OS MMXX

MIT ;)

https://pdos.csail.mit.edu/6.828/2020
a
Peter Tomcsányi: Plánovanie procesov a vlákien

Planovanie procesov

Tema

Tema

- V sucasnosti su viacjadrove CPU bezne
- Ale pocet jadier je maly
- Pouzivatel chce spustat vela programov

- Treba vymysliet, ako maly pocet jadier CPU zdielat medzi vela procesov

Tema

- Zdielania by malo byt pre proces transparentne

- Zvacsa sa pouziva abstrakcia – kazdemu procesu sa vytvori iluzia procesora, ktory ma iba sam pre seba
- Ide o multiplexing procesov na hw vypoctoveho systemu

Motivacia

- Preco pisat os s podporou behu viacerych uloh sucasne?

Motivacia

- Preco pisat os s podporou behu viacerych uloh sucasne?

- Poziadavka pouzivatelov (hudba, net…)
- Poziadavka algoritmu (sito na prvocisla)
- Poziadavka efektivity (urychlenie vypoctu)

Vlakno (Thread)

- Vlakno je abstrakcia na zjednodusenie programovania

Vlakno (Thread)

- Vlakno je abstrakcia na zjednodusenie programovania

- Vlakno = nezavisle seriove vykonavanie (registre, pc, zasobnik)
- Dve hlavne strategie nasadenia vlakien

Vlakno (Thread)

- Vlakno je abstrakcia na zjednodusenie programovania

- Vlakno = nezavisle seriove vykonavanie (registre, pc, zasobnik)
- Dve hlavne strategie nasadenia vlakien:
  1. Viac CPU, na kazdom CPU bezi 1 vlakno
  2. Kazde CPU “prepina” medzi viacerymi vlaknami (v 1 case sa ovsem vykonava iba 1 vlakno na 1 CPU)

Vlakno (Thread)

- Vlakna mozu, ale nemusia zdielat pamat

- Linux: vlakna uzivatelskeho procesu zdielaju pamat
- Xv6:
  - Vlakna jadra: zdielaju spolocnu pamat
  - Vlakna user programu:
    - Kazdy user program pozostava z 1 vlakna
    - Vlakna NEzdielaju spolocnu pamat

Multiplexing

- Multiplexing vytvara iluziu podobne ako je to v pripade pamate pomocou strankovania

- Ide o prepnutie kodu na procesore
- V xv6 sa prepnutie robi v 2 pripadoch
  - Mechanizmus sleep – wakeup
  - Vynutene prepnutie procesu, ak nevyuziva sleep

Multiplexing

- Ako urobit prepnutie jedneho procesu na iny?
- Ako to urobit tak, aby o tom proces “nevedel”? (Xv6 vyuziva prerusenie casovaca)
- Ako urobit prepnutie bezpecnym, ked sa o to pokusaju sucasne viacere jadra naraz?
- Ako spravne uvolnit pamat ukonceneho procesu (nemoze to urobit on sam, vzhladom na zasobnik jadra…)

Multiplexing

- Ako sa vysporiadat s procesmi, ktore robia iba vypocty a ziadne systemove volania?
- Co ma robit procesor, ked planovac nenajde ziaden vykonatelny proces?
- …

Vypoctovo orientovane procesy

Vypoctovo orientovane procesy

- Kazde CPU ma casovac, ktory v pravidelnych intervaloch informuje o “tikoch”
- Kernel vyuziva prerusenie casovaca na prerusenie behu vypoctovo orientovaneho procesu

Vypoctovo orientovane procesy

- Kazde CPU ma casovac, ktory v pravidelnych intervaloch informuje o “tikoch”
- Kernel vyuziva prerusenie casovaca na prerusenie behu vypoctovo orientovaneho procesu

- Ide o tzv. “preemptivne” planovanie (vynutene)
- Jestvuje aj “kooperativne”, pri ktorom samotny proces sa musi vzdat dobrovolne procesora

