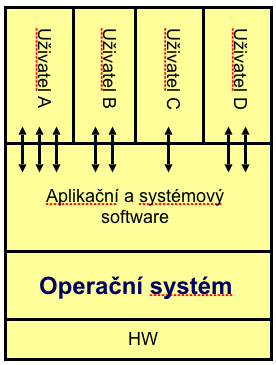
Operační systémy

Architektury operačních systémů, aplikační programovací rozhraní operačních systémů. Periferie, jejich správa, ovladače. Procesy a vlákna, synchronizace procesů a vláken.   
*PB152/PB153*

# Operační systém

Neexistuje jednotná definice OS, spíš se definuje jako souhrn činností spíš než částí.

OS je správce prostředků počítače – řídí přístup k CPU, pamětem, I/O zařízením. A řídící složka programů – řídí spouštění, snaží se zabránit chybám a vzájemnému ovlivňování.



Podle využití jsou OS různě nastavené.

* Stolní systémy – důraz na komfort pro uživatele
* Distribuované systémy – musí mít ošéfovanou komunikaci
* Real-time systémy – důraz na běh v reálném čase
* OS mobilních zařízení – omezen nedostatkem zdrojů (od baterky až po procesor)

# Provázání s Hardware

OS je svázán s hardware, na kterém má běžet, a často potřebuje konkrétní funkce, aby mohl běžet korektně.

Vývoj od OS pro **dávkové systémy** – přes děrné štítky se jeden program nahrál do paměti.

Přes **multiprogramování** – v paměti je už více úloh najednou. Úloha má CPU, dokud nepožádá o IO, pak čeká a je zavedena jiná úloha. Už musí existovat ochrana proti přepsání jedné úlohy druhou.

Po jednoprocesorový **multitasking**. Rozšíření multiprogramování, přidání časovače. Později rozšířeno o víceprocesorový multitasking (teprve tady přichází pravý paralelismus).

## Procesor bez přerušení

Při provádění IO operace se neustále ptá, jestli už byla dokončena – CPU není efektivně využitý.

Loop

Získej další instrukci

Proveď insrukci

End loop

## Procesor s přerušením

Loop

Získej další instrukci

Proveď instrukci

Pokud došlo k přerušení (a přerušení je provoleno), ulož právě prováděný kód a začni obslužnou rutinu

End loop

* Přerušení = signál, že je potřeba něco zpracovat
* Asynchronní
* Při zpracování přerušení je další přerušení defaultně zakázáno (maskováno)
  + Jde obejít
* OS je řízen přerušeními
* Hw/Sw přerušení

## Ochrana CPU

Uživatelský vs. Privilegovaný režim. Některé instrukce jde pustit pouze v privilegovaném režimu (IO, nastavování některých registrů).

Privilegovaný -> uživatelsky – pomocí speciální instrukce CPU

Uživatelský -> privilegovaný – pomocí přerušení

OS je pravidelně spouštěn časovačem. Může tak kontrolovat, co se děje.

## DMA (Direct Memory Access)

Způsob přenosu dat mezi IO zařízením a pamětí – IO zařízení nahraje data přímo do paměti (namísto aby je přenášel CPU po jednom bytu).

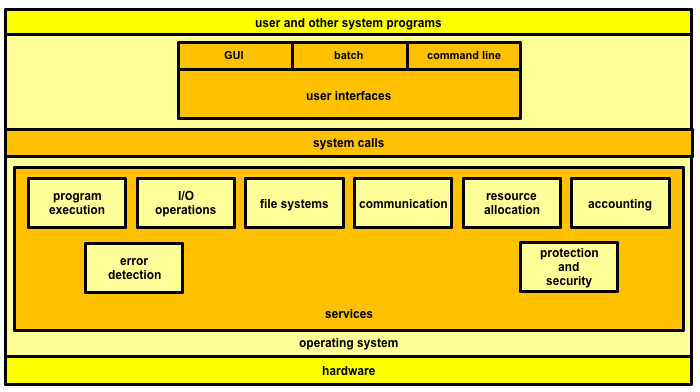
CPU požádá o přenos dat, po skončení přenosu se o tom dozví pomocí přerušení. Má k dispozici celý blok dat.

# Služby OS

Typického dnešního OS.

