OS MMXX

MIT ;)

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

Prerusenia

Tema

Tema

- Stlacenie klavesy
- Pohyb mysou
- Tik casovaca
- Udaje pripravene na vstupe sietovej karty
- …

Tema

- Stlacenie klavesy
- Pohyb mysou
- Tik casovaca
- Udaje pripravene na vstupe sietovej karty
- …

- HW si vyzaduje OKAMZITU pozornost!

Tema

- CPU musi
  - Odlozit aktualnu cinnost (ulozit aktualny stav)
  - Obsluzit hw (obsluzit prerusenie)
  - Obnovit vykonavanie cinnosti pred prerusenim

Tema

- CPU musi
  - Odlozit aktualnu cinnost (ulozit aktualny stav)
  - Obsluzit hw (obsluzit prerusenie)
  - Obnovit vykonavanie cinnosti pred prerusenim
- Na spracovanie preruseni na RISC-V sa pouziva ten isty mechanizmus ako pre
  - Systemove volania (syscalls)
  - Vynimky (exceptions)

Komplikacie preruseni

- Prerusenia su asynchronne

Komplikacie preruseni

- Prerusenia su asynchronne
  - Kod vykonavany na CPU pred prichodom prerusenia nijako nesuvisi s prerusenim! Nie je medzi nim a prerusenim ziadna kauzalita

Komplikacie preruseni

- Prerusenia su asynchronne
  - Kod vykonavany na CPU pred prichodom prerusenia nijako nesuvisi s prerusenim! Nie je medzi nim a prerusenim ziadna kauzalita
  - Kod obsluhy prerusenia nebezi v “kontexte” procesu (v xv6 nema zmysel vyuzivat myproc())

Komplikacie preruseni

- Prerusenia su asynchronne
  - Kod vykonavany na CPU pred prichodom prerusenia nijako nesuvisi s prerusenim! Nie je medzi nim a prerusenim ziadna kauzalita
  - Kod obsluhy prerusenia nebezi v “kontexte” procesu (v xv6 nema zmysel vyuzivat myproc())
- Viac konkurentne vykonavanych veci

Komplikacie preruseni

- Prerusenia su asynchronne
  - Kod vykonavany na CPU pred prichodom prerusenia nijako nesuvisi s prerusenim! Nie je medzi nim a prerusenim ziadna kauzalita
  - Kod obsluhy prerusenia nebezi v “kontexte” procesu (v xv6 nema zmysel vyuzivat myproc())
- Viac konkurentne vykonavanych veci
  - CPU vykonava kod, zaroven sa nieco deje na zariadeni

Komplikacie preruseni

- Prerusenia su asynchronne
  - Kod vykonavany na CPU pred prichodom prerusenia nijako nesuvisi s prerusenim! Nie je medzi nim a prerusenim ziadna kauzalita
  - Kod obsluhy prerusenia nebezi v “kontexte” procesu (v xv6 nema zmysel vyuzivat myproc())
- Viac konkurentne vykonavanych veci
  - CPU vykonava kod, zaroven sa nieco deje na zariadeni
- Programovanie zariadeni
  - Moze byt znacne zlozite naprogramovat obsluhu

Zdroje preruseni

Zdroje preruseni

- Vodice zo zariadeni napojene na specialnu zbernicu (bud priamo do CPU alebo cez cip, ktory dalej spracuva zdroj prerusenia)
- Na SiFive zakladnej doske (RISC-V CPU) prerusenia zariadeni idu cez PLIC
- PLIC dalej smeruje vzniknute prerusenie na to jadro CPU, ktore moze prerusenie obsluzit
  - CPU moze mat vypnute spracovanie preruseni
  - Ak nie je ziadne CPU dostupne, PLIC uchovava prerusenie, pokym nejake CPU nezapne obsluhu

