# 11. Teorie operačních systémů

# Co to je OS

Operační systém je prostředník mezi hardwarem (technickým vybavením počítače) a konkrétním programem, který uživatel používá.

Operační systém je v [informatice](https://cs.wikipedia.org/wiki/Informa%C4%8Dn%C3%AD_technologie) základní [programové](https://cs.wikipedia.org/wiki/Po%C4%8D%C3%ADta%C4%8Dov%C3%BD_program) vybavení [počítače](https://cs.wikipedia.org/wiki/Po%C4%8D%C3%ADta%C4%8D) (tj. [software](https://cs.wikipedia.org/wiki/Software)), které je zavedeno do [paměti](https://cs.wikipedia.org/wiki/Opera%C4%8Dn%C3%AD_pam%C4%9B%C5%A5) počítače při jeho startu a zůstává v činnosti až do jeho vypnutí. Skládá se z [jádra](https://cs.wikipedia.org/wiki/J%C3%A1dro_(informatika)) (kernel) a pomocných [systémových nástrojů](https://cs.wikipedia.org/wiki/Syst%C3%A9mov%C3%BD_software). Hlavním úkolem operačního systému je zajistit uživateli možnost ovládat počítač, vytvořit pro [procesy](https://cs.wikipedia.org/wiki/Proces_(program)) stabilní [aplikační rozhraní](https://cs.wikipedia.org/wiki/API) (API) a přidělovat jim systémové zdroje.

## Druhy OS

Jednouživatelské jedno úlohové (single-user single-task) - žádné prostředky ochrany souborů a disků, neumožňuje běh více procesů (programů) najednou, například: MS-DOS

Jednouživatelské více úlohové (single-user multi-task) - multitasking, možnost paralelního (současného) běhu několika procesů, kooperativní multitasking, preemptivní multitasking

Víceuživatelské více úlohové (multi-user multi-task) - možnost zpracovávat požadavky více uživatelů přihlášených do systému, například: Windows NT, Windows 2000, Unix, Linux, Windows XP

# Jádro

Jádro je v informatice označení pro jádro operačního systému (kernel). Jádro je zavedeno do operační paměti při startu (bootování) počítače a je mu předáno řízení. U pokročilých operačních systémů jádro nikdy neztrácí kontrolu nad počítačem a po celou dobu jeho běhu koordinuje činnost ostatních běžících procesů.

Hlavní úkol jádra spočívá v přidělování paměti a času procesoru či pocesorů, programů, ovládání zařízení PC (pomocí ovladačů) a abstrakci funkcí (aby bylo například možné načítat soubory z pevného disku a z jednotky CD ROM stejným příkazem). Pro zajištění bezpečnosti OS je nutné, aby procesor podporoval 2 mody činnosti: Omezení pro aplikace a privilegovaný (se speciálním strojovými instrukcemi) pro jádro.

Privilegovanému modu se proto někdy říká kernel mód. Podle architektury OS se typicky rozlišuje **mikrojádro** (jádro je velice jednoduché) a obsahuje pouze zcela základní funkce, zbytek OS se nachází mimo toto jádro v aplikacích (serverech, např. Symbian OS) a **makrojádro** v (monolitické jádro – jádro je rozsáhlé, obsahuje velké množství funkcí pro všechny aspekty činnosti OS včetně např. souborového systému). Toto jádro je typické pro OS Unix. Jakýmsi kompromisem je hybridní jádro, které se snaží kombinovat výhody mikro jádra i monolitického jádra (Windows MT, OS X).

## Základní povinnosti jádra

Základní účely jádra spočívá v ovládání prostředků počítače a umožnění ostatním programům běžet a používat je. Typicky se prostředky skládají z:

### Procesor

Centrální procesorová jednotka neboli procesor. Jde o základní část počítače. Ta má za zodpovědnost běh nebo vykonávání programů. Jádro má za úkol rozhodování o tom, který z programů bude přidělen procesoru nebo procesorům (na každém procesoru v modernějších procesorech, na každém jádře může v jednom okamžiku běžet právě jeden program).

