Opakování na KIV/ZOS zápočet

Otázky teorie

- Co je to přerušení?
    - Přerušení (**Interrupt**) je signál pro procesor, který informuje, že došlo k **události vyžadující okamžitou pozornost**. Přerušením procesor pozastaví aktuálně prováděný program a **přejde k obsluze této události**, často spuštěním kódu tzv. obslužné rutiny přerušení (**ISR - Interrupt Service Routine**). Po dokončení obsluhy přerušení se procesor vrátí k původní činnosti
        - Hardwarové přerušení: Například když uživatel stiskne klávesu, myš pošle signál při pohybu, nebo se ukončí požadavek na čtení/zápis na disk.
        - Softwarové přerušení: Vytvořené příkazem v softwaru (např. INT instrukce v assembleru). Používá se pro vyvolání systémových volání.
- Co je to systémové volání (**System Call**)?
    - Mechanismus, kterým aplikace (uživatelský kód) žádá operační systém o provedení určité služby nebo operace na úrovni jádra (čtení/zápis do souboru, alokace paměti, vytvoření procesu, ...). Lze ho realizovat pomocí softwarových přerušení (na Linuxu INT 0x80 instrukce) a voláním systémových nebo knihovních funkcí.
    - Příklady volání:
        - `open()`, `read()`, `write()`, `close()`
        - `fork()`, `exec()`, `exit()`
        - `malloc()`, `free()` - funkce knihovny glibc, interně volají `brk()` nebo `mmap()`
        - `socket()`, `bind()`, `listen()`, `accept()`
        - `kill()`, `signal()`, `wait()`
        - `syscall()`, `ioctl()`, `mmap()`
- Jaké jsou různé typy přerušení a čím se od sebe liší?
    - Externí přerušení: Přichází z **vnějších zařízení**, např. klávesnice, myš nebo síťová karta.
    - Vnitřní přerušení (interní): Pochází z **procesoru nebo probíhajícího programu**, např. dělení nulou nebo pokus o přístup do neplatné paměti.
    - Softwarové přerušení: Vyvolané programem, často za účelem provedení systémového volání.
        - Výjimky (exceptions): Typ přerušení, které procesor sám vyvolá, když dojde k chybě, např. chybný přístup k paměti nebo dělení nulou.

- Jaký je rozdíl mezi maskovatelnými a nemaskovatelnými přerušeními?
    - Maskovatelná přerušení: Mohou být procesorem **ignorována** (tzv. "maskována") pomocí speciálního registrového příznaku. Procesor pak reaguje jen na nemaskovatelná přerušení, nebo až když tento příznak vypne.
        - Zahozená přerušení: Maskovatelná přerušení mohou být za určitých okolností ignorována nebo ztracena, pokud je procesor dlouhodobě nastaví na ignoraci, nebo pokud jsou přerušení rychlejší, než je může zpracovat.
    - Nemaskovatelná přerušení (NMI - Non-Maskable Interrupts, INT 0x02): Přerušení, která procesor **nemůže ignorovat**. Používají se pro kritické události, např. chyby hardwaru, které vyžadují okamžitou akci (např. problémy s napájením).
- Jaký je typický proces, který probíhá, když dojde k přerušení (na úrovni registrů - co přesně se kam ukládá a proč)?
    1. Uložení kontextu: Procesor uloží stav aktuálního programu, tj. obsah registrů (Program Counter, stavové registry - v podstatě registr CS:IP a FLAGS) do zásobníku nebo vyhrazeného místa v paměti.
    2. Přepne do privilegovaného režimu
    3. Zakáže přerušení (v registru FLAGS je IF – nastaví na zákaz přerušení)
    4. Nastavení ISR (Interrupt Service Routine): Procesor získá adresu ISR buď z tabulky přerušení nebo z pevného umístění, pokud se jedná o kritické přerušení.
    5. Obsluha přerušení: ISR zpracuje přerušení a provede potřebné kroky.
    6. Obnovení kontextu: Po dokončení ISR se původní stav obnoví, přejde se zpět do uživatelského režimu a procesor pokračuje v původním programu.
- CS:IP vs vrchol zásobníku:
    - CS:IP (obecně **Program Counter**) určuje, kde v paměti se nachází aktuální instrukce ke zpracování.
    - SS:SP/ESP/RSP (obecně **Stack Pointer**) určuje, kde v paměti leží vrchol zásobníku, tedy místo, kde jsou uložena poslední data.
- Rozdíl mezi asynchronními a synchronními událostmi
    - Asynchronní události: Jsou **neplánované** a mohou se vyskytnout **kdykoli** během programu, např. přerušení od hardwarového zařízení.
    - Synchronní události: Vyskytují se jako **přímý výsledek vykonávání programu**, např. výjimka vyvolaná dělením nulou.
- Rozdíly a použití INT, IRQ a dalších?
    - INT: **Softwarové přerušení**, které se používá k vyvolání obslužného kódu přerušení pomocí instrukce INT v assembleru. Například INT 0x80 je běžná instrukce v Linuxu pro volání systémových funkcí.
    - IRQ (Interrupt Request Line): Jedná se o **fyzické linky nebo signály používané hardwarovými zařízeními** k žádosti o pozornost procesoru. Každé zařízení má přiřazenu vlastní linku IRQ, např. klávesnice může mít IRQ 1, myš IRQ 12.

- K čemu slouží a co obsahuje tabulka vektorů přerušení?
    - Tabulka vektorů přerušení (**Interrupt Vector Table**, IVT) je struktura (mapa) **obsahující adresy příslušných obslužných rutin** pro určité přerušení.
    - Procesor přistupuje k tabulce vektorů přerušení na základě čísla přerušení (vektor přerušení - INT ...), které získá při každém přerušení.
        - Vektory 0–31 jsou rezervovány pro výjimky a systémové chyby (např. dělení nulou).
        - Vyšší vektory (např. 32 a výše) se používají pro hardwarová a softwarová přerušení.
- Umístění tabulky vektorů přerušení v systému
    - Na procesorech x86 bývá tabulka často umístěna na adrese **0x0000:0000** v uživatelském (user) režimu.
    - V chráněném (privileged) režimu si může systém definovat umístění tabulky podle potřeby a **adresu tabulky uloží v registru IDTR** (Interrupt Descriptor Table Register).
- Co se děje při dělení nulou, neplatné instrukci, výpadku stránky paměti?
    - Dělení nulou:
        1. Detekce: Procesor rozpozná dělení nulou během provádění instrukce dělení. Každý procesor je navržen tak, aby kontroloval dělení nulou před provedením této operace.
        2. Reakce procesoru: Při zjištění dělení nulou procesor vyvolá výjimku (INT 0x00). Procesor tím přeruší aktuální běh programu a předá kontrolu obslužné rutině výjimky dělení nulou.
        3. Reakce operačního systému: Operační systém zpravidla ukončí proces, který tuto operaci vyvolal, aby zabránil šíření chyb. Může také vrátit specifickou chybovou zprávu nebo log do systémového záznamu, aby administrátor či vývojář mohl problém analyzovat.
    - Neplatná instrukce:
        - Neplatná instrukce nastane, když procesor narazí na neplatný nebo architekturou nepodporovaný kód instrukce.
            - Spouštění instrukcí z neinicializované/zkorumpované paměti.
            - Pokus o spuštění nepodporovaných nebo potenciálně škodlivých instrukcí.
            - Bug nebo chyba v kompilátoru.
        - Obsluha principielně stejná jako u dělení nulou.
    - Výpadek stránky paměti:
        - Výpadek stránky nastává, když se virtuální adresa, na kterou program odkazuje, nenachází v aktuální fyzické (RAM) paměti. Tato data mohou být na disku nebo mohou být ještě nezaplněná (tzv. "null page").
        - Procesor detekuje výpadek stránky pomocí MMU (Memory Management Unit), která kontroluje, zda je požadovaná stránka dostupná. Pokud není, vyvolá výjimku výpadku stránky.
            1. Vyhledání dat na disku (např. v souboru stránkování).
            2. Načtení stránky do paměti.
            3. Aktualizaci tabulky stránek, aby odkazovala na správnou fyzickou adresu.

