OS 2018 v. 05

MIT ;)

https://pdos.csail.mit.edu/6.828/2018

Ochrana pamate

- VAP – diagram
- Beziaci proces nahodne zapise na nejaku adresu
  - Co sa stane?
  - Co sa moze stat?
  - Ako tomu zabranit?

Ochrana pamate

- Izolacia procesov (pamate)

- Kazdy proces vlastna pamat
- Kazdy proces moze zapisovat a citat SVOJU pamat
- Ziaden proces nemoze pristupovat k pamati ineho procesu

- Ako zabezpecit multiplex roznych pamati na jednu RAM-ku, pricom tieto pamate NESMU byt medzi sebou pristupne?

Ochrana pamate

- Riesenie: hardver

- Segmentacia
- Strankovanie

- Nas zaujima iba strankovanie (JOS)

Strankovanie

- CPU → MMU → RAM

  VA PA
- Program pouziva IBA virtualne adresy!!!
- Jadro musi povedat MMU, ako prevadzat VA na PA
  - MMU ma na to tabulku (page table)
  - Index do tabulky: va
  - Vystup: pa
- MMU dokaze robit ochranu pristupu do pamate (user, write)

x86 MMU

- 1 stranka = 4096B = 4kB
- Zarovnane, kazda stranka zacina na hranici 4kB

  t.j. adresa_zaciatku_stranky % 4kB == 0
- Teda index do tabulky stranok je prvych 20 bitov

  4kB → 12 bitov

  mame 32 bitovu VA

  takze 32-12 = 20 :)

Polozka tabulky stranok (PTE)

- Prvych 20 bitov je hornych 20 bitov FYZICKEJ adresy

  ozn. “physical page number” → cislo fyzickej stranky v RAM
- MMU meni prvych 20 bitov VA tym, ze ich nahradi hodnotou PPN
- Spodnych 12 bitov PTE su priznaky
  - Present
  - Writable
  - User
  - ...

Tabulka stranok

- Kde sa v pamati nachadza?
- v RAM
  - MMU pracuje s tabulkou
  - OS dokaze do nej tiez pristupovat (R/W)

Tabulka stranok

- Majme 32 bitovy adresny priestor
- Tabulka ma 2^20 poloziek
- Kazda polozka 32 bitov == 4B
- Velkost tabulky: 2^20 * 2^2 = 2^22 = 4MB

- Kazdy program by musel zaberat 4MB RAM pre tabulku!
  - Pre skorsie pocitace to bolo dost vela...
  - miliony poloziek nevyuzitych...

Dvojurovnove strankovanie

- PD – page directory
  - Obsahuje 1024 poloziek PTE
  - Kazda polozka urcuje stranku, kde sa nachadza PT (druha uroven stranok)
- PT – page table
  - Obsahuje 1024 poloziek PTE
  - Kazda polozka urcuje stranku, kde sa nachadza operand (cielove PPN)

Dvojurovnove strankovanie

- Pokryva cely adresny priestor
  - 2^10 poloziek PD
  - 2^10 poloziek kazda PT
  - Spolu 2^10*2^10 = 2^20
- Pre male programy staci 1 PD a niekolko PT
  - 4kB pre PD a 9*4kB pre PT (napriklad)
  - Spolu 40kB pre popis dat
  - Kazda tabulka 4MB, 9x4 = 36MB pre program

Page Directory

- Ako procesor vie, kde v RAM PD sidli?
- Register CR3 drzi FYZICKU adresu PD
- PD drzi fyzicku adresu PTE stranok
  - t.j. jednotlive stranky nemusia ist v pamati za sebou, mozu byt “porozhadzovane” po celej RAM ;)

Priznak 'P'

- Co ked 'P' nie je nastavene?
- Alebo co ked je nastavene 'P' ale 'W' nie, a niekto sa pokusa zapisovat do stranky?
- “page fault” – vypadok stranky
  - CPU ulozi stav registrov, prepne sa do jadra
  - Jadro moze urobit nejaku aktivitu (kill, zaznamena chybu, natiahne stranku a restartuje proces...)

Preco strankovanie a nie segmentacia

- Strankovanie riesi sirsiu mnozinu problemov nez segmentacia

- Spojity virtualny adresny priestor (bez fragmentacie)
- Technika 'copy-on-write' napriklad pri volani fork()
  - Vsetky stranky rodica nastavene na citanie
  - Pokus o zapis generuje vynimku
- Technika 'lazy-allocation' pri poziadavke procesu o pamat
  - Pamat sa “prideli” az pri pristupe k nej
  - Pokus o pristup generuje vynimku

Preco pouzivat strankovanie v jadre?