### Paměť Počítače

Paměť je používána k ukládání jak strojových instrukcí programu, tak i dat. Obvykle musí být v paměti přítomno obojí, aby se mohl provést program. Často chce paměť používat mnoho programů, dohromady požadujících více paměti, než má počítač k dispozici. Jádro má za povinnost rozhodování, kterou část paměti může každý proces použít a rozhodnout, co dělat, když je nedostatek volné paměti.

### Jakékoli I/O zařízení

Jakékoli I/O zařízení přítomné v PC jako jsou diskové jednotky, tiskárny, displeje atd. Jádro přidělí žádostem aplikací příslušná I/O zařízení a pečuje vhodným způsobem o užívání zařízení.

Jádro také obvykle poskytuje metody pro synchronizaci a komunikaci mezi procesy (nazývané meziprocesová komunikace neboli IPC). Jádro může realizovat tyto prvky samo, nebo se spolehnout na některý z procesů, který běží, aby poskytoval prostředky ostatním procesům, také v tomto případě musí poskytovat některé způsoby IPC, aby umožnil ostatním procesům přistupovat k tomuto zařízení.

## Správa procesu

Hlavním úkolem jádra je umožnit vykonávání aplikací a podporovat je metodami jako je abstrakce HW. K běhu aplikace jádro typicky nastavuje adresný prostor pro aplikaci, nahrává soubory obsahující aplikační kód do paměti (třeba pomocí stránkováni paměti) nastavuje zásobník pro program a poté předává řízení na pozici uvnitř programu, kde začíná jeho vykonávání. Multitasking jádra umožňuje poskytovat uživateli iluzi současného běhu libovolného počtu procesů na počítači. Typický počet procesů, které mohou na systému běžet zároveň je rovný počtu nainstalovaných procesorů. Nicméně to nemusí tak být, když procesor podporuje zároveň simultánní multithreadning.

### Preemptivní multitasking

V preemptivním multitaskingovém systému jádro dává každému programu procesor na určitý časový úsek a přepíná z procesu na proces tak rychle, že to uživateli připadá, jako by procesy běžely zároveň. Jádro používá plánovací algoritmus k výběru, který proces poběží další a kolik času mu bude přiděleno. Vybraný algoritmus může dovolit některým procesům, aby měli vyšší prioritu než ostatní. Jádro obvykle také poskytuje procesům prostředky pro komunikaci (již zmínění IPC) např. sdílenou paměť, předávání zpráv nebo vzdálené spouštění procedur.

### Kooperativní multitasking

Ostatní systémy – většinou na méně výkonných PC, můžou poskytovat kooperativní multitasking, kde každému procesu je dovoleno běžet nepřerušeně, dokud nevyšle speciální žádost, která ohlásí jádru, že může přepnout na jiný proces. Tyto dotazy jsou známí pod jménem Yieloling a typicky se vyskytují v souvislosti s mezi procesovou komunikací nebo čekáním na událost.

Starší verze OS Windows a MAC používali kooperativní multitasking, ale přešli na preemptivní, jakmile výkon počítačů, pro které byly určeny, vzrostl. OS může také podporovat multiprocesing (Symetrický – SMP nebo s neuniformním přístupem k paměti – NUMA). V tomto případě různé programy a vlákna mohou běžet na různých procesorech. Jádro musí být pro takový systém navrženo (musí podporovat běh dvou částí jednoho kódu zároveň. To obvykle vyžaduje použití synchronizačního mechanizmu k zajištění, že žádné dva procesy nemohou měnit jedna data ve stejný čas.

