

09. Memory management

ZOS 2014, L. Pešíčka



2. zápočtový test

11. 12. 2014 (čtvrtek)

učebna je rezervovaná EP130 od 18:20

psát se začne cca 18:30

45 minut na test

sedejte si tak, aby zůstala volná řada a i
sousední místo vedle vás

Podrobnosti obsahu – Courseware - Testy



Otázky na procvičení (Teri)

- Courseware – Přednášky – Opakování
Portál s otázkami na procvičení
- <http://students.kiv.zcu.cz/teri/>
Testy -> ZOS
i pro přípravu na zkoušku
je ze všech přednášek, není jen 1.-8.

1 Seznam testů > 2 KIV/ZOS > 3 Otázka

Otázka

Znění otázky

Spustím-li pomocí `execl(...)` jiný program, změní se PID procesu?

Odpověď - ano/ne

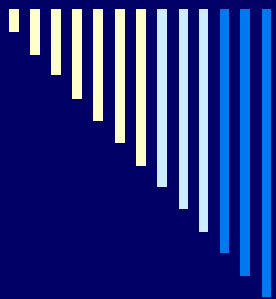
odpovědi:	zaškrtněte správnou:
ano	<input type="radio"/>
ne	<input type="radio"/>

<< zpět

| **hotovo** |

další >>

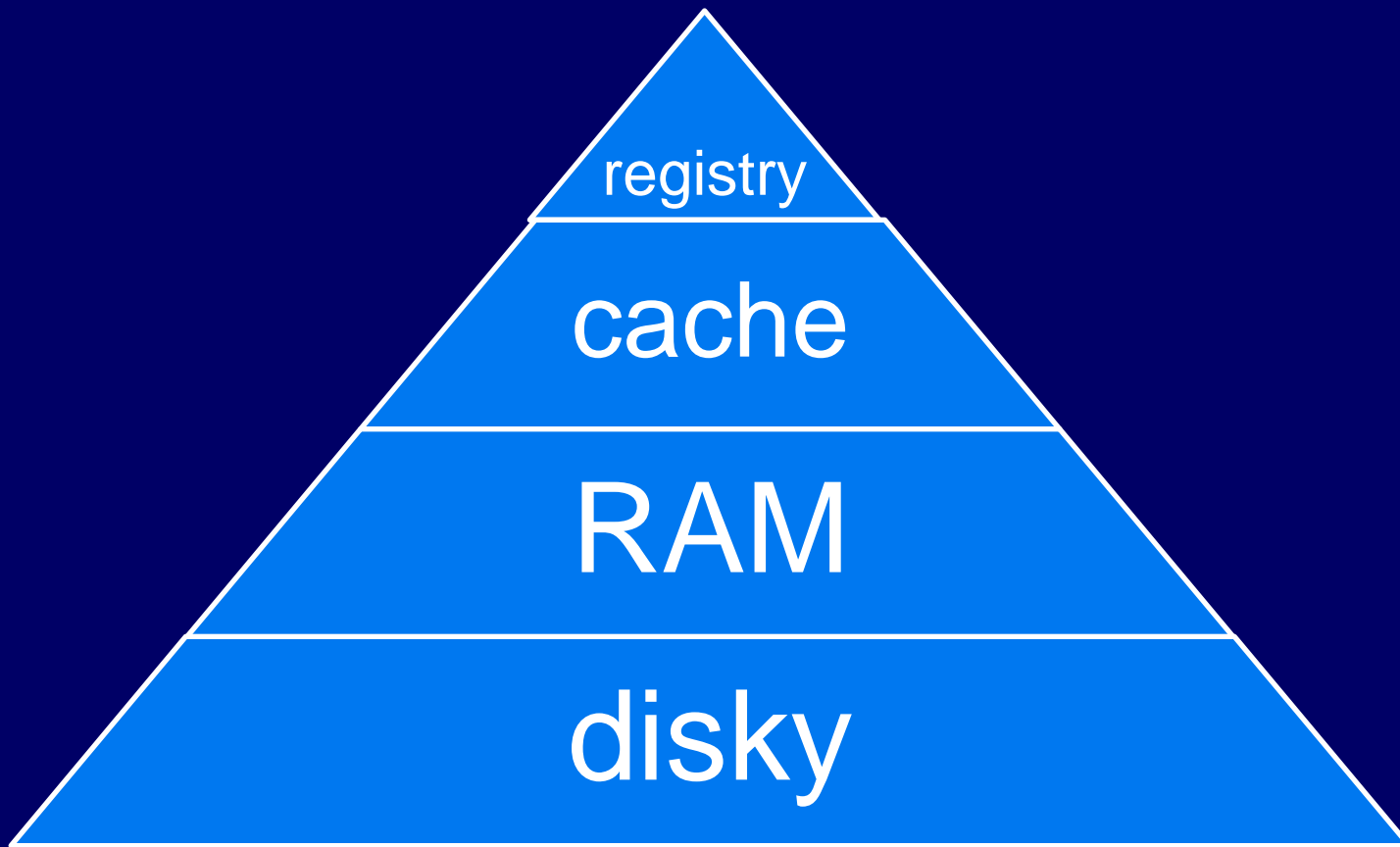
Nápověda k otázce.



Správa paměti

- „paměťová pyramida“
- **absolutní adresa**
- **relativní adresa**
 - *počet bytů od absolutní adresy (nějakého počátku)*
- **fyzický prostor adres**
 - fyzicky k dispozici výpočetnímu systému
- **logický adresní prostor**
 - využívají procesy

drahé, rychlé, malá kapacita



levné, pomalé, velká kapacita



Modul pro správu paměti

- informace o přidělení paměti

- která část paměti je volná
- přidělená (a kterému procesu)

- přidělování paměti na žádost

- uvolnění paměti

- zařazení k volné paměti

- odebírá paměť procesům

- ochrana paměti


- přístup k paměti jiného procesu
 - přístup k paměti OS
-



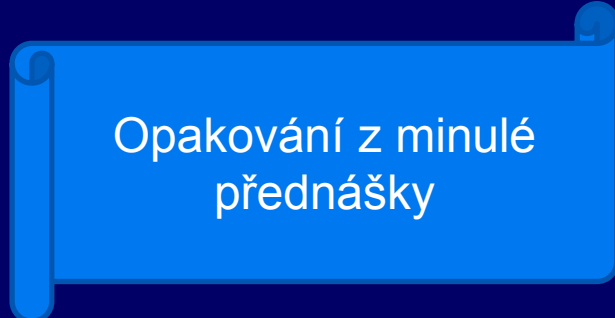
Memory management

□ Základní mechanismy

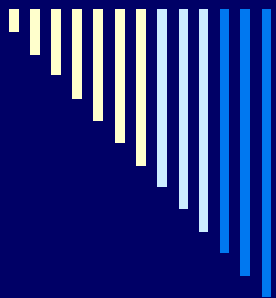
- Bez odkládání a stránkování
- Jednoprogramové systémy
- Multiprogramování s pevným přidělením paměti
- Multiprogramování s proměnnou velikostí oblasti
- Správa paměti
 - Bitové mapy
 - Seznamy
 - First fit, best fit, next fit
 - Buddy system