Mechanizmus obsluhy preruseni

Mechanizmus obsluhy preruseni

- Prerusenie informuje jadro o tom, ze nejaky hw vyzaduje pozornost
- Ovladac (kod v jadre) vie, ako obsluhu zariadenia uskutocnit
- Najjednoduchsia obsluha je priame volanie ovladaca z obsluhy prerusenia (tak to robi xv6), ale je mozna aj sofistikovanejsia schema – pre obsluhu sa vytvori a naplanuje vlastne vlakno jadra, pripadne sa obsluha viacerych preruseni spoji do jednej atd

Mechanizmus obsluhy preruseni

- Obsluha prerusenia NEBEZI v kontexte procesu

- Co to znamena pre xv6
  - myproc() moze vratit 0
  - copyin(), copyout() sa nedaju pouzit
  - Preco?

Programovanie zariadenia

- Zvacsa sa pouziva mapovanie pamate
- Pomocou virtualnych adries je mozne pristupovat priamo k internym registrom (pamati) samotneho zariadenia
- Priamo sa pouzivaju instrukcie load/store
- Programovanie UART vid napr. byterunner.com/16550.html

Pripadova studia xv6: $ ls

Pripadova studia xv6: $ ls

- Vypis znaku $ na konzolu
  - Ovladac posle znak do FIFO odosielacej fronty UART zariadenia
  - UART vygeneruje prerusenie, ked sa znak posle, cim informuje ovladac, ze moze poslat dalsi znak
- Nacitanie a vypis ‘ls’
  - Pouzivatel stlaci klavesu, co sposobi prerusenie UART
  - Ovladac nacita znak z FIFO prijimacej fronty UART

Ako kernel rozozna zariadenia

Ako kernel rozozna zariadenia

- Kazde zariadenie ma jedinecne cislo zdroja IRQ (Interrupt ReQuest)

- IRQ je definovane hw platformou (medzi platformami sa zvacsa lisia)
  - V Qemu ma UART0 pridelene IRQ 10 (vid kernel/memlayout.h)
  - Na doske SiFive ma UART0 ine IRQ cislo

Podpora preruseni na RISC-V CPU

Podpora preruseni na RISC-V CPU

- sie (supervisor interrupt enable register)
  - bit pre sw prerusenie, externy zdroj a casovac
- sip (supervisor interrupt pending register)
  - bit pre sw prerusenie, externy zdroj a casovac
- sstatus (supervisor status register)
  - jeden bit urcujuci, ci su prerusenia zapnute

Podpora preruseni na RISC-V CPU

- scause (supervisor cause register)
  - Cislo prerusenia
- stvec (supervisor trap vector register)
  - Adresa kodu obsluhy preruseni
- mideleg (machine interrupt delegate register)
  - Vsetky vynimky (teda aj prerusenia) sa standardne obsluhuju obsluhou v M-mode
  - Register mideleg umoznuje nastavit automaticke smerovanie preruseni do obsluhy v S-mode

Inicializacia preruseni v xv6

kernel/start.c start()

w_sie(r_sie() | SIE_SEIE|SIE_STIE|SIE_SSIE)

kernel/main.c main()

consoleinit()

uartinit()

plicinit()

scheduler()

intr_on()

w_sstatus(r_sstatus() | SSTATUS_SIE)

Zobrazenie ‘$’

- user/init.c main()
  - Init otvara fd 0, 1, 2 pre konzolovy vstup/vystup
  - Shell ich pomocou mechanizmu fork() zdedi

Zobrazenie ‘$’

- user/init.c main()
  - Init otvara fd 0, 1, 2 pre konzolovy vstup/vystup
  - Shell ich pomocou mechanizmu fork() zdedi

- Vsetky zariadenia sa v Unix-like systemoch interpretuju ako subory
  - printf() → putc() → write()

Zobrazenie ‘$’

sys_write()

filewrite()

consolewrite() v kernel/console.c

uartputc()

- Vlozenie znaku do FIFO UART
- Navrat do userspace
- Zaroven v tom istom case UART posiela znak na konzolu

Zobrazenie ‘$’

- Shell v getcmd() vyvolal sys_read(), caka na vstup

- UART po dokonceni posielania znaku na konzolu vygeneruje prerusenie
- PLIC posunie prerusenie nejakemu jadru CPU
- Co urobi CPU?