## Správa paměti

Jádro má úplný přístup do paměti systému a musí umožnit procesům přistupovat k této paměti bezpečně, jak potřebují. Často prvním krokem bývá virtuální adresování. Obvykle dosažené stránkováním nebo segmentací. Virtuální adresování umožňuje jádru nahradit fyzickou adresou jinou – virtuální. Virtuální adresový prostor může být pro každý proces jiný. Část paměti, ke které přistupuje jeden proces na určité virtuální adrese, může být jiná, než paměti, ke které má na stejné adrese přístup jiný proces. To umožňuje každému programu se chovat, jakoby byly jediný běžící program a chrání aplikace před kolizí. Na mnoha počítačových systémech může virtuální adresa odkazovat na data, která zrovna nejsou v paměti.

Vrstva nepřímosti poskytovaná virtuálním adresováním umožňuje OS používat jiná datová uložiště, jako je pevný disk k ukládání dat, která by jinak byla v hlavní paměti (RAM). Díky tomu může OS umožnit programům používat více paměti, než má systém fyzicky k dispozici. Když program potřebuje data, která právě nejsou v paměti, jednotka pro správu paměti (obvykle je součásti Procesoru) oznámí jádru, že tam nejsou a jádro odpoví zapsáním obsahu nečinného bloku paměti na disk a nahradí daty, která si vyžádal program. Poté může program pokračovat od bodu, kde byl zastaven. Tento koncept je znám jako stránkování na žádost.

Virtuálním adresováním také umožňuje vytváření z virtuální části paměti ve dvou rozdělených oblastech, jedna bývá rezervována pro jádro a ostatní pro aplikace. Procesor nepovoluje aplikacím, aby adresovali paměť jádra a tedy, aby aplikace poškodila běžící jádro. Toto důležité rozdělení paměťového prostoru hodně přispívá vhodnější koncepci v oblasti jader a je téměř univerzální ve většině systémů.

## Správa zařízení

Pro vykonávání užitečné funkce proces potřebuje přistupovat k periferním zařízením připojených k PC, která jsou ovládaná jádrem přes ovladače zařízení. Např. k zobrazení něčeho na obrazovce musí aplikace vytvořit požadavek pro jádro, které ho předá k řadiči displeje, který poté zodpovědný za skutečné vykreslení znaků nebo pixelů.

Jádro musí udržovat seznam dostupných zařízení. Tento seznam může být znám předem (např. ve vloženém systému, kde jádro je přepsáno, když se dostupný HW změní), nastavení uživatelem (nejčastěji na starších počítačích v systémech, které nebyly navrženy pro osobní používání) nebo zjištění OS za běhu, to je obvykle známe pod pojmem Plug and Play (USB zařízení).

## Systémová chování

K vykonání užitečné práce musí být proces schopen přistupovat ke službám jádra. Každé jádro to realizuje jinak, ale musí poskytnout knihovny (API), které poté vyvolávají související funkce jádra.

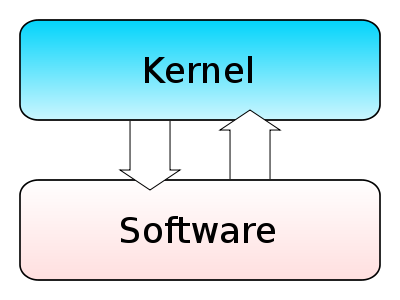
Metody vyvolávání funkcí jádra jsou u každého různé:

1. Softwarové přerušení (dostupné na většině procesorech)
2. Volací brána – Je to speciální adresa, kterou lze zavolat z aplikace, ale podpora procesoru přesměruje volání do jádra.
3. Speciální systémové instrukce
4. Pomocí fronty v paměti (Aplikace, která vytváří velké množství požadavků, ale nepotřebuje čekat na výsledek, může přidat požadavek do speciální oblasti paměti, kterou jádro pravidelně prohlíží, aby vyřídilo shromážděné požadavky)

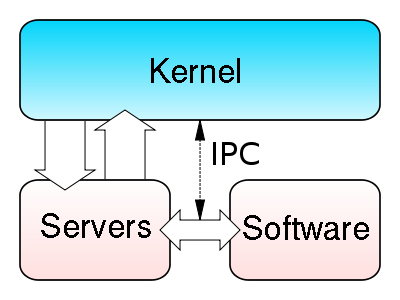
## Jádra jader (přidat obrázky)

