Notater til OSYDA50 - Operativsystemer

Thomas Nordli <tn@hive.no>

25. november 2010

Kompendium 2 / 2010 Høgskolen i Vestfold
Copyright: Høgskolen i Vestfold / Thomas Nordli
ISSN 0808-1328

Innhold

1 01-Maskinvare-bakgrunn 4
1.1 Hoved-deler i datamaskin 4
1.2 Instruksjons-syklus 6
1.3 Avbrudd 9
1.4 Minnehierarkiet 11
1.5 I/O - kommunikasjon 13
1.6 Oppgaver: Bli kjent med systemet 16

2 OS-overblikk 17
2.1 Repetisjon av maskinvarebakgrunn 17
2.2 Målsetning med OS 17
2.3 Operativsystemers utviklingshistorie 18
2.4 Viktige teoretiske fremskritt i OS-utviklingen 23
2.5 Utvikling mot moderne OS 26
2.6 Eksempler 26
2.7 Oppgaver 27

3 03-linux-intro 30
3.1 Repetisjon OS-overblikk 30
3.2 Løsningsforslag til oppgave 2.1 30
3.3 Historie/bakgrunn 32
3.4 Filbehandling 34
3.5 Redirigering 36
3.6 Bruk av skall (shell) 38
3.7 Oppgave 3 43

4 04 - Shell scripting 45
4.1 Repetisjon 45
4.2 Shell-skripting 45
4.3 Forgrunn/bakgrunn 49
4.4 Oppgaver 49

5 05 - Filer og filsystemer 52
5.1 Løsningsforslag 52
5.2 Generelt om filsystem 56
5.3 Filhåndtering i Linux 58
5.4 Oppgaver 65

6 06 - Filsystem-sikkerhet og brukere i Linux 68
6.1 Løsningsforslag 68
6.2 Filsystem-sikkerhet 72
6.3 Brukere i Linux 74
6.4 Oppgaver 77

7 07-prosesser 79
7.1 Løsningsforlag 79
7.2 Om prosesser 82
7.3 Representasjon/beskrivelse av prosesser 83
7.4 Noen tilstands-modeller 86
7.5 Kontroll av prosesser 88
7.6 Alternativer for operativsystemets kjøring 94
7.7 Oppgaver 95

8 08-Tråder og SMP 97
8.1 Løsningsforslag 97
8.2 Tråder 98
8.3 SMP (09.1) 104
8.4 Oppgaver 108

9 10 Samtidig behandling 111
9.1 Løsningsforslag 111
9.2 Del 1 (09.2) 113
9.3 Del 2 (10) 124
9.4 Oppgaver 131

10 11 - minne og signaler 134
10.1 Løsninger 134
10.2 Repetisjon: Pipes 136
10.3 Signaler 138
10.4 Minnehåndtering 142
10.5 Oppgaver 147

11 12 Sideveksling, virtuelt minne og sikkerhetstrusler 149
11.1 Kort repetisjon fra forrige gang 149
11.2 Sideveksling (paging) 150
11.3 Virtuelt minne 151
11.4 Sikkerhetstrusler 153
11.5 Oppgaver 156

12 13 - Sikkerhetstiltak (og litt mer om trusler) 158
12.1 Autentisering 158
12.2 Tilgangskontroll 160
12.3 IDS - Intrusion Detection System 162
12.4 Antivirus 163
12.5 antivirus 163
12.6 forsvar mot buffer overflyt-angrep 164
12.7 Fysiske tiltak 164
12.8 Redundans 165
12.9 BIOS 165

12.10 Sikkerhet på linux 166
12.11 (Ekstra for spesielt interesserte) Et tenkt tilfelle: Innbrudd i fire akter 168
12.12 Oppgaver 173

13 Løsningsforslag på eksamen høst 2008 176

14 Løsningsforslag på eksamen høst 2009 183

Om skriftet

Skriftet *Notater til OSYDA50* er et kompendium som inneholder undervisningsmateriale. Undervisningsmaterialet er produsert i forbindelse med undervisning i faget *Operativsystemer.* , høsten 2010. Faget ble tilbudt som obligatorisk fag, for ingeniørstudenter med *grunnleggende ferdigheter i programmering*. Det var også forventet at de fulgte kurset *Algoritmer og datastrukturer* parallellt.

Kompendiet er en sammenstilling av:

- notater
- eksempler
- oppgaver
- løsningsforslag

, som ble brukt i forbindelse med undervisningen.

Om forfatteren

Forfatteren, Thomas Nordli, var fagansvarlig for kurset skoleåret 2010/2011. Han er ansatt som høgskolelektor ved *Fakultet for teknologi og maritime fag* ved *Høgskolen i Vestfold*, hvor han underviser i *Operativsystemer*, *Databaser* og *Programmering*.

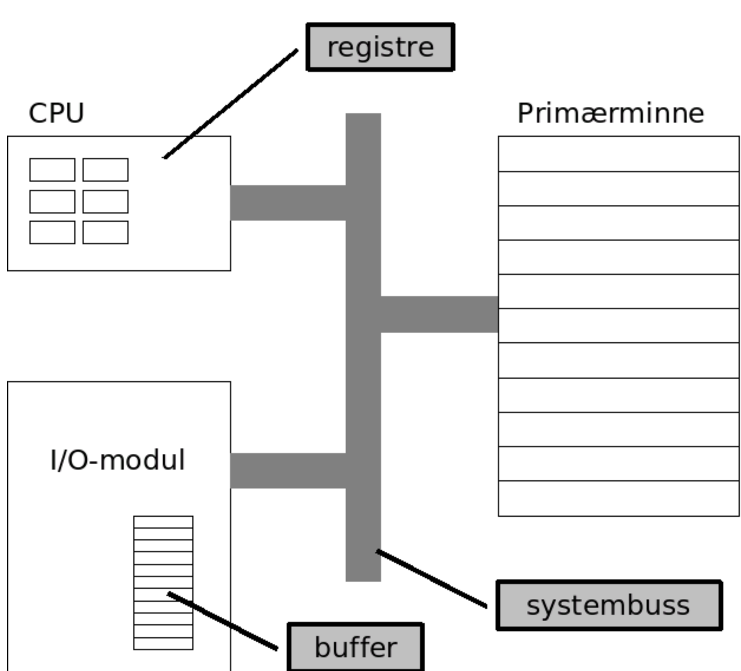
Kapittel 1

01-Maskinvare-bakgrunn

- Et operativsystem kan anses som programvare som kontrollerer maskinvaren i et datasystem, og tilbyr et forenklet grensensnitt mot datasystemets tjenester. I denne forelsningen ser vi nærmere på maskinvaren.

1.1 Hoved-deler i datamaskin

figur

- 

cpu - kontrollerer og beregner

- Et register består av en rekke flip-flops

Typer av prosessor-registre

registre tilgjengelig for brukere

tilgjengelige via maskinkode-instruksjoner

- generelle
- spesielle (f.eks. for flytalls-/heltalls-operasjoner)
- adresseregistre
- stakkpeker
- segmentregister
- indeksregister
- dataregistre

kontroll- og status-registre

- MAR - Memory Address Register
- MBR - Memory Buffer Register
- I/OAR - Input/Output Address Register
- I/OBR - Input/Output Buffer Register
- PC - Program Counter
- IR - Instruction Register

primær-minne

- ”et sett av lokasjoner definert av sekvensielt nummererte adresser”
- Data kan skrives til og leses fra hver enkelt adresserbare lokasjon
- inneholder både program (instruksjoner) og data

I/O moduler

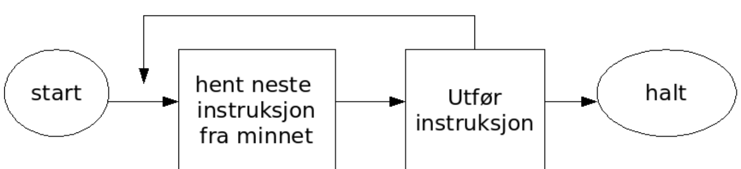
- flytter data mellom datamaskin og eksternt miljø (f.eks. sekundært minne - disker)
- Inneholder bufre for mellomlagring
- Systembuss - kommunikasjonskanal mellom CPU, minne og I/O

Spørsmål:

- Kan du se noen farer/utfordringer med at data og instruksjoner befinner seg i samme lager?

1.2 Instruksjons-syklus

figur

- 

Fire typer instruksjoner

- Prosessor ↔ primærminnet
- Prosessor ↔ I/O
- Behandling av data
- Kontroll (endre instruksjons-sekvensen)

Eksempel: Programkjøring (basert på eksempelet i seksjon 1.3 i pensumboka (Operating Systems Internals and Design Pricicples, Stallings 2009).

Eksempelsystemet har

tre registre

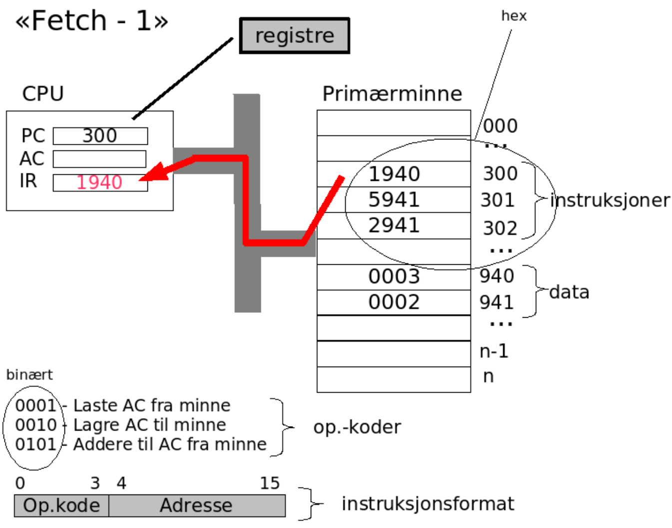
- PC – programteller (innholder adressen til neste instruksjon)
- AC – akkumulator (generelt registe
- IR – inneholder instruksjonen som sist ble hentet fra primærminnet

tre 16 bits instruksjoner

- Laste AC fra minnet
- Lagre AC til minnet
- Addere AC med minnelokasjon og skrive svaret tilbake i AC

