

OS MMXXI

MIT ;)

<https://pdos.csail.mit.edu/6.828>

Výpadky stránek

Téma

- Implementácia rôznych trikov pomocou VM
- Špeciálne zameranie na výpadky stránok
 - Oneskorená (lenivá) alokácia – *lazy allocation*
 - Alokácia pri zápise – *fork COW (copy-on-write)*
 - Mapovanie súboru do pamäte – *memory mapping*

Téma

- Načo sú tieto triky dobré
- Zlepšenie výkonu/efektivity
- Nová funkcionálnosť systému

Téma

- Načo sú tieto triky dobré
- Zlepšenie výkonu/efektivity
 - Oneskorená alokácia
 - Jedna nulová stránka pre celý systém
 - COW fork
- Nová funkcionálnosť systému
 - Mapovanie pamäte (mmap)

Reakcia xv6 na výpadok stránky

- Ako reaguje xv6 na výpadky stránok

Reakcia xv6 na výpadok stránky

- Ako reaguje xv6 na výpadky stránok
- `usertrap()`: unexpected scause ...
- Vid' príklad `user/echo.c`
- Ukončenie procesu

Reakcia xv6 na výpadok stránky

- Ako reaguje xv6 na výpadky stránok
- `usertrap()`: unexpected scause ...
- Vid' príklad `user/echo.c`
- Ukončenie procesu
- Čo výpadok stránky v jadre?
- Napríklad `kernel/sysproc.c sys_exit()`

Opakovanie

- Výhody VM
 1. Izolácia – každý proces má svoj vlastný adresný priestor
 2. Nepriamočiarosť (*level-of-indirection*) – preklad VA na FA umožňuje triky
 - Zdieľanie dát v ramke (trampolína)
 - Stráženie pretečenia zásobníka (*guard page*)
 - ...

Statické / dynamické mapovanie

- Jadro OS má kontrolu nad prekladom VA → FA
- Doteraz sme sa venovali pevne nastavenému mapovaniu (v zmysle, že mapovanie musí byť nastavené PRED prístupom k VA)
- VM však umožňuje aj mapovanie “za chodu”

Statické / dynamické mapovanie

- Jadro OS má kontrolu nad prekladom VA → FA
- Doteraz sme sa venovali pevne nastavenému mapovaniu (v zmysle, že mapovanie musí byť nastavené PRED prístupom k VA)
- VM však umožňuje aj mapovanie “za chodu”
 - Pri pokuse o prístup na VA sa generuje výnimka
 - Jadro OS pre danú VA vytvorí mapovanie a reštartuje proces od inštrukcie, ktorá spôsobila výnimku (proces platne pristúpi k údajom na VA)

Terminológia RISC-V

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

Terminológia RISC-V

- *Exception* (výnimka)
- *Trap* (presun riadenia)
- *Interrupt* (prerušenie)

Terminológia RISC-V

- *Exception* (výnimka) – výnimočný stav vyvolaný inštrukciou vykonávaného programu
- *Trap* (presun riadenia) – **synchronny** prenos riadenia do kódu obsluhy spôsobený výnimočným stavom, ktorý zapríčinil vykonávaný program
- *Interrupt* (prerušenie) – externá udalosť, ktorá sa vyskytne **asynchrónne** voči vykonávanému kódu

Supervisor Cause (scause) register

Interrupt	Exception Code	Description
1	0	<i>Reserved</i>
1	1	Supervisor software interrupt
1	2–4	<i>Reserved</i>
1	5	Supervisor timer interrupt
1	6–8	<i>Reserved</i>
1	9	Supervisor external interrupt
1	10–15	<i>Reserved</i>
1	≥ 16	<i>Available for platform use</i>
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	10–11	<i>Reserved</i>
0	12	Instruction page fault
0	13	Load page fault
0	14	<i>Reserved</i>
0	15	Store/AMO page fault
0	16–23	<i>Reserved</i>
0	24–31	<i>Available for custom use</i>
0	32–47	<i>Reserved</i>
0	48–63	<i>Available for custom use</i>
0	≥ 64	<i>Reserved</i>

Ciel' tohto týždňa

- Pri výpadku stránky v užívateľskom programe
- Zistiť, či sa jedná o legitímnu adresu programu
- Zistiť, či sa jedná o legitímny nárok prístupu
- Ak áno, aktualizovať tabuľku stránok procesu
- A reštartovať proces od inštrukcie, ktorá spôsobila výpadok stránky (vykonanie inštrukcie sa musí zopakovať)

Čo potrebujeme

1. VA, ktorá spôsobila výpadok (*faulting VA*) (na RISC-V je táto hodnota uložená v registri `$stval`)
2. Typ výpadku stránky (pre RISC-V vid' tabuľku 4.2 v "RISC-V privileged.pdf", hodnota registra `$scause` – *read, write, instruction*)