Co robi CPU pri prijati prerusenia?

Co robi CPU pri prijati prerusenia?

1. Ak ide o trap zo zariadenia a SIE bit je 0, nepokracuj v spracovani

Co robi CPU pri prijati prerusenia?

1. Ak ide o trap zo zariadenia a SIE bit je 0, nepokracuj v spracovani
2. Vypni spracovanie preruseni vynulovanim SIE

Co robi CPU pri prijati prerusenia?

1. Ak ide o trap zo zariadenia a SIE bit je 0, nepokracuj v spracovani
2. Vypni spracovanie preruseni vynulovanim SIE
3. Skopiruj $pc do $sepc

Co robi CPU pri prijati prerusenia?

1. Ak ide o trap zo zariadenia a SIE bit je 0, nepokracuj v spracovani
2. Vypni spracovanie preruseni vynulovanim SIE
3. Skopiruj $pc do $sepc
4. Uchovaj aktualny mod (user alebo supervisor) do bitu SPP v $sstatus

Co robi CPU pri prijati prerusenia?

1. Ak ide o trap zo zariadenia a SIE bit je 0, nepokracuj v spracovani
2. Vypni spracovanie preruseni vynulovanim SIE
3. Skopiruj $pc do $sepc
4. Uchovaj aktualny mod (user alebo supervisor) do bitu SPP v $sstatus
5. Nastav $scause podla zdroja prerusenia

Co robi CPU pri prijati prerusenia?

1. Ak ide o trap zo zariadenia a SIE bit je 0, nepokracuj v spracovani
2. Vypni spracovanie preruseni vynulovanim SIE
3. Skopiruj $pc do $sepc
4. Uchovaj aktualny mod (user alebo supervisor) do bitu SPP v $sstatus
5. Nastav $scause podla zdroja prerusenia
6. Nastav mod CPU na supervisor

Co robi CPU pri prijati prerusenia?

1. Ak ide o trap zo zariadenia a SIE bit je 0, nepokracuj v spracovani
2. Vypni spracovanie preruseni vynulovanim SIE
3. Skopiruj $pc do $sepc
4. Uchovaj aktualny mod (user alebo supervisor) do bitu SPP v $sstatus
5. Nastav $scause podla zdroja prerusenia
6. Nastav mod CPU na supervisor
7. Skopiruj $stvec do $pc

Co robi CPU pri prijati prerusenia?

1. Ak ide o trap zo zariadenia a SIE bit je 0, nepokracuj v spracovani
2. Vypni spracovanie preruseni vynulovanim SIE
3. Skopiruj $pc do $sepc
4. Uchovaj aktualny mod (user alebo supervisor) do bitu SPP v $sstatus
5. Nastav $scause podla zdroja prerusenia
6. Nastav mod CPU na supervisor
7. Skopiruj $stvec do $pc
8. Pokracuj vykonavanim instrukcie podla $pc

Co robi CPU pri prijati prerusenia?

- $stvec obsahuje bud adresu kernelvec() alebo usertrap() (podla toho, ci je prerusenie vyvolane z user alebo kernel priestoru)

- Rovnaky mechanizmus sa pouziva aj pre vynimky a instrukciu ecall (systemove volanie)

Co robi CPU pri prijati prerusenia?

kernelvec()/usertrap() volaju devintr()

- ak ide o externe prerusenie
  - plic_claim() zisti, o ktore zariadenie ide
  - ak UART, uartintr()
    - pokym je znak na vstupe, vypis ho
    - posli na vystup aj znaky z vyrov. pamate
  - plic_complete()

return z kernelvec()/usertrap() obnovi prerusene vykonavanie kodu

Viacere prerusenia sucasne

- Co ked sa v jednom case vyskytne viacero preruseni?