1. **Monolitické jádro** – V monolitickém jádře všechny služby OS běží spolu s hlavním vláknem jádra a tedy i ve stejné oblasti paměti. To umožňuje neomezený a efektivní přístup k HW. Mnozí vývojáři zastávají názor, že monolitické systémy je jednoduší navrhnout i implementovat než ostatní řešení a jsou extrémně účinné, když jsou dobře napsané.

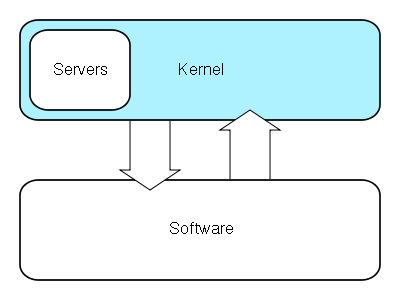
Hlavní nevýhodou je závislost mezi systémovými komponentami – chyba v libovolném ovladači zařízení může shodit celý systém (a je fakt, že velká jádra mohou být těžko udržovatelná).



1. **Mirko jádro** – Vykonává základní služby. Přístupem mikro jádra je definovaní jednoduché abstrakce hardware se soupravou primitivních funkcí nebo systémových volání implementující minimální služby OS jako je správa paměti nebo multitasking (IPC). Ostatní služby včetně té, které běžně poskytuje jádro, jsou realizovány v uživatelském prostoru. Mikro jádra jsou jednodušší než monolitická, avšak systémová volání způsobují řetězové změny kontextu, které mohou ve velkém množství zvýšit režii (nárok na systémové zdroje) systému tak, že budou pomalejší, než jednoduchá volání u monolitických jader.



1. **Hybridní jádro** – Snaží se zkombinovat rychlost a jednoduchost designu monolitického jádra s bezpečnostními výhodami mikro jader. Hybridní jádra jsou něco mezi monolitickým a mikro jádrem. To znamená, že některé služby (jako souborový systém nebo implementace…) běží v jádře ke zredukování režie proti mikro jádrům, ale jiné části monolitického jádra (ovladače zařízení) běží jako server v uživatelském prostoru.



1. **Monolitické jádro vs Mikro jádro** - S rostoucí velikostí jádra se objevuje množství problémů koncepce monolitického jádra. Jeden z nejvýraznějších je zvětšování paměťového otisku (množství paměti zabrané přímo jádrem). Virtuální paměť tento problém do určité míry odstraňuje, ale ne všechny počítačové architektury mají podporu virtuální paměti. Ke snížení velikosti jádra se musí vykonávat rozsáhlé úpravy k opatrnému odstranění nepatrného kódu, které mohou být velmi složité vzhledem k nejasné vzájemné závislosti mezi částí jádra s miliony řádků zdrojového kódu. Některé nevýhody monolitického jádra je možné odstranit používáním modulů: Ovladače zařízení stále běží v prostoru jádra se vší efektivitou, kterou to vynáší, ale jsou do značné míry nezávislé a je možné nahrávat jen ty, které jsou pro daný počítač zapotřebí. Příkladem takového systému je například Linux, FreeBSD.
2. **Nano jádro** - V Nano jádře jsou téměř všechny služby, dokonce i ty nejzákladnější jako správce přerušení nebo časovač – řešeno ovladači zařízení. Tím má vlastní jádro ještě menší požadavky na paměť než mikrokernel

# Vlákno

**Vlákno** (vlákno řízení, angl. thread) označuje v informatice odlehčený proces, pomocí jehož se snižuje režie OS při změně kontextu(K tomu aby mohl procesor vykonávat více operací, musí přesouvat pozornost od jednoho vlákna k druhému), které je nutné pro zajištění multitaskingu (zdánlivého běhu více úloh zároveň, který je zajištěn jejich rychlým střídáním na procesoru) nebo při masivních paralelních výpočtech. Zatím co běžné procesy jsou navzájem striktně odděleny, sdílí vlákna nejen společný paměťový prostor, ale i další datové struktury.

