Udite Part 1

jarek.kligl7

June 2024

1 Intro

XDD INTRO NEVIM NECHAM TO TU AT SE ZASMEJEME
(Vtip to je kdyz se lide smeji)
EEEEEE CKO PLS TRNECKO
NIC VI NECHI

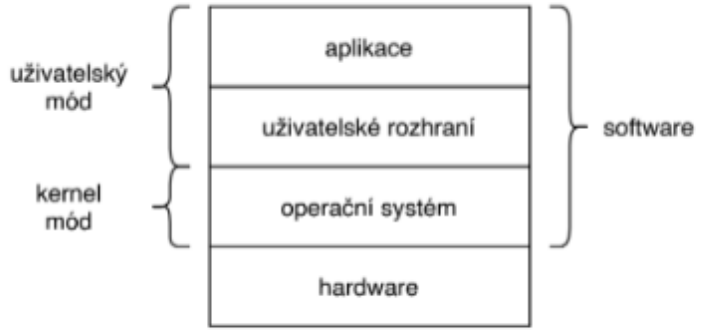
2 Operační systém a jeho architektura

Co je OS?

- Zakladni vybaveni pocitace (Neformalne).
- Rozhrani mezi uzivatelem a Hardwareme.
- Abstrakce nad HW.
- **Odpoved NE EXISTUJE !!!!!!**

Proc se zabivat studiem OS?

- Lepsi Uzivatele (Pointless).
- **Lepsi Programator**.
  Program bude Effektivnejsi ked vim jak OS Funguje a TECKA.



Obrázek: Operační systém a jeho umístění.

Figure 1: s 53 obrazky

Co to je OS? (2)

- Kernel MOD
  - Jaderny Mod
    Zni to dost Nuklearne ale
    takze lepsi jenom kernel
  - Specialni privilivige mod
  - proces muze delat cokoliv co je umozneno
- Uzivatelsky mod
  - Restrikce zabranuji vykonovani nekretych nebezpecnych veci

Os je na urovni Kernel Modu

2 zakladni pohledy

- Abstrakce nad HW
  - HW je pod OS.
  - HW je slozita vec.
  - lisi se vyrobce od vyrobce.
  - nutna znalost kazdeho jednotliveho w do hloubky pro napsani uplne ez veci.
  - nacteni souboru by byla manipulace s hw (nelze).
  - **muzeme pracovat s rozhranim abstraktni vrstvy**.
  - **to co je nad os (uzivatelske interface)**
- sprava zdroju
  - spravedliva sprava zdroju (sounds like comunism tbh).
  - OS JE STAT A procesy jsou lide ve statu xddd.
  - pirsutp k procesu, pameti , a I/O zarizenim.
  - **pod os ten konktretni HW.**

tak a co umoznuje (DELA) OS?

- Interaktivni prace s PC
  - **uzivatelske rohrani** UI (USER INTERFACE)
  - umoznuje uzivately HW omezene pouzivat
  - **OVLADACE**
    Umoznuji zprostredkovat funkcionalitu HW
- sprava software
  - spustit program
  - prostredi pro beh
- sprava HW
  - Paralizace (soubeh)
  - spravedlive rozdelovani zdroju
    procesor, pamet, ...
  - hlavni pridelovac
  - duraz na bezpecnost a **effektivitu**
- organizace a zprava dat
  - volne misto
    hlavni pamet , druhotna
  - reprezentace souboru
- **poskutije sluzby programu**
  - poskytuje sluzby programum
  - aplikace jsou zakaznici a os jim poskituje sluzby
  - **API** - Aplication Interface
  - skrz sys sluzby
  - os ma sluzby a aplikace je pouzivaji
  - cteni souboru (abstrakce nad HW)
  - (sys calls v assembly treba)
  - os roztoci disk , najde sektor kde se nachazi atd
  - diky abstrakci , diky api si zazadame o nacteni souboru
    a ten je nam vracen

Obrázek: Architektura OS.

Figure 2: ???

architektura OS uz ted je tezke to vymezit , uzivatelske rohrani je/neni. v Linuxu neni atd.. , muze jich byt i vice trnecka tohle vsecko povazuje za OS.

Jadro (Kernel) Nejdulezitejsi cast OS.
To jedine neni sporne , kazdy Os nutne obsahuje Jadro.
Kernel mod. Poskytuje nejzakladnejsi abstrakci nad HW, Sprava zdroju , izolace, bezpecnost.
Sys Calls

Druhy Jader

- Monoliticke
  - Pros++
  - Vsecko je tak nejak v jednom
  - jeden jediny program
  - Jadro + sys bookshelfs xddd + sys sluzby
  - nemusime resit prepinani
  - CONS (CAR. CDR)
  - nizka uroven Abstrakce

  - Strasne Velke programy
  - udrzet tak velky program narocne
  - -/-
  - **Vrstvena Architekruta**
    V tom jadre jsou nejake Fce ktere se navzajem volaji
    Jsou tam Abstrakcni bariery , minimalizace komunika
    mezi vrstvama
    zajisteni ze casti spolu logicky souvisi
    Jaky vetsi software toto vyzaduje
  - **BSD, LINUX**

- mikro
  - Jenom nejnutnejsi veci
  - meziprocesorova komunikace, planovani behu procesoru
  - drobna srpava pameti (ne nutne)
  - **Snadna Udrzitelnost (velky benefit)**
    v prumeru na 1000 radku 2-10 chyb (nenavidime programatory)
  - ty monolity maji vic jak 5 mil radku coz je hodne moc chyb
  - (Linux ma 27 milionu to je kurva moc chyb jeee beckdory uwu to se deje xddddddddd kurva porad a porad)
  - poskytuji vetsi Abstrakci
  - nevyhody –
  - **pomalejsi nez monoliticke**
  - proto se nepouzivaji moc
  - **MIMIX**

- Hibridni
  - Kombinace mikro a mono
  - zlata stredni cesta
  - **micro jadro + sirsi okoli**
  - Windows NT, Mac OS
  - Windows NT
    je obcas povazovan za mikro ale to narazi na tom jaka je definice
    vidime tu nejednoznacnost lisi se to literatura od literatury
  - Windows NT - je mysleno veskere Windows i 11 10 vsecky!!!

- Exo
  - **Akademicke Prostredi**
  - **mala abstrakce nad HW**

  - porad je tam mezi vrstva ale
  - odpovidaji definici firmware (ale tohle vzit jen s nadsazkou)
  - rychlesji pristup k HW, Lepsi vyuziti zdroju
  - omezenejsi na urcite veci
- Uni
  - **Beh pouze jedne aplikace**
  - abstrakce ale ne sprava zdroju
  - zdroje jsou prideleny, nebo odejmuty pouze jedne aplikaci
  - v chytre elektronice treba

Systemove Knihovny Monhem hezci rozhrani - abstrakce nad Kernelem
API - vola to funkce jadra
Naprikald Libc , Msvcrt obsahuje fce Printf() v C.
Knihovny jsou nutnost pro funkcy programu, (chci pouzit print musim mit linklou knihovnu)

Moznosti jak poskytnout knihovnu programu

- Staticke
  - K programu se **prilinkuji** ty knihovny
  - **redundance** - Redundance (z lat. redundare, přetékat, přebývat) znamená informační nebo funkční nadbytek, například větší množství informace, prvků nebo zařízení, než je nezbytné. (Zrdoj wikipedie)
- Dynamicke (sdilene)
  - knihovna je v Operacni pameti
  - program vy kde se nachazi ta knihovna
  - v ruznych os zpusobuji ruzne problemy
  - zavislosti na ruznych verzy knihovny
  - ve Windows jsou oznaconany jako **DLL** (Dynamic-link library)
  - DLL Hell - Zavyslost na urcite knihovne, mam jinou verzy nefuguje to

Posix (Portable Operating System Interface) Usnadneni pro programatora
Standart, specifikace rozhrani pro OS.
Systemove knihovny
Muzu napsat Libovolny Program ,a kdyz pouzivam knihovny ze tohoto standartu tak budou fungovat na vsech OS s podporou Posix.
Unix-like Systemy (Mac OS , LINUX BDS Solaris ...)
Do windows ta podpora jde nejak pridat ale neni implicitni (Microsoft XD MOMENT)
Pica kdyz toto pisu tak se citim jak outrata kurva.

Systemove Sluzby Deamon (”Démon”)

- Klient - Server Architektura
- Regulerni programy
  nemaji treba ani UI
  Na pozadi vykonavaji funkcionalitu konkretni sluzby.
- Webovy server, Vzdalena plocha ...
- Udrzba systemu , zprava pameti ...
  Opakujici se ulohy
- Operace vyzadane Jadrem
  Nekdy co je se nemusi delat v Kernel Modu
  tak jadro pozada konkretni sluzbu aby to zaridila

Uzivatelske Rozhrani OS

- CLI (Command Line Interface) - Prikazovy radek
  Velka moznost Automatizace
  Muze udelat vsechno na te urovni kde se nachazy
  Nemuze treba nativne pristoupit k HW
  Stale abstrakce!!!
- GUI (Graphical User Interface)- Graficke rozhrani]
  Z pohledu Uzivatele prijatelne
  ,ale nedostacujici pro programatoy nebo spravce
  Zakladni stavebni prvky pro tovrbu okenich aplikaci

Poskytovano Systemovima Knihovnama a sluzbama.
Zakladni stavebni kamen pro UI

Systemove Nastroje Hodne **sporne jestli to je soucast OS**.
Ale kdyz si OS naistalujete tak se tam nachazeji
Male programky ktere umoznuji **OS ovladat, pripadne nastavovat**
Pro windows - Editor registru
Linux - Textovy editor
fylozofie Linuxu vsechno je soubor staci txt editor aby jsme to pozmenily. LS (Linux), Dir (Windows) - vypise obsah adresare.

