2.6 OS design considerations for multiprocessor and multicore

Friday, April 3, 2020 5:08 PM

SMP OS

I et SMP-system kan kjerne (kernel) execute på enhver prosessor, og hver prosessor har typisk sin egen selv-planlegging fra alle de tilgjengelige prosesser eller tråder. Kjernen kan konstrueres som multiprosesser eller flertråder, hvilket lar deler av kjernen execute i parallell. SMP-tilnærmingen gjør OS-et mer komplisert. OS-designeren må passe på kompleksitet vedrørende delte ressurser og koordinere handlinger ut ifra flere deler av OS-et som executer samtidig. Derfor må det innføres teknikker for å løse og synkronisere kall til ressurser.

Et SMP OS håndterer prosessor og andre computerressurser så sluttbruker kan se systemet på samme måte som et multiprogrammert uniprosessorsystem. En bruker kan konstruere applikasjoner som bruker flere prosesser eller flere tråder inni en prosess uten hensyn til om en enkel prosessor eller flere prosessorer vil være tilgjengelig. Dermed må et multiprosessor-OS tilby all funksjonalitet av et multiprogrammert system, i tillegg til andre funksjoner som tilrettelegger for multiprossesorer. Disse hovedfunksjonene innbefatter:

- **Samtidighet i prosesser eller tråder:** Kjernerutiner må være innadgående for å tillate flere prosessorer til å execute den samme kjernekoden samtidig. Med flere prosessor som executer det samme eller forskjellige deler av kjernen, må kjernetabeller og håndteringsstrukturer håndteres ordentlig for å unngå datakorrupsjon eller invalide operasjoner.
- **Planlegging (scheduling):** Enhver prosessor kan utføre planlegging, hvilket kompliserer oppgaven av å utlede planleggingsrettningslinjer og å sikre at korrupsjon av planleggingsdatastrukturer blir unngått. Hvis kjernenivå flertråding (*kernel-level threading*) blir brukt, så eksisterer muligheten til å planlegge flere tråder fra den samme prosessen samtidig på multiprosessorer. Multiprosessorplanlegging dekkes i kap 10.
- **Synkronisering:** Med flere aktive prosesser som potensielt har aksess til det delte adresserommet eller delte I/O-ressurser, så må det tilrettelegges for effektiv synkronisering. Synkronisering er en tilrettelegging som håndhever gjensidig utelukkelse (mutex) og hendelsesforløp. En vanlig synkroniseringsmekanisme brukt i multiprosessor operativsystemer er låser (Se kap 5).
- **Minnehåndtering:** Minnehåndtering på en multiprosessor må håndtere alle problemene som gjelder for uniprosessor computere (Se kap 7 & 8). I tillegg må OS-et kunne utnytte den tilgjengelige hardwareparallelliteten for å kunne oppnå best ytelse. Pagingmekanismer på ulike prosessorer må koordineres for å håndheve konsistens (*consistency*) når flere prossorer deler en page eller segment og må bestemme page-erstatningen. Gjenbruken av fysiske pages er den største bekymringen; det må garanteres at en fysisk page ikke lenger kan aksesseres med sitt gamle innhold før pagen blir satt til ny bruk.
- **Pålitelighet og feiltoleranse:** OS-et bør tilby en grasiøs degradering ved prosessorfeil (?). Planleggeren og andre deler av OS-et må gjenkjenne tapet av en prosessor og restrukturere håndteringstabeller i samsvar.

Fordi problemene med multiprosessor OS-design generelt involverer utvidelser av løsninger til multiprogrammerte uniprosessorløsninger, behandler vi ikke multiprosessor operativsystemer adskilt.

Multikjerne OS

Problemene man må ta hensyn til for multikjerne OS inkluderer alle de samme designproblemene diskutert så langt i denne seksjonen, men det er flere bekymringer som dukker opp. Problemet ligger i skaleringen av potensiell parallellisme. I samsvar med utviklingen av teknologi har vi nå entret en periode med "mange-kjerner" ("*many-core*") multikjerne systemer, der det er eksempelvis ti eller flere kjerner på en enkel chip.