Vlakna v xv6

Vlakna v xv6

- Brutalny obrazok vlakien xv6 c.1

Vlakna v xv6

- Brutalny obrazok vlakien xv6 c.1

- Kazdy proces ma 2 vlakna
  - Uzivatelske (kod user programu)
  - Kernel (systemove volania, obsluha preruseni)
- Vsetky vlakna v kernel priestore zdielaju VM (dosledok: jadro xv6 je multivlaknove, konkurentne)

Vlakna v xv6

- V prednaske pouzivame pojem “proces”, “vlakno jadra” a “vlakno” ako synonyma

- Vo vseobecnosti sa pojmy “davka”, “proces” a “vlakno” rozlisuju
  - Vlakno – najmensia (nedelitelna) jednotka toku riadenia (s ktorou moze manipulovat planovac)
  - Proces – zoskupenie vlakien zdielajucich spolocny pamatovy priestor
  - Davka – zoskupenie procesov so spolocnymi charakteristikami (napr. terminal, vlastnik)

Vlakna v xv6

- Brutalny obrazok vlakien xv6 c.2

Vlakna v xv6

- Brutalny obrazok vlakien xv6 c.2

- TF (trapframe) obsahuje registre uzivatelskeho programu
- CTX (context) obsahuje registre procesora, ktore sa mozu medzi volaniami funkcii menit (neobnovuju sa zo zasobnika)
  - ide o tzv. *callee saved* registre v RISC-V architekture
  - *caller saved* registre uklada C kod na zasobnik

Vlakna v xv6

- Brutalny obrazok vlakien xv6 c.2

- Prechod z kodu jedneho procesu na kod ineho procesu je v xv6 nepriamy
  - user v. →kernel v. (user registre sa ulozia do TF)
  - kernel vlakno→scheduler vlakno(kernel registre sa uschovaju v CTX)
  - scheduler vlano→kernel vlakno (kernel registre sa obnovia z CTX)
  - kernel v.→user v.(user registre sa obnovia z TF)

Vlakna v xv6

- Co je v terminologii xv6 prepnutie kontextu (*context switch*)?
  - Zmena toku riadenia z jedneho vlakna kernelu na ine
  - Nejedna sa o zmenu kernel→user alebo user→kernel
- Zmena user vlakna na ine user vlakno nie je mozna, nakolko kazdy user program pozostava z jedineho vlakna

Cvicenie

- Prva uloha cvicenia – rozsirenie user programov o viacere vlakna

- Aktualne je v xv6 mapovanie vlakien user:kernel 1:1
- Cielom cvicenia je dosiahnut stav n:1
  - Ake su vyhody viacerych vlakien?
  - Ake su obmedzenia riesenia n:1?
- Vid brutalny obrazok c.3

Cvicenie

- Prva uloha cvicenia – rozsirenie user programov o viacere vlakna

- Prepinanie vlakien v user programoch je *kooperativne*
- Samotne vlakno musi vyvolat funkciu thread_yield(), aby sa vzdalo procesora
  - Kernel nic “nevie” o vlaknach implementovanych v user priestore

Vlakna planovaca

Vlakna planovaca

- Vzdy jedno na 1 CPU
  - Zasobnik start.c:stack0, kontext proc.h:struct cpu
- Kernel vlakna user programu
  - Vzdy sa prepnu do planovaca toho CPU, na ktorom dane vlakno bezi
  - Planovac hlada na beh RUNNABLE vlakno
  - Nikdy sa nerobi priamy prechod medzi kernel vlaknami user programov!

Vlakna planovaca

- Preco pouzit “medzistupen” planovaca (ked na cviceni sa prepinanie medzi user vlaknami robi priamo)?