Typy OS Podle pozadavku
Kazdy Os je vyroben s ucelem nebo s fylozofii.
NELZE TO BRAT DOGMATICKY!!!!!!!
Ne linux neni jen pro programatory :SKULL: TIKTOK PICO

- univerzalni
  - Bezne Pocitace (Desktop)
  - Tak jak je to popsane doted
- embedded
  - Jednoucelove zarizeni
  - Chytra Elektronika (Televize)
  - maly vykon - rizeni omezenych zdroju
- real-time
  - kladen pozadavek na odezvu v nejakem case (Klidne i nekolik Sekund)
  - **Hard**
    striknti termin jinak je zle!!
  - **Soft**
    Obcas je to mozne obejit,
    nic moc se nedeje kdyz je to trochu obcas presvyhnuto
- distribuovane
  - Os bezi na mnoha pocitacich
  - sdili mezi sebou prostredky
  - sbernici nahradim pocitacovou siti (zjednodusene hodne)!!
  - mnohem vypocetni vykon
  - Cluster - spojeni vyce pocitacu

Vykonavani programu .
Program - sada intrukci zapsana v programovacim jazyce ulozena v pameti
Instrukce Procesoru

- Jednoduche operace co cpu dokaze vykonavat
- treba pricti 1 k pameti registeru
- omezene operace
- instrukcni sada procesuru
  ciselne oklicovane instrukce
  000000001 000011100011 ...
- Assembly language - jazyk symbolickych adres
- misto 1 a 0 mame mov atd ... (abstrakce nad 01)
- jde se v tom nomrlane programovat
- lowlevel

program v programovacim jazyce

- instrukce pro procesor
- python, c ... je abstrakce nad Assembly
- Interptetovany
  za behu se preklada vetsinou mezi krok bytecode
- Kompilovany
  Preklad do intrukci pri kompilaci (dopredu se to cele prelozi)

Registry

- PC - Program Counter
  uchovava adresu nasledujici instrukce (cislo jenz reprezentuje adressu)
- IR - Instrukcni Registr
  Ciselny kod reprezentujici aktualni zpracovavanou instrukci
- MAR memory addres register
  misto v pameti od kud se bude cist nebo kam se bude zapisovat
- MBR memory buffer register
  dana hodnota ktera byla prectena nebo bude se zapisovat
- Analogicky I/0
  AR - Misto od kud cist
  BR - To co jsem precetl

zpracovani instrukce (procesoru)

1. V Pc mame adresu intstrukce MOV do MAR.
   MAR ukazuje na to same misto jak PC
2. provedu nacteni z mista urceneho MAR
3. Nactena pamet je MOV do MBR
4. instrukce MBR MOV do IR
   (mam v instrukcnim registru ciselnou reprezentaci ten instrukce co musim vykonat)
5. ADD PC 1
6. dekodovani instrukce (jeji identifikace)
7. Vykonani instrukce

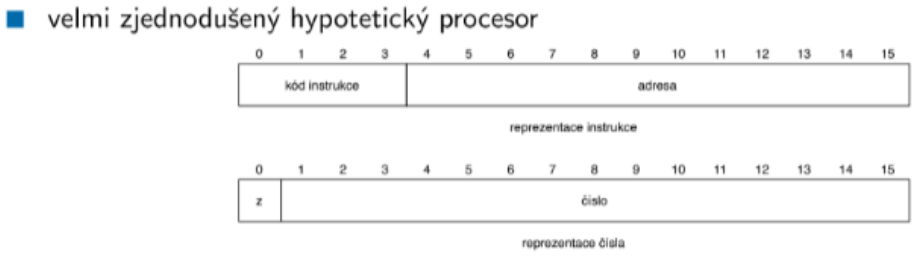


Figure 3: Rengar was here

Priklad Vykonavani programu

- Ma 16 bitu
- adresni prostor ma 12 bitu
  $2^{12} = 4096$
  3 hexadecimalni cisla
- registry
  1. PC - & nasledujici instrukce
  2. IR - adresa aktualni instrukce
  3. AC - registr pro ulozeni dat
  4. MAR , MBR - Vynechame
- instrukce procesoru (4 byty = 1 hexadecimanli cislo)
  1. 0001 nacteni z pameti AC (1)
  2. 0010 ulozeni z AC do pameti (2)
  3. 0101 pricteni hodnoty z pameti k hodnote v AC (5)

Obrazky jak to probiha naslajdech tady neni nic moc navic nebo alt zdroj:
https://www.youtube.com/watch?v=jFDMZpkUWCw

2.1 Preruseni

Muze bezet vice programu nez jeden, musi bezet i OS naraz
Preruseni postupneho zpracovani instrukci
Bezna zalezitost (I/O, Casovac, chyby v programu & HW)
Reknu disku nacti neco on je pomaly chvili mu to trva,
tak pokracujeme nekde jinde mezitim.
Slouzi k predani rizeni obsluze preruseni (soucast OS)
Proces Preruseni (Symple)

1. Je dokoncena aktualni instrukce
   PRIMITIVA NELZE PRERUSIT V PULCE

2. ulozeni aktualniho stavu
    - Stav Registru
    - Program Status word
3. zmena stavu registru (dle obsluhy preruseni)
   zmenim PC a hodnoty registru...
   Obsluha preruseni - nejaka cast co ridi co se ma delat a jak pokracovat...
4. pokracovani ve vykonavani instrukci
   nactou se data s Program status word
   aktualizuji se registry a pokracuje se dal

Preruseni muze nastat i pri vykonavani obsluhy preruseni..
prerusime program na miste A a pokracujeme na miste B pak ho znovu prerusime a dostaneme se na misto C a pak treba znovu na D a pak zase na A atd...

Prvních 32 přerušení je vyhrazeno pro výjimky, přerušení generovaná přímo procesorem. Ne všechny z nich se využívají, na 8086 jich existovalo jen prvních sedm, později (počínaje procesorem i386) přibyly další:

1. dělení nulou
2. krokovací přerušení – na i386 rozšířeno na víceúčelové ladicí přerušení
3. NMI – externí nemaskovatelné přerušení
4. breakpoint – vyvolaný již zmíněnou instrukcí `int03`
5. přetečení – je vyvolané instrukcí `into`, pokud je v příznacích zaznamenáno přetečení
6. překročení mezí – volané instrukcí `bound`, pokud byly překročeny meze pole (přidáno v 80186)
7. chybná instrukce
8. nedostupnost koprocesoru – nepřítomnost nebo nepřepnutá úloha – dříve bylo hlášeno externím nemaskovatelným přerušením
9. dvojitá chyba – vyvoláno, pokud dojde k výjimce při vyvolávání výjimky
10. překročení limitu segmentu koprocesorem
11. chybný TSS – chyba při pokusu o změnu úlohy
12. nepřístupný segment
13. překročení limitu zásobníku
14. obecná chyba ochrany – většina chyb související se segmentací kromě těch obsloužených ostatními výjimkami
15. výpadek stránky – pokus o přístup ke stránce, která je namapována, avšak není v operační paměti přítomna
16. (nepoužito)
17. chyba koprocesoru
18. chyba zarovnání – pokus o přístup k adrese nedělitelné odpovídající mocninou dvou, pokud je zapnuta kontrola zarovnání

Figure 4: caption i quess

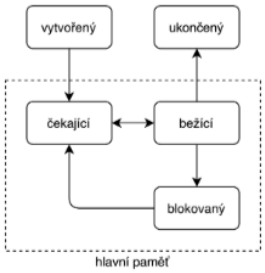
zdroj: wikipedie - preruseni

3 Procesy a vlákna

Proces .
soudobe OS imituji jenom ze 2 programy bezi zaroven
program nema vedet ze je zaroven spusten s jinym programem

- **abstrakce nad bezicim programem**
- zakladni **jednotka pro praci s programy**
  Kdyz spustime program spustime process
- pristup ke zdrojum
- **izolace samotnych procesu**
  duvod bezpecnost
- proces urdrzuje i jine informace nez samotny program
  - Os nutne nepotrebuje vedet co je to za program
  - procesu se prideluji to zdroje, pamet ...
  -
- OS udrzuje informace o procesoru tzv. **kontext**
  informace pro prepinani procesu , planovani , **pridelovani zdroju**

Proces: Životní cyklus



22 / 85

Figure 5: XDDD

- **Vytvoreny** - os provedl ukony pro zahajeni procesu
- **Ukonceny** - ten co uz na dale nebezi
  POUZE BEZICI PROGRAM MUZE BYT UKONECNY
  kdybychom chteli ukoncit cekajici - technycka komplikace
- Stavy v hlavni pameti:
  - Cekajici - okamzik kdyz nebezi
  - Bezici - Procesor aktualne vykonava instrukce konktretniho procesu
  - Blokovany - I/0 pokud bezici proces ceka na pomale I/O
    akce je podminena , **dostane se do cekajiciho** az je akce dokoncena

