ZOS Opakování

L. Pešička, 31.5.2015

Upozornění

- Text se může dopouštět různých zjednodušení, není náhradou učebního textu, jde jen o opakování
- Cílem je ujistit se, že rozumíte SOUVISLOSTEM
- Text je místy neformální, důraz je kladen na pochopení

Proč operační systém?

- **Zjednává pořádek**
  - V systému běží více procesů a každému nemůže povolil vše, co by proces chtěl
- **Usnadňuje práci**
  - Nemusíme sami programovat, jak vypsat na obrazovku řetězec, přehrát zvuk, nebo jak přečíst data ze souboru na disku – v něčem nám pomůžou knihovny (např. glibc) a v něčem OS

OS kontroluje, co procesy smí

- **Zjednává pořádek**
  - V systému běží **více procesů** a každý si nemůže dělat, co ho zrovna napadne – např. vzít libovolný soubor na disku a číst z něj nebo ho dokonce přepsat
  - > procesy nebudou mít přímý přístup k hardwaru
  - přístup k HW bude zprostředkovávat operační systém, a OS se rozhodne, komu operaci povolí a sám ji za proces provede

Rozbor

Z předchozího slidu vznikají 2 otázky:

Jak CPU zabrání, aby se nevykonala instrukce přímého přístupu k hardwaru? (I.)

Jak OS bude vědět, že danému procesu přístup k souboru umožní, a jinému třeba odmítne? (II.)

Otázka 1) CPU

- Procesor (CPU) ví, v kterém režimu se nachází – dle informací v nějakém registru příznaků
- V privilegovaném režimu (běží v něm jádro) povolí všechny instrukce
- V uživatelském režimu **zakáže** vybrané instrukce (např. IN, OUT) – pokud se o ně proces pokusí, se zlou se potáže (výjimka)

Otázka 2) Jak OS rozhoduje?

- Proces P1 je spuštěný uživatelem U1
- Proces P1 chce přistoupit k souboru s1.txt
- Pokud jsou u souboru s1.txt nastavená práva (unixová nebo ACL),může operační systém **ověřit**, zda pro uživatele U1 je přístup povolen či ne (případně ověří pro skupinu) a tedy povolit či zamítnout požadovanou činnost
- Některé filesystémy jako FAT práva nemají, takže rozhodování OS je potom snadné ☺

SOUVISLOSTI !!

- Uvědomte si souvislosti
  - na cvičeních jsme poznali linuxová práva
    chmod 777 s1.txt, chown, …
  - Na přednáškách jsme poznali ACL práva
    více uživatelů, více skupin
    zamítací práva (všichni studenti kromě ..)
    ve formě tabulky
  - Nyní víme, jak práva k souborům využije operační systém pro ověření žádosti procesu, který chce otevřít a zapsat data do souboru

Příklad (!!)

- Na disku /dev/sda je v oblasti /dev/sda1 souborový systém **ext3**
- Je zde soubor **ahoj.txt** a uživatel Tom má práva rw- k tomuto souboru
- Uživatel Tom spustí program prog1, následkem čehož v systému běží proces p1 (má záznam v tabulce procesů)
- Proces p1 chce otevřít soubor **ahoj.txt** pro čtení a zápis, sám to udělat nemůže (přímý přístup k hw není), tak požádá OS o službu pomocí systémového volání
- OS žádost ověří, práva vyhovují, soubor otevře a vrátí procesu identifikátor otevřeného souboru

Poznámka autora

- Dokud Vám není jasný příklad z předchozího slidu, nemá cenu číst dál

Co má chudák CPU práce

- Vykonávání instrukcí vašich procesů není jediná činnost, které se musí CPU věnovat
- V procesu můžete **vyvolat softwarové přerušení** (instrukcí INT, např. INT 0x80 – chcete službu OS)
- V procesu může nastat **výjimka** (dělení nulou, neplatná instrukce, požadovaná stránka paměti není v RAM)
- Může dojít k **hardwarovému přerušení** (tik časovače, někdo stiskne klávesu na klávesnici, síťové kartě přijde rámec, zvukovce dojdou data)

HW přerušení – žádost o pozornost

- Co udělá malé dítě, když chce vaší pozornost?
  Začne brečet

- Co udělá HW periferie, časovač, síťová karta, když potřebuje systému sdělit, že je třeba něco vykonat?
  Vyvolá **hardwarové přerušení**

Jak umí CPU přerušení obsloužit?