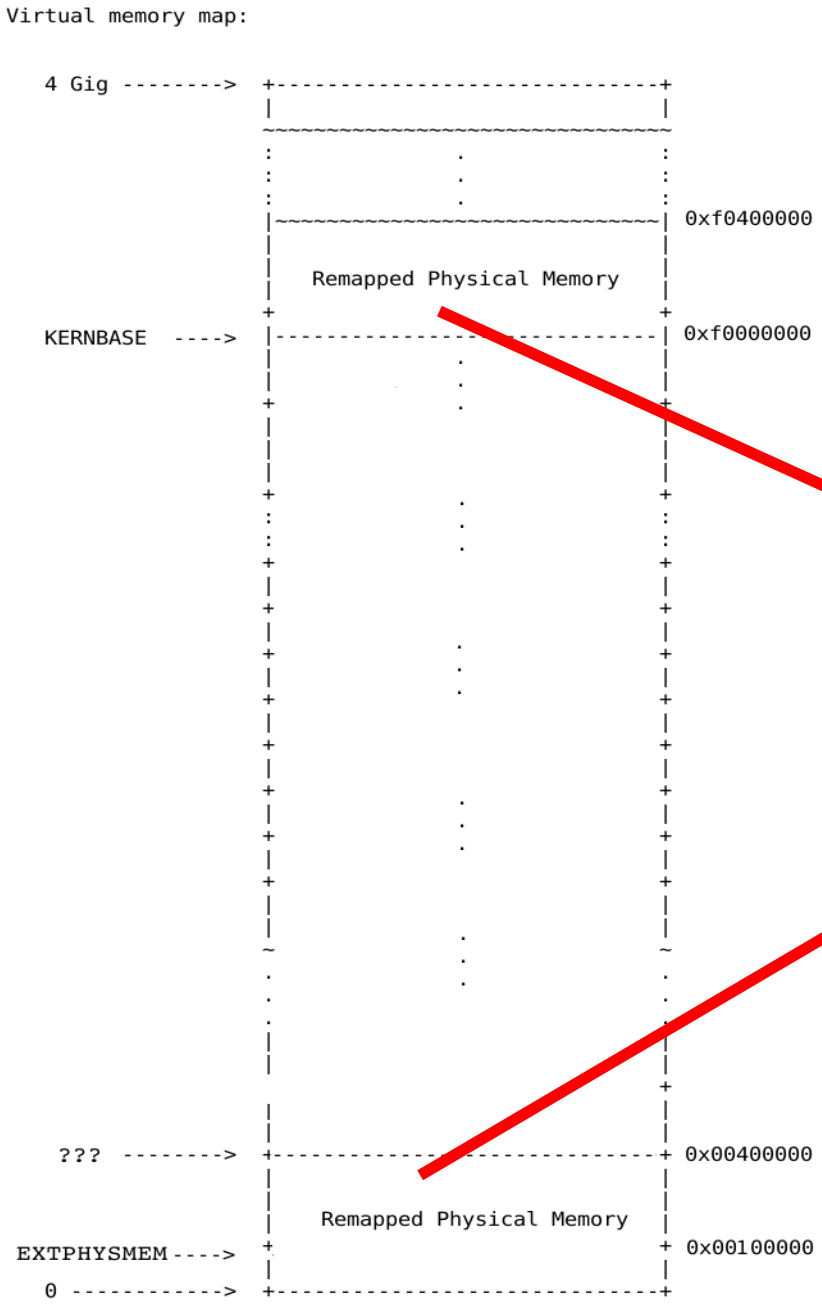
- Jadro ma aj tak pristup vsade, tak naco?
- Inak sa spytajme: moze bezat kernel bez strankovania?