Supervisor Cause (scause) register

Interrupt	Exception Code	Description
1	0	<i>Reserved</i>
1	1	Supervisor software interrupt
1	2–4	<i>Reserved</i>
1	5	Supervisor timer interrupt
1	6–8	<i>Reserved</i>
1	9	Supervisor external interrupt
1	10–15	<i>Reserved</i>
1	≥ 16	<i>Available for platform use</i>
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	10–11	<i>Reserved</i>
0	12	Instruction page fault
0	13	Load page fault
0	14	<i>Reserved</i>
0	15	Store/AMO page fault
0	16–23	<i>Reserved</i>
0	24–31	<i>Available for custom use</i>
0	32–47	<i>Reserved</i>
0	48–63	<i>Available for custom use</i>
0	≥ 64	<i>Reserved</i>

Čo potrebujeme

1. VA, ktorá spôsobila výpadok (`$stval`)
2. Typ výpadku stránky (`$scause`)
3. Mód CPU a inštrukciu, kde k výpadku prišlo
 - U/S mód: implicitne sa vyvolá `user trap` alebo `kernel trap`
 - PC: `$sepc`, hodnota uložená v `tf` → `epc` pri móde U

Zdroj informácií

- pre RISC-V vid' tabuľka 4.2 v "RISC-V privileged.pdf" na strane 66
- <https://uim.fei.stuba.sk/wp-content/uploads/2018/02/riscv-privileged.pdf>
- <https://uim.fei.stuba.sk/predmet/b-os/>
- záložka Literatúra

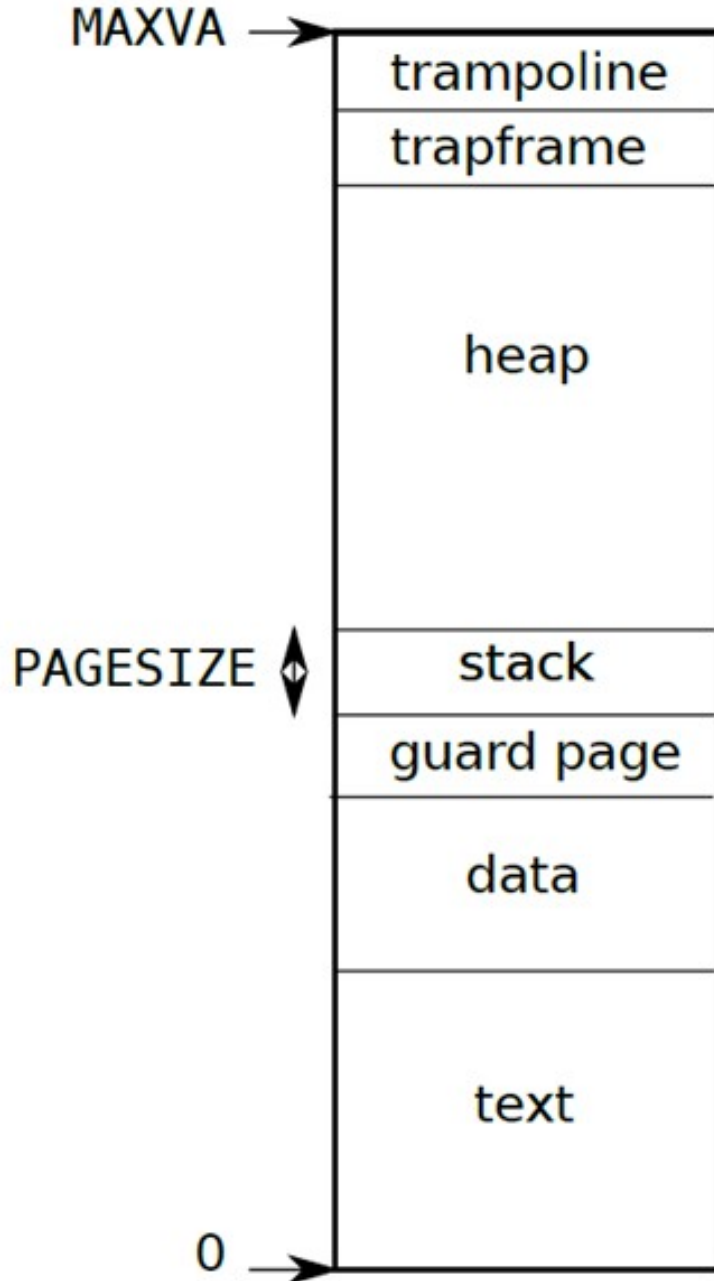
Čo potrebujeme

- Načo nám treba register PC?
- Aby kód jadra vedel reštartovať užívateľský program presne od tej inštrukcie, ktorá vyvolala výpadok
- Inštrukcia sa musí zopakovať (jej vykonanie)
- Ak jadro dobre všetko nastavilo, pri opakovaní inštrukcie sa už výnimka neobjaví