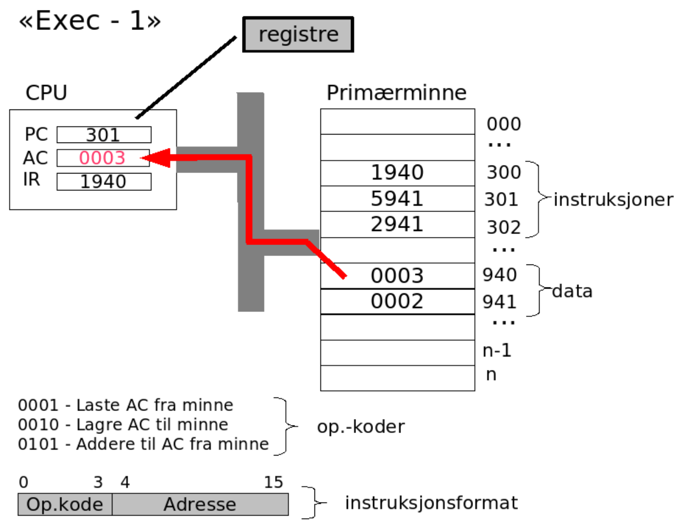
Fetch 1

-



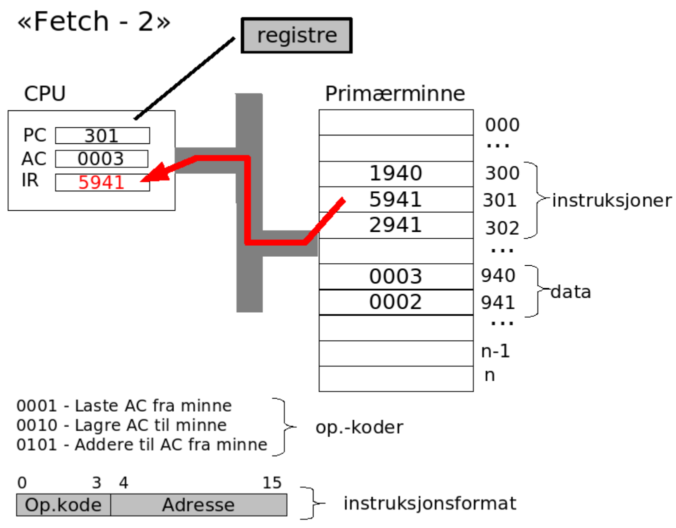
Exec 1

•



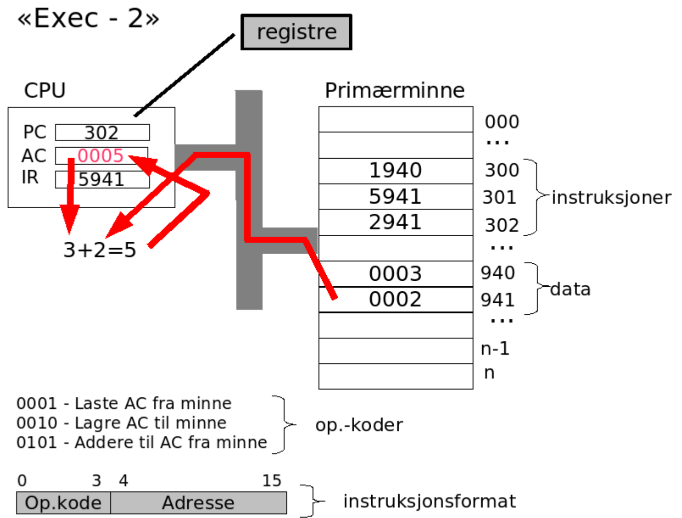
Fetch 2

•



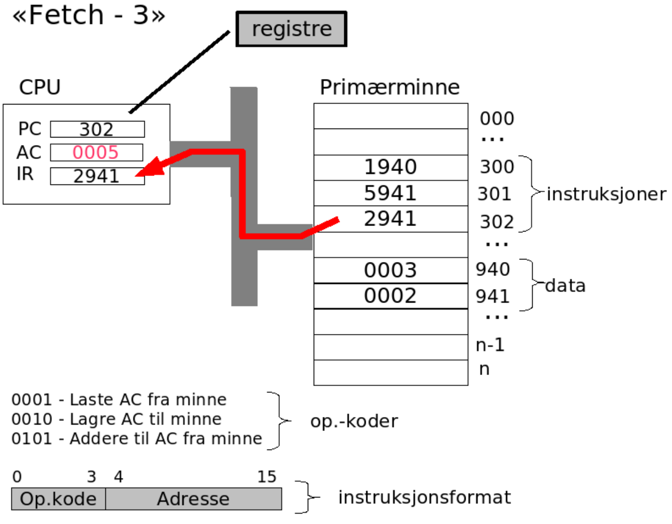
Exec 2

-



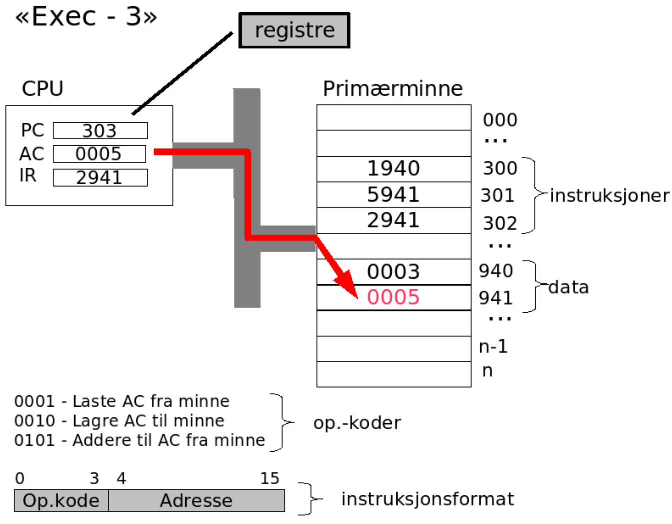
Fetch 3

-



Exec 3

-



1.3 Avbrudd

- Mekanisme som gjør det mulig for andre moduler, (f.eks. I/O-moduler) å avbryte den normale sekvensen (fetch execute).
- Hensikten er primært å øke prosessorens utnyttelsgrad – den slipper å vente.

Typer av avbrudd

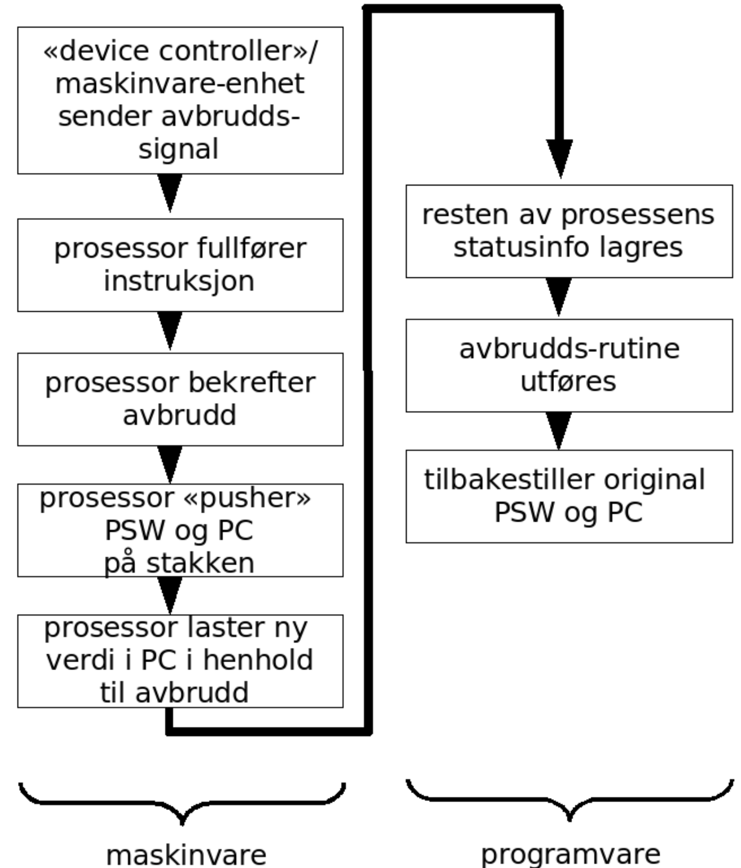
- Avbrudd ved maskinvarefeil
- I/O-avbrudd
- Avbrudd ved utløp av "timer"
- Programvareavbrudd

Eksempel

- Regnestykket under gir en ide om hastighetsforskjeller mellom CPU og I/O-enheter
- CPU: 1GHz -> 1E9 instruksjon/sekund
- Harddisk: 7200 rpm -> 4 ms på en halv runde
- -> CPU er 4 mill. ganger raskere enn HD

Utførelse av avbrudd

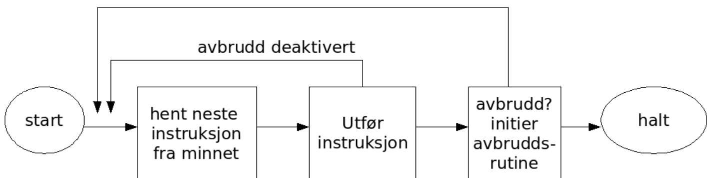
figur



- (basert på figur 1.10 i Operating Systems Internals and Design Pricicples, Stallings 2009)

fetch and execute m/avbrudd

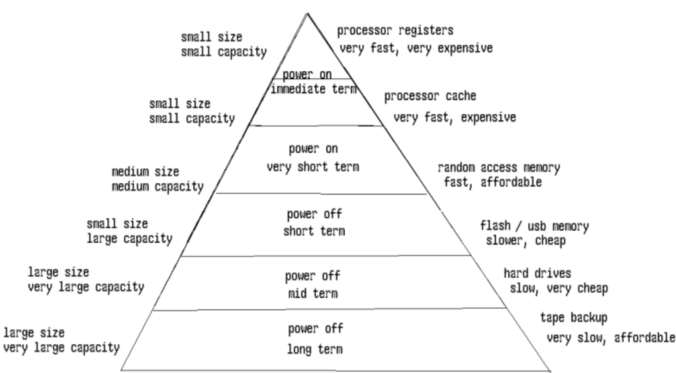
figur



- (basert på figur 1.7 i Operating Systems Internals and Design Pricicples, Stallings 2009)

1.4 Minnehierarkiet

figur



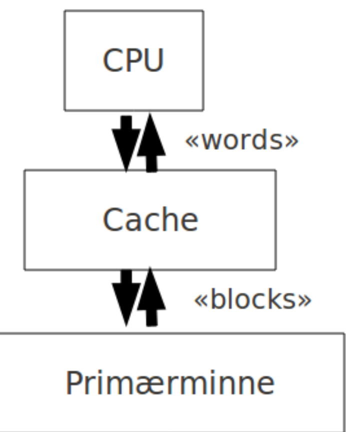
- (bildet er bildet er offentlig eiendom – "in the public domain")

Nedover i hierarkiet

- Lavere kostnad
- Økende kapasitet
- Økende "access time"
- Sjeldnere i bruk

hurtigminne - cache

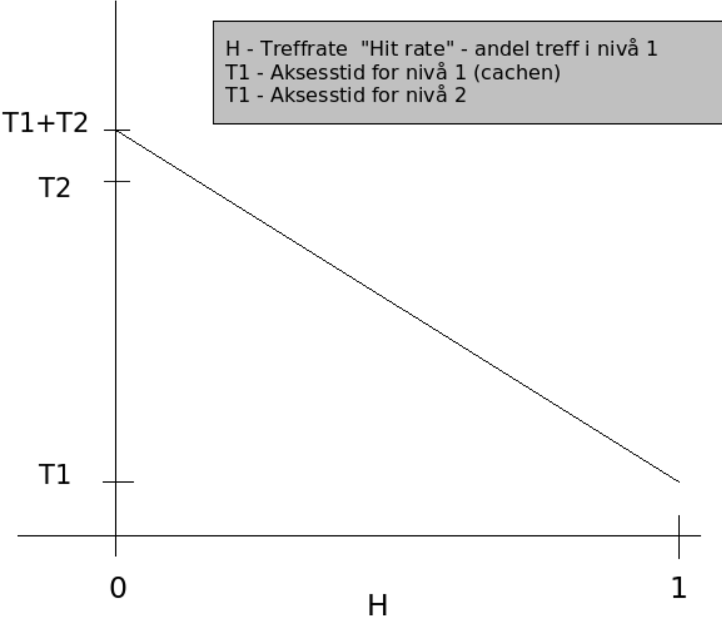
figur

- 

- Baserer seg på antagelsen om at vi har en tendens til at påfølgende referanser er til nærliggende lokasjoner ("Locality of Reference")

Treffrate

Figur



- (basert på figur 1.15 i Operating Systems Internals and Design Pricicples, Stallings 2009)
- Gjennomsnittlig accesstid som funksjon av treffraten i førstenivå-cachen.