- Když začne dítě brečet, víme že mu máme dát najíst nebo pohoupat kočárek.
- Jak má chudák CPU vědět, jak uspokojit žádost zařízení (síťovky, zvukovky,…) ? Díky tabulce vektorů přerušení

- HW přerušení je distribuováno jako signál na nějakém drátu od zařízení k řadiči přerušení

- Podle čísla drátu **řadič přerušení** (integrovaný obvod) určí index do tabulky vektorů přerušení (např. drát 1 index 5, drát 2 index 2 apod.)
- Podobně u SW přerušení - indexem je číslo za instrukcí INT: INT Ox80 je index 128, INT 7 je 7 ☺

Jak umí CPU přerušení obsloužit?

- Tabulka vektorů přerušení obsahuje pro každý index (0-255) **ADRESU** podprogramu (rutiny), který se vykoná pro obsluhu přerušení

- Ať už jsme byli předtím v uživatelském nebo privilegovaném režimu, obsluha probíhá v privilegovaném režimu
- Předtím je potřeba na zásobník uložit informace o přerušeném procesu, abychom se mohli k němu zpět vrátit (kde pokračovat CS:IP, stavy registrů, v jakém režimu jsme byli)

Tabulka vektorů přerušení (v reálném režimu CPU)

- 256 položek (0 až 255)
- Od adresy 0 v paměti
- Každá položka má velikost 4 byty – obsahuje ADRESU obslužné rutiny přerušení
- Velikost této datové struktury je tedy 1KB (256x4B=1024B)
- V složitějším režimu CPU je velikost této struktury 2KB (256 položek po 8 bytech)

SOUVISLOSTI

- Uvědomte si, jak vypadá přeplánování na jiný proces v interaktivních systémech (Linux,Win)
- Pravidelně tiká časovač uvnitř počítače => vyvolá se vždy HW přerušení
- V obsluze přerušení se mimo jiné počítá, kolik tiků už proběhlo
- Když počet tiků x doba_mezi_tiky odpovídá časovému kvantu, systém ví, že se má pustit plánovač, který rozhodne, zda bude pokračovat přerušený proces, či zda poběží někdo jiný

Souvislosti

- Z předchozího plyne:

- Chci-li interaktivní operační systém => HW počítače musí disponovat časovačem, který pravidelně generuje hardwarová přerušení

Přeplánování procesu

- Každý proces má záznam v tabulce procesů (datová struktura v RAM)
- Položka tabulky procesů se nazývá PCB (Process Control Blok)
- Co PCB daného procesu obsahuje?
  - Vše potřebné, aby bylo možné pozastavený proces znovu pustit
  - Říká se tomu kontext procesu

Rozbor

- Pokud je náš proces vykonáván na CPU, má nějaké stavy registrů, otevřené soubory, přidělenou paměť
- Např. EAX = 4, EBX= 2, ECX = 6
- Hodnota CS:IP ukazuje na instrukci, která se bude vykonávat

Rozbor

- Pokud systém náš proces odstaví, je třeba, aby se tyto všechny hodnoty uschovali (EAX, EBX, CS:IP, …)
- Proces co běží po nás bude mít jiné stavy registrů, např. EAX si nastaví na 8
- Poté, co bude pokračovat náš přerušený proces, tak očekává, že v EAX má svojí hodnotu 4

Rozbor

1. Dojde k přeplánování na jiný proces
   - kontext našeho procesu se uschová

2. Spustí se jiný přerušený proces
   - načte se kontext tohoto spouštěného procesu, včetně hodnoty CS:IP a ví se tedy kde pokračovat

Kdy je třeba uchovat stav procesu?