Celý proces se
musí vejít do
paměti



Opakování z minulé
přednášky



Statická a dynamická relopace



Relokace a ochrana

- Problémy při multiprogramování
(více programů současně v paměti):
 - Relokace
 - Programy běží na různých (fyzických) adresách
 - jednou je ve fyzické paměti od adresy 1000, jindy od 2000
 - Ochrana
 - Paměť musí být chráněna před zasahováním jiných programů
-

ukázka překladu .c programu

```
eryx1> ls
main.c  makefile
eryx1> make
gcc -lpthread -O3 -c main.c
gcc -lpthread -O3 -o fork_sm main.o
eryx1> ls
fork_sm main.c main.o makefile
eryx1>
eryx1> file main.o
main.o: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
eryx1>
eryx1> file fork_sm
fork_sm: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically l
inked (uses shared libs), for GNU/Linux 2.6.8, not stripped
eryx1>
eryx1> █
```

zdrojový soubor
objektový modul
spustitelný soubor

main.c
main.o
fork_sm



Relokace při zavedení do paměti

jak je program **vytvořen a spuštěn**:

překladač + linker

□ Překlad a sestavení programu

- Aplikace ve vysokoúrovňovém jazyce
- Větší SW – rozděleny do modulů – musejí být přeloženy a sestaveny do spustitelného programu
- **Objektové moduly**
 - Výsledkem překladu
 - Příkazy ve zdrojovém textu – přeloženy do stroj. instrukcí
 - Zůstávají symbolické odkazy – adresy prom., procedur,fcí



Relokace při zavedení do paměti 2

- Výsledný spustitelný program
 - Sestavení (**linkování**) modulů a knihoven

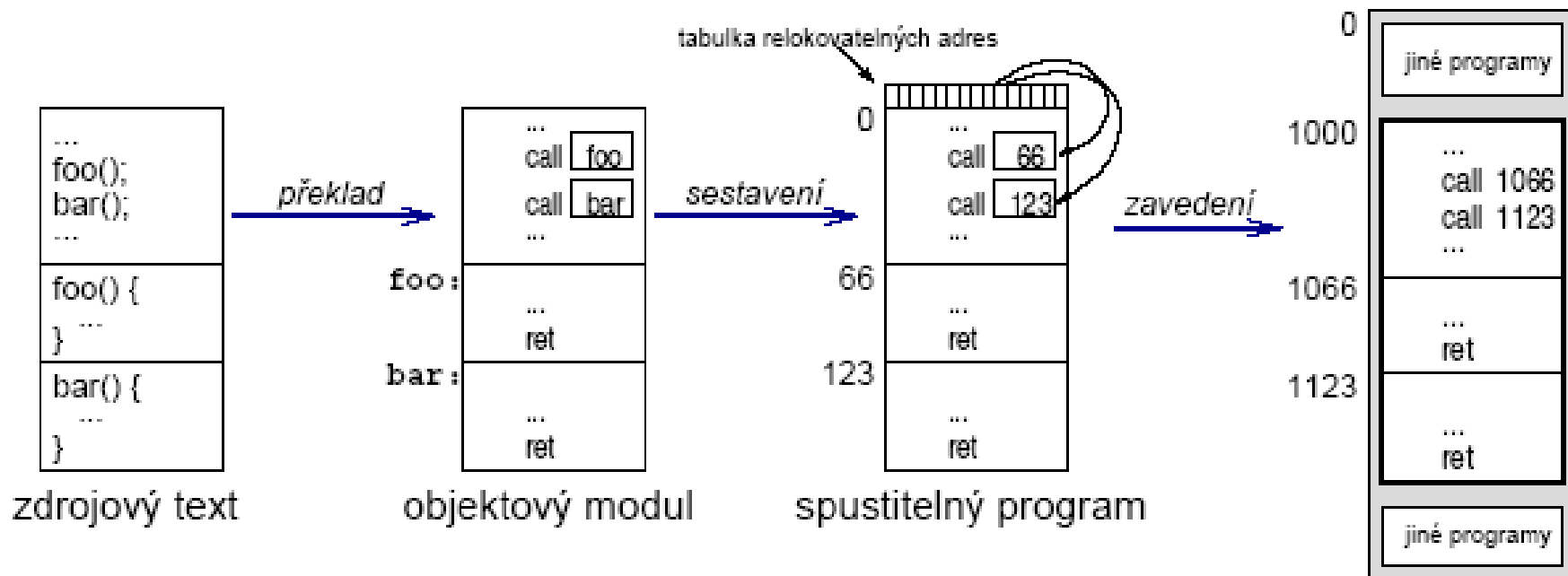
- Při sestavení se řeší hlavně **externí reference**
 - Všechna místa výskytů referencí – seznam
 - Když už je adresa známa – vloží se všude, kde se používá
 - Symbolické odkazy se převedou na číselné hodnoty
 - Výsledek – spustitelný program



Relokace při zavedení do paměti 3

- Komplikace při více programech v paměti
 - Příklad
 - 1. instrukcí programu volání podprogramu *call 66*
 - Program v paměti od adresy *1000*, ve skutečnosti provede *call 1066*
- Jedno z řešení – **modifikovat instrukce programu při zavedení do paměti**
 - **Linker** – do spustitelného programu přidá seznam nebo bitmapu označující místa v kódu obsahující adresu
 - Při zavádění programu do paměti se každé adrese **přičte adresa začátku oblasti**

Relokace při zavedení do paměti





Statická relokalace

- ❑ Popsanému způsobu se říká statická relokalace
- ❑ Adresy se natvrdo přepíše správnými
- ❑ Např. OS/MFT od IBM

dále budou popsány dva odlišné mechanismy ochrany paměti:

- mechanismus přístupového klíče
- mechanismus báze a limitu



Ochrana – přístupový klíč

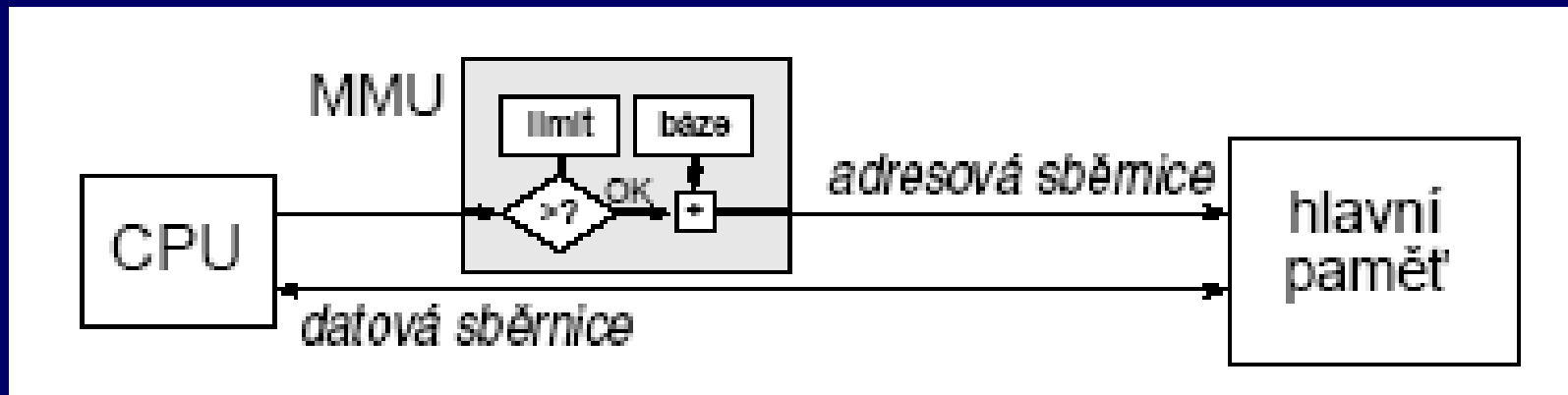
- Proces mohl zasahovat do paměti jiných procesů
- IBM 360 – **přístupový klíč**
 - Paměť rozdělena do bloků 2KB
 - Každý blok – sdružený hw 4 bitový kód ochrany
 - PSW procesoru obsahuje 4 bitový klíč
 - Při pokusu o přístup k paměti jejíž kód ochrany se liší od klíče PSW – výjimka
 - Kód ochrany a klíč může měnit jen OS (privilegované instrukce)
 - Výsledek – ochrana paměti