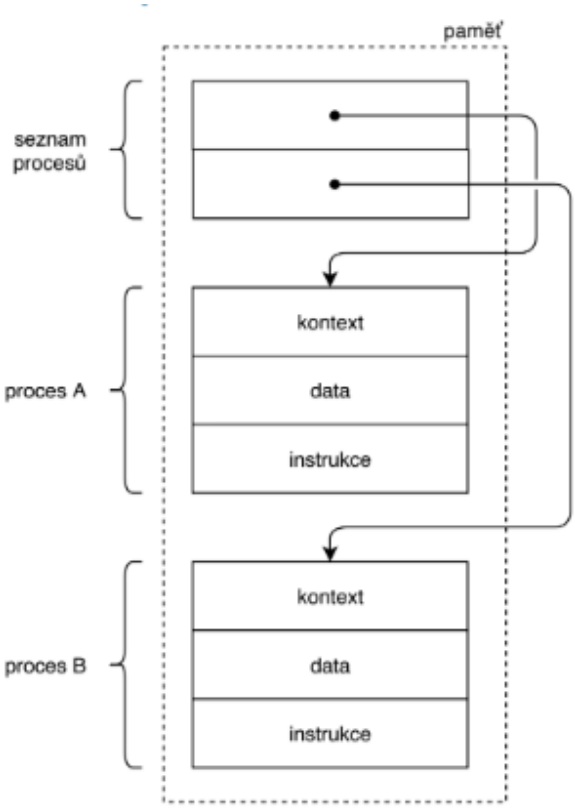


Figure 6: Implementace

Potrebujeme rezijni informaci - kontext
A pamet
Mamem pamet a 2 procesy
proces A je reprezentovan kontextem , instrukcema a datama,
data - prostor kde muze realizovat vypocet
Os pri supsteny prideli pamet nacte program do hlavni pameti a pak uz se to vykonava
Nutny SEZNAM PROCESU

V jeden Okamzik **muze na pc s 1 CPU bezet pouze jeden proces** konkurentniho behu je docileno neustalym prepInanIm vice procesu (**time-sharing**)

- konkurentni beh
  neustale prepinani procesu
- paralerni beh
  soubeh sa pomoci vice jader treba...

Bezi vice procesu -> **potreba rizeni**
ne vsihni muzou v dany okamzik mit pristup k procesoru pameti

Komunikace meziprocesorova komunikace (IPC)
poskytuje to jadro a nadstavba sys knihoven

izolace ale pres OS muze probyhat komunikace

Treba pisi program ktery rozdelim na vice casti
muzu to rozdelit a komunikovat pres OS
Proces komunikuje s OS

Prepinani procesu
context switch
procesy ulozeny v process table
zaznam v process table = **process control block (PCB)** -
jeden radek proc.table
PCB uchovava uplny stav programnu

- registry
- pamet
- otevrene soubory
- ...

Proces prepnuti:

1. save stavu procesu do PCB
2. aktualizace PCB (napr. zmena stavu)
   jestli je bezisi , cekajici , blokujici ...
3. save PCB (do prislusne fronty dle stavu)
4. vyber noveho procesu

5. aktualizace PCB (napr. zmena stavu cekajici na bezici)
6. nacteni dat z PCB
7. aktualizace dle PCB

Os ma oddelenou tabulku pro bezici procesy , cekajici , blokujici ...
Typy pristupu:

- kooperativni
  ponechano procesu
  sam si rekne uz chci skoncit
  Procesy dostanou tolik casu kolik potrebuji
  proces potrebuje slusnost
  napsat program ktery by nikdy nevratil rizeni neni tezky
  nepouziva se to

- preemptivni
  zajistuje OS

Pozadavek:
co nejvice prace vs interaktivita (prituchudne)
nejefektivnejsi je nepoustet uzivatele ke stolu
coz necheme (z vyjimkama)
interaktivita snizuje vykon
procesy je treba organizovat
Planovani behu - vyber procesu, ktere "pobezi"
zajistuje planovac, soucast OS

prepnuti procesu je rezie - zabira to cas

proces ma CPU pridelen na dane casove kvantum (prevadi se intrukce daneho programu)
preprnuti procesu - rezie
kriticka a kompilovana zalezitost
typy planovani:

- dle casu
  - kratkodobe - cekajici a bezici stav prepinani
  - strednedobe - ty co cekaji na I/O
  - dlouhodobe - vubec bude spusten?
    os povoli sputeni ma dost zdroju?
- dle typu ulohy

  - davkove zpracovani - chci aby se spustil a bezel co nejrychleji
    vyuziti na super pc nebo pri uceni treba AI ...
  - interaktivni - potreba pracovat s OS
  - realny cas - potreba zajisti konktertni odezvi

na beznych OS kompromis mezi davkovym a interaktivnim
Uzivatelske rohrani je interaktivni
MS-Word blizko davkocemu zpracovani - bezi nejakou dobu a chci ho hodne vyuzivat

dva typy procesu:

1. **CPU-bound proces**
   vykonavany program obsahuje minimum I/O operaci
2. **I/O-bound proces**
   Vykonavany program obsahuje hodne I/O operaci
   cekani na I/0

Strategie planovani

- Cyklicka obsluha (round robin)
  - procesy se pravidelne stridaji
  - nemusi bezet stejne dlouho
    treba kdyz nejaky se dodela atd
  - nevhodne pro interaktivni praci
  - **jde se porad dokola**
- spravedlive prideleni casu
  - procesy bezi stejne dlouho
  - procesy bezi t/n pri t casu a n procesech
  - nevhodne pro interaktivni praci
  - se to vubec nepouziva
- priority fronta, procesu je pridelena priorita
  - priorita je vetsinou cislo
  - interaktivni prace
  - procesum s visi prioritou bezi casteji nez s nizi

  - nespravedlive deleni prostredku -> **vyhladoveni**
    chci zrovna ted pracovat s prioritou 1 a dostanu se k nemu az moc moc pozde

- mnoho dlasich

cyklicka obsluha a spravedlice prideleni casu je **nevhodne pro interaktivni praci** - protoze mame n procesu cheme pracovat hodne s $procesem_1$ pri vysokem n musi se vzdy projit vsech n procesy aby jsme zas pracovali s $procesem_1$
dnes se bezne pouziva **kombinace cyklicke obsluhy a prioritni fronty**
treba pokud okno ma fokus tak se tomu zvetsi priorita
uzivatesky nastavitelne taky
ridi si to jak os tak procesy samotne
aby to bylo dobry tak to je komplikovany

jak je planovan beh OS?
OS = program
planovani behu

1. Funkce OS vykonavany mimo procesy (dnes nepouzivane)
   mam procesy $p_1, p_2 ... p_n$ a pak OS jako vrstva pod procesy (ridi vykonavani) OS NENI PROCES!!!!!
2. **funkce OS vykonovany uvnitr uzivatelskych procesu (bezne)**
   - vsecky procesy maji sdilenou cast OS
   - casti OS sdilene mezi vsemi procesy
   - kernel zasobnik pro volani v kernel modu
     tam se jadro jak se rika muze vyradit
   - **prepina se pouze mod procesoru (mensi rezie)**
3. funkce OS vykonavany samostatne procesy
   - vhodne v pripade vice procesoru
   - funkce OS jsou samostane procesy
   - OS jenom planuje a planuje i svuj beh
   - Nevyhoda:
     prepinani procesu je rezie
     (musime nacist konext s tabulky ...)
   - **distribuobane systemy**

Vlakno
abstrakce nad vykonavanym kodem (obvykle jeho casi)
vlakno je abstrakce nad casti instrukci
vice vlaken v ramci procesu
vlakna sdili pamet (v ramci procesu)
Zadna izolace (v ramci procesu)
vyuziti:

- asinc zpracovani
- neblokujici I/O
- GUI

Sprava Vlaken
prace s procesem (vytvoreni, prepinani) ma (velkou) rezii
prace s vlakny je vyrazne jednodusi

- **nedochazi k prepinani procesu**
- rychlejsi

realizace

- sys knihovna (user-level vlakna)
  prepnuti nevyzaduje kernel mod, sprava v ramci procesu
  svet ve svete "proces je os pro vlakna v tom procesu"
  sys cally ale zastavi vsecky vlakna naraz

- jadro (kernel-lvl vlakna)
  prepnuti vyzaduje kernel mod, vlakna lze planovat jako procesy
- kombinace

Vlakna	***Procesy***	***:***
1	1	UNIX
m	1	Windows NT, Linux, MacOS
1	m	Vlakno muze by presuno mezi procesy (distr. sys. , cloud)
m	n	

m:n teoreticka zalezitost existovalo to (nepotrebne pro nas)

4 Synchronizace procesů a vláken

procesy a vlakna mezi sebou soutezi (race)
procesy jsou izolovane -> komunika pres zpravy
vlakna -> komunikace pres sdilenou pamet -
jednodusi komunikace ale zpusobuje problemy (treba race condicion)
k prepnuti vlaken dochazi "samo" oni o tom nevjedi ze jsou na rade ci nejsou
nebo nedejboze jsou spusteni naraz

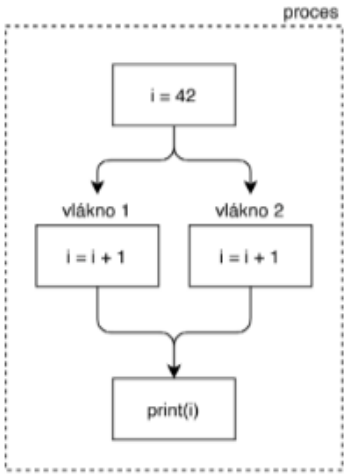


Figure 7: au chyba aaaa

Chyba soubehu (race condition) - nekonzistentni vysledek v dusledku chybneho nespravneho behu procesu/vlaken