- Proces -> přerušení
  - Je potřeba uschovat registry, včetně CS:IP kde pokračovat
  - Dáme na zásobník
- Proces -> přerušení -> přeplánování
  - Je čas přeplánovat na jiný proces
  - Uložení kontextu našeho procesu do PCB
    - Registry na zásobníku, další data mohou být i jinde
    - Např. přepnutí tabulky stránek, seznam otevřených procesů
    - PCB ví, kde jsou důležitá data „uschována“
  - Načtení kontextu dalšího procesu z PCB

Ukázka, jak může vypadat rutina pro obsluhu přerušení

- http://www-kiv.zcu.cz/~luki/vyuka/zos/stara/cv2/index.html

```
push ax          ; uložení registrů na zásobník
push bx
push cx
push dx
push si
push di
push ds
push es
push bp
mov bp, sp       ; uchování hodnoty SP v BP
sub sp, velikost_oblasti_lokálních_proměnných
                 ; vyhrazení oblasti lokálních proměnných
mov ax, seg data
mov ds, ax       ; do DS odkaz na datový segment
```

VSTUP do obsluhy přerušení

Při výstupu z procedury je obsah všech registrů obnoven a provede se instrukce návratu z přerušení:

```
mov sp, bp
pop bp           ; totéž pro ES, DS, DI, SI, DX, CX
...
pop ax
iret
```

VYSTUP z obsluhy přerušení

Moduly OS

- **správa procesu** (proces, vlákno, přerušení, přeplánovaní procesu, plánování procesů)
- **správa paměti** (stránkovaná, segmentovaná, segmentace se stránkováním, swapovací soubor, PFF, trashing, sdílená paměť - glibc 1x v paměti)
- **správa i/o** (ram prostor, i/o prostor IN a OUT, pozornost zařízeni přerušením, ovladač - v režimu jádra)
- soubory (VFS, volné bloky x obsazené, jak realizovat adresář, FAT, i-uzly, NTFS, žurnálování)
- **bezpečnost** (napříč: uživ/privileg režim, ACL, nesáhnu do cizí paměti, ...)

Vytvoření procesu

int id;

id = **fork**();

if (id == 0) printf(“jsem potomek”);

else printf(“jsem rodič”);