Eksempel

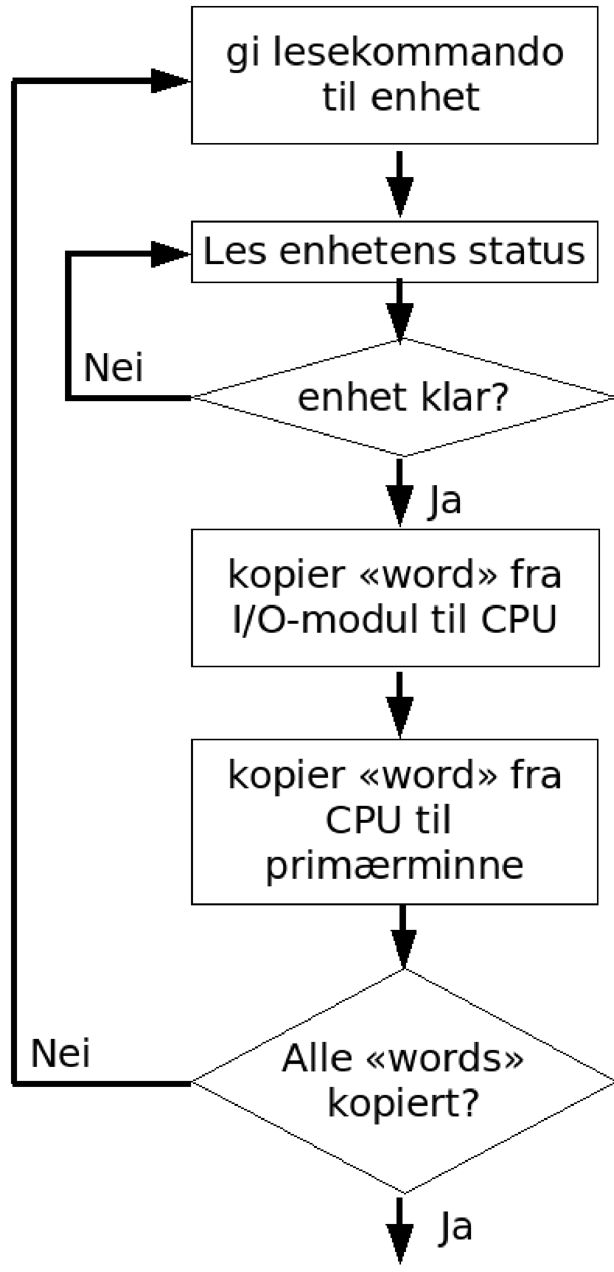
- Regnestykket under er hentet fra eksemplet på s. 27 i pensumboka (Operating Systems Internals and Design Pricicples, Stallings 2009).
- H=0.95
- T1=0.1 ms
- T2=1 ms
- (H)(T1)+(1-H)*(T1+T2)=
- (0.95)(0.1ms)+(0.05)(0.1ms+1ms)=
- 0.095+0.055=0.15ms

1.5 I/O - kommunikasjon

Programmert I/O

- I/O modul setter bits i statusregister - ikke noe signal til prosessor når den er ferdig med en operasjon
- prosessor: Kontrollerer, sjekker status og overfører

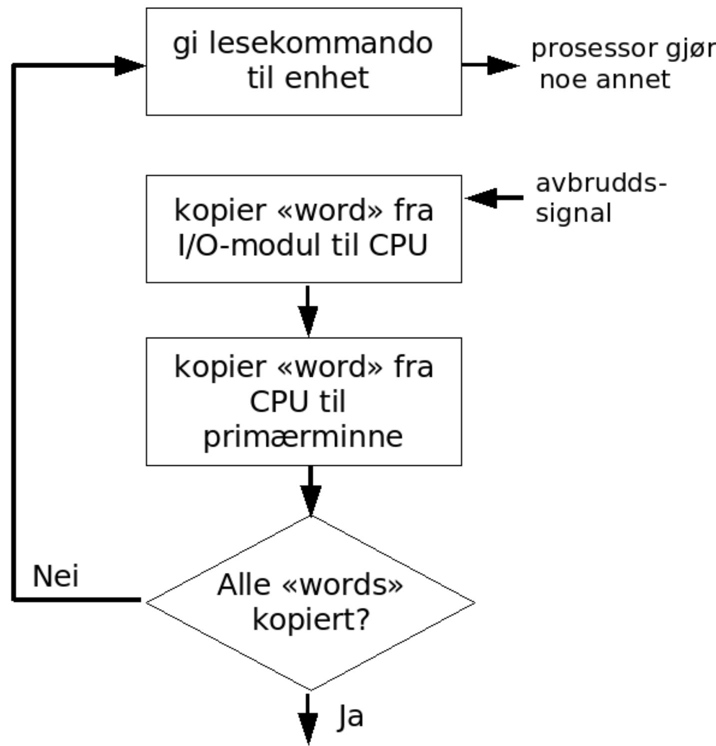
figur



- (basert på figur 1.19 i Operating Systems Internals and Design Pricicples, Stallings 2009)

Avbruddsdrevet I/O

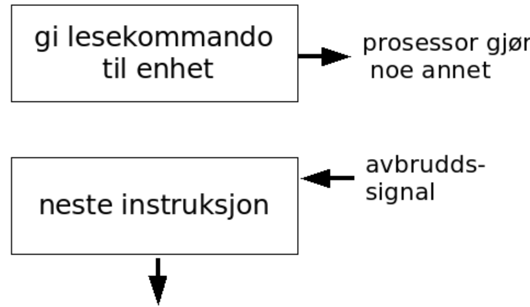
figur



- (basert på figur 1.19 i Operating Systems Internals and Design Pricicples, Stallings 2009)

DMA (Direkte minnetilgang - ”Direct Memory Access”)

figur



- (basert på figur 1.19 i Operating Systems Internals and Design Pricicples, Stallings 2009)

1.6 Oppgaver: Bli kjent med systemet

Oppgave 1.1: Aktivering

- Kontakt faglærer og få kontoen på debbie.hive.no aktivert.

Oppgave 1.2: Innlogging

Innlogging fra windows

- Logg inn på debbie.hive.no fra en maskin med windows ved hjelp av terminalemulatoren putty.

Forsøk å få starte editoren emacs, slik at den kommer opp grafisk modus i et eget vindu. Kommando: emacs. For at emacs skal komme opp i grafisk modus, må

- 1) Xming (eller tilsvarende) kjøre på windows-maskinen og

- 2) X11 forwarding må være slått på i putty (SSH->X11).

Innlogging fra linux/unix/mac

- Logg inn på debbie.hive.no fra en maskin med ved hjelp av ssh.

- Kommando: ssh -X brukernavn@debbie.hive.no

- Forsøk å få starte editoren emacs, slik at den kommer opp grafisk modus i et eget vindu.

Oppgave 1.3: Et C-program

- Bruk editoren emacs til å skrive et program i C. Kompiler programmet. Kommando: gcc -Wall filnavn.c

- Kjør programmet. Kommando: ./a.out

Oppgave 1.4: Hvem der?

Finn ut hvem er pålogget. Kommandoer:

- finger

- who

- w

- Skriv melding til pålogget bruker. Kommando: write brukernavn

Kapittel 2

OS-overblikk

2.1 Repetisjon av maskinvarebakgrunn

Hva er

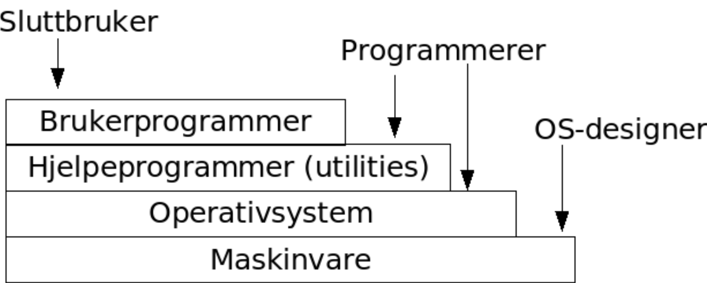
- hoved-delene i en datamaskin?
- instruksjons-syklus?
- minnehierarkiet?
- de tre I/O-kommunikasjonsmetodene?

2.2 Målsetning med OS

Bekvemmelighet - det skal være enkelt å benytte datasystemet

- De viktigste systemprogrammene kan sies å utgjøre operativsystemet. Resten kalles hjelpeprogrammer (utilities).