- i = i + 1 **neni atomicka!**!!!
  atomicka = nedelitelna
- ve skutecnosti je to nekolik operaci:
  1. move i do registru A
  2. ADD A 1
  3. move A do registru i
- kdykoliv (po dokonceni intrukce procesoru) muze dojit k prepnuti vlakna (ci procesu)

```
(defun xd (i)
  (setf i (+ i 1)))


CL-USER 11 : 1 > (disassemble 'xd)
410001647C:
      0:      49396275          cmpq  [r10+75], rsp
      4:      774E              ja    L3
      6:      4883F901          cmpq  rcx, 1
     10:      7548              jne   L3
     12:      4155              push  r13
     14:      4153              push  r11
     16:      4157              push  r15
     18:      55                push  rbp
     19:      4889E5            moveq rbp, rsp
     22:      4989DF            moveq r15, rbx
     25:      4989FD            moveq r13, rdi
     28:      41F6C507          testb r13b, 7
     32:      751C              jne   L2
     34:      4D89EB            moveq r11, r13
     37:      4983C308          addq  r11, 8
     41:      7013              jo    L2
L1:  43:      4C89DF            moveq rdi, r11
     46:      B901000000        move  ecx, 1
     51:      4889EC            moveq rsp, rbp
     54:      5D                pop   rbp
     55:      415F              pop   r15
     57:      415B              pop   r11
     59:      415D              pop   r13
     61:      C3                ret
L2:  62:      4C89EF            moveq rdi, r13
     65:      BE08000000        move  esi, 8
     70:      498B9E070E0000    moveq rbx, [r14+E07]
     77:      FFD3              call  rbx
     79:      4989FB            moveq r11, rdi
     82:      EBD7              jmp   L1
L3:  84:      41FF662F          jmp   [r14+2F]
     88:      90                nop
     89:      90                nop
     90:      90                nop
     91:      90                nop
```

Figure 8: Neni to atomicka instrukce je tam vice move instrukci atd...

vlákno 1	vlákno 2	i
načti		42
přičti		42
zapiš		43
	načti	43
	přičti	43
	zapiš	44

vlákno 1	vlákno 2	i
načti		42
	načti	42
přičti		42
	přičti	42
zapiš		43
	zapiš	43

Figure 9: Chyba soubehu

Slavne programatorske chyby

- Y2K (pad systemu po celem svete, chybny vypocet prestupneho roku)
- HearthBleed (chyba v zabezpeceni, opomenuti)
- Mariner 1 (zniceni sondy, znamenko)
  r s pluhem misto -r
  cena jednoho znamenka 18 milionu dolaru
- Mars Climare Orbiter (zniceni sondy, ruzne jednotky)
  imagine mit jine jednotky lol
- Ariene 5 (zniceni sondy, pricteni 16bitoveho cisla)
  skoda yes milionu
- Therac-25 (smrt 6 pacientu na radiaci, chyba soubehu) Low level learning video
- mnoho dalsich:
  - rakety Patriot
  - burza
  - letiste Heathrow (zavazadla se nedorucila dal)
  - Youtube (pocet zhlednuti pteteceni (gangamstyle song))
  - kalkulacka Windows (vysledek odmocniny)
  - ...

Odladit chybu soubehu je velice tezke odlatit (pokazde to bezi jinak)

Synchronizace
zabraneni Race condition
urcena pomoci behu procesu a vlaken

- pomoci zprav (typicky procesy)
  muzou i vlakna ale retardovane
- sdilenych promenych (typicky vlaken)
- sys calls
  v pripade kernel vlaken
- aktivniho a pasivniho cekani
  program pernamentne bezi a jenom ceka nez se neco stane

Zakladni nastroje:

- atomicke operace (HW podpora, ale staci pouze atomicke ulozeni)
  operace ktera nebude prerusena
- sync. primitiva
  ridi a signalizuji pristup k **kriticke sekci**
  - zamek (mutex)
    zamek na dverich mistnost je kriticka sekce , zamcete vyblbnete se, nikdo za vama nemuze a pak odejdete a pred tim odemcete a klic predate
  - semafor
    sinchronizuje vice veci
  - monitor
  - bariery

  implementovano pomoci atomickych operaci
- **kriticka sekce** = misto pristupu ke sdilenym (kritickym) prostredku
  pozadavky na kritickou sekci:
  kriticka sekce necheme aby se tu prerusovo!!!! vstup do kriticke sekce se oznacuje jako vstupni protokol
  vystup do kriticke sekce se oznacuje jako vystupni protokol
  1. vzajemne vylouceni (pouze jeden muze vztoupit)
  2. absence zbytecneho cekani
  3. zaruceny vzstup
  4. zaruceny vystup

```
sdilena_promenna = nil

loop forever # producent
  data = produce()
  wait sdilena_promenna == nil
  sdilena_promenna = data # kritická sekce

loop forever # konzument
  wait sdilena_promenna != nil
  data = sdilena_promenna # kritická sekce
  sdilena_promenna = nil # kritická sekce
  consume(data)
```

Figure 10: Enter Caption

Priklad
jednoducha verze problemu: producent-konzument

1. producent zapisuje hodnotu do sdilene promene
2. konzument cte hodnotu ze zdilene promene
3. producent nesmi prepsat neprectenou hodnotu
4. konzument nesmi opakovane cist stejnou hodnotu

Nil - nebo jakakoliv pomocna hodnota ktera nemuze byt
predpokladame atomicitu instrukce zapsani
Wait - aktivni cekani

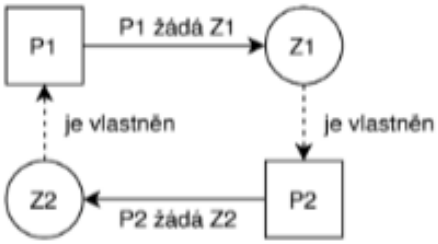


Figure 11: Mtvy zamek

DeadLock
chyba synchronizace
krizovatka bez svetelnejch znacek a prijedou tam 4 auta vsichni davaji prednost s prava a nikdo nemuze jet
cekani na vylucne vlastnene zdroje = uvaznuti
podminky vzniku (Coffmanovy podminky):

1. vzajemne vylouceni (zdroj vlastni pouze jeden)
   trivialne splneno protoze je to podminka na kritickou sikce
2. vlastnik zdroje muze zadat o dalsi zdroje
3. absence **preempace**
   zdroj musi byt vracen, nelze jej odebrat
4. cyklicke cekani

Reseni:

- ignorovani (bezne OS)
  vsimne si toho uzivatel a tu aplikaci sestreli
  zamrznuti aplikace
- detekce a nasledne zootaveni
  detekce je zalozena na analize grafu najdeme cyklus treba funcki circle-p moc se to nedela
- zamezeni vzniku
  atakuji nejakou coffmanovu podminku
- vyhybani se vzniku
  nespustim proces ktery by mohl to zapricinit

Dalsi problemy:

- Vyhladoveni (starvation)

  - procesu/vlaknu neni poskytnut pristup do kriticke sekce
  - neposkytnutim zdroju
- livelock
  - procesy/vlakna bezi
  - ale nevykonavaji zadnou praci
  - vsichni cekaji na kriticke sekci ale nikdo do ni nakonec nevztoupi
  - pojebe se treba nutnost zaruceneho vztupu

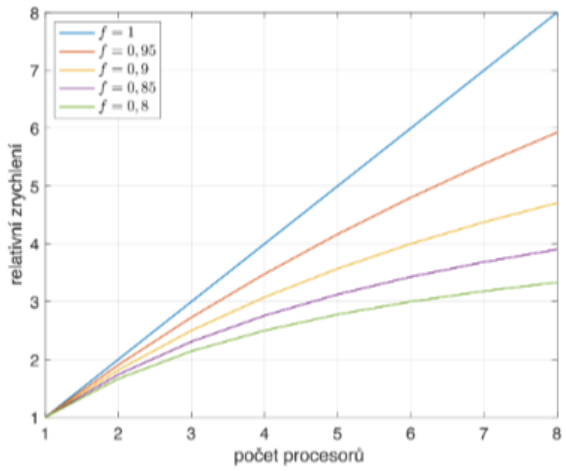


Figure 12: graf zoberazeni

Zrychleni pomoci paralelismu Amdahluv zakon (max predpokladane zlepseni systemu)

$$zrychleni = \frac{cas \quad behu \quad na \quad 1 \quad procesoru}{cas \quad pararel. \quad behu \quad na \quad n \quad procesorech} = \frac{1}{(1-f) + \frac{f}{n}}$$

- **f** libovolne paralelizovatelny kod (1 je 100 procent)
- **(1 - f)** cast kodu, kterou nelze vykonavat paralelne
  predstavte si to jako komplementarni jev i quess

- nerealisticky optimisticke

Realne mnohem horsi (pro f = 0.95 se limitne blizi k 20)
rezie pri prepinani -> pokles zrychleni s narustajicim poctem procesoru
databazove systemy se blizi k tomuto idealu

Amdahluv zakon - zrychleni pri konstantim mnozstvim dat
zrychleni systemu je omezene neparalelizovatelnou casti

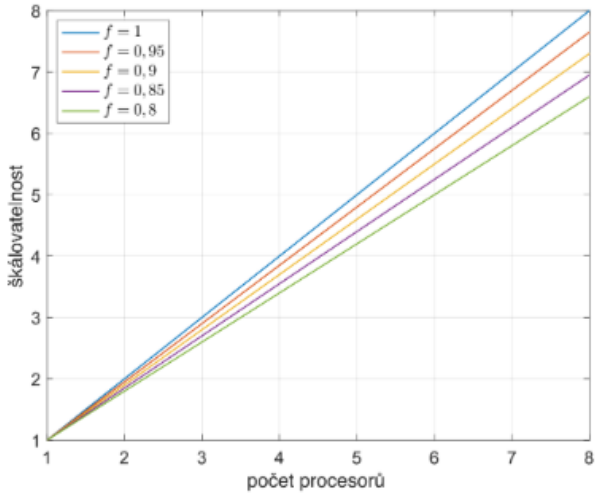


Figure 13: skalovatelnost

Gustafson-Barisisuv zakon

- kdyz mam vysoky vykon pouziji ho

  ->

  vice procesoru = vice zpracovanych dat na stejny cas, presnejsi vysledek...