==========

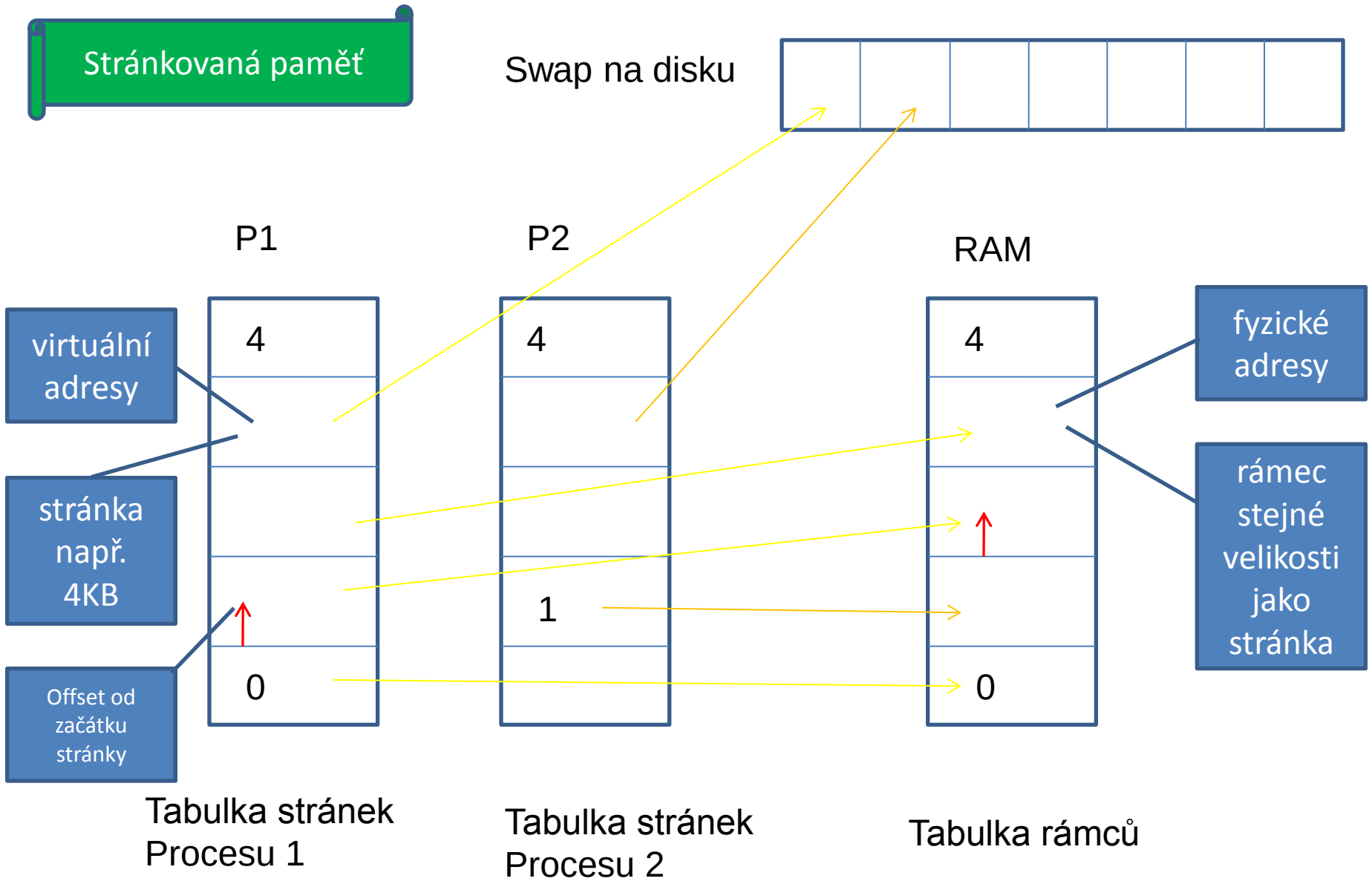
execl();

A jak ho využívá shell

```
#include <stdio.h>
#include <unistd.h>

int main(void) {
 int id;

 id = fork();

if (id == 0) {
        printf("jsem potomek\n");
        execl("/bin/date",  "date",  NULL);
}
 else {
  printf("jsem rodic, potomek ma PID %d\n",id);
  wait();
} }
```

Další možnost na některých platformách:
execl(“/bin/date”, NULL)

úkol:
dejte za execl
printf(“Sem uz nedojdu”);
úkol2: zkuste spustit neexistující program


Stránkovaná paměť
Swap na disku
P1
P2
RAM
virtuální adresy
stránka např. 4KB
Offset od začátku stránky
4
0
4
1
4
0
fyzické adresy
rámec stejné velikosti jako stránka
Tabulka stránek Procesu 1
Tabulka stránek Procesu 2
Tabulka rámců

