1. Definice OS: typy OS; design OS: abstrakce a operace nad nimi – služby, systémová volání. (blok 1) / (pr 01)
   * Základní definice (s vysvětlením).2
     1. Rozšíření stroje (virtualizace) – pohled „shora“
        1. Zjednodušující interface, abstrakce
        2. Příklad: čtení/zápis na disk
     2. Správce prostředků – pohled „zdola“
        1. Procesory, paměti, V/V zařízení
        2. Příklad: tisk na tiskárnu
        3. Multiplexing (sharing) – sdílení prostředků
           1. V čase (CPU)
           2. V prostoru (RAM)
   * Vlastnosti moderního OS.4
     1. Preemptivní (a efektivní) plánování procesů
     2. Izolace procesů
     3. Efektivní správa paměti
        1. Operační paměť – logické adresování, virtualizace
        2. Úložiště – souborové systémy
     4. Víceuživatelský OS, izolace uživatelů
        1. Implementace oprávnění (procesů, souborů)
     5. Podpora IPC – komunikace a synchronizace
   * Typy OS (alespoň šest).3
     1. Mainframe OS – sálové počítače
        1. Obrovská kapacita V/V operací – IBM OS/360, z/OS
     2. Serverové OS
        1. Síťové služby – UNIX, BSD, Linux, Windows Server
     3. Distribuované (rozptýlené, vícepočítačové) OS
        1. Clustery, paralelní počítače
        2. Obvykle modifikace existujících OS (Linux), QNX
     4. Osobní OS – PC: mac OS, Linux, Windows
     5. Real-time OS – důležité je dodržení termínů
        1. QNX, VxWorks, RT-Linux
     6. Vestavěné (embedded) OS
        1. PDA, TV-sety, mikrovlnky, mobily
        2. QNX, VxWorks, PalmOS, iOS, Linux
     7. Smart card OS, SIM card OS
        1. Specializované miniaturní vestavěné systémy
        2. Platební karty – Chip OS (COS, MACOS), MULTOS
   * Definice koncepcí (abstrakcí) a systémového volání.2
     1. Procesy
        1. Programy, které běží v systému (tedy jejich instance)
        2. Proces má alokovaný v paměti svůj adresový prostor (core image)
        3. Pro OS je proces datová struktura obvykle nazývaná PCB (
        4. Nad ní jsou pak prováděny operace spuštění, ukončení procesu, jeho zastavení apod.
        5. Navíc se implementují operace pro komunikaci mezi procesy (IPC – Inter-Process Communication), signály (alarm, V/V operace, …) a další.
        6. Porcesy používají zdroje – systémová zařízení (HW i SW) obecně nazývaná prostředky
        7. Mnohdy nelze prostředky sdílet a vzniká konkurence, procesy se tak mohou dostat do stavu deadlock(„mrtvý bod“)
        8. OS nabízí procesům datové struktury a operace, které jim umožní vyhnout se takovým stavům
     2. Správa paměti
        1. OS i procesy potřebují pro svůj běh paměť – alokují si prostor
        2. Aby proces mohl běžet na libovolných adresách, rozlišujeme adresy fyzické a logické
        3. Umožňuje-li systém zahrnout pro procesy jak RAM, tak i sekundární paměť (disk) pro odkládání procesů nebo jejich částí, hovoříme o virtuální paměti
     3. Vstupy a výstupy
        1. OS má subsystém správy vstupně-výstupních (V/V, anglicky I/O – Input/Output) zařízení
        2. Zajišťuje tak potřebnou abstrakci HW, použitím ovladačů (device drivers)
     4. Správa úložišť
        1. Pro snadnější manipulaci s daty na paměťových médiích vznikl koncept souborů a souborových systémů
        2. Data se ukládají podle sounáležitostí do souborů, ty pak do hierarchické (stromové) struktury adresářů
        3. Soubor je identifikován svou cestou od kořenového adresáře (kořene adresářového stromu)
        4. Rozlišujeme cestu absolutní – od kořene – a relativní – od aktuálního umístění v hierarchii, tj. od pracovního adresáře
        5. V některých os existují speciální soubory, zejména pro práci s HW – blokové (pro práci se zařízeními, jejichž nejmenší jednotka je blok – disky), znakové (pro zařízení schopná adresovat jediný znak či bajt – terminál, zvuková karta), roura (pro komunikaci procesů)
     5. Systémová volání
   * Popište průběh systémového volání.3
     1. Obvykle probíhá přes knihovnu
     2. Uložení parametrů na stack (zásobník, část paměti procesu pro tento účel) (jako u jiného volání funkce (podprogramu))
     3. Program volá systémovou funkci v knihovně
     4. Provede se skok do jádra OS, obvykle pomocí instrukce TRAP
     5. Jádro OS provede vyhodnocení služby (dispatch) a následně volá příslušný ovladač nebo funkci
     6. Po dokončení operace může následovat návrat do knihovny a zpět do procesu. Volání ale může být blokující (proces nemůže pokračovat, dokud nemá výsledky), takže OS mezitím může aktivovat jiný proces
   * Jmenujte příklady systémových volání a co obstarávají (alespoň tři).1
     1. Procesy
        1. Vznik, nahrazení, čekání na ukončení, ukončení
     2. Soubory (V/V)
        1. Otevření, zavření, čtení, zápis, stat (zjištění atributů), ioctl (provedení speciální operace)
     3. Adresáře a souborové systémy
        1. Vytvoření, zrušení, odkazy, připojování FS
     4. Ostatní (práva, signály, …)
        1. Změna práv, signál, zjištění času, …
2. Architektura jádra OS: monolitický systém, vrstvený systém, virtualizace na úrovni jádra OS, mikrojádro. (blok 1) / (pr 01)
   * Monolitické jádro.2
     1. Jádro má všechny potřebné funkce
     2. vše v jednom – vnitřně nečleněné jádro
        1. z hlediska hierarchie volání procedur
        2. každá funkce (procedura) může volat libovolnou jinou
     3. mohou mít i strukturu (pro lepší přehlednost kódu):
        1. hlavní program (obsahuje dispatcher)
        2. obslužné procedury (pro systémová volání, ovladače)
        3. užitkové procedury (pro použití s ostatními procedurami)
     4. procedury mají pevně definované rozhraní
     5. Do této kategorie patří unixové systémy, Linux, Windows.
   * Vrstvené jádro a čím se liší od monolitického s vnitřní strukturou vrstev.2
     1. Jádro OS je rozděleno na oddělené vrstvy
     2. vrstva smí volat jen procedury stejné nebo nejbližší nižší vrstvy – zajištěno HW (úrovně oprávnění)
        1. Na tomto principu napsán systém MULTICS
     3. Dále existuje projekt popsaný E. W. Dijkstrou – THE multiprogramming system (odvozeno z pojmenování univerzity Technische Hogeschool Eindhoven). V návrhu je následující členění jednotlivých vrstev:
        + 5 – operátor – uživatel (Dijkstra poznamenal: „not implemented by us“),
        + 4 – uživatelské programy – kompilace, provádění, tisk,
        + 3 – správa V/V zařízení, buffering,
        + 2 – komunikace mezi procesy a konzolí operátora,
        + 1 – správa paměti, 0 – alokace CPU a multiprogramming – plánování procesů.
   * Mikrojádro: vlastnosti, funkce.2 + 3
     1. Trendem moderních systémů je mít jádro co nejjednodušší
        1. zvyšuje přehlednost kódu, snižuje riziko chyb programátora a v neposlední řadě zvyšuje spolehlivost systému jako celku
     2. pracuje v privilegovaném režimu procesoru (kernel mode)
     3. stará se pouze o plánování procesů (alokace CPU), komunikaci mezi nimi (IPC) a ochranu a virtualizaci paměti.
     4. Klientské procesy pak mají na starost ostatní funkce OS – správa paměti, souborových systémů, ovladače zařízení, …
     5. náročné na implementaci i režii
     6. Mezi zástupce patří
        1. projekt GNU/Hurd (Hird of Unix‑Replacing Daemons, kde Hird znamená Hurd of Interfaces Representing Depth).
        2. QNX [kjuniks] – unixový real‑time OS (vhodný pro vestavěná zařízení i pro distribuované systémy)
        3. MINIX – projekt A. S. Tanenbauma pro výukové účely (Vrije Universiteit, Amsterdam, 1987).
           1. Původně modifikace jádra 4.2BSD nazvaná Mach (vyvíjená na univerzitě Carnegie Mellon) směřuje právě k architektuře mikrojádra.
           2. Ne všechny implementace jsou však plně mikrojádrové.
   * Virtualizace na úrovni jádra.3
     1. srdcem je VM monitor
        1. stará se o alokaci procesoru pro jednotlivé instance systému – multiprogramming
     2. Historie virtualizace sahá až do 70. let 20. století, kdy společnost IBM uvolnila systém VM/370 (Virtual Machine Facility/370)
     3. jediné jádro OS + kontejnery: izolace skupin procesů, uživatelů (vč. správce), sítě i souborových systémů
     4. Virtualizované jádro operačního systému izoluje samostatná prostředí (skupiny procesů) s vlastním administrátorem, přičemž jádro běží pouze jedenkrát
        1. Zástupci jsou např. OS Solaris (od verze 11) s tzv. kontejnery nebo Linux‑VServer
   * Příklady OS z různých kategorií.3
     1. Virtualizované jádro
        1. OS Solaris (od verze 11)
        2. Linux‑VServer
     2. Vícevrstvé OS
        1. MULTICS
     3. Monolitické OS
        1. unixové systémy
        2. Linux
        3. Windows
     4. Mikrojádro
        1. GNU/Hurd
        2. QNX [kjuniks] – unixový real‑time OS
        3. MINIX 3
3. Návrh OS a jeho bezpečnost: důvody náročnosti implementace OS, principy jeho vývoje; zabezpečení systému, uživatelských dat a procesů (vyjma útoků na systém, to je jiný okruh). (blok 13) / (pr 10)
   * Cíle návrhu OS.5
     1. cíle se liší v závislosti na určení OS
        1. vestavěné, serverové, osobní systémy
     2. hlavní společné cíle při navrhování OS
        1. definovat abstrakce
        2. poskytnout základní (primitivní) operace
        3. zajistit izolaci
        4. spravovat hardware
   * Důvody náročnosti.4
     1. rozsáhlost, komplexnost, složitost
        1. Existují i rozsáhlejší a komplexnější systémy než operační systém, např. letadlo. Většina takových systémů lze však rozložit na izolované menší podsystémy mnohem snadněji než OS. Je to dáno zejména tím, že podsystémy se navzájem tolik neovlivňují.
     2. konkurence – uživatelé, procesy, prostředky
        1. Operační systém je víceúlohový, existuje v něm typicky mnoho procesů souběžně. Procesy soupeří o prostředky, zejména o procesor a paměť. Pokud pracují se sdílenými daty, vznikají problémy souběhu (race conditions) a kritické sekce. OS musí nabídnout programátorům prostředky, jak tyto situace řešit.
     3. ochrana – nepřátelské prostředí
        1. Úzce s konkurencí souvisí ochrana. Procesy jsou vlastněny uživateli, kteří si chtějí chránit svá data. Je třeba navrhnout systém tak, aby jim tuto možnost ochrany prostředků dal. Jedná se zejména o implementaci izolace procesů a oprávnění na souborových systémech.
     4. sdílení dat a prostředků
        1. Zdánlivě v kontrastu s předchozím bodem je možnost sdílení dat a prostředků. Nicméně se jedná v podstatě o stejnou věc. Izolaci je třeba navrhnout tak, aby nebyla absolutní, ale aby byla selektivní, tedy aby umožnila sdílení, je-li to vyžadováno. Podobně implementace přístupových práv na souborových systémech by měla umožnit nejen chránit, ale také selektivně sdílet data mezi uživateli.
     5. Životnost
        1. Operační systém „žije“ dlouho. Zatímco aplikační software je poměrně snadné vyměnit a nahradit jiným nebo novějším, operační systém se svým prostředím je dlouhodobějšího charakteru. Uživatelé jen neradi instalují celý systém znovu.
        2. Z tohoto hlediska je velmi důležité, aby operační systém byl navržen tak, aby v budoucnu s příchodem nového hardwaru (zejména nových periferií) nezpůsoboval pokud možno žádné potíže – aby byl připraven. Návrh systému ovladačů by měl být takový, aby pro vývojáře hardwaru bylo snadné začlenit nový ovladač.
     6. obecnost použití
        1. Vývojář operačního systému obvykle neví, co všechno bude chtít uživatel na systému dělat, jaké aplikace programátoři vyvinou a kde všude bude jejich systém nasazen. Pokud má vyhovět co největší množině uživatelů, musí být dobře škálovatelný. Příkladem takového systému je třeba QNX nebo Linux. Oba lze nalézt v celé řadě různých systémů od vestavěných zařízení až po distribuované systémy.
     7. Přenositelnost
        1. S vývojem hardwaru se může stát, že bude existovat (designově, cenově) lepší platforma, na které by mohl navrhovaný operační systém běžet. Pokud je návrh vypracován s ohledem na přenositelnost, může být poměrně snadné systém přenést provozovat i na jiném než původním hardwaru.
     8. zpětná kompatibilita
        1. Většina uživatelů vyžaduje zpětnou kompatibilitu operačního systému s předchozím systémem (nebo jeho předešlou verzí). Mají-li si nainstalovat novější verzi, nebudou spokojeni, pokud jim stávající používané aplikace na novém systému nepoběží.
        2. V designu operačního systému je tedy třeba dbát na to, aby návrh byl dostatečně robustní a aby v budoucnu nezpůsobil komplikace. Jakákoli chyba se pak může vléct s vývojem dalších verzí a může být velmi obtížné se takovéto designové chyby v budoucnu zbavit.
   * Principy vývoje.3
     1. Jednoduchost
        1. méně je více – KISS (Keep It Simple, Stupid)
     2. Úplnost
        1. OS by měl dělat to, co má, ale nic navíc
           1. minimum mechanismů s jasným chováním
           2. Každá část, vlastnost, funkce, systémové volání by měla dělat jednu věc a tu dělat dobře (efektivně) a správně (korektně).
     3. efektivita implementace
        1. pokud by měla být vlastnost (systémové volání) neefektivní, pravděpodobně nemá cenu ji mít
        2. programátor (uživatel OS) by měl také vědět cenu těchto volání
           1. např. posunutí pozice v souboru:
           2. lseek – prosté posunutí pozice (změna proměnné)
           3. read – provedení V/V operace
        3. je-li intuitivní předpoklad ceny volání špatný, budou psát programátoři neefektivní programy
   * Zabezpečení systému (procesů, dat na médiu, přenášených dat).3
     1. domény ochrany
        1. stanovení domén a práv
     2. ACL – Access Control List
        1. stanovení seznamu práv na objekty pro uživatele
     3. capability – C-list
        1. stanovení práv procesům
           1. MAC (Mandatory Access Control)
           2. Linux od r. 1998: SELinux (NSA), AppArmor (SuSE)
     4. firewally a komunikační filtry
     5. Chránit data lze také šifrováním, a to i na úrovni souborových systémů
4. CPU: provádění instrukce, pipeline, atomicita a přerušitelnost procesu a průběhu zpracování instrukce, přerušovací systém, průběh zpracování přerušení, časovač, sdílení času. (blok 2) / (pr 02)
   * Fáze provádění instrukce.4
     1. fetch – načtení instrukce
     2. decode – dekódování (určení činnosti)
     3. execute – provedení
     4. write – zápis výsledku
   * Pipeline, zvyšování výkonu CPU.4
     1. Používají se následující techniky:
        1. **spekulativní provádění instrukcí** – některé instrukce jsou prováděny mimo pořadí (je‑li to pro CPU výhodnější a nenaruší‑li to výpočet),
        2. **použití pipeline** – paralelní zpracování instrukcí na základě rozdělení zpracování instrukce na nezávislé kroky, díky čemuž lze zahájit zpracovávání jedné instrukce ještě před dokončením předchozí
        3. **paralelizace instrukcí** – takové procesory se označují jako superskalární CPU, jsou schopné zpracovávat více instrukcí současně (mají duplicitní součásti),
        4. **hyperthreading** (sdílení subčástí CPU mezi vlákny)
        5. **více jader** – duplicitní CPU v jednom pouzdře, v podstatě se jedná o víceprocesorový systém.
   * Přerušitelnost procesu, průběhu zpracování instrukce.2
     1. k přerušení procesu může dojít pouze na hranicích instrukcí
        1. mezi dokončením jedné instrukce a zahájením další
     2. V/V přerušení – dokončení operace, chybový stav
     3. časovač – důležitý pro preempci – generuje přerušení v daných intervalech
     4. chyby procesu, výjimky – pokus o přístup do zakázané oblasti paměti, neplatná instrukce (privilegovaná v režimu user)
     5. chyby HW – chyba parity paměti, výpadek napájení
   * Přerušovací systém: účel, průběh přerušení.5
     1. umožní efektivní využití CPU při asynchronních přenosech dat (např. DMA)
     2. Průběh zpracování přerušení
        1. CPU provádí proces a HW vygeneruje IRQ (interrupt request = požadavek na přerušení)
           1. CPU: dokončení rozpracované instrukce
           2. CPU: skok na obslužnou rutinu přerušení

před tím se uloží adresa návratu (na systémový stack)