- **skalovatelnost** = pri rustu systemu jsou zachovany jeho vlastnosti

$$zrychleni = \frac{(1-t) + n \cdot t}{(1-t) - t} = n + (1-n) \cdot (1-t)$$

- **t** doba behu (procento) libovolne paralelizovatelneho kodu
- **(1 - t)** complementarni jev (neparalelizovatlny kod)

"sekvencni cast (1 - t)" typycky neroste s velikosti problemu (inicializace, agregace vysledku)

Amdahluv vs Gustafson-Barsisuv zakon

- dva odlisne pohledy ale prekpavive ekvivalentni
- Amdahl - limitovane zrychleni (cas) daneho problemu
- Gustafson - dany cas, neomezene zrychleni
- neni to divne
  - auto jede z A do B, vzdalene 200 km
  - za hodinu 100 km
  - Amdahl, i kdyz zbytek cesty posjede libovolnou rychlosti, prumerna rychlost nebude vetsi nez 200 km/h
  - Gustafson: pokud auto pojede dostatecne dlouho dosathne libovolne prumerne rychlosto
- **oba zakony ukazuji pohled na skalovatelnost systemu**
  s rostoucimi prostredky roste vykon

Udalosti rizeni

WARNING TOTO JSU JEN PREPSANE SLIDES toto v prednasce 4 roky zpatky nebylo :(vlakno prestavuje pro OS mensi rezii nez proces (presto je zde rezie
prostredky jsou omezene -> pocet vlaken je omzeney
C10K problem -
v pripade I/O uloh lze efektivne resit bez pouziti vlaken
rizeni na urovni funkci:

- korutiny
- funkce maji vice vstupnich bodu
- async. programovani
- treba planovac na urovni procesu

naprikald web. server

konec warningu

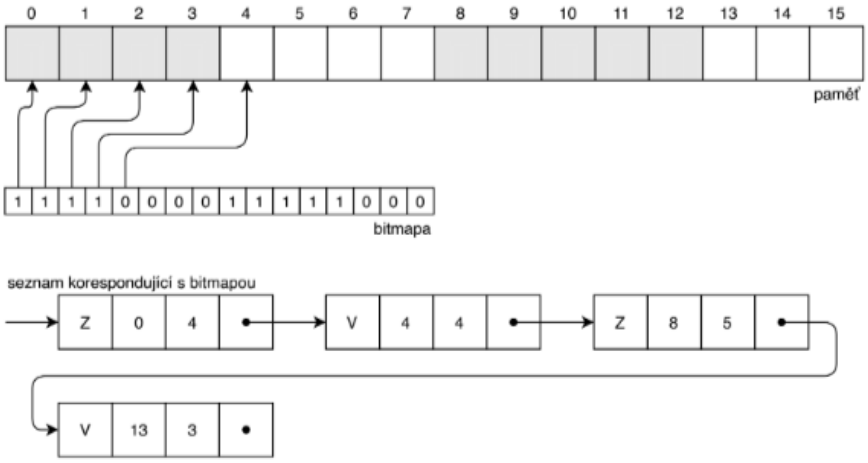


Figure 14: evidence

5 Fragmentace pameti

Sprava operacni pameti

rika se pameti je vzdycky dost kdyztak se da koupit nova ale neni to pravda hlavne u ram
nedostatecna velikost RAM -> nutstnost spravy
proces zada o pamet OS mu ji prideli

- jednoulohove OS
  fajn jedna pamet pridelim mu ji vsecku
- vyceulohove -> nutsnost izolace pameti procesoru
  jinak by mohlo dojit k vaznemu poskozeni
  bezne maji v kontextu ulozeny informaci kde jsou alokovant
  procesor zajisti aby proces nemohl sahat nekam kam nema
  treba v C to vyhodi erro jak zname s zpc kdyby to tak nebylo tak muzeme napsat program co vyjebe os - nechtene
  ale existuje rada technyk jak to obejit

pouze souvysla cast pameti
jinak chaos ciste s praktyckych duvodu a pak dalsich
evidence volneho mista = rezie

Evidence volneho mista

1. spojovy seznam

2. pochopitelnejsi
3. mame strukturu co uchovava zabrane misto a volne mist
4. toto misto je zabrane zacina na adrese x a zabira y mista
5. a takto se strida zabrane a volne
    -
6. bitmapa
    - Pouzivanejsi
    - efektivnejsi a uspornejsi
    - mame pamet ty sedy policka jsou zabrany
    - sekvence bitu kdy 0 znamena volny a 1 zabrany
    - v nehorsim pripade to zabere par megabytu
    - je to pole
    - je to pouzivany i na pevnym disku

pridelovani **souvyslich** bloku pameti
treba zvolit velikost bloku

1. stejna velikost bloku
   nevyuzite misto v bloku -> vnitrni fragmentace

2. promenliva velikost bloku
   po uvolneni nedostatecne misto pro vetsi bloky -> vnejsi fragmentace

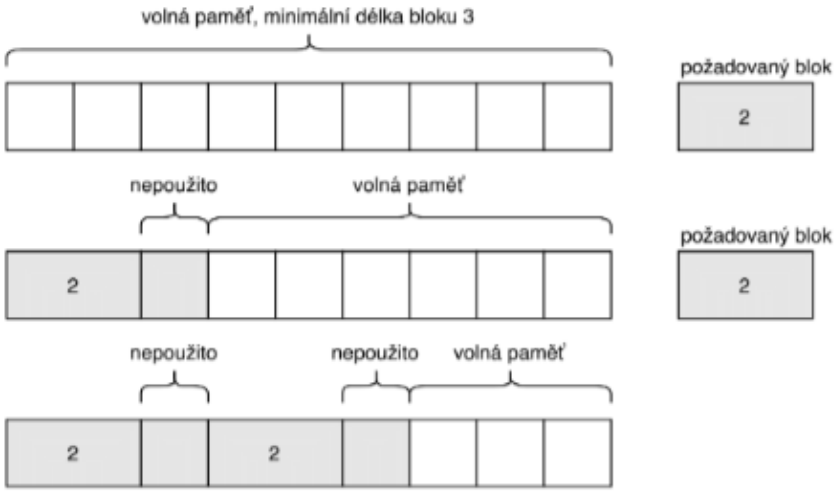


Figure 15: priklad

Vnitrni fragmentace mame pevnou delku bloku treba 3
chceme 2 a ale dostaneme 3
mame misto o velikosti 1 ktere ale nemuzeme nijak vyuzit :(

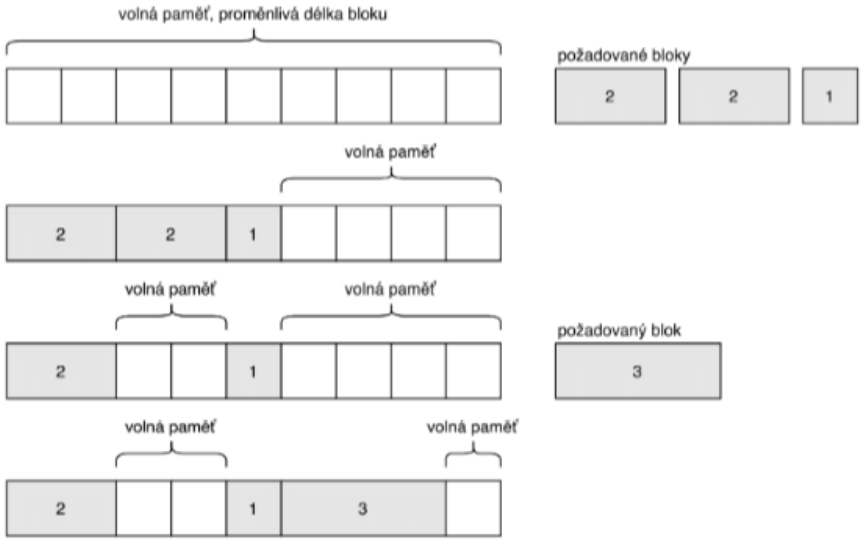


Figure 16: example

Vnejsi fragmentace

po uvolneni tam vznike misto tak ci tak
treba viz obrazek tam zniknlo volne misto velikosti 2 a do nej se nevleze blok velikosti 3