Zoznam trikov s VM

- alokácia pamäte na žiadosť (*lazy/on-demand*)
- jedna stránka núl (*zero-filled page*)
- COW fork (*copy-on-write*)
- stránkovanie na žiadosť (*on-demand ver. 2*)
- VA väčšia než RAM (*swap*)
- mapovanie súborov do pamäte (*mmap*)
- zdieľanie pamäte medzi procesmi (*shared*)

sbrk, fork a exec



Lazy alokácia v1 (pre sbrk)

- `sbrk()` v `xv6` je primitívny – vždy procesu dá, čo chce
- Aplikácia často nevyužije všetku pridelenú pamäť
 - Napr. alokuje *buffer* na načítanie vstupu, ale väčšinou sa využije iba prvých pár bajtov
- `sbrk()` tak často alokuje pamäť, ktorá sa NIKDY v procese nevyužije, ale nikto iný ju nemôže využívať!

Lazy alokácia v1 (pre sbr k)

- Moderné OS alokujú pamäť oneskorene (lenivo, *lazy*)
- Ako:
 - Fyzická ram sa alokuje až vtedy, keď je treba
 - Pri volaní `sbr k ()` sa zväčší `p -> sz`, ale nerobí sa žiadna alokácia (mapovanie)
 - Keď sa proces pokúsi prístupit' k "alokovanej" pamäti, nastane výpadok stránky!!!
 - Jadro v obsluhu výpadku alokuje a namapuje potrebnú pamäť
 - Aplikácia sa reštartuje od miesta výpadku

Lazy alokácia v1 (pre sbr k)

- Aké sú výhody takéhoto prístupu?
- Alokujú sa menej fyzickej pamäte (ak sa k pamäti z *user* programu nepristúpi, nepríde k žiadnemu výpadku, žiadna pamäť sa nebude alokovať)
- Cenou tohto prístupu je strata istej efektivity – pamäť sa musí alokovať nie pri jednom systémovom volaní (sbr k), ale pri každom výpadku stránky

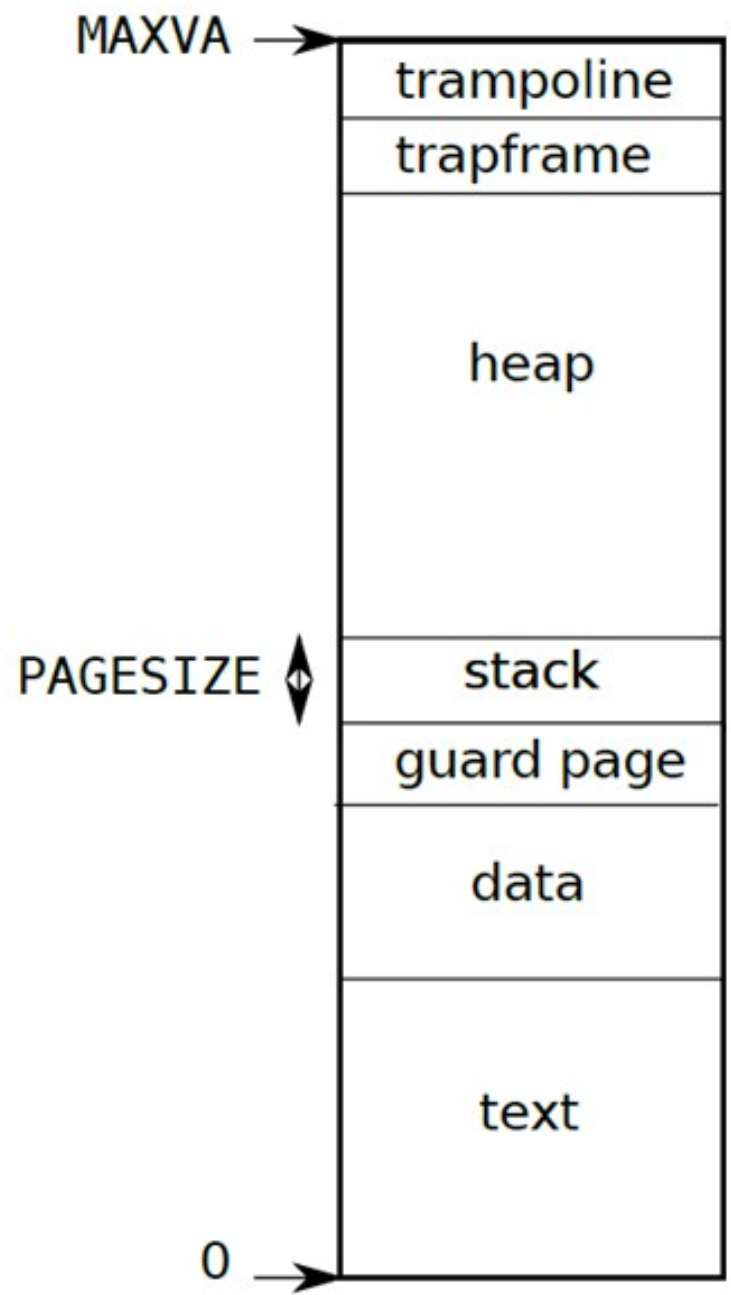
Lazy alokácia v1 (pre sbrk)

- Ako na to prakticky v xv6?