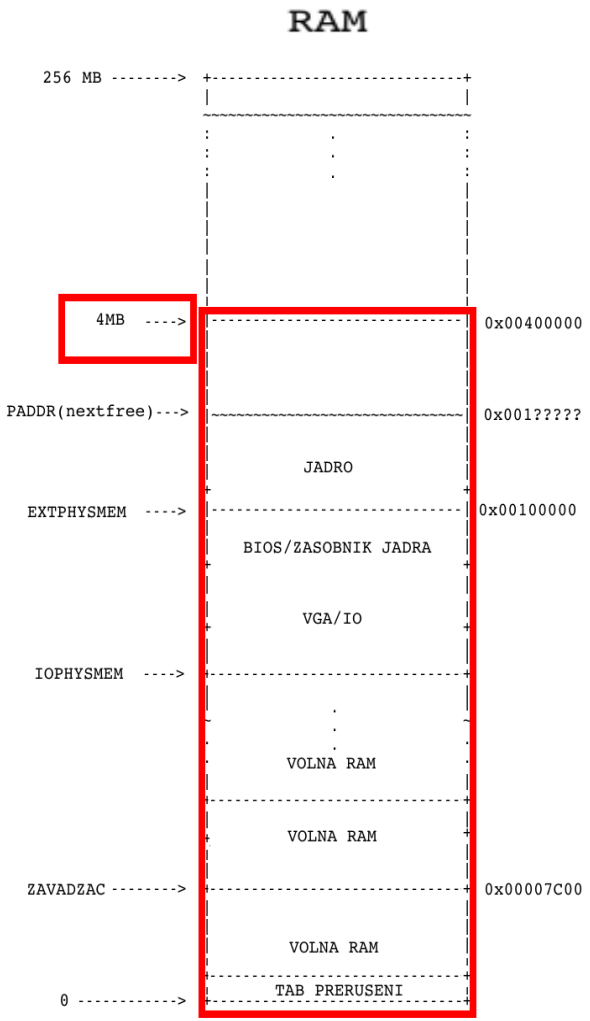
  t.j. Iba na fyzickych adresach?
- Odpoved: MOZE :) (priklad – singularity OS)
- Preco teda strankovanie?
  - Eliminacia fragmentacie pamate
  - OS bezi zvacsa na roznych platformach