- Privilegovaná instrukce?
    - Privilegovaná instrukce je typ instrukce, která může být vykonána pouze v privilegovaném režimu procesoru, často označovaném jako režim jádra nebo supervisor mode.
    - Příklady:
        - Instrukce pro manipulaci s hardwarovými prostředky (instrukce pro zapnutí nebo vypnutí přerušení, ...).
        - Nastavení tabulky stránek (tabulka překladu virtuálních adres na fyzické adresy).
        - Nastavení systémového časovače.
- Čím se liší monolitický OS a OS založený na mikrojádře (včetně struktury, spravování prostředků, atd.)?
    - Monolitický OS je postavený tak, že všechny základní služby a moduly **běží v jednom velkém jádře** (kernel), které má plný přístup k hardwaru a systémovým prostředkům. Tento typ OS obsahuje jádro, které zahrnuje různé služby, jako jsou správa paměti, správa souborového systému, správa procesů a zařízení, přímo v jádře.
    - Struktura:
        - Jedno velké jádro: Všechny základní systémové funkce a ovladače jsou součástí jediného jádra, což znamená, že se veškeré operace systému provádí v režimu jádra (kernel mode).
        - Jednotná paměť: Všechny komponenty jádra sdílejí jednu paměťovou oblast, což usnadňuje komunikaci mezi moduly (například správa paměti a ovladače zařízení).
    - Správa prostředků
        - Přímý přístup: Všechny komponenty monolitického jádra mají přímý přístup ke všem prostředkům a mohou je přímo ovládat.
        - Rychlá komunikace: Protože všechny služby běží v rámci stejného adresního prostoru, je komunikace mezi moduly rychlá a efektivní.
    - Výhody:
        - Výkon: Monolitický OS je zpravidla rychlejší, protože nedochází k nutnosti přepínání kontextů při komunikaci mezi moduly (jsou všechny v jádře).
        - Efektivní komunikace: Všechny moduly jsou přímo propojené, což umožňuje rychlou komunikaci a spolupráci.
    - Nevýhody:
        - Stabilita: Pokud se v jedné části jádra objeví chyba (například v ovladači zařízení), může spadnout celý systém.
        - Bezpečnost: Protože všechny moduly sdílí jeden adresní prostor, chyba nebo zranitelnost v jednom modulu může ohrozit celé jádro.

- OS založený na mikrojádře má **jádro velmi malé a základní**, které obsahuje pouze ty **nejzákladnější funkce**, jako je správa procesů, základní komunikace mezi procesy (IPC - Inter-Process Communication) a základní správu paměti. Většina ostatních služeb (např. správa souborového systému, ovladače zařízení) běží mimo jádro v uživatelském prostoru.

- Struktura:

  - Malé jádro (mikrojádro): Mikrojádro obsahuje pouze nezbytné funkce, jako je přepínání procesů, základní správa paměti a IPC.
  - Modulární struktura: Další služby jako ovladače, správa souborů nebo síťové protokoly běží jako samostatné procesy v uživatelském režimu.

- Správa prostředků:

  - Modulární přístup: Každá služba, jako jsou ovladače zařízení nebo souborový systém, běží jako samostatný proces v uživatelském režimu. S jádrem komunikuje přes mechanismus IPC.
  - Zabezpečená komunikace: Komunikace mezi jádrem a ostatními službami probíhá přes zprávy. Každý modul má vlastní adresový prostor, což omezuje dopad chyb a zvyšuje bezpečnost.

- Výhody:

  - Stabilita: Protože jednotlivé služby běží odděleně, pád jednoho modulu nezpůsobí pád celého systému.
  - Bezpečnost: Chyby a zranitelnosti v jednotlivých modulech mají menší dopad, protože každý modul běží v odděleném paměťovém prostoru.
  - Modularita a flexibilita: Služby lze snadno upravovat nebo aktualizovat, aniž by bylo nutné přepracovat celé jádro.

- Nevýhody:

  - Výkon: Mikrojádra bývají pomalejší, protože dochází k přepínání mezi kontexty při komunikaci mezi moduly v uživatelském a jádrovém režimu.
  - Složitost komunikace: Protože většina systémových funkcí probíhá mimo jádro, systém je závislý na efektivní IPC, což může být obtížné a časově náročné.

- IPC:
    - IPC (**Inter-Process Communication**), tedy meziprocesová komunikace, označuje mechanismy, které umožňují procesům v operačním systému vzájemně komunikovat a sdílet data. Procesy mohou běžet na stejném systému nebo v různých prostředích (například ve více vláknech či kontejnerech)
    - Mechanismy:
        - Sdílená paměť (**Shared Memory**)
        - Fronty zpráv (**Message Queues**)
            - Posílání zpráv mezi procesy
        - Signály (**Signals**)
        - Pojmenované a nepojmenované roury (**Pipes**)
            - Po streamech
            - Nepojmenované jenom mezi rodičem a potomkem
        - Sokety (**Sockets**)
            - Síťová komunikace
        - Paměťové mapování souborů (**Memory-Mapped Files**)
            - Sdílení velkých souborů mezi procesy
        - Sychronizační primitiva (semafory, zámky, ...)
- Co je to kritická sekce, souběh?
    - Kritická sekce je část kódu, která přistupuje k **sdíleným prostředkům** (například proměnným, paměti nebo souborům), ke kterým může mít současně přístup více vláken nebo procesů. Vzhledem k tomu, že více procesů může běžet současně, může dojít k kolizím a nekonzistentním výsledkům, pokud by dva nebo více procesů přistupovaly ke sdíleným prostředkům ve stejnou dobu.
        - Je potřeba zajistit synchronizaci přístupu k těmto prostředkům (např. pomocí zámků, semaforů nebo podmíněných proměnných), aby se zabránilo chybám a nekonzistencím dat.
    - Souběh nastává, když více procesů nebo vláken **běží zdánlivě současně** a přistupují ke **sdíleným zdrojům**.
        - Umožňuje systému efektivněji využívat prostředky a zpracovávat více úloh současně, což je užitečné v prostředích s více jádry nebo v distribuovaných systémech.