1. sbrk(): $p \rightarrow sz = p \rightarrow sz + n$

2. pgfault():

- Kontrola VA v intervale $\langle \dots; \dots \rangle$
- Alokuj rámec v ram-ke
- Vynuluj rámec (prečo?)
- Namapuj príslušnú stránku VA na rámec
- Reštartuj *user* program od inštrukcie, ktorá spôsobila výpadok



Lazy alokácia v1 (pre sbrk)

- Ošetrenie chýb alokácie RAM pri štandardnom (nie *lazy*) `sbrk()` je triviálne
- Ako je to v prípade oneskorenej alokácie?

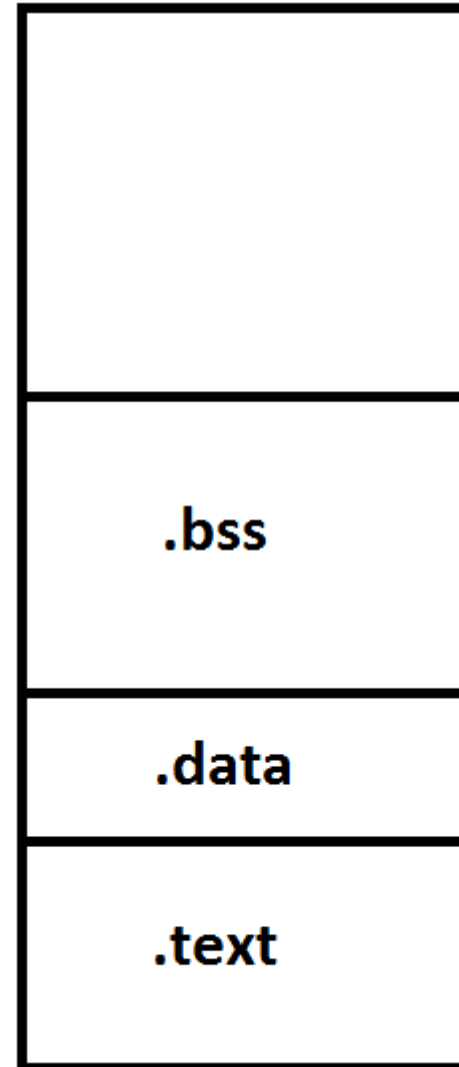
`pgfault()`:

- Kontrola VA v intervale $\langle \dots; \dots \rangle$
- **Alokuj rámec v ram-ke**
- Vynuluj rámec (prečo?)
- Namapuj príslušnú stránku VA na rámec
- Reštartuj *user* program od inštrukcie, ktorá spôsobila výpadok