Vlakna planovaca

- Preco pouzit “medzistupen” planovaca (ked na cviceni sa prepinanie medzi user vlaknami robi priamo)?
  - Zjednodusenie navrhu a kodu pri ukoncovani procesu (musi sa uvolnit zasobnik kernel vlakna!)
  - Planovac neustale v cykle hlada spustitelne vlakno; ak take nenajde, “zaspi” (dorucenie prerusenia ho “zobudi”)
  - Co ak bezi v systeme menej vlakien ako je CPU? Aky zasobnik pouzije obsluha zvysnych CPU?

Vlakna planovaca

- Co ostava stale zachovane (invarianty)

Vlakna planovaca

- Co ostava stale zachovane (invarianty)

1. Jedno jadro CPU v jednom case vykonava iba jedno vlakno (bud scheduler alebo kernel vlakno user programu)
2. Bud vlakno bezi na prave jednom jadre CPU, alebo su jeho registre uchovane v kontexte
3. Ak vlakno jadra nie je prave vykonavane, jeho kontext uchovava stav z volania swtch()

Struktura proc v xv6

- p→trapframe
- p→context
- p→kstack
- p→state
- p→lock

Struktura proc v xv6

- p→trapframe: uchovava registre user vlakna
- p→context: uchovava registre kernel vlakna
- p→kstack: ukazuje na kernel zasobnik vlakna
- p→state: running, runnable, sleeping, …
- p→lock: chrani integritu p→state a inych

Exkurz *lock*

- Co je zamok (*lock*)? Synchronizacny mechanizmus umoznujuci implementovat serializaciu (radenie za sebou)

Exkurz *lock*

- Co je zamok (*lock*)? Synchronizacny mechanizmus umoznujuci implementovat serializaciu (radenie za sebou)

- Ako?
  - Nadobuda 2 stavy: odomknuty, zamknuty
  - Manipulacia so stavmi pomocou metod: acquire() a release()

Exkurz *lock*

- Metoda acquire()
  - Ak je zamok volny/odomknuty, volajuci moze pokracovat dalej vo vykonavani kodu a zaroven sa stav zamku zmeni na “zamknuty”
  - Ak je zamok obsadeny/zamknuty, volajuci nepokracuje vo vykonavani kodu dalej, ale caka na uvolnenie/odomknutie zamku

Exkurz *lock*

- Metoda acquire()
  - Ak je zamok volny/odomknuty, volajuci moze pokracovat dalej vo vykonavani kodu a zaroven sa stav zamku zmeni na “zamknuty”
  - Ak je zamok obsadeny/zamknuty, volajuci nepokracuje vo vykonavani kodu dalej, ale caka na uvolnenie/odomknutie zamku
- Cakanie je pre volajuceho transparentne (nevie o nom, nijako sa o to nestara)
- Podla typu cakania rozoznavame ADT *Spinlock* a *Sleeplock*

Exkurz *lock*

- Rozdiel medzi Sleeplock a Spinlock
  - Sleeplock pri cakani na uvolnenie zamku nevytazuje CPU; stav procesu sa v xv6 oznaci ako SLEEPING, takze ho planovac nebude planovat
  - Spinlock pri cakani vyuziva cyklus (v cykle neustale testuje dostupnost zamku)

Exkurz *lock*

- Rozdiel medzi Sleeplock a Spinlock
  - Sleeplock pri cakani na uvolnenie zamku nevytazuje CPU; stav procesu sa v xv6 oznaci ako SLEEPING, takze ho planovac nebude planovat
  - Spinlock pri cakani vyuziva cyklus (v cykle neustale testuje dostupnost zamku)
- Metoda release()
  - Meni stav zamku na volny/odomknuty
  - V pripade ADT *Sleeplock* zobudi procesy cakajuce (ak nejake vobec su) v metode acquire() na ziskanie zamku