* + - * 1. rutina: uloží se kontext (registry z CPU)

do tabulky procesů (nebo na systémový stack)

* + - * 1. rutina: obsluha přerušení (IRQ časovače: plánovač)
        2. rutina: vrátí se kontext (do CPU), návrat do procesu

plánovač může naplánovat také jiný proces

1. Vstupně-výstupní zařízení: ovladače a techniky programování vstupu a výstupu, DMA. Paměť cache, procesorová cache a střední přístupová doba do paměti. (blok 2) / (pr 02)
   * Metoda 1 komunikace se vstupně-výstupním zařízením.2
     1. přístup pouze pomocí instrukcí CPU
        1. zápis na V/V, kontrola stavu, čekání – neproduktivní
        2. Při přenosu bloku dat je cyklus následující:
           1. Zápis do (čtení z) registru V/V zařízení
           2. kontrola stavu zápisu (čtení) – neproduktivní čekání
           3. pokračování krokem 1
   * Metoda 2 komunikace se vstupně-výstupním zařízením.2
     1. přístup pomocí instrukcí CPU s přerušovacím signálem
        1. zápis na V/V, provádění jiných operací – produktivní
        2. po dokončení: přerušení – pokračování zápisu
        3. Pokud to hardware umožňuje, lze místo čekání na dokončení operace v cyklu využít přerušovacího signálu (IRQ – Interrupt ReQuest)
        4. Cyklus přenosu bloku dat pak vypadá následovně:
           1. Zápis do (čtení z) registru V/V zařízení
           2. provádění jiných operací – produktivní využití CPU,
           3. po dokončení přenosu generuje V/V zařízení přerušovací signál, který přeruší operace v kroku 2, lze tedy pokračovat předáváním dalších dat → krok 1
   * Metoda 3 komunikace se vstupně-výstupním zařízením.3
     1. DMA (Direct Memory Access) a IRQ
        1. zápis adresy a rozsahu dat v RAM do V/V, příkaz
        2. provádění jiných operací – produktivní
        3. po dokončení: přerušení a obsluha
        4. Pří použití přímého přístupu zařízení do operační paměti je postup následující:
           1. Zápis adresy dat, množství dat a příkazu do registru V/V zařízení
           2. provádění jiných operací – produktivní využití CPU
           3. po dokončení přenosu stanoveného množství dat generuje V/V zařízení přerušovací signál, který přeruší operace v kroku 2, čímž je přenos dokončen.
   * Definice cache.2
     1. rychlá mezipaměť (např. mezi RAM a CPU)
   * Důvod existence konceptu paměti cache.1
     1. výroba velmi rychlé paměti je drahá, pro operační paměť se nepoužívá ta nejlepší technologie, ale jiná, přijatelně rychlá a cenově dostupnější. Díky tomu je však procesor, který pracuje několikrát rychleji než paměť, zdržován při každém přístupu do operační paměti
     2. cache – je rychlostí stejná jako procesor (nebo je mu rychlostí mnohem bližší)
   * Důvod efektivity používání cache.2
     1. proces má tendenci přistupovat ke stejné části paměti jako v bezprostředně předcházejícím čase (např. při vykonávání cyklu), což se označuje jako **princip lokality odkazů**.
     2. Díky tomuto principu může paměť cache fungovat efektivně i při malých velikostech a činitel úspěšnosti (Hit Ratio), kterým se určuje efektivita paměti cache, se pak bude blížit jedné i při malé kapacitě paměti.
   * Výpočet střední přístupové doby do paměti.3
     1. TS = TC + (1 − HR) ⋅TOP
     2. kde TC je přístupová doba do paměti cache a TOP je přístupová doba do operační paměti, přičemž TC << TOP. Bude‑li HR blízko jedné, střední přístupová doba se bude blížit přístupové době do cache.
2. Požadavky OS na HW nutný pro jeho implementaci: zejména na procesor, správu a adresování paměti. Požadavky na moderní OS. (pr 02)
   * Nutné vlastnosti komponent – CPU, paměti, přítomnost subsystémů, jejich účel.4× 2
     1. **CPU – Central Processing Unit**
        1. ALU – aritmeticko-logické jednotky
        2. FPU – numerický koprocesor (Floating Point Unit)
        3. registry:
           1. program counter, instruction register, stack pointer
           2. PSW (program status word) – příznaky C, N, Z, V

C = carry, N = negative, Z = zero, V = overflow

* + - * 1. ostatní registry (obecné, datové, adresní, privátní, …)
    1. **Paměť**
       1. registry procesoru, cache, operační paměť
       2. ochrana operační paměti
          1. paměti procesů navzájem
          2. jádro × procesy
       3. relokace
          1. zavedení procesu na libovolnou adresu
          2. virtuální adresa × fyzická adresa

MMU (Memory Management Unit)

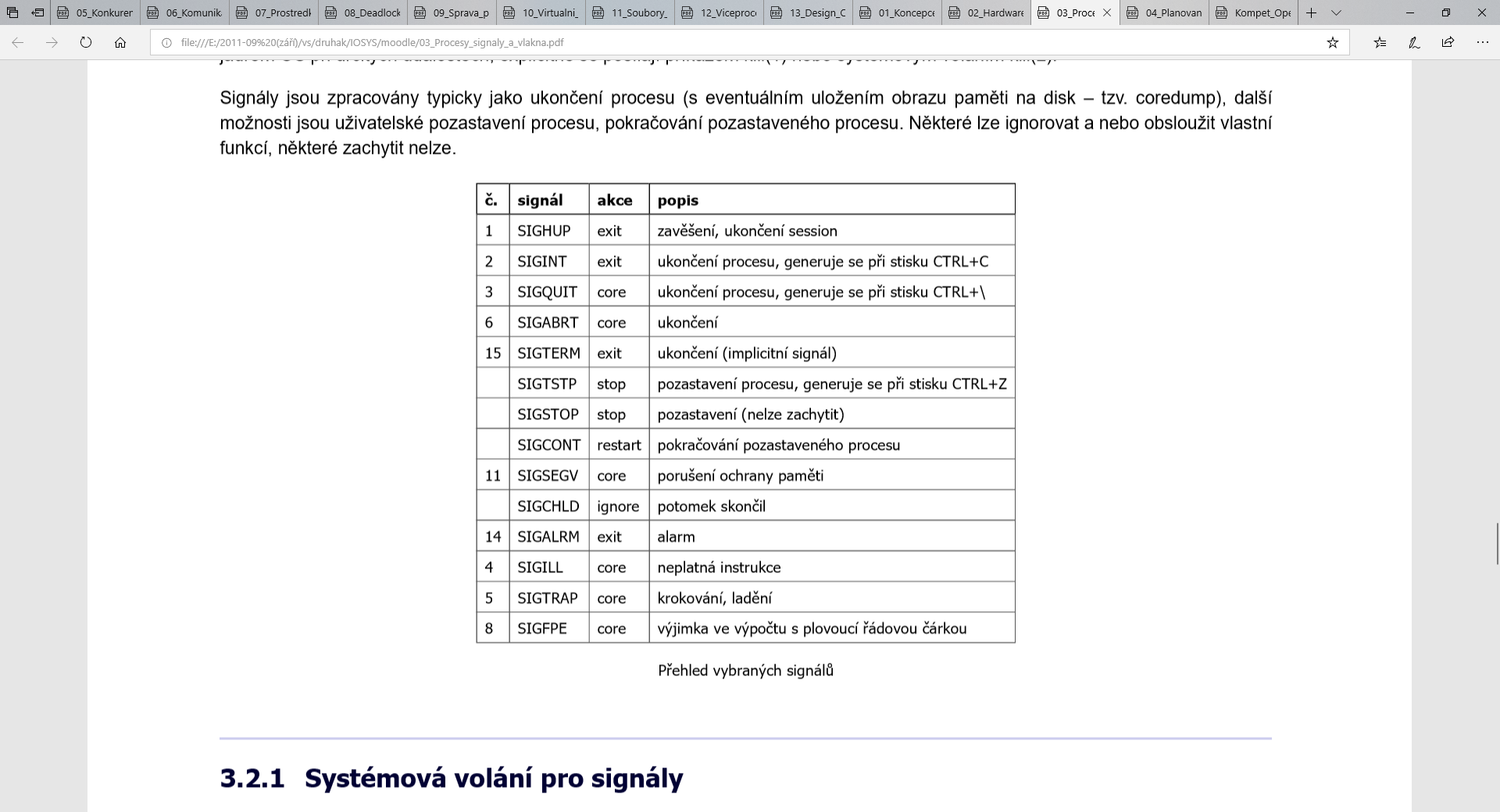
* + 1. **Vstupně-výstupní zařízení**
       1. zařízení, řadič (řídicí jednotka, controller)
       2. OS zjednodušuje práci s V/V zařízeními
          1. ovladače zařízení pro OS

zařazení přímo do jádra OS

načtení ovladačů při spuštění systému

načtení ovladačů za běhu systému – USB, IEEE 1394

* + - 1. registry na zařízení, V/V porty, přerušení
      2. DMA (Direct Memory Access)
    1. **Přerušení (interrupt, IRQ)**
       1. V/V přerušení
          1. dokončení operace, chybový stav
       2. časovač – důležitý pro preempci
          1. generuje přerušení v daných intervalech
       3. chyby procesu, výjimky
          1. pokus o přístup do zakázané oblasti paměti, neplatná instrukce (privilegovaná v režimu user)
       4. chyby HW
          1. chyba parity paměti, výpadek napájení
    2. **Sběrnice**
       1. komunikace částí počítače mezi sebou
          1. interní, lokální: CPU, FSB – paměť
          2. interní: ISA, PCI, AGP, PCI-E, SCSI, IDE, EIDE, ATAPI, ATA, Ultra ATA, SATA
          3. externí: PCMCIA, PC Card, ExpressCard, SCSI, eSATA, IEEE 1394, USB, …
       2. OS spravuje zařízení připojená na sběrnice
          1. plug-and-play
          2. BIOS (Basic Input Output System)
  + Navazující vlastnosti moderních OS: procesy, paměť, další vlastnosti.2 + 2 + 3
    1. přerušovací systém
       1. umožní efektivní využití CPU, nutné pro DMA
    2. časovač – pravidelně generované přerušení
       1. umožní preempci
    3. CPU s podporou režimů kernel a user
       1. ochrana paměti – měnit přístup jen v režimu kernel
       2. neprivilegovaná operace v režimu user – TRAP
    4. CPU s podporou virtualizace paměti
       1. logické adresování – umožnění relokace