Klíč je
spjatý s
procesem

Možnou metodou ochrany
paměti je ochrana přístupovým
klíčem

Ochrana - mechanismus báze a limitu

- Jednotka správy paměti MMU (je uvnitř CPU)
- Dva registry – báze a limit
- Báze – počáteční adresa oblasti
- Limit – velikost oblasti



Zde je MMU jen pro názornost mimo CPU



Mechanismus báze a limitu

□ Funkce MMU

- převádí adresu od procesu na adresu do fyzické paměti
- Nejprve zkontroluje, zda adresa není větší než limit
 - Ano – výjimka, Ne – k adrese přičte bázi

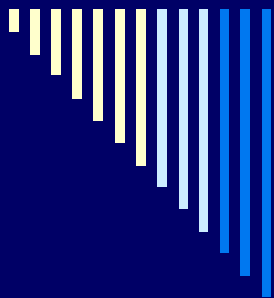
□ Pokud báze 1000, limit 60

- Přístup na adresu 55 – ok, výsledek 1055
- Přístup na adresu 66 – není ok, výjimka



Dynamická relokalace

- Provádí se **dynamicky za běhu**
 - patří sem uvedený mechanismus báze a limitu
 - Nastavení **báze a limitu** může měnit pouze OS (privilegované instrukce)
-
- Např. 8086 – slabší varianta (nemá limit, jen báze)
 - Bázové registry = segmentové registry DS,SS,CS,ES



Správa paměti s odkládáním celých procesů

(Proces se vejde do fyzické paměti)



Poznámky

- Pro **dávkové systémy** – dosud uvedené mechanismy - přiměřené (jednoduchost, efektivita)
- **Systémy se sdílením času** – můžeme mít více procesů, než se jich **vejde** do paměti **současně**
- 2 strategie
 - **Odkládání celých procesů** (swapping)
 - Nadbytečný proces se odloží na disk
 - Např. UNIX Version 7; co platí pro velikost procesu?
 - **Virtuální paměť** – v paměti nemusí být procesy celé
 - Překrývání (overlays), virtuální paměť – bude dále



Odkládání celých procesů

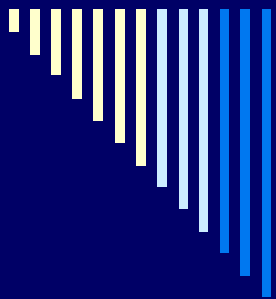
co víme o velikosti procesu?

- data procesu mohou **růst**
- pro proces alokováno **o něco více** paměti, než je třeba
- potřeba více paměti, než je alokováno:
 - **přesunout** proces do větší oblasti (díry)
 - překážející proces **odložit** – prostor pro růst procesu
 - **odložit** žadatele o paměť, dokud nebude prostor
 - proces **zrušit** (odkládací paměť je plná)



Odkládání celých procesů

- proces – dva rostoucí segmenty
 - data, zásobník (co se kde alokuje?)
 - možnost rozrůstání **proti sobě**
 - překročení velikosti – přesun, odložit, zrušit



Alokace odkládací oblasti

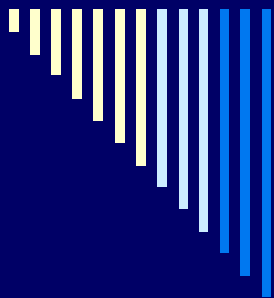
tj. jak vyhradit prostor pro proces na disku:

- na celou dobu běhu programu („pořád do stejného místa“)
- alokace při každém odložení

stejně algoritmy jako pro přidělení paměti

velikost oblasti na disku

– násobek alokační jednotky disku



Virtuální paměť

Proces > dostupná fyzická paměť

{ proces může být i větší
než dostupná fyzická paměť }



Virtuální paměť

- program větší než dostupná fyzická paměť
- mechanismus překrývání (overlays)
- virtuální paměť

Virtuální paměť je to, co se dnes nejčastěji používá



Překrývání (overlays)

- **program** – rozdělen na **moduly**
 - start – spuštěna část 0, při skončení zavede část 1 ...
 - časté zavádění některých modulů
 - více překryvných modulů + data v paměti současně
 - moduly zaváděny dle potřeby (nejen 0,1,2,...)
 - mechanismus odkládání (jako odkládání procesů)
 - kdo zařizuje zavádění modulů?
 - kdo navrhuje rozdělení dat na moduly?
-



Překrývání

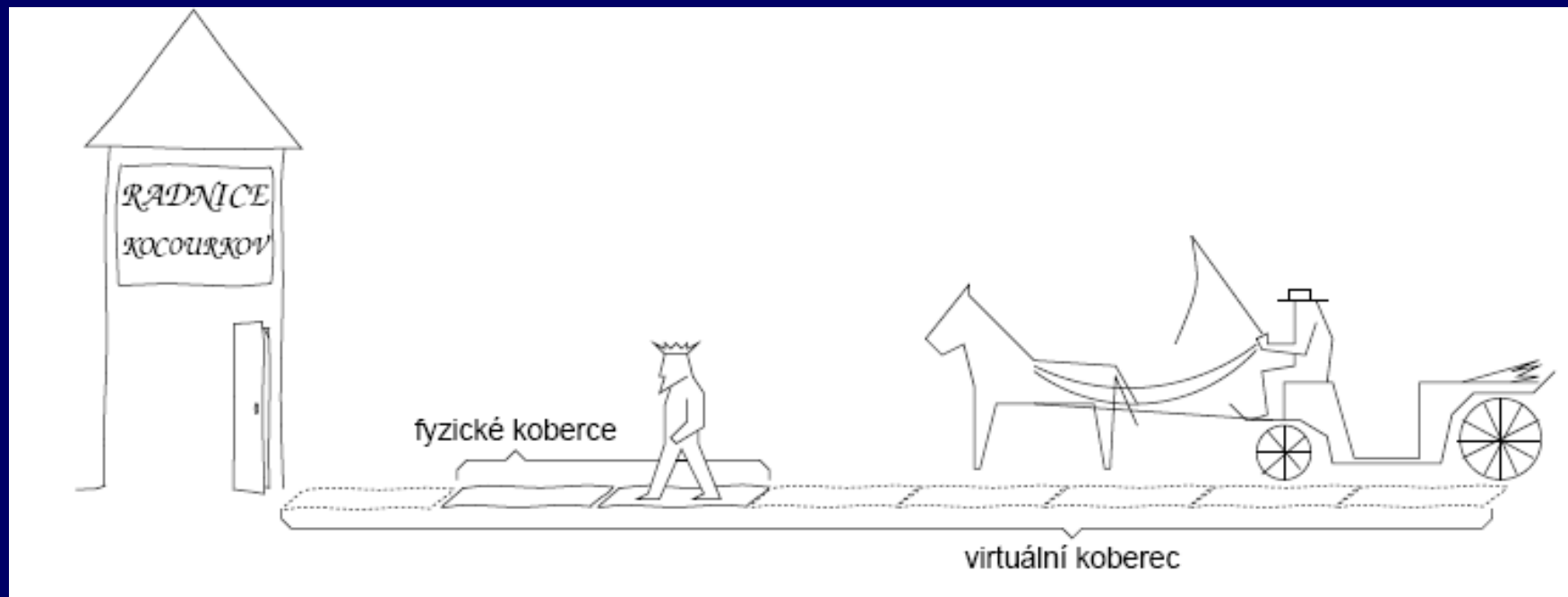
- zavádění modulů **zařizuje OS**
- **rozdělení** programů i dat na části – navrhuje **programátor**
 - vliv rozdělení na výkonnost, komplikované
 - pro každou úlohu nové rozdělení
- příklad – overlay.pas
- snaha, aby se o vše postaral OS