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 ☺

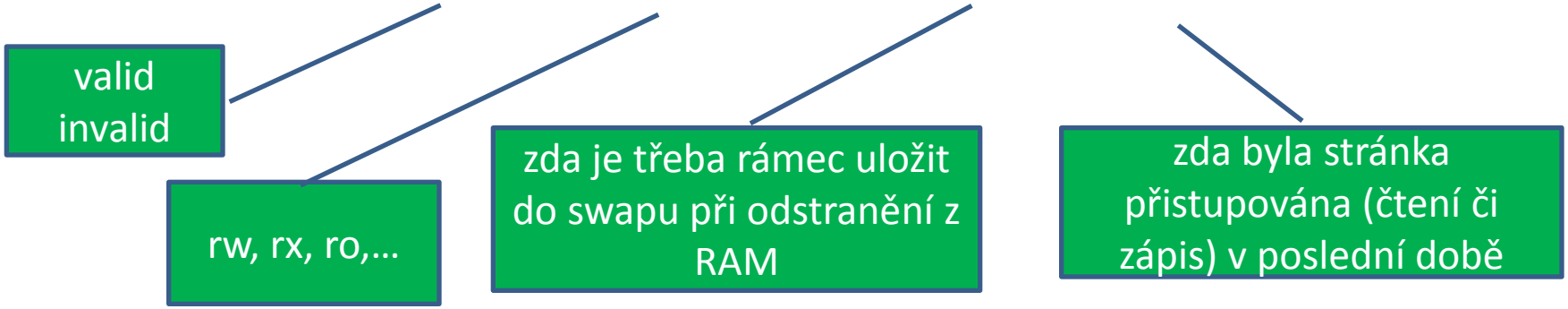
Je dána VA 4099:
4099 / 4096 = 1, offset 3
Tabulka_stranek_naseho_procesu [1] = 2 .. druhý rámec
FA = 2 * 4096 + 3 = 8195

Výpadek stránky:

Stránka není v operační paměti, ale ve swapu na disku

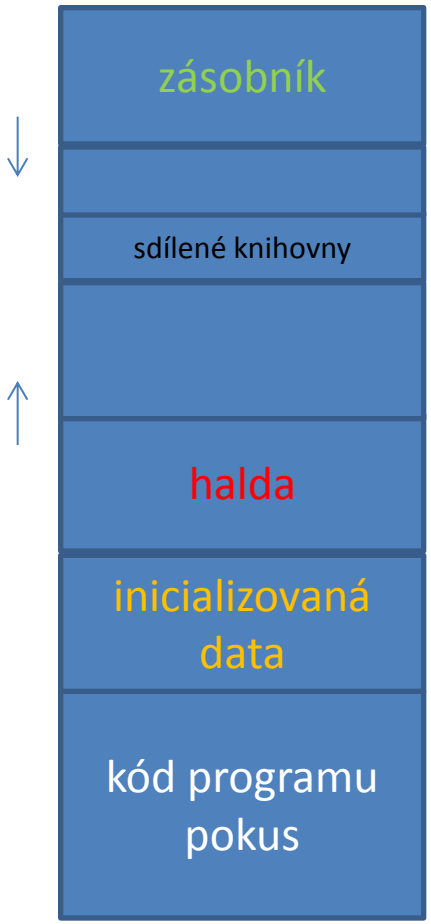
Tabulka stránek - podrobněji

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



Rozdělení paměti pro proces (!!!)