Exkurz *lock*

- Vyuzitie zamkov
  1. Ochrana integrity dat
  2. Vylucny pristup ku zdroju

Exkurz *lock*

- Vyuzitie zamkov
  1. Ochrana integrity dat
  2. Vylucny pristup ku zdroju

- Kod medzi metodami acquire() a release() oznacujeme ako KO (kriticka oblast)
- Vykonavat ho moze vzdy iba JEDEN tok riadenia (vid prechod cez turniket – vzdy iba jeden)

Exkurz *lock*

- Vzhladom na ziskanie a uvolnenie zamku rozlisujeme 2 pristupy

1. Vlakno, ktore ziskalo zamok, zamok uvolnuje
2. Jedno vlakno ziska zamok, ine vlakno uvolnuje

Exkurz *lock*

- Vzhladom na ziskanie a uvolnenie zamku rozlisujeme 2 pristupy

1. Vlakno, ktore ziskalo zamok, zamok uvolnuje
   - Najcastejsi pripad vyuzitia zamkov, vid napriklad druha a tretia uloha cvicenia
2. Jedno vlakno ziska zamok, ine vlakno uvolnuje
   - Velmi ucinny mechanizmus na odovzdavanie “poverenia” (tokenu) – vid napr. Petriho siete
   - Napriklad prepinanie vlakien v jadre xv6

Ukazka prepnutia v xv6

- user/spin.c

- Dva procesy, ktore vytazuju procesor (tzv. *CPU-bound* procesy)
- Spustime qemu s iba 1 CPU
- Budeme pozorovat, ako xv6 urobi prepnutie medzi nimi

Ukazka prepnutia v xv6

- make CPUS=1 qemu-gdb
- V druhom okne spustime gdb a zadame `continue`
- V okne qemu spustime program spin

Ukazka prepnutia v xv6

- make CPUS=1 qemu-gdb
- V druhom okne spustime gdb a zadame `continue`
- V okne qemu spustime program spin

- Na vystupe vidime, ako sa striedaju napriek tomu, ze v kode je nekonecny cyklus
- Xv6 vynucuje ich striedanie na 1 CPU, ktore je k dispozicii

Ukazka prepnutia v xv6

- Dame breakpoint do prerusenia casovaca

- (gdb) Ctrl+c
- (gdb) b trap.c:207
- (gdb) c
- (gdb) finish
- (gdb) where

Ukazka prepnutia v xv6

- Kde sme? V usertrap(), po spracovani prerusenia z casovaca
- Aky uzivatelsky program bezal v case vyvolania prerusenia?
  - (gdb) print p->name
  - (gdb) print p->pid
  - (gdb) print/x *(p->trapframe)
  - (gdb) print/x p->trapframe->epc

Ukazka prepnutia v xv6

- Kde sme? V usertrap(), po spracovani prerusenia z casovaca
- Aky uzivatelsky program bezal v case vyvolania prerusenia?
  - (gdb) print p->name
  - (gdb) print p->pid
  - (gdb) print/x *(p->trapframe)
  - (gdb) print/x p->trapframe->epc
    - Pozrime do user/spin.asm, kde prislo k preruseniu uzivatelskeho kodu prerusenim casovaca…

Ukazka prepnutia v xv6

- Pomocou `step` sa presunme v gdb do funkcie yield()
- (gdb) next
- (gdb) print p->state

- O 2 riadky nizsie sa meni stav procesu z RUNNING na RUNNABLE
- Aby neprislo k zlej situacii (akej?), je potrebne pouzit zamok

Ukazka prepnutia v xv6

- (gdb) next 2
- (gdb) step // vojdeme do funkcie sched()
  - Najprv su kontroly, tie preskocime, aby sme sa dostali pred vykonanie swtch()
- (gdb) next 7 // pripadne este 1x `step`

- V tomto volani swtch() prepina kontext medzi kernel vlaknom a vlaknom planovaca