Virtuální paměť

- potřebujeme rozsáhlý adresový prostor
- ve skutečné paměti je **pouze část** adresového prostoru
 - jinak by to bylo příliš drahé
- zbytek může být odložen na disku
- **kterou část mít ve fyzické paměti?**
 - tu co právě potřebujeme 😊

Historie – královský koberec



Na pokrytí celé cesty stačí pouze dva fyzické koberce



Virtuální adresy

- fyzická paměť slouží jako cache virtuálního adresního prostoru procesů (!)
- procesor – používá virtuální adresy
- Pokud požadovaná část VAProstoru JE ve fyzické paměti
 - MMU převede $VA \Rightarrow FA$, přístup k paměti
- požadovaná část NENÍ ve fyzické paměti
 - OS ji musí načíst z disku do RAMky
 - I/O operace – přidělení CPU jinému procesu
- většina systémů virtuální paměti používá stránkování



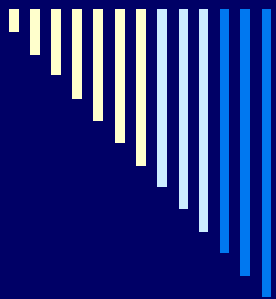
Mechanismus stránkování (paging)

- program používá virtuální adresy
- Musíme rychle zjistit, zda je požadovaná adresa v paměti
 - ANO – převod VA => FA
 - NE – je třeba zavést z disku do paměti
- co nejrychlejší – děje se při každém přístupu do paměti



Pojmy – důležité !!!

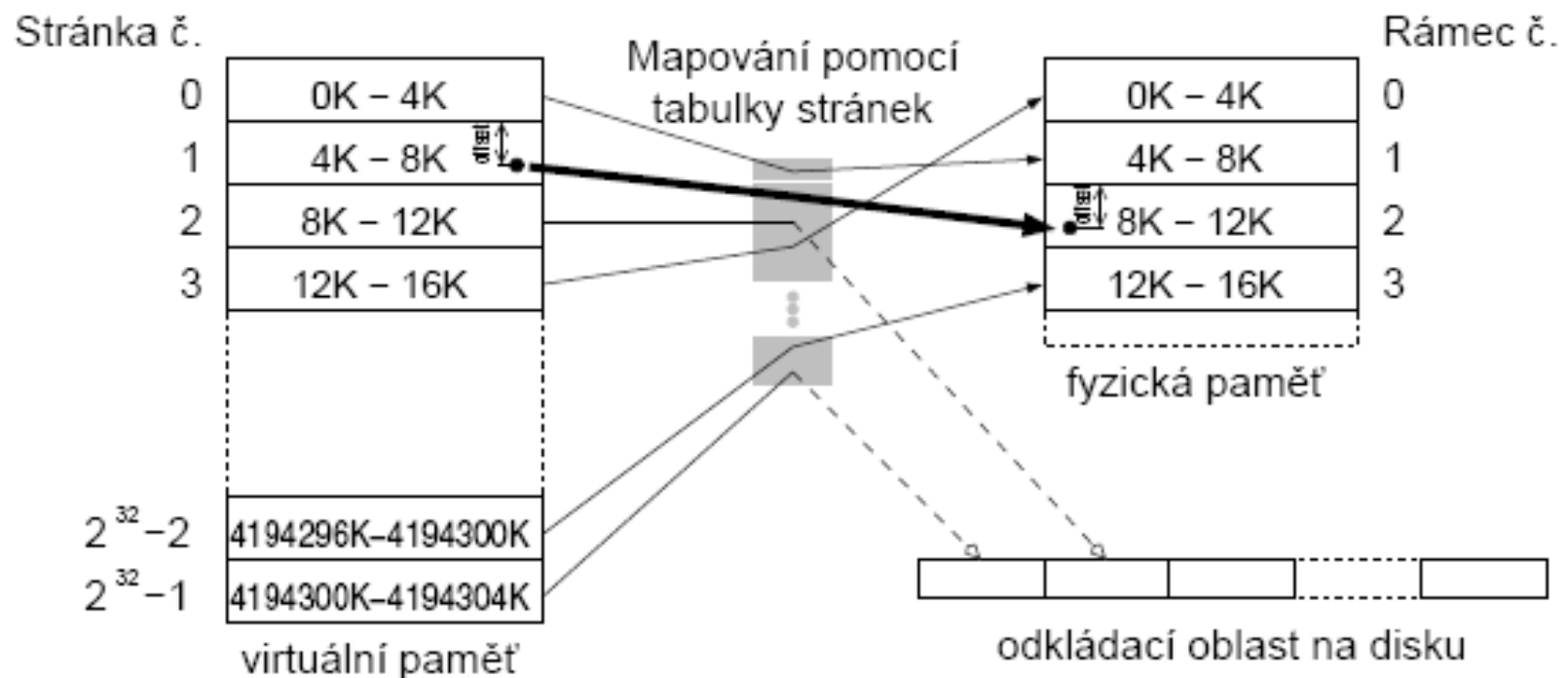
- **VAP – stránky** (pages) pevné délky
 - délka mocnina 2, nejčastěji 4KB, běžně 512B - 8KB
 - **fyzická paměť – rámce** (page frames) stejné délky
 - rámec může obsahovat PRÁVĚ JEDNU stránku
 - na známém místě v paměti – **tabulka stránek**
 - tabulka stránek poskytuje mapování virtuálních stránek na rámce
-



Opakování

- ☐ virtuální adresní prostor
- ☐ fyzický adresní prostor
- ☐ procesy používají VA nebo FA?
- ☐ co dělá MMU?
- ☐ k čemu slouží tabulka stránek?
- ☐ stránka
- ☐ rámec

Stránky jsou mapovány na rámce v RAM, nebo jsou uložené v odkládací paměti na disku



