# OS MMXX

MIT ;)

https://pdos.csail.mit.edu/6.828/2020

# Vypadky stranok

# Tema

# Tema

- Implementacia roznych trikov pomocou VM

# Tema

- Implementacia roznych trikov pomocou VM
- Specialne zameranie na vypadky stranok

# Tema

- Implementacia roznych trikov pomocou VM
- Specialne zameranie na vypadky stranok
  - Oneskorena (leniva) alokacia – Lazy allocation
  - Fork COW (copy-on-write)
  - Mapovanie suboru do pamate – Memory mapping

# Tema

- Naco su tieto triky dobre

# Tema

- Naco su tieto triky dobre

- Zlepsenie vykonu/efektivity
  - Oneskorena alokacia
  - Jedna nulova stranka pre cely system
  - COW fork

# Tema

- Naco su tieto triky dobre

- Zlepsenie vykonu/efektivity
  - Oneskorena alokacia
  - Jedna nulova stranka pre cely system
  - COW fork
- Nova funkcionalita systemu
  - Mapovanie pamate (mmap)

# Tema

- Naco su tieto triky dobre

- Zlepsenie vykonu/efektivity
  - **Oneskorena alokacia** (lab tento tyzden)
  - Jedna nulova stranka pre cely system
  - **COW fork** (dalsi lab)
- Nova funkcionalita systemu
  - Mapovanie pamate (mmap)

# Reakcia xv6 na vypadok stranky

- Ako reaguje xv6 na vypadky stranok

# Reakcia xv6 na vypadok stranky

- Ako reaguje xv6 na vypadky stranok

- usertrap(): unexpected scause …
- Vid priklad user/ls.c

# Reakcia xv6 na vypadok stranky

- Ako reaguje xv6 na vypadky stranok

- usertrap(): unexpected scause …
- Vid priklad user/ls.c
- Ukoncenie procesu

# Reakcia xv6 na vypadok stranky

- Ako reaguje xv6 na vypadky stranok

- usertrap(): unexpected scause …
- Vid priklad user/ls.c
- Ukoncenie procesu

- Co vypadok stranky v kerneli?
- Vid priklad kernel/trap.c

# Opakovanie

- Vyhody VM

# Opakovanie

- Vyhody VM

1. Izolacia – kazdy proces ma svoj vlastny adresny priestor

# Opakovanie

- Vyhody VM

1. Izolacia – kazdy proces ma svoj vlastny adresny priestor
2. Nepriamociarost (level-of-indirection) – preklad VA na FA umoznuje triky
   - Zdielanie dat v ramke (trampolina)
   - Strazenie pretecenia zasobnika (guard page)
   - …

# Staticke / dynamicke mapovanie

- Jadro OS ma kontrolu nad prekladom VA→FA

# Staticke / dynamicke mapovanie

- Jadro OS ma kontrolu nad prekladom VA→FA

- Doteraz sme sa venovali pevne nastavenemu mapovaniu (v zmysle ze mapovanie musi byt nastavene PRED pristupom k VA)

# Staticke / dynamicke mapovanie

- Jadro OS ma kontrolu nad prekladom VA→FA

- Doteraz sme sa venovali pevne nastavenemu mapovaniu (v zmysle ze mapovanie musi byt nastavene PRED pristupom k VA)
- VM vsak umoznuje aj mapovanie “za chodu”

# Staticke / dynamicke mapovanie

- Jadro OS ma kontrolu nad prekladom VA→FA

- Doteraz sme sa venovali pevne nastavenemu mapovaniu (v zmysle ze mapovanie musi byt nastavene PRED pristupom k VA)
- VM vsak umoznuje aj mapovanie “za chodu”
  - Pri pokuse o pristup na VA sa generuje vynimka

# Staticke / dynamicke mapovanie

- Jadro OS ma kontrolu nad prekladom VA→FA

- Doteraz sme sa venovali pevne nastavenemu mapovaniu (v zmysle ze mapovanie musi byt nastavene PRED pristupom k VA)
- VM vsak umoznuje aj mapovanie “za chodu”
  - Pri pokuse o pristup na VA sa generuje vynimka
  - Jadro OS pre danu VA vytvori mapovanie a restartuje proces od instrukcie, ktora sposobila vynimku

# Staticke / dynamicke mapovanie

- Jadro OS ma kontrolu nad prekladom VA→FA

- Doteraz sme sa venovali pevne nastavenemu mapovaniu (v zmysle ze mapovanie musi byt nastavene PRED pristupom k VA)
- VM vsak umoznuje aj mapovanie “za chodu”
  - Pri pokuse o pristup na VA sa generuje vynimka
  - Jadro OS pre danu VA vytvori mapovanie a restartuje proces od instrukcie, ktora sposobila vynimku
  - Proces platne pristupi k udajom na VA

# Terminologia RISC-V

- SiFive Interrupt Cookbook

- https://sifive.cdn.prismic.io/sifive/0d163928-2128-42be-a75a-464df65e04e0_sifive-interrupt-cookbook.pdf

# Terminologia RISC-V

- Exception (vynimka)
- Trap (presun riadenia)
- Interrupt (prerusenie)

# Terminologia RISC-V

- Exception (vynimka) – vynimocny stav vyvolany instrukciou vykonavaneho programu
- Trap (presun riadenia) – sychronny prenos riadenia do kodu obsluhy sposobeny vynimocnym stavom, ktory zapricinil vykonavany program
- Interrupt (prerusenie) – externa udalost, ktora sa vyskytne asynchronne voci vykonavanemu kodu

Prevzate z https://sifive.cdn.prismic.io/sifive/0d163928-2128-42be-a75a-464df65e04e0_sifive-interrupt-cookbook.pdf

# Machine Cause (mcause) register

| Interrupt | Code | Description |
|---|---|---|
| 1 | 0 | User software interrupt |
| 1 | 1 | Supervisor software interrupt |
| 1 | 2 | Reserved |
| 1 | 3 | Machine software interrupt |
| 1 | 4 | User timer interrupt |
| 1 | 5 | Supervisor timer interrupt |
| 1 | 6 | Reserved |
| 1 | 7 | Machine timer interrupt |
| 1 | 8 | User external interrupt |
| 1 | 9 | Supervisor external interrupt |
| 1 | 10 | Reserved |
| 1 | 11 | Machine external interrupt |
| 1 | >=12 && <16 | Reserved |
| 1 | >=16 | Implementation defined local interrupts |
| 0 | 0 | Instruction address misaligned |
| 0 | 1 | Instruction access fault |
| 0 | 2 | Illegal instruction |
| 0 | 3 | Breakpoint |
| 0 | 4 | Load address misaligned |
| 0 | 5 | Load access fault |
| 0 | 6 | Store/AMO address misaligned |
| 0 | 7 | Store/AMO access fault |
| 0 | 8 | Environment call from U-mode |
| 0 | 9 | Environment call from S-mode |
| 0 | 10 | Reserved |
| 0 | 11 | Environment call from M-mode |
| 0 | 12 | Instruction page fault |
| 0 | 13 | Load page fault |
| 0 | 14 | Reserved |
| 0 | 15 | Store/AMO page fault |
| 0 | >= 16 | Reserved |