Figur

- 

Typiske områder hvor OS gir tjenester

- programkjøring
- kontrollert filtilgang
- tilgangskontroll til systemet
- feildeteksjon og respons
- ”bokføring” - bruks-statistikk og overvåking av ytelsesparametre
- (programvareutvikling - vanligvis ”utilities”)

Effektivitet - mest mulig effektiv utnyttelse av resurssene i datasystemet

Resursser

- Prosessorer
- Minne
- I/O-enheter (via I/O-kontrollere)
- Operativysystemet kontrollerer flytting, lagring og behandling av data

Skiller seg fra andre kontrollsystemer

- er ikke eksternt system - er et vanlig program
- oppgir ofte kontrollen og avhenger av prosessoren for å få den tilbake

Fleksibilitet

Et OS bør kunne utvikles over tid for å

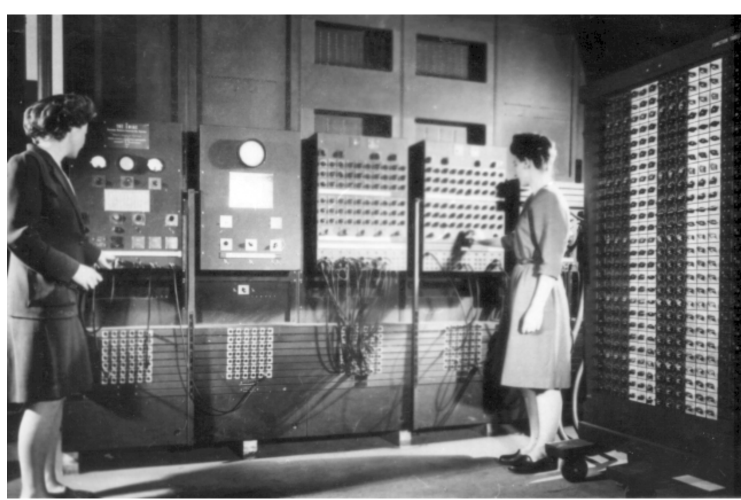
- håndtering av nye typer maskinvare/tjenester
- innføring av feil-/sikkerhets-fikser

2.3 Operativsystemers utviklingshistorie

seriell prossessering

- 1940-1950: uten OS

Foto ENIAC med to operatører (U. S. Army Photo - http://ftp.arl.army.mil/ mi-ke/comphist/)

- 

- Brukeren interagerer direkte med maskinvaren - brytere
- Programmene, i form av maskinkodeinstruksjoner, lastes fra innlesings-enheter (f.eks. hullkortleser)
- Fordeling av kjøretid: Reservering av tid på papirark

Oppsettingstiden for hver jobb

- lasting av programvare fra sekundærlager (magnetbånd/hullkort) til primærlager
- lenking
- lagring av objekt-program (kompilert program) på sekundærlager
- ... mye manuelt arbeid - bytting av magnetbånd/hullkortbunker

Kjøretiden

- treg maskinvare –> kjøretiden lang (timer - dager)

Effektiviseringtiltak: Felles programmvare

- Bibliotek med fellesfunksjoner
- "Linker"
- "Loader"
- "Debugger"
- I/O-driver-rutiner

enkle system for satsvis "batch" behandling ("simple batch system")

Maskinvaren ble raskere

- kjøretiden ble vesentlig kortere
- oppsettingstiden ble ikke tilsvarende forkortet (p.g.a. mye manuelle rutiner)
- Dyr maskinvare –> ønske om å maksimere prosessorens utnyttelsesgrad.
- Konseptet ved ved enkle batch-systemer øker utnyttelsesgraden ved å forkorte jobbenes oppsesttingstid.

Første OS

- i midten av femtiårene

for IBM 701

- http://www.computer-history.info/Page4.dir/pages/IBM.701.dir/index.html

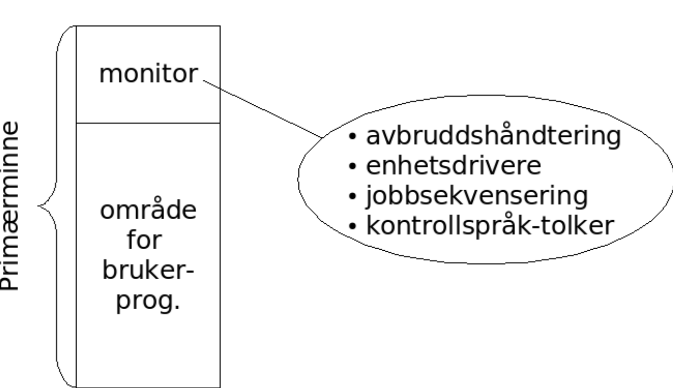
`http://www.computer-history.info/Page4.dir/pages/IBM.701.dir/index.html`

- av General Motors

monitor

- program som laster og kjører jobber fra kø på innlesningsenhet
- etter avslutning av hver jobb, tilbakeføres kontrollen til monitor
- den delen av monitoren som alltid er i minnet kalles "resident monitor"

Figur av monitor i minnet

- 

- monitor kan også innbefatte av hjelpe-programmer/-funksjoner

Brukeren er ikke lenger direkte i kontakt med maskinvaren - kun via operatør.

- operatør tar i mot jobber
- operatør ordner jobbene i bunker - ”batches”
- operatør legger i jobbene i kø
- monitor laster og kjører jobber fra kø

JCL - Job Control Language

Eksempelet i seksjon 2.2 i pensumboka (Operating Systems Internals and Design Pricicples, Stallings 2009) er gjenfortalt under.

-

```
$JOB   ( jobb starter her )
$FTN   ( fortrankompilator lastes og startes )
..     ( fortran-kildekode - som leses av kompilatoren og
..       kompileres. Resultatet, objektprogrammet,
..       lagres på sekundærminne )
$LOAD  ( objektprogrammet lastes fra sekundærminne til pirmærminne )
$RUN   ( kjøring av objektprogrammet starter )
..     ( eventuelle data for inlesning av programmet )
$END ( job avsluttes - monitor kjøres )
```

ønskelig HW-støtte for monitor

- beskyttelse av minnet
- tidtaker
- priviligerte instrukjsoner
- avbrudd (ikke tilgjengelig i tidlige maskiner)

operasjonsmoda

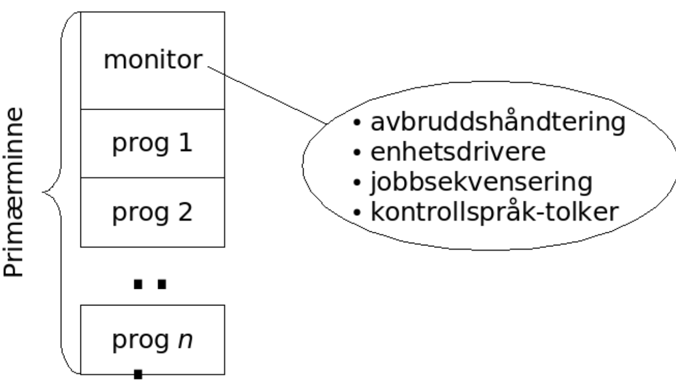
- usermode
- kernelmode

multiprogramkjørende system for satsvis behandling (”multiprogrammed batch system”)

- I/O-enheter tregere enn prosessor -> lav utnyttelsesgrad

flere jobber holdes i minnet samtidig

Figur

- 

- ved I/O forespørsel velges enn annen jobb for kjøring
- ved fullføring av I/O velges ny jobb for kjøring

HW-krav

- I/O-avbrudd
- DMA
- Resultat: Høyere grad av ytelse –> større gjennomstrømning

Kjøretidsdeling (”time sharing”)

- Flere bruker maskinen direkte

Foto av PDP 7 på institutt for informatikk ved Universitetet i Oslo

-

- Lisens for bruk av blidet

`http://creativecommons.org/licenses/sa/1.0/`

- N brukere; hver bruker får 1/N av kapasiteten
- Lav responstid er vesentlig
- En klokke sender avbrudds-signal ved faste intervaller
- Ved klokkeavbrudd velges ny bruker

EksempelOS: CTSS - Compatible Time-Sharing System

- Tidlig "time sharing"-system:
- Inspirasjonskilde for utviklerne av UNIX
- Klokke avbrøt hvert 0,2 sek.

2.4 Viktige teoretiske fremskritt i OS-utviklingen

Prosesser

Ulike definisjoner

- Et program under utførelse
- En instans av et program som kjører på en datamaskin
- Entitet som kan tildeles og kjøres på en prosessor

En enhet av aktivitet karakterisert av

- en en enkel sekvensiell tråd av utførelse
- en tilstand
- tilhørende system resursser
- Begrepet først brukt under utvikling av Mulitcs på 60-tallet

Problemer ved utviklingen av "batch"-, "time share" og "real time"-OS

- feilaktig synkronisering
- feil ved gjensidig utelukkelse
- uberegnelig utfall av progamkjøring
- vranglåser
- Problemene imøtegåes med systematisk måte å kontrollere programkjøringene

Prosess består av

- et kjørbart program
- tilhørende data

kjøre-kontekst

- info nødveding for å kontrollere prosessen
- f.eks. ulike prosessor-registre, prioritet, tid-brukt, etc.
- Vi ser på fig. 2.8 typisk prosess-implementasjon

`http://debbie.hive.no/osyda50/figurer/typiskProssesimplementasjon.png`

Minnebehandling

Krav til fleksibelt miljø med støtte for modulær programmering

- isolasjon av prosesser
- automatisk (og transparent) tilordning av minne (i minnehierarkiet)

støtte for modulære programmer

- definere moduler
- opprette/slette moduler
- endre størrelse på moduler
- Beskyttelse og tilgangskontroll
- Langtidslagring

Filsystemer

- langtidslagring
- lagring i navgitte objekter - filer
- filer - enheter/objekter i tilgangskontroll og beskyttelse

Virtuelt minne

- adressere minnet "fra logisk ståsted"
- virutell adresse
- virkelig/fysisk adresse
- Vi ser på fig. 2.10 virtuell minneadressernig.

`http://debbie.hive.no/osyda50/figurer/virtuell_minneadressering.png`

Beskyttelse av informasjon og sikkerhet

- Tilgjengelighet
- Konfidensialitet
- Data-integritet
- Autentisitet/ekthet

Skedulering (fordeling) og resurss håndtering

Hensyn

- Rettferdighet
- Differensiering mellom klasser

Effektivitet

- høyest mulig gjennomføringshastighet/"throughput"
- lavest mulig responstid
- flest mulig brukere
- Vi ser på fig. 2.11 key elements of os for multiprogramming.

`http://debbie.hive.no/osyda50/figurer/multiprogramming.png`

- Eksempel: Animasjon av 'round robin'

`http://cs.gmu.edu/cne/workbenches/rndrobin/rndrobin.html`

Systemstruktur

Størrelsen på koden blir et problem

- Forsinket levering
- Latente bugs
- Ytelse ofte lavere enn forventet
- modulær
- veldefinerte grensesnitt mellom modulene

store OS

- millioner-titalls millioner kodelinjer
- modulært er ikke tilstrekkelig –> hierarkisk lagdelt

`http://debbie.hive.no/osyda50/figurer/multiprogramming.png`

2.5 Utvikling mot moderne OS

- mikrokjerne
- flertrådskjøring
- symmetrisk multiprosessering
- distribuerte OS
- objektorientert design

2.6 Eksempler

UNIX-arkitektur

-

`http://debbie.hive.no/osyda50/figurer/unixarkitektur.png`

-

`http://debbie.hive.no/osyda50/figurer/unixkjerne.png`

Linux-arkitektur

-

`http://debbie.hive.no/osyda50/figurer/linuxkjerne.png`

loadable modules

dynamisk lenking

- lastes i minnet og lenkes til kjernen under kjøring
- insmod
- fjernes og avlenkes fra kjernen under kjøring
- rmmod

"stackable modules"

- hierarkisk ordnet
- moduler kan forutsette at andre moduler er lastet
- reduserer kodereplikering
- sørger for å ikke fjerne moduler som er i bruk

2.7 Oppgaver

2.1

Tenk deg at du har et flerprogramkjørende satsvis system med en bunke (batch) med tre jobber:

1.