Ukazka prepnutia v xv6

- swtch()
  - Ulozi aktualne hodnoty registrov do prveho argumentu (p->context)
  - Obnovi hodnoty registrov ulozene na adrese druheho argumentu (c->context)
    - Pozri cpus[0].context.ra – obsahuje adresu, kam sa po vykonani swtch() preda riadenie
    - Pozri cpus[0].context.sp – obsahuje adresu vrcholu zasobnika, ktory sa pouzije
  - Ide o funkciu v ASM, aby nemusela pouzivat zasobnik

Ukazka prepnutia v xv6

- Podme do funkcie swtch
- (gdb) tbreak swtch
- (gdb) c
  - Sme vo funkci kernel/swtch.S
  - a0 obsahuje prvy argument (p->context)
  - a1 obsahuje druhy argument (cpus[0].context)
  - Funkcia uklada hodnoty registrov CPU do 1. arg
  - Nacitava hodnoty registrov CPU z 2. arg
  - Potom vyvola navrat pomocou `ret`

Ukazka prepnutia v xv6

- Otazka 1
- swtch() neuklada ani neobnovuje $pc (program counter); ako potom “vie”, kde ma pokracovat vykonavanie po zmene kontextu?

- Otazka 2
- Preco swtch() uklada iba 14 registrov (ra, sp, s0 az s11) a ostatne nie?
  - Registre user vlakna su VSETKY odlozene v TF; tu hovorime o registroch kernel vlakna

Ukazka prepnutia v xv6

- Zobrazme si registre na zaciatku funkcie
  - (gdb) p/x $pc // swtch
  - (gdb) p/x $ra // sched
  - (gdb) p/x $sp // proc[pid-1].kstack+???
- Podme na koniec funkcie a zobrazme si ich znovu
  - (gdb) stepi 28 // mali by sme byt pred `ret`
  - (gdb) p/x $pc // swtch
  - (gdb) p/x $ra // scheduler !!!
  - (gdb) p/x $sp // stack0+???

Ukazka prepnutia v xv6

- (gdb) where
- (gdb) stepi
- Vykonavanie je v scheduler(), vo vlakne planovaca jadra 0, kod bezi na zasobniku tohto vlakna!

Ukazka prepnutia v xv6

- Pre vlakno planovaca sa “javi” beh kodu ako obycajny navrat z funkcie swtch()
  - Tuto musel planovac vyvolat niekedy v minulosti, cim sposobil “prepnutie” na kod vlakna jadra procesu
  - Toto predosle zavolanie swtch() ulozilo kontext vlakna planovaca (vsimni si argumenty funkcie – tu je to opacne ako v sched())
  - Premenna p ukazuje na proces, ktoreho beh bol preruseny

Ukazka prepnutia v xv6

- (gdb) print p->name
- (gdb) print p->pid
- (gdb) print p->state

Ukazka prepnutia v xv6

- Funkcia yield() ziskala zamok (lock), planovac ho teraz uvolni
- Z pohladu kodu sa javi, ze scheduler() zamkne aj odomkne
- Ale v skutocnosti
  - scheduler() zamkne, yield() odomkne
  - yield() zamkne, scheduler() odomkne
- Netypicke vyuzitie zamku – posunutie tokenu niekomu inemu (“prihraj loptu”)

Ukazka prepnutia v xv6

- Je mozne uvolnit zamok p→lock tesne pred zavolanim swtch()? (ci uz vo funkcii scheduler() alebo yield())

Ukazka prepnutia v xv6

- Je mozne uvolnit zamok p→lock tesne pred zavolanim swtch()? (ci uz vo funkcii scheduler() alebo yield())
  - Vyvolajme funkciu swtch() a CPU1 stihne ulozit iba par registrov (alebo aj ziaden), ale nie vsetky
  - Kedze p→status je RUNNABLE, tak CPU2 naplanuje tento proces na beh (v scheduler() sa vyvola swtch())
  - Vlakno jadra na CPU2 sa obnovi s poskodenym kontextom, co moze viest ku chybe (ktora sa veeeeeeeelmi tazko hlada)