- Co je nevýhodou aktivního čekání?
    - Aktivní čekáni je v zásadě smyčka, která neustále **aktivně se pokouší pokračovat**, ale zatím nemůže (kontrolování podmínky, ...). Může **zbytečně zatěžovat procesor**, který by mohl být využit pro jiné úkoly.
    - Zatěžování procesoru
    - Zvýšená spotřeba energie
    - Potenciální zpomalování systému
    - Možnosti řešení:
        - Blokovací synchronizace: Místo aktivního čekání se vlákno může zablokovat (například použitím semaforu nebo mutexu) a čekat, dokud nebude zdroj dostupný. Procesor pak může mezitím vykonávat jiné úkoly.
        - Podmínkové proměnné: Vlákna mohou být uspána, dokud se nesplní konkrétní podmínka, a procesor je probudí, když je kritická sekce volná.
        - Manuální uspání: Vlákno může být periodicky uspáno a probuzeno, aby se snížila zátěž procesoru.
- Čím se liší dávkové a interaktivní systémy?
    - Dávkové systémy (batch systems) jsou navrženy tak, aby zpracovávaly úlohy, které jsou **seskupeny do dávek** (anglicky "batch") a **spuštěny sekvenčně** nebo v naplánovaném pořadí **bez přímé interakce s uživatelem** během jejich vykonávání. Uživatelé obvykle připraví úlohy předem a pošlou je do systému ke zpracování. Po odevzdání úloh uživatel nečeká na okamžitý výsledek, ale výsledky se mu vrátí po dokončení celé dávky.
        - Bez interakce uživatele během zpracování
        - Zpracování podle pořadí nebo priority
        - Maximální využití prostředků
        - Zpracování velkých datových souborů; vědecké výpočty; automatizované reporty; ...
        - Výhody: Efektivní využití prostředků, snadné plánování a řízení úloh, minimalizace nečinnosti systému, vhodné pro velké výpočetní úlohy.
        - Nevýhody: Omezená interaktivita, obtížnější ladění a testování, obtížnější reakce na chyby nebo neočekávané situace.
    - Interaktivní systémy jsou navrženy tak, aby umožňovaly **přímou interakci s uživatelem**, což znamená, že uživatelé mohou zadávat příkazy nebo požadavky a očekávat okamžitou nebo **rychlou odezvu od systému**. Tyto systémy umožňují uživateli interaktivně zadávat příkazy, upravovat data nebo kontrolovat průběh úloh v reálném čase.
        - Přímá interakce uživatele
        - Rychlá odezva
        - Zpracování v reálném čase
        - Operační systémy pro osobní počítače; mobilní telefony; DB systémy; webové aplikace, ...
        - Výhody: Rychlá odezva, interaktivní uživatelské rozhraní, vhodné pro osobní počítače a mobilní zařízení.
        - Nevýhody: Nižší efektivita využití prostředků, obtížnější plánování a řízení úloh, vyšší nároky na výkon a odezvu systému (kvůli interaktivitě a odezvě).

- Co se dodržuje u realtimových systémů?
    - Realtimové systémy jsou navrženy tak, aby **dodržovaly časové limity - deadliny**. Tyto systémy musí být schopny reagovat na události v reálném čase a vykonávat úlohy v předem definovaných časových intervalech.
    - Hard real-time: Systém musí splnit časový limit, jinak může dojít k závažným následkům (např. havárie).
    - Soft real-time: Systém by měl dodržet časový limit, ale jeho porušení nemusí mít fatální následky.
- Preemptivní:
    - Preemptivní znamená, že proces, který právě běží na procesoru, **může být nuceně přerušen** operačním systémem, aby mohl být procesor přidělen jinému procesu. Toto přerušení (tzv. preempce) je klíčovou vlastností mnoha moderních operačních systémů a plánovacích algoritmů, která umožňuje efektivní správu procesů a zdrojů.
    - Příklady preemtivních plánování:
        - Round Robin (RR)
        - Shortest Remaining Time (SRT)
        - Priority Scheduling (s preempcí)
        - Multilevel Queue Scheduling (s preempcí)
- Nepreemptivní:
    - Nepreemptivní znamená, že proces, který běží na procesoru, **nemůže být nuceně přerušen** operačním systémem, dokud nedokončí svou práci nebo se sám nevzdá procesoru. To znamená, že proces může být na procesoru po delší dobu, což může vést k zablokování nebo zpomalení systému.
    - Příklady nepreemptivních plánovacích algoritmů:
        - First-Come, First-Served (FCFS)
        - Shortest Job Next (SJN)
        - Priority Based Scheduling (bez preempce)
        - Multilevel Queue Scheduling (bez preempce)
- MFT:
    - MFT (**Multiprogramming with a Fixed number of Tasks**) je typ plánovače používaný v dávkových systémech (batch systems), kde je paměť rozdělena na pevný počet částí (partitions) s pevnou velikostí. Každá část paměti je přidělena jednomu procesu. Plánování v tomto prostředí se zaměřuje na efektivní využití dostupné paměti a CPU.
    - Upřednostňuje I/O vázané operace.

- Příklad uvíznutí a vyhladovnění?
    - Starvation (vyhladovnění) je obecná situace, kdy jeden nebo více procesů nebo vláken **nedostávají dostatečné prostředky** (např. CPU čas, paměť, přístup k datům, ...) k dokončení své práce. Procesy/vlákna jsou proto **blokována nebo zpožděna** a nemohou pokračovat.
    - Deadlock (uvíznutí) je situace, kdy dva nebo více procesů nebo vláken jsou **blokovány** a **čekají na prostředky**, které si navzájem drží (dochází k tomu když jsou splněny všechny Coffmanovy podmínky). Žádný z procesů nemůže pokračovat, protože každý z nich čeká na uvolnění prostředků, které drží druhý proces.
    - Livelock je situace, kdy dva nebo více procesů nebo vláken jsou blokovány, ale **stále aktivně reagují** na situaci a snaží se vyřešit problém. Výsledkem je, že se procesy nebo vlákna neustále snaží vyřešit konflikt, ale tím opět blokují jeden druhého a nedojde k pokroku.
        - Příklad: Dva procesy, které se snaží projít dveřmi, ale každý z nich ustoupí, když vidí, že druhý proces se snaží projít, což vede k tomu, že se nikdo nedostane dál. Toto opakují stále dokola, aniž by se někdo dostal skrz dveře.