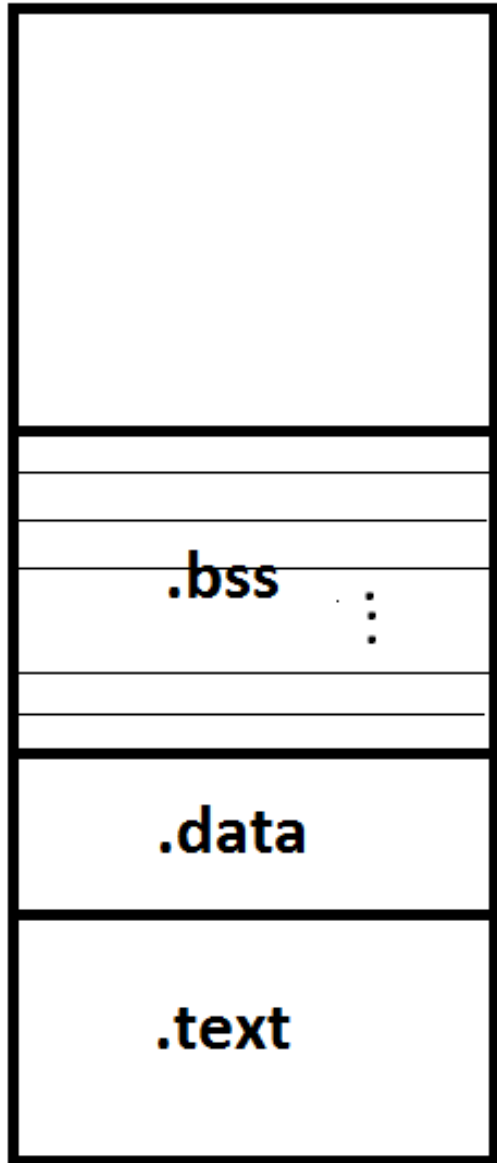
Stránka núl na požiadanie

Stránka núl na požiadanie

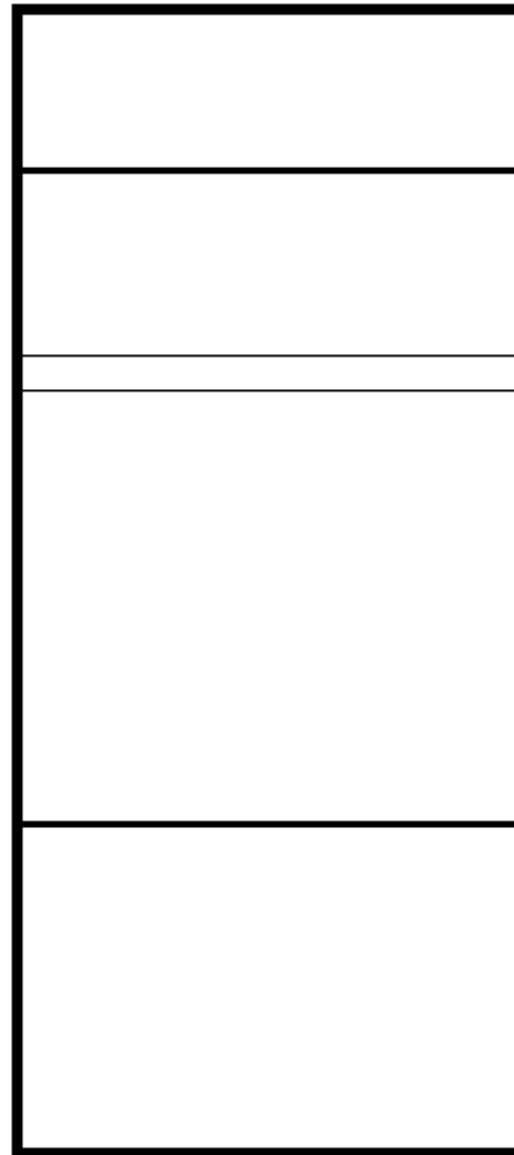
- *Zero-fill on demand*



VAP



FAP



RAM

Stránka núl na požiadanie

- Čo má robiť obsluha `pgfault()`
- Overiť, či VA ukazuje do *zero-page*
- Alokovať nový rámec
- Vynulovať ho / skopírovať ho
- Vytvoriť mapovanie do rámca pre VA (`PTE_W`)
- Reštartovať inštrukciu

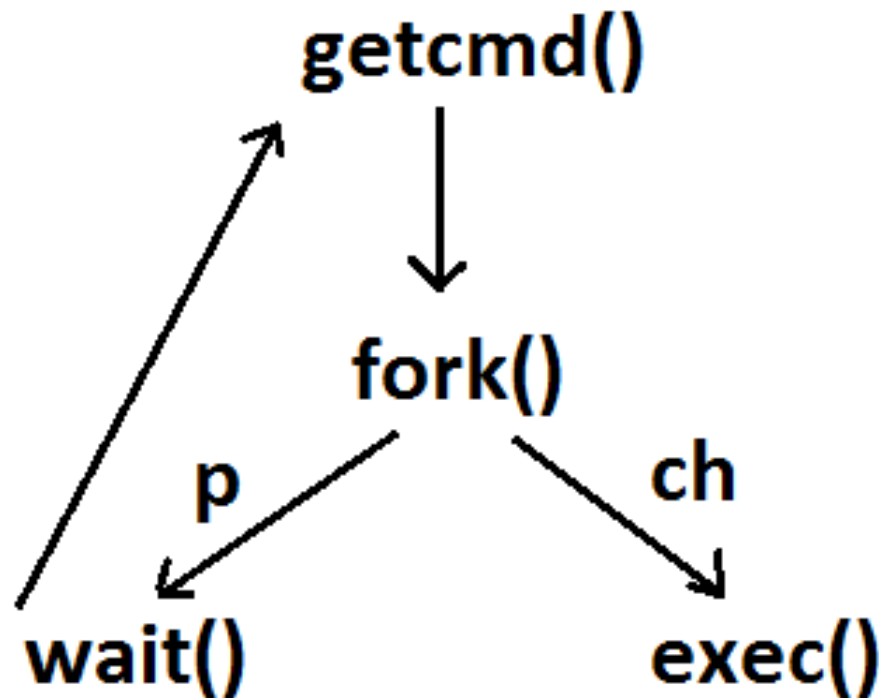
Stránka núl na požiadanie

- Načo je to dobré?
 1. Program využíva iba toľko pamäte, koľko aktuálne potrebuje – podobne ako pri *lazy* alokácii
 2. `exec ()` je efektívnejší – netrvá tak dlho (žiadna nulová stránka sa reálne nealokuje)

Copy-on-write fork

Copy-on-write fork

- Implementácia je predmetom cvičenia tento týždeň
- Ako funguje spustenie príkazu v sh?



