procesy jsou ovšem oproti souborům speciální tím, že sami provádějí nějakou činnost naším jménem. Pro většinu objektů jejich přístupová práva pouze specifikují, co se s nimi může dít, pro procesy ale i

specifikuje, co mohou dělat. Z našeho pohledu jsou tedy procesy subjekty a ne objekty. Procesy jsou aktivním prvkem systému.
Vlastnictví souborů: Každý soubor patří nějakému uživateli. Přístupová práva specifikují, kdo může provádět jakou činnost s tímto souborem. Soubor je pasivní.

Jsou dva základní přístupy, jak organizovat, kdo co může.
Discretionaly access control: Vlastník souboru rozhoduje, kdo jiný a jak může se souborem pracovat. Slabý článek je vlastník, může špatně nastavit práva a způsobit bezpečnostní chybu.
Mandatory access control: O tom, jaká práva má který subjekt nebo objekt rozhoduje nějaká centrální autorita. Předpokládáme, že centrální autorita bude méně chybová.

Virtuální systémoví uživatelé: Často nejsou uživatelé vázáni ke skutečné osobě. Ne každá služba se k někomu váže, např webový server atd. I tyto služby ale potřebují vlastnit soubory a procesy a musí být nějak omezeny. Abstrakce uživatelů pro lidi se dá dobře použít i na služby.

Každá entita v systému má mít jenom ta práva, která nezbytně potřebuje (principle of lest privilege).
Je tedy těžší pro útočníky zneužívat chyby.
Separace privilegií: Různé části systému vyžadují různá privilegia. Systém jde tedy rozdělit na menší části, kterým přidělíme oprávnění, které nezbytně potřebují. Například systém jako celek potřebuje umět komunikovat po síti, ale reálně toto právo vyžaduje jen menší část procesů. Tedy server, který bude posílat emaily potřebuje umět komunikovat po síti, ale proces, který emaily ukládá na disk toto právo už nepotřebuje, je tedy vhodné tyto části z důvodu beztpečnosti odělit od sebe.
Komponenty komunikují velmi jednoduchým IPC (z důvodu jednoduché implementace a tedy i snažší zajištění bezpečnosti).
Základní separace procesů je realizován tím, že každý proces má vlastní adresní prostor. Pokud je potřeba, procesy si mohou vyžádat sdílenou paměť. V samotných procesech není nutné aplikovat nějaký systém oprávnění, jelikož procesy bývají zcela izolované od ostatních částí systému, mají jen svůj adresní prostor. Situace je ale jiná v souborovém systému, který je sdílený adresářový prostor, defaultně každý vidí všechno. Každý proces vidí celý souborový systém.
Politika přístupových práv: Snažíme se specifikovat, kdo co může. Máme tři kusy informace, subjekt (uživatel), slovesa (akce, které můžeme provést), objekt. Je velké množství způsobů, jak kódovat tyto informace.
Musíme být schopni subjekty pojmenovat. Uživatelé mají typicky jména nebo čísla. Pokud jde o jeden počítač, je identifikace jednoduchá. Situace se stává složitější s více počítači.
V POSIXových systémech máme dva typy subjektů, uživatele a skupiny. Uživatel může patřit do nějaké skupiny a ta má nějaká práva.
Dále existuje superuživatel, kterému se přístupová práva nekontrolují. Obvykle administrátor.
Uživatelé ale nejsou vždy správná abstrakce. Vytváření uživatelů je relativně drahé, pouze superuživatel může vytvářet nové uživatele. V některých případech je lepší vytvářet dvojice uživatel-program.

Je žádoucí speciální privilegia rozdistribuovat mezi ostatní uživatele, abychom nemuseli tak často využívat superuživatele, superuživatel je slabé místo systému.
Zdravím,

rád bych se jen pro jistotu zeptal, jestli je možné se normálně přihlásit na termín 18. 6. jako na řádný, jelikož má v hlavičce napsáno pouze "opravný" a ne "řádný a opravný". Nebo jestli jediný řádný termín, na který je možno se přihlásit, je ten 5. 6.
Sandbox: Snaha limitovat možou škodu způsobenou útokem přes nějakoý program. Program před tím, než se podívá na nějaký vbstup, zahodí všechna privilegia, která může. Útočník pak nezíská téměř žádná práva.

11. Virtualizace a kontejnery

Hypervisory

Co je to hypervisor? Kus softwaru, který umožňuje spouštět více OS. Jakési metajádro, které spouští jiná jádra. Smyslem je typicky zlepšit efektivní použití hardwaru.
Proč? Sdílení operačního systému má svá omezení. Izolace uživatelů od sebe není zcela dostatečné, nezodpovědný uživatel může ohrozit systém.
Rozhraní OS mezi OS a aplikacemi je složité, rozhraní mezi jádrem a hardwarem je relativně jednoduché. Rozhraní mezi hypervisorem a jinými jádry může být velmi jednoduché -> větší bezpečnost.
Virtualizace se používá pro izolaci jednotlivých kusů systému od sebe a pro lepší správu zdrojů.