Komponenty, správa

* Procesů
* Operační paměti
* Souborů
* IO zařízení
* Sekundární paměti
* Síťových služeb
* Ochranný systém
* Interpret příkazů (shell, gui…)



## Správa procesů

Proces je spuštěný program, jde o jednotku práce systému (procesy jádra OS, uživatelské procesy). Pro své spuštění potřebuje zdroje, které mu jsou přiděleny pomocí OS.

* Vytváření / ukončování procesů
* Potlačování / obnovování procesů
* Mechanismy pro synchronizaci
* Mechanismy pro komunikaci
* Mechanismy pro detekci a řešení uváznutí

## Správa paměti

Při spuštění procesu se nahrává paměť do paměti. Za běhu je potřeba paměť procesu přidávat / odebírat, po ukončení je potřeba paměť uvolnit (bezpečně, aby se k ní nedostal další program).

* Znalost kdo používá jakou část paměti
* Alokace / dealkoace paměti podle požadavků procesů
* Rozhodování, který proces zavést do paměti

## Správa souborů

Pro OS je soubor posloupnost bytů. OS maskuje přístup do různorodých IO zařízení pomocí jednotného rozhraní.

## Správa IO zařízení

Snaha o skrytí specifik jednotlivých zařízení, některé OS všechny navenek prezentují jako speciální soubory.

Realizováno pomocí ovladačů.

## Správa sekundární paměti

Typicky disky, většinou se spravují formou souborů, resp. Souborového systému.

OS se stará o přidělování místa, plánuje jeho činnost.

## Ochranný systém

Realizuje potřebu ochránit proces od vlivů dalších procesů, reálně jde o řízení přístupu ke zdrojům a paměti.

Pomocí HW OS zajišťuje, že procesor může používat pouze svoji paměť.

Časovač zajišťuje, že si žádný proces nemůže ukrást procesor jenom pro sebe.

Režim CPU brání uživatelským procesům spouštět privilegované instrukce

## Interpret příkazů

Uživatelské prostředí. Od terminálu až po GUI. Může jít o součást jádra OS, ale taky může jít o speciální program.

## Interní služby OS

Zařizují provoz samotného OS. Jde o alokování zdrojů, účtování a ochranu.

## Systémová volání

Rozhraní pro uživatelské procesy. Snaha o standardizaci – POSIX, Win32.

# Architektura / struktura

Různé přístupy, ve skrze se liší velikostí jádra. Monolit / modulární / mikrojádro.

## Vrstvová architektura

OS se dělí do vrstev. Nejnižší vrstvou je hardware, nejvyšší uživatelské rozhraní. Každá vrstva může používat pouze funkce vrstvy těsně pod sebou.

Vysoce modulární, za cenu vyšší režie a tím pádem pomalejšího běhu. => v reálu se používá s omezeným počtem vrstev

## Klasický OS (non-process kernel OS)

OS je samostatná entita, vždy běží v privilegovaném režimu. Jako procesy běží pouze uživatelské programy.

Program a data OS jsou ve sdíleném adresovém prostoru a sdílí je všechny uživatelské procesy. Celý OS lze provádět v kontextu uživatelského procesu (leží v jeho adresovém prostoru). Přerušení vyvolá implicitně pouze přepnutí režimu procesoru, ne změnu kontextu. Kontext se mění jenom pokud to vyžaduje plánování.

## Procesově orientovaný OS

OS = souhrn systémových procesů, jádro je separuje, ale taky dovoluje synchronizaci. Snaha o co nejmenší dobu běhu v privilegovaném režimu.

## OS s mikrojádrem

Minimalizace procesově orientovaného OS, který poskytuje jen minimum funkčnosti.

Primitivní správa paměti.

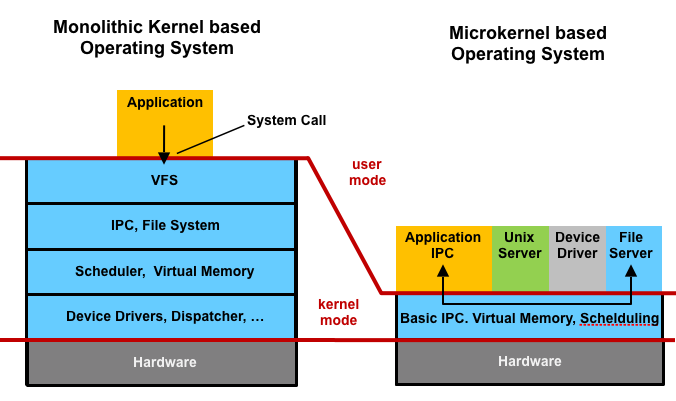
Komunikace mezi procesy (IPC – interprocess communication)

Většina funkčnosti / služeb se přesouvá do uživatelské oblasti – ovladače, systém souborů, virtualizace paměti, …

Výhody

* Snadná přenositelnost OS
* Teoreticky vyšší stabilita (malé moduly jsou jednoduché a tím pádem lépe testovatelné a spíš se chovají korektně)
* Vyšší bezpečnost (méně kódu běží v privilegovaném režimu)
* Flexibilita
* Jednotné poskytování služeb (výměnou zpráv)

Na druhou stranu vede ke zvýšené režii.