- Co jsou Coffmanovy podmínky?
    - Podmínky, které když jsou splněny, tak může nastat **deadlock**.
    - Vzájemné vyloučení (**Mutual Exclusion**)
        - Popis: Určité prostředky (např. tiskárna, soubor, paměťový blok) mohou být přiděleny pouze jednomu procesu/vlákno v daném okamžiku. Jinými slovy, prostředek nemůže být sdílen mezi více procesy, dokud jej proces, který jej drží, neuvolní.
        - Příklad: Pokud proces A používá tiskárnu, proces B ji nemůže použít, dokud ji A nedokončí a neuvolní.
    - Držení a čekání (**Hold and Wait**)
        - Popis: Proces, který již drží jeden nebo více prostředků, žádá o další prostředky, které jsou aktuálně přiděleny jiným procesům. Proces přitom nezpřístupní prostředky, které již drží, dokud nezíská všechny požadované prostředky.
        - Příklad: Proces A drží prostředek X a čeká na prostředek Y. Proces B drží prostředek Y a čeká na prostředek X. Ani jeden z procesů neuvolní prostředek, který drží, což vede k cyklickému čekání.
    - Neodnímatelnost (**No Preemption**)
        - Popis: Prostředky nemohou být nuceně odebrány procesu, který je drží. Může je uvolnit pouze proces, který je drží, a to až poté, co dokončí svou práci.
        - Příklad: Pokud proces A drží tiskárnu, systém ji nemůže přidělit procesu B, dokud ji proces A sám neuvolní.
    - Cyklické čekání (**Circular Wait**)
        - Popis: Existuje cyklus procesů, kde každý proces čeká na prostředek, který drží jiný proces v tomto cyklu.
        - Příklad: Proces A čeká na prostředek, který drží proces B, proces B čeká na prostředek, který drží proces C, a proces C čeká na prostředek, který drží proces A. Tím vznikne cyklické čekání.

- Co je to RoundRobin plánování?
    - Plánovací algoritmus, který rozhoduje o **přidělení času procesoru** jednotlivým procesům v **cyklickém pořadí**. Každý proces dostane časový úsek (kvantum) k běhu, po jehož uplynutí je proces přesunut na konec fronty a je přidělen čas jinému procesu. Pokud proces dokončí svou práci dříve, než uplyne jeho kvantum, je vyřazen z fronty a uvolní místo pro další procesy.
- Jak se rozšíří RoundRobin když potřebuji priority? (více front dle priorit, RR v rámci fronty)
    - Použít přístup multilevel queue scheduling (plánování s více frontami), kde každý proces je zařazen do fronty podle své priority.
        - Fronta s vysokou prioritou
        - Fronta se střední prioritou
        - Fronta s nízkou prioritou
    - Round Robin v rámci každé fronty.
    - Plánovač nejprve obslouží procesy z fronty s nejvyšší prioritou. Teprve když je fronta s vyšší prioritou prázdná, začne plánovač vykonávat procesy z fronty s nižší prioritou.
    - Předbíhání nižších priorit (preemption): Pokud se během vykonávání procesu z nižší prioritní fronty objeví nový proces ve frontě s vyšší prioritou, systém přeruší (předběhne) proces z nižší fronty a začne vykonávat proces z vyšší fronty.
    - Další možnosti rozšíření:
        - Variabilní délka kvanta na základě priority
        - Změna priority procesu během běhu
    - Výhody:
        - Procesy s vyšší prioritou jsou upřednostněny, což zlepšuje odezvu důležitých úloh.
        - Round Robin zajišťuje spravedlivé sdílení času procesů na stejné prioritní úrovni.
    - Nevýhody:
        - Procesy s nízkou prioritou mohou trpět vyhladověním, pokud systém často přijímá procesy s vysokou prioritou.
        - Systém je složitější na implementaci, protože je nutné spravovat několik front a přepínání mezi nimi.

- Jaké jsou datové struktury a základní operace semaforu, monitoru, mutexu?
  - Semafor:
    - Synchronizační mechanismus, který používá celočíselnou proměnnou pro **sledování počtu dostupných prostředků**. Může být binární (podobný zámku) nebo počítaný (umožňuje více vláken přistupovat ke sdílenému prostředku). Procesy/vlákna, která nemohou získat semafor, jsou uložena ve frontě čekajících.
    - Základní operace:
      - `P()` (wait/probe):
        - Snižuje hodnotu semaforu o 1 (vpouští 1 proces do kritické sekce).
        - Pokud je výsledná hodnota semaforu < 0, proces nebo vlákno se zablokuje a je zařazeno do fronty čekajících.
      - `V()` (signal):
        - Zvyšuje hodnotu semaforu o 1 (značí, že 1 proces opouští kritickou sekci).
        - Pokud je v frontě čekajících proces, probudí jeden z nich.
      - V C:
        - `sem_init()`: Inicializace semaforu.
        - `sem_wait()`: `P()` operace.
        - `sem_trywait()`: Pokus o `P()` operaci bez blokování.
        - `sem_post()`: `V()` operace.
        - `sem_destroy()`: Zrušení semaforu.
      - V Javě:
        - `acquire()`: `P()` operace.
        - `tryAcquire()`: Pokus o `P()` operaci bez blokování.
        - `release()`: `V()` operace.
      - Základní stuktury:
        - Počítadlo: Celé číslo, které určuje počet dostupných prostředků.
        - Fronta čekajících: Fronta procesů nebo vláken, které čekají na uvolnění semaforu.
        - Zámek: Zajišťuje vzájemné vyloučení pro kritické sekce.
  - Monitor:
    - Vyšší synchronizační abstrakce, která kombinuje **vzájemné vyloučení** (mutual exclusion) a **podmínkové proměnné** (condition variables) v rámci jedné struktury.
    - Základní operace:
      - `wait()`: Proces se pozastaví a čeká na splnění podmínky. Interní zámek je uvolněn, aby ostatní procesy mohly pokračovat.
      - `signal()`: Probudí jeden z procesů čekajících na podmínkové proměnné. Pokud žádný proces nečeká, operace nemá efekt.
      - `broadcast()`: Probudí všechny procesy čekající na podmínkové proměnné. Ty by správně měly zkontrolovat podmínkovou proměnnou a rozhodnout, zda mohou pokračovat.
    - V C:
      - Nutné implementovat pomocí zámků a podmínkových proměnných.

- V Javě:
  - Pomocí klíčového slova `synchronized`.
  - `syncronized` blok: Zajišťuje vzájemné vyloučení pro kritické sekce.
  - `wait()`: Čeká na splnění podmínky.
  - `notify()`: Probudí jeden z čekajících procesů.
  - `notifyAll()`: Probudí všechny čekající procesy.
- Základní struktury:
  - Podmínková proměnná: Slouží k synchronizaci mezi procesy a k čekání na splnění určité podmínky.
  - Zámek: Zajišťuje vzájemné vyloučení pro kritické sekce.