OS, který vlákna nepodporuje má technicky jedno vlákno na každý proces, zatím co podpoře vláken je možné v rámci jednoho procesu vytvořit mnoho vláken. Vlákna usnadňují díky sdílené paměti vzájemnou komunikaci, což však přináší další komplikace v podobě souběhu (race condition).

## Charakteristika Vlákna

Rozdílem mezi multithreadovým a multiprocesovým multitasingem je kromě sdílené paměti (která některé věci usnadňuje a jiné naopak ztěžuje) režie při přepínaní:

1. Přepnutí mezi vlákny bývá výrazně rychlejší, neboť vlákna sdílejí stejnou paměť a uživatelská práva svého mateřského procesu
2. Při přepínání vláken není nutné volat jádro (Vlákna na uživatelské úrovni)
3. Vytváření a rušení vláken je rychlejší než vytváření a rušení procesů
4. Vlákno spotřebuje méně paměti, což je důležité pro aplikace, které používají stovky nebo více vláken

Vlákna je možné vytvořit čistě na aplikační úrovni bez nativní podpory OS (využitím sdílené paměti a dalších technik). Takto vzniklá vlákna je poté možné spouštět postupně v jednotlivých procesech OS nebo takzvaně m:n, tedy ve vlákně OS současně spouštět větší počet aplikačních vláken. Toto řešení však není tak dobré jako s podporou OS – Například volání služby OS zablokuje „větší“ vlákno OS a ne jenom aplikační vlákna (ale pro některé úlohy může být stále rychlejší).

## Druhy Vláken

Vlákna na uživatelské úrovni (ULT)

Správu vláken provádí takzvaná vláknová knihovna (Thread library) na úrovni aplikačního procesu. Jádro OS o jejich existenci neví a přepínání mezi vlákny nevyžaduje volání funkcí jádra. Nepřepíná se ani kontext procesu ani režim procesoru. Přepínání vláken a jeho plánování je specifické pro konkrétní aplikaci. Programátor má možnost, zvolit si nejvhodnější plánovací strategii a příslušný algoritmus. V OS není vyžadováno multiprogramování, stačí speciální knihovna. Pro provozovaní vláken na uživatelské úrovni je potřeba speciální knihovna, která musí umožňovat:

* rušení a ukládaní vláken
* předávaní dat a zpráv mezi vlákny
* plánovaní běhu vláken
* uchovávaní a obnova kontextu vláken

**Výhody:**

* Nezávislost na podpore vláken v OS
* Přepínání mezi vlákny je nezávislé na OS
* Výrazně rychlejší tvorba a přepínaní vláken
* plná kontrola procesu nad správou běhu vláken

**Nevýhody:**

* Volaní služby jedním vláknem zablokuje všechna vlákna procesu
* Nutnost dodatečného programovaní (řízení vláken programátorem)

Vlákna na úrovni jádra (KLT)

Veškerá zpráva vláken je řízena OS a každé vlákno v uživatelském prostoru je zobrazeno na vlákno v jádře. Jádro samo vlákna vytváří, ruší a plánuje.

**Výhody:**

* Volání systému neblokuje ostatní vlákna téhož procesu
* Jeden proces může využít více procesorů
* Tvorba rušení mezi vlákny je levnější než mezi procesy
* I programy jádra mohou mít více vláknový charakter

**Nevýhody:**

* Správa je nákladnější než u čistě uživatelských vláken.