Viacere prerusenia sucasne

- Co ked sa v jednom case vyskytne viacero preruseni?

- PLIC zabezpecuje, ze kazde zariadenie moze vygenerovat iba 1 prerusenie, pokym nie je obsluha dokoncena
- To znamena, ze sucasne sa mou vyskytnut prerusenia od roznych zariadeni

Viacere prerusenia sucasne

- PLIC dokaze pridelit (alebo skor CPU si dokaze prevziat od PLIC vo funkcii plic_claim()) spracovanie prerusenia roznym jadram CPU

Viacere prerusenia sucasne

- PLIC dokaze pridelit (alebo skor CPU si dokaze prevziat od PLIC vo funkcii plic_claim()) spracovanie prerusenia roznym jadram CPU
- Takze spracovanie viacerych preruseni MOZE prebiehat sucasne! (t.j. paralelne)

Viacere prerusenia sucasne

- PLIC dokaze pridelit (alebo skor CPU si dokaze prevziat od PLIC vo funkcii plic_claim()) spracovanie prerusenia roznym jadram CPU
- Takze spracovanie viacerych preruseni MOZE prebiehat sucasne! (t.j. paralelne)
- Ak ziadne jadro CPU neprevezne prerusenie na spracovanie, prerusenie ostava nespracovane (angl. *pending*), pokym ho niektore jadro nespracuje

Prerusenia a konkurentnost

- Prerusenia vnasaju problematiku viacerych typov konkurencie (angl. *concurrency*) vykonavania cinnosti

Prerusenia a konkurentnost

- Prerusenia vnasaju problematiku viacerych typov konkurencie (angl. *concurrency*) vykonavania cinnosti

1. Medzi zariadenim a CPU (problem producent/konzument)

Prerusenia a konkurentnost

- Prerusenia vnasaju problematiku viacerych typov konkurencie (angl. *concurrency*) vykonavania cinnosti

1. Medzi zariadenim a CPU (problem producent/konzument)
2. Prerusenie uzivatelskeho programu OK, ale co prerusenie kernel kodu? Napr. userret() (riesenie – vypinanie preruseni, aby sme dosiahli atomicitu operacie)