Ukazka prepnutia v xv6

- Uloha zamku p→lock

Ukazka prepnutia v xv6

- Uloha zamku p→lock
- Atomicita nasledovnych operacii
  - p→state = RUNNABLE
  - Ulozenie registrov do p→context
  - Ukoncenie pouzivania zasobnika jadra p→kstack

Ukazka prepnutia v xv6

- Uloha zamku p→lock
- Atomicita nasledovnych operacii
  - p→state = RUNNABLE
  - Ulozenie registrov do p→context
  - Ukoncenie pouzivania zasobnika jadra p→kstack
- Atomicita a neprerusitelnost operacii
  - p→state = RUNNING
  - Presun hodnot registrov z p→context do CPU; nesmie nastat intr, lebo by sa kontext prepisal este neinicializovanymi hodnotami registrov CPU

Ukazka prepnutia v xv6

- Prejdime na miesto, kde scheduler() najde proces, ktory je RUNNABLE
  - (gdb) tbreak proc.c:474
  - (gdb) c
  - (gdb) print p->name // meno je rovnake… fork()
  - (gdb) print p->pid // ide o iny proces!!!

Ukazka prepnutia v xv6

- Prejdime na miesto, kde scheduler() najde proces, ktory je RUNNABLE
  - (gdb) tbreak proc.c:474
  - (gdb) c
  - (gdb) print p->name // meno je rovnake… fork()
  - (gdb) print p->pid // ide o iny proces!!!
- Pozrime, kde bude pokracovat vykonavanie vlakna tohto procesu v jadre po swtch()
  - (gdb) print/x p->context
  - (gdb) x/4i p->context->ra // funkcia sched()

Ukazka prepnutia v xv6

- Podme znovu do funkcie sched()
  - (gdb) tbreak swtch
  - (gdb) c
  - (gdb) stepi 28 // sme pred vykonanim instrukcie ret
  - (gdb) print/x $ra
  - (gdb) where
    - Sme po obsluhe prerusenia casovaca v kontexte ineho procesu ako sme zacali
    - Proces bol preruseny, zavolal yield(), sched(), swtch()
    - Teraz sa jeho beh obnovil a vrati sa do user priestoru

Ukazka prepnutia v xv6

- Poznamka1
  - Iba swtch() prepisuje kontexty (okrem inicializacie)
  - Iba sched() a scheduler() volaju swtch()
  - Preto plati, ze context.ra
    - vlakna jadra procesu vzdy ukazuje do sched()
    - vlakna planovaca vzdy ukazuje do scheduler()

Ukazka prepnutia v xv6

- Poznamka1
  - Iba swtch() prepisuje kontexty (okrem inicializacie)
  - Iba sched() a scheduler() volaju swtch()
  - Preto plati, ze context.ra
    - vlakna jadra procesu vzdy ukazuje do sched()
    - vlakna planovaca vzdy ukazuje do scheduler()
- Poznamka 2
  - sched()→swtch()→scheduler()→swtch()→sched()
  - Vo vseobecnosti sa nejedna o navrat do funkcie sched() rovnakeho vlakna jadra

Exkurz koprogram/korutina

- Korutina (koprogram, *co-routine*) vs podprogram

Exkurz koprogram/korutina

- Korutina (koprogram, *co-routine*) vs. podprogram
  - Podprogram po svojom zavolani zacne na zaciatku svojej definicie a 1x skonci; nedrzi stav premennych medzi jednotlivymi vyvolaniami

Exkurz koprogram/korutina

- Korutina (koprogram, *co-routine*) vs. podprogram
  - Podprogram po svojom zavolani zacne na zaciatku svojej definicie a 1x skonci; nedrzi stav premennych medzi jednotlivymi vyvolaniami
  - Koprogram pri svojom prvom spusteni zacne od zaciatku svojej definicie a potom “zdanlivo” skonci volanim ineho koprogramu; iny koprogram moze “odovzdat riadenie” do tohto bodu “skoncenia” prveho koprogramu. Medzi jednotlivymi obnoveniami behu udrziava stav premennych.