Příklady systému – Windows NT4-XP, Linux a BSD Unix v4.4

Vlákna na obojím

Některé OS podporují oba způsoby běhu vláken. Přiřazování uživatelských vláken k systémovým se děje automaticky (bez programátora) anebo programátor může nastavit počet vláken na úrovni jádra. Kombinace podporují od Windows XP+, FreeBSD v5+

## Kdy se vlákna používají

1. **Obsluha periférií**
   * U některých zařízení je potřeba periodicky testovat HW
   * Vláknu pak nemusí zbývat mnoho času na obsluhu uživatelského rozhraní
   * Jedno vlákno pro komunikaci s uživatelem a druhé obsluhuje HW
2. **Síťová komunikace** 
   * Jedno vlákno akceptuje výchozí komunikace
   * Jedno vlákno odesílá data
   * Jedno vlákno zpracovává data
3. **Vyvolávání dojmu rychlé odezvy programu**
   * Práce s velkým objemem dat uložených v DB
   * Hlavní vlákno pouze obsluhuje uživatelské rozhraní, další pracuje s DB
4. **Urychlení výpočtu**
   * Lze-li spustit na víceprocesorovém stroji kooperující vlákna na několika procesorech
5. **Vhodné pro architekturu aplikace**
   * Simulace – Jedno vlákno počítá vlastní simulaci
   * Další vlákno periodicky vzorkuje stav simulace a zobrazuje ho
   * Primární vlákno obsluhuje uživatelské rozhraní
6. **Efektivita** 
   * Některé aplikace jsou ze své podstaty nevhodná pro jedno vláknovou architekturu
   * Použití vláken může vést k výraznému zpřehlednění programového kódu
   * V moderním OS už beztak běží několik vláken – pár navíc nehraje roli. Každý OS má maximální strop na počet vláken kdy je plánování procesů je stále ještě efektivní.

## Vlákna je vhodné použít

Pokud aplikace splňuje některé z následujících kritérií

1. Je složená z nezávislých úloh
2. Může být blokována po dlouhou dobu
3. Obsahuje výpočetně náročnou část
4. Musí reagovat na asynchronní událost
5. Obsahuje úlohy
6. S nižší nebo vyšší prioritou než zbytek aplikace

## Výhody vláken obecně

Odezva

Vlákna můžou umožnit aplikaci zachování dobré – rychlé odezvy na uživatelský vstup. V programu s jedním vláknem se může stát, že hlavní vykonávaná činnost programu blokuje úlohu, která potřebuje dlouhou dobu na zpracování a může se zdát, že je celý program zamrzlý. Tím, že se pro dlouho trvající úlohu vytvoří pracovní vlákno, které poběží současně s hlavním vláknem programu, je možné zachovat rychlou odezvu prostředí programu a přitom provádět další operace na pozadí.

Rychlejší běh programu

Tato výhoda využití více vláken v programech umožňuje rychlejší průběh na počítačích s více CPU nebo s CPU o více jádrech. Je tomu tak díky kapacitě vícejaderných procesorů vlákna vykonávat skutečné souběžné operace.

Nižší spotřeba výkonu

S využitím vláken může aplikace spravovat více klientů souběžně při nižší spotřebě výkonu než by bylo nutné při opakovaném (duplicitním) spouštění téhož programu pro každého z klientů.

Lepší využití systému

Typický příklad lze demonstrovat na práci se systémem souborů. Při využití vláken lze dosáhnout vyššího datového toku a nižší odezvy přičemž data z rychlé paměti (paměť typu cache) mohou být získány jedním vláknem zatím, co jiné vlákno získává data z pomalejšího typu paměti, například externího disku, aniž by na sebe vlákna musela vzájemně čekat.

Zjednodušené sdílení a komunikace

Na rozdíl od procesů, které vyžadují předávání zpráv procesům nebo možnosti sdílení paměti k využití mezi procesové komunikace (IPC) mohou vlákna komunikovat skrze data, kód či soubory, které sdílí.