- Tidfordeling mellom CPU og I/O:
- 00-10s CPU
- 11-50s I/O
- 51-100s CPU

2.

- Tidfordeling mellom CPU og I/O:
- 00-30s CPU
- 31-50s I/O
- 51-70s CPU

3.

- Tidfordeling mellom CPU og I/O:
- 00-40s CPU
- 41-110s I/O
- 111-120s CPU

a)

- Regn ut gjennomsnittelig omløpstid (Turnaround time) for batchen. Med omløpstid menes tiden det tar fra en jobb settes opp for kjøring og til den er fullført.

b)

- Regn ut gjennomstrømningen (troughput) for batchen. D.v.s. antall fullførte jobber pr. tidsenhet.

c)

- Regn ut prosessor-utnyttelsen (processor utilization) for batchen. Utnyttelsesgraden firnner du ved å regne ut hvor stor prosent av medgått tid, som prosessoren har vært aktiv.

2.2

a)

- Logg inn på debbie.hive.no og start programmet 'vimtutor' ved å skrive 'vimtutor' og trykke enter på kommandolinja. Les og følg instruksjonene.

b)

- Bruk deretter programmet 'vi' (eller 'vim') til å besvare oppgavene i "Repetisjon av maskinvarebakgrunn" over. Husk å lagre det du skriver i en fil. Kall filen 2.2.txt.

c)

- Bruk programmet 'mail' for å sende det du skrev (og lagret i fila 2.2.txt) til faglærer med kommandoen 'cat 2.2.txt | mail tn'. Kommandoen skrives på kommandolinja på debbie.

2.3

- I denne oppgaven skal du sette deg inn i hvordan unix kommandoene 'at' og 'batch' fungerer.

a)

- Logg inn på debbie og åpne manualen til kommandoen 'at' med kommandoen 'man at'. Les hvordan kommanoden 'at' og 'batch' fungerer. For å avslutte lesing av manualen kan du taste 'q'.

- For å lage en pdf-fil med penere formatering av manualen kan følgende kommandoer brukes: man -t at | ps2pdf - at.pdf

- For å se på pdf-fila kan følgende kommandoe brukes: xpdf at.pdf &

b)

- Du skal nå bruke det du leste om i a). Sett opp en batch-jobb på debbie som laster ned fila http://debbie.hive.no/osyda50/pdfs/01.pdf når prosessorbelastninger er lav. Nedlastingen skal gjøres med programmet 'wget'. Hvordan wget brukes må du selv finne ut av ved å lese i manualen. Gi kommandoen 'man wget' for å lese manualen til wget. Husk at du avslutter manualen ved å taste 'q'.

- Du får utskrift av jobben pr. e-post når jobben er gjort.

- For å lese e-post på debbie, kan du bruke programmet 'mail' (som også har en manualside).

- Hvis du vil at e-post-meldingene fra debbie skal havne sammen din vanlige e-post, kan du instruere e-post-systemet på debbie til å videresende din post til en annen konto. Dette gjør du ved å lage en tekstfil med en tekst-editor (f.eks. vim). Tekstfila må inneholde e-post-adressen du vil at systemet skal videresende posten til. Du kan også oppgi flere adresser. En adresse pr. linje. Filen må lagres med navnet '.forward'. Du kan da lese e-posten i den e-postleseren du pleier å bruke. Fila '.forward' må ligge i din hjemmekatalog. Hjemmekatalogen er den katalogen du "havner i" når du logger inn.

c)

- Sett opp en batch-jobb på debbie som sender en e-post til faglærer nøyaktig kl. 1300 i dag. Bruk følgende kommado for å sende e-post til faglærer: 'echo Takk for i dag | mail tn'.

d)

- Se om du få frem en liste over batch-jobbene dine.

Kapittel 3

03-linux-intro

3.1 Repetisjon OS-overblikk

- Hva vil du si er målsetning for et operativsystem?
- Nevn noen viktige teoretiske fremskritt i OS-utviklingen. Forklar også kort hva som menes med dem.
- Nevn noen utviklingstrekk hos moderne operativsystemer.

Hva menes med

- seriell prossessering?
- enkle system for satsvis ”batch” behandling (”simple batch system”)?
- multiprogramkjørende system for satsvis behandling (”multiprogrammed batch system”)?
- kjøretidsdeling (”time sharing”)?

3.2 Løsningsforslag til oppgave 2.1

Tenk deg at du har et flerprogramkjørende satsvis system med en bunke (batch) med tre jobber:

1.

- Tidfordeling mellom CPU og I/O:
- 00-10s CPU
- 11-50s I/O
- 51-100s CPU

2.

- Tidfordeling mellom CPU og I/O:
- 00-30s CPU
- 31-50s I/O
- 51-70s CPU

3.

- Tidfordeling mellom CPU og I/O:
- 00-40s CPU
- 41-110s I/O
- 111-120s CPU
-

```
Tegner opp jobbene.

- c er 10s med CPU-aktivitet
- i er 10s med I/O-aktivitet

1: ciiiiccccc
2: ccciicc
3: ccccciiiiiiic

Jeg gjør følgende antagelser:

- systemet har en prosesseringsenhet (enkeltkjernet CPU).
- systemet har en "round robin" fordelingsalgoritme
- de tre jobbene ber om ulike I/O tjenester (ingen køing for I/O-tjenester)

Figuren under viser hvordan de tre jobbene fordeles i tid på prosessoren.
Tiden løper fra venstre mot høyre.

- c er 10s med CPU-aktivitet
- i er 10s med I/O-aktivitet
- k er 10s hvor jobben ligger i kø, klart til å kjøre

Den nederste linjen viser hvilken jobb prosessoren jobber med til enhver tid.

ciiiikkkccccc---
-ccciikkkkkkkcc-
----ccccciiiiiiic
1222333311111223
```

a)

- Regn ut gjennomsnittelig omløpstid (Turnaround time) for batchen. Med omløpstid menes tiden det tar fra en jobb settes opp for kjøring og til den er fullført.
-

```
1 bruker 130s
2 bruker 140s
3 bruker 120s

snittet blir 130
```

b)

- Regn ut gjennomstrømningen (troughput) for batchen. D.v.s. antall fullførte jobber pr. tidsenhet.
- I løpet av 160 sekunder fullføres 3 jobber. 3 jobber/ 160 s = 0,02 jobb/sek.

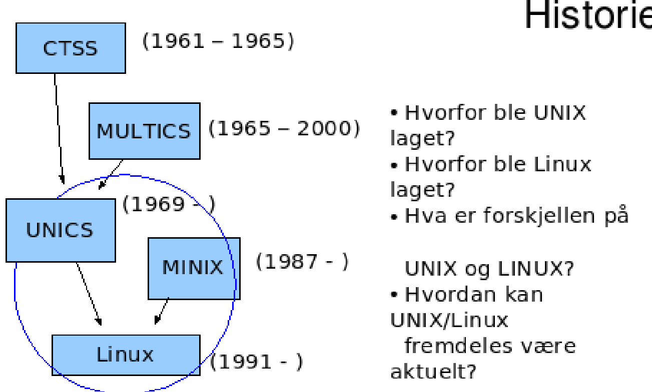
c)

- Regn ut prosessor-utnyttelsen (processor utilization) for batchen. Utnyttelsesgraden firnner du ved å regne ut hvor stor prosent av medgått tid, som prosessoren har vært aktiv.
- Ser av figuren at prosessoren har hvert aktiv hele tiden. Utnyttelsesgraden er dermed 100%

3.3 Historie/bakgrunn

historie

-



bakgrunn

•

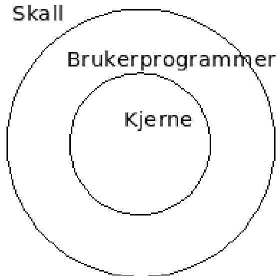
Bakgrunn

- GNU, FSF, GPL
- Distribusjoner
  - Debian
  - RedHat
  - Suse
- Standarder
  - POSIX, LSB

UNIX-arkitektur

•

UNIX-Arkitektur



• API til kjernen består av systemkall

3.4 Filbehandling

UNIX-filtyper

•

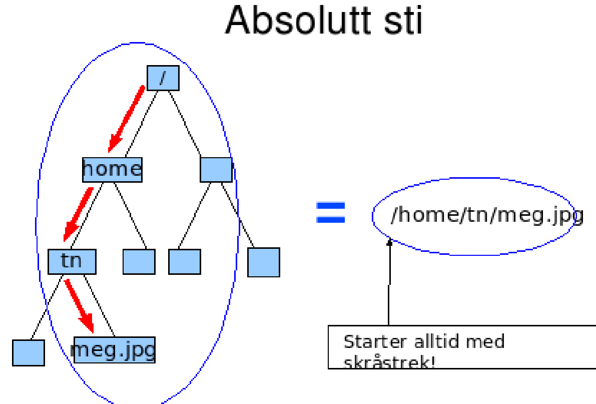
UNIX-filtyper

1. Vanlige filer
2. Kataloger
3. Karakter-enhetsfiler
4. Blokk-enhetsfiler
5. Symbolske lenker
6. "Local domain sockets"
7. Navngitte rør (FIFO's)

Hvorfor er så mye forskjellig representert som filer?