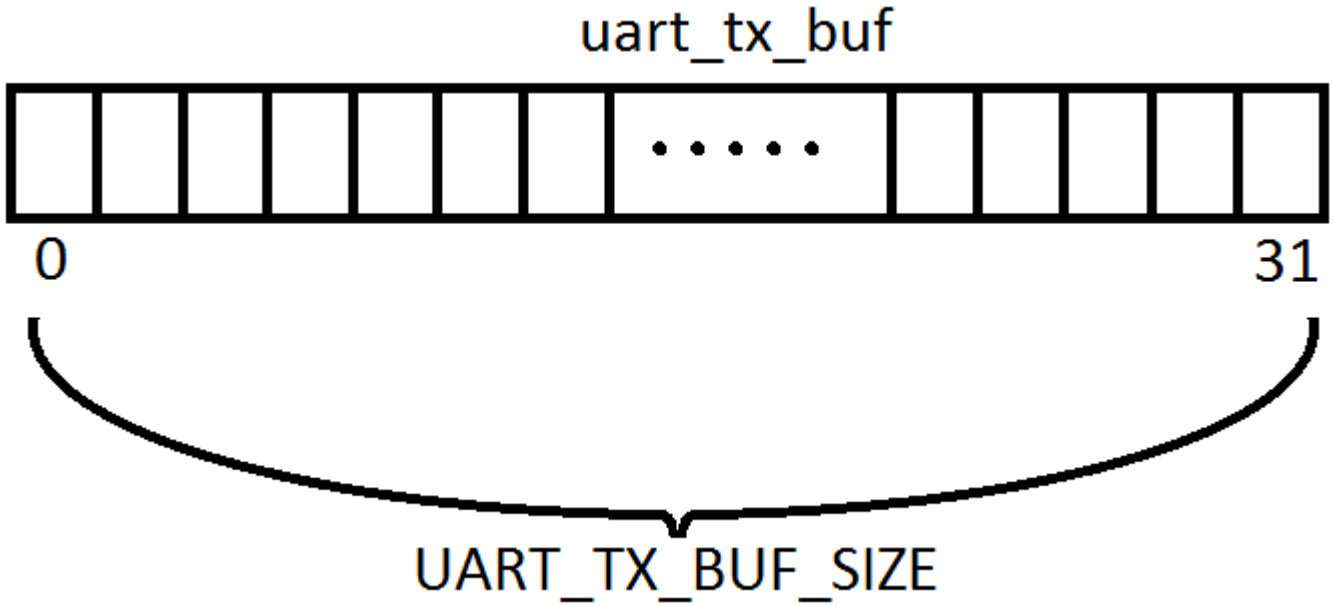
Prerusenia a konkurentnost

- Prerusenia vnasaju problematiku viacerych typov konkurencie (angl. *concurrency*) vykonavania cinnosti

1. Medzi zariadenim a CPU (problem producent/konzument)
2. Prerusenie uzivatelskeho programu OK, ale co prerusenie kernel kodu? Napr. userret() (riesenie – vypinanie preruseni, aby sme dosiahli atomicitu operacie)
3. Paralelne vykonavanie roznych casti kodu vyuzivajuce tu istu pamat (riesenie - zamky)

1. Producent/konzument

- Napriklad vypisovanie na monitor
  - Shell je producent
  - Zariadenie UART je konzument



1. Producent/konzument

- sys_write()→…→uartputc() v kernel/uart.c
  - Vklada do uart_tx_buf
  - Ak je buf plny, caka (stav procesu *SLEEPING*)
  - Bezi v kontexte procesu!!! (systemove volanie)

- devintr()→…→uartintr()→uartstart()
  - Vybera z uart_tx_buf a posiela na zariadenie
  - Ak je buf prazdny, nic neurobi
  - Budi z cakania producentov!
  - Vyvolane z devintr() nebezi v kontexte procesu!!!

1. Producent/konzument

- Uz vieme, ako sa vypise prompt ‘$ ‘
- Teraz sa pozrieme na nacitavanie znakov (napr. ‘ls’)

1. Producent/konzument

- Uz vieme, ako sa vypise prompt ‘$ ‘
- Teraz sa pozrieme na nacitavanie znakov (napr. ‘ls’)

- Nacitanie znakov z klavesnice
  - shell je konzument (sys_read())
  - klavesnica je producent (pri stlaceni sa generuje hw prerusenie)
  - Prislusny kod xv6 v kernel/console.c

1. Producent/konzument

- sys_read()→fileread()→consoleread()

1. Producent/konzument

- sys_read()→fileread()→consoleread()
  - Vyuziva cons.buf
  - Ak je buffer prazdny, proces sa uspi
  - Vola sa v kontexte procesu!!!

1. Producent/konzument

- sys_read()→fileread()→consoleread()
  - Vyuziva cons.buf
  - Ak je buffer prazdny, proces sa uspi
  - Vola sa v kontexte procesu!!!

- devintr()→…→uartintr()→consoleintr()
  - Vzdy mimo kontext procesu!!!
  - Vklada do cons.buf (co ak je buf plny?)
  - Budi konzumentov cakajucich na vstup z klavesnice
    - Za akej podmienky nastane prebudenie?

2. Prerusenie prerusi beziaci kod

- Co v pripade, ze nejake prerusenie prerusi vykonavany kod?

2. Prerusenie prerusi beziaci kod

- Co v pripade, ze nejake prerusenie prerusi vykonavany kod?