Copy-on-write fork

- `fork()` skopíruje celý VAP rodiča
- `exec()` celý VAP procesu zruší a nahradí ho VAP nového procesu

- Brutálny obrázok

Copy-on-write fork

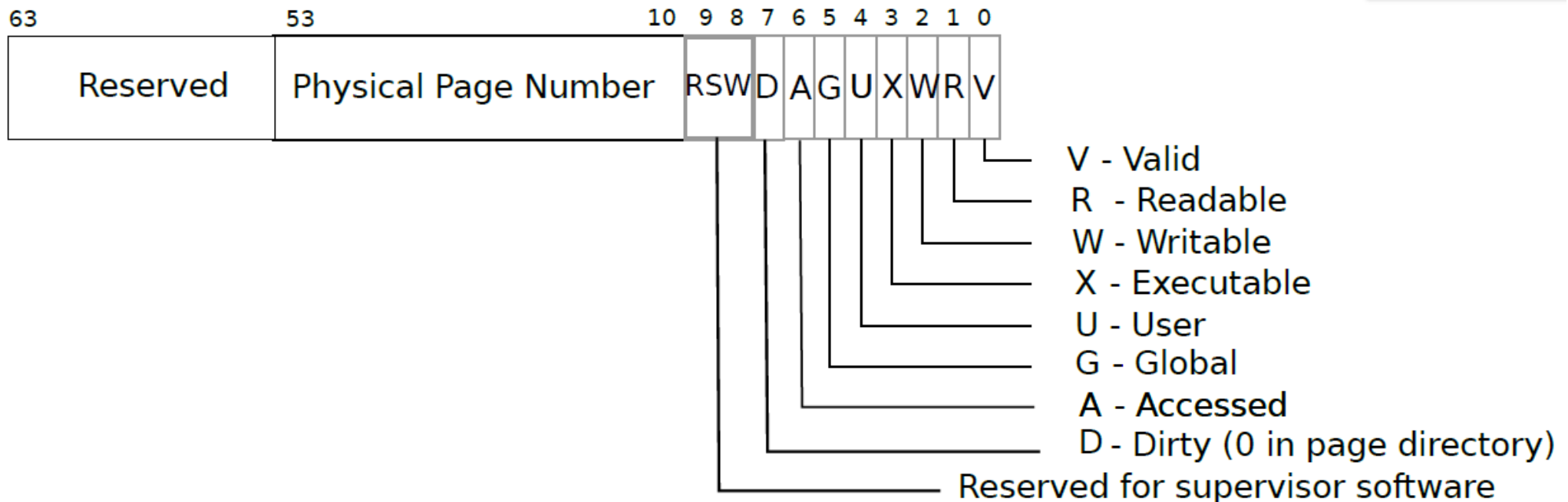
- Cieľ COW forku – všetky VA detského procesu (totožné s VA rodiča) budú ukazovať na tie isté údaje v RAM
- Pozor na stránky, ktoré majú PTE_W!
 - Aj v rodičovi, aj v potomkovi musíme odstrániť PTE_W
 - Dôsledok: pri pokuse o zápis sa vygeneruje výnimka

Copy-on-write fork

- Čo má urobiť `pgfault()`
- Skontroluj, či stránka zodpovedajúca VA má “nárok” byť zapisovateľná
- Alokuj rámec v RAM
- Skopíruj do neho údaje z PTE_R stránky
- Vytvor do neho mapovanie pre VA (PTE_W)
- Reštartuj inštrukciu

Copy-on-write fork

- Skontroluj, či stránka zodpovedajúca VA má “nárok” byť zapisovateľná. Ako?
- Využijeme jeden z RSW bitov PTE záznamu



Copy-on-write fork

- Skontroluj, či stránka zodpovedajúca VA má “nárok” byť zapisovateľná. Ako?
- Využijeme jeden z RSW bitov PTE záznamu
 - Pri kopírovaní VAP rodiča vo `fork()` všetkým mapovaniam PTE_W odstránime PTE_W a pridáme nami využitý bit (označme ho napr. PTE_COW)
 - V obsluhu `pgfault()` skontrolujeme, či mapovanie danej VA obsahuje PTE_COW: ak áno, `pgfault()` urobí, čo má (alokácia, kópia, mapovanie, reštart inštrukcie)

Copy-on-write fork

- Prečo musíme robiť zbytočnú operáciu alokácie/kopírovania/mapovania v prípade, že po poslednom `pgfault()` nám ostane iba jeden jediný proces, ktorý ukazuje na pôvodné data vo FAP? Vid' obrázok...
- Nedá sa to urobiť nejako tak, aby sme túto situáciu vyriešili?