- Mutex:
  - Mutex (**mutual exclusion**) je synchronizační primitivum, které umožňuje pouze jednomu vláknu najednou přístup ke sdílenému prostředku. Na rozdíl od semaforu má mutex koncept vlastnictví – vlákno, které mutex zamkne, ho musí také odemknout.
  - Základní operace mutexu:
    - Zamknutí mutexu:
      - Vlákno nebo proces, které chce vstoupit do kritické sekce, musí mutex zamknout. Pokud je mutex již zamčen, vlákno čeká, dokud se mutex neuvolní.
    - Odemknutí mutexu:
      - Po dokončení kritické sekce vlákno odemkne mutex, aby ho mohlo použít jiné vlákno.
    - Zkouška zamčení:
      - Některé implementace umožňují pokusit se zamknout mutex bez čekání.
    - V C:
      - `pthread_mutex_init()`: Inicializace mutexu.
      - `pthread_mutex_lock()`: Zamknutí mutexu.
      - `pthread_mutex_trylock()`: Pokus o zamknutí mutexu bez blokování.
      - `pthread_mutex_unlock()`: Odemknutí mutexu.
      - `pthread_mutex_destroy()`: Zrušení mutexu.
    - V Javě (pomocí `ReentrantLock`):
      - `lock()`: Zamknutí mutexu.
      - `tryLock()`: Pokus o zamknutí mutexu bez blokování.
      - `unlock()`: Odemknutí mutexu.
  - Základní struktury mutexu:
    - Zámek:
      - Mutex funguje jako binární zámek, který může být buď zamčený (obsazený), nebo odemčený (volný).
    - Vlastnictví (v některých implementacích):
      - V některých systémech, jako Java ReentrantLock, mutex podporuje sledování vlákna, které ho vlastní, a zajistí, že pouze toto vlákno může mutex odemknout.

- Čím se liší mutex a semafor?
    - Mutex dovoluje pouze jednomu vláknu nebo procesu **přistupovat ke sdílenému prostředku**. Proces/Vlákno, které zamkne mutex, ho musí také odemknout.
    - Semafor dovoluje **více vláknům nebo procesům** přistupovat ke sdílenému prostředku (pokud není binární). Operace semaforu mohou procesy/vlákna používat kdykoliv, bez ohledu na to, který proces je vlastníkem semaforu (nutnost k zajištění mnohonásobného přístupu).
- Hoareho monitor:
    - Pokud vlákno zavolá funkci signal (probuzení čekajícího vlákna na podmínkové proměnné), **probouzené vlákno získá okamžitě kontrolu nad monitorem**. Vlákno, které volalo signal, je pozastaveno, umístěno do signal fronty a čeká, dokud probuzené vlákno dokončí svou činnost v monitoru.
    - Výhody:
        - Model zaručuje silnější konzistenci: při probuzení vlákna je stav monitoru přesně takový, jaký probouzené vlákno očekává.
        - Jednodušší analyzovatelnost a odvození správnosti, protože programátor nemusí uvažovat o změnách stavu monitoru mezi signal a návratem k běhu vlákna.
    - Nevýhody:
        - Vyšší nároky na plánovač a implementaci monitoru, protože signal musí přepnout kontext okamžitě.
- Hansenovo monitor:
    - Pokud **vlákno zavolá signal**, tak **pokračuje v běhu**. **Probuzené vlákno je pouze přesunuto do fronty čekajících vláken** (ready queue) a bude naplánováno později.
    - Signal musí být jako poslední volání v monitoru.
    - Výhody:
        - Jednodušší implementace, protože vlákno nemusí přerušit svůj běh ihned po volání signal.
        - Lepší výkon v situacích, kdy se vlákno po signal rychle dokončí a nezabírá zbytečně CPU čas.
    - Nevýhody:
        - Stav monitoru může být změněn jinými vlákny předtím, než probuzené vlákno získá kontrolu nad monitorem.
        - Programátor musí pečlivě zajistit, že probuzené vlákno bude vždy očekávat stav, který nemusí být přesně takový, jaký byl při volání signal (je více omezen).

- Co znamená pojem dvojí kopírování při výměně zpráv mezi procesy?
    - Označuje mechanismus meziprocesové komunikace (IPC - Inter-Process Communication), při kterém jsou data mezi procesy přenášena pomocí **dvou kroků kopírování**.
        - První kopírování (**odesílatel → jádro**):
            - Data, která odesílající proces chce předat, jsou zkopírována z jeho uživatelského prostoru (user space) do jádrového prostoru (kernel space).
            - Tato operace zajišťuje, že jádro operačního systému má kontrolu nad daty a může je bezpečně spravovat.
        - Druhé kopírování (**jádro → příjemce**):
            - Data jsou z jádrového prostoru zkopírována do uživatelského prostoru příjemce.
            - Tento krok předává zprávu procesu, který ji má přijmout, aniž by měl přímý přístup k paměti odesílatele.
- Jaké výhody a nevýhody má randes-vous oproti dvojímu kopírování?
    - Randes-vous je synchronní metoda komunikace, kde odesílající i přijímající proces musí být **současně připraveny k výměně dat**. K přenosu zpráv obvykle dochází přímo, bez nutnosti mezibufferu v jádře operačního systému.
    - Výhody:
        - Rychlejší komunikace: Randes-vous může být rychlejší než dvojí kopírování, protože data jsou přenášena přímo mezi procesy bez nutnosti kopírování do jádra.
        - Snížená režie: Randes-vous může snížit režii spojenou s kopírováním dat mezi uživatelským a jádrovým prostorem.
        - Vhodnější pro malá data: Randes-vous může být vhodnější pro přenos malých datových bloků, kde je režie kopírování v jádře zbytečná.
    - Nevýhody:
        - Synchronizace: Randes-vous vyžaduje synchronizaci mezi odesílatelem a příjemcem, což může být náročné v případě, že procesy nejsou synchronizovány.
        - Blokování: Pokud jeden z procesů není připraven k výměně dat, je nutné čekat, což může vést k blokování a ztrátě výkonu.
- Co je to IPC?
    - IPC (Inter-Process Communication) je obecný termín pro **komunikaci mezi procesy** nebo vlákny v rámci operačního systému. Slouží k výměně dat, synchronizaci, sdílení prostředků a koordinaci mezi různými procesy nebo vlákny.
- Co je to proces?
    - Instance běžícího programu.
    - Má PID, vyžaduje čas na CPU, zabírá paměť.
    - Každý proces má svůj vlastní adresní prostor.
    - Každý nový proces má i vlastní **hlavní vlákno**, které vykonává kód programu.

