Višeprocesorski sustavi

1. Svojstva višeprocesorskih sustava

1. Zašto su višeprocesorski sustavi danas svugdje?

Do 2000. se svake dvije godine dvostruko povećavala frekvencija procesora. Današnja tehnologija više ne može efikasno raditi na višim frekvencijama pa povećanje snage uglavnom ide kroz veći broj jezgri na procesoru.

2. Navesti osnovne vrste višeprocesorskih sustava i gdje se one koriste.

simetrični višeprocesorski sustavi	• svaki procesor je zaseban čip i ima svoj priručni spremnik • zajednička memorija je odvojena i pristupa joj se preko sabirnice • "stari" višeprocesorski sustavi
višejezgreni procesori	• jedan čip s više jezgri, jedan spremnik u više razina od kojih su neke dijeljene između jezgri • zajednička memorija odvojena • većina današnjih sustava
simetrični višeprocesorski sustavi s višejezgrenim procesima	• svaki procesor je višejezgreni • koristi se u poslužiteljima
raspodijeljeni višeprocesorski sustavi	• više procesorskih kartica, na svakoj više procesora

3. Navesti prednosti i nedostatke simetričnih i asimetričnih višeprocesorskih sustava.

Simetrični: prednost je da svi procesori jednako brzo pristupaju zajedničkom spremniku. Mana je što je to vjerojatno sporije

Asimetrični: mana je što različiti procesori različitom brzinom pristupaju zajedničkom spremniku pa onaj koji je daleko od memorije će sporo pristupiti. Prednost je što je taj pristup vjerojatno u prosjeku brži.

4. Zbog kojih svojstava današnje sustave možemo nazivati i nehomogenim višeprocesorskim sustavima?

Homogenost se odnosi na jednakost procesora. Većina procesora danas može svakoj jezgri dinamički odrediti frekvenciju rada i to je tehnički narušavanje homogenosti, ali ne toliko da ih zovemo nehomogenima. Moderni višeprocesorski sustavi koji su namijenjeni za baterijom pogonjena računala imaju brze i spore jezgre pa ih OS koristi ovisno o situaciji. To je prava nehomogenost.

5. Što je to sklopovska višedretvenost (npr. hyperthreading)?

Procesor može stati dok izvodi jednu dretvu (čeka dohvat podataka iz memorije, dovršava složenu instrukciju). Može za to vrijeme imati u pripremi instrukcije druge dretve koje mogu uletjeti i obavljati se za vrijeme čekanja. Prema van to izgleda kao da imamo dvije jezgre/dva procesora.

2. Problemi pri ostvarenju operacijskog sustava za višeprocesorske sustave

1. Navesti načine za očuvanje konzistentnosti strukture podataka jezgre u višeprocesorskom sustavu.

Zaključavanje: gotovo svaka netrivijalna struktura podataka ima svoj ključ

Atomarne operacije: koriste se posebnim mogućnostima procesora koje prevoditelj mora uključiti u odgovarajućim situacijama

Oprezno korištenje

2. Koje dvije osnovne vrste zaključavanja postoje? Koja su njihova dobra i loša svojstva?

Zaključavanje može biti blokirajuće (dretva se zaustavlja i miče na stranu) → ima svoj trošak u kućanskim poslovima

ili radno čekanje (dretva u petlji provjerava dok se neki uvjet ne ispuni) → efikasno samo ako se zaključaju kratki dijelovi kôda, ako se cijelo vrijeme dohvaća i provjerava jedna vrijednost to može imati loš utjecaj na priručni spremnik

3. Kada ima smisla koristiti atomarne operacije?

Za kratke operacije; kada nam je prihvatljivo da stvar radi sporije jer smo ograničili (ili ukinuli) optimizaciju

4. Zašto samo korištenje atomarnih operacija ponekad nije dovoljno?

Nekad je problem paralelnom pristupu podatka jednostavno prekompliciran da se ispravno riješi atomarnim operacijama, a nekad je i skupo jer njihovo korištenje uvodi dodatne skupe popratne radnje.

5. Koja od navedenih "razina atomarnosti" je najzahtjevnija pri radu:
a) konzistencija nad slijedom operacija (total ordering)
b) konzistencija nad jednom operacijom (relaksirana)
c) konzistencija nad povezanim varijablama (acquire/release)
d) konzistencija nad nepovezanim varijablama (acquire/consume)?

a) zato što je poštivanje redosljeda svih označenih operacija najzahtjevnije za ostvariti. Treba puno dodatnih instrukcija i traži najviše vremena pri radu.