## # Konkrétně – Linux

Monolitické jádro, do kterého lze za běhu přidávat kód – moduly. Nevyžaduje podpis ovladače (ale umožňuje tuhle funkcionalitu)

## # Konkrétně – Windows

Do jádra se dají vkládat ovladače, které pak běží v privilegovaném režimu. Aktuálně musí být ovladač podepsaný, jinak nelze zavést.

# # Bootování

BIOS provede inicializaci komponent. Na základě konfigurace zjistí, odkud zavést OS, pak zavede jádro a spustí ho.

# Programové rozhraní

Rozhraní OS, které zpřístupňuje uživatelským procesům. Typy

* Správa procesů – fork, exec, wait, abort, …
* Správa souborů
* Správa IO zařízení
* Informace o prostředí (time, …)
* Detekce chyb a chybové řízení
* Komunikace (mezi procesy)

Syscall vede k přerušení, procesor se přepne do privilegovaného režimu a řízení převezme OS.

OS může mít ještě vnitřní služby, které slouží k efektivnímu provozování OS – přidělování zdrojů, bezpečnost apod.

POSIX, WIN32 standardy

# Periferie

Všechno možné, rozmanité, ale naštěstí standardizované. Běžně používané prvky

* Port (většinou 4 registry)
  + Data-in – pro čtení ze zařízení
  + Data-out – pro zápis do zařízení
  + Status
  + Control – ovládání zařízení
* Sběrnice
* Řadič

Se kterými umí OS/procesor pracovat pomocí standardních instrukcí (IN, OUT). Ty obsahují i adresu zařízení.

Přístupy

* Polling
* Přerušení
  + DMA

Pro samotné periferie existují ovladače – vrstva pracující přímo s hw periferie. Aplikačním procesům i některým částem oS jsou periferie poskytovány pomocí aplikačního rozhraní.

Bloková / znaková zařízení - Disk / myš

* Blokující
* Neblokující
  + IO zařízení bufferuje vstup
  + Proces si pak okamžitě přečte data z bufferu
* Asynchronní
  + Přerušení

Některé operace s periferiemi je potřeba plánovat (přístup k disku z více procesů). OS se pak stará o plánování / buffering.

Caching / spooling / rezervace zařízení.

Spooling udržuje frontu dat určených pro nějakou periferii, pokud zařízení pracuje pouze sekvenčeně (tiskárna).

## # Síťová zařízení

Přístup se značně liší od znakových i blokových zařízení => mívají samostatné rozhraní.

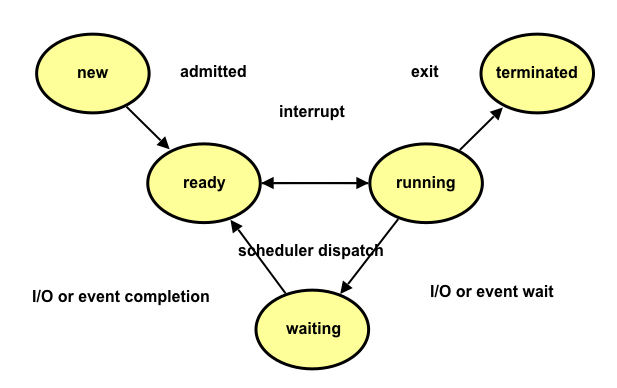
# Procesy a vlákna

Proces = aktivní program. Obsahuje

* Čítač instrukcí (= kód programu)
* Zásobník (= lokální proměnné)
* Datovou sekci
* Instrukční sekci (= aktuální / následující instrukce)

Procesy jsou spouštěny dalším procesem.

## Stav procesu



nový = zavádí se do paměti, není schopen vykonávat činnost

běžící = některý procesor právě vykonává instrukci procesu

čekající = čeká na nějakou událost

připravený = čeká na přidělení procesoru

ukončený = ukončil svoje provádění

O přesunu mezi stavy rozhoduje OS.