Copy-on-write fork

- Pri COW musíme byť opatrní na viacerých miestach, zvlášť pri uvoľňovaní stránok
- Napríklad systémové volanie `exit()`
 - Môže jadro uvoľniť stránky procesu?
 - Ani pri `PTE_COW`, ani pri `PTE_R` nevieme, koľko procesov má rámec FAP namapovaný do svojho VAP!

Copy-on-write fork

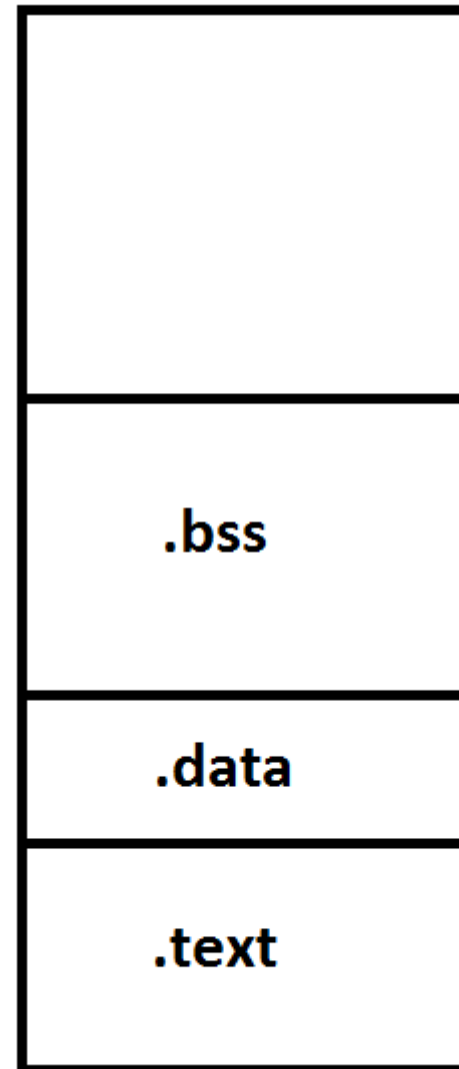
- Ako vyriešiť uvoľňovanie stránok?
- Počítadlo počtu referencií na rámec FAP
- Pri každom namapovaní sa počítadlo zvýši
- Pri každom odmapovaní sa počítadlo zníži
- Keď hodnota počítadla klesne na 0, rámec sa uvoľní

Copy-on-write fork

- Na zavedenie počítačadla pre každý rámec potrebujeme vhodné datové štruktúry
- COW fork je predmetom tohto týždňa v rámci cvičení

Stránkovanie na žiadosť v2

- *On-demand paging (exec)*



Stránkovanie na žiadosť v2

- *On-demand paging (exec)*
- `exec ()` nahráva kompletne celý súbor do pamäte (vid' `kernel/exec.c`), čo je extrémne náročná operácia
- Disk je totiž o dost' rádov pomalší než CPU, takže CPU nemôže efektívne využiť čas na činnosť

Stránkovanie na žiadosť v2

- Cieľom je pri sys. volaní `exec()` načítať iba najnutnejšie údaje z disku, alokovať štruktúru stránok, ale nenahrávať údaje do RAM
- príslušné záznamy PTE nebudú obsahovať `PTE_V`

Stránkovanie na žiadosť v2

- Čo má robiť `pgfault()`
- Skontroluje, či má ísť o mapovanie do súboru
- Ak áno, alokuje rámec, načíta príslušné údaje do rámca, upraví PTE záznam pre VA, reštartuje inštrukciu

Stránkovanie na žiadosť v2

- Činnosť `pgfault()` vyžaduje nejaké metadáta o tom, kde sa stránka na disku nachádza
- Tieto informácie sú zvyčajne uložené v štruktúre nazývanej VMA (*Virtual Memory Area*)

swap

- Využitie väčšieho VAP než je RAM

swap

- Využitie väčšieho VAP než je RAM
- Cieľom je umožniť aplikáciám zdanie takej veľkej RAM, ako potrebujú (bez ohľadu na veľkosť samotnej RAM)