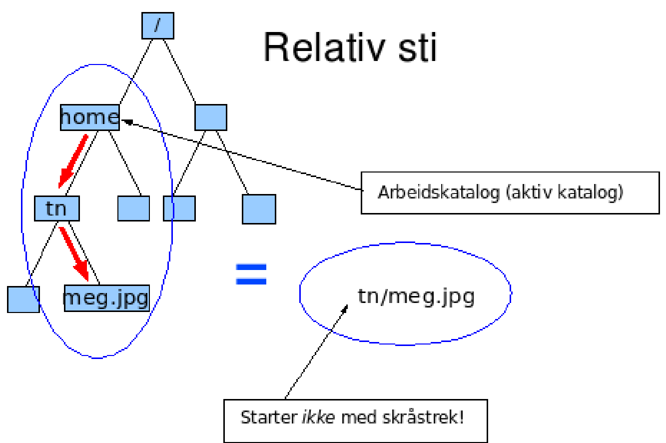
Absuolutt sti

•



Relativ sti

•



UNIX-kommandoer

•

UNIX kommanoder

- ls – lister innholdet i en katalog
- pwd – lister aktiv katalog (arbeidskatalog/ «current directory»
- mkdir – lager katalog
- rmdir – fjerner tom katalog
- rm -r – fjerner katalog med innhold (r'en står for for rekursiv)

flere unix-kommandoer

-

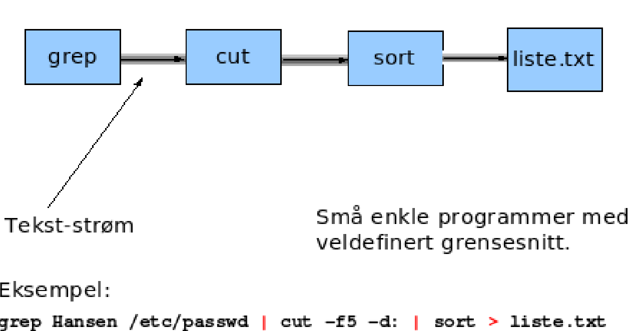
UNIX-kommandoer

- `touch`
  - endrer tids-stempel
  - oppretter tom fil hvis filen ikke eksister
- `cp` – kopierer fil
- `mv`
  - flytter
  - brukes også ved bytte navn
- `rm` - fjerner fil

3.5 Redirigering

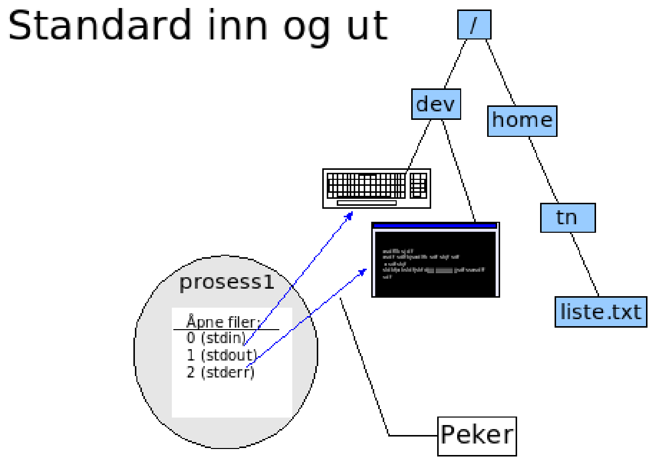
pipline

-



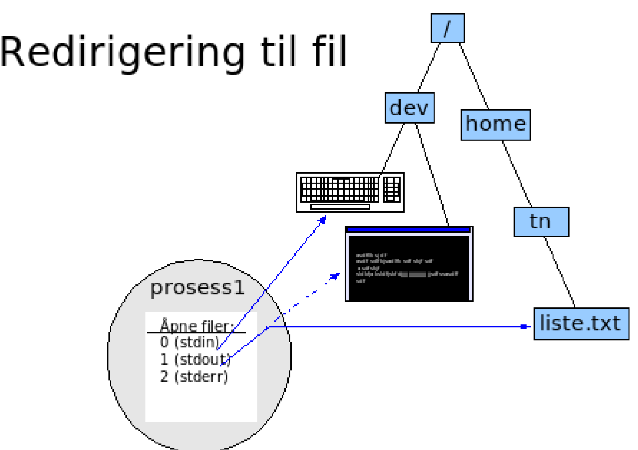
stdin og stdout

•



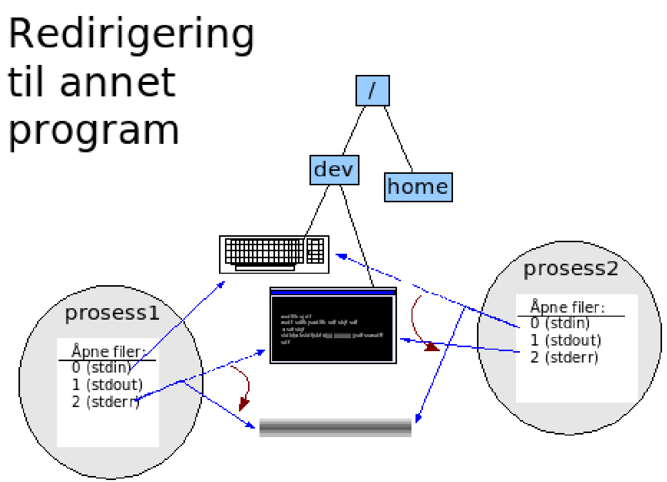
Redirigering til fil

•



Redirigering til prosess

-

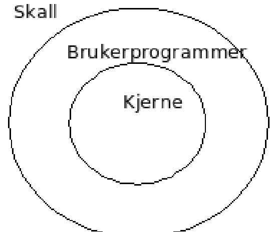


3.6 Bruk av skall (shell)

Skallets rolle?

-

Skallets rolle



Hva er formålet med skallet?

Ulike skall

•

Ulike skall

- sh - Bourne shell (skrevet av Steve Bourne)
- bash - Bourne again shell (standard skallet i linux)
- csh – C shell (mer "C-aktig" syntax enn *bash*)
- mc – Midnight Commander (Norton Commander klone)

I dette kurset konsentrerer vi oss om standardskallet *bash*.

finne hjelp

•

Hjelp

Hjelpekommandoer

- **man**
- **info**
- **--help**
- **apropos**

Linux Documentation Project:

- http://tldp.org
- http://ldp.linux.no (norsk avspeiling)

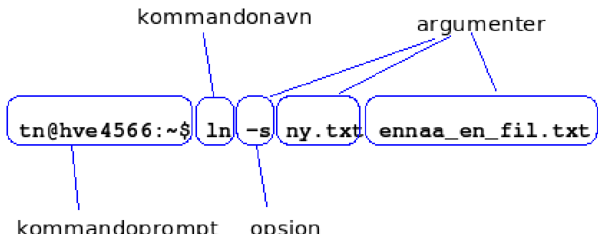
Kommandoer for å lete etter filer

- **locate**
- **find**

Kommandolinjen

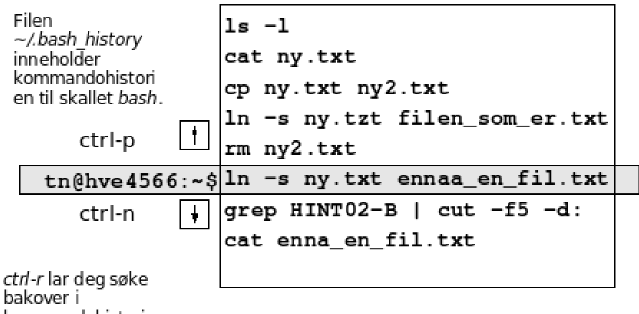
-

Kommandolinjen

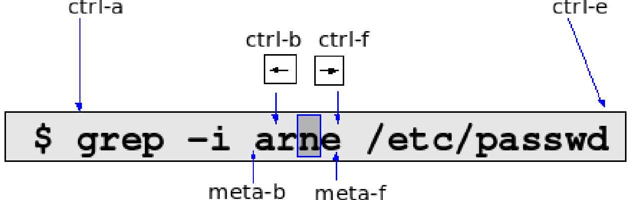


Kommandohistorie

-

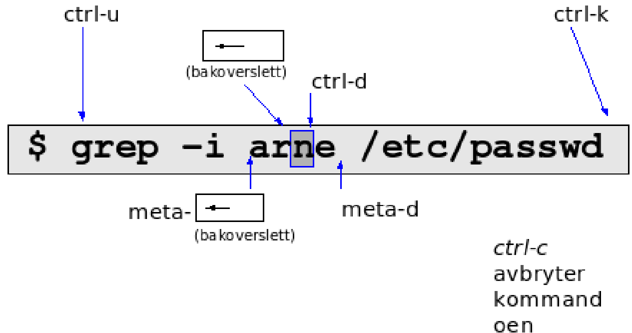


Navigering på kommandolinjen

- 

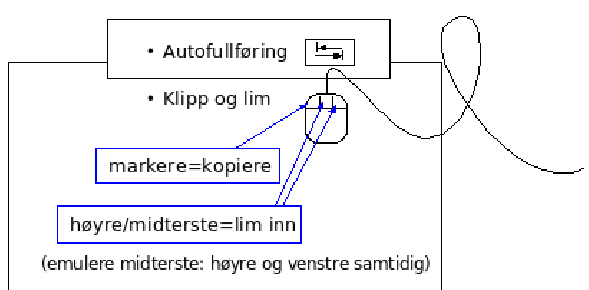
Sletting på kommandolinjen

-



Skrivehjelp

•



Jokertegn

•

Jokertegn

```
$ ls
eks1.pl  eks2.sh  eks4.sh  eks5.pl~  eks5.sh~
eks1.sh  eks3.sh  eks5.pl  eks5.sh   liste_med_kommandoer

$ echo *pl
eks1.pl eks5.pl

$ echo eks5???
eks5.pl eks5.sh
```

Omskrivningene skjer før kommandoen utføres

Andre omskrivninger

•

Andre omskrivninger

```
$ echo Vertsnavnet er $HOSTNAME
Vertsnavnet er hve4566

$ echo Arb.kat. er $(pwd)
Arb.kat. er /home/tn/public_html/net203/forelesning2/

$echo Hjemmekatalogen min er ~
Hjemmekatalogen min er /home/tn
```

Omskrivningene skjer før kommandoen utføres

3.7 Oppgave 3

- Oppgavene under er tenkt løst på kommandolinjen.

a)

- 1. Logg deg inn som en vanlig bruker
- 2. Lag tre kataloger inni hverandre

- 3. Lag en fil i den katalogen du lagde sist.
- 4. Kopier denne filen inn i de to andre katalogene du lagde.
- 5. Slett alle filene du lagde v.h.a. en enkelt kommando.

b)

- Gjennomgå tutorial på: http://people.ischool.berkeley.edu/ kevin/unix-tutorial/section4.html. Den siste oppgave. Den om filrettigheter, kan du droppe.

`http://people.ischool.berkeley.edu/~kevin/unix-tutorial/section4.html`

Kapittel 4

04 - Shell scripting

4.1 Repetisjon

- Hvilke filtyper finner vi i UNIX?
- Nevn noen kommandoer for filbehandling i UNIX.

Hva menes med

- Absuolutt sti?
- Relativ sti?
- Standard inn (stdin)?
- Standard ut (stdout)?
- Standard error (stderr)?
- pipeline?
- Hvordan får vi til redirigering til/fra filer i shellet?
- Hvordan får vi til redirigering mellom prosesser i shellet?