Stránkovaná paměť

Swap na disku



P1

P2

RAM

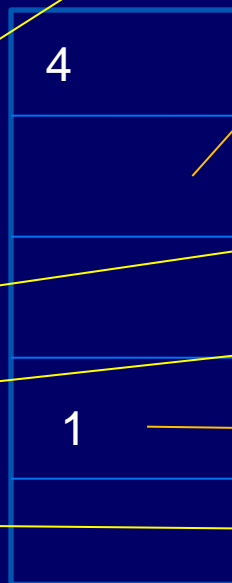
virtuální adresy

stránka např. 4KB

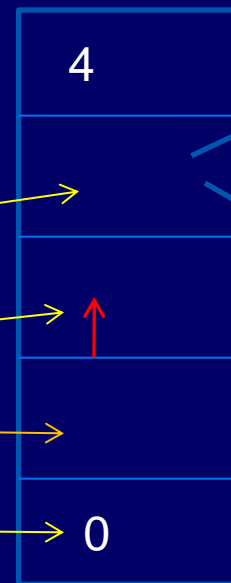
Offset od začátku stránky



Tabulka stránek
Procesu 1



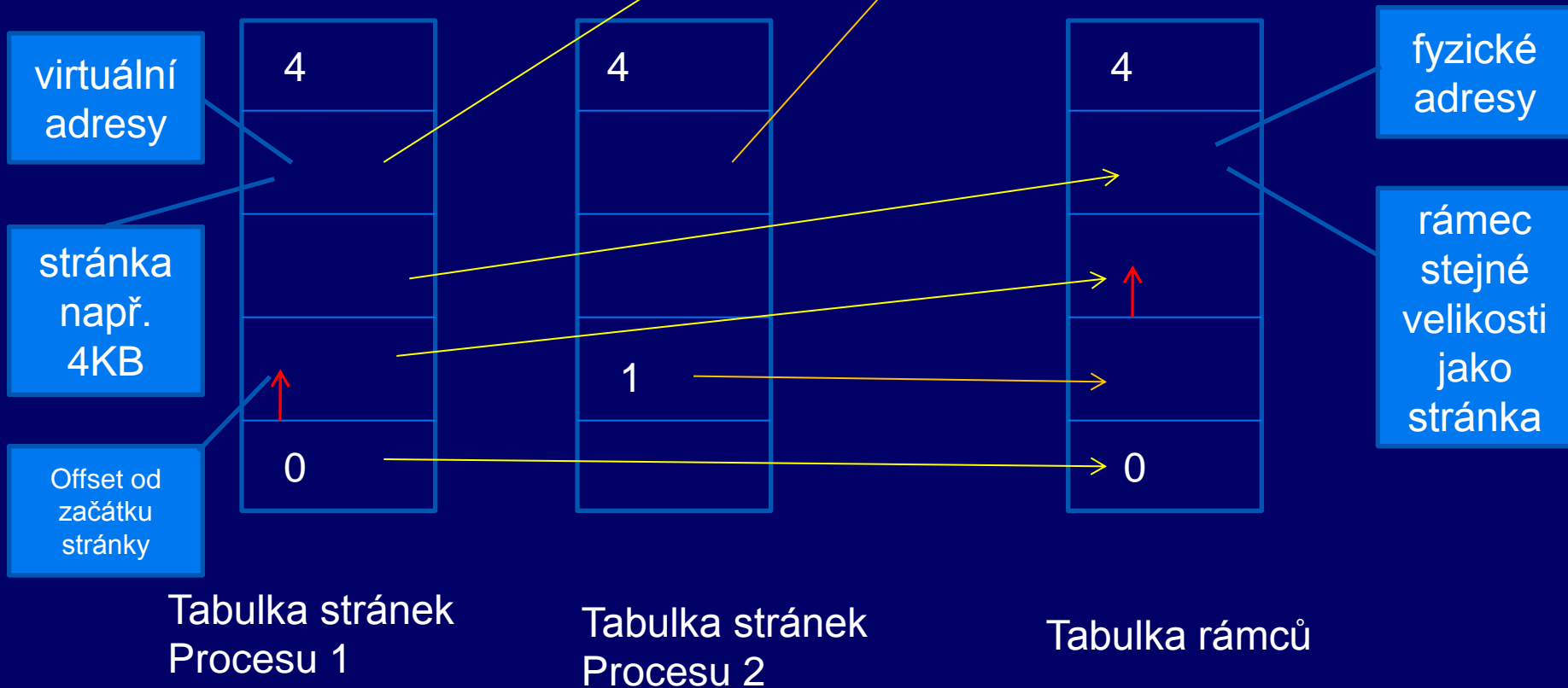
Tabulka stránek
Procesu 2



Tabulka rámců

fyzické adresy

rámec stejné velikosti jako stránka



Tabulka stránek procesu: 1
Velikost stránky: 4096 B

stránka	rámec	další atributy
0	0	
1	2	
2	3	
3	x	swap: 0
4		

Pokud bychom počítali fyzické adresy pro proces 2, používali bychom tabulku stránek procesu 2

Je dána VA 500, vypočítejte fyzickou adresu.
Je dána VA 12300, vypočítejte fyzickou adresu ☺

Je dána VA 4099:
 $4099 / 4096 = 1$, offset 3
Tabulka_stranek_naseho_procesu [1] = 2 .. druhý rámec
 $FA = 2 * 4096 + 3 = 8195$

Výpadek stránky:

Stránka není v operační paměti, ale ve swapu na disku

Tabulka stránek - podrobněji

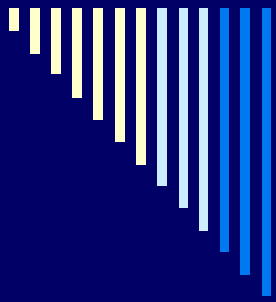
Číslo stránky	Číslo rámce	příznak platnosti	Příznaky ochrany	Bit modifikace (dirty)	Bit referenced	Adresa ve swapu
0	3	valid	rx	1	1	---
1	4	valid	rw	1	1	---
2	---	invalid	ro	0	0	4096

valid
invalid

rw, rx, ro,...

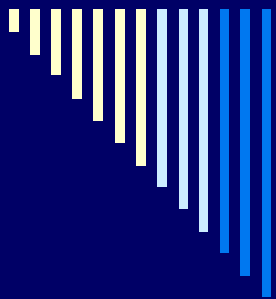
zda je třeba rámec
uložit do swapu při
odstranění z RAM

zda byla stránka
přístupována (čtení či
zápis) v poslední době



Tabulka stránek (TS) - podrobněji

- součástí PCB (tabulka procesů)
 - obsahuje info kde leží jeho tabulka stránek
- velikost záznamu v TS .. 32 bitů
- číslo rámce .. 20 bitů



Výpočet adresy - stránkování

Pojmy:

VA virtuální adresa

FA fyzická adresa

str číslo stránky

offset offset

ramec číslo rámce

Dále předpokládáme velikost stránky 4096B



Příklad s uvedením výpočtu

Je dána $VA(p1) = 100$. Určte FA.

Velikost stránky je 4096 bytů (4KB).

Tabulka stránek procesu p1 je následující:

Číslo stránky	rámec
0	1
1	2
2	---
3	0

Nezapomeň: máme-li více procesů, každý má svojí tabulku stránek.



Výpočet adresy – stránkování

1. Virtuální adresu rozdělíme na číslo stránky a offset
 - **Str** = $VA \div 4096$ (dělení, 4096 je velikost stránky)
 - **Offset** = $VA \bmod 4096$ (zbytek po dělení)
2. Převod pomocí tabulky stránek
převédeme číslo stránky na **číslo rámce**
 - **tab_str[0] = 1** (pro stránku 0 je číslo rámce 1)
 - **tab_str[1] = 2**
 - **tab_str[2] = --** stránka není namapována
 - **tab_str[3] = 0**
 - Pro $VA = 100$ je stránka 0, offset 100 => tedy rámec 1