1. Implementace procesu v OS: proces a program, tabulka procesů, přepínání kontextu, stav procesu, třístavový model, příčiny změn stavů. (blok 3) / (pr 03)
   * Proces a program: definice.2
     1. Proces je instancí programu v operační paměti systému. Lze použít přirovnání z oblasti gastronomie: program je recept, proces je příprava pochutiny dle receptu.
   * Metadata procesu (alespoň šest).3
     1. záznam o jeho identifikaci
     2. atributech
     3. stavu
     4. alokovaných zdrojích jako paměťový adresní prostor
     5. otevřené soubory
     6. sokety
   * Průběh přepnutí kontextu.3
     1. Procesu, kterému se odebírá CPU, je třeba uložit rozpracovaný stav výpočtu (kontext), aby později mohl korektně pokračovat v běhu
     2. Tento kontext (stav) se ukládá obvykle do PCB
     3. Proces (kód) je na procesoru vykonáván, dokud proces nezavolá službu systému nebo nedojde k (hardwarovému) přerušovacímu signálu, zde časovače
     4. Signál přerušení je zpracován tak, že se skočí na obslužnou rutinu (do jádra OS), což je pro přerušení časovače plánovač
     5. Plánovač uloží kontext procesu (stav registrů) do PCB, následně vybere dle plánovacího algoritmu další připravený proces, který má běžet
     6. Stav tohoto procesu nahraje do procesoru a přenechá mu procesor
     7. Ošetřuje-li jádro OS požadavek na přerušení, je třeba také uložit stav prováděného procesu.
     8. Následně dojde k obsluze (typicky ovladačem v jádře OS) a následně se k činnosti dostane opět plánovač.
     9. Ten rozhodne, který proces poběží dále, tj. kterému se přidělí CPU.
     10. Může to být předchozí proces (ten přerušený), ale může to být právě proces, který čekal na událost, která nastala obsloužením tohoto přerušení.
     11. Opětovně spouštěnému procesu je třeba do procesoru nahrát jeho předchozí uložený stav.
   * Třístavový model: popis stavů.3
     1. běžící (running) – používá CPU
     2. připravený (ready) – pozastaven jádrem OS – může běžet, až mu systém přidělí CPU
     3. blokovaný (blocked) – čekající na vnější událost – nemůže běžet, dokud událost nenastala; např. čeká na uživatelský vstup, dokončení čtení z disku apod.
   * Třístavový model: příčiny přechodů mezi stavy.4
     1. Běžící -> blokovaný – blokující systémové volání
     2. Běžící -> připravený – preempce
     3. Připravený -> běžící – aktivace
     4. Blokovaný -> připravený – nastala událost
2. Procesy v OS UNIX/Linux: vznik a zánik procesu, systémová volání fork(2), exec(3), exit(3), wait(2), kill(2); hierarchie procesů, stavy procesů v Linuxu (podle příkazu ps); posixové signály a jejich zpracování. (blok 3) / (pr 03)
   * Příčiny vzniku a zániku procesu, související systémová volání, hierarchie.7
     1. Vznik fork(2), :
        1. vytvoření jádrem (typicky při startu OS)
        2. pomocí systémového volání
        3. požadavek uživatele (např. pomocí tlačítka u vestavěných systémů nebo u dávkových systémů).
     2. Zánik procesu \_exit(2), exit(3), kill(2):
        1. normální – systémovým voláním (dokončení programu)
        2. při chybě, dobrovolné (detekce chyby, ukončení)
        3. fatální chyba, nedobrovolné – jádrem OS (programem neošetřená chyba jako provádění neplatné operace, přístup do neplatné adresy pro proces)
        4. zabití jiným procesem nebo jádrem – signálem (uživatelský požadavek nebo potřeba systému např. při vyčerpání zdrojů).
   * Stavy procesů v Linuxu.4
     1. R – připravený nebo běžící (runnable/running)
     2. S – blokovaný (sleep)
     3. Z – ukončený (defunct, zombie)
     4. T – pozastavený nebo krokovaný (stopped, traced)
     5. D – nepřerušitelný spánek (uninterruptible sleep); v jádře
     6. X – mrtvý (dead) – nemělo by se nikdy vyskytovat
     7. W – paging – jen před jádrem řady 2.6, nemělo by se již vyskytovat.
   * Posixové signály, možnosti jejich zpracování a příklady.4
     1. jednoduché zprávy procesům, které jsou zpracovány buď jádrem nebo navázanou registrovanou funkcí
     2. Popis lze najít v manuálových stránkách – signal(7)
     3. Implicitně jsou posílány jádrem OS při určitých událostech, explicitně se posílají příkazem kill(1) nebo systémovým voláním kill(2).
     4. Jsou zpracovány typicky jako ukončení procesu (s eventuálním uložením obrazu paměti na disk – tzv. coredump), další možnosti jsou uživatelské pozastavení procesu, pokračování pozastaveného procesu. Některé lze ignorovat a nebo obsloužit vlastní funkcí, některé zachytit nelze.
     5. Přehled systémových volání pro signály:
        1. poslání signálu procesu – kill(2)
        2. nastavení obsluhy signálu (zastaralé, zavrženo) – signal(2)
        3. změna obsluhy signálu – sigaction(2)
        4. blokování signálů, nastavení množin signálů – sigemptyset(3), sigfillset(3), sigaddset(3), sigdelset(3), sigismember(3)
        5. nastavení poslání upozornění – alarm(2)
        6. čekání na konkrétní signály – sigsuspend(2)
        7. nastavení upozornění po čase – setitimer(2), getitimer(2)
     6. 
3. Vlákna: motivace zavedení vláken, proces × vlákno, možné implementace vláken, obecné problémy při implementaci a používání vláken; knihovna posixových vláken: mutex, bariéra, podmínková proměnná a jejich použití. (blok 3) / (pr 03)
   * Motivace zavedení vláken.1
     1. stejný důvod jako pro zavedení multitaskingu procesů, tedy zvýšení výkonu a zjednodušení programování
   * Společné a samostatné položky metadat procesů/vláken, důvody.3 + 2
     1. Vlákna mají samostatně evidované následující položky (v tabulce vláken):
        1. program counter a registry (uložený kontext procesoru) – nutné, aby mohlo každé vlákno vykonávat jinou funkci
        2. stav vlákna – každé vlákno může být v jiném stavu (blokované, připravené, běžící); souvisí úzce s uloženým kontextem (s registry),
        3. stack (zásobník) – nutné kvůli nezasahování si vláken do svých výpočtů.
     2. Ostatní položky jsou sdílené v tabulce procesů (PCB)
        1. adresní prostor
        2. prostředky (jako otevřené soubory)
        3. environment
        4. pracovní adresář
        5. obsluha signálů
        6. nástroje IPC (semafory, sockety)
        7. účtování (accounting)
   * Implementace s podporou OS a bez ní, výhody a nevýhody.4
     1. implementace bez podpory jádra OS
        1. + lepší režie – rychlejší vznik, přepnutí kontextu
        2. + nevyžaduje se přechod do režimu jádra
        3. + strategie plánovače se dá přizpůsobit aplikaci
        4. − složitá implementace (neblokující volání, plánovač)
        5. − page-fault zastaví všechna vlákna
     2. implementace v jádře OS
        1. + lze provádět i vlákno procesu, jehož jiné vlákno způsobilo page-fault
        2. + není třeba neblokujících volání
        3. − horší režie – přechod do režimu jádra
        4. − pevná strategie plánovače vláken
   * Komplikace při používání vláken.2
     1. globální proměnné (např. errno)
        1. potřeba samostatných alokací pro každé vlákno
     2. nereentrantní volání některých knih. funkcí
        1. např. alokace paměti (malloc)
     3. přístup ke sdíleným proměnným (kritické sekce)
     4. znalost implementace signálů a jejich obsluhy
        1. které vlákno má dostat signál, které se má přerušit
     5. stack (automatické zvětšení při přetečení?)
   * Posixová knihovna vláken: řešení problémů souběhu.3
     1. posixová knihovna pthread.h(7)
        1. #include <pthread.h>
     2. POSIX.1c – redefinice globální proměnné errno
        1. #include <errno.h>
     3. kompilace se symbolem \_REENTRANT
        1. gcc -D\_REENTRANT prog.c -lpthread -o prog
        2. zajistí reentrantnost funkcí
        3. zajistí přenositelnost
4. Plánovač. Cíle plánování, režimy plánování, plánovací kriteria (cíle) pro plánovací algoritmy, plánovací algoritmy. (blok 4) / (pr 04)
   * Úloha plánovače, režimy plánování procesů.3
     1. rozhoduje, který proces (vlákno) má CPU
        1. vybírá připravený proces pro běh
        2. přiděluje CPU
        3. odebírá CPU běžícím procesům
        4. preempce
        5. může se aktivovat také při systémových voláních
     2. řídí se plánovacím algoritmem
     3. Režimy plánování
        1. Nepreemptivní
           1. proces se musí sám vzdát CPU (nebo blokovat)
        2. Preemptivní
           1. plánovač rozhoduje, kdy který proces má CPU
           2. (efektivně) plánovat lze pouze v případě, že je k dispozici přerušovací systém a časovač
           3. časovač „tiká“ typicky na frekvenci 100 Hz

přerušení nastává tedy každých 10 ms

plánování spotřebovává také čas CPU – režie

* + Cíle plánování (obecně a dle určení OS).3
    1. rovnováha zatížení systému – snaha využívat všechny části systému zatížené a tím optimalizovat jeho výkon
    2. spravedlnost – každý proces by měl stejně času
    3. dodržování strategie, která může být na různých systémech jiná:
       1. Interaktivní systémy
          1. **minimalizovat odezvu** (response time) – čas mezi zadáním příkazu a jeho provedením
          2. **proporcionalita** – vyhovění očekáváním uživatelů, kteří mají nějakou představu o tom, jak dlouho by co mělo trvat
       2. Dávkové systémy
          1. **maximalizovat propustnost** (throughput) – počet vykonaných úloh za jednotku času (typicky h)
          2. **minimalizovat obrat** (turnaround time) – průměrný čas na vykonání úlohy
          3. **využití CPU** – využívat maximálně procesor, aby nevznikaly prostoje, kdy je procesor nevyužit.
       3. Systémy real-time
          1. **respektovat lhůty** – zabránění ztráty dat,
          2. **předvídatelnost** – zabránění degradace kvality; je třeba dopředu počítat s možným problémem, např. multimediální systémy načítají dopředu audiovizuální data z média.
  + Popište plánovací algoritmy: historické.3
    1. fronta jednotlivých úloh (FIFO), víceúlohová FIFO
       1. Nepreemptivní algoritmus.
       2. Nové úlohy se zařadí do fronty a po ukončení aktuálního procesu se přidělí CPU procesu, který čekal ve frontě nejdéle.
       3. Krátké procesy musejí zbytečně dlouho čekat.
       4. Zvýhodněny jsou výpočtově orientované procesy bez V/V operací, které čekají pouze jednou.
       5. Procesy s mnoha V/V operacemi čekají při každém dokončení operace, než se jiný proces vzdá CPU
    2. podle odhadu doby běhu úlohy
       1. Shortest Job First (nejkratší úloha první)
       2. Shortest Remaining Time Next (nejkratší zbývající následuje)
  + Popište plánovací algoritmy: moderní a specifické.6
    1. round-robin (cyklická obsluha) – spravedlivé střídání úloh
       1. Typické plánování moderních OS je založené na cyklické obsluze připravených procesů. Jedná se o preemptivní algoritmus, kdy plánovač je periodicky aktivován při obsluze přerušení časovače.
       2. Každý proces dostane přiděleno časové kvantum pro využívání CPU. Přepnutí na jiný proces (context switch) je prováděno při vypršení tohoto kvanta nebo při volání blokujícího systémového volání procesem, který má právě předělen procesor. Délku kvanta je třeba optimalizovat, aby byl čas procesoru efektivně využíván.
    2. Priority Based Scheduling (prioritní plánování) – dle důležitosti úlohy
       1. Každý proces má přidělenou prioritu, což je obvykle celé číslo. Při výběru, který proces dostane CPU, se dává přednost procesu s vyšší prioritou. Obvykle algoritmus pracuje s více frontami pro připravené procesy – pro každou prioritu samostatná fronta.
       2. Nízká priorita může mít za následek „vyhladovění“ procesu (starvation) – proces s nízkou prioritou se nedostane k CPU. Tento případ lze řešit např. dynamickou složkou priority, která se mění v závislosti na době čekání nebo historii běhu procesu.
    3. uživatelsky férové – spravedlivé mezi uživateli
    4. termínové – dodržení lhůt na systémech real-time

1. Požadavky na plánování v systémech reálného času. Možnosti plánování vláken na víceprocesorových systémech (SMP). Časové a periodické plánování úloh uživatelem – příkazy at a crontab. (pr 04)
   * Kritéria pro plánování v systémech reálného času, plánovatelnost.4
     1. pro úlohy reálného času je důležité **dokončení ve stanoveném termínu** (nikoliv rychlost)
     2. systém je plánovatelný, je li ‑ **suma časů potřebných na obsloužení událostí dělená jejich periodami menší nebo rovna jedné**
     3. lze plánovat staticky (table-driven)
        1. tabulky stanoví, kdy která úloha má být spuštěna
     4. **aperiodické události: dynamické** plánování
   * Možné optimalizace plánování vláken na systémech SMP.4× 2
     1. Používají se různá (heuristická) pravidla (i při globální frontě):
        1. Afinita vlákna k CPU–použij procesor, kde vlákno již běželo (možná, že v cache CPU budou ještě údaje z minulého běhu)
        2. Afinita vlákna k CPU při globální frontě–neber první proces z fronty, ale prozkoumej více procesů na začátku fronty a hledej proces, který běžel na daném procesoru
        3. Použij nejméně využívaný procesor
   * Příkazy pro nastavení spouštění úloh v daném čase a démony, které to obstarávají. Orientačně možnosti konfigurace.3
     1. **at**
        1. naplánované (odložené) spuštění programů (skriptů)
     2. **crontab**
        1. soubor obsahující rozvrh cron vstupů, které se mají vykonat ve specifikovaných časech
     3. **cron**
        1. daemon, který běží při startu systému z /etc/init.d
2. Požadavky na paměť, alokace, adresování, pevné (statické) a proměnné (dynamické) dělení paměti (fixed partitioning, variable partitioning), fragmentace paměti, typy fragmentace, umisťovací algoritmy. (pr 07)
   * Požadavky na paměť (na její vlastnosti), předpoklady.3
     1. relokace paměti procesu
     2. ochrana paměti
     3. možnost sdílení paměti mezi procesy
     4. logická a fyzická organizace
        1. logické členění paměti (procesu)
        2. adresace (logické adresy → fyzické adresy)
   * Pevné dělení paměti, nevýhody, umisťování procesů.4
     1. paměť je rozdělena na pevně definované oblasti
     2. stejná velikost všech oblastí
        1. jakýkoli malý proces zabere celou oblast
           1. neefektivní využití paměti
           2. vnitřní fragmentace (internal fragmentation) – proces nevyužije veškerou přidělenou paměť
     3. nestejná velikost oblastí
        1. redukuje se vnitřní fragmentace
        2. ale neodstraňuje se úplně
     4. Umisťovací algoritmus
     5. stejná velikost oblastí – libovolná volná oblast
     6. nestejná velikost oblastí
        1. procesy čekající na přidělení paměti tvoří frontu
           1. fronta samostatná pro každou velikost oblasti

fronta se pro proces vybírá tak, aby se minimalizovala vnitřní fragmentace – nejmenší oblast, kam se vejde

blokuje procesy, které by mohly být ve větší volné oblasti

* + - * 1. fronta jediná

přidělí se nejmenší volná oblast, do které se proces vejde

rychlejší, ale za cenu větší vnitřní fragmentace

* + Dynamické dělení paměti, nevýhody.4
    1. proměnný počet oblastí i jejich velikost
    2. proces dostane tolik paměti, kolik potřebuje
       1. odstraněna vnitřní fragmentace
    3. po ukončení procesu vznikají v paměti mezery
       1. sem lze umístit jen proces, který se tam ještě vejde
       2. při umístění procesu do mezery obvykle zůstane mnohem menší mezera – ještě obtížněji využitelná
       3. vnější fragmentace (external fragmentation)
          1. odstranit lze defragmentací – relokací paměti procesů tak, aby vznikla souvislá volná oblast – setřesení
  + Umisťovací algoritmy.4
    1. best-fit
       1. vybere stejnou nebo nejmenší volnou oblast, do které se proces vejde
       2. nejméně výkonná metoda
       3. nejmenší možná fragmentace
          1. vždy je použita nejmenší vyhovující oblast
       4. fragmenty jsou malé, ale rychle přibývají
          1. nutnost častého provádění setřesení (compaction) obsazené paměti
    2. first-fit
       1. paměť se prohledává vždy od začátku
       2. vybere první volnou oblast, do které se proces vejde
       3. rychlejší než best-fit
       4. prohledávání zpomaluje výskyt velkého počtu obsazených oblastí na začátku paměti
          1. tato část paměti se vždy zbytečně prohledává
    3. next-fit
       1. paměť se prohledává vždy od oblasti, do které se naposledy umisťovalo
       2. vybere první volnou oblast, do které se proces vejde
          1. je-li po umístění procesu do volné oblasti zbytek oblasti ještě dostatečně velký, umístí se další proces sem
       3. nejčastěji se umisťuje na konci paměti
          1. obvykle tam je nejvíce volného místa – tendence dělit velké oblasti paměti na menší
          2. nejrychlejší metoda
    4. worst-fit
       1. vybere stejně velkou volnou oblast jako proces, pokud existuje, jinak největší volnou oblast
          1. paměť se prohledává do nalezení stejně velké oblasti
          2. při nenalezení stejně velké oblasti prohledáváme celou paměť
       2. tendence dělit velké oblasti paměti na menší
          1. může mít za následek nemožnost přidělení paměti velkému procesu
       3. nejhorší využití paměti – lowest memory utilization