- Majme napriklad kod, ktory alokuje na zasobniku miesto a ulozi tam navratovu adresu:
  1. addi sp, sp, -48
  2. sd ra, 40(sp)
- Moze sa vykonat nejaky kod MEDZI riadkami 1 a 2?

2. Prerusenie prerusi beziaci kod

- Co v pripade, ze nejake prerusenie prerusi vykonavany kod?

- Majme napriklad kod, ktory alokuje na zasobniku miesto a ulozi tam navratovu adresu:
  1. addi sp, sp, -48
  2. sd ra, 40(sp)
- Moze sa vykonat nejaky kod MEDZI riadkami 1 a 2?
  - Ano, obsluha preruseni! Napriklad casovac, uart…

2. Prerusenie prerusi beziaci kod

- Co “hrozi” v takom pripade uzivatelskemu procesu?

2. Prerusenie prerusi beziaci kod

- Co “hrozi” v takom pripade uzivatelskemu procesu?
  - Vykonavanie obsluhy pobezi v kernel priestore
  - Stav uzivatelskeho programu bude obnoveny v tej podobe, v akej bol pri vyvolani vynimocneho stavu sposobeneho prerusenim

2. Prerusenie prerusi beziaci kod

- Co “hrozi” v takom pripade uzivatelskemu procesu?
  - Vykonavanie obsluhy pobezi v kernel priestore
  - Stav uzivatelskeho programu bude obnoveny v tej podobe, v akej bol pri vyvolani vynimocneho stavu sposobeneho prerusenim
- Co “hrozi” kodu jadra? Je to podobne jednoduche, ako v pripade uzivatelskeho kodu?

2. Prerusenie prerusi beziaci kod

- Majme nasledovne kody jadra: beziaci kod jadra a kod vykonavany obsluhou prerusenia

```
beziaci_kod:                    obsluha_prerusenia:
    x = 0
    if (x == 0)                     x = 1
        f()
```

- Ako je to s volanim funkcie f()?

2. Prerusenie prerusi beziaci kod

- Aby sa funkcia f() vzdy zarucene vykonala, nesmie nastat prerusenie!
- Preco?

2. Prerusenie prerusi beziaci kod

- Aby sa funkcia f() vzdy zarucene vykonala, nesmie nastat prerusenie!
- Preco?
  - Obsluha prerusenia moze byt vyvolana iba na hranici instrukcii, nie uprostred vykonavania nejakej instr.
  - Predpokladame, ze nastavenie premennej (x = 0) a testovanie premennej (if x == 0) su minimalne 2 instrukcie!
  - Ak je na CPU povolena obsluha preruseni, medzi tymito dvoma riadkami kodu sa moze spustit obsluha prerusenia, ktora zmeni hodnotu premennej x!

2. Prerusenie prerusi beziaci kod

- Aby sa funkcia f() vzdy zarucene vykonala, nesmie nastat prerusenie!

- Ako zabezpecit “atomicke” vykonanie nejakeho bloku instrukcii (riadkov kodu)?

2. Prerusenie prerusi beziaci kod

- Aby sa funkcia f() vzdy zarucene vykonala, nesmie nastat prerusenie!

- Ako zabezpecit “atomicke” vykonanie nejakeho bloku instrukcii (riadkov kodu)?
- Vypnutim spracovania instrukcii
  - Vid funkcia kernel/riscv.h: intr_off()
  - w_sstatus(r_sstatus() & ~SSTATUS_SIE);

2. Prerusenie prerusi beziaci kod

- Aby sa funkcia f() vzdy zarucene vykonala, nesmie nastat prerusenie!

- Ako zabezpecit “atomicke” vykonanie nejakeho bloku instrukcii (riadkov kodu)?
- Vypnutim spracovania instrukcii
  - Vid funkcia kernel/riscv.h: intr_off()
  - w_sstatus(r_sstatus() & ~SSTATUS_SIE);
- Kde v kode sa tato funkcia vyuziva? Moze kernel obsluhovat prerusenie v kode trampoliny?

