Postcode en Woonplaats:

Jaar van eerste inschrijving:

Datum: 29-06-2004

Naam docent:

A 1

instruction fetch unit

↓

instruction decode unit

↓

operand fetch unit

↓

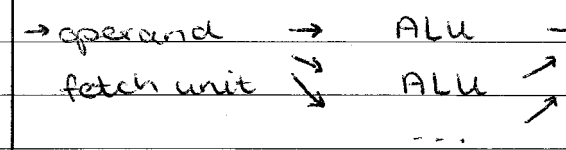
instruction execute unit

↓

write back unit

Een pipeline is te vergelijken met een lopende band. Elke cycle gaat er een pakketje in en alle vorige pakketjes schuiven daar naar de volgende arbeider die iets met het pakketje doet. Dus iedere cycle is elke arbeider bezig en iedere cycle is er een pakketje klaar.
Dit is natuurlijk beter dan als er één pakketje binnenkomt, langs alle medewerkers gaat, eruit komt en dat pas dan een nieuw pakketje verwerkt wordt. In het eerste geval is er per cycle een pakketje klaar, in het tweede geval is er per zoveel cycles pas een pakketje klaar. En omdat een gebruiker graag een snelle computer heeft, maken we gebruik van een pipeline
Helemaal bovenaan is een voorbeeld van een pipeline aangegeven.
Hier werken we niet met pakketjes, maar met instructies. De instruction fetch unit zorgt ervoor dat er elke cycle een instructie opgehaald is Deze instructie achtereelkaar gedecodeerd, de operanden worden ervoor opgehaald, de instructie wordt uitgevoerd en de nieuwe gegevens worden opgeslagen.
Bovengenoemde units kunnen steeds met een nieuwe instructie beginnen, ze werken onafhankelijk van elkaar (ze hoeven niet op elkaar te zitten wachten)
Stel dat een cycle n ms duurt. Dan verwerkt de gegeven pipeline een instructie per n ms. In het andere beschreven geval zal het 5n ms duren per

een bepaalde taak wat langer duurt dan de andere taken. De cycle is hierdoor groter (kost meer tijd) en dat is niet wenselijk. Het betekent namelijk dat de andere arbeiders ~~elke~~ even ~~elke~~ cycle moeten wachten.
Bij een pipeline heeft de instruction execute unit wat meer tijd nodig om zijn taken uit te voeren dan de andere units. Hierdoor moet er met een grotere cycle gewerkt worden en zullen instructies minder snel verwerkt worden. Een oplossing is het gebruik van een superscalaire pipeline.



In plaats van een gewone instruction execute unit, maken we gebruik van bovenstaande pritipe. Voordat de ene instructie helemaal door de nieuwe execute unit is, wordt er ook een andere in gestopt. Zo zullen de instructies sneller na elkaar verwerkt worden, dus zullen er binnen bepaalde tijd meer instructies verwerkt worden.

A3

methode 1: ~~LRU~~ least recently used
Bij een page fault wordt gekeken naar welke pagina het langst in het geheugen zit, die wordt eventueel teruggeschreven naar de schijf en de benodigde pagina wordt ervoor in de plaats in het geheugen geladen. Wanneer we te maken hebben met een lus in het programma, kan dit erg slecht uitpakken. Stel dat we hebben: pagina 0, ..., pagina 7
in het geheugen en 8 we hebben pagina 8 nodig.
Dan wordt pagina 0 vervangen:
pagina 8, pagina 1, pagina 2, ..., pagina 7.
Maar nu ~~begint~~ 0 moet de lus herhaald worden, dan moet dus pagina 0 ingeladen worden en deze vervangt

Postcode en Woonplaats: | Jaar van eerste inschrijving: | Datum: Naam docent:

(A3) is de pagina die we het laatst uit het geheugen gegooid hebben.

methode 2:
Per plaats in het geheugen (voor een pagina) staat vast welke pagina's er allemaal kunnen ~~se~~ worden ingeladen. Wanneer een pagina ~~moet~~ nodig is en deze staat nog niet in het geheugen, wordt deze dus op zijn plek ingeladen nadat de vorige pagina eventueel teruggeschreven is.
Nu kan het voorkomen dat pagina 0 en pagina 8 ~~ste~~ dezelfde plek in het geheugen toegewezen hebben gekregen. Wanneer deze beide pagina's vaak (om en om) nodig zijn, zal dit veel page faults opleveren. Deze situatie is dus niet gewenst.

A2 In kernel-mode kunnen alle (machine)instructies uitgevoerd worden. In user-mode is dit niet het geval. De user-mode zal de gebruiker niet toelaten om allerlei belangrijke interne dingen te veranderen. De memory management unit zorgt ervoor dat dit goed gaat. ~~Als~~ Wanneer iemand probeert dingen in het geheugen van de computer aan te passen, zal de memory management unit checken ~~of het~~ in welke mode de gebruiker is en of deze actie toegestaan is.
Als een exception optreedt, zal de computer automatisch in user-mode overgaan. De gebruiker kan dan geen schade aanrichten terwijl de exceptie afgehandeld wordt.