1. Problém nedostatku operační paměti, odkládání obsahu paměti na disk, virtuální paměť, motivace k zavedení virtuální paměti, výhody virtualizace paměti, princip lokality odkazů, thrashing. (pr 07)
   * Možná řešení nedostatku RAM (s vysvětlením principu).2 + 2
     1. **překrývání** (overlaying)
        1. různé moduly programu nejsou vyžadovány současně
        2. moduly používají stejnou fyzickou oblast paměti
     2. **swapping**
        1. odkládání procesu z operační paměti na sekundární
        2. využití levné sekundární paměti (disku)
   * Virtualizace paměti, motivace, důsledky, pojmy (RS, S), princip fungování a HW podpora.6
     1. Relokace paměti procesu
        1. přemístění paměti procesu na jiné adresy
        2. program nesmí pracovat s absolutními adresami fyzické operační paměti
           1. programátor nemůže vědět, na které adresy je program do paměti zaveden
           2. proces může být pozastaven a jeho paměť odložena na disk (swap)
        3. proces musí pracovat s logickými adresami
           1. za běhu se převedou na skutečné fyzické adresy
     2. procesor musí umět pracovat s logickou (virtuální) adresou
        1. CPU předává virtuální adresu paměťové jednotce
     3. MMU (Memory Management Unit) převádí virtuální adresu na adresu skutečnou (v RAM)
     4. nenachází-li se adresovaná část v RAM, je třeba generovat přerušení a předat řízení OS
        1. OS vydá příkaz pro nahrání příslušné části paměti z disku do RAM
     5. resident set (RS)
        1. část adresového prostoru procesu, která je v RAM
     6. sekundární paměť (disk), kam lze odložit (dočasně) nepožívané části adresového prostoru procesu – přistupuje-li proces k místu (data, kód), které není v RAM, generuje se přerušení (a skok do OS)
        1. OS pak proces blokuje, dokud nenahraje příslušnou část jeho adresového prostoru do RAM
        2. po dokončení čtení je generováno přerušení a proces je převeden do stavu připravený
   * Princip lokality odkazů.2
     1. proces má tendenci přistupovat k okolí svého adresového prostoru, kam přistupoval nedávno
        1. týká se dat (proměnné ve stejné datové části)
        2. i kódu (zejména při provádění cyklů)
     2. lze obvykle odvodit, kterou část paměti bude proces v nejbližší budoucnosti používat
        1. virtuální paměť může tedy fungovat efektivně
           1. z disku se může dopředu načíst do RAM více částí paměti procesu z okolí právě vyžadované oblasti
   * Thrashing, příčiny.3
     1. vzniká při špatné organizaci odkládání částí paměti procesu na disk
        1. část paměti procesu je odložena na disk těsně před tím, než ji proces potřebuje
           1. tuto část paměti je pak třeba následně nahrát zpět do RAM
           2. proces pak zbytečně čeká v blokovaném stavu
        2. může být následkem nedostatečného dodržení principu lokality odkazů
           1. neoptimalizovaného překladu (kompilátorem)
           2. neefektivního programování (programátorem)
2. Stránkování paměti, převod adresy, vlastnosti stránkování, sdílení stránek, volba velikosti stránky, řešení problému rozsáhlých stránkových tabulek, TLB. (pr 07)
   * Princip, vlastnosti (spojitost, fragmentace, sdílení).5
     1. fyzická operační paměť RAM je rozdělena na oblasti malé velikosti – rámce (frames)
     2. logický adresový prostor procesu se rozdělí na stejně velké části jako RAM – stránky (pages)
     3. OS udržuje pro každý proces tabulku přiřazení stránek k rámcům
     4. logická adresa se skládá z čísla stránky a offsetu
        1. offset je relativní adresa vzhledem k začátku stránky
        2. číslo stránky = logická adresa / velikost stránky
   * Stránkové tabulky, převod adresy.4
     1. stránková tabulka obsahuje příznaky (řídicí bity)
        1. přítomnost stránky v RAM (present)
        2. stránka je používaná (referenced, accessed)
        3. modifikovaný obsah (modified, dirty)
           1. nebyla-li stránka v RAM modifikovaná a je již na disku, není třeba ji znovu zapisovat při uvolňování rámce
   * Volba velikosti stránky a důsledky.3
     1. Menší
        1. paměť se alokuje po stránkách pevné velikosti
           1. vzniká vnitřní fragmentace
        2. menší velikost stránky
           1. menší vnitřní fragmentace
           2. více stránek na proces

větší stránková tabulka – zabírá další paměť

* + - 1. více rámců v RAM = lepší hospodaření s RAM
         1. v RAM budou stránky z okolí posledních odkazů na paměť a počet neúspěšných přístupů se bude snižovat
    1. větší
       1. větší velikost stránky
          1. větší vnitřní fragmentace
          2. menší stránková tabulka
          3. méně rámců v RAM = horší hospodaření s RAM

v RAM budou díky velikosti stránek i bloky paměti, které nejsou aktuálně využívány – princip lokality

při nedostatku RAM se bude počet neúspěšných přístupů zvyšovat – hrozí thrashing

* + - * 1. sekundární paměť je efektivnější při přenosu dat po větších blocích
  + Řešení rozsáhlosti stránkových tabulek, TLB.3
    1. velikost tabulky může být velká, proto se tabulka umisťuje také do virtuální paměti
       1. v RAM nemusí být tabulka celá
       2. část může být na disku (swap)
    2. Translation Lookaside Buffer (TLB)
       1. speciální cache pro položky stránkové tabulky
       2. obsahuje několik posledních použitých položek stránkové (příp. invertované) tabulky
       3. při hledání položky stránkové tabulky se prohledá nejprve TLB
          1. kontrola přítomnosti v TLB může být paralelizována
          2. nalezení (hit) – převod virtuální adresy na fyzickou
          3. nenalezení (miss) – doplnění položky do TLB

doplnění může být z části tabulky v RAM nebo z disku

1. Řídicí bity ve stránkových tabulkách, strategie zavádění, umisťování, nahrazování a uklízení (čištění) stránek paměti, vliv velikosti resident-set na běh procesů. (blok 10) / (pr 07)
   * Důležité řídicí bity a jejich význam.3
     1. 32bitový režim – dolních 12 bitů je řídicích
        1. stránková tabulka (PT), stránkový adresář (PD)
           1. 0 – P – present – použitá položka
           2. 1 – R/W – zápis povolen
           3. 2 – U/S – user/supervisor – přístup v uživatelském režimu
           4. 3 – PWT – page-level write-through
           5. 4 – PCD – page-level cache disable
           6. 5 – A – accessed (referenced) – stránka byla použita
           7. 6 – PT: D – dirty (modified) – změněna, PD: ignored
           8. 7 – PT: PAT/reserved, PD: PS – page size (4KiB/4MiB)
           9. 8 – PT: G/ignored – global (podle CR4.PGE), PD: ignored
           10. 9–11 – ignored
   * Strategie zavádění, účel, algoritmy.3
     1. fetch policy
        1. určuje, která stránka se má zavést do RAM
     2. demand paging – zavádění stránek jen tehdy, jsou-li potřeba
        1. vyskytne-li se odkaz na paměťové místo v dané stránce
        2. způsobuje neúspěšné přístupy při zavádění procesu
     3. lookahead paging – zavádění stránek předem
        1. dle principu lokality odkazů se zavedou stránky z okolí té právě vyžadované (využívá se sekvenční čtení z disku)
   * Strategie umisťování, účel, algoritmy.1
     1. placement policy
     2. určuje, do které části RAM se stránka zavede
     3. typicky se tím OS nezabývá
        1. vybírá obvykle libovolný volný rámec
        2. přístupová doba (access time) je stejná pro všechny adresy v RAM
        3. skutečnou adresu používá pouze HW
   * Strategie nahrazování, účel, algoritmy.4
     1. replacement policy (RP)
        1. určuje, které stránky se z RAM odloží na disk, je li potřeba načíst jiné stránky
     2. ideální strategie – odstranění stránky, která bude nejdelší dobu nevyužitá
        1. nelze předem přesně určit – pouze se odhaduje
     3. pro stránky, které nesmí být odstraněny z RAM, se používá zamykání rámců (frame locking)
        1. jádro OS, V/V buffery, klíčové řídicí struktury OS
   * Strategie uklízení (čištění), účel, algoritmy.3
     1. cleaning policy
        1. určuje, kdy se (modifikované) stránky uloží na disk
     2. demand cleaning
        1. ukládání stránky z rámce vybraného pro nahrazení
           1. proces čekající po page fault čeká na přenos 2 stránek
     3. precleaning
        1. periodické ukládání stránek v dávce
           1. může vést ke zbytečným zápisům
           2. obsah stránky se po uložení na disk může opět změnit
   * Volba velikosti resident-set.1
     1. pevná alokace – fixed-allocation policy
        1. procesu je při nahrávání vyhrazen pevný počet rámců RAM (podle kritérií)
           1. rovné či proporcionální rozdělení rámců mezi procesy
           2. při page fault, se musí uvolnit rámec stejného procesu
     2. proměnná alokace – variable-allocation policy
        1. počet rámců procesu se průběžně může měnit
           1. zvětšuje se pří vysoké frekvenci výskytů page fault
           2. snižuje se při nízké frekvenci výskytů page fault
        2. vyžaduje režii OS při odhadu chování procesů
2. Požadavky na OS pro práci v reálném čase, rozdělení RT OS, pojmy latence a odezva (na úrovni přerušení); vestavěné systémy, OS pro ně a typické požadavky na ně. (blok 12) / (pr 09)
   * Rozdělení systémů RT.3
     1. hard real time
        1. existují absolutní časové limity, při jejichž překročení je odezva zcela bezcenná, systém selže
     2. soft real time
        1. časové limity jsou pouze přibližné, jejich překročení pouze sníží užitečnost systému
     3. firm real time
        1. odezva po časovém limitu je bezcenná, nicméně systém může snést několik málo zmeškání
   * Typické vlastnosti RTOS, příklady.5 + 1
     1. rychlé přepínání kontextu
        1. např. s HW podporou – více sad registrů, aby se při přerušení nemusely ukládat registry do RAM
     2. prioritní plánování, preempce procesů i jádra
        1. umožňuje systému rychlou reakci na události
        2. typická je architektura mikrojádro
     3. malé rozměry
        1. OS obsahuje jen nejnutnější prvky
        2. často používané také jako vestavěný systém
     4. rychlý souborový systém
        1. rychlé čtení a ukládání dat snižuje dobu odezvy
     5. podpora speciálních systémových služeb
        1. alarm, timeout apod.
     6. spolehlivost
     7. multitasking s komunikací procesů (IPC)
        1. spolupracující procesy musí být schopny rychle komunikovat a vzájemně se synchronizovat
        2. semafory, signály, fronty zpráv, …
   * Definujte pojmy latence a odezva (na úrovni přerušovacího systému).2
     1. Odezva - za jak dlouho operační systém reaguje na požadavek přerušení
     2. Latence - doba mezi okamžikem příchodu požadavku na přerušení a okamžikem, kdy se začne provádět odpovídající obslužný program
   * Definice a typické vlastnosti (alespoň šest) vestavěných systémů.4
     1. počítačové systémy, které jsou součástí jiných (obvykle technických) systémů
        1. obvykle představují jejich řídicí složku
        2. nebo tvoří jejich podsystémy
     2. obvyklá je schopnost práce v reálném čase
     3. vlastnosti:
        1. **malá spotřeba** – časté je bateriové napájení a omezená možnost chlazení
        2. **odolnost** – kvůli přenosnosti je třeba, aby systémy odolaly mnohem větší škále nepříznivých vlivů: horko, mráz, vibrace, nárazy, kolísání napájení, rušení, blesky, vlhkost a zkrápění vodou, koroze a v neposlední řadě též nesprávné zacházení
        3. **malé rozměry a váha** – mnoho vestavěných systémů je přenosných, vyskytují se jako součásti dopravních prostředků, kde přebytek váhy znamená vyšší provozní náklady
        4. **reaktivita** – výpočty probíhají jako odezva na externí události, je třeba krátká odezva
        5. **funkce v reálném čase** – správnost je částečně funkcí času
        6. **spolehlivost a bezpečnost** – systémy musejí fungovat správně, ale hlavně nesmí fungovat nepřijatelně
        7. **cenová citlivost** – snížení ceny o jednotky až desítky Kč může znamenat zvýšení prodeje i o miliony kusů
3. Víceprocesorové systémy, rozdělení dle vazby a dle symetrie, granulovatelnost úlohy, souvislost s vazbou (víceprocesorových systémů) a stupně paralelismu, distribuované (rozptýlené, clusterové) OS. (blok 12) / (pr 09)
   * Kategorie počítačových systémů z hlediska paralelizace zpracování dat.4
     1. SISD (single instruction, single data)
        1. jeden procesor zpracovává jednu množinu dat jedním proudem instrukcí
     2. SIMD (single instruction, multiple data)
        1. jedním proudem instrukcí zpracovává více procesorů více různých množin dat
           1. každá instrukce se provede současně v n procesorech
           2. každý procesor zpracovává jiná data
   * Rozdělení víceprocesorových systémů dle vazby.2
     1. **S volnou vazbou** (loosely coupled)
        1. každý procesor má vlastní operační paměť a vstupně-výstupní subsystém.
        2. typy vazby:
           1. společná sběrnice – Common Bus (např. VME)
           2. společné disky – Common Disk
           3. nic společného – Common Nothing (vazba LAN).
     2. **S těsnou vazbou** (tightly coupled)
        1. procesory sdílejí operační paměť
        2. řízen jedním operačním systémem
   * Rozdělení víceprocesorových systémů dle symetrie.2
     1. **Symetrické** – SMP (Symmetric MultiProcessing)
        1. shodné procesory
        2. Jádro OS, procesy i vlákna mohou být prováděny libovolným procesorem.
     2. **Asymetrické**
        1. procesory jsou funkčně specializované:
           1. FPU, grafické procesory, V/V procesory
        2. Jádro a většina úloh běží na CPU, některé specializované úlohy či jejich části jsou zpracovávané na specializovaných procesorech.
        3. systém je řízen centrálním procesorem
   * Granularita úlohy: účel, jak se pozná, že úloha je/není granulovatelná.4
     1. každou úlohu můžeme rozčlenit na úseky, které lze provést samostatně
        1. takové úseky lze ve víceprocesorovém systému provádět paralelně na různých procesorech
     2. pokud jeden úsek potřebuje pro svoji činnost výsledky jiného, musí na tyto výsledky čekat
        1. je nutná komunikace a synchronizace
   * Granularita a vhodné stupně vazby.3
     1. hrubě granulovatelná úloha – méně úseků
        1. vhodné jsou kooperující procesy
     2. jemněji granulovatelná úloha – úseky kratší
        1. požadavky na komunikaci a synchronizaci častější
        2. vhodnější jsou vlákna – pokud běží na oddělených procesorech, přinášejí výrazné zvýšení výkonnosti
4. Soutěžení procesů (o prostředky), obecné problémy souběhu, vzájemné vylučování, kritická sekce, předpoklady pro řešení KS, požadované vlastnosti řešení KS, typy řešení. (blok 5) / (pr 05)
   * Definice kritické sekce v programu.4
     1. část kódu, kde proces manipuluje se sdíleným prostředkem a současná manipulace jiným procesem by vedla k problému (nekonzistenci)
     2. Struktura programu s KS
        1. **vstupní sekce** (entry section)
           1. implementace povolení vstupu do KS
        2. **kritická sekce** (critical section)
           1. manipulace se sdílenými prostředky
        3. **výstupní sekce** (exit section)
           1. implementace uvolnění KS pro jiné procesy
        4. **zbytková sekce** (remainder section)
           1. zbytek kódu procesu
   * Předpoklady pro řešení přístupu do kritické sekce. (Co se předpokládá, že platí.)4
     1. procesy se provádějí nenulovou rychlostí
     2. žádné předpoklady o relativní rychlosti procesů
     3. uvažujeme i víceprocesorové systémy
        1. předpoklad: paměťové místo smí v jednom okamžiku zpřístupnit vždy pouze jediný procesor
     4. žádné předpoklady o prokládaném provádění
        1. procesy se nemusí pravidelně střídat v běhu
     5. stačí specifikovat vstupní a výstupní sekci
   * Požadované vlastnosti řešení přístupu do kritické sekce (s vysvětlením).3
     1. **vzájemné vylučování** (mutual exclusion)
        1. vždy jediný proces v KS
     2. **pokrok v přidělování** (progress)
        1. na rozhodování o vstupu do volné KS se mohou podílet výhradně procesy, které nejsou ve ZS
        2. toto rozhodnutí musí padnout v konečném čase
        3. volná KS ⇒ požadavku musí být vyhověno
     3. **omezené čekání** (bounded waiting)
        1. mezi požadavkem na vstup do KS a vyhověním smí do KS vstoupit pouze omezený počet jiných procesů
   * Typy řešení přístupu do kritické sekce (s příklady).4
     1. **SW řešení**
        1. použití algoritmu pro vstupní a výstupní sekci
           1. aktivní čekání
           2. Petersonův algoritmus
           3. Algoritmus bakery
     2. **HW řešení**
        1. využití speciálních instrukcí procesoru
           1. zákaz přerušení
           2. instrukce test-and-set
           3. instrukce xchg
     3. **řešení OS**
        1. nabízí programátorovi prostředky pro řešení KS (datové typy a funkce)
           1. semafor
           2. binární semafor (mutex)
     4. **řešení programovacího jazyka**
        1. konkurenční / souběžné programování
5. Řízení přístupu do kritické sekce pomocí SW metod, příklady nevhodných (obecně nefunkčních) SW řešení, SW algoritmy, vlastnosti (nedostatky) SW metod. (blok 5) / (pr 05)
   * SW algoritmus s proměnnou locked, důvod nefunkčnosti.1 + 1
     1. sdílená proměnná locked
        1. udává obsazenost KS
        2. inicializovaná na 0
     2. proces Pi čeká, dokud je KS obsazena (locked ≠ 0)
        1. jakmile je KS volná, nastaví se locked = 1
     3. pokud oba procesy současně zjistí volnou KS, oba nastaví obsazeno a vstoupí do ní
        1. **požadavek vzájemného vylučování není splněn!**
   * SW algoritmus s proměnnou turn, důvod nefunkčnosti.1 + 1
     1. sdílená proměnná turn
        1. inicializovaná na 0 nebo 1
     2. kritická sekce (KS) procesu Pi se provádí, jestliže turn = i
     3. proces Pi aktivně čeká (busy waiting), když je proces Pj v kritické sekci
        1. **požadavek vzájemného vylučování je** **splněn**
     4. **požadavek pokroku není splněn!**
        1. vyžadována alternace kritických sekcí
   * SW algoritmus s proměnnými flag[*i*], důvod nefunkčnosti.1 + 1
     1. sdílená proměnná flag – pro každý proces: flag[i]
     2. požadavek vstupu do KS: flag[i] = true
     3. **požadavek vzájemného vylučování je splněn**
     4. **požadavek pokroku není splněn!**
        1. po sekvenci: P0 : flag[0] = true;
        2. P1 : flag[1] = true; **– DEADLOCK**
   * Funkční SW algoritmy pro řízení přístupu do KS, krátká charakteristika.2 + 2
     1. Petersonův algoritmus
        1. Inicializace
           1. flag[i] = false, i = 0..1
           2. turn = 0 nebo 1
        2. signalizace připravenosti ke vstupu do KS nastavením flag[i] = true
        3. pokud se oba procesy pokusí vstoupit do KS současně, pouze jeden bude mít potřebnou hodnotu proměnné turn
     2. Leslie Lamport's bakery algorithm
        + 1. každý proces dostane před vstupem do KS číslo
          2. držitel nejmenšího čísla smí vstoupit do KS
          3. dostanou-li Pi a Pj stejná čísla, přednost má Pmin(i, j)
          4. ve výstupní sekci nastaví proces přidělené číslo na 0
        1. poznámky k zápisu:
           1. (a, b) < (c, d), když a < c nebo když a = c a b < d
           2. max(a0 , …, ak ) je takové b >= ai pro i = 0, …, k
        2. sdílená data
           1. bool choosing[n]; – proces vybírá číslo