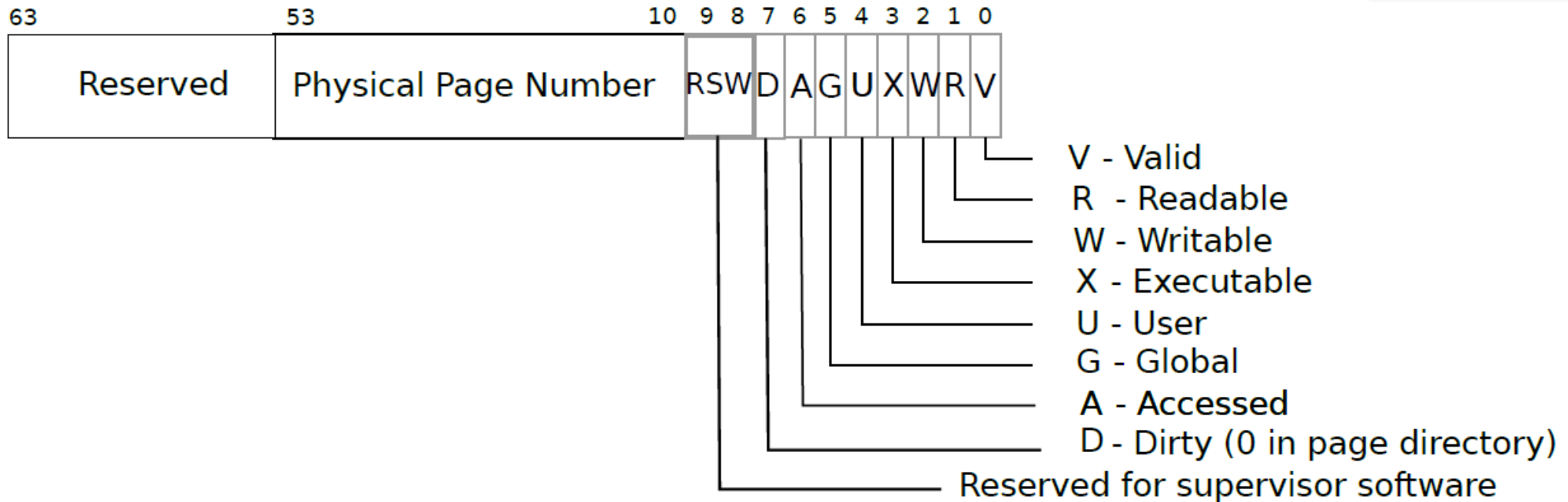
swap

- Ako to urobiť?
- Málo frekventované stránky procesu odložiť na disk, a toto miesto použiť na nové stránky, ktoré chce proces používať
- Odloženie stránky na disk a jej nahratie pri ďalšom prístupe musí byť pre proces transparentné

swap

- Ako vybrať “obet” (*victim*), ktorá bude odložená na disk?
- Rôzne algoritmy, najčastejšie sa používa LRU (*Least Recently Used*)
- Jednoduchá verzia tohto algoritmu sa implementuje s podporou hw
 - Bit “A” v PTE zázname
 - Bit “D” v PTE zázname

swap



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

swap

- Bit “A” v PTE zázname
- Pri každom prístupe MMU ku stránke sa tento bit nastaví (to už vieme)
- Algoritmus výberu obete
 - Každých N tikov vynuluje bit A
 - V procese výberu obete prehľadáva stránky, a tú, ktorá nemá nastavený bit A, označí za obeť
 - Algoritmus môže zohľadňovať aj bit D

swap

- Podobne ako pri “*on-demand*” stránkovaní sú potrebné pomocné dátové štruktúry, aj v tomto prípade treba vedieť, ktoré údaje na disku zodpovedajú ktorej stránke vo VAP
- Kedy sa hľadá obeť?
- Keď nie je voľný rámec RAM, t.j. keď `ka1loc()` vráti 0 (*null*)

swap

- Ako vyzerá náčrt fungovania?
- Ak nie je voľná RAM
 - Nájdi kandidáta na vyhodenie z RAM (napr. LRU algoritmus)
 - Ulož dáta “*victim*” stránky na disk
 - Zneplatni mapovanie “*victim*” stránky
 - Použi voľný rámec

Mapovanie súboru do pamäte

- `mmap()` – *memory mapped files*

Mapovanie súboru do pamäte

- `mmap()` – *memory mapped files*
- Cieľom je manipulovať s obsahom súboru nie pomocou explicitných volaní `read()`, `seek()`, `write()`
- ale pomocou inštrukcií `ld` (*load*), `st` (*store*), ktoré manipulujú s pamäťou RAM
- `mmap(va, len, protection, flags, fd, offset)`

Mapovanie súboru do pamäte

- Jadro načítava údaje zo súboru do pamäte technikou “*on-demand*” (v obsluhu `pgfault()`)
- Ak je RAM plná, napr. algoritmom LRU môže nepoužívané časti súboru uvoľniť
- Systémové volanie `unmap(va, len)` zapíše do súboru iba pozmenené (využije sa “D” bit v PTE zázname) časti súboru

Mapovanie súboru do pamäte

- Na činnosť mapovania sa znovu využívajú pomocné údaje VMA (*virtual memory area*)
- Čo v prípade, že viacero procesov chce manipulovať súčasne s obsahom súboru?
- Podobne, ako keď viacero procesov súčasne používa systémové volania `read()` alebo `write()` nad tým istým súborom

Zdieľaná virtuálna pamäť

- Cieľom je umožniť procesom na rôznych uzloch siete zdieľať virtuálnu pamäť
- Vytvoriť zdanie zdieľania fyzickej pamäte

Domáce čítanie

- Chapter 4: Operating system organization
knižky “xv6: a simple, Unix-like teaching operating system”
- So zameraním na časť 4.6: “Page-fault exceptions”