**Table 1:** Machine Cause (`mcause`) Register

# Ciel tohto tyzdna

- Pri vypadku stranky v uzivatelskom programe

# Ciel tohto tyzdna

- Pri vypadku stranky v uzivatelskom programe
- Zistit, ci sa jedna o legitimnu adresu programu

# Ciel tohto tyzdna

- Pri vypadku stranky v uzivatelskom programe

- Zistit, ci sa jedna o legitimnu adresu programu
- Ak ano, aktualizovat tabulku stranok procesu

# Ciel tohto tyzdna

- Pri vypadku stranky v uzivatelskom programe

- Zistit, ci sa jedna o legitimnu adresu programu
- Ak ano, aktualizovat tabulku stranok procesu
- A restartovat proces od instrukcie, ktora sposobila vypadok stranky

# Co potrebujeme

# Co potrebujeme

1. VA, ktora sposobila vypadok (faulting VA)

# Co potrebujeme

1. VA, ktora sposobila vypadok (faulting VA) (na RISC-V je tato hodnota ulozena v reg $stval)

# Co potrebujeme

1. VA, ktora sposobila vypadok (faulting VA) (na RISC-V je tato hodnota ulozena v reg $stval)

2. Typ vypadku stranky

# Co potrebujeme

1. VA, ktora sposobila vypadok (faulting VA) (na RISC-V je tato hodnota ulozena v reg $stval)

2. Typ vypadku stranky (pre RISC-V vid tabulka 4.2 v “RISC-V privileged.pdf”, hodnota registra $scause – read, write, instruction)

Prevzate z https://sifive.cdn.prismic.io/sifive/0d163928-2128-42be-a75a-464df65e04e0_sifive-interrupt-cookbook.pdf

# Machine Cause (mcause) register

| Interrupt | Code | Description |
|---|---|---|
| 1 | 0 | User software interrupt |
| 1 | 1 | Supervisor software interrupt |
| 1 | 2 | Reserved |
| 1 | 3 | Machine software interrupt |
| 1 | 4 | User timer interrupt |
| 1 | 5 | Supervisor timer interrupt |
| 1 | 6 | Reserved |
| 1 | 7 | Machine timer interrupt |
| 1 | 8 | User external interrupt |
| 1 | 9 | Supervisor external interrupt |
| 1 | 10 | Reserved |
| 1 | 11 | Machine external interrupt |
| 1 | >=12 && <16 | Reserved |
| 1 | >=16 | Implementation defined local interrupts |
| 0 | 0 | Instruction address misaligned |
| 0 | 1 | Instruction access fault |
| 0 | 2 | Illegal instruction |
| 0 | 3 | Breakpoint |
| 0 | 4 | Load address misaligned |
| 0 | 5 | Load access fault |
| 0 | 6 | Store/AMO address misaligned |
| 0 | 7 | Store/AMO access fault |
| 0 | 8 | Environment call from U-mode |
| 0 | 9 | Environment call from S-mode |
| 0 | 10 | Reserved |
| 0 | 11 | Environment call from M-mode |
| 0 | 12 | Instruction page fault |
| 0 | 13 | Load page fault |
| 0 | 14 | Reserved |
| 0 | 15 | Store/AMO page fault |
| 0 | >= 16 | Reserved |

**Table 1:** Machine Cause (`mcause`) Register

# Co potrebujeme

1. VA, ktora sposobila vypadok (faulting VA) (na RISC-V je tato hodnota ulozena v reg $stval)

2. Typ vypadku stranky (pre RISC-V vid tabulka 4.2 v “RISC-V privileged.pdf”, hodnota registra $scause – read, write, instruction)

3. Mod CPU a instrukciu, kde k vypadku prislo

# Co potrebujeme

1. VA, ktora sposobila vypadok (faulting VA) (na RISC-V je tato hodnota ulozena v reg $stval)

2. Typ vypadku stranky (pre RISC-V vid tabulka 4.2 v “RISC-V privileged.pdf”, hodnota registra $scause – read, write, instruction)

3. Mod CPU a instrukciu, kde k vypadku prislo
   - U/S mod: implicitne sa vyvola usertrap/kerneltrap
   - User PC: $sepc, hodnota ulozena v tf→epc

# Zdroj informacii

- pre RISC-V vid tabulka 4.2 v “RISC-V privileged.pdf”

- https://uim.fei.stuba.sk/predmet/b-os/
- tab Literatura

- https://uim.fei.stuba.sk/wp-content/uploads/2018/02/riscv-privileged.pdf

# Co potrebujeme

- Naco nam treba register PC?

# Co potrebujeme

- Naco nam treba register PC?

