Forståelsesspørsmål 1

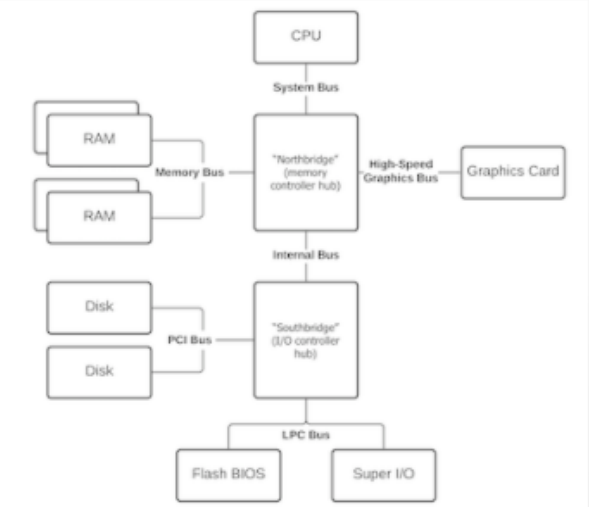


Figure 1: Intel Hub Architecture

Norsk

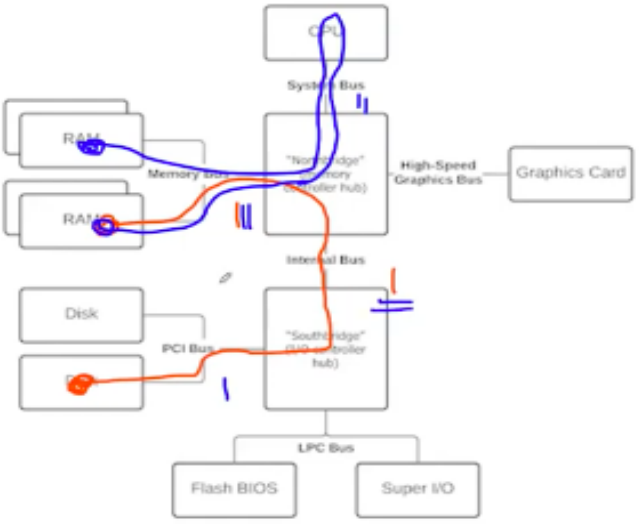
Tenk på den (forenklede) Intel Hub-arkitekturen som er vist i figur 1. Et program som kjører på et generelt operativsystem ber om en enkel datablokk, som ikke finnes i prosessoren og som må hentes fra disken. Etter vellykket gjennomføring av forespørselen:

- Hvor mange ganger har dataene gått over den interne bussen?
- Og hvor mange ganger over minnebussen?
- Hvis mer enn en gang (i noen av de ovennevnte tilfellene): Av hvilken grunn? Åpenbart fører dette til ytelsesoverhead. Hva er avveiningene?

Hvor mange ganger har dataene gått over den interne bussen? (Se videoen på nytt) og hvor mange ganger over minnebussen?

Det står i oppgaveteksten at “programmet kjører på et generelt operativsystem som ber om en enkel datablokk som ikke finnes i prosessoren og som må hentes fra disken”. Datablokken må da innhentes fra disken og overføre denne datablokken til et allokert minneplass ved RAM(siden RAM er en minneenhet). Overføringen foregår slik at datablokken overføres fra “Disk” mot “Southbridge” gjennom “PCI-Bus”. “Southbridge” overfører datablokken videre til “Northbridge” gjennom den “interne bussen”. Dette regnes som 1 overføring hvor dataene har gått over den interne bussen. Deretter vil vi lagre datablokken over til RAM gjennom “Minne-bus”. Men selvom datablokken er lagret i RAM, så kan ikke applikasjonen aksessere dataen siden hver applikasjon har sitt eget minneområdet og husk dette med privilegiernivåer hvor Applikasjon har ikke tilgang til å gå inn i kjernen av CPU´en og henter minne fra disken. Derfor må dataen fra RAM altså kjernen minneområde,

overføres til applikasjonens minneområde. Datablokken må da overføres gjennom CPU´en mot applikasjonens egen minne. Bildet under forklarer det som er blitt nevnt over.



Så poenget med denne oppgaven som Pål sier, er at vi skal forklare hvordan datablokken overføres og så lenge det er forklart, så skal man få poeng for det. Altså, jeg kunne ha skrevet 3,4,5 ganger, men det essensielle, er å skjønne at datablokken må overføres til applikasjonens minneområde og siden applikasjonen har tilgangen til dette, så må vi ta hensyn til dette ved at datablokken traverserer gjennom CPU´en til sitt eget minneområde. Han hevder også at vi kunne gå et steg over hvor vi da må ta hensyn til dette med Cache og Register siden realistisk sett så vil ikke applikasjonen aksessere data gjennom RAM. Vi husker fra IN1020 med minnehierarki og ved RAM så tar det en stund å aksessere minnet. Det er gjennom Registeret og Cache vi egentlig aksessere data fra. Derfor så må vi huske å inkludere 2 til. For å konkludere så kan man si at noen spørsmål har ikke alltid et fasitsvar, men kan være flere. Det viktigste er at man forklarer drøftingen bak svaret, da får man poeng.