Designutfordringen for mange-kjerner multikjerne systemer er å effektivt utnytte multikjerne prosesseringskraften og på intelligentvis håndtere de substansielle *on-chip*-ressursene. En sentral bekymring er hvordan matche den iboende parallellismen i mange-kjerne systemer med ytelsekravene til applikasjoner. For det første er det hardwareparallelisme inni hver kjerneprosessor, kjent som *instruksjonsnivåparallellitet*, som både kan og ikke kan utnyttes av applikasjonsprogrammerere og kompilatorer. For det andre så er det potensialet for multiprogrammering og flertådring execution inni hver prosessor. Til slutt er det potensialet for at en enkel applikasjon executer i samtidige prosesser eller tråder på tvers av flere kjerner (cores). Uten sterk og effektiv OS-støtte for de siste to typene av parallellisme, så vil ikke hardwareressurser bli utnyttet effektivt.

Kort fortalt: På grunn av de store fremskrittene i multikjerne teknologi så har OS-designere slitt med problemet om hvordan best utvinne parallellisme fra computer-arbeidsmengder.

Parallellitet inni applikasjoner

Mange applikasjoner kan i prinsippet deles inn i flere oppgaver som kan executes i parallell. Disse oppgavene kan implementeres som flere prosesser, muligens hver med flere tråder. Utfordringen for utvikler er å bestemme hvordan best dele inn applikasjonsarbeidet til uavhengige executable (utførbare)

oppgaver. Utvikler må bestemme hvilke deler som kan eller bør executes asynkront eller i parallell. Det er hovedsakelig kompilatoren og programmeringsspråkfunksjoner som støtter prosessdesignet av parallellprogrammering. Likevel kan OS-et støtte dette prosessdesignet, ved minimum, å effektivt tildele ressurser mellom parallelle oppgaver deinert av utvikler. Det finnes verktøy utviklere kan bruke for å enklere implementere parallellitet i multikjerne systemer; blant annet GCD (*Grand Central Dispatch*). Dette finnes blant annet for UNIX-baserte Mac OS X og iOS operativsystemer.

Virtuell maskin tilnærming

Et annet alternativ er å innse at forsøket på å multiprogrammere individuelle kjerner til å støtte flere applikasjoner kan være unødvendig bruk av ressurser, gitt den stadige utviklingen av antall kjerner på hver chip. Man kan istedenfor dedikere én eller flere kjerner til en gitt prosess, og så la prosessoren alene bruke sin innsats på den prosessen. På denne måten unngår man overhead ved oppgavebytting og planleggingsvalg. Multikjerne OS-et kan da opptre som en *hypervisor* som tar høynivåvalg til å tildele kjerner til applikasjoner, men da vet lite om hvordan ressurstildelingen foregår.

Tidligere ble et program kjørt på en enkel prosessor. Med multiprogrammering har hver applikasjon en illusjon om at den kjører på en bestemt prosessor. Multiprogrammering er jo basert på konseptet av en prosess, som igjen er en abstraksjon av et executionmiljø. For å håndtere prosesser må OS-et kreve beskyttet rom, urørt fra brukere og programgrensesnitt. Av denne grunnen er det viktig med distinktheten av en kjernemodus (*kernel mode*) og brukermodus (*user mode*). På denne måten abstraherte kjernemodus og brukermodus en prosessor til to prosessor. Med alle disse virtuelle prosessorene oppstår spørsmålet om hvem som får den fysiske prosessoren. Overheadet av bytting mellom alle disse prosessorene kan utvikle seg til et punkt hvor det går utover responshastigheten, og da spesielt ved multikjerner. Men for mange-kjerne systemer kan vi droppe distinktheten av kjerne- og brukermodus – i denne tilnærmingen oppfører OS-et seg mer som en hypervisor. Programmene tar selv ansvar for mye av ressurshåndteringen. OS-et tildeler en applikasjon, en prosessor, tilstrekkelig minne, og programmet selv, ved å bruke metadata generert av kompilatoren.