2. Prerusenie prerusi beziaci kod

- Vratme sa k prikladu nacitania vstupu z klavesnice – v akej funkcii sa nachadza kod jadra?

2. Prerusenie prerusi beziaci kod

- Vratme sa k prikladu nacitania vstupu z klavesnice – v akej funkcii sa nachadza kod jadra?

$ (shell je v sys_read(), aby ziskal vstup z klav.)

2. Prerusenie prerusi beziaci kod

- Vratme sa k prikladu nacitania vstupu z klavesnice – v akej funkcii sa nachadza kod jadra?

$ (shell je v sys_read(), aby ziskal vstup z klav.)

usertrap() – vyvolane sys volanim (ecall)

w_stvec((uint64)kernelvec) !!!!!!!

…

consoleread()

sleep()

scheduler()

intr_on()

2. Prerusenie prerusi beziaci kod

$ l (pouzivatel stlacil klavesu ‘l’, UART prerusenie)

2. Prerusenie prerusi beziaci kod

$ l (pouzivatel stlacil klavesu ‘l’, UART prerusenie)

kernelvec() – pretoze $stvec obsahuje tuto adresu!!!

– na aky zasobnik sa ulozia registre CPU?

2. Prerusenie prerusi beziaci kod

kernelvec()

kerneltrap()

devintr()

uartintr()

c = uartgetc()

consoleintr(c)

obsluha specialnych sekvencii (ctrl)

poslanie znaku ‘l’ na vystup (uartput_sync())

vlozenie c do cons.buf

zobudenie konzumentov v consoleread()

navrat z devintr()

navrat z kerneltrap()

obnovenie stavu registrov CPU

sret

2. Prerusenie prerusi beziaci kod

kernelvec()

kerneltrap()

devintr()

uartintr()

c = uartgetc()

consoleintr(c)

obsluha specialnych sekvencii (ctrl)

poslanie znaku ‘l’ na vystup (uartput_sync())

vlozenie c do cons.buf

zobudenie konzumentov v consoleread()

navrat z devintr()

navrat z kerneltrap()

obnovenie stavu registrov CPU

sret

KAM sa vrati tok vykonavania instrukciou sret?

2. Prerusenie prerusi beziaci kod

- Kam sa vrati tok riadenia instrukciou sret?

- Tam, kde prislo k preruseniu beziaceho kodu pri prichode prerusenia
- V nasom pripade to je cyklus vo funkci scheduler()

- Vid pomocou `make CPUS=1 qemu-gdb`

3. Konkurentny pristup k datam

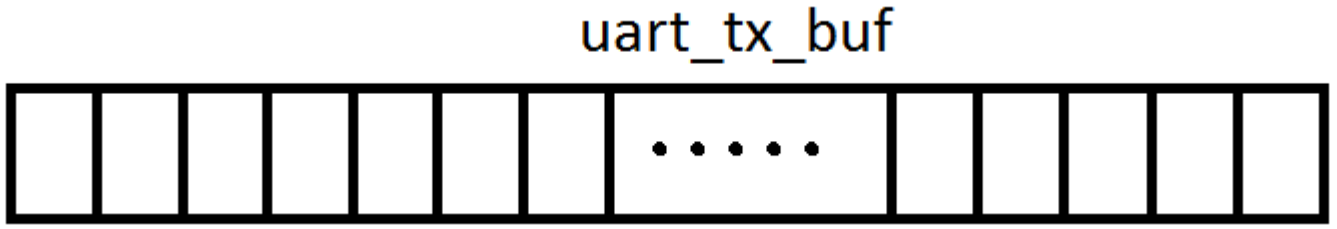
- Poslednou urovnou konkurencie je pristup k tym istym pamatovym oblastiam konkurentne/paralelne z roznych tokov vykonavania kodu

