1 Operační systém, proces, vlákno, algoritmy plánování procesů a vláken. Komunikace mezi procesy a vlákny, časově závislé chyby, kritické sekce, synchronizační nástroje, problém uváznutí, možnosti řešení. Správa paměti. Virtuální paměť - stránkování, algoritmy pro náhradu stránek, segmentace. Souborové systémy, organizace dat na vnějších pamětích, principy, řešení, ochrany. (A4B33OSS)

1.1 Operační systém

- zakrývá detaily hardware, aby mohly programy PC snáze ovládat (poskytuje API)
- přiděluje prostředky programům (paměť, procesor...)
- Unixové systémy splňují standard POSIX, který sjednocuje API, systémová volání jsou stejná napříč systémy (open, read, write, fork, mkdir, link...)

1.2 Proces

- je to program označený číslem procesu PID, info o něm je uložené v registru Process Control Block
- tvoří ho obsahy registrů procesoru a uživatelských registrů, otevřené soubory, použitá paměť...
- proces je chápán jako obal vláken, vlákna jsou součástí procesu

- procesy mohou vytvářet nové procesy - potomky - pomocí volání fork/clone. V POSIX jsou všechny procesy potomkem procesu init s PID 0
- proces se ukončí pomocí exit, nebo chybou, nebo na žádost rodiče, nebo při zániku rodiče
- proces se může odpojit od rodiče, pak se mu říká démon
- program je seznam instrukcí

1.3 Vlákno

- existuje v rámci procesu, často má proces jen jedno vlákno
- zdroje jsou sdílené mezi procesem a všemi jeho vlákny
- ukončení procesu zničí všechna jeho vlákna
- správu vláken řeší v POSIX knihovna pthreads

1.4 Algoritmy plánování procesů a vláken

- **Nepreemptivní plánování (N):** Jakmile se procesu přidělí procesor, nelze mu ho vzít. Používá se v uzavřených systémech, kde jsou všechny procesy "slušné" a pustí ostatní včas k procesoru.
- **Preemptivní plánování (P):** Procesu schopnému běhu se procesor odebere a spustí se jiný proces

1.4.1 Různé algoritmy

Plánování se znázorňuje Ganttovým diagramem. Většinou se "doba procesu" musí odhadovat. Je potřeba to řešit chytře, protože časté přepínání je náročné. Když se vybírá proces podle priority, tak se zase může stát, že nějaký proces se vůbec nedostane na řadu (stárnutí).

- First come first serve (N): Procesy se obsluhují v pořadí, v jakém přišly. Nevýhodou je, že všechny krátké procesy budou čekat, než skončí jeden dlouhý.
- Shortest process next (N): Bere se vždy nejkratší proces.
- Shortest remaining time (P): Pokud se objeví proces, který je kratší, než zbývající doba aktuálního procesu, tak se aktuální proces přeruší a spustí se nový proces.
- Round robin (P): Všechny procesy se střídají rovným dílem
- feedback: Procesy, které trvaly nezvykle dlouho, se při příštím běhu dá horší priorita

- RMS: Používá se v systémech, kde se procesy spouští pravidelně s různými periodami. Přednost se dává procesu s kratší periodou

1.5 Kritická sekce

- úsek programu, kde se přistupuju ke sdílenému prostředku
- pokud je v kritické sekci vždy jen jeden proces, vše je ok. Když jich je víc, můžou si navzájem přepisovat sdílený prostředek (třeba proměnnou), protože procesy mohou být v procesoru vyměněny v nevhodnou chvíli. Těsně po tom, co jeden z nich si načte sdílený prostředek k sobě, jsou procesy vyměněny, a ten druhý hned potom změní sdílený prostředek, ale první dál pracuje se starou hodnotou.

1.5.1 Řešení kritické sekce

- Pomocná proměnná: Procesy označují v proměnné, jestli už do kritické sekce vstoupily. Pokud vidí, že v kritické sekci už nějaký proces je, tak čekají (sleep). To ale jen posune kritickou sekci na tuto proměnnou, takže to nefunguje.
- Tři pravidla
  - V kritické sekci smí být v každém okamžiku jen jeden program
  - Výběr procesu k běhu se nesmí nekonečně odkládat
  - Čekání na vstup do kritické sekce musí být omezená
- Petersonovo řešení na aplikační úrovni: Udělají se dvě pomocné proměnné. Proces postupně označí obě, a pak vstoupí do kritické sekce. Pokud je jedna z proměnných aktivovaná, tak čeká a neoznačuje tu druhou.
- HW řešení: Některé procesory mají instrukci TestAndSet, kterou se čte a mění pomocná proměnná. Během této instrukce nejde proces přerušit.
- Semafor: Konstrukce některých programovacích jazyků, která zajišťuje, aby ke zdroji vždy přistoupil jen jeden proces. Kromě semaforu se taky používá Monitor nebo SpinLock.