4.2 Shell-skripting

Innlesing av kommandoer

-

Innlesing av kommadoer

Tastaturet er representert som en fil. *Bash* leser kommandoer fra tastaturet. Kan skallet da lese kommandoer fra en annen fil?

```
HILSEN="God dag"
echo $HILSEN verden
```

filnavn: `liste_med_kommandoer`

Eksempler:

```
cat liste_med_kommandoer | bash
bash < liste_med_kommandoer
bash liste_med_kommandoer
```

Skript som kjørbar fil

-

Skript som kjørbar fil

Går det an å få en liste over kommandoer til å oppføre seg mer som et kompilert program?'

Gjøre filen kjørbar:

```
chmod u+x sti/til/liste_over_kommandoer
```

Oppgi stien til tolkeren, som skal tolke kommandoene i fila:

```
#!/bin/bash
# Kommentarer begynner
# også med '#'

HILSEN="God dag"
echo $HILSEN verden
```

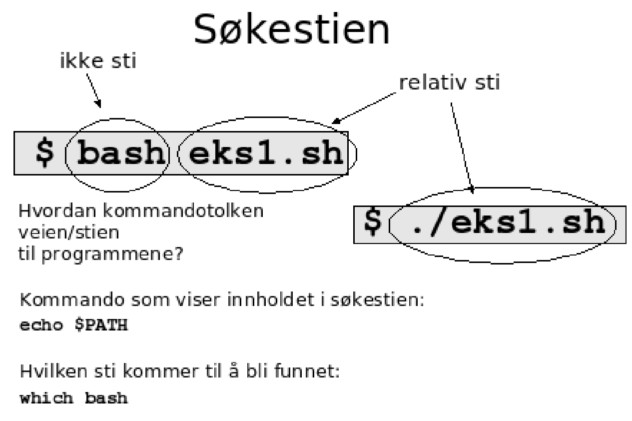
filnavn: `eks1.sh`

```
#!/usr/bin/perl
# Kommentarer i perl begynner
# også med '#'

$HILSEN="God dag";
print "$HILSEN verden\n";
```

filnavn: `eks1.pl`

Søkestien

-

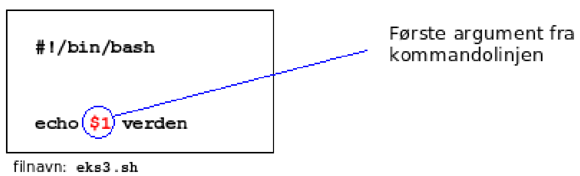


Plassere en variabel i miljøet

•

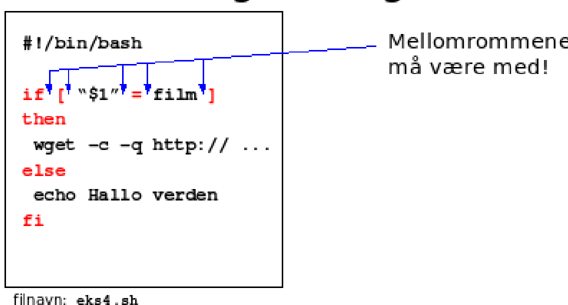
Tilgang til kommandolinjeargumenter

•



Valgsetninger

•



Løkker

•

Løkker

```
#!/bin/bash

for I in 1 2 3 4
do
  echo $I;
done
```

filnavn: eks5.sh

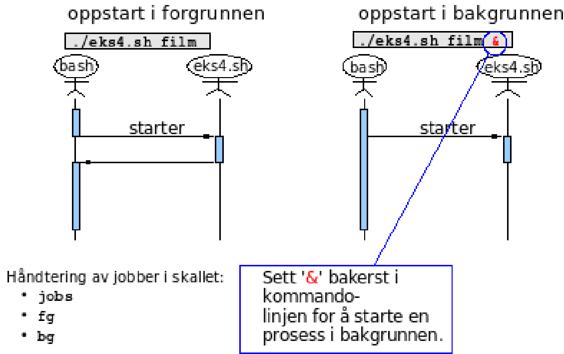
```
#!/usr/bin/perl

foreach $I (1, 2, 3, 4){
   print "$I\n";
}
```

filnavn: eks5.pl

4.3 Forgrunn/bakgrunn

•

Forgrunn/bakgrunn



4.4 Oppgaver

Noen kommandoer som kan være nyttige for oppgaveløsningen.

- Produsere en tallsekvens fra null til ti:
-

```
seq 10
```

- Løkke som skriver tallene 1 2 3:
-

```
for I in 1 2 3; do echo $I; done
```

- Skrive ut første kommandolinjeargument:
-

```
echo $1
```

- Tilordne utskriften av en kommando til en variabel:
-

```
MEG=$(whoami)
```

- Plassere en variabel i miljøet:
-

```
export MEG
```

- Opprette en tom fil
-

```
touch filnavn
```

- Kompilere C-program:
-

```
gcc -Wall filnavn.c
```

- Slette fil
-

```
rm filnavn
```

- Legge en variabel til miljøet, slik at den arves av prosessens etterkommere:
-

```
export VARIABELNAVN
```

4.1 (tidligere 3.2)

- Lag et skript som:
- 1. komplierer et C-program
- 2. kjører programmet
- 3. sletter kildekoden og den kjørbare fila

4.2 (tidligere 3.3)

a)

- Lag et I/O-intensivt og et CPU-intensivt porgram.

b)

- Lag et skript som starter begge.

c)

- Hvordan vil du få skriptet til å fullføre raskest mulig?

4.3 (tidligere 3.4)

- Lag et bash-skript som lager to tusen tomme filer i samme katalog. Filene ska hete: fil_1, fil_2, fil_3, ...
- Slett katalogene etterpå med en enkelt kommando på kommandolinjen.

4.4 (tidligere 3.5)

- Endre skripet i forrige oppgave slik at antall filer bestemmes av et kommandolinjeargument.

4.5 (tidligere 3.6)

- Utvid skriptet i forrige oppgave slik at antall filer bestemmes av en miljøvariabel (environment variable) dersom ikke kommandolinjeargument er oppgitt.

Kapittel 5

05 - Filer og filsystemer

5.1 Løsningsforslag

4.1

Oppgave

- Lag et skript som:
- 1. komplierer et C-program
- 2. kjører programmet
- 3. sletter kildekoden og den kjørbare fila

Løsningsforslag

- losninger/04/4.1.sh

Listing 5.1: losninger/04/4.1.sh

```
1 #!/bin/sh
2
3 gcc -Wall kopi.c -o kopi
4 ./kopi
5 rm kopi.c kopi
```

4.2

Oppgave

a)

- Lag et I/O-intensivt og et CPU-intensivt porgram.

b)

- Lag et skript som starter begge.

c)

- Hvordan vil du få skriptet til å fullføre raskest mulig?

Løsningsforslag 1 - med c-programmer

a)

- losninger/04/io-intensiv.c

Listing 5.2: losninger/04/io-intensiv.c

```
#include <stdio.h>
#include <unistd.h>

int main (){

  FILE *fil;
  int i;


  int buffer[BUFSIZ];

  for(i=0; i<20; i++){

    fil = fopen("film", "r");

    while ( fread(buffer,BUFSIZ,1,fil) )
      sync();

    fclose(fil);

  }

  return 0;

}
```

- losninger/04/cpu-intensiv.c

Listing 5.3: losninger/04/cpu-intensiv.c

```
#include <unistd.h>

int main (){

  double x = 3.14;
  int i,j;

  for (i=0; i < 5000; i++)
    for (j=0; j < 2500; j++)
      x=x*x;

  return 0;

}
```

b) og c)

- losninger/04/4.2.b.sh

Listing 5.4: losninger/04/4.2.b.sh

```
#!/bin/sh

# Ved aa starte det I/O-sensitive programmet foerst,
# og la det jobbe i bakgrunnen,
# vil ikke cpu-jobben maatte vente til io-jobben er ferdig.

./io &
./cpu
```

Løsningsforslag 2 - med shellskript

a)

- losninger/04/last_film.sh

Listing 5.5: losninger/04/last_film.sh

```
#!/bin/bash
wget -c -q http://www.warriorsofthe.net/mirror/warriors-700-se-VBR.
    mpg
```

- losninger/04/spill_film.sh

Listing 5.6: losninger/04/spill_film.sh

```
#!/bin/bash
vlc warriors-700-se-VBR.mpg
```

b) og c)

- losninger/04/4.2.b-alternativ.sh

Listing 5.7: losninger/04/4.2.b-alternativ.sh

```
#!/bin/sh

# Ved aa starte det I/O-sensitive programmet (filnedlastingen)
    foerst,
# og la det jobbe i bakgrunnen,
# vil ikke den cpu-intensive jobben (filmavspillingen) maatte vente
     til io-jobben (nedlastingen) er ferdig.

./last_film.sh &
./spill_film.sh
```

4.3

Oppgave

- Lag et bash-skript som lager to tusen tomme filer i samme katalog. Filene ska hete: fil_1, fil_2, fil_3, ...
- Slett katalogene etterpå med en enkelt kommando på kommandolinjen.

Løsningsforslag

- losninger/04/4.3.sh

Listing 5.8: losninger/04/4.3.sh

```
1 #!/bin/bash
2
3 for I in ${seq 2000}
4 do
5     touch fil_$I
6 done
```

- For å slette alle filene kan kommandoen 'rm fil_*' gis.

4.4

Oppgave

- Endre skripet i forrige oppgave slik at antall filer bestemmes av et kommandolinjeargument.

Løsningsforslag

- losninger/04/4.4.sh

Listing 5.9: losninger/04/4.4.sh

```
1 #!/bin/bash
2
3 for i in $(seq 1 $1)
4 do
5
6   touch fil_$i
7
8 done
```

4.5

Oppgave

- Utvid skriptet i forrige oppgave slik at antall filer bestemmes av en miljøvariabel (environment variable) dersom ikke kommandolinjeargument er oppgitt.

Løsningsforslag

- losninger/04/4.5.sh

Listing 5.10: losninger/04/4.5.sh

```
1 #!/bin/bash
2 # Skriptet forventer miljoe-variabelen ANT_FILER
3
4 if [ "$1" == "" ]
5 then
```

```

    J=$ANT_FILER

else

    J=$1
fi

for i in $(seq 1 $J)
do

  touch fil_$i

done
```

5.2 Generelt om filsystem

Filsystem

•

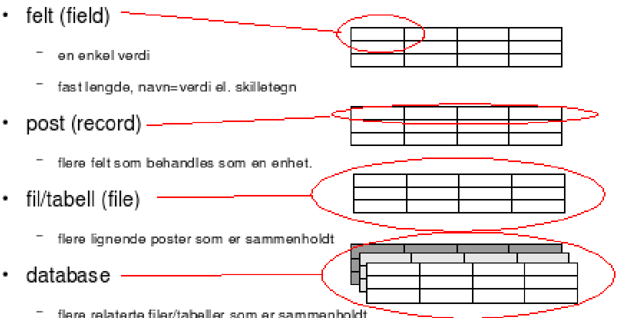
Filsystem

- Filer er viktige i mange, applikasjoner
- Mål for filsystem
  - Langtidslagring av data
  - Muligheter for deling av data mellom prosesser
  - Struktur (f.eks tre-struktur)
- Filhåndtering er helt eller delvis et OS-anliggende. Minimum betjene I/O-enheter.