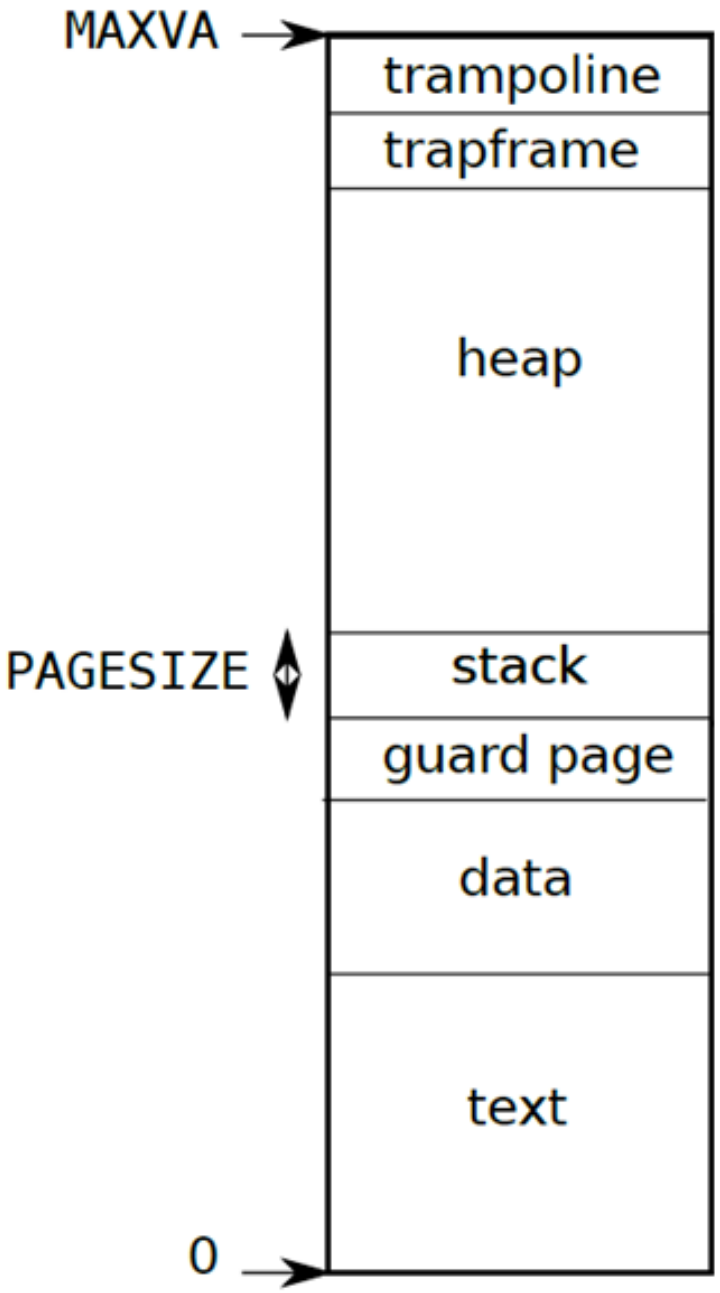
- Aby kod jadra vedel restartovat uzivatelsky program presne od tej instrukcie, ktora vyvolala vypadok
- Instrukcia sa musi zopakovat (jej vykonanie)
- Ak jadro dobre vsetko nastavilo, pri opakovani instrukcie sa uz vynimka neobjavi

# Zoznam trikov s VM

# Zoznam trikov s VM

- alokacia pamate na ziadost (lazy/on-demand)
- jedna stranka nul (zero-filled page)
- COW fork (copy-on-write)
- strankovanie na ziadost (on-demand ver. 2)
- VA vacsia nez RAM (swap)
- mapovanie suborov do pamate (mmap)
- zdielanie pamate medzi procesmi (shared)

# Lazy alokacia (pre sbrk)



# Lazy alokacia (pre sbrk)

- sbrk() v xv6 je primitivny – vzdy procesu da, co chce

# Lazy alokacia (pre sbrk)

- sbrk() v xv6 je primitivny – vzdy procesu da, co chce
- User aplikacia casto nevyuzije vsetku pridelenu pamat
  - Napr. alokuje buffer na nacitanie vstupu, ale vacsinou sa vyuzije iba prvych par bajtov

# Lazy alokacia (pre sbrk)

- sbrk() v xv6 je primitivny – vzdy procesu da, co chce
- User aplikacia casto nevyuzije vsetku pridelenu pamat
  - Napr. alokuje buffer na nacitanie vstupu, ale vacsinou sa vyuzije iba prvych par bajtov
- sbrk() tak alokuje pamat, ktora sa NIKDY v procese nevyuzije, ale nikto iny ju nemoze vyuzivat!

# Lazy alokacia (pre sbrk)

- Moderne OS alokuju pamat oneskorene (lenivo, lazy)
- Ako?

# Lazy alokacia (pre sbrk)

- Moderne OS alokuju pamat oneskorene (lenivo, lazy)
- Ako:
  - Fyzicka ram sa alokuje az vtedy, ked je treba

# Lazy alokacia (pre sbrk)

- Moderne OS alokuju pamat oneskorene (lenivo, lazy)
- Ako:
  - Fyzicka ram sa alokuje az vtedy, ked je treba
  - Pri volani sbrk() sa zvacsi p->sz, ale nerobi sa ziadna alokacia (mapovanie)

# Lazy alokacia (pre sbrk)

- Moderne OS alokuju pamat oneskorene (lenivo, lazy)
- Ako:
  - Fyzicka ram sa alokuje az vtedy, ked je treba
  - Pri volani sbrk() sa zvacsi p->sz, ale nerobi sa ziadna alokacia (mapovanie)
  - Ked sa proces pokusi pristupit k “alokovanej” pamati, nastane vypadok stranky!!!

# Lazy alokacia (pre sbrk)

- Moderne OS alokuju pamat oneskorene (lenivo, lazy)
- Ako:
  - Fyzicka ram sa alokuje az vtedy, ked je treba
  - Pri volani sbrk() sa zvacsi p->sz, ale nerobi sa ziadna alokacia (mapovanie)
  - Ked sa proces pokusi pristupit k “alokovanej” pamati, nastane vypadok stranky!!!
  - Jadro v obsluhe vypadku alokuje a namapuje potrebnu pamat

# Lazy alokacia (pre sbrk)

- Moderne OS alokuju pamat oneskorene (lenivo, lazy)
- Ako:
  - Fyzicka ram sa alokuje az vtedy, ked je treba
  - Pri volani sbrk() sa zvacsi p->sz, ale nerobi sa ziadna alokacia (mapovanie)
  - Ked sa proces pokusi pristupit k “alokovanej” pamati, nastane vypadok stranky!!!
  - Jadro v obsluhe vypadku alokuje a namapuje potrebnu pamat
  - User prorgam sa restartuje od miesta vypadku

# Lazy alokacia (pre sbrk)

- Ake su vyhody takehoto pristupu?

# Lazy alokacia (pre sbrk)

- Ake su vyhody takehoto pristupu?

- Alokuje sa menej fyzickej pamate (ak sa k pamati z user programu nepristupi, nepride k ziadnemu vypadku, ziadna pamat sa nebude alokovat)

# Lazy alokacia (pre sbrk)

- Ake su vyhody takehoto pristupu?
- Alokuje sa menej fyzickej pamate (ak sa k pamati z user programu nepristupi, nepride k ziadnemu vypadku, ziadna pamat sa nebude alokovat)
- Cenou tohto pristupu je strata istej efektivity – pamat sa musi alokovat nie pri jednom systemovom volani (sbrk), ale pri kazdom vypadku stranky

# Lazy alokacia (pre sbrk)

- Ako na to prakticky v xv6?