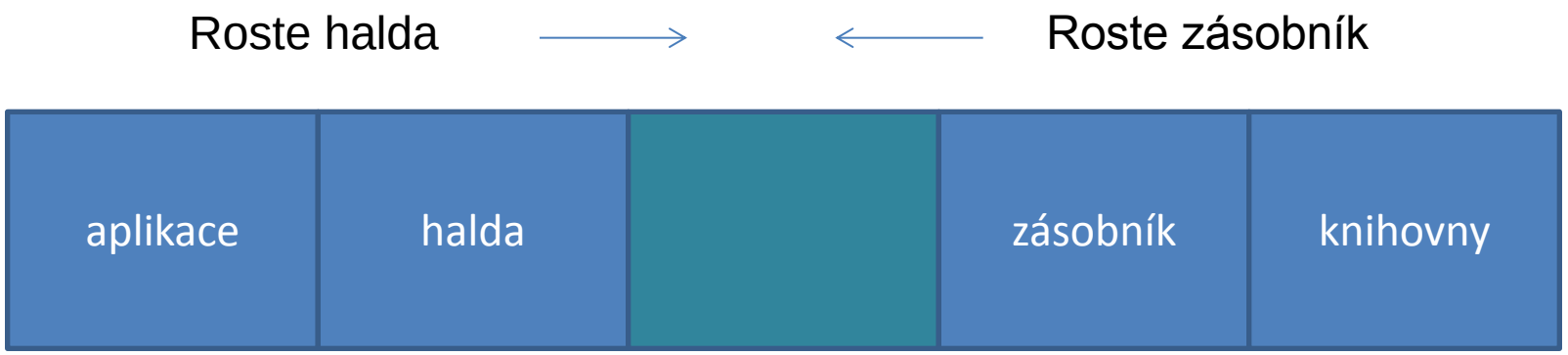
```
pokus.c:

int x =5;  int y = 7;        // inic. data

void fce1() {
    int pom1, pom2; // na zásobníku
     … }

int main (void) {
   …
   malloc(1000);     // halda
   fce1();
   return 0;
}
```