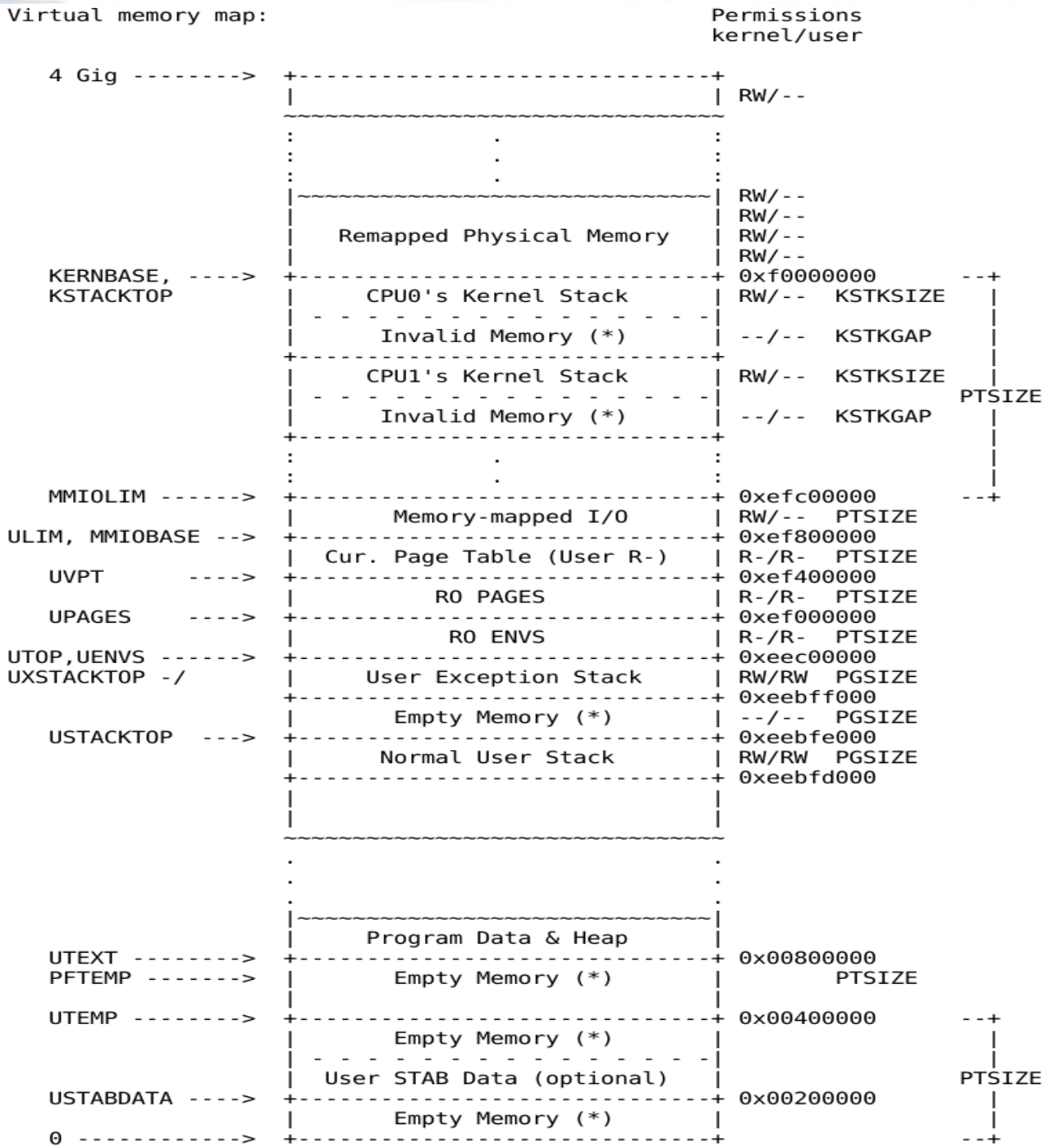
Virtualny adresny priestor procesov

- Kazdy proces ma svoj vlastny VAP
- Ma vlastne tabulky stranok (PD a PT tabulky)
- Kazdy proces ma TO ISTE MAPOVANIE kernel oblasti
- Jadro prepina medzi roznymi virtualnymi adresnymi priestormi zmenou registra CR3 (pri prepinani procesov)

JOS pri mem_init()






Virtual memory map:
Permissions
kernel/user
4 Gig
RW/--
Remapped Physical Memory
RW/--
KERNBASE,
KSTACKTOP
0xf0000000
CPU0's Kernel Stack
RW/--
KSTKSIZE
Invalid Memory (*)
--/--
KSTKGAP
CPU1's Kernel Stack
RW/--
KSTKSIZE
Invalid Memory (*)
--/--
KSTKGAP
PTSIZE
MMIOLIM
0xefc00000
Memory-mapped I/O
RW/--
PTSIZE
ULIM, MMIOBASE
0xef800000
Cur. Page Table (User R-)
R-/R-
PTSIZE
UVPT
0xef400000
RO PAGES
R-/R-
PTSIZE
UPAGES
0xef000000
RO ENVS
R-/R-
PTSIZE
UTOP,UENVS
0xeec00000
UXSTACKTOP -/
User Exception Stack
RW/RW
PGSIZE
0xeebff000
Empty Memory (*)
--/--
PGSIZE
USTACKTOP
0xeebfe000
Normal User Stack
RW/RW
PGSIZE
0xeebfd000
Program Data & Heap
UTEXT
0x00800000
PFTEMP
Empty Memory (*)
PTSIZE
UTEMP
0x00400000
Empty Memory (*)
User STAB Data (optional)
PTSIZE
USTABDATA
0x00200000
Empty Memory (*)
0

Co ziskame pomocou VAP

- Suvisly adresny priestor pre procesy
- Mapovanie spolocnych datovych struktur: lahke prepnutie pri sys volani alebo preruseni
- Kedze je kernel mapovany v kazdom user VAP rovnako, lahko sa prepina medzi procesmi
- Jadro ma lahky pristup k user VAP, napriklad ked ma prevziat parametre sys volania
- Jadro ma lahky pristup do RAM
  - PA(x) → x+KERN_BASE
  - X vo VAP → X-KERN_BASE vo PAP

JOS

- (15r, 10 m) pte_t* **pgdir_walk**(pgdir, va, create)
- (8r, 6m) void **boot_map_region**(pgdir, va, size, pa, perm)
- (10r, 14m) int **page_insert**(pgdir, pp, va, perm)
- (10r, 5m) struct PageInfo* **page_lookup**(pgdir, va, pte_store)
- (10r, 10m) void **page_remove**(pgdir, va)

- (5r, 10m) doplnenie **mem_init**()

- Oprava chyb 15m

walkdir(va)

- Vrat adresy prislusneho PTE pre danu VA
- Simuluje pracu MMU!!!!

- Makro PDX(va) extrahuje hornych 10 bitov
- Makro PTX(va) extrahuje druhych 10 bitov
- Makro PTE_ADDR(va) vrati co?

- PTE_P, PTE_W, PTE_U, ...
- PGNUM(x), PGOFF(x), PGADDR(d, t, o), ...

- KADDR(x), PADDR(x), page2pa(x), page2kva(x), pa2page(x)

walkdir(va) - algoritmus

- pde = &pgdir[PDX(va)] je adresa PDE (teda v PD)
- Ak *pde je NULL
  - Ak create je false, return NULL
  - Alokuj tab. stranok (PTE table) (page_alloc)
  - Do *pde vloz PA stranky (page2pa!!!) + flags
- Teraz prislusne PTE bude v stranke, na ktoru ukazuje *pde, na offsete PTX(va)
  - PTE_ADDR extrahuje PA z *pde
  - KADDR premeni PA na adresy jadra ;)