1.6 Deadlock

- Máme dva procesy, každý z nich si zabere jeden ze dvou zdrojů a pak čekají, než se uvolní ten druhý zdroj, tím se vzájemně zablokují. Jako příklad se dává 5 večeřících filozofů, kteří sedí u kruhového stolu a mají 5 talířů a 5 vidliček, ale k jídlu potřebují dvě vidličky. Pokud se budou všichni filozofové chovat stejně, nikdy se nenají.
- Deadlock se vizualizuje pomocí grafu Resource Allocation Graph (RAG). Procesy a zdroje jsou vrcholy. Pokud proces P čeká na zdroj Z, vede hrana z P do Z. Pokud

je zdroj Z zabrán procesem P, pak vede hrana ze Z do P. Pokud je v grafu cyklus, tak je systém v Deadlocku

- Bankéřův algoritmus: Dovolí spustit jen ty procesy, u kterých ví, že jim za současného stavu dokáže půjčit zdroje. Nevýhodou je, že předem chce vědět seznam zdrojů, které bude proces potřebovat, což je ne vždy znát.
- Deadlock nelze efektivně řešit, lze mu předcházet (Bankéř), nebo ho detekovat a řešit zpětně.

1.7 Správa paměti

- Adresy z Fyzického Adresního Prostoru (FAP) popisují konkrétní místa v paměti (Fyzické adresy, FA)
- Adresy z Logiského Adresního Prostoru (LAP) popisují adresy, se kterými pracují aplikace (Logické adres, LA)
- Adresy z LAP se překládají do FAP
- Adresa, která má délku 32 b popisuje prostor o velikosti 2^32 = cca 4GB

1.7.1 Segmentace

- Paměť se rozdělí na víc segmentů, které mohou mít různou velikost
- Paměť se adresuje jako číslo segmentu a pak pozice v segmentu, to se pak pomocí tabulky ST převede na FA
- Přístup do špatného segmentu vyvolá chybu segmentace (segfault)
- Segmenty lze využít, když je paměť větší než dovoluje adresa popsat (adresa je delší, než povoluje vstup do procesoru), nebo když chceme oddělit paměti procesů od sebe

1.7.2 Virtualizace

- Virtualizace se hodí, když je paměť menší než to, co by šlo popsat adresou. Proto se část paměti uloží jinde

1.7.3 Stránkování

- FAP je rozdělený na rámce stejné velikosti, a těm odpovídají stejně velké stránky v LAP
- Překlad mezi rámci a stránkami je prováděn pomocí page table (PT)

- Adresa je určena číslem stránky p a offsetem ve stránce. Překládá se jen číslo stránky na rámec, offset je stejný.
- Aby se překlad adres zrychlil, ukládají se stránky do cache TLB
- Když se cache naplní, tak je potřeba najít stránku, která se přesune na disk, aby bylo v tabulce místo pro novou
  - First in First Out (FIFO): smažeme nejstarší stránku
  - Last Recently Used (LRU): smažeme stránku, která byla nejdéle nepoužita
- Cache může být víceúrovňová, taky se dá použít hashování

1.8 Souborový systém

- Soubor je pojmenovaná množina dat, která má: jméno, typ, umístění, velikost, ochranu, vlastníka, datum změny/vytvoření/apod
- POSIX má práva read/write/execute a jdou určit pro user/group/other
- Adresář je struktura k hledání pojmenovaných dat
- Ve stromové struktuře jsou soubory listy

1.8.1 FAT (File Allocation Table)

- Z roku 1977, v různých verzích (FAT16, FAT32) ho používaly windows až do XP
- Rozdělí disk na clustery a v tabulce si ukládá, který soubor je ve kterých clusterech
- Pokud soubor nezaplní celý cluster, zůstane místo na disku nevyužitelné
- Max. velikost disku je 2TB
- Stále se používá na flashky

1.8.2 NTFS (New Technology File System)

- Microsoft FS nahrazující FAT v roce 1993
- Podporuje větší disky i soubory než FAT, umí šifrovat a zálohovat data (journaling)

1.8.3 EXT (Extended File System)

- FS pro Unixové systémy, nejnovější je EXT4
- Údaje o souboru jsou uložené v inode: umístění na disku, majitel, práva, čas změny...

- Na disku je uložená tabulka s přehledem inode, z ní se čtou inode a z nich pak umístění souboru
- Změny v souborech jde vrátit zpět (journaling)

1.9 Interpretované jazyky

- Bash, BASIC (starší verze), Perl, Python, Matlab, Java
- Vykonává se přímo kód, nebo jeho předkompilovaná verze (byte code v Javě)
- Je nezávislý na cílovém stroji, ale běží pomaleji

1.10 Kompilované jazyky

- C/C++, kód se kompiluje

1. preprocessing: dají se pryč komentáře, nahradí se makra
2. compiling: C kód se přeloží na instrukce Assembleru
3. assembling: Vznikne object code (*.o) se strojovými instrukcemi
4. linking: do strojových instrukcí se přidají chybějící kusy (knihovny)