Exkurz koprogram/korutina

- Vzajomne vyvolanie korutin je v inom vztahu ako volanie obycajneho podprogramu
  - Pri obycajnom podprograme je tento volany, vztah je asymetricky (volajuci – volany)
  - Pri dvoch korutinach sa tieto navzajom volaju, vztah je symetricky (volajuci – volajuci alebo volany – volany, je to jedno)

Exkurz koprogram/korutina

- Vzajomne vyvolanie korutin je v inom vztahu ako volanie obycajneho podprogramu
  - Pri obycajnom podprograme je tento volany, vztah je asymetricky (volajuci – volany)
  - Pri dvoch korutinach sa tieto navzajom volaju, vztah je symetricky (volajuci – volajuci alebo volany – volany, je to jedno)
- Viac o synchronizacnych mechanizmoch, koprogramoch a inych veciach na volitelnom predmete PPaDS inzinierskeho studia

Korutiny v xv6

- sched() a scheduler() su navzajom korutiny
- Navzajom sa “poznaju” – “vedia”, ktora kam odovzdava riadenie a skadial riadenie pride
- Navzajom kooperuju v ramci zdielania stavu (p→lock a p→state)

Ako je to v jadre xv6?

- Je preemptivne planovanie platne aj pre beh vlakien jadra (nielen uzivatelskych procesov)?

Ako je to v jadre xv6?

- Je preemptivne planovanie platne aj pre beh vlakien jadra (nielen uzivatelskych procesov)?
- Ano, pri preruseni casovaca (vid kerneltrap())
- Kam sa ukladaju registre v tomto pripade?
  - Do p→trapframe sa nemozu (uzivatelsky kod)
  - Do p→context sa nemozu (pouzivaju ho iba korutiny sched() a scheduler() pri volani swtch())

Ako je to v jadre xv6?

- Je preemptivne planovanie platne aj pre beh vlakien jadra (nielen uzivatelskych procesov)?
- Ano, pri preruseni casovaca (vid kerneltrap())
- Kam sa ukladaju registre v tomto pripade?
  - Do p→trapframe sa nemozu (uzivatelsky kod)
  - Do p→context sa nemozu (pouzivaju ho iba korutiny sched() a scheduler() pri volani swtch())
  - Odpoved je v subore kernelvec.S – na aktualny zasobnik jadra

Ako je to v jadre xv6?

- Preco planovac cim skor zapina prerusenia pomocou intr_on()?

Ako je to v jadre xv6?

- Preco planovac cim skor zapina prerusenia pomocou intr_on()?
  - Co ked su vsetky procesy cakajuce (napr. na disk alebo konzolu)?
  - Zapnutie preruseni dava moznost zariadeniam signalizovat pripravenost dat, takze sa stav cakajucich vlakien moze zmenit na RUNNABLE
  - V opacnom pripade hrozi uviaznutie systemu

Ako je to v jadre xv6?

- Preco je tak prisna kontrola ohladom drzania zamkov v sched()? Konkretne, iba jeden jediny moze byt drzany, a to p→lock?

Ako je to v jadre xv6?

- Preco je tak prisna kontrola ohladom drzania zamkov v sched()? Konkretne, iba jeden jediny moze byt drzany, a to p→lock?
  - Suvisi to s principom, ako funguje spinlock; o tom este budeme hovorit na dalsej prednaske
  - V kratkosti prijmime fakt, ze funkcia acquire() musi bezat s vypnutymi preruseniami
  - Majme nasledovnu situaciu

Ako je to v jadre xv6?