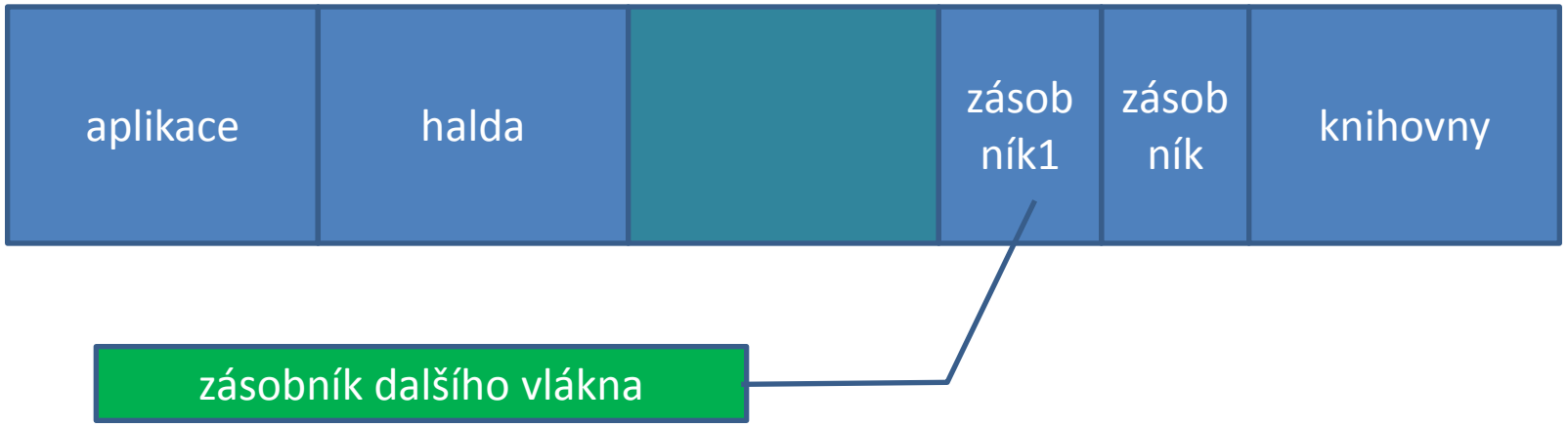
2+2: 0..2GB proces, 2GB..4GB OS

3+1: 3GB proces, 1GB OS

Rozdělení paměti pro proces

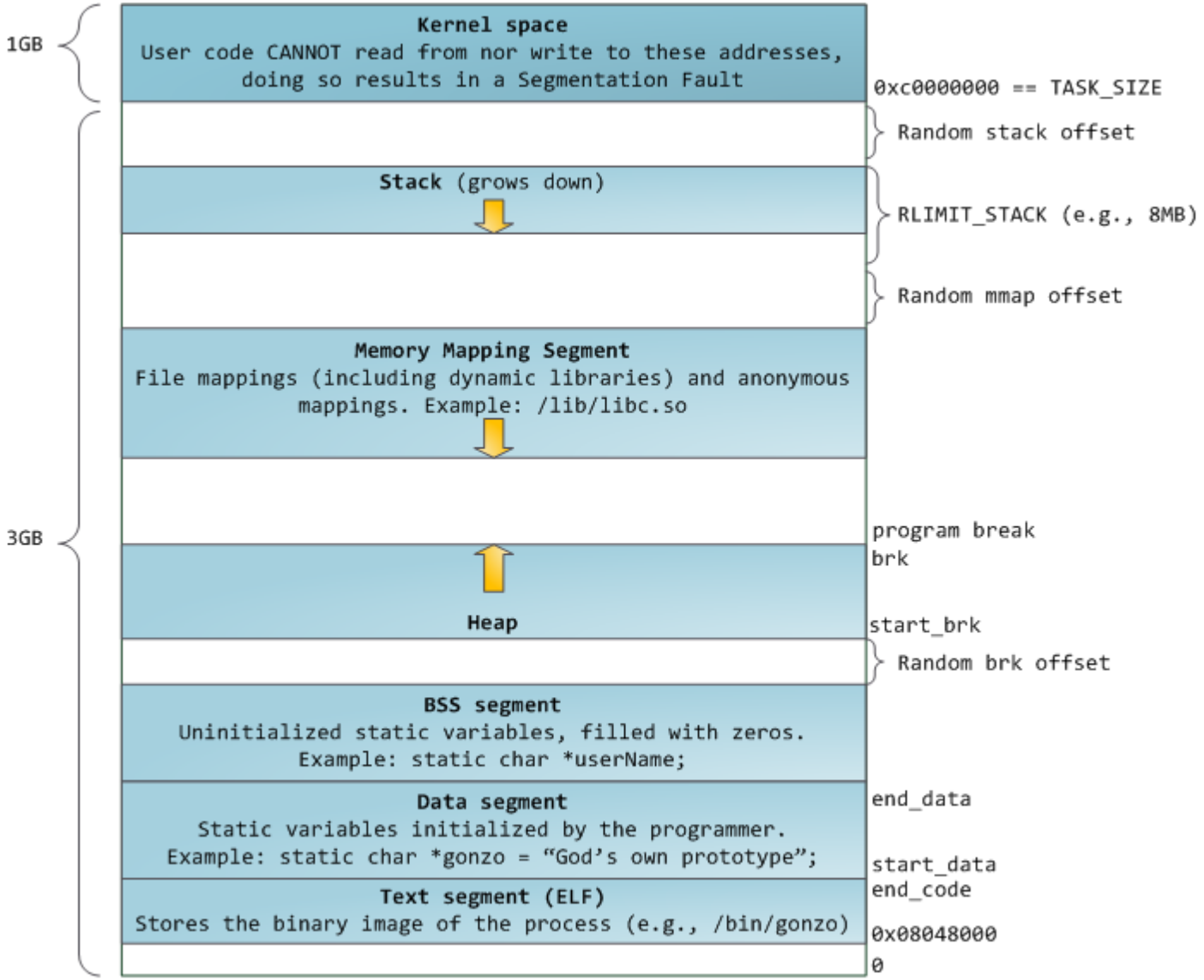


Máme-li více vláken => více zásobníků, limit velikosti zásobníku