Algoritmy pro vyber bloku:

- first fit
  prvni ktery tam pasuje
  chci blok delky 3 a vezmu prvni co tam pasuje
- best fit
  vybiram ten co tam nejlepe pasuje
- worst fit
  vybiram ten co tam pasuje nejhur
  nejhorsi udajne

Nutny kompromis
zalezi na situaci Problemy:

1. fragmentace zabranuje effektivnimu vyuziti pameti (1/3 je nepouzita)
2. neresi situaci, kdy se program do pameti (jiz) nevejde

Buddy system
alter. ke stejne a promenlive velikosti bloku -> **kompromis**
dostupna pamet 2^u
bloky o velikosti 2^k , $l \leq k \leq u$
reprezentace bin-tree, treba ulozit v pameti
Pridelani pamet:

- pokud je blok vetsi nez 2^k je rozdelen na dva
- rekurzivne se pokracuje dokud neni nalezen nejmensi blok do ktereho se 2^k vejde

Uvolneni pameti:

- pokud je sourozenec volny dojde k margnuti do vetsiho bloku
- rekurzivne se pokracuje dokud lze bloky spojovat

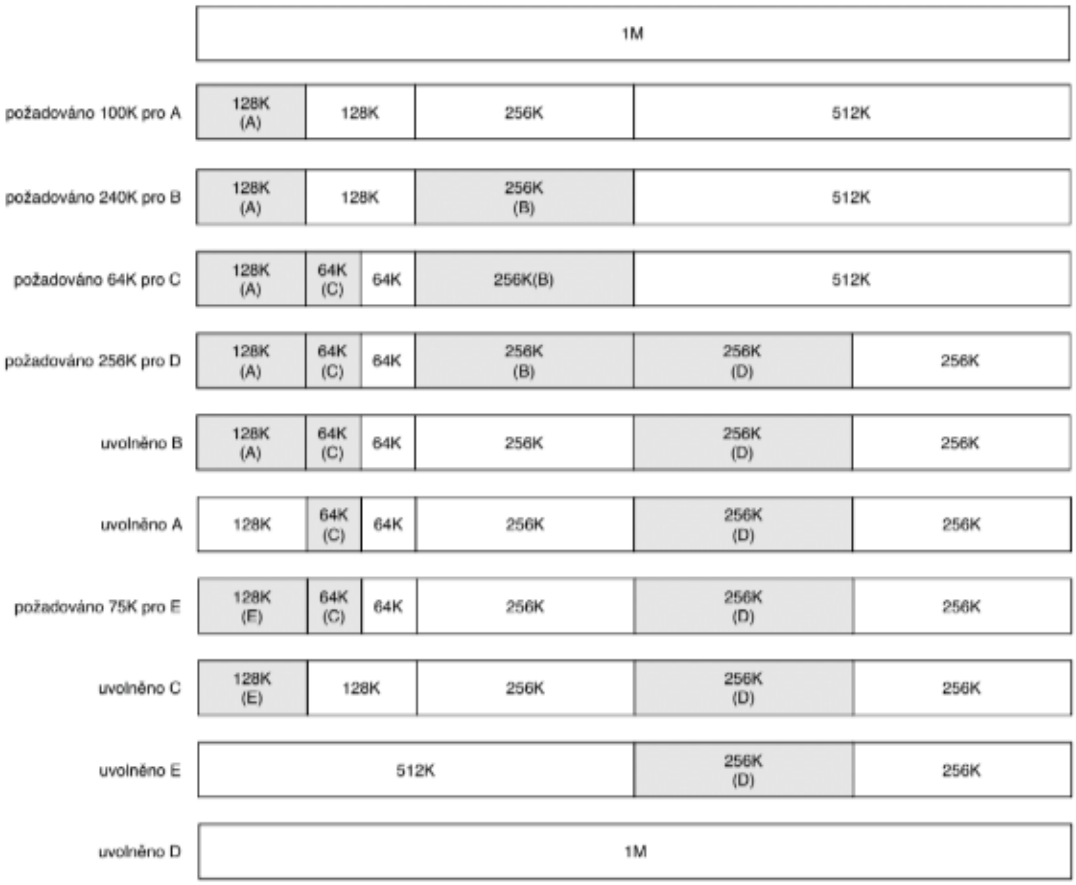


Figure 17: ehm nejak mi ujela tu stranka xdddddd

6 Segmentace, stránkování, virtuální paměť.

strankovani fyzicka adreasace - "tak jak je to tam vedle sebe", puzivaji ji obvody **logicka adresace** - "tak jak je to logicky chapano"

- **abstrakce nad pameti**
- adresni prostor programu rozdelen na stranky (logicka adresace)
- stranky pro ulozeny v pameti ramcich (fyzicka adresace)
- velikosti stranky/ramce dana OS (**obvykle 4** nebo 8kb, ale i jine)

- OS eviduje volne ramce a ramce pridelene procesu

Adresuji konkretni stranky ktere jsou ulozeny v ramcich (ramce jsou fyzicke misto v pameti)
stranka je jen kocept kterou pouziva OS

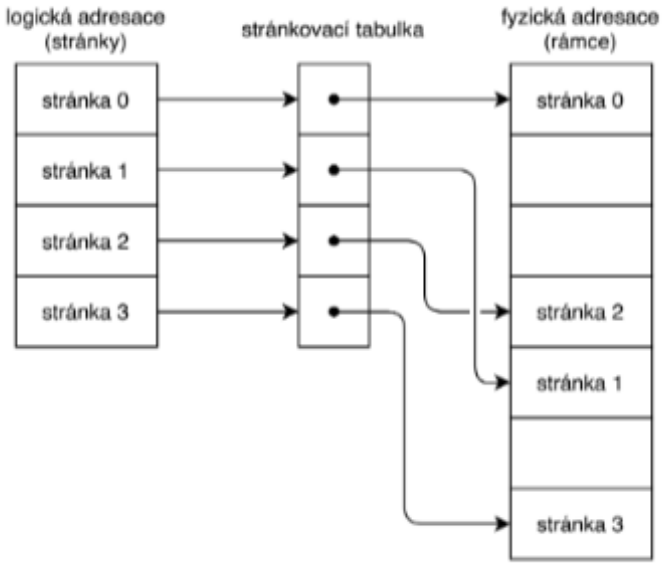


Figure 18: Enter Caption

Princip Stranek Logicka adreasce - jednotlive stranky
Fyzicka adresace - skutecna pamet
velikost stranky = velikost ramce
Strankovaci tabulka - eviduje odkazy kde jsou ty stranky
prvni stranka na prvnim radku tabulky, druha stranka na 2 radku tabulky
i nepouzite stranky maji v tabulce misto
stranka 1 nemusi byt hned po ni stranka 2
to ze to nemusi byt za sebou je to dulezite
resime tim vnejsi fragmentaci
vnitrni ne uplne protoze mame fixni velikost stranky
cim mensi blok tim vetsi rezie

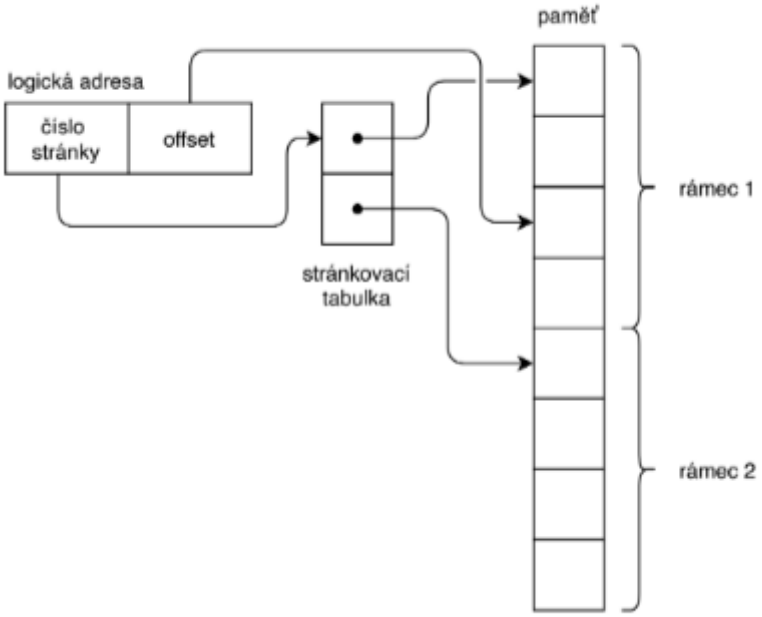


Figure 19: Enter Caption

Adresace logicka adresa - cislo stranky x offset
vemte si to jako by stranka bhyla velky array
v ramci je informaci vic - offsetem vybirame konktretni vec
musime mit HW podporu na prevedeni logicke adresy na fyzickou
tlb cashe - uchovava adresy poslednich stranek

dochazi k plytvani mista (mame hodne stranek) - muzeme mit ulozeno neco na strance 1 a pak az na strance 7712 a mezi tim ty stranky budou "volne" ale budou tam
index je klic algoritmy
reseni:
hiearchie 4 vrstev tabulek
Tabulka 1 odkazuje na dalsi tabulky az nakonec na tu adresu