Paralelní běh

Aplikace, které chtějí využít výhodu procesoru s více jádry nebo systémy s více procesory mohou využít vlákna k rozdělení dat a úloh do paralelně podřízených úloh, které poté budou zpracovány souběžně na více jádrech (procesorech) daného systému. Výpočetní prostředí jako jsou CUDA na GPU využívají vláken díky velkému množství jader GPU, na kterých běží souběžně stovky úloh.

## Nevýhody vláken

1. Vlákna by se měla používat jen tam, kde je to opravdu nutné, protože tvorba vlákna je poněkud zdlouhavá operace. K vyřešení tohoto problému se u více vláknových aplikací používá mechanizmus sdružování vláken.
2. Každé vytvořené vlákno vytváří v paměti vlastní zásobník, do kterých jsou ukládány mezivýsledky, stavy proměnných, adresy apod.
3. I když je teoreticky možné vytvářet obrovský počet vláken, maximální počet vláken je omezen platformou.
4. Zvýšená složitost kódu programu
5. Daleko složitější je sledovat tok programu
6. Sdílení prostředků, za které je většinou zodpovědný programátor
7. K řízení sdílení prostředků se používá synchronizace, která zaručuje, že stav dat nebude změněn z více vláken zároveň

## Modely více vláknových aplikací

Modely řeší způsob vytváření a rozdělování práce mezi vlákna. Nejznámější jsou:

### BOSS/WORKER

Hlavní vlákno řídí rozdělení úlohy jiným vláknům – Hlavní vlákno je zodpovědné za vyřizování požadavků, pracuje v cyklu:

* Příchod požadavku
* Vytvoření vlákna pro řešení příslušného úkolu
* Návrat požadavku

Úkol je řízen: Příslušným vláknem řešící úkol a zároveň Hlavním vláknem

### PEER

Vlákna běží paralelně bez specifického vedoucího.

* Neobsahuje hlavní vlákno,
* První vlákno po vytvoření ostatních vláken:
  + se stává jedním z ostatních vláken (rovnocenným),
  + pozastavuje svoji činnost do doby, než ostatní vlákna končí.
* Každé vlákno je zodpovědné za svůj vstup a výstup

### Pipeline

Zpracování dat sekvencí operací (předpokládá dlouhý vstupní proud dat)

* Dlouhý vstupní proud
* Sekvence operací - Každá vstupní jednotka musí projít všemi částmi zpracování
* V každé části jsou v daném čase zpracované různé jednotky vstupů (nezávislost jednotek)

## Více vláknové aplikace a řazení

Hlavním problémem více vláknových informací souvisí se synchronizací, a jsou to:

1. **Uváznutí** – DeadLock
2. **Souběh** – Race conditions – přístup více vláken ke sdíleným proměnným a alespoň jedno vlákno nevyužívá synchronizačních mechanizmu. Vlákno čte hodnotu, zatím co jiné vlákno zapisuje. Zápis a čtení nejsou atomické a data mohou být neplatná.
3. **Vyhladovění** – Je to stav, kde jsou vláknu neustále odepírány prostředky. Bez těchto prostředků program nikdy nedokončí svůj úkol.

## Příklady více vláknových aplikací

**Servery** – obsluhují více klientů najednou. Obsluha typicky znamená přístup k několika sdíleným zdrojům a hodně I/O operací.

**Výpočetní aplikace** – Na víceprocesorovém systému lze výpočet urychlit rozdělením úloh na více procesových.

**Aplikace reálného času** – Lze využít specifických plánovačů.

Více vláknová aplikace je výkonnější než složité asynchronní programování, protože vlákno čeká na příslušnou událost namísto přerušovaných vykonávání kódu a přepínání kontextu.

## Podpora ze strany programovacích jazyků

Mnoho implementací jazyku C a C++ zajišťuje podporu vláken samostatně, ale taky zahrnují i přístup k nativnímu API vláken zajištěného OS. Některé programovací jazyky vyšší úrovně, často multiplatformní jako jsou JAVA, PYTHLON, .NET umožnují řešení vláken vývojáři aplikace a abstrahují specifika vláken podle dostupného systému za běhu programu.