Výpočet adresy - stránkování

3. Z čísla rámce a offsetu sestavíme fyzickou adresu:

$$FA = \text{ramec} * 4096 + \text{offset}$$

$$FA = 1 * 4096 + 100$$

$$FA = 4196 \text{ v daném případě}$$

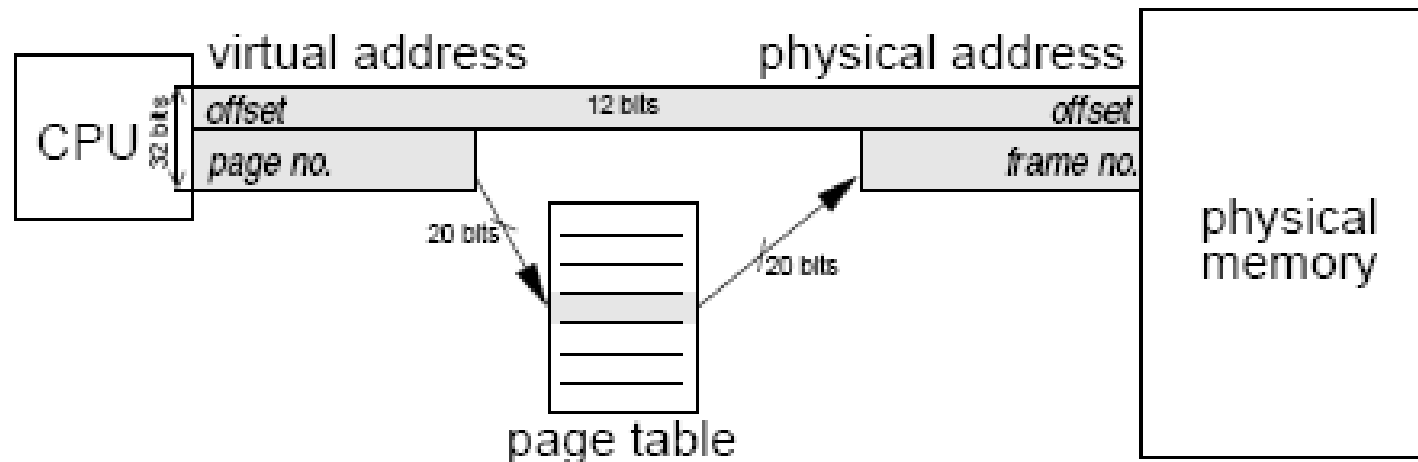
tedy žádné složité dělení není třeba, vezmou se nižší a vyšší bity tj. adresní vodiče

V reálném systému dělení znamená rozdělení na vyšší a nižší bity adresy (proto mocnina dvou velikost str.)

Nižší bity – offset

Vyšší bity – číslo stránky

Stránkování



32 bit adresa – 20 bitů číslo stránky, 12 bitů offset
Offset zůstává beze změny



Výpadek stránky (!!!)

- viz příklad, pro adresu 8192 str 2, offset 0
- Výpadek stránky
 - Stránka není mapována
 - Výpadek stránky způsobí **výjimku**, zachycena OS (pomocí **přerušení**)
 - OS **iniciuje zavádění** stránky a **přepne na jiný** proces
 - Po zavedení stránky OS upraví mapování (tabulku stránek)
 - Proces může pokračovat
 - Vyřešit: KAM stránku zavést a ODKUD ?



Výpadek stránky

Pokud daná stránka procesu není namapována na určitý rámec ve fyzické paměti a chceme k ní přistoupit

dojde k **výpadku stránky** – vyvolání **přerušení** operačního systému.

Operační systém se postará o to, aby danou stránku zavedl do nějakého rámce ve fyzické paměti, nastavil mapování a poté může přístup proběhnout.



Náročnost

□ Velký rozsah tabulky stránek

- Např. 1 milion stránek, ne všechny obsazeny

□ Rychlý přístup

- Nemůžeme pokaždé přistupovat k tabulce stránek
- Různá HW řešení, kopie části tabulky v MMU ...

Tabulka stránek může být velmi rozsáhlá – pro urychlení např. kopie části tabulky stránky v MMU (memory management unit)



Vnější fragmentace

□ Vnější / externí

- Zůstávají nepřidělené (nepřidělitelné) úseky paměti
- Např. dynamické přidělování – malé díry

Při **stránkování vnější fragmentace nenastává**, všechny stránky jsou přidělitelné (jsou stejně velké)



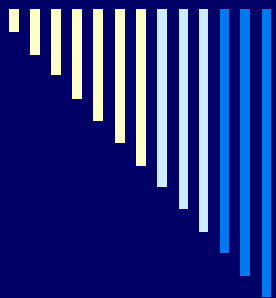
Vnitřní fragmentace

□ Vnitřní fragmentace

- Část **přidělené** oblasti je **nevyužita**
(dostaneme přidělenou stránku, ale využijeme z ní jen část !)

Stránkování:

V průměru polovina poslední stránky procesu je prázdná



Stránkování - poznámky

Čisté stránkování - bez odkládací oblasti (swapu) !!

Souvislý logický adresní prostor procesu
mapován do
nesouvislých částí paměti

OS udržuje:

- 1 tabulka rámců
- Tabulku stránek pro každý proces



Tabulka rámců

- Pro správu FYZICKÉ paměti
- Je třeba informace, které rámce jsou volné vs. obsazené

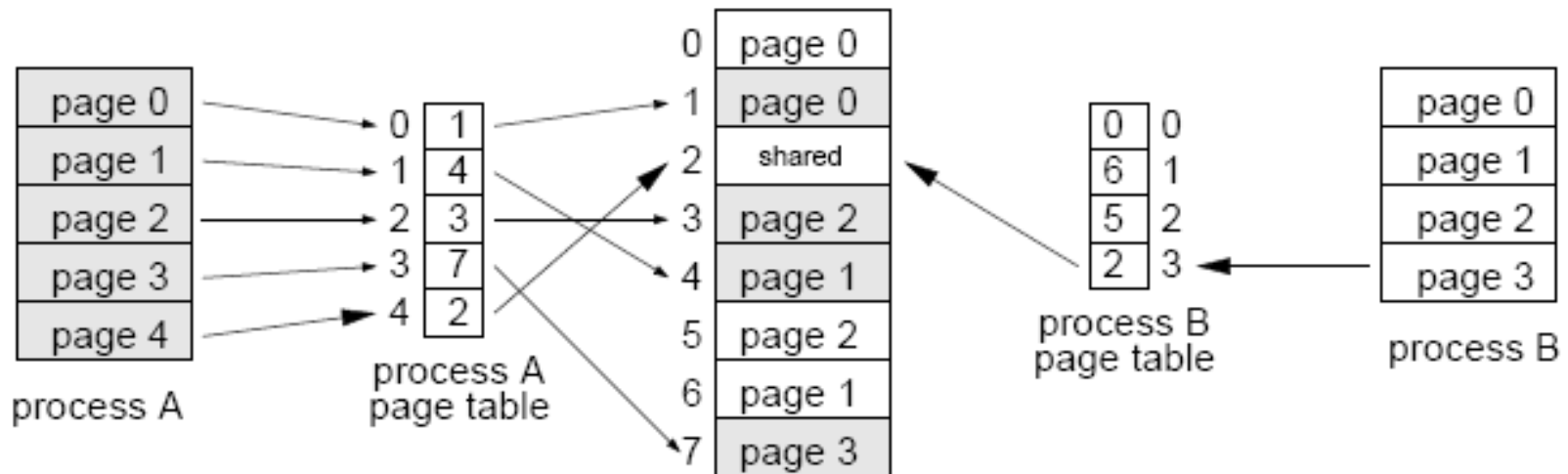




Tabulka stránek procesu

- Mapuje číslo stránky na číslo fyzického rámce
- Další informace – např. příznaky ochrany
- Řeší problém relokace a ochrany
 - Relokace – mapování VA na FA
 - Ochrana – v tabulce stránek **pouze** stránky, ke kterým má proces **přístup**
- Přepnutí na jiný proces
 - MMU přepne na jinou tabulku stránek

Stránkování



Stránkování umožňuje i přístup do **sdílené paměti**, v každém procesu může být dokonce sdílená paměť mapována **od jiné adresy**



Problémy

- Velikost tabulky stránek
 - Pomůže víceúrovňová struktura
- Rychlost převodu VA -> FA
 - TLB (Transaction Look-aside Buffer)