inicializace všech na false

* + - * 1. int number[n]; – přidělené číslo číslo

inicializace všech na nulu

* + - 1. korektnost algoritmu závisí na faktu
         1. je-li Pi v KS a Pk si právě vybral své číslo, pak (numberi , i ) < (numberk , k)
  + Nevýhody SW algoritmů (včetně vysvětlení).5
    1. splněním všech tří kritérií (vzájemné vylučování, pokrok, omezené čekání) je řešení odolné vůči chybám ve zbytkové sekci (ZS)
       1. chyba ve ZS je totéž, co neomezeně dlouhá ZS
    2. řešení však nemůže zajistit odolnost vůči chybám v KS
       1. pokud proces Pi havaruje v KS,
       2. pro ostatní procesy je stále v KS
       3. a ty se do ní nedostanou
    3. Softwarové řešení má také další nedostatek, a to je aktivní čekání (označuje se též jako spin lock). Procesy čekající na vstup do obsazené KS spotřebovávají v cyklu zbytečně a neproduktivně čas procesoru. Toto čekání se nepříznivě projeví zvláště tehdy, pokud je KS dlouhá. Mnohem efektivnější by bylo čekající procesy blokovat. Na to je však potřeba podpora operačního systému. Z toho plyne, že softwarové řešení je vhodné pouze tehdy, když je KS velmi krátká.
    4. Použije-li se softwarové řešení v systému se striktně prioritním plánováním, může snadno dojít k uváznutí (deadlock): Bude-li proces s nízkou prioritou přerušen při provádění KS procesem s vyšší prioritou, který bude rovněž požadovat vstup do KS, bude proces s vyšší prioritou aktivně čekat, ale nikdy se nedočká, neboť proces s nižší prioritou nedostane šanci KS opustit.

1. Řízení přístupu do kritické sekce pomocí HW metod, výchozí předpoklady pro HW řešení, algoritmy využívající HW instrukce, vlastnosti (nedostatky) HW metod. (blok 5) / (pr 05)
   * Hardwarové předpoklady pro řízení přístupu do KS.3
     1. procesy se provádějí v procesoru kontinuálně, dokud nevyvolají službu OS nebo nejsou přerušeny
     2. k přerušení procesu může dojít pouze na hranicích instrukcí
        1. mezi dokončením jedné instrukce a zahájením další
     3. přístup k paměti je obvykle výlučný
        1. důležité zejména pro systémy SMP
   * HW podpora pro řízení přístupu do KS, možná řešení.2 + 3
     1. Zákaz přerušení
        1. Proces běžící na procesoru nemůže být nijak přerušen jiným procesem, dokud není aktivován plánovač operačního systému.
        2. Plánovač může být aktivován dvěma způsoby:
           1. proces zavolá službu OS (systémové volání) – jádro OS může aktivovat plánovač
           2. časovač vygeneruje signál přerušení a procesor skočí na obslužnou rutinu do jádra OS – aktivuje se plánovač
        3. Pokud v KS není volána služba OS, splňuje toto řešení podmínku vzájemného vylučování.
        4. Na víceprocesorových systémech ovšem zákaz přerušení nepomůže, neboť během vykonávání KS může na dalším procesoru běžet jiný proces, který také může vstoupit do KS. Další nevýhodou je zvyšování latence systému. Během zákazu přerušení se pozdrží jakékoli reakce systému a zhorší se tím odezva. Zákaz a povolení přerušení jsou navíc instrukce povolené pouze v privilegovaném režimu, nedají se tedy používat univerzálně.
        5. Zákaz přerušení pro řízení přístupu do kritické sekce je obecně nevhodné řešení.
     2. Speciální instrukce
        1. přístup k paměťovému místu je obvykle výlučný
        2. lze tedy navrhnout instrukci, která atomicky provede dvě akce s jedním paměťovým místem
           1. čtení a zápis jako nedělitelná operace
        3. provedení instrukce nelze přerušit
           1. context switch probíhá pouze na hranicích instrukcí
           2. zaručí nám tak vzájemné vylučování, a to i na víceprocesorových systémech
           3. ostatní požadavky je třeba řešit algoritmicky
        4. instrukce test-and-set
           1. jediná instrukce procesoru přečte příznak a současně ho nastaví