Hvor mange ganger over minnebussen? (Se videoen på nytt)

Forståelsesspørsmål 2

Norsk

Programmeringsspråk som C, Java og Python, inndeles ofte i to-nivåer: vi snakker om såkalte lavnivå- og høynivåspråk. Selv om denne dikotomien ikke er entydig, har den likevel forklaringsverdi. I hovedsak er et språk på lavt nivå *nærmere* maskinkoden, de faktiske instruksjonene som kjører på CPU, enn et språk på høyt nivå er.

- Hva er det mest lavnivåspråket du kan tenke på, og hvordan begrunner du dette?
- Fra brukerens snarere enn maskinens perspektiv, hva er noen særtrekk ved begge typer språk? Vurder for eksempel brukervennlighet, feilsøking, portabilitet og mer.
- Nevn noen fordeler og ulemper med begge typer språk, avhengig av formålet. Vurder for eksempel systemprogrammering vs applikasjonsprogrammering.

Hva er det mest lavnivåspråket du kan tenke på, og hvordan begrunner du dette?

- Assemblerkode er det jeg kan tenke på når det kommer til lavt-nivåspråk helt i dybden ettersom vi utfører instruksjoner i CPU-en. Om dette er instruksjoner som “Fetch, Add, Get, Sub” osv. LMC er et eksempel på et programmeringsspråk som anvender av assemblerkode.

Fra brukererns snarere enn maskinens perspektiv, hva er noen særttrekk ved begge typer språk? Vurder for eksempel brukervennlighet, feilsøking, protalitibilet og mer.

- Lavtnivåspråk er ikke lett å portere, gjenbruke på datamaskiner med andre prosessorarkitekturer.

- Lavtnivåspråk har full kontroll over alle muligheter som maskinen tilbyr, men man må skrive mye koder for å lage komplekse programmer(tenk på in1020-LMC obligen, der var det mange instrukser som måtte skrives inn).

- Høynivåspråk tilbyr abstraksjoner som kan være fjernt fra hardware-evner og gjøre programmering enkelt og rask. Ulemper er dette med at det kan forekomme problemer ved diverse konsepter som while, for-loops, nullpointer osv.

Nevn noen fordeler og ulemper med begge typer språk, avhengig av formålet. Vurder for eksempel systemprogrammering vs applikasjonsprogrammering

Fordeler

- Lavtnivå: Kontroll, hastighet og god bruk av ressurser
- Høytnivå: Kan lage komplekse programmer, trenger ikke bry seg om detaljer innenfor arkitekturen og kan kode mer effektivt i forhold til programmer sammenlignet med lavtnivå(tenker da på IN1020-obligen med LMC)

Ulemper

- Lavtnivå: Mye kode som må skrives inn for et gitt program. Lite porterbar som vil si at instruksjonene er ofte arkitekturavhengig siden instruksene kan være forskjellig fra arkitektur til arkitektur. Tilslutt er det lett å gjøre feil ved programmene.
- Høytnivå: Kan lage programmer med algoritmer som er trege, koden er ikke nødvendigvis optimert for en prosessor.

Forståelsesspørsmål 3

Norsk

C regnes som et programmeringsspråk på lavt nivå. Oppfunnet for mer enn et halvt århundre siden, har den fortsatt en sterk posisjon innen programvareutvikling: Mer enn 95 % av Linux-kjernen er implementert i C. Den er også dominerende i andre operativsystemer og innebygde systemer.

- Hva gjør C spesielt godt egnet for programmering av operativsystemer?
- Hvordan skiller lavnivå- og høynivåspråk seg i sin tilnærming til hukommelse? Hvordan påvirker denne forskjellen deres brukbarhet for programmering av operativsystemer?
- Er det mulig å implementere et operativsystem på et språk på høyt nivå, slik som Lisp? Hva med et annet språk på høyt nivå, for eksempel om Python? Oppgi grunnene dine.

Hva gjør C spesielt godt egnet av operativsystemer?

- Gjennom “C” så kan du ta kontroll over hver eneste byte som er synlig for koden som kjører i et høytnivåspråk. F.eks si at du har kode som består av binære trær. Gjennom “C”, så kan vi utnytte disse bytene i koden som bygger opp binære trær.
- Du kan tolke noen bytes som tekst, som bokstav, peker, integer osv

Hvordan skiller lavnivå- og høynivåspråk seg i sin tilnærming til hukommelse? Hvordan påvirker denne forskjellen deres brukbarhet for programmering av operativsystemer?

- Lavtnivå: Som nevnt tidligere så har man mye kontroll i forhold til disk og minne utnyttelse ved “C”, sammenlignet med Python.

Er det mulig å implementere et operativsystem på et språk på høyt nivå, slik som Lisp? Hva med et annet språk på høyt nivå, for eksempel som Python? Oppgi grunnene dine.

- Beskyttelsesmekanismer som er en del av språket gjør det umulig å skrive OS´er med dem.
- Noen høynivåspråk mangler konstrukter for å kunne ta kontroll for å adressere noe som helst.
- OS´er må kunne overføre forvaltning av ressurser til prosesser og høynivåspråk tillater ikke at man gir fra seg forvaltning av ressurser som minne.