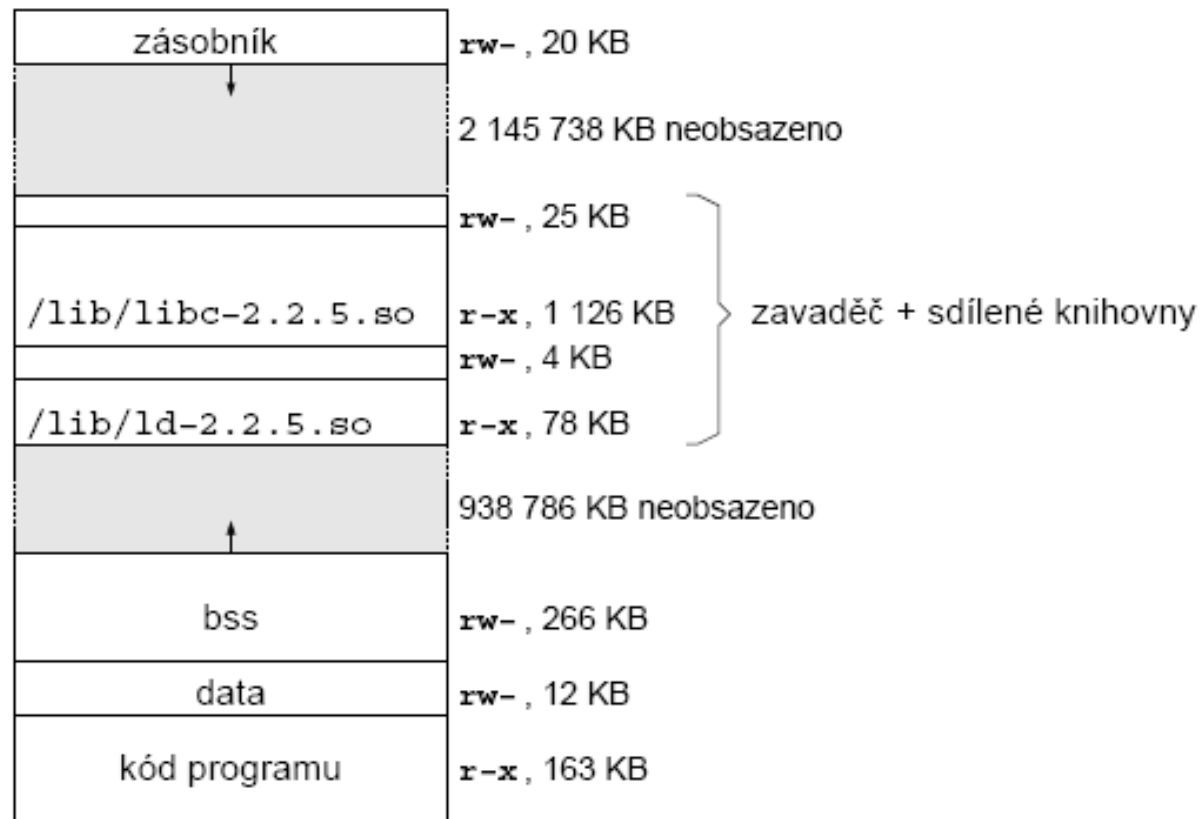
na dalších slidech budou tyto problémy dále rozebrány



Velikost tabulky stránek

- VA 32 bitů
 - stránka 4KB (12 bitů)
 - Stránek 2^{20} (20 bitů)
 - Každá položka 4B .. $2^{20} \cdot 4 = 4\text{MB}$ celkem pro **každý** proces
 - Proces využívá jen část VA
 - Kód
 - Data (inicializovaná, a neinicializovaná)
 - Sdílené knihovny a jejich data
 - Od nejvyšší adresy zásobník - roste dolů
-

Rozdělení paměti pro proces

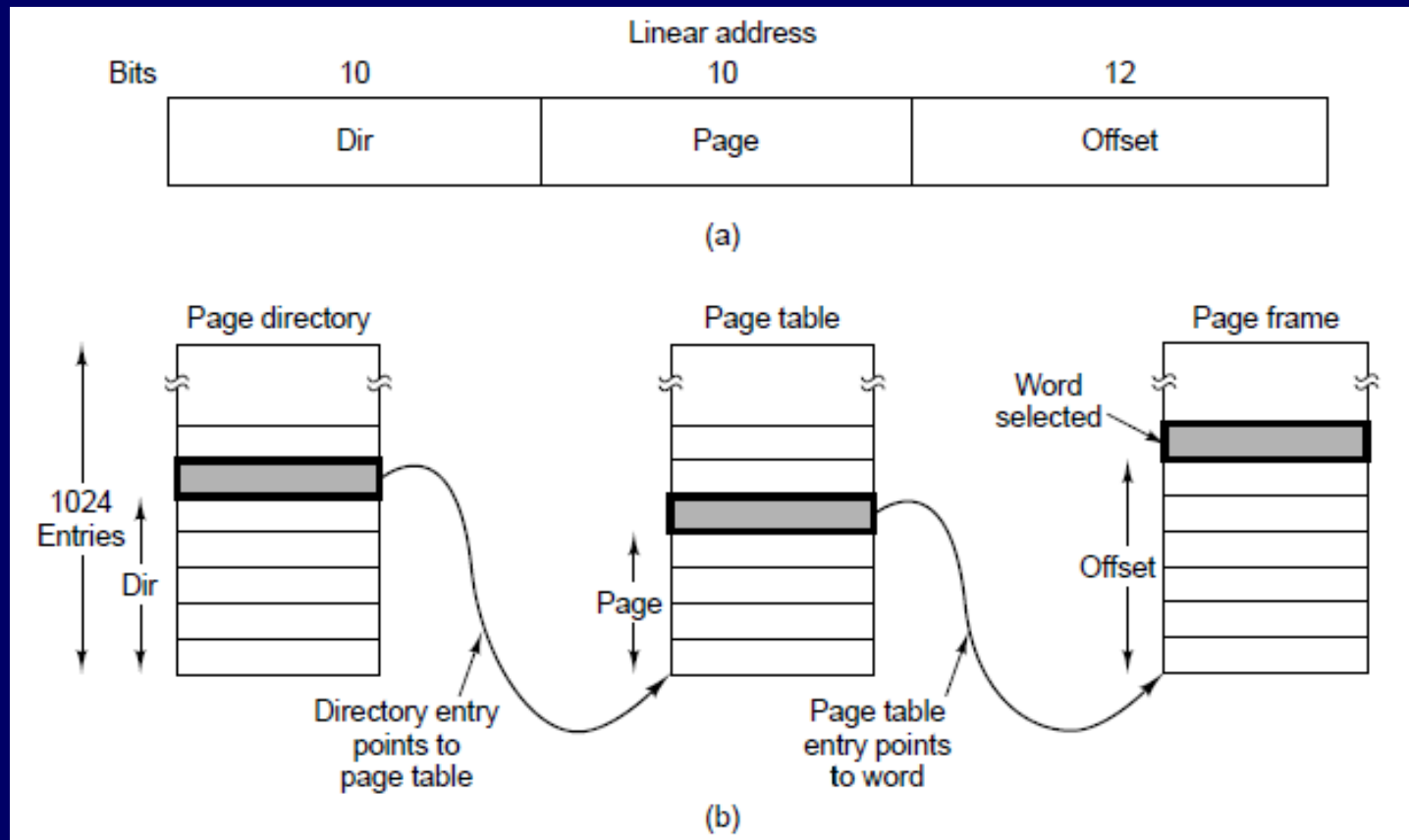




Velikost tabulky stránek

- Mít v tabulce stránek jen ty, představující existující paměť => víceúrovňová tabulka stránek
 - VA 32 bitů
 - PT1 – 10 bitů , index do tab. stránek 1. úrovně
 - PT2 – 10 bitů, index do tab. stránek 2. úrovně
 - Offset – 12bitů
 - PT1=0 (kód a data), PT1=1 (sdílené knihovny)
PT=1023 (zásobník); ostatní nepřirázeno!
-

Velikost tabulky stránek

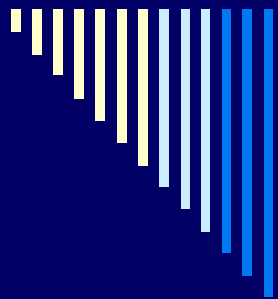




Rychlost převodu (!)