# Lazy alokacia (pre sbrk)

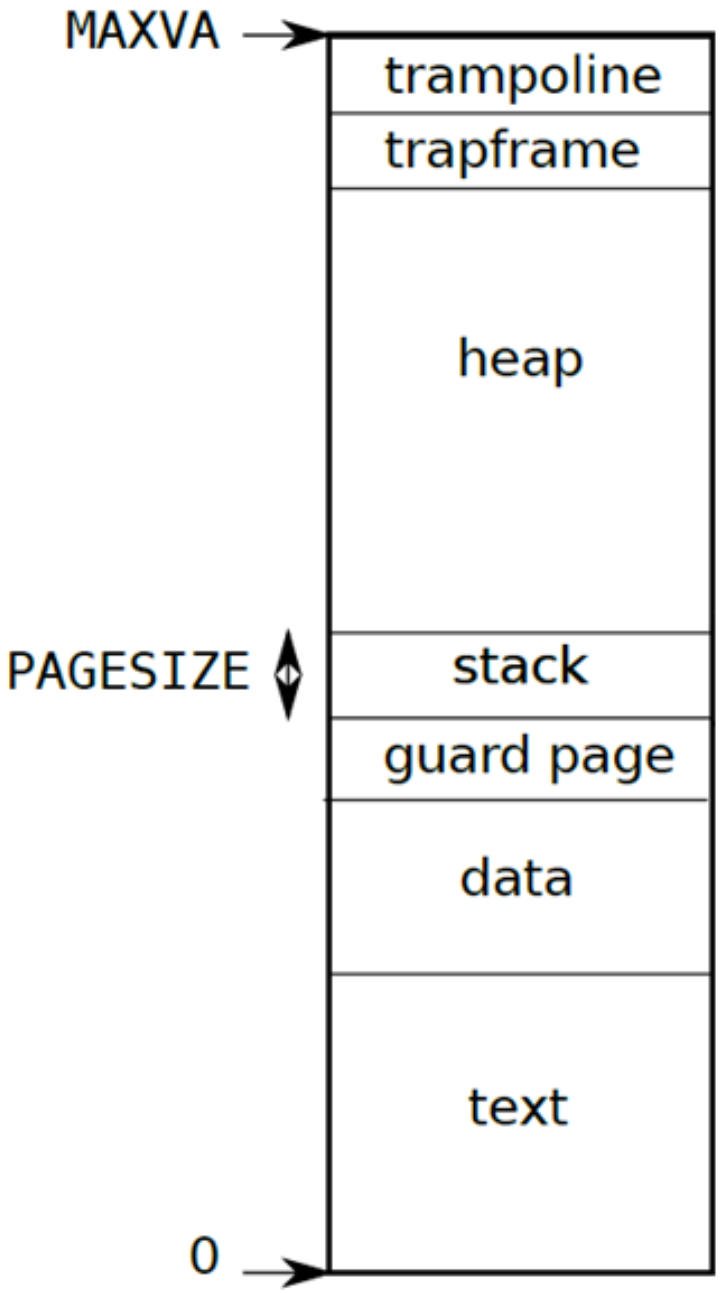
- Ako na to prakticky v xv6?

1. sbrk(): p→sz = p→sz+n

# Lazy alokacia (pre sbrk)

- Ako na to prakticky v xv6?

1. sbrk(): p→sz = p→sz+n
2. pgfault():
   - Kontrola VA v intervale <……; ……..)


MAXVA
trampoline
trapframe
heap
PAGESIZE
stack
guard page
data
text
0

# Lazy alokacia (pre sbrk)

- Ako na to prakticky v xv6?

1. sbrk(): p→sz = p→sz+n
2. pgfault():
   - Kontrola VA v intervale <……; ……..)
   - Alokuj ramec v ram-ke
   - Vynuluj ramec (preco?)
   - Namapuj prislusnu stranku VA na ramec
   - Restartuj user program od instrukcie, ktora sposobila vypadok

# Lazy alokacia (pre sbrk)

- Osetrenie chyb alokacie RAM pri standardnom (nie lazy) sbrk() je trivialne

# Lazy alokacia (pre sbrk)

- Osetrenie chyb alokacie RAM pri standardnom (nie lazy) sbrk() je trivialne
- Ako je to v pripade oneskorenej alokacie?

# Lazy alokacia (pre sbrk)

- Osetrenie chyb alokacie RAM pri standardnom (nie lazy) sbrk() je trivialne
- Ako je to v pripade oneskorenej alokacie?

pgfault():

- Kontrola VA v intervale <……; ……..)
- Alokuj ramec v ram-ke
- Vynuluj ramec (preco?)
- Namapuj prislusnu stranku VA na ramec
- Restartuj user program od instrukcie, ktora sposobila vypadok

# Lazy alokacia (pre sbrk)

- Osetrenie chyb alokacie RAM pri standardnom (nie lazy) sbrk() je trivialne
- Ako je to v pripade oneskorenej alokacie?

pgfault():

- Kontrola VA v intervale <……; ……..)
- Alokuj ramec v ram-ke
- Vynuluj ramec (preco?)
- Namapuj prislusnu stranku VA na ramec
- Restartuj user program od instrukcie, ktora sposobila vypadok

# Lazy alokacia (pre sbrk)

- kernel/sysproc.c:sys_sbrk()
  - Zakomentuj growproc()
  - Zvacsi p→sz
- make qemu
- echo cafte

# Lazy alokacia (pre sbrk)

- kernel/sysproc.c:sys_sbrk()
  - Zakomentuj growproc()
  - Zvacsi p→sz
- make qemu
- echo cafte

- usertrap(): unexpected scause 0xf sepc=0x12b4 stval=0x4008

# Lazy alokacia (pre sbrk)

- kernel/sysproc.c:sys_sbrk()
  - Zakomentuj growproc()
  - Zvacsi p→sz
- make qemu
- echo cafte

- usertrap(): unexpected scause 0xf sepc=0x12b4 stval=0x4008
  - 0x12b4 je v user/echo.asm?

# Lazy alokacia (pre sbrk)

- kernel/sysproc.c:sys_sbrk()
  - Zakomentuj growproc()
  - Zvacsi p→sz
- make qemu
- echo cafte

- usertrap(): unexpected scause 0xf sepc=0x12b4 stval=0x4008
  - 0x12b4 je v user/echo.asm?
  - Co je 0x12b4 v user/sh.asm?

# Lazy alokacia (pre sbrk)

- kernel/trap.c:usertrap()
  - else if(r_scause() == 0xf) …
  - Pridat kod, objasnit
- make qemu
- echo cafte

# Lazy alokacia (pre sbrk)

- kernel/trap.c:usertrap()
  - else if(r_scause() == 0xf) …
  - Pridat kod, objasnit
- make qemu
- echo cafte

- Prvy vypadok OK ;)
- Druhy vypadok OK
- Preco panic: uvmunmap: not mapped?