## Informace o procesu

Process Control Block (jmenuje se různě v různých OS) = tabulka s informacemi potřebnými pro definici a správu procesu

* Jaké zdroje používá
  + Kolik paměti a jaké paměti
  + Kolik zdrojů zpotřeboval
  + Otevřené soubory
* Čítač instrukcí – při pozastavování procesu se ukládá jaká instrukce má následovat
* Registry procesoru

Process Control Block se při přerušení ukládá. Proces to nezaznamená.

## Přepnutí kontextu

Uloží se „starý“ proces a jeho Process Control Block.

Zavede se nový proces.

Děje se hodně často. Výrobci HW se to snaží usnadňovat.

## Plánování

Fronty plánování.

Krátkodobý plánovač.

Strategický plánovač.

Střednědobý plánovač.

…

# Vlákna

(taky „sleď“)

= objekt, který vzniká v rámci procesu, je viditelný pouze uvnitř procesu a má nějaký stav

Pokud OS nepodporuje vlákna, jednotknou činnosti je proces.

Pokud OS podporuje vlákna, jednotkou činnosti je vlákno.

Každé vlákno má vlastní

* Zásobník
* PC = program counter
* Registry
* TCB = thread context block

Zdroje procesu sdílí všechna vlákna jednoho procesu.

* Pokud vlákno změní nelokální obsah (mimo zásobník), změnu vidí i ostatní vlákna
* Soubor otevřený jedním vláknem vidí i ostatní vlákna
* Vlákna se samostatně neodkládají (protože sdílí paměť)
* Problémy při paralelním přístupu ke stejným zdrojům

## Stavy

* Běží
* Připravené
* Čekající

Ukončení procesu ukončuje všechna jeho vlákna.

## Implementace na uživatelské úrovni

Jádro o jádrech neví, implementace pomocí vláknové knihovny na úrovni aplikace.

* Přepojování mezi jádry nepožaduje provádění funkcí jádra
* Nepřepíná se kontext/režim procesoru
* Aplikace se plánuje sama a volí si nejvhodnější algoritmus

Ale taky

* Jádro manipuluje pouze s celými procesy
* Pokud vlákno zavolá syscall, zablokuje celý proces
* Vlákna nemůžou běžet paralelně na různých procesorech

## Vlákna na úrovni jádra

Jádro může plánovat samostatná vlákna, takže ta můžou běžet paralelně na různých procesorech. K blokování dochází na úrovni vlákna.

Přepojování mezi vlákny stejného procesu jde přes jádro => pomalejší.

Přístupy lze kombinovat.

# Synchronizace procesů / vláken

Procesy a vlákna si občas potřebují povídat. Eg. Rodič pustí dítě, aby uklidilo kuchyň a pak chce vědět o výsledku.

Při synchronizaci/komunikaci dochází k všemožným problémům

* Uváznutí
* Stárnutí – dva procesy si opakovaně posílají zprávy a třetí proces stárne

Proto při sdílení prostředků, kdy procesy používají a modifikují stejná data

* Operace zápisu musí být vzájemně výlučné
* Operace zápisu musí být vzájemně výlučné se čtením
  + Samotné čtení může být realizováno souběžně
* Pro realizaci se může použít kritická sekce
  + Najednou může být v kritické sekci pouze jeden proces
  + Vlákno / proces si zažádá o přístup do kritické sekce a dostane ho jenom pokud tam aktuálně není nikdo jiný
  + Může nastat deadlock / livelock
    - Žádné dva procesy nemůžou provádět kritickou sekci spjatou se stejným zdrojem
    - Žádný proces čekající na kritickou sekci nesmí být odkládán nekonečně dlouho
  + Aktivní čekání

Dalším řešením je používat atomické akce (= jedna instrukce procesoru)

* Test\_and\_set
* Exchange / swap

Race condition = přístup více procesů / vláken ke stejnému zdroji tak, že se ovlivňují

## Semafory

Podpora na úrovni OS

* Proces je uspán a probuzen až když má na „semaforu zelenou“
* Může dojít k uváznutí

Binární a obecné semafory (můžou nabývat více hodnot)

## Monitory

Umožňuje bezpečné sdílení abstraktního datového typu souběžnými procesy. Umožňuje vzájemné vyloučení i možnost signalizovat dalším vláknům.

## Další možnosti

Signály / roury / zprávy?