Princip Lokality

vyhodne s pohledu programatora , muzeme psat programy vice efektivne
kdyz pristupuji do pameti tak se cte oblast treba cely ramec
pokud pracuji jen v ramci tak to bude rychle
pokud budu pracovat ve vice ramcich tak bud je mam ulozeny v cashe nebo ne pak musim spocitat tu adresu (rezie navic)
priklad treba nasobeni velkych matic
jde o to jak se cte ta pamet
memi se radek , chci precist dalsi radek ten uz v cashe neni

v hodne zohlednit jak to funguje muze resultnout ve vetsi vykon
proto treba se vyuzivaji b stromy
aby se limitovaly diskove opera - ten vypocet te adresy z logicke do fyzicke

Segmentace pameti
rozdeleni pameti procesu na logicke casti (segmenty)
obecne je segmentace **nezavysla na strankovani**

- segmentace je mozna i bez strankovani (specialni pripad sravy pameti)
- velikost segmentu muze byt nezavysla na velikosti stranky

abstrakce nad strankovanim
nad strankami zavedu segment
segment je nezavysli na adresaci
pro jednoduchost segment = skupina stranek
kazdy segment ma pristupova opravneni (napriklad oddeleni zasobnik, halda)
dubod bezpecnost
v ramci procesy , i v ramci jednoho konkretniho procesu
aby nedoslo k tomuz ze data zasobniku jsou prepsany haldou
sdileni semgentu = setri pamet
tabulka semgentu

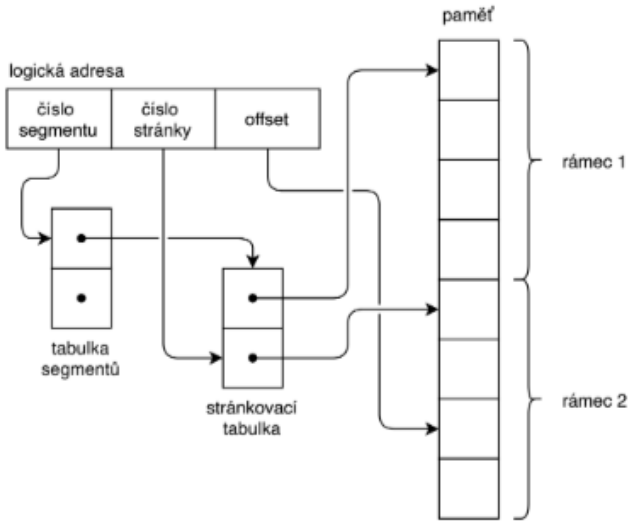


Figure 20: Enter Caption

princip segmentace segmentovaci tabulka -
logicka adresa cislo semgmentu x cislo steanky x offset

sprava pameti na urovni programu v ramci procesu je pridelena pamet, to co jest na obrazku je rozdelena cast data procesu, coz jsou jednotlive segmenty jednotlivim castem se muze menit velikost
diky segmentace os je schopen pohlidat aby se nazvajem neprepisovaly
muzeme rict ze nejaka cast kodu ma opravneni jen cist atd

Strankovani programatora nemusi zajimat, segmentace ano
z pohledu programatora se to odehrava na urovni segmentu a tam to lze ovlivnit
treba kdyz napisu fukci ktera vola funkci mimo segment tak rezie
OOP podporuje modularitu
lepsi zazadat naraz o vic pameti , nez porad o kousky nove a nove

- dynamicky rostouci datove struktury
- modularita

Manualni = ponechana na programatorovy
lze ovlivnit pocty vypadku stranky
malloc , alloc , free ...
rychlost bez zbytecne rezie
je to tezsi a muze

automaticka = garbage collector (John MCCarthy)
pro jazyk LISP
zjednoduseni vyvoje na ukor HW prostredku
ruzne modely a implemetace (pocitani odkazu, mark&sweep, kopirujici, ...)
vetsi rezie , sam urdzuje informace o pametovych mistech ktere jsou pouzivant

Figure 21: Enter Caption

vyrtualni pamet nejlavnejsi z uzivatelskeho pohledu
mam fyzicke ramce , stranky (logicka adresace) zajima me jenom toto,to jak je to ve fyzicke pameti nas zajimat nemusi, programu muze byt jedno ze je rozdelen do nekolka stranek
pokud nejaky stranky nejsou pouzivany tak je nemusime udrzovat v hlavni pameti a presnume je na pevny disk
Swap oddil - misto kam se ukladaji nepouzivane stranky
kazdy porcces je dan casti pameti kde se to odehrava
muzeme spustit vyrazne vic procesu nez se vejdou do pameti
vzdy nacteme jen konkretni stranky ktere jsou aktivni
nevyhody: rezie prace s I/O (diskem)
PageFault - pristup k strance ktera neni v pameti
kdyz neni stranka v hlavni pameti tak dojde k
Vymena ramcu:

- pokud existuje volny ramec pouziji ho
- pokud ne vybiram obet

prace s swap oddilem je pomala

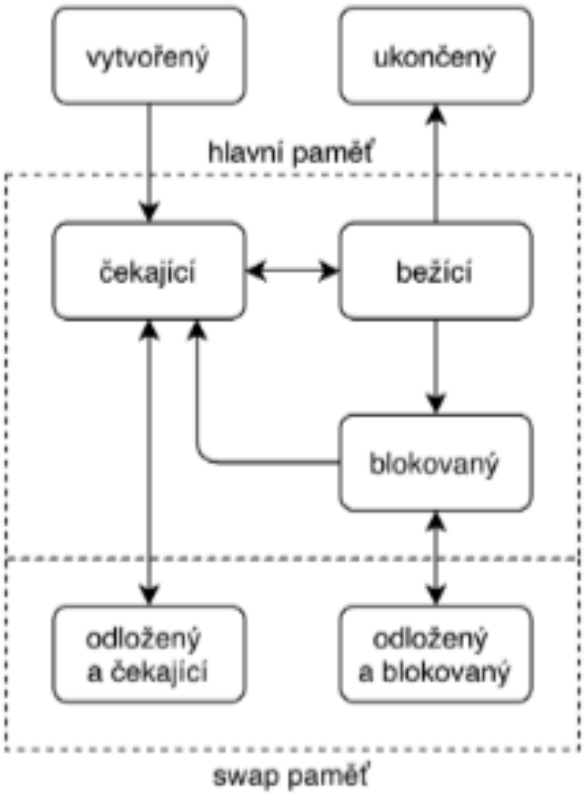


Figure 22: Enter Caption

procse muze byt ulozen v nepouzivane strance -> novy zivotni cyklus procesu
Odlozeny cekajici - cekajici ve swap oddilu
Odlozeny blokovany - blokovany ve swap oddilu
Bezici proces nemuze by ve swap oddilu!!

Jak vybrat Obet???

- Idealni obet
  vybrat ramec co nebude nidky pouzit
  nelze nemame kristalovou koly
- Nejstarsi stranka
  beladyho anomalie - cim vic ramcu tim vic vypadku
  nejstarsi stranky koresponduji s OS
- nejdele nepouzita stranka
  to ze stranka nebyla pouzivana neznamena ze ji ted nebudu potrebovat
- nejmene pouzivana stranka
  nejmensi skoda
- dalsi...

Neni mozny urcit idealni obet, kompromis
i urceni idealni obeti by stalo spousto rezie (nevyhodne)

Vymena stranky
obracena strankovaci tabulka - datova struktura (haha algo xdd like and subsribe)
pomaly operace I/O
Pridelovani ramcu

- kolik procesu?
- nekolik strategii
  - pevny pocet
    nevejde se to tam , caste page fault
  - promenlyvy pocet ramcu
  - pracovani mnozina
    pridelim mu tolik kolik potrebuje na zacatku a pak neco navic
    (Caste pouziti)

malo stranek v pameti?

- neustale vypadky pameti
- OS spusti dalsi procesy
  - pocitam nema co delat spusti dalsi program
  - dochazi k dalsimu a dalsimu vypadku
- Trasching - Opakovane page Faults diky os viz drive
  nic se nedeje rezie spotrebbovana na vymenu stranek

7 Správa diskového prostoru

sprava dat **persistentni data** - preziji vypnuti pocitace a zkonceni procesu
velikost nemusi by takova rychlost
I/O zarizeni (pevny disk)
uloveni dat v ramci procesu je velmi obtizene
dlouhodobe uchovavani dat

- moznost ulozeni velkeho mnozstvi dat
- data musi "prezit" proces, ktery je vyuziva
- k datum muze pristupovat vice procesu

operace cteni a zapis nejsouj dostatecne

- jak najit data?
- jak zajistit vlastnictvi dat?
- jak poznat, kde lze data zapsat?
- -> nutnost spravy

evidence volneho mista (bloku)
stejne jak v RAM

- seznam
- bitmapa (pouzivanejsi i zde)

Soubor
abstrakce nad Daty
data jsou anonimni , tim ze rekneme ze tady neco zacina a tady neco konci
zavedena abstrakce a tim se zhmnotni neco co je soubor

zakladni logicka jednotka informace vytvorena/pouzivana procesem
persistentni ulozeni dat

vetsina OS ma filozofie vsechno je soubor
sprava souboru = souborovy system

- Napriklad
  - FAT
    USB treba
  - ext4 , ext3
  - ntfs apfs
- maji to i flaeky treba
- sluzby zabezpeceni , zalohovani , optimalizace , komprese , de-duplikace
  (mame 2 soubory stejny ale budou se evidovat jen jako 1)

vlastnosti souboru:

- typ souboru
  strukutura samotnyho souboru
- meta-data - atributy
- operace
- opravneni