- Co je to PCB?
    - PCB (**Process Control Block**) je datová struktura v jádře operačního systému, která obsahuje informace o aktuálním stavu procesu.
    - Obsahuje:
        - Identifikátor procesu (PID).
        - Stav procesu (běžící, čekající, ukončený).
        - Registrace CPU (hodnoty registrů při přepínání kontextu).
        - Informace o paměti (tabulky stránek, segmenty).
        - Otevřené soubory a I/O zařízení.
        - Priority a plánovací informace.
- Co je to PCT?
    - PCT (**Process Control Table**) je tabulka spravovaná operačním systémem, která uchovává informace o všech PCB.
    - Obsahuje:
        - Seznam všech procesů a jejich PCB.
        - Slouží k rychlému vyhledávání a správě procesů.
        - Umožňuje efektivní plánování, přepínání kontextu a řízení životního cyklu procesů.
- Co je to TLB?
    - TLB (**Translation Lookaside Buffer**) je speciální hardwarová cache v procesoru, přesněji jeho MMU, která má za úkol urychlit a v některých architekturách vůbec umožnit překlad virtuálních adres na fyzické.
- Co je to MMU?
    - MMU (**Memory Management Unit**) je hardwarová jednotka v procesoru, která se stará o překlad virtuálních adres na fyzické adresy a řízení přístupu k paměti.
- Co je to virtuální paměť?
    - Virtuální paměť je technika operačního systému, která umožňuje procesům pracovat s pamětí, která je větší než fyzická paměť počítače. Procesy vidí paměť jako spojitý lineární adresní prostor, který je rozdělen na stránky a mapován na fyzickou paměť pomocí MMU.
- Co je to vlákno?
    - Část procesu, která může být vykonávána paralelně s jinými částmi procesu
    - Sdílí paměť s ostatními vlákny v rámci procesu
    - Sdílí PID s ostatními vlákny v rámci procesu

- Stručné definice pojmů spjatých s disky a souborovými systémy:
  - Filesystem: Struktura pro organizaci a ukládání dat na disku (např. ext4, NTFS, FAT32).
  - Partition (oddíl): Logicky oddělená část disku, která může obsahovat jeden souborový systém.
  - Sector: Nejmenší fyzická jednotka pro ukládání dat na disku, obvykle o velikosti 512 bajtů nebo 4 KB.
  - Cluster/Block: Logická jednotka ukládání dat, skládající se z jednoho nebo více sektorů, používaná souborovým systémem.
  - Partition Table: Tabulka popisující rozdělení disku na oddíly, například MBR nebo GPT.
  - MBR (Master Boot Record): Tradiční struktura pro správu oddílů na disku, omezená na 4 primární oddíly a 2 TB.
  - GPT (GUID Partition Table): Moderní struktura pro správu oddílů, podporuje větší disky a více oddílů než MBR.
  - Bootloader: Program uložený na začátku disku, který inicializuje operační systém při startu.
  - Mountpoint: Místo v adresářovém stromu, kde je připojen oddíl nebo zařízení.
  - Swap Space: Prostor na disku používaný jako dočasné rozšíření operační paměti (RAM).
  - Logical Volume: Flexibilní oddíl vytvořený pomocí LVM, který může být dynamicky zvětšován nebo zmenšován.
  - RAID (Redundant Array of Independent Disks): Technologie pro spojení více disků do jednoho logického celku pro zvýšení výkonu nebo redundance.
  - Sector Alignment: Zarovnání oddílů na fyzické sektory disku pro optimalizaci výkonu.
  - Disk I/O: Čtení a zápis dat mezi operačním systémem a úložným zařízením.
  - Partition Scheme: Metoda rozdělení disku (např. MBR nebo GPT), určující, jak jsou data a oddíly organizovány.
  - Defragmentace: Proces optimalizace disku, který přeskupuje data tak, aby byla uložena v kontinuálních blocích a zvýšila se rychlost čtení a zápisu.

Otázky praxe

- Jakým příkazem si vypíšu běžící procesy?
  - `$ ps aux`
  - `$ top`
- Jak vypíšu login přihlášeného uživatele na daném terminálu?
  - `$ whoami`
  - `$ who | grep <název terminálu>` (who | grep tty1)
  - `$ w | grep <název terminálu>` (w | grep tty1)
- Jak vypíšu druhou až pátou řádku ze souboru s1.txt?
  - `$ head -n 5 s1.txt | tail -n 4`
  - `$ sed -n '2,5p' s1.txt`
  - `$ awk 'NR >= 2 && NR <= 5' s1.txt`
- Co je uloženo v /etc/passwd a co v /etc/shadow?
  - /etc/passwd: základní informace o uživatelských účtech v systému.
    - `username:x:UID:GID:gecos:home_directory:shell`
      - x je placeholder pro hesla, která se zde již neukládají oproti starým Unix verzím.
  - /etc/shadow: obsahuje hashem zabezpečené informace o heslech
    - `username:hashed_password:last_change:min:max:warn:inactive:expire`

- Jak funguje nastavení přístupových práv pomocí příkazu chmod?
  - Práva: r - read, w - write, e - execute
  - Práva pro u - user, g - group, o - others (kdokoliv jiný), a - all
  - Symbolický způsob nastavení práv
    - Přístupová práva lze nastavovat symbolicky pomocí kategorií (u, g, o, a) a operátorů:
    - Operátory:
      - +: Přidání práva.
      - -: Odebrání práva.
      - =: Nastavení přesně zadaných práv (ostatní práva se odeberou).
    - `$ chmod g+r soubor.txt`
    - `$ chmod u=rwx,g=,o= soubor.txt`
    - `$ chmod a+x soubor.txt`
  - Numerický (oktální) způsob nastavení práv
    - Každé právo je reprezentované 3 bitovým číslem (tzn. dekadicky trojčíslí, kdy první číslice reprezentuje práva vlastníka, druhá skupiny a třetí ostatních).

Vlastník	Skupina	Ostatní
u	g	o
rwx	rwx	rwx
– – –	– – –	– – –
1 1 1	1 0 0	1 0 0

    - např. 111 100 100 (vlastník vše, group a ostatní pouze čtení)
      - po převodu všech 3 bitů do decimálu => 7 4 4
      - `$ chmod 744 soubor.txt`
        - ekvivalent k `$ chmod u=rwx,g=r,o=r soubor.txt`
  - Argumenty:
    - -R
      - rekurzivně včetně podadresářů
    - --reference=soubor1.txt soubor2.txt
      - Nastav podle soubor1.txt

- Co dělá ls -i a ls -al?
  - `$ ls -i`
    - Zobrazí inode číslo každého souboru nebo adresáře vedle jeho názvu
  - `$ ls -al`
    - Zobrazí všechny soubory v aktuálním adresáři s podrobnými informacemi

- Co udělá echo $2 v příkazovém skriptu?
  - Vypíše druhý argument z příkazové řádky