N2
- application
- s...
- p...
- transport
- network
- data link
- physical layer

bij internet:
- application
- transport
- internet
- data link
- physical layer

In de header zal worden geleden (data link) waar het pakketje naartoe moet. De router zoekt dan in een tabel op naar welke router het pakketje vervolgens moet gaan en zorgt ervoor dat het pakketje doorgestuurd wordt.

In de header van een pakket staat soms ook hoe lang een pakketje rondgestuurd mag worden. Dit moet regelmatig gecontrolleerd worden. Wanneer de tijd verstreken is, moet het pakketje niet meer verder verstuurd worden.

Wanneer een data-pakketje aangekomen is waar het moet zijn (fysieke laag), zal de checksum van het pakketje gecontrolleerd worden (als mede het sequentienummer). Als alles in orde is, wordt de data in het pakketje aan de applicatielaag doorgegeven.

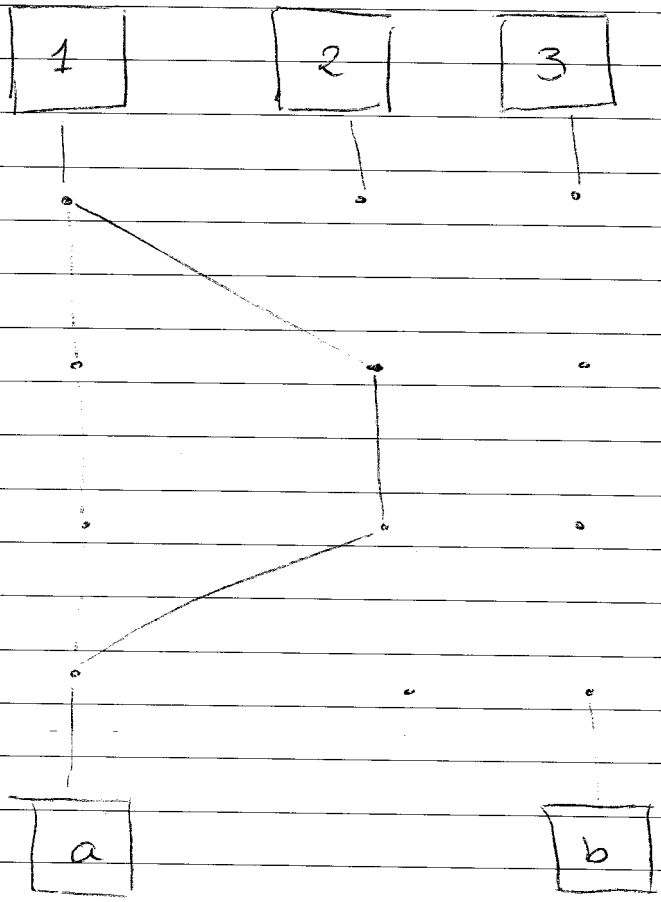
N3 Wat een gebruiker graag wil, is een hoge / goede quality of service. We willen natuurlijk een betrouwbaar netwerk: er moet geen informatie verloren gaan, alles wat verstuurd wordt moet in goede staat aankomen. Ook is snelheid belangrijk. Eigenlijk wil je als gebruiker dat alles wat je verstuurt meteen aankomt, je wil alles zo snel mogelijk. Redelijk lage kosten zijn ook belangrijk.

Een hogere snelheid kan bijvoorbeeld bereikt worden door gebruik te maken van een sliding window protocol. Door niet steeds te wachten op bevestiging van een vorig pakketje, voordat je een nieuw pakketje verstuurt, kunnen er meer pakketjes binnen een bepaalde tijd verstuurd (en ontvangen) worden.

Een goede kwaliteit van ontvangen data is te bereiken met goede methodes om checksums te berekenen. Het liefst willen we per pakketje een

(N3)

voorbeeld:



Stel dat 1 iets uitzendt, bijvoorbeeld een film. Wanneer a het wil ontvangen, kan een verbinding opgezet worden als in oranje is aangegeven. Maar als b iets later besluit ook mee te willen genieten, moet voor hem een geheel nieuwe verbinding opgezet worden. Wanneer we van de verbinding als in het zwart aangegeven is gebruik maken, kan ontvanger b heel makkelijk aansluiten als hij dat wil. Met veel ontvangers is het tweede geval efficiënter. Daardoor wordt dus de quality of service hoger. (Natuurlijk wordt hier gebruik gemaakt van een zeer goede checksum en een protocol dat minstens zo goed is als het sliding window-protocol).

N1

c: capaciteit, aantal bits per seconde dat maximaal verstuurd kan worden

d: hoeveel verstuurd wordt per seconde (aantal bits)

(N.)

d veel kleiner dan c levert heel weinig tot geen vertraging.

$$\text{vertraging} = \frac{1}{1 - \frac{d}{c}}$$

Hoe meer de capaciteit uitgebuit wordt (d dichtbij c, $d < c$ geldt altijd), hoe meer vertraging ($\frac{d}{c}$ dichter bij 1, $1 - \frac{d}{c}$ klein $\rightarrow$ $\frac{1}{1 - \frac{d}{c}}$ groot).

3

$d = c$ kan in de formule niet, dit zou onmogelijk veel vertraging opleveren. In andere woorden: er wordt niets meer verstuurd. Dus c is echt een absolute bovengrens voor d.