# Lazy alokacia (pre sbrk)

- Preco panic: uvmunmap: not mapped?
  - Ziadnu VA z danej stranky proces nepouzil, a preto nebolo mapovanie vytvorene
  - Upravme uvmunmap(): continue namiesto panic

# Lazy alokacia (pre sbrk)

- Preco panic: uvmunmap: not mapped?
  - Ziadnu VA z danej stranky proces nepouzil, a preto nebolo mapovanie vytvorene
  - Upravme uvmunmap(): continue namiesto panic

- make qemu
- echo cafte

# Lazy alokacia (pre sbrk)

- Preco panic: uvmunmap: not mapped?
  - Ziadnu VA z danej stranky proces nepouzil, a preto nebolo mapovanie vytvorene
  - Upravme uvmunmap(): continue namiesto panic

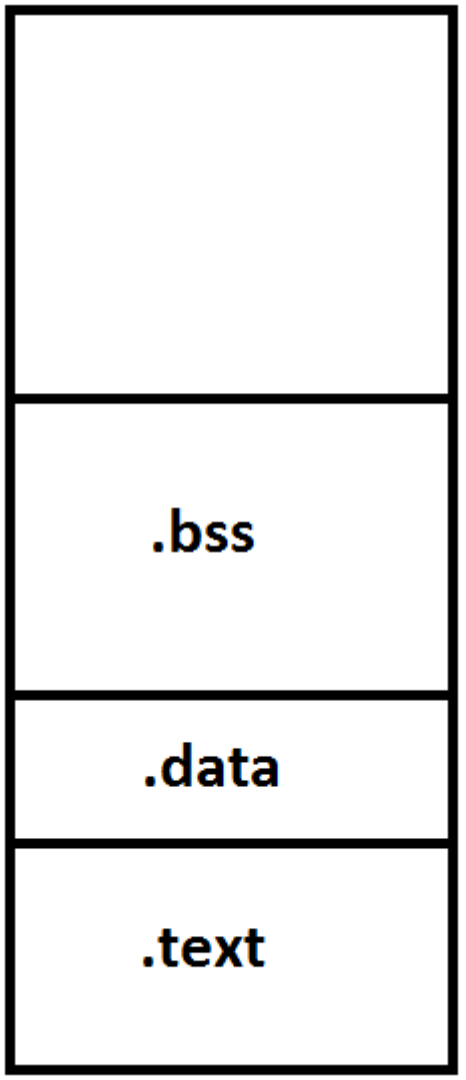
- make qemu
- echo cafte

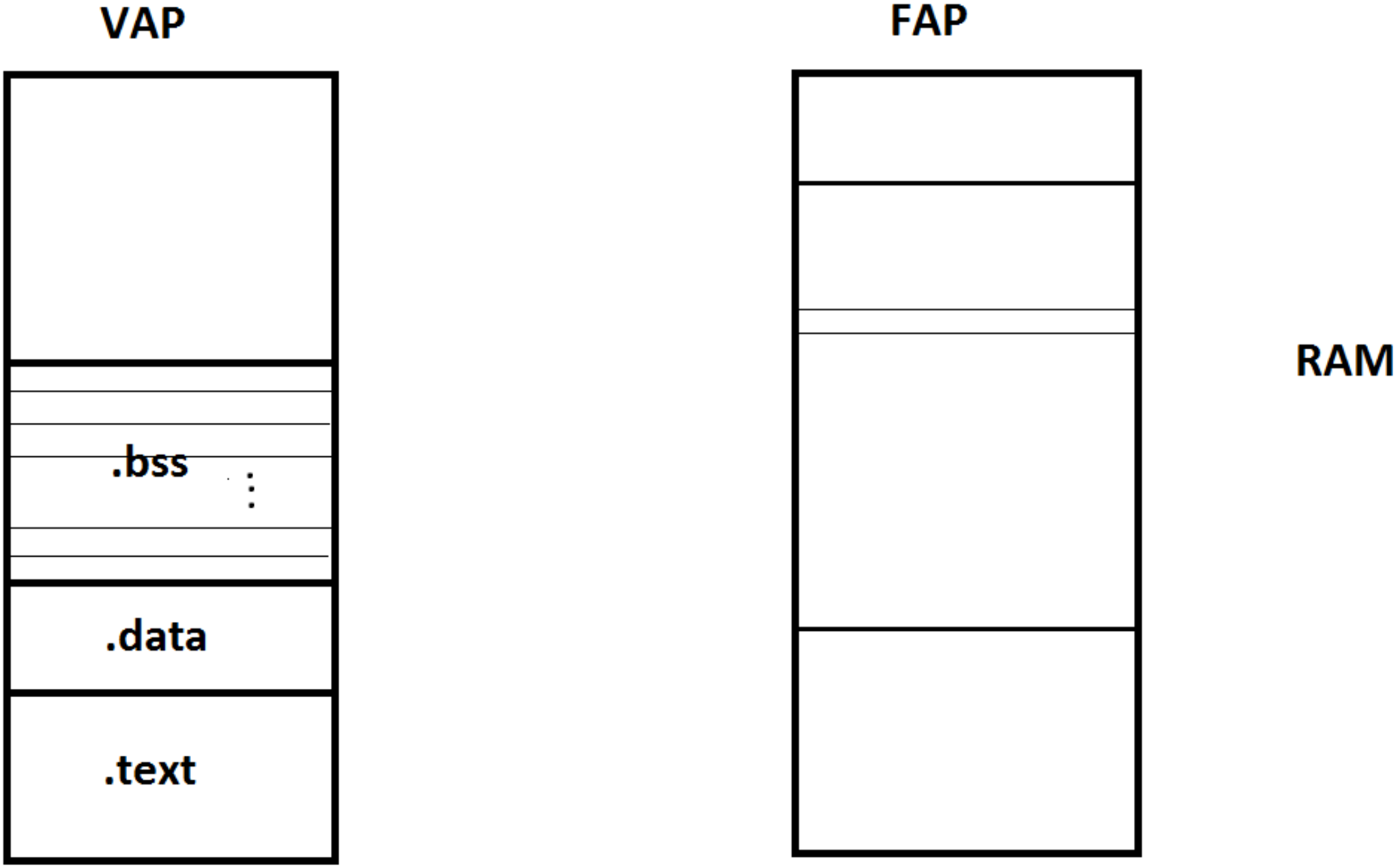
- Velmi zakladne a jednoducha schema lazy alokacie; viac na cviceni tohto tyzdna…

# Stranka nul na poziadanie

# Stranka nul na poziadanie

- Zero-fill on demand




VAP
.bss
.data
.text
FAP
RAM

# Stranka nul na poziadanie

- Co ma robit obsluha pgfault()?

# Stranka nul na poziadanie

- Co ma robit obsluha pgfault()

- Overit, ci VA ukazuje do zero-page
- Alokovat novy ramec
- Vynulovat ho / skopirovat ho
- Vytvorit mapovanie do ramca pre VA (write!)
- Restartovat instrukciu

# Stranka nul na poziadanie

- Naco je to dobre?

# Stranka nul na poziadanie

- Naco je to dobre?