Hypervisory se dělí na dva základní typy:
Typ 1: tzv. Bare Metal, implementují minimalistický OS, který běží přímo na hardwaru, kde jednotlivé procesy tohoto mni OS tvoří větší operační systémy.
Typ 2: tzv. Hosted. Běží v jiném operačním systému. Obvykle potřebuje podporu jádra. Šel by implementovat jako samostatný proces, ale obvykle je podporován jádrem.

Desktopová virtualizace: Architektura x86 postrádá virtuální supervisor mód. Ten umožňuje spustit virtualizovaný OS tak, aby nemusel běžet v privilegovaném režimu. Virtuální OS má pocit, že může používat privilegovaný režim, ačkoliv nemůže. Pokud virtuální OS potřebuje použít instrukce z privilegovaného režimu, je řízení předáno hypervisoru, který je vykoná sám.

Z principu není problém softwarově emulovat celá počítač, velký problém je ale rychlost. Provedení jedné instrukce virtuálního počítače zabere stovky až tisíce instrukcí reálného počítače.

Paravirtualizace: Zhruba rok 2005. Je zbytečně složité vyřešit problém virtualizace použitím virtuálního počítače. Není nutné vyrobit virtuální počítač, který věrně emuluje hardware,

lepší cesta je vyrobit OS, který neočekává, že poběží na skutečném hardwaru. Takový OS nemusí používat drahý privilegovaný režim, musí požádat hypervisor.

Rok 2005 přinesl virtuální supervisor mód procesoru v architektuře x86. 2008: MMU virtualizace.
Přestalo být tedy nutné vytvářet softwarová řešení problému virtualizace.

Princip paravirtualizace se ale do jisté míry zachoval u ovladačů ve virtuálních OS. Je snažší naprogramovat speciální ovladače pro virtuální OS, než se snažit zařízení virtuálně emulovat.

Paravirtualizace zařízení a procesoru na sobě není závislá. Ačkoliv používáme hardwarovou virtualizaci OS, tak můžeme používat paravirtualizace zařízení.

Virtuální počítač: Obvykle pod názve Virtual Machine. Všechno v počítači je virtuální. Mnohem snažší pro správu a ovládání než skutečný počítač.
Virtuální počítač potřebuje některé klíčové zdroje, CPU, blokové zařzení, na kterém je možno vytvořit souborový systém, síťové spojení a zařízení.

Sdílení CPU: OS systémy musí sdílet procesor. Stejný princip jako střídání procesů. Hypervisor má plánovač, který střídání zařizuje.

Sdílení RAM: Velmi podobný klasickému stránkování. Může být softwarové nebo hardwarové. Fixní virtuální paměť pro každý OS, probíhá dvojitý překlad adres. Virtuální OS dostane nějaký prostor virtuálních adres, se kterými pracuje jako s fyzickými.
Stínová tabulka stránek: Virtualizovaný OS chce měnit mapování adres, ale nemůže přistupovat přímo k fyzické MMU. Nemůže teyd provést intrukci na změnu tabulky, pokud se o to pokusí, řízení se předá hypervisoru. Ten udělá kopii stránkové tabulky virtuálního OS, ve které přepočítá všechny adresy, které jsou v adresovém prostoru VOS. Svému fyzickému MMU předá tuto přeloženou tabulku.
Sdílení sítě: Paravirtuální síťová karta, ta má jednoduché operace, transportuje rámce mezi hostitelským a guest systémem. Nachystá data do paměti a dá vědět hypervisoru.

VM může být snadno pozastaven. Registry a obsah RAM může být zkopírován na disk a OS může být znovu spuštěn kdykoliv později ve stejném stavu.
Uložený stav může být poslán i přes síť. Tento způsob se označuje jako paused migration.
Live migration: používá asynchronní paměťové snapshoty. Host kopíruje stránky a značkuje je jako pouze pro čtení. Snapshot je potom poslaný někam jinam. Obvykle během několika milisekund.

Kontejnery

Co jsou to kontejnery? Kontejnery jsou jakási meziúroveň mezi skutečným virtuálním počítačem a klasickými procesy. Můžeme se na ně dívat jako na virtuální počítač bez virtuálního jádra. Můžeme se na ně dívat jako mocnější procesy.
Proč? Jsou situace, kdy chceme vyrobit OS a něco s ním provádět a poté ho zase zahodit. Virtuální počítač nějakou dobu nabíhá, je potřeba vše inicializovat, docela drahé. Docela velké paměťové nároky.

Více kontejnerů má společné jádro. Neumožňuje tedy takovou univerzálnost (nemůžu spustit linuxovou aplikaci jako kontejner na Windows a naopak, to u plně virtuálních strojů není problém) Spousta zdrojů společná, dá se tedy ušetřit.
Výrazně se ušetří bootovací čas. Naproti tomu virtuální stroj lze snadno uspat, poslat po sítí atd. To u kontejnerů nelze.