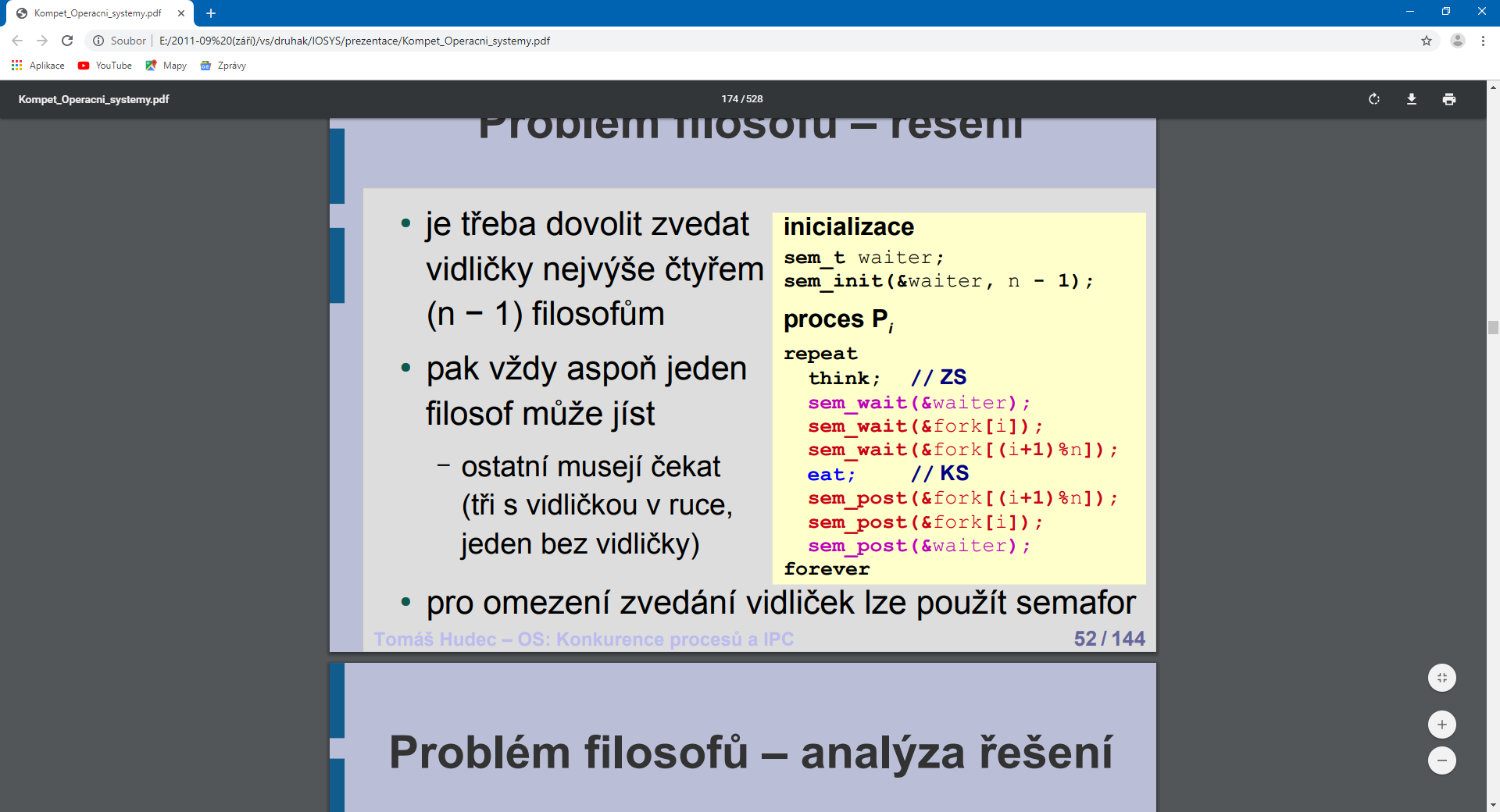
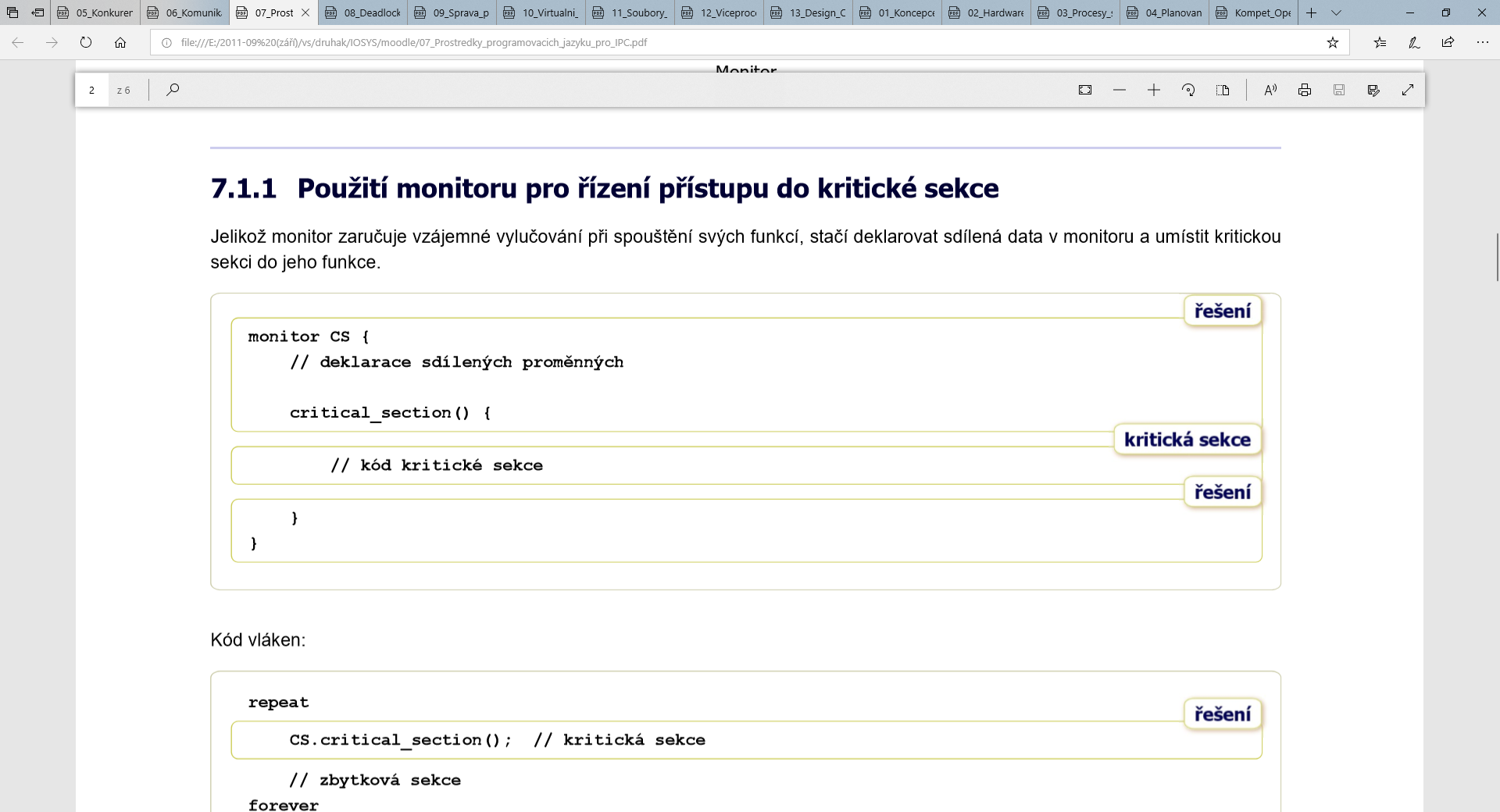
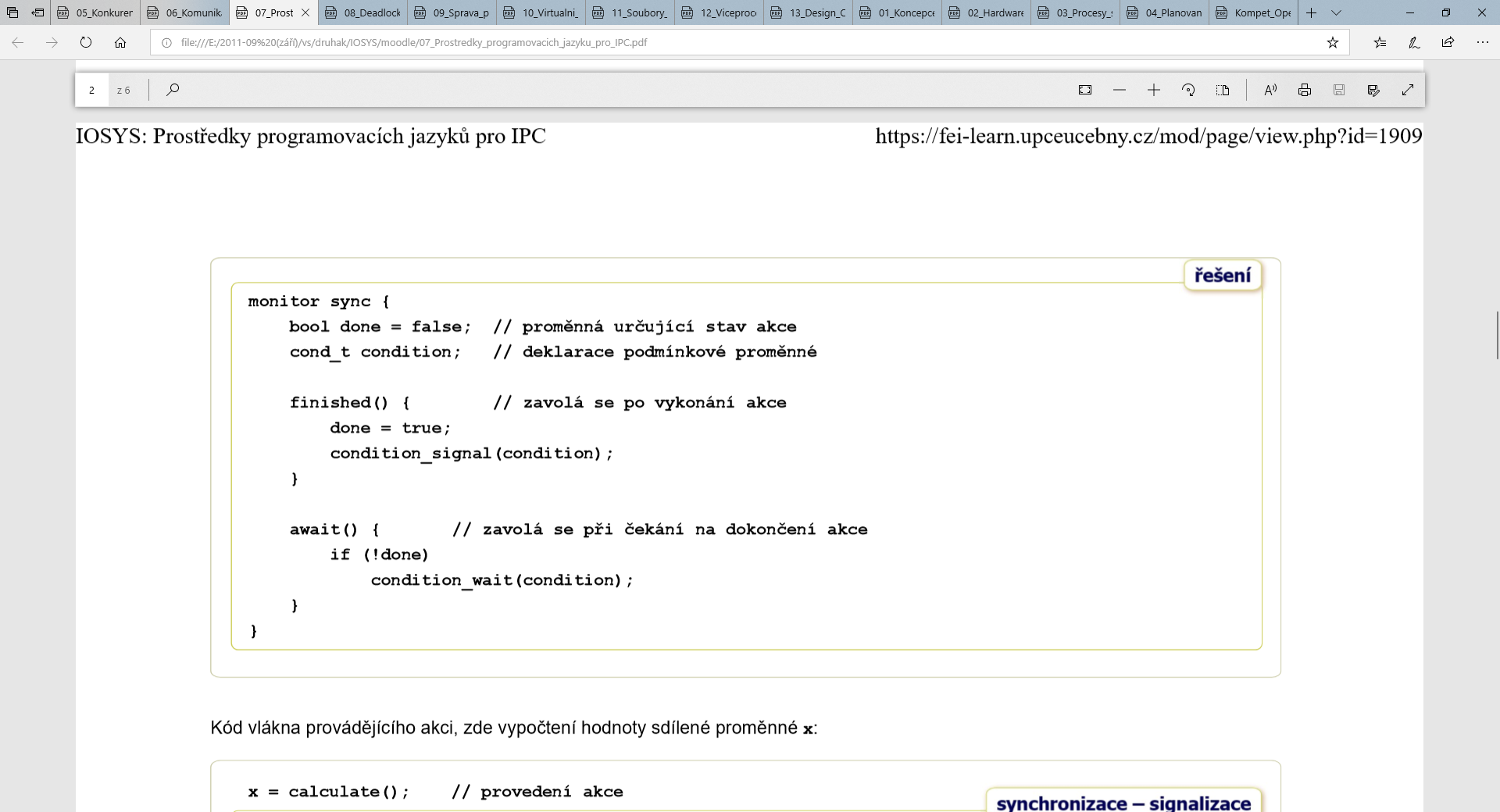
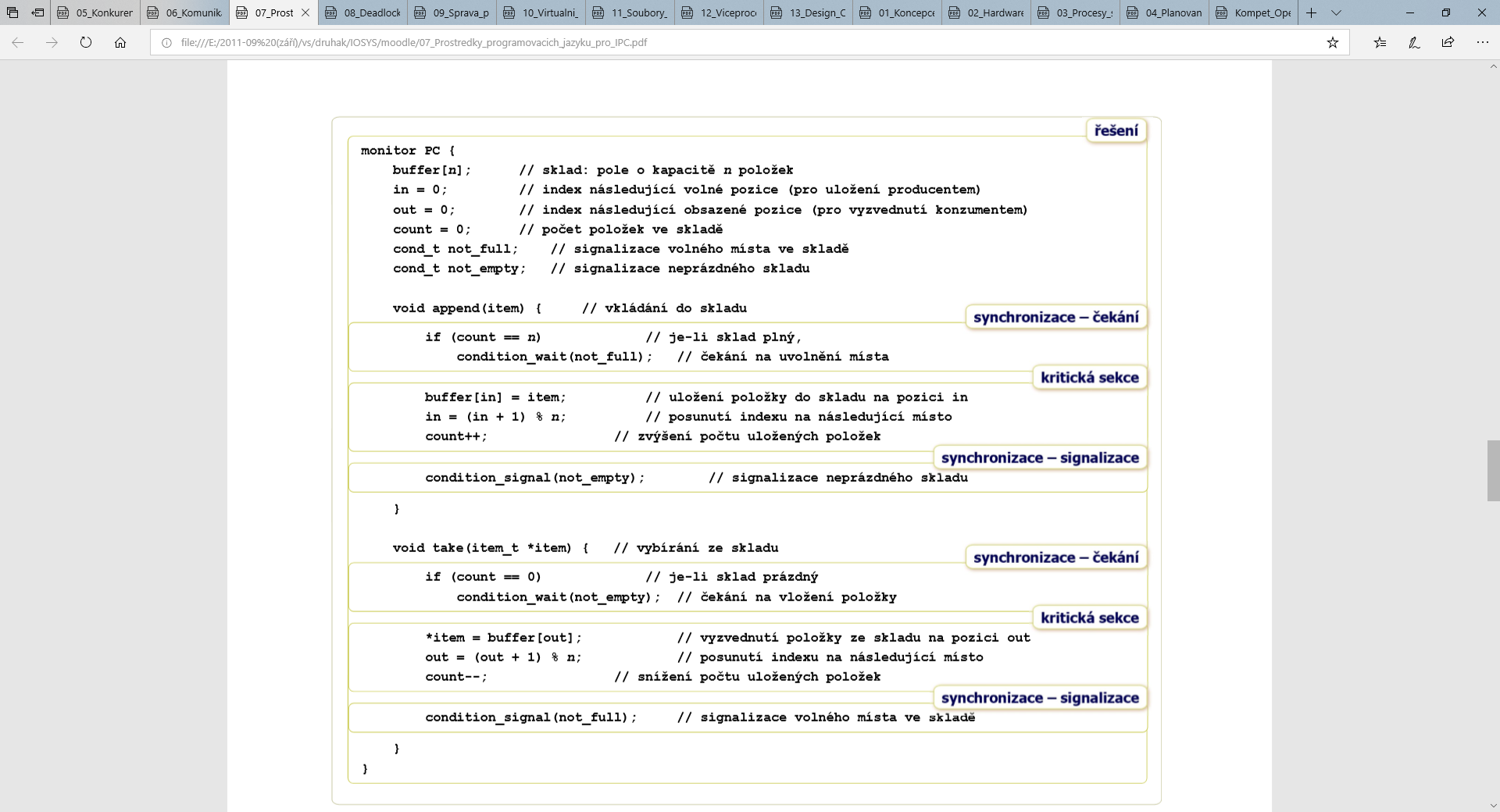
byl-li příznak již nastaven, nové nastavení nic nemění

KS je obsazena

nebyl-li příznak nastaven, proces smí vstoupit do KS

instrukce je nepřerušitelná, proto jiný proces vyhodnotí touto instrukcí příznak správně

* + - * 1. lze využít pro více různých KS
      1. instrukce xchg
  + Vlastností (nedostatky) HW řešení.4 + 3
    1. Hardwarová řešení neodstraňují nedostatek softwarových řešení
       1. je stále třeba aktivního čekání.
       2. Řeší pouze vzájemné vylučování, ostatní požadované vlastnosti řízení přístupu do kritické sekce je třeba řešit algoritmicky.
       3. nabízejí velmi krátkou vstupní i výstupní sekci
          1. Instrukce test-and-set pak má další výhodu na víceprocesorových systémech s pamětí cache, kdy obvykle způsobí aktualizaci cache na ostatních procesorech, což je nezbytné pro správnou funkci algoritmů.