1. Program vyuziva iba tolko pamate, kolko aktualne potrebuje – podobne ako pri lazy alokacii
2. exec() je efektivnejsi – netrva tak dlho (ziadna nulova stranka sa realne nealokuje)

# Copy-on-write fork

# Copy-on-write fork

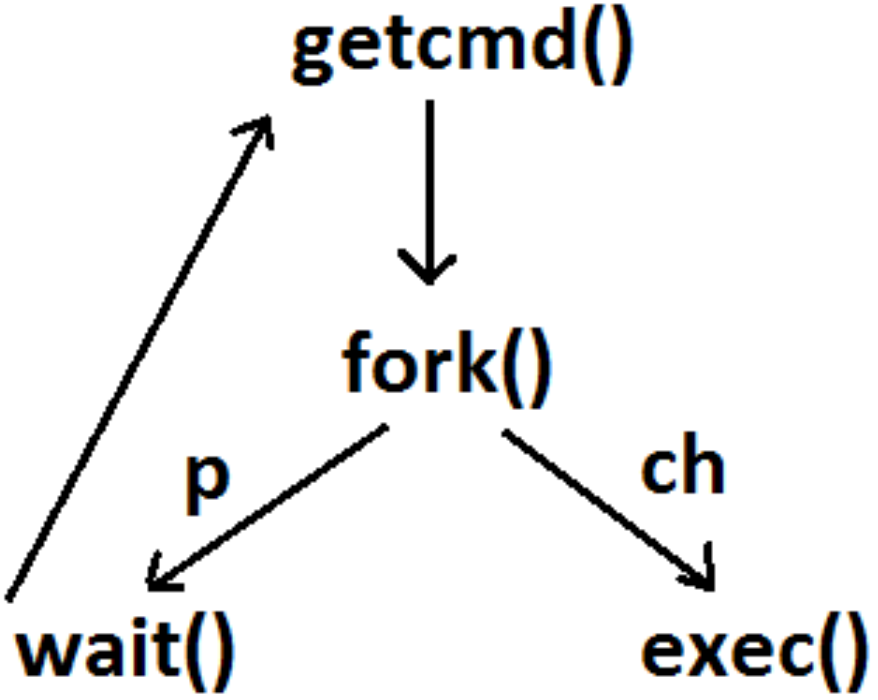
- Implementacia bude predmetom vlastneho cvicenia

# Copy-on-write fork

- Implementacia bude predmetom vlastneho cvicenia
- Ako funguje spustenie prikazu v sh?

# Copy-on-write fork

- Implementacia bude predmetom vlastneho cvicenia
- Ako funguje spustenie prikazu v sh?



# Copy-on-write fork

- fork() skopiruje cely VAP rodica
- exec() cely VAP procesu zrusi a nahradi ho VAP noveho procesu

- Obrazok na tabulu

# Copy-on-write fork

- Ciel COW forku – vsetky VA detskeho procesu (totozne s VA rodica) budu ukazovat na tie iste udaje v RAM

# Copy-on-write fork

- Ciel COW forku – vsetky VA detskeho procesu (totozne s VA rodica) budu ukazovat na tie iste udaje v RAM

- Pozor na stranky, ktore maju PTE_W!

# Copy-on-write fork

- Ciel COW forku – vsetky VA detskeho procesu (totozne s VA rodica) budu ukazovat na tie iste udaje v RAM

- Pozor na stranky, ktore maju PTE_W!
  - Aj v rodicovi, aj v potomkovi ich musime zmenit na PTE_R
  - Pri pokuse o zapis sa vygeneruje vynimka

# Copy-on-write fork

- Co ma urobit pgfault()?

# Copy-on-write fork

- Co ma urobit pgfault()

- Skontroluj, ci stranka zodpovedajuca VA ma “narok” byt zapisovatelna
- Alokuj ramec
- Skopiruj do neho udaje z PTE_R stranky
- Vytvor do neho mapovanie pre VA (PTE_W)
- Restartuj instrukciu

# Copy-on-write fork

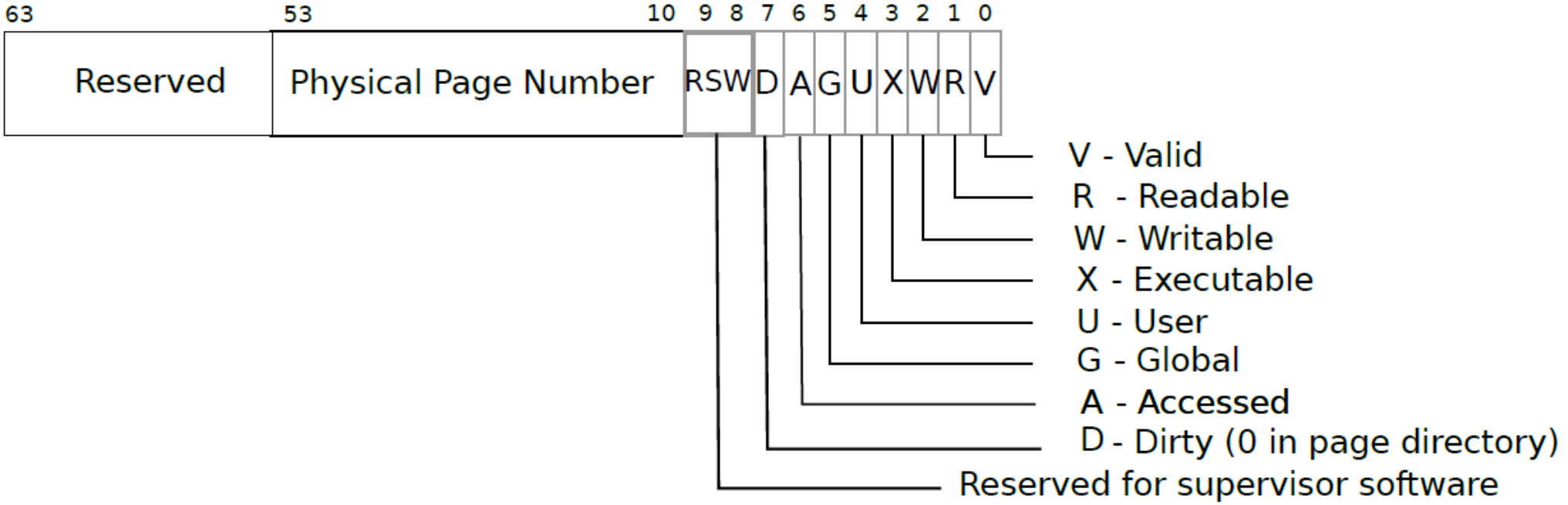
- Skontroluj, ci stranka zodpovedajuca VA ma “narok” byt zapisovatelna. Ako?