Sprava: pridelovani pameti soubor je proud dat 0001111000011100
ulozeno po blocich
velikost bloku

- velky bloky
  rychle, ale nevyuzite misto (vnitrni fragmentace)
- mensi bloky
  velka rezie
- **compromis** to co se pouziva

pridelovani bloku (ma vliv na vykon):

- souvysla alokace

- spojita alokace
- indexova alokace
- i-node

Souvysla alokace treba znat velikost souboru
vnejsi fragmentace
nutna defragmentace - neni uz dneska potreba
Random acces pristup do pameti - konstantni prisutp

spojita alokace
spojovy seznam
mozny ukladat i na preskacku
potlacena vnejsi fragmentace
poruseni principu lokality - nutna konsolidace (preskladavani dat)
konsolidace je neco ako defragmentace (narocni proces) Cteni souboru - pomalejsi , a musim odstranit odkaz z dat (rezie)

indexova alokace
kombinace predchozich
odkazy na dalsi bloky ulozeny v tabulce (file allocation table, FAT)
FAT tabulka zabira misto (velikost zavisla na velikosti disku)
odkazy nejsou soucasti dat
alokacni tabulka je na zacatku ulozena
kompromis mezi obema dvema

I-node (index node)
linux , unix (to spravne)
bloky obsahuji atributy souboru a rezijni informace
soubor reprezentuje jako mnozinu odkazu na dalsi bloky a to muzou byt dalsi odkazy na dalsi bloky nebo
wiki asi to tu jisti xd
zavisle na poctu souboru ne velikosti disku
skoro az ideal (dle trnecky)

Nazev souboru
omezena delka (MS-DOS)
variabilni delka - nekonstantni
z pohledu uzivatele drobnost ale z pohledu souboroveho systemu zasadni
jak implementovat variballni delku

- vyhradime dostatek mista - plytvani (nemuzeme dovolit)
  vnitrni fragmentace

- promenliva delka
  vnejsi fragmentace

- omezeni delky na malou velikost - jsme zpatky u dosu necheme
- zanev souboru = odkaz (princip lokality je v pici)
  to se pouziva ale je to kompromis

Adresar
organizace souboru = adresarova struktura

- stromova struktura (ne nutne lze udelat ciklus "linkama")
- zapis cesty lomitko, zpetne lomitko
- adresate . , ..
- abolutni a relativni cesty vuci aktualnimu procesu
- operace s adresari

implementace

- seznam - sekvencni prochazeni soboru (zleee)
- hash - problem s kolizemi
- b-stromy - dnes se pouziva nejvic (z alga zname lets go osicka cooking)

svazek organizace diskovyho prostoru struktura disku je skoro az konstantni casti:

1. master boot record - informace o jednotlivich svazchich a dalsi rezijni informace
2. tabulka svazku
3. samotne svazky

struktura svazku

1. bootblock - s tudma se muze bootovat (maji to i ty se kterych se nebootuje
2. superblock
3. evidence volneho misgta
4. jednotlive soubory
5. korenovy adresar
6. soubory a slozky

zurnalovani (meli by jsme neco vedet)
data se zapisuji nejdriv do cashe
pri chybe pri praci s cashe nebo se vypne elektrina
priklad:
zapisi informace o souboru kolik bude zabirat mista a nastavim vlastnictvi na me pak vypnu pocitam zustanou tam puvodni data ale budu mit pristup k nim protoze budu vlastnikem - jiz to lenze provest gg
zurnalovani - **pouze konzistentni stavy**
princip transakce (postup operaci ktere se vykonavaji)
pokud transakce dojde do konce je povazovana za dokoncenou
pokud selze - neni povazovana za dokoncenou
vede se zapisnicek zmen ktere se maji provest
jakmile jsou zmeny provedeny jsou odstraneny
pri chybe se podiva do zapisnicku co se ma jeste provest
treba se vypne pc pak se pri startu podivame do zapisnicku a udelame zpatky zmeny co se meli provest pro zotaveni z chyby

uzivatelske rozhrani
obvykle system "oken" - windwos manager
cela rada knihoven

- kazdy os ma sve typicke (u windows nejde menit)
- existuji portabilni (GTK , Qt, Swing)
  nemusi vyhovovovat zvyklostem daneho OS
  Musi to jen byt tim os podporovano

u kazdeho os se to chova trochu jinak vznika s toho paskvil obzvlast u mobilnich telefonu

poskytuji programum (pres api) zakladni gragicke prvky (tlacitka popisky seznamyh)
k samotnemu vykreslovani se vratime pozdeji (u grafice)
zavolam knihovnu hello knihovno vykresli mi tlacitko a mam tlacitko

Vyrtualizace
beh **vice** pocitacu **na jednom fyzickem hardware**
dualboot - mam ja xd (2 os na jednom systemu) neni to virtualizace bezi jen jeden
hypervisor - ten co vytvati iluzi vicenasobnyho HW
Vyrtualni stroj je nezavysly na ostatnich - moznost ruznuch OS
nutna podpora hw - v dnesni dobe skoro vsechny (nejake cheap sracky ne)
pouziti:

- sandboxing
  z bezpecnostich a testovacich duvodu
- cloud
  zaklad cloudu

benefity : uspora a udrzba (muzu mnoho os provozovat na jednom pc) je konsolidace bezpecna? - mam 20 pc vs 1 pc a 20 virtualnich pc
Hw chyby jsou mene castejsi nez softwarovy
letecka doprava je statisticky bezpecnejsi ale 9/11 bylo uplne v pici (nice yaping ale true)
typy virutalizace:

- plna virtualizace (VirtualBox , vmWare ...)
  virutalni stroj netusi ze bezi virtualne
- para-virtualizace (Xen)
  stroj vy ze je virtualizovany tak muze to tezit rychlostne
  vyzadovan specialni os ktery to muze ridit a jidnou archi Texture
- kontejnerizace (dokr)
  slaba vyrtualizace ,vyrtualizuji pouze aplikaci (mysli si ze je sama)
  ale prostredi je puvodni os spis program ktery to realizuje
  snadna sprava
- aplikacni virtualizace (Dzejva , had , ostre videni)
  programovaci jazyky, program vytvori vyrtualni stroj na kterem bezi ten code

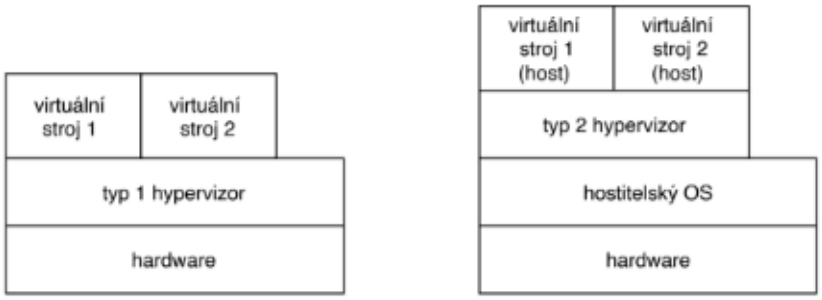


Figure 23: Enter Caption

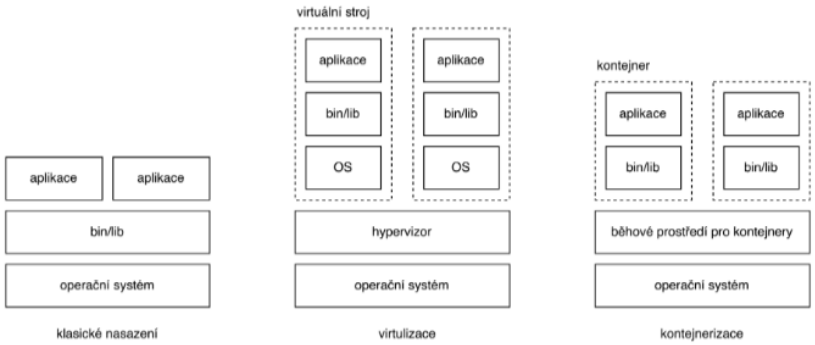
hypervizor:

1. typ 1 (MS Hyper V)
   mam HW a samotny hypervisor je nad harware (v urovni os)
   nad tim mame ty vyrtualni stroje
   Paravirtualizace vyzaduje typ 1 hypervisor

2. typ 2 (Linux KVM, VirtualBox)
   mam hw nad tim hostitelsky os
   nad tim mame hyper visor
   nad tim mame ty vyrtualni stroje

7.1 TOHLE NEBYLO NA TE PREDNACE TAKZE TU JSOU JEN SCREENSHOTY SLIDU DUVOD JE TO ABY TO BYLO VSE NA JEDNOM MISTE NEVYHODA VETSI REZIE Z ME STRANY Compromis - dam sem slidy ale nedohledam nic navic

Virtualizace: Srovnání nasazení aplikace



83 / 85

Figure 24: Enter Caption

Odbočka: Distribuované systémy

- propojení skrze počítačovou síť (naše další velké téma)
- uzly jsou navzájem nezávislé, doposud procesy běžely na jednom uzlu, nyní na různých uzlech → třeba synchronizace
- synchronizace = zasílání zpráv (obecně nespolehlivé, problém zpoždění)
- různé časy na uzlech
- problémy
  - vzájemné vyloučení (analogie kritické sekce)
  - volba lídra
  - shoda
  - replikace (problém konzistence dat)
  - globální stav

84 / 85

Figure 25: Enter Caption