1. Nástroj OS: semafor, jeho popis včetně systémových volání, použití semaforu pro řízení přístupu do KS a pro synchronizaci, problém obědvajících filozofů. (blok 5) / (pr 05)
   * Popište nástroj OS: semafor.2
     1. synchronizační nástroj – prostředek OS
     2. nevyžaduje aktivní čekání
        1. dokud je KS obsazená
           1. čekající proces je blokován a
           2. zařazen do fronty procesů čekajících na uvolnění KS
        2. po uvolnění KS je z fronty vybrán další proces
   * Vysvětlete systémová volání semaforu.1 + 2 + 2
     1. init inicializuje čítač
     2. wait snižuje čítač
        1. je-li záporný
           1. zařadí volající vlákno do fronty a blokuje je
     3. signal zvyšuje čítač
        1. je-li fronta neprázdná
           1. vybere vlákno z fronty
           2. zařadí je do seznamu připravených vláken
   * Popište řešení KS používající semafor.3
     1. sdílený semafor s se inicializuje na počet procesů smějících vstoupit do KS
        1. typicky na jeden
     2. vstupní sekce volá wait
     3. výstupní sekce volá signal
     4. obecně lze povolit neblokující volání wait pro n procesů
        1. inicializujeme-li semafor na n
   * Popište řešení synchronizace vláken pomocí semaforu.3
     1. příkaz S2 v procesu P1 musí být proveden až po provedení S1 v P0
     2. čítač semaforu sync inicializujeme na 0
     3. proces P1 před provedením S2 počká na signál od P0
     4. provede-li se nejprve signal, pak wait neblokuje
   * Definujte problém obědvajících filozofů a vyřešte jej pomocí semaforů.2
     1. u stolu sedí pět filosofů
        1. každý buď přemýšlí,
        2. nebo jí
     2. při jídle každý potřebuje dvě vidličky
     3. k dispozici je pouze pět vidliček
     4. řešení
     5. je třeba dovolit zvedat vidličky nejvýše čtyřem (n − 1) filosofům
     6. pak vždy aspoň jeden filosof může jíst
        1. ostatní musejí čekat (tři s vidličkou v ruce, jeden bez vidličky)
     7. pro omezení zvedání vidliček lze použít semafor
     8. 
2. Nástroj OS: předávání zpráv, popis systémových volání a možností blokování, použití fronty zpráv pro řízení přístupu do KS a pro synchronizaci, problém svázaných producentů a konzumentů. (blok 6) / (pr 05)
   * Popište nástroj OS: předávání zpráv.3
     1. komunikační prostředek OS pro procesy
     2. je nutné vzájemné vylučování
        1. dochází k výměně informací
        2. zajišťuje OS
     3. systémová volání
        1. send(cíl, zpráva)
        2. receive(zdroj, zpráva)
   * Popište typickou implementaci blokování systémových volání pro předávání zpráv.3
     1. typická implementace blokování **send** a **receive** pro fronty zpráv s omezenou velikostí
     2. neblokující **send**
        1. blokuje pouze při zaplnění fronty zpráv
     3. blokující **receive**
        1. příjemce je blokován, není-li dostupná žádná zpráva
     4. neblokující varianty lze nastavit parametrem, volání pak místo blokování vrací chybu
   * Popište řešení KS používající frontu zpráv.3
     1. sdílená fronta mutex se inicializuje zasláním jedné zprávy
     2. vstupní sekce volá blokující receive
     3. výstupní sekce volá neblokující send
     4. první proces, který provede receive, se dostane do KS, ostatní jsou blokovány
   * Popište řešení synchronizace vláken pomocí fronty zpráv.3
     1. příkaz S2 v procesu P1 musí být proveden až po provedení S1 v P0
     2. vyprázdníme mailbox sync
     3. P1 před provedením S2 volá blokující receive
     4. P0 po provedení S1 volá neblokující send
     5. provede-li se nejprve send, receive neblokuje
   * Definujte problém svázaných producentů a konzumentů a vyřešte jej pomocí fronty zpráv.3
     1. do fronty zpráv storage zasílají producenti položky
        1. jedná se o sklad (buffer)
     2. kapacita skladu je daná maximální velikostí fronty
        1. zaplní-li producenti sklad, bude send blokovat
     3. konzumenti vybírají položky voláním receive
     4. funguje pro více producentů i konzumentů
3. Nástroj programovacích jazyků: koncept monitoru, problém producentů a konzumentů a jeho řešení pomocí monitoru. (blok 7) / (pr 05)
   * Popište koncept monitoru: jeho účel, strukturu a základní vlastnosti.2 + 2
     1. konstrukce ve vyšším programovacím jazyce pro stejné služby jako semafory (synchronizace a vzájemné vylučování), snadněji ovladatelné
     2. SW modul (podobný objektu / třídě)
        1. lokální proměnné – sdílená data
           1. tato data nejsou viditelná vně monitoru
        2. funkce zpřístupňující lokální data
        3. inicializační část
     3. v monitoru (jeho funkci) smí být v daném okamžiku pouze jediné vlákno
        1. monitor tak zajišťuje vzájemné vylučování
        2. synchronizaci lze zajistit podmínkovými proměnnými
     4. synchronizační nástroj monitoru
     5. podmínkové proměnné jsou lokální v monitoru a dostupné pouze pomocí funkcí monitoru
     6. lze je měnit pouze dvěma funkcemi monitoru
        1. cwait(cv) – blokuje vlákno, dokud není zavoláno:
        2. csignal(cv) – obnoví provádění vlákna blokovaného podmínkovou proměnnou cv
           1. je-li takových vláken více, vybere se jedno z nich
           2. **není-li žádné takové vlákno, neprovede se nic**
   * Popište řešení KS pomocí monitoru.2
     1. Jelikož monitor zaručuje vzájemné vylučování při spouštění svých funkcí, stačí deklarovat sdílená data v monitoru a umístit kritickou sekci do jeho funkce.
     2. 
   * Popište řešení synchronizace vláken pomocí monitoru.3
     1. Pro synchronizaci lze použít podmínkové proměnné monitoru v kombinaci s proměnnou vypovídající o stavu dokončení akce.
     2. 
   * Definujte problém svázaných producentů a konzumentů a vyřešte jej pomocí monitoru, zaměřte se též na podmínky pro synchronizaci a zdůvodněte její použití.6
     1. Problém:
        1. Přístup do skladu musí být řízen tak, aby producenti ukládali data pouze na volná místa a konzumenti vybírali každý jiná data z obsazených míst. Konzumenti musejí čekat, pokud je sklad prázdný, producenti musejí čekat, pokud je zaplněný.
     2. Řešení:
        1. Stačí definovat sdílená data uvnitř monitoru (monitor zajistí vzájemné vylučování) a pro synchronizaci použít podmínkové proměnné. Funkce pro vložení a výběr položek budou součástí monitoru.
        2. Vzájemné vylučování je zaručeno, neboť manipulace se sdílenými daty probíhá výhradně v monitoru.
        3. 
4. Nástroje knihovny posixových vláken: mutex, bariéra, podmínková proměnná. (blok 7) / (pr 05)
   * Popište mutex, jeho účel, způsob použití a vlastnosti.2 + 3
     1. místo čítače je použita Booleovská proměnná
        1. inicializace na false
     2. mutex – odvozeno od využití pro vzájemné vylučování – mutual exclusion
        1. nelze jej (samostatně) využít pro synchronizaci
     3. typicky se operace wait a signal označují termíny lock a unlock
     4. mutex je zámek, který zaručuje
        1. atomicitu a vzájemné vylučování
           1. lze zamknout pouze jediným vláknem
        2. neaktivní čekání
           1. vlákno je při pokusu zamknout zamčený mutex blokováno
     5. tři typy mutexů (dva jsou rozšiřující)
        1. fast mutex – lze zamknout pouze jednou
        2. recursive mutex – lze zamknout „na více západů“
        3. error checking mutex – zamykání zamčeného selže
   * Popište bariéru, její účel, způsob použití a vlastnosti (včetně návratové hodnoty funkce čekání).2 + 3
     1. umožňuje vláknu neaktivně čekat na ostatní vlákna
     2. jakmile k bariéře dospěje daný počet vláken, bariéra propustí všechna vlákna – paralelní běh
     3. definováno normou POSIX.1-2001 a Single UNIX Specification, Version 3
     4. nutno definovat jeden z následujících symbolů před všemi direktivami #include
        1. #define \_XOPEN\_SOURCE 600 // obvykle tento
        2. #define \_POSIX\_C\_SOURCE 200112L
     5. návratová hodnota čekací funkce, která vrátí právě jednomu vláknu hodnotu PTHREAD\_BARRIER\_SERIAL\_THREAD a ostatním hodnotu nula. Jiná hodnota pak znamená chybový stav.
   * Popište podmínkovou proměnnou, její účel, způsob použití a vlastnosti.2 + 3
     1. slouží k synchronizaci vláken
     2. umožňuje vláknu neaktivně čekat na událost
     3. nezaručuje exkluzivitu přístupu
        1. je třeba k ní přistupovat pomocí mutexu
     4. na událost může čekat i několik vláken
     5. událost oznamuje některé vlákno signálem
        1. signál může probudit jediné vlákno
        2. lze poslat i všesměrový signál a probudit všechna vlákna
        3. nečeká-li žádné vlákno, je signál ztracen
           1. proto se používá podmínková proměnná společně s testem, zda je třeba čekat
           2. Protože test na potřebu čekání nutně používá sdílenou proměnnou, je nutné zajistit vzájemné vylučování například pomocí mutexu
5. Stav uváznutí (deadlock) a vyhladovění (definice a rozdíl), nutné podmínky pro vznik stavu uváznutí, předcházení a řešení problému stavu uváznutí, algoritmus bankéře. (blok 8) / (pr 06)
   * Definujte stav uváznutí (deadlock).2
     1. skupina procesů je ve stavu uváznutí, když každý proces ve skupině čeká na událost, kterou může vyvolat pouze jiný proces ze skupiny
     2. protože událost nelze vyvolat jinak než procesem ve skupině, budou tyto procesy nekonečně čekat
     3. předpokládáme, že procesy nelze probudit asynchronně
        1. např. signálem, při jehož obsluze by se mohla událost vyvolat
   * Definujte stav vyhladovění (starvation), uveďte příklad.1
     1. nekončící čekání procesu na přidělení prostředku, proces nečiní žádný pokrok
        1. proces v soupeření o prostředek neustále prohrává
        2. např. nízká priorita při soutěži o procesorový čas
     2. nejedná se o deadlock
        1. proces by učinil pokrok, kdyby jej ostatní procesy neustále „nepředbíhaly“
        2. v jistém smyslu lze však uvažovat, že důsledkem stavu uváznutí (deadlock / livelock) je vyhladovění
   * Vysvětlete podmínky pro vznik stavu uváznutí.4
     1. vzájemné vylučování (mutual exclusion)
        1. prostředky lze vlastnit pouze jediným procesem
     2. alokace a čekání (hold and wait)
        1. proces vlastnící prostředek může požadovat další
     3. neodnímatelné prostředky (non-preemptable)
        1. OS je nemůže odejmout, musí být explicitně uvolněny vlastnícím procesem
     4. cyklické čekání (circular wait)
        1. řetěz vzájemně čekajících procesů uzavírá cyklus
   * Navrhněte způsoby řešení stavu uváznutí.3
     1. ignorování problému (pštrosí přístup)
     2. detekce a obnovení
        1. obnova (rollback) stavu bez uváznutí (checkpoint)
        2. násilné odebrání prostředku
        3. zabití některého procesu (př.: Linux OOM-killer)
     3. vyloučení možnosti vzniku (prevence)
        1. negování alespoň jedné z předešlých podmínek
           1. splnění všech předešlých podmínek je nutné, aby vznikl deadlock – stačí vyloučit jedinou
   * Navrhněte způsoby prevence vzniku stavu uváznutí.3
     1. Negace vzájemného vylučování
        1. spooling (např. tiskárna) (Simultaneous Peripheral Operations On-Line)
        2. Jedná se o vytvoření fronty požadavků typicky pro přenos dat na periferie. Daný prostředek je vlastněn trvale jediným procesem obsluhujícím postupně tyto požadavky
     2. Negace alokace a čekání
        1. procesy přinuceny alokovat všechny zdroje naráz
        2. Pokud proces požaduje další prostředek, nejprve musí uvolnit všechny prostředky a pak je může požadovat společně s dalším.
        3. Dokud nejsou všechny požadované zdroje dostupné, proces je blokován.
        4. Nevýhodou je, že blokované procesy mohou na přidělení prostředků čekat velmi dlouho.
        5. Další nevýhodou je, že některé prostředky zůstávají dlouho nevyužity, i když by využity být mohly.
     3. Negace neodnímatelných prostředků
        1. Pro některé prostředky lze zavést možnost násilného odebrání.
        2. Detekuje-li operační systém stav zablokování, jednomu ze zablokovaných procesů je odebrán násilně jeden z prostředků účastnící se blokace.
        3. Takovéto násilné odebrání samozřejmě není možné zavést u všech prostředků – např. vypalování média (DVD).
     4. Negace cyklického čekání
        1. Systém nesmí dovolit alokovat prostředek procesu, pokud taková alokace znamená zablokování. Jedno z možných řešení je definování lineárního uspořádání na prostředcích a přidělovat vždy pouze prostředky s vyšším pořadovým číslem, než jaké má jakýkoli procesu již přidělený prostředek.
        2. Podobně jako negace podmínky postupné alokace má tento způsob nevýhodu v tom, že může nastat zbytečné odmítnutí požadavku na přidělení a některé prostředky zůstanou dlouho nevyužity.
     5. Prevence vzniku zablokování dovolením pouze bezpečných stavů
        1. Bezpečný stav můžeme definovat jako situaci, kdy není stav zablokování a existuje plánovací pořadí, při kterém všechny procesy mohou být dokončeny, i když všechny procesy naráz budou požadovat maximum svých potřebných prostředků. Předpokládá se, že proces alokuje své deklarované maximum a pak se ukončí.
        2. Nebezpečný stav je pak situace, kdy v budoucnu může (ale nemusí vždy) nastat zablokování: Pokud každý proces bude požadovat pro své dokončení maximum svých deklarovaných zdrojů a ani jednomu nebude moci systém vyhovět, jedná se o stav, který není bezpečný.
   * Popište bankéřův algoritmus aplikovaný na procesy.2
     1. algoritmus publikoval Dijkstra 1965
     2. řešení situace bankéře při jednání s klienty, kterým poskytuje půjčku
        1. pokud požadavek klienta vede k nebezpečnému stavu, je tento požadavek odmítnut
     3. výchozí předpoklady
        1. pevný počet prostředků
        2. každý proces deklaruje své maximální požadavky
        3. postupná alokace prostředků
     4. prostředek je na požadavek přidělen jen tehdy, vede-li situace do bezpečného stavu
     5. slabiny algoritmu
        1. nelze vždy garantovat pevný počet prostředků
        2. proces nemusí znát maximální požadavky předem
     6. důsledek: algoritmus je mnohdy prakticky nepoužitelný
6. IPC: komunikace procesů a vláken, možné prostředky komunikace. (pr 05)
   * Vyjmenujte možné prostředky pro komunikaci procesů / vláken.5
     1. soubor, databáze
     2. roura
     3. socket
     4. fronty zpráv
     5. sdílená paměť
   * Krátce charakterizujte jmenované prostředky.5
     1. soubor, databáze
        1. pomalé, náhodný přístup, současný přístup je třeba řídit
     2. roura
        1. proudový přístup (FIFO), jednosměrná komunikace
     3. socket
        1. proudový přístup (FIFO), obousměrná síťová komunikace
     4. fronty zpráv
        1. exkluzivní přístup (FIFO), (jednosměrná) komunikace
     5. sdílená paměť
        1. nejrychlejší, náhodný přístup, současný přístup nutno řídit
   * Popište funkce pro sokety.5
     1. klientský proces a socket
        1. překlad adresy – getaddrinfo(3) / gethostbyname(3)
        2. alokace socketu – socket(2)
        3. svázání socketu s lokálním portem – bind(2)
           1. pouze volitelně, jinak OS přiřadí port automaticky
        4. navázání spojení (proudový socket) – connect(2)
           1. pro datagramový socket nastaví jen výchozí adresu cíle
        5. komunikace – write(2) /   send(2), read(2) /   recv(2)
           1. pro datagramový sendto(2) a recvfrom(2)
        6. zavření – close(2)
     2. serverový proces a socket
        1. nastavení lokální adresy a portu – getaddrinfo(3)
        2. alokace socketu – socket(2)
        3. svázání socketu s lokálním portem – bind(2)
        4. zahájení naslouchání – listen(2)
        5. přijetí spojení (pouze proudový socket) – accept(2)
           1. vytvoří nový socket pro komunikaci s klientem
        6. komunikace – read(2) /   recv(2), write(2) /   send(2)
           1. pro datagramový recvfrom(2) a sendto(2)
        7. zavření – close(2)
7. Dělení disku na oddíly, zavaděč OS, důvody dělení, MBR, GPT, swap. (pr 08)
   * Popište důvody pro rozdělení disku na oddíly.4
     1. jiný souborový systém (např. pro jiný OS)
     2. snadnější správa – oddělení systému od dat
        1. např. adresáře uživatelů nemohou zaplnit systémový disk
     3. možnost snadnější obnovy části dat při přepsání či při HW chybě – chyba se může týkat jen některého oddílu
     4. oddíl pro swap – na rozdíl od souboru oddíl netrpí datovou fragmentací – rychlejší přístup
   * Popište MBR a způsob dělení disku na oddíly.3
     1. první sektor disku
     2. tabulka rozdělení disku na oddíly (čtyři záznamy)
        1. primární – lze z něj zavést OS sekundárním zavaděčem
        2. rozšířený – dělí se na logické disky
     3. zavaděč systému, může být univerzální – skok na sekundární zavaděč na aktivním primárním oddíle
   * Popište typů oddílů (pro dělení MBR).3
     1. primární – lze z něj zavést OS sekundárním zavaděčem
     2. rozšířený – dělí se na logické disky
   * Charakterizujte GPT.3
     1. (GUID Partition Table) – standard UEFI
        1. za „protective MBR“ + záloha na konci disku
        2. až 128 (primárních) oddílů (1 záznam má 128 B)
        3. odstraňuje limity MBR, nutné pro disky nad 2 TiB
   * Vysvětlete rozdíly mezi odkládacím prostorem (swap) na samostatném diskovém oddíle a v souboru.2
     1. oddíl pro swap – na rozdíl od souboru oddíl netrpí datovou fragmentací – rychlejší přístup
8. Souborový systém, metadata, speciální soubory. (blok 11) / (pr 08)
   * Popište obecnou strukturu souborových systémů (metadata souborových systémů a souborů).2 + 3
     1. umožňuje uživatelům a procesům přístup k souborům (podle jména) na paměťovém médiu
     2. vytváří logickou (adresářovou) strukturu souborů
     3. eviduje metadata souborů (atributy, oprávnění)
     4. organizuje uspořádání souborů na médiu
        1. programátoři nemusejí vyvíjet vlastní SW prostředky pro manipulaci s daty na médiu, alokaci místa apod.
     5. sjednocuje přístup k datům různého typu a původu
        1. vstup a výstup na různé periferie se neliší
     6. metadata souborového systému (FS)
        1. velikost FS, seznam volných bloků, odkaz na evidenci metadat souborů
     7. metadata souborů – atributy
        1. typ, velikost, čas změny, oprávnění, umístění, …
   * Uveďte atributy souborů (alespoň šest).3
     1. read-only – soubor je pouze pro čtení, nelze smazat ani přepsat
     2. hidden – skrytý, nezobrazuje se ve výpise adresáře
     3. system – systémový, podobně jako skrytý plus navíc další omezení (např. zákaz odstranění či přesunutí)
     4. archive – soubor je připraven k archivaci (záloze), atribut se nastavuje vždy při zápisu do souboru.
     5. not content indexed – neindexovaný obsah souboru
     6. off-line – data jsou na jiném (vzdáleném) zařízení (soubor není přístupný při provozu off-line), může též znamenat, že soubor je jen lokální kopií (cache)
     7. temporary – dočasný soubor
     8. compressed – automaticky komprimovaný soubor
     9. encrypted – šifrovaný pomocí EFS
     10. sparse – řídký soubor.
   * Uveďte možná oprávnění na soubor (alespoň šest).3
     1. **žádné** – uživatel se souborem nemá právo jakkoli manipulovat, ani nemusí být schopen zjistit existenci souboru
     2. **znalost existence** – možnost zjistit metadata (jméno, velikost, vlastníka, oprávnění apod.)
     3. **provádění** (execute) – jedná-li se o uložený program, smí jej uživatel spustit, ale nemá právo jej číst (kopírovat)
     4. **čtení** (read) – soubor lze otevřít ke čtení pro libovolné účely (výpis, kopírování, někdy i provádění), ale nelze v souboru dělat změny
     5. **přidávání** (append) – do souboru lze přidávat další záznamy, ale nelze nijak modifikovat již uložená data
     6. **přepisování, zápis** (update, write) – lze zapisovat i přepisovat
     7. **mazání** (delete) – uživatel smí soubor odstranit
     8. **vytvoření** (create) – uživatel smí v adresáři vytvořit nový soubor
     9. **změna práv** (changing permissions) – uživatel může nastavit přístupová práva.
   * Popište zvláštní typy souborů (alespoň čtyři).2
     1. **Odkazy**
        1. V adresáři lze vytvořit novou položku (nové jméno), která se bude odkazovat na stejná data a metadata jako nějaký existující soubor.
        2. Takto vzniklá položka se nazývá tvrdý (pevný) odkaz (hard link). Jedná se tedy o nové jméno pro již existující soubor včetně jeho metadat.
        3. Nové a původní jméno nelze od sebe odlišit (nelze poznat, které jméno vzniklo dříve), položky jsou vzájemně ekvivalentní. Nejedná se tak o speciální typ souboru, ale o vlastnost souborového systému. V metadatech souboru se eviduje počet takovýchto odkazů. K uvolnění dat dojde až při odstranění posledního odkazu. Tvrdé odkazy nelze vytvářet na adresáře (mohlo by to vést k porušení stromová hierarchie). Tvrdé odkazy nelze vytvářet mezi různými souborovými systémy.
        4. Jiný typ odkazu je symbolický odkaz (symbolic link). Jedná se o speciální typ souboru, v jehož datové části je uvedeno nové jméno. Při přístupu k symbolickému odkazu provede systém textovou záměnu jmen (jméno symbolického odkazu je nahrazeno novým jménem). Tato změna je pro proces transparentní. Odkaz může být buď absolutní (nové jméno začíná lomítkem), nebo relativní. Symbolický odkaz se liší od zástupce (shortcut), což je normální soubor, který je interpretovaný grafickým uživatelským prostředím.
        5. Na souborovém systému NTFS lze vytvořit ještě tzv. junction point – speciální odkaz na (lokální) adresář.
     2. **Komunikační speciální soubory**
        1. Pro komunikaci mezi procesy lze použít rouru nebo socket. Obojí lze na některých souborových systémech vytvořit také jako speciální typ souboru. Jméno tohoto souboru je pak adresou pro komunikaci.
        2. Pojmenovaná roura (named pipe) je jednosměrný komunikační nástroj ve formě souboru. Tento speciální soubor má dva nezávislé ukazatele – jeden pro čtení a druhý pro zápis. Jeden proces smí zapisovat a druhý číst. Roura se chová jako FIFO. Co bylo přečteno se z ní odstraní.
        3. Alternativní komunikační nástroj je socket. Na rozdíl od roury umožňuje procesům komunikovat obousměrně. Jako adresu lze opět použít jméno souboru, které tak tvoří na souborovém systému speciální typ souboru, tzv. (POSIX Local) IPC socket. Na rozdíl od síťového soketu umožňuje tento typ komunikovat pouze procesům běžícím na stejném systému. Tento typ souborů je používán zejména unixovými systémy, proto se obvykle používá spíše pojem Unix domain socket.
     3. **Soubory pro HW zařízení**
        1. Unixové operační systémy používají pro jednotný přístup k hardwaru stejný model jako k souborům. Pro periferie tak vznikly speciální soubory, pomocí kterých mohou procesy k těmto periferiím přistupovat.
        2. Operace s periferiemi jsou stejné jako se soubory:
           1. zahájení práce – otevření souboru
           2. čtení nebo zápis dat,
           3. ukončení práce – zavření souboru.
        3. Rozlišují se dva typy zařízení: bloková (s náhodným přístupem po blocích) – disky – a znaková (proudový přístup po bajtech) – terminály, tiskárny, skenery, zvukové karty apod. Podle toho jsou tedy speciální soubory blokové a znakové.
     4. **Identifikace**
        1. POSIX: první znak na řádku ve výpise ls -l
        2. directory – adresář: jména souborů s čísly i-uzlů
        3. link – symbolický odkaz: obsahuje jméno cíle
           1. pozn.: hard link není speciální soubor
        4. pipe – pojmenovaná roura
        5. socket – pojmenovaný soket
        6. block – blokový speciální soubor: bloková zařízení
        7. char – znakový speciální soubor: znaková zařízení
           1. identifikace zařízení pomocí čísel (hlavní a vedlejší)
   * Charakterizujte typy odkazů a jejich reprezentaci na souborovém systému.1 + 1
     1. V adresáři lze vytvořit novou položku (nové jméno), která se bude odkazovat na stejná data a metadata jako nějaký existující soubor.
     2. Takto vzniklá položka se nazývá **tvrdý (pevný) odkaz** (hard link). Jedná se tedy o nové jméno pro již existující soubor včetně jeho metadat.
     3. Nové a původní jméno nelze od sebe odlišit (nelze poznat, které jméno vzniklo dříve), položky jsou vzájemně ekvivalentní. Nejedná se tak o speciální typ souboru, ale o vlastnost souborového systému. V metadatech souboru se eviduje počet takovýchto odkazů. K uvolnění dat dojde až při odstranění posledního odkazu. Tvrdé odkazy nelze vytvářet na adresáře (mohlo by to vést k porušení stromová hierarchie). Tvrdé odkazy nelze vytvářet mezi různými souborovými systémy.
     4. Jiný typ odkazu je **symbolický odkaz** (symbolic link). Jedná se o speciální typ souboru, v jehož datové části je uvedeno nové jméno. Při přístupu k symbolickému odkazu provede systém textovou záměnu jmen (jméno symbolického odkazu je nahrazeno novým jménem). Tato změna je pro proces transparentní. Odkaz může být buď absolutní (nové jméno začíná lomítkem), nebo relativní. Symbolický odkaz se liší od zástupce (shortcut), což je normální soubor, který je interpretovaný grafickým uživatelským prostředím.
     5. Na souborovém systému NTFS lze vytvořit ještě tzv. **junction** **point** – speciální odkaz na (lokální) adresář.
9. Konzistence metadat souborových systémů: příčiny vzniku nekonzistencí, metody zachování konzistence, vlastnosti metod, příklady souborových systémů. (blok 11) / (pr 08)
   * Popište příčiny vzniku nekonzistence metadat souborového systému.3
     1. zápis dat do souboru znamená provedení operací na různých místech na médiu
        1. metadata souborového systému
           1. alokace nových bloků, aktualizace seznamu volných bloků
        2. datová část souborového systému
           1. zápis nových dat souboru
        3. metadata souboru
           1. aktualizace přidělených bloků, velikosti, času změny
     2. změny probíhají postupně – vznik nekonzistence
        1. po pádu systému je třeba provést kontrolu
   * Popište princip žurnálování.5
     1. zabraňuje ztrátě konzistence dat při pádu systému
        1. žurnál nezabraňuje ztrátě dat při pádu systému
     2. změny metadat (případně i dat) se zapisují do transakčního logu, což je kruhový buffer
        1. teprve po potvrzení zápisu do žurnálu se provede patřičná změna souborového systému – dvojí zápis = zpomalení
     3. při startu systému po pádu není třeba kontrolovat konzistenci celého souborového systému – zrychlení
        1. prochází se žurnál a zkontroluje se (a případně se opraví) konzistence pouze na místech posledních změn
   * Popište princip metody copy-on-write.4
     1. kopírování bloku při změně
     2. nekonzistence vzniká při změně informací – zápisu
     3. souborový systém nikdy nemodifikuje bloky na místě
        1. při přepisu se modifikuje kopie bloku
        2. stejným způsobem se aktualizují metadata
        3. po dokončení zápisu se atomicky provede zneplatnění původních dat a metadat a potvrdí se platnost nových
        4. souborový systém je tedy vždy konzistentní
        5. data se nezapisují dvakrát (nevýhoda žurnálu odstraněna)
        6. nevýhoda: větší datová fragmentace
     4. souborové systémy: ZFS, BtrFS, NILFS, ReFS
   * Uveďte příklady souborových systémů ve vztahu k metodě zachování konzistence (alespoň dva pro každou metodu, pro žurnálování dvojí způsob).3
     1. souborové systémy bez konzistence po pádu
        1. FAT, exFAT, ext2, HFS (Apple), HPFS (IBM OS/2)
     2. žurnálovací souborové systémy
        1. úplné: ext3, ext4, ReiserFS, UDF (transakční)
        2. pouze metadata: NTFS (MS, transakční), JFS (IBM), XFS (SGI), HFS+ (od Mac OS X 10.2.2), UFS (od Solaris 7, FreeBSD 7.0), VMFS (VMware)
     3. souborové systémy s podporou copy-on-write
        1. ZFS, BtrFS, NILFS, ReFS
10. Typy úložišť, RAID, způsob alokace dat souborů. (pr 08)
    * Popište typy úložišť (DAS, NAS, SAN).3
      1. DAS (Directly Attached Storage)
         1. lokální paměťové úložiště (blokové zařízení)
         2. SCSI, SATA, IDE
      2. NAS (Network Attached Storage)
         1. souborový systém zpřístupněný síťovým protokolem
         2. NFS, CIFS (SMB, Samba), AFS
      3. SAN (Storage Area Network)
         1. blokové zařízení zpřístupněné síťovým protokolem
         2. iSCSI, Fibre Channel
    * Popište vícedisková úložiště (RAID), účel.3
      1. Redundant Array of Independent Disks
         1. zapojení více disků do jednoho virtuálního zařízení
            1. HW: závislé na řadiči, SW: výpočet parity v CPU
         2. výhodou je rychlost a/nebo redundance
    * Krátce charakterizujte úrovně RAID 0, 1 a 5 (způsob zapojení, ukládání dat, vlastnosti).3
      1. RAID-0 – stripping: prokládané ukládání
         1. rychlejší, ale výpadek disku znamená ztrátu všeho
      2. RAID-1 – mirroring: zrcadlené ukládání
         1. rychlejší, redundantní, kapacita jen jednoho disku
      3. RAID-5 – stripping + parity: prokládané s paritou
         1. rychlejší, redundantní, kapacita snížená o jeden disk
    * Uveďte a charakterizujte možné způsoby alokace dat pro soubory.3× 2
      1. souvislá (contiguous allocation)
         1. souboru jsou přiděleny po sobě jdoucí bloky
         2. v metadatech souboru se eviduje:
            1. adresa prvního (počátečního) bloku
            2. velikost souboru (v alokačních blocích)
         3. při alokaci prostoru na médiu dochází k vnější fragmentaci
            1. vznikají díry, které je obtížné využít
            2. soubor nemůže (bez přemístění) růst nad limit volného prostoru za posledním blokem
      2. řetězená (chained allocation)
         + 1. alokují se jednotlivé bloky
           2. alokační tabulka obsahuje zřetězený seznam