# Copy-on-write fork

- Skontroluj, ci stranka zodpovedajuca VA ma “narok” byt zapisovatelna. Ako?
- Vyuzijeme jeden z RSW bitov PTE zaznamu

# Copy-on-write fork

- Skontroluj, ci stranka zodpovedajuca VA ma “narok” byt zapisovatelna. Ako?
- Vyuzijeme jeden z RSW bitov PTE zaznamu



# Copy-on-write fork

- Skontroluj, ci stranka zodpovedajuca VA ma “narok” byt zapisovatelna. Ako?
- Vyuzijeme jeden z RSW bitov PTE zaznamu
  - Pri kopirovani VAP rodica vo fork() vsetkym mapovaniam PTE_W odstranime PTE_W a pridame nami vyuzity bit (oznacme ho napr. PTE_COW)

# Copy-on-write fork

- Skontroluj, ci stranka zodpovedajuca VA ma “narok” byt zapisovatelna. Ako?
- Vyuzijeme jeden z RSW bitov PTE zaznamu
  - Pri kopirovani VAP rodica vo fork() vsetkym mapovaniam PTE_W odstranime PTE_W a pridame nami vyuzity bit (oznacme ho napr. PTE_COW)
  - V obsluhe pgfault() skontrolujeme, ci mapovanie danej VA obsahuje PTE_COW: ak ano, pgfault() urobi co ma (alokacia, kopia, mapovanie, restart)

# Copy-on-write fork

- Preco musime robit zbytocnu operaciu alokacie/kopirovania/mapovania v pripade, ze po poslednom pgfault() nam ostane iba jeden jediny proces, ktory ukazuje na povodne data vo FAP? Vid obrazok…

- Neda sa to urobit nejako tak, aby sme tuto situaciu vyriesili?

# Copy-on-write fork

- Pri COW musime byt opatrni na viacerych miestach, zvlast pri uvolnovani stranok

# Copy-on-write fork

- Pri COW musime byt opatrni na viacerych miestach, zvlast pri uvolnovani stranok

- Napriklad systemove volanie exit()
  - Moze kernel uvolnit stranky procesu?

# Copy-on-write fork

- Pri COW musime byt opatrni na viacerych miestach, zvlast pri uvolnovani stranok

- Napriklad systemove volanie exit()
  - Moze kernel uvolnit stranky procesu?
  - Ani pri PTE_COW, ani pri PTE_R nevieme, kolko procesov ma ramec FAP namapovany do svojho VAP!

# Copy-on-write fork

- Ako vyriesit uvolnovanie stranok?

# Copy-on-write fork

- Ako vyriesit uvolnovanie stranok?
- Pocitadlo poctu referencii na ramec FAP

# Copy-on-write fork

- Ako vyriesit uvolnovanie stranok?

- Pocitadlo poctu referencii na ramec FAP
- Pri kazdom namapovani sa pocitadlo zvysi
- Pri kazdom odmapovani sa pocitadlo znizi

# Copy-on-write fork

- Ako vyriesit uvolnovanie stranok?

- Pocitadlo poctu referencii na ramec FAP
- Pri kazdom namapovani sa pocitadlo zvysi
- Pri kazdom odmapovani sa pocitadlo znizi

- Ked hodnota pocitadla klesne na 0, ramec sa uvolni

# Copy-on-write fork

- Na zavedenie pocitadla pre kazdy ramec potrebujeme vhodne datove struktury
- COW fork bude predmetom jedneho tyzdna v ramci cviceni

# Strankovanie na ziadost

- On-demand paging (exec)

# Strankovanie na ziadost

- On-demand paging (exec)

- exec() nahrava kompletne cely subor do pamate (vid kernel/exec.c), co je extremne narocna operacia
- Disk je totiz o dost radov pomalsi nez CPU, takze CPU nemoze efektivne vyuzit cas na cinnost

# Strankovanie na ziadost

- Cielom je pri sys volani exec() nacitat iba najnutnejsie udaje z disku, alokovat strukturu stranok, ale nenahravat udaje do RAM

- prislusne zaznamy PTE nebudu obsahovat PTE_V

# Strankovanie na ziadost

- Co ma robit pgfault()

# Strankovanie na ziadost

- Co ma robit pgfault()
- Skontroluje, ci ma ist o mapovanie do suboru
- Ak ano, alokuje ramec, nacita prislusne udaje do ramca, upravi PTE zaznam pre VA, restartuje instrukciu

# Strankovanie na ziadost

- Cinnost pgfault() vyzaduje nejake metadata o tom, kde sa stranka na disku nachadza
- Tieto informacie su zvycajne ulozene v strukture nazyvanej VMA (virtual memory area)

# swapovanie

- Vyuzitie vacsieho VAP nez je RAM

# swapovanie

- Vyuzitie vacsieho VAP nez je RAM
- Cielom je umoznit aplikaciam zdanie takej velkej RAM, ako potrebuju (bez ohladu na velkost samotnej RAM)

# swapovanie

- Ako to urobit?

# swapovanie

- Ako to urobit?

- Malo frekventovane stranky procesu odlozit na disk, a toto miesto pouzit na nove stranky, ktore chce proces pouzivat

# swapovanie

- Ako to urobit?

- Malo frekventovane stranky procesu odlozit na disk, a toto miesto pouzit na nove stranky, ktore chce proces pouzivat
- Odlozenie stranky na disk a jej nahratie pri dalsom pristupe musi byt pre proces transparentne

# swapovanie

- Ako vybrat “obet” (victim), ktora bude odlozena na disk?

# swapovanie

- Ako vybrat “obet” (victim), ktora bude odlozena na disk?