3. Konkurentny pristup k datam

- Priklad
  - majme dva uzivatelske programy, kazdy sa vykonava na samostatnom jadre CPU
  - nech sa oba programy v tom istom casovom okamihu pokusia vykonat printf(“ahoj %d\n”, pid)
  - kernel/uartc.c: uartputc()

3. Konkurentny pristup k datam

- Priklad
  - majme dva uzivatelske programy, kazdy sa vykonava na samostatnom jadre CPU
  - nech sa oba programy v tom istom casovom okamihu pokusia vykonat printf(“ahoj %d\n”, pid)
  - kernel/uartc.c: uartputc()



3. Konkurentny pristup k datam

- Riesenie – pouzitie zamkov (angl. *lock*)

- Vid funkcie acquire() a release() v kernel/uart.c: uartputc()

3. Konkurentny pristup k datam

- Riesenie – pouzitie zamkov (angl. *lock*)

- Vid funkcie acquire() a release() v kernel/uart.c: uartputc()

- Zamky sa vyuzivaju na vynutenie vykonavania istej casti kodu iba jedinym tokom riadenia
- Priklad zo zivota: turniket
- Viac informacii o zamkoch v xv6 bude na dalsej prednaske

Vyvoj preruseni

- Kedysi bol tento pristup navrhnuty a vyvinuty, aby urychlil cinnost CPU
- V sucasnosti su prerusenia prilis pomale pre niektore zariadenia
  - Napr. gigabit ethernet dokaze preniest 1.5 milion paketov za sekundu
  - To je viac nez 1 za mikrosekundu
  - Spracovanie prerusenia trva radovo v mikrosekundach
  - Ako potom taketo zariadenia obsluhovat?

Vyvoj preruseni

- Ak je obsluha prerusenia prilis pomala klasickym pristupom, je mozne vyuzit techniku “dopytovania sa”, tzv. *polling*

Vyvoj preruseni

- Ak je obsluha prerusenia prilis pomala klasickym pristupom, je mozne vyuzit techniku “dopytovania sa”, tzv. *polling*
- CPU neustale v cykle kontroluje, ci niektore zariadenie nevyzaduje pozornost
  - Toto cakanie v cykle je neefektivne (nevyuzije sa CPU naplno), ak je zariadenie pomale
  - Jeden priklad v xv6: uartputc_sync()
  - Ale ak je zariadenie mega super rychle, setri sa cas CPU (ziadna zmena kontextu atd)

Vyvoj preruseni

- Ak je obsluha prerusenia prilis pomala klasickym pristupom, je mozne vyuzit techniku “dopytovania sa”, tzv. *polling*

- Preco alebo kedy pouzivat tuto techniku?

Vyvoj preruseni

- Ak je obsluha prerusenia prilis pomala klasickym pristupom, je mozne vyuzit techniku “dopytovania sa”, tzv. *polling*

- Preco alebo kedy pouzivat tuto techniku?
- Ak je generovanie udalosti tak rychle, ze musia neustale cakat na spracovanie – vtedy nie je nutne o vygenerovani udalosti informovat, pretoze vieme, ze vzdy je k dispozicii nejaka udalost cakajuca na spracovanie

Prerusenia vs *polling*

Prerusenia vs *polling*

- Pre zariadenia, ktore chrlia udalosti – polling
- Pre pomale zariadenia (typu klavesnica) – irq

Prerusenia vs *polling*

- Pre zariadenia, ktore chrlia udalosti – polling
- Pre pomale zariadenia (typu klavesnica) – irq

- Automaticke prepinanie medzi oboma modmi cinnosti
- Presmerovanie spracovania preruseni do user priestoru
  - Vypadky stranok
  - Obsluha nejakych zariadeni (napr. disk, siet)

Domace citanie

Chapter 5

Interrupts and device drivers

xv6: a simple, Unix-like teaching operating system