- Pocas sched() P1 drzi spinlock L1
- Obnovi sa beh procesu P2, a ten sa pokusi vykonat acquire(L1)
- Kedze acquire() bezi s vypnutymi preruseniami
  - tak casovac nemoze dorucit prerusenie,
  - preto sa P2 nemoze vzdat CPU,
  - takze P1 nemoze byt naplanovany na beh,
  - a teda nemoze byt uvolneny zamok L1.
- Nastava uviaznutie (*deadlock*)

Planovacia politika

- Aku strategiu pri planovani implementuje xv6?

Planovacia politika

- Aku strategiu pri planovani implementuje xv6?
- “Ide piesen dokola” (*Round Robin*)

Planovacia politika

- Aku strategiu pri planovani implementuje xv6?
- “Ide piesen dokola” (*Round Robin*)

- Ake dalsie strategie pozname?
  - FCFS (“kto prv pride, ten skor melie”)
  - SJF (“najkratsi najskor”)
  - Priorine planovanie

Planovacia politika

Peter Tomcsányi: http://edu.fmph.uniba.sk/~tomcsanyi/06a.pdf

- FCFS (First-Come First-Served)
  - Do fronty sa radia podla casu prichodu
  - Kazdy proces bezi, pokym neskonci alebo pokym neodide do stavu cakania (na nieco)
  - Znevyhodnuje vstupno/vystupne procesy
- SJF (Shortest Job First)
  - Do fronty sa radia podla dlzky svojho behu
  - V praxi nerealizovatelne bez nejakej modifikacie, nakolko dopredu nevieme urcit cas behu procesu

Planovacia politika

Peter Tomcsányi: http://edu.fmph.uniba.sk/~tomcsanyi/06a.pdf

- SRTN (Shortest Remaining Time Next)
  - Preemptivna verzia algoritmu SJF
  - Cas do skoncenia sa prepocitava (napr. pri prichode dalsej ulohy do fronty, pri odlozeni na cakanie atd)
  - Vybera sa ten proces, ktoremu ostava najmenej casu do skoncenia
- RR (Round Robin)
  - Preemptivny; na zaklade casoveho kvanta
  - Pri velkom kvante rastie cas odozvy, pri malom klesa efektivita (vela rezie na zmenu kontextu)

Planovacia politika

Peter Tomcsányi: http://edu.fmph.uniba.sk/~tomcsanyi/06a.pdf

- Prioritne planovanie
  - Zvyhodnuje procesy na zaklade priority
  - Znizuje ferovost
  - Dynamicka priorita upravuje ferovost (napr. prepocitanie priority podla casu stravenom na CPU)
  - V praxi casto verzia, kde je viacero prioritnych front

Planovacia politika

Peter Tomcsányi: http://edu.fmph.uniba.sk/~tomcsanyi/06a.pdf

- SPN (Shortest Process Next)
  - Alternativa zakladneho algoritmu SJF
  - Za najkratsi sa povazuje ten, ktory najmenej cakal v dobe medzi dvoma cakaniami na beh
- Algoritmus Loterie
  - Kazdy proces ma “los”
  - Niektore mozu mat viac “losov”
  - Spolupracujuce procesy si mozu “losy” odovzdavat
  - Ide o alternativu prioritneho planovania – ma predvidatelnejsie spravanie

Planovacia politika

Peter Tomcsányi: http://edu.fmph.uniba.sk/~tomcsanyi/06a.pdf

- Algoritmus feroveho podielu
  - Procesy sa planuju na beh tak, aby kazdy pouzivatel systemu “minul” rovnaky podiel casu na procesore
  - Ak ma uzivatel1 4 procesy a uzivatel2 6 procesov, tak podla tohto algoritmu 2/5 casu na CPU stravia procesy uzivatela 1 a 3/5 casu procesy uzivatela 2
- Garantovane planovanie
  - Funguje na zaklade dosahovania podmienky
  - Napriklad pri N procesoch kazdy ma mat 1/N casu CPU

Domace citanie

Chapter 7

Scheduling

xv6: a simple, Unix-like teaching operating system

Nieco aj pre narocnych

https://graphitemaster.github.io/fibers/