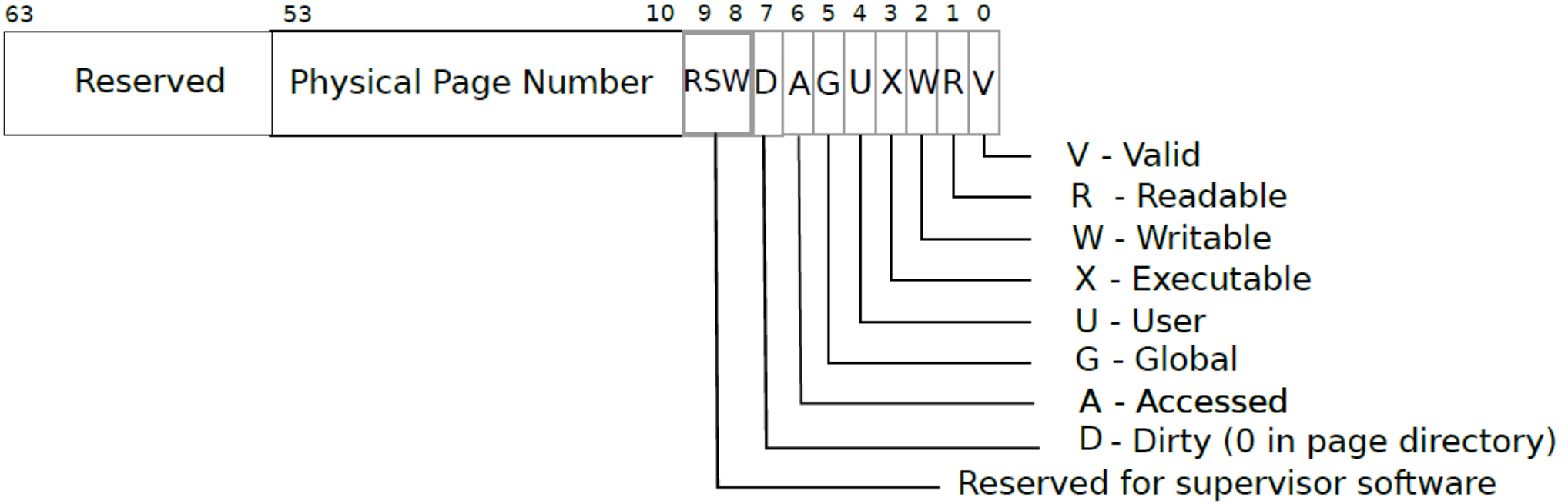
- Rozne algoritmy, najcastejsie sa pouziva LRU (Least Recently Used)
- Jednoducha verzia tohto algoritmu sa implementuje s podporou hw

# swapovanie

- Ako vybrat “obet” (victim), ktora bude odlozena na disk?

- Rozne algoritmy, najcastejsie sa pouziva LRU (Least Recently Used)
- Jednoducha verzia tohto algoritmu sa implementuje s podporou hw
  - Bit “A” v PTE zazname
  - Bit “D” v PTE zazname

# swapovanie




- Jednoducha verzia tohto algoritmu sa implementuje s podporou hw
  - Bit “A” v PTE zazname
  - Bit “D” v PTE zazname

# swapovanie

- Bit “A” v PTE zazname
- Pri kazdom pristupe MMU ku stranke sa tento bit nastavi

- Algoritmus vyberu obete

# swapovanie

- Bit “A” v PTE zazname
- Pri kazdom pristupe MMU ku stranke sa tento bit nastavi

- Algoritmus vyberu obete
  - Kazdych N tikov vynuluje bit A
  - V procese vyberu obete prehladava stranky, a tu, ktora nema nastaveny bit A, oznaci za obet

# swapovanie

- Bit “A” v PTE zazname
- Pri kazdom pristupe MMU ku stranke sa tento bit nastavi

- Algoritmus vyberu obete
  - Kazdych N tikov vynuluje bit A
  - V procese vyberu obete prehladava stranky, a tu, ktora nema nastaveny bit A, oznaci za obet
  - Algoritmus moze zohladnovat aj bit D

# swapovanie

- Podobne ako pri “on-demand” strankovani su potrebne pomocne datove struktury, aj v tomto pripade treba vediet, ktore udaje na disku zodpovedaju ktorej stranke vo VAP

# swapovanie

- Podobne ako pri “on-demand” strankovani su potrebne pomocne datove struktury, aj v tomto pripade treba vediet, ktore udaje na disku zodpovedaju ktorej stranke vo VAP

- Kedy sa hlada obet?

# swapovanie

- Podobne ako pri “on-demand” strankovani su potrebne pomocne datove struktury, aj v tomto pripade treba vediet, ktore udaje na disku zodpovedaju ktorej stranke vo VAP

- Kedy sa hlada obet?
- Ked nie je volny ramec RAM, t.j. ked kalloc() vrati NULL

# swapovanie

- Ako vyzera nacrt fungovania?

# swapovanie

- Ako vyzera nacrt fungovania?

- Ak nie je volna RAM
  - Najdi kandidata na vyhodenie z RAM (napr. LRU algoritmus)
  - Uloz data “victim” stranky na disk
  - Zneplatni mapovanie “victim” stranky
  - Pouzi volny ramec

# Mapovanie suboru do pamate

- mmap() – memory mapped files

# Mapovanie suboru do pamate

- mmap() – memory mapped files

- Cielom je manipulovat s obsahom suboru nie pomocu explicitnych volani read(), seek(), write()
- ale pomocou instrukcii ld (load), st (store), ktore manipuluju s pamatou RAM

# Mapovanie suboru do pamate

- mmap() – memory mapped files

- Cielom je manipulovat s obsahom suboru nie pomocu explicitnych volani read(), seek(), write()
- ale pomocou instrukcii ld (load), st (store), ktore manipuluju s pamatou RAM

- mmap(va, len, protection, flags, fd, offset)

# Mapovanie suboru do pamate

- Kernel nacitava udaje zo suboru do pamate technikou “on-demand” (v obsluhe pgfault())

- Ak je RAM plna, napr. algoritmom LRU moze nepouzivane casti suboru uvolnit

# Mapovanie suboru do pamate

- Kernel nacitava udaje zo suboru do pamate technikou “on-demand” (v obsluhe pgfault())
- Ak je RAM plna, napr. algoritmom LRU moze nepouzivane casti suboru uvolnit
- Systemove volanie unmap(va, len) zapise do suboru iba pozmenene (vyuzije sa “D” bit v PTE zazname) casti suboru

# Mapovanie suboru do pamate

- Na cinnost mapovania sa znovu vyuzivaju pomocne udaje VMA (virtual memory area)

# Mapovanie suboru do pamate

- Na cinnost mapovania sa znovu vyuzivaju pomocne udaje VMA (virtual memory area)

- Co v pripade, ze viacero procesov chce manipulovat sucasne s obsahom suboru?

# Mapovanie suboru do pamate

- Na cinnost mapovania sa znovu vyuzivaju pomocne udaje VMA (virtual memory area)

- Co v pripade, ze viacero procesov chce manipulovat sucasne s obsahom suboru?
- Podobne, ako ked viacero procesov sucasne pouziva systemove volania read()/write() nad tym istym suborom

# Zdielana virtualna pamat

# Zdielana virtualna pamat

- Cielom je umoznit procesom na roznych uzloch siete zdielat virtualnu pamat

# Zdielana virtualna pamat

- Cielom je umoznit procesom na roznych uzloch siete zdielat virtualnu pamat
- Vytvorit zdanie zdielania fyzickej pamate

# Domace citanie

- Chapter 4: Operating system organization knizky “xv6: a simple, Unix-like teaching operating system”

- So zameranim na cast 4.6: “Page-fault exceptions”