- Jak vypsat návratovou hodnotu posledního příkazu?
  - `$ echo $?`

- Jaký význam má první řádka skriptu #!/bin/bash ?
  - Jedná se o shebang a určuje jakým příkazovým procesorem by se měl skript vykonávat
- K čemu slouží příkazy jobs, fg?
  - `$ jobs` - zobrazuje seznam všech aktuálně běžících nebo pozastavených úloh (jobs) v shellu
  - `$ fg` - přepíná úlohu (job) z pozadí nebo pozastaveného stavu do popředí
    - `$ fg [job_ID]`
- Jaký je rozdíl mezi du -h a df -h?
  - `$ du -h`
    - Zobrazuje velikost souborů a adresářů v aktuálním adresáři nebo na specifikovaném místě.
    - -h: Zobrazení velikostí ve snadno čitelném formátu (např. KB, MB, GB).
  - `$ df -h`
    - Zobrazuje informace o dostupném a obsazeném místě na diskových oddílech.
    - -h: Zobrazení velikostí ve snadno čitelném formátu.
- Jak vypsat počet řádků souboru s1.txt?
  - `$ cat s1.txt | wc -l`
  - `$ wc -l s1.txt`
- Jakým příkazem vyhledám řádky obsahující slovo "example" v souboru s1.txt?
  - `$ cat s1.txt | grep example`
  - `$ cat zp.md | nl | grep example` - zobrazí včetně čísla řádku
- Jakým příkazem seřadím řádky souboru s1.txt?
  - `$ cat s1.txt | sort`
  - flag -r pro reverzní řazení
- Jakým příkazem odstraním duplicitní řádky ze souboru s1.txt?
  - `$ cat s1.txt | sort | uniq`
  - uniq předpokládá, že duplicitní řádky jsou pod sebou, takže je nutné nejdříve seřadit
    - flag -d pro zobrazení pouze duplicitních řádků
    - flag -c pro zobrazení počtu duplicitních řádků pro každý řádek
- Jakým příkazem přidám před každý neprázdný řádek souboru s1.txt číslo řádku?
  - `$ cat s1.txt | nl`
- Co dělá příkaz more?
  - Zobrazuje obsah souboru po částech úměrných velikosti terminálu.
- Co dělá příkaz file?
  - Vrací informace o typu souboru.

- PS, TOP, UNAME -A
    - `ps` - informace o procesech
    - `ps aux` - i procesy dalších uživatelů
    - `top` - další program pro zobrazení informací o procesech (více interaktivní než ps)
    - `uname -a` - informace o verzi jádra
- Co dělá příkaz mount?
    - Zobrazuje připojené souborové systémy
    - Dovoluje připojit nové souborové systémy
        - `$ mount -t ext3 /dev/sda4 /mnt/data`
            - (typ fs, co připojujeme, kam)
- /etc/fstab
    - Konfigurační soubor, který obsahuje informace o souborových systémech a jejich připojení při startu systému.
- /etc/mtab
    - Soubor obsahující informace o aktuálně připojených souborových systémech
- Co je sticky bit?
    - Sticky bit je speciální přístupové právo, které může být nastaveno na adresář a ovlivňuje, kdo může mazat soubory v tomto adresáři.
    - Pokud je sticky bit nastaveno na adresář, pouze vlastník souboru nebo superuživatel může mazat soubory v tomto adresáři.
    - Sticky bit je reprezentován jako "t" na konci práv adresáře.
- Linux FHS (Filesystem Hierarchy Standard)
    - / (Root)
        - Kořenový adresář, základ celého souborového systému.
        - Všechny ostatní adresáře a soubory jsou od něj odvozeny.
    - /bin
        - Obsahuje základní binární soubory (programy), které jsou nezbytné pro fungování systému i pro uživatele.
        - Příklady:
            - ls, cp, mv, cat, bash.
    - /sbin
        - Obsahuje systémové binární soubory, které jsou určeny pro správce systému (superuživatele).
        - Příklady:
            - ifconfig, iptables, reboot, mount.

- /etc
  - Konfigurační soubory systému a aplikací.
  - Příklady:
    - /etc/fstab (souborové systémy),
    - /etc/hosts (síťová konfigurace),
    - /etc/passwd (uživatelské účty).
- /usr
  - Obsahuje uživatelské programy, knihovny, dokumentaci a další soubory, které nejsou nezbytné pro spuštění systému.
  - Podadresáře:
    - /usr/bin – programy pro běžné uživatele,
    - /usr/sbin – systémové nástroje,
    - /usr/lib – sdílené knihovny,
    - /usr/share – sdílená data, dokumentace.
- /var
  - Proměnlivé soubory (odtud název "var"), které se mění za běhu systému.
  - Příklady:
    - /var/log – systémové logy,
    - /var/spool – fronty tiskových úloh a e-mailů,
    - /var/www – data webových serverů.
- /home
  - Domovské adresáře uživatelů. Každý uživatel má svůj vlastní adresář, např. /home/uzivatel.
  - Obsahuje uživatelské soubory, konfigurace, dokumenty, stažené soubory apod.
- /root
  - Domovský adresář uživatele root (správce systému).
  - Odlišný od /home, protože je vyhrazený pro superuživatele.
- /boot
  - Obsahuje soubory potřebné pro spuštění systému (bootování).
  - Příklady:
    - Jádro systému (vmlinuz),
    - Inicializační ramdisk (initrd),
    - Konfigurační soubor zavaděče (grub).
- /lib a /lib64
  - Sdílené knihovny a moduly potřebné pro základní funkčnost systému a programů v /bin a /sbin.
  - Příklady:
    - Dynamické knihovny, např. libc.so.
- /opt
  - Volitelné balíčky a aplikace, obvykle třetích stran.
  - Typicky se zde instalují programy, které nejsou spravovány balíčkovacím systémem distribuce.
- /tmp
  - Dočasné soubory vytvářené aplikacemi a systémem.
  - Data zde nejsou trvalá a často jsou při restartu systému mazána.

    - /dev
        - Obsahuje speciální soubory, které reprezentují zařízení (device files).
        - Příklady:
            - /dev/sda – diskové zařízení,
            - /dev/null – speciální soubor pro zahazování dat.
    - /proc
        - Virtuální souborový systém, který poskytuje informace o běžících procesech a stavu jádra.
        - Příklady:
            - /proc/cpuinfo – informace o procesoru,
            - /proc/meminfo – informace o paměti.
    - /sys
        - Další virtuální souborový systém poskytující informace o hardwaru a zařízení připojených k systému.
        - Využívá se pro interakci s jádrem.
    - /media
        - Připojovací body pro vyměnitelné disky, např. USB, CD/DVD.
        - Typicky automaticky spravováno moderními systémy.
    - /mnt
        - Připojovací bod pro dočasně připojené souborové systémy (např. externí disky, síťové disky).
        - Používá se hlavně ručně administrátorem.
    - /srv
        - Data, která jsou poskytována službami (servery), jako jsou webové nebo FTP servery.
        - Příklad: /srv/www může obsahovat data webového serveru.