- Každý přístup – sáhne do tabulky stránek
 - 2x více paměťových přístupů
 - musíme sáhnout do tabulky stránek a pak do paměti kam chceme

- TLB (Transaction Look-aside Buffer) (!!!!)
 - HW cache
 - Dosáhneme zpomalení jen 5 až 10 %
 - Přepnutí kontextu na jiný proces
 - problém (vymazání cache,..)
 - než se TLB opět zaplní – pomalý přístup



Obsah položky v tabulce stránek (!!!)

- Číslo rámce
- Příznak platnosti (valid / invalid)
- Příznaky ochrany (rw, ro, ..)
- Bit modified (dirty)
 - zápis do stránky nastaví na 1
- Bit referenced
 - Přístup pro čtení / zápis nastaví na 1
- Další ...



Invertovaná tabulka stránek

- VA 64bitů , stránka 4KB, 2^{52} stránek – moc

Invertovaná tabulka stránek

- Položky pro každý fyzický rámec
 - Omezený počet – dán velikostí RAM
 - VA 64bitů, 4KB stránky, 256MB RAM – 65536 položek
- Forma položky: (id procesu, číslo stránky)

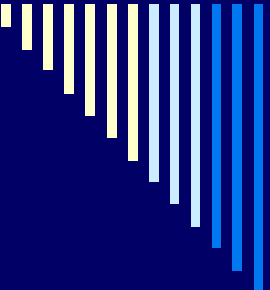


Invertovaná tabulka stránek - převod

- Pokud je položka v TLB
 - zařídí HW, jinak OS (SW)

SW:

- Prohledávání invertované tabulky stránek
 - Položka nalezena – (číslo stránky, číslo rámce) do TLB
 - Tabulka hashovaná podle virtuální adresy (pro optim.)
-



Stránkování na žádost (využívá odkládací prostor)

- Vytvoření procesu
 - Vytvoří prázdnou tabulku stránek
 - Alokace místa na disku pro odkládání stránek
 - Některé implementace – odkládací oblast inicializuje kódem programu a daty ze spustitelného souboru
 - Při běhu
 - Žádná stránka v paměti,
 - 1. přístup – **výpadek stránky (page fault)**
 - OS zavede požadovanou stránku do paměti
 - Postupně v paměti tzv. **pracovní množina** stránek
-

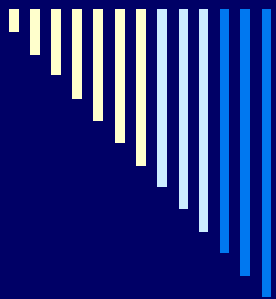


Pracovní množina stránek

Má-li proces svou **pracovní množinu stránek v paměti**,
může pracovat **bez mnoha výpadků**

dokud se pracovní množina stránek **nezmění**, např.
do **další fáze** výpočtu

Pracovní množina stránek daného procesu – kolik stránek
musí mít ve fyzické paměti, aby mohl nějaký čas pracovat bez
výpadků stránky



Ošetření výpadku - scénář

Počítáme v uživatelském režimu, následuje instrukce přečti číslo na adrese 7000

MMU zjistí, že příslušná stránka procesu není v RAM. Vyvolá se přerušení výpadek stránky.

V tabulce stránek má informaci, že stránka leží ve swapu (a kde).



Ošetření výpadku stránky (důležité !)

1. Výpadek – mechanismem **přerušení** (!!) vyvolán OS
 2. OS zjistí, pro kterou stránku nastal výpadek
 3. OS určí umístění stránky na disku
 - Často tato informace přímo v tabulce stránek
 4. Najde rámec, do kterého bude stránka zavedena
 - Co když jsou všechny rámce obsazené?
 5. Načte požadovanou stránku do rámce
 6. Změní odpovídající mapovací položku v tabulce stránek
 7. Návrat..
 8. CPU provede instrukci , která způsobila výpadek
-

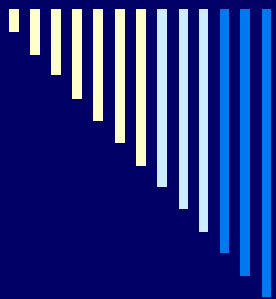


Problém

- Všechny rámce obsazené,
kterou stránku vyhodit ??

Algoritmy nahrazování stránek

Všechny rámce v paměti RAM jsou plné. Přesto musíme nějaký z nich uvolnit (odložit na disk), abychom mohli do RAM dát ten, který potřebujeme. Jak rozhodnout, který rámec vyhodit?



Algoritmy nahrazování stránek

- Uvolnit rámec pro stránku, co s **původní stránkou**?
- Pokud byla stránka modifikována ($\text{dirty}=1$), uložit na disk
- Pokud modifikovaná nebyla a má již stejnou kopii na disku (swap), pouze uvolněna

Algoritmy nahrazování stránek

□ Kterou stránku vyhodit?

Takovou, která se dlouho nebude potřebovat..
Chtělo by křišťálovou kouli...





Algoritmus FIFO

- Udržovat seznam stránek v pořadí, ve kterém byly zavedeny
- Vyhazujeme nejstarší stránku (nejdéle zavedenou – první na seznamu)

Není nejvhodnější

Často používané stránky mohou být v paměti dlouho
(*analogie s obchodem, nejdéle zavedený výrobek – chleba*)

Trpí Beladyho anomálií



Beladyho anomálie

Předpokládáme:

Čím více bude rámců paměti, tím nastane méně výpadků.

Belady našel příklad pro algoritmus FIFO, kdy to neplatí.

* algoritmus FIFO, řetězec odkazů (referencí): 0 1 2 3 0 1 4 0 1 2 3 4

3 rámce: ref.: 0 1 2 3 0 1 4 0 1 2 3 4

```
-----  
1| . 0 1 2 3 0 1 4 4 4 2 3 3  
2| . . 0 1 2 3 0 1 1 1 4 2 2  
3| . . . 0 1 2 3 0 0 0 1 4 4  
-----
```

P P P P P P P P P P = 9 výpadků

4 rámce: ref.: 0 1 2 3 0 1 4 0 1 2 3 4

```
-----  
1| . 0 1 2 3 3 3 4 0 1 2 3 4  
2| . . 0 1 2 2 2 3 4 0 1 2 3  
3| . . . 0 1 1 1 2 3 4 0 1 2  
4| . . . . 0 0 0 1 2 3 4 0 1  
-----
```

P P P P P P P P P P = 10 výpadků

* tj. pro 3 rámce nastane 9 výpadků, pro 4 rámce 10 výpadků

* objev pana Beladyho způsobil vývoj teorie stránkovacích algoritmů a jejich vlastností