6. Zašto ponekad i samo spremanje/čitanje jedne varijable može biti problem (zašto se koriste READ_ONCE i WRITE_ONCE)?

Kako bi se izbjegle neočekivane posljedice optimiranja prevoditelja. Nekad je atomarni kôd preskup pa koristimo READ/WRITE_ONCE za korištenje zajedničkih varijabli u kôdu (izvan zaključanog dijela kôda.

7. Zašto se za "red pripravnih dretvi" na višeprocesorskom operacijskom sustavu zapravo koristi više redova, po jedan za svaki procesor?

Zbog efikasnjeg korištenja priručnog spremnika. Dretve koje se nakon nekog vremena nastave izvoditi na istom procesoru će pronaći svoje podatke u priručnom spremniku i moći će ih nastaviti koristiti.

3. Raspoređivanje dretvi u višeprocesorskim sustavima

1. Koji se kriterij rasporedivanja najčešće koristi za vremenski kritične dretve (SCHED_FIFO, SCHED_RR) a koji kriterij za nekritične (obične, SCHED_OTHER)?

Kritične:

SCHED_FIFO: prema prioritetu pa prema redu prispijeća
SCHED_RR: prema prioritetu pa ravnopravna podjela vremena
SCHED_SPORADIC: sporadični poslovi, prioritet se može smanjiti
SCHED_DEADLINE: prema krajnjem trenutku završetka

Postoji i RMPA za periodičke poslove: oni koji se češće javljaju imaju veći prioritet i LLF: prema najmanjoj labavosti – prioritet imaju oni koji imaju najmanje vremena između trenutka do kad moraju završiti i vremena koje je potrebno za izvođenje.

Nekritične: CFS i MFQ

2. Zašto se za svaki procesor stvara zasebni red pripravnih dretvi?

Zbog performansi: efikasnijeg korištenja priručnog spremika

3. Kada se koristi guranje a kada povlačenje dretvi kod rasporedivanja kritičnih dretvi?

U procesoru se uvijek mora izvoditi najprioritetnija dretva. Kada ona završi s poslom treba se uzeti sljedeća najprioritetnija. Ako je ta sljedeća iz nekog tuđeg pripravnog reda, umjesto da izvodi dretvu iz svog reda pripravnih (koja je manjeg prioriteta) događa se povlačenje iz tog tuđeg reda i izvodi se ta dretva.

Ako pak jedna dretva odblokira neke druge dretve i one su prioritetnije od onih koje se trenutno izvode na drugim procesorima treba ih gurnuti prema njima tako da se osigura da su sve aktivne dretve ujedno i prioritetnije od svih u redu pripravnih dretvi.

4. Kada se kod nekritičnih dretvi neke prebacuju iz jednog reda pripravnih u drugi (drugog procesora)?

Potrebno je povremeno uravnotežiti opterećenja procesora – balansirati redove pripravnih dretvi, da sve ravnopravno dobiju procesorsko vrijeme

4. Primjeri iz implementacije raspoređivanja u Linuxu

1. Navesti klase rasporedivača u Linuxu namijenjene za korisničke dretve.

STOP: interni, koristi se pri micanju svih zadataka s nekog procesora
DEADLINE: vremenski kritički zadaci, prema kriteriju najbližeg krajnjeg trenutka završetka
REALTIME: prioritetno raspoređivanje kritičnih zadataka (FIFO, RR)
CFS: rapoređivanje normalnih zadataka podjelom vremena (OTHER, BATCH, IDLE)

*nisam sigurna da su svi navedeni namijenjeni za korisničke dretve

2. Koja struktura podataka se koristi za ostvarenje "reda pripravnih dretvi" kod CFS-a?
crveno-crna stabla
Bonus: STOP i IDLE imaju samo 1 zadatak, DEADLINE neku strukturu gdje su zadaci logički posloženi prema trenutku kad moraju biti gotovi, REALTIME za svaki prioritet ima zasebnu listu.

3. Što je to grupa zadataka (task_group) u kontekstu CFS-a?

Zadaci koji su na neki način povezani, po svojstvima. Njih se može grupirati, i te grupe se mogu grupirati u neku hijerarhijsku strukturu. To može biti ručno ili automatski. Podjela vremena se tada vrši na temelju grupa.

4. Koja je osnovna zamisao u korištenju procesorskih domena (engl. scheduling domains)?

Želi se uzeti u obzir heterogenost procesora. Procesi sličnih svojstava stavljaju se u skupine (domene) koje isto mogu biti hijerarhijski povezane. Svaka domena ima jednu ili više grupa zadataka koje se rasproeđuju. Za svaku domenu se definira skup pravila kojima se povećava iskoristivost svojstava pojedinih procesora.