- Přesměrování vstupu, výstupu a chybového výstupu

    - `$ command > file.txt` - přesměrování výstupu do souboru
    - `$ command >> file.txt` - přidání výstupu na konec souboru
    - `$ command < file.txt` - přesměrování vstupu ze souboru
    - `$ command 2> error.txt` - přesměrování chybového výstupu do souboru
    - `$ command > file.txt 2>&1` - přesměrování výstupu i chybového výstupu do souboru

- umask

    - Maska přístupových práv při vytváření souborů

- Symbolický link a hardlink
  - Symbolický link (symlink):
    - Odkaz na soubor nebo adresář, který může být umístěn v jiném adresáři nebo na jiném oddílu.
    - Symbolický link obsahuje cestu k cílovému souboru nebo adresáři.
    - Při smazání cílového souboru nebo adresáře zůstane symbolický link neplatný.
    - Vytvoření: `$ ln -s /cesta/k/cilovy/soubor /cesta/k/symbolicky/link`
  - Hard link:
    - Odkaz na inode souboru, což znamená, že může existovat více vstupů v adresářové struktuře, které odkazují na stejná data.
    - Hard link nemůže odkazovat na adresáře a musí být ve stejném souborovém systému.
    - Při smazání původního souboru nebo adresáře zůstane hard link stále platný. Při změně dat přes inode se změny přes hardlink projeví.
    - Vytvoření: `$ ln /cesta/k/cilovy/soubor /cesta/k/hard/link`
- Nahrazování slov
  - `$ echo MaLa VELKA Pismena | tr '[A-Z]' '[a-z]'`
  - vypíše:
    - "mala velka pismena"
  - znaky z množiny [A-Z] nahrazuje znaky [a-z]
- Vypsat seřazeně jména aktuálně přihlášených uživatelů
  - `$ who | cut -d' ' -f1 | sort | uniq`
    - `who` - vypíše aktuálně přihlášené uživatele
    - `cut -d' ' -f1` - rozdělí řádky podle mezer a vrátí první sloupec (jméno uživatele)
    - `sort` - seřadí jména
    - `uniq` - odstraní duplicitní jména
- Příkaz tee
  - Příkaz `tee` umožňuje zároveň zapisovat výstup do souboru a zobrazovat ho na standardní výstup.
  - `$ ls | tee soubor.txt`
  - Hodí se na pipeování: `$ ls | tee soubor.txt | grep "soubor"`
- Vnitřní proměnné shellu
  - $0 jméno skriptu
  - $1, $2, ... poziční parametry skriptu,
  - $* seznam parametrů skriptu kromě jména skriptu,
  - $# počet parametrů,
  - $$ identifikační číslo procesu (PID) aktuálního SHELLu,
  - $! PID procesu spuštěného na pozadí,
  - $? návratový kód naposledy prováděného příkazu,
  - $@ seznam parametrů ve tvaru "$1" "$2" "$3" "$4".

- Příkaz `yield`:
  - Slouží k tomu, aby aktuálně běžící proces nebo vlákno dobrovolně předalo svůj časový úsek na procesoru jiným čekajícím procesům nebo vláknům.
- Příkaz `stat` a filesystémy:
  - `stat()`:
    - Vrací informace o konkrétním souboru.
  - `lstat()`:
    - Stejné jako stat(), ale pokud je soubor symbolický odkaz, vrací informace o odkazu samotném, nikoli o cílovém souboru.
  - `fstat()`:
    - Získává informace o otevřeném souboru na základě jeho popisovače (file descriptor).
- Příkaz `lsblk -f` v Linuxu:
  - Slouží k zobrazení informací o blocích zařízení (block devices) v systému, přičemž volba -f přidává informace o souborových systémech a jejich vlastnostech.
- Rozdíl mezi ACL a klasickými UNIX právy:
  - ACL (**Access Control List**) umožňuje definovat specifičtější přístupová práva než klasické UNIX práva.
- Idle thread:
  - Windows: Zajišťuje, že CPU nezůstane nevyužité, pokud nejsou jiné procesy připraveny (**System Idle Process** s nejnižší prioritou - uživatelsky nenastavitelná; jsou vytvořena vlákna pro každé fyzické jádro CPU). Není to proces ve smyslu uživatelských procesů.
  - Linux: Na Linuxu se nejedná o samostatný proces s vlákny, ale o speciální vlákno v jádře, které běží, když není co dělat.
  - Obrazně:

Porovnání	Windows (System Idle Process)	Linux (Idle Task)
Běží v uživatelském prostoru?	Ne, běží v jádře	Ne, běží v jádře
Viditelnost	Viditelný v Správci úloh (PID 0)	Neviditelný v uživatelských nástrojích
Účel	Spravuje nečinnost CPU a snižuje spotřebu energie	Spravuje nečinnost CPU a snižuje spotřebu energie
Název	System Idle Process	Idle Task

- Základní procesy a vlákna ve Windows vs Linuxu:
  - V obou systémech jsou procesy a jejich vlákna identifikovány pomocí PID.
  - Rozdíly:
    - Windows:
      - Každé vlákno má kromě shodného PID procesu, ze kterého pochází, také TID (Thread ID).
      - TID je unikátní identifikátor vlákna v rámci procesu.
      - PID 0 je **System Idle Process** (nejedná se o proces ve smyslu uživatelských procesů, ale o konstrukt jádra pro reprezentaci nečinnosti CPU).
      - PID 4 je **System Process** (reprezentuje jádro systému a jeho vlákna; např. ovladače zařízení, správu paměti, přerušení)
      - PID 1-3 jsou ze začátku vyhrazeny pro pomocné uživatelské procesy při startu OS (např. *smss.exe*, *csrss.exe*, *winlogon.exe*).
      - Nové procesy dostávají PID inkrementálně až do maximální hodnoty. Následně se začíná od prvního volného PID.
    - Linux:
      - Každé vlákno má kromě shodného PID procesu, ze kterého pochází, také LWP (Lightweight Process ID).
      - LWP je unikátní identifikátor vlákna v rámci procesu.
      - PID 0 representuje **Idle Task** (úloha jádra pro reprezentaci nečinnosti CPU).
      - PID 1 je **init/systemd** (první proces spuštěný po startu systému).
      - Další PIDs jsou:
        - **kthreadd** (jádrové vlákno pro správu jiných vláken).
        - **kswapd** (jádrové vlákno pro správu swapování paměti).
        - **ksoftirqd** (jádrové vlákno pro obsluhu softwarových přerušení).
      - Následující PIDs jsou pro uživatelské procesy.
      - Nové procesy dostávají PID na základě volného čísla v tabulce procesů.
    - Obrazně:

Popis	Windows	Linux
První PID	PID 0: System Idle Process	PID 0: Idle Task
Procesy jádra	PID 4: System Process	PID 2: kthreadd (kernel thread daemon)
PID 1	Dynamicky přidělen pro smss.exe nebo jiné	Vždy init (systemd nebo ekvivalent)
Uživatelský prostor	PIDs začínají dynamicky po úlohách jádra	Začíná na PID 1 po úlohách jádra

- Příkaz `killall`:
  - Ukončuje všechny procesy se zadaným názvem.
- Nejhorší priorita pomocí `nice`:
  - +19 (nejnižší priorita na úrovni uživatelského plánování).

- Sirotek vs Zombie:
    - Sirotek (**orphan**):
        - Proces, jehož rodičovský proces skončil dříve než on.
        - Adoptován procesem init (PID 1).
    - Zombie:
        - Proces, který skončil, ale jeho rodičovský proces ještě nezískal jeho návratový kód.
        - Zabírá zdroje, dokud není rodičovský proces informován o jeho ukončení.
- Preemptivní systém - obsahuje přechod **běžící -> připravený**:
    - Ano, pokud je proces přerušen vyšší prioritou.
- `clone()` a implementace vláken v Linuxu:
    - Ano, `clone()` se používá k vytvoření vláken s volitelným sdílením prostředků.