každý blok obsahuje odkaz na následující blok

* + - * 1. odstraněna vnější fragmentace

souboru lze přidělit libovolný další blok

* + - * 1. logicky sousedící bloky mohou být na různých místech na médiu – datová fragmentace
        2. př.: souborový systém FAT (DOS / Windows)
    1. indexová (indexed allocation)
       - 1. index obsahuje seznam přidělených bloků – odstraněna vnější fragmentace
         2. index se může odkazovat na blok s indexem

nepřímý index (víceúrovňový)

* + - * 1. bloky mohou mít i různou velikost

redukce vnitřní fragmentace

* + - * 1. př.: unixové souborové systémy

i uzel obsahuje několik přímých odkazů na bloky ‑ a (nepřímé) odkazy na další indexy bloků

1. Vzdálený přístup k OS, telnet, SSH, autentizace, autorizace, zásady tvorby hesla, typy útoků na systém a prevence. (blok 13) / (pr 10)
   * Definujte pojmy autentizace a autorizace.2
     1. **autentizace** (autentifikace) – ověření identity
     2. **autorizace** – ověření oprávnění
   * Uveďte možné metody autentizace (alespoň čtyři).2
     1. **jméno a heslo** – důležité je užívat správná hesla
     2. **hesla na jedno použití** (tokens)
     3. **soukromý klíč** (např. SSH)
     4. **použití smart card nebo podobného HW**
     5. **biometrie** – otisky prstů, oční sítnice apod
   * Popište princip zabezpečení přihlašování pomocí protokolu SSH.3
   * Uveďte riziko přihlašování pomocí protokolu SSH a jak mu předcházet.2 + 2
   * Uveďte zásady tvorby hesla (čtyři).2
   * Popište alespoň čtyři typy útoku na systém.2
     1. trojské koně
        1. nevinně vypadající programy, které mají „funkci“ navíc
     2. login spoofing
        1. imitace přihlašovací obrazovky OS
     3. viry, červi
     4. zadní vrátka (trap door, back door)
        1. vložená metoda přístupu navíc, např. při speciální kombinaci vstupních dat se zaručí další práva
     5. logické (časované) bomby
        1. po určité době nebo za určitých okolností se spouští škodlivý kód
     6. buffer /   / / stack overflow
        1. chyba neošetření maximální délky vstupu
        2. delší vstup pak přepíše další proměnné nebo dokonce kód a způsobí neočekávané chování programu
     7. skenování portů (port scanning)
        1. zjišťování spuštěných služeb a jejich obslužných programů pro následný pokus o využití známé bezpečnostní slabiny
     8. DoS / DDoS – (Distributed) Denial of Service
        1. zahlcení serveru množstvím požadavků, takže se služba zastaví nebo alespoň velmi zpomalí
        2. distribuovaná verze – požadavky přicházejí z mnoha různých (již napadených) uzlů sítě
        3. při podvržení adresy lze dosáhnout blokace části sítě