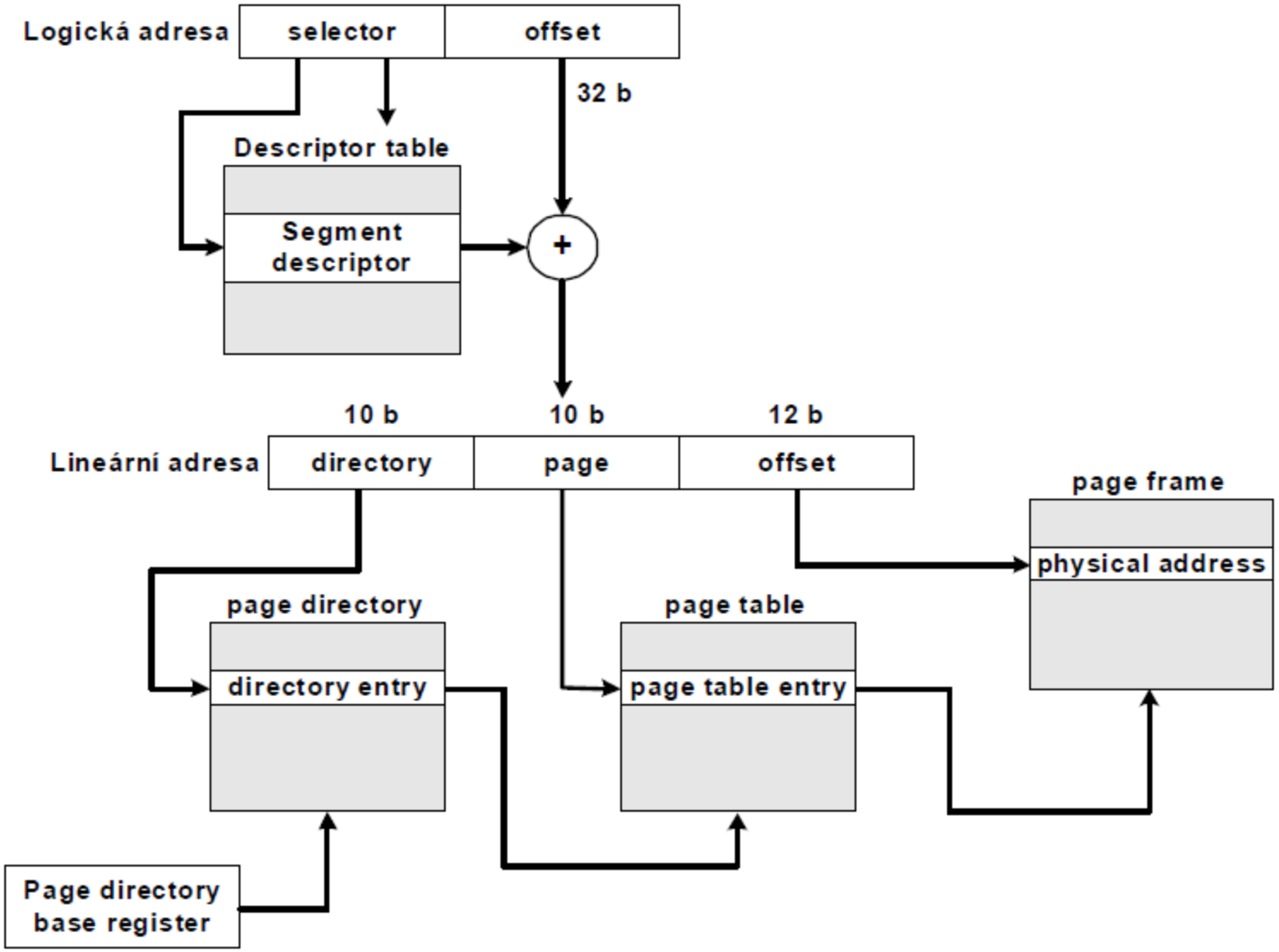
Další ukázka rozdělení paměti

zapamatovat pojem BSS



zdroj: http://duartes.org/gustavo/blog/category/linux DOPORUČUJI !!

Komplexní schéma převodu VA na FA (pamatovat !!!!)



obrázek: J. Lažanský