Noen termer

•

Noen termer



Eksempel på fil med poster og felt: `/etc/passwd`

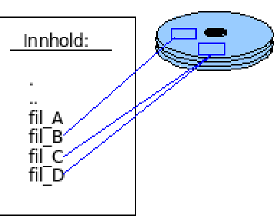
Kataloger

-

Kataloger (mapper)

Filkataloger inneholder informasjon om en samling filer

- F.eks. tidstempling, rettigheter, størrelse, eiendomsforhold, hvor filene er lagret, etc.
- For brukeren en kobling mellom filer og filers navn.
- Alle kataloger kan inneholde kataloger – slik danner katalog-strukturen en hierarkisk trestruktur.
- En bestemt katalog kalles *rotkatalogen* den kan ikke være i en annen katalog – den kan ikke ha noen *foreldrekatalog*.

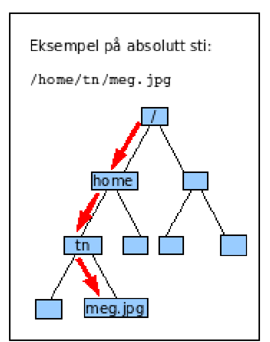


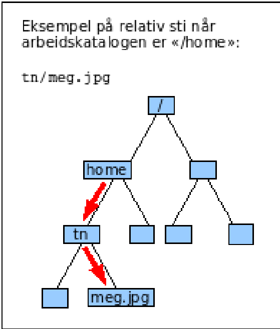
Navngivning av filer

-

Navngivning av filer

- symbolsk for brukere (i stedet for numerisk)
- hver fil må kunne nevnes entytdig





Fil-funksjonalitet

-

Typisk funksjonalitet for filhåndterings-system

- create - eks.: `creat(2)`/`open(2)`
- delete - eks.: `unlink(2)`
- open - eks.: `open(2)`
- read – eks.: read(2)
- write – eks.: write(2)
- close – eks.: close(2)
- Tilgangskontroll ...

Vi ser på et eksemel:

- `/home/thomas/osyda50/04/eksempler/open_eksempel.c`

5.3 Filhåndtering i Linux

Filtyper i Linux

-

Filtyper i Linux

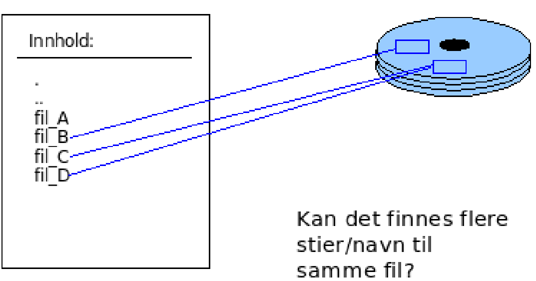
1. Vanlige filer
2. Kataloger
3. Karakter-enhetsfiler
4. Blokk-enhetsfiler
5. Symbolske lenker
6. "Local domain sockets"
7. Navngitte rør (FIFO's)

Hvorfor er så mye forskjellig representert som filer?

flere stier til samme fil?

•

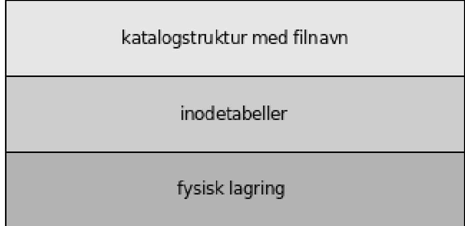
Kataloger/mapper



Forenklet lagdelt modell

•

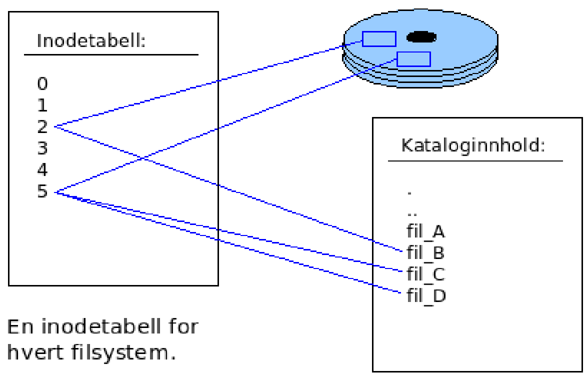
Forenklet lagdelt modell av filsystemet linux



Inodetabeller

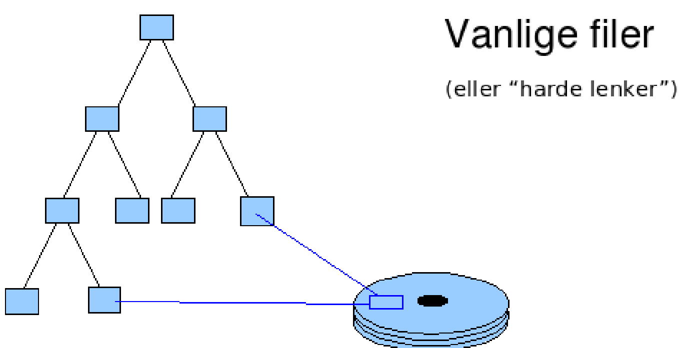
•

Inodetabeller



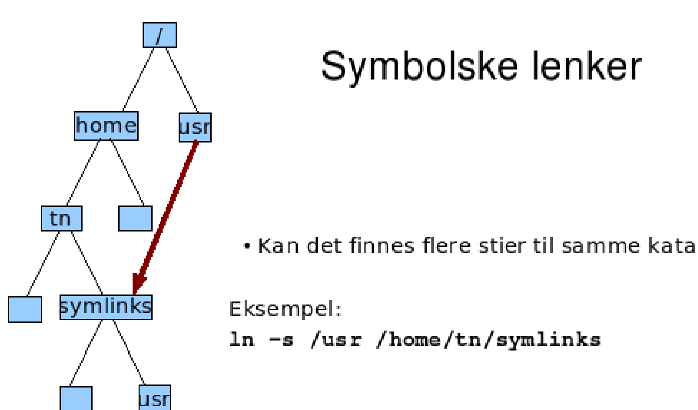
Hardlinks

•



Symlinks

•



Løkker i filsystemet

•

Løkker i filsystemet

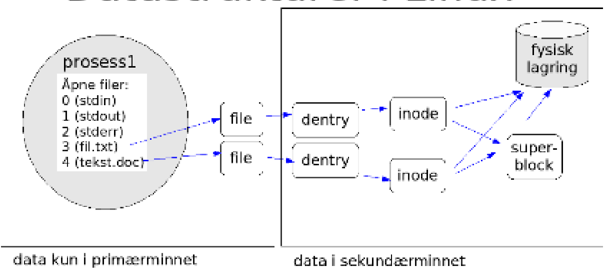
Ved bruk av symbolske lenker, kan løkker oppstå.
Kjernen oppdager løkken og gir feilmelding:

```
[eks]# mkdir testkatalog
[eks]# cd testkatalog/
[testkatalog]# ln -s ../testkatalog/filB filB
[testkatalog]# cat filB
cat: filB: For mange nivåer med symbolske linker
[testkatalog]#
```

Datastrukutrer i Linux

•

Datastrukturer i Linux

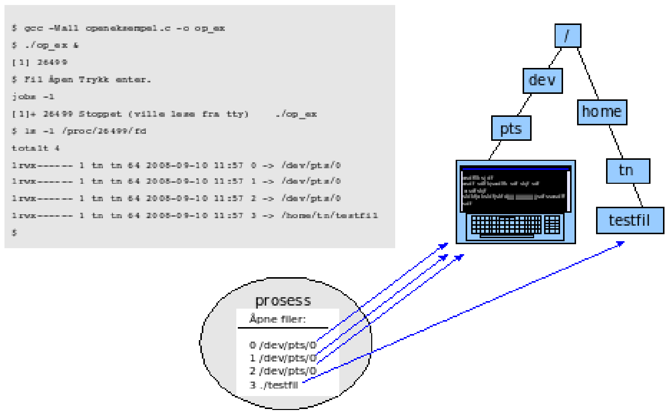


- struct file <linux/fs.h> (linje 782-819)
  http://lxr.linux.no/linux+v2.6.26.5/include/linux/fs.h#L782
- struct dentry <linux/dcache.h> (linje 81-114)
  http://lxr.linux.no/linux+v2.6.26.5/include/linux/dcache.h#L81
- struct inode <linux/fs.h> (linje 593-658)
  http://lxr.linux.no/linux+v2.6.26.5/include/linux/fs.h#L593
- struct superblock <linux/fs.h> (linje 1029-1095)
  http://lxr.linux.no/linux+v2.6.26.5/include/linux/fs.h#L1029

Eksempel: Åpne filer

•

Eksempel på åpne filer



- openeksempel.c

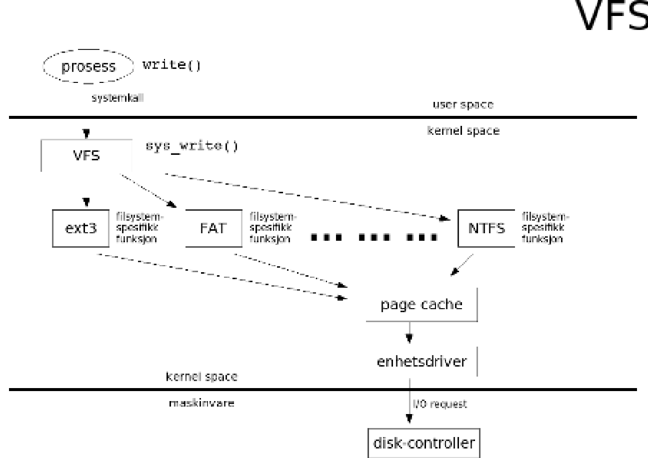
Listing 5.11: eksempler/05/openeksempel.c

```
1  #include <sys/types.h>
2  #include <fcntl.h>
3  #include <unistd.h>
4  #include <string.h>
5  #include <sys/stat.h>
6
7  /*
8
9   Demonstrasjon av systemkall:
10
11  - open(2)
12  - write(2)
13  - read(2)
14  - close(2)
15
16 */
17
18 int main()
19 {
20
21   char buffer[100];
22   int fd, ant;
23
24   fd = open("testfil", O_RDWR | O_EXCL | O_CREAT, S_IRUSR);
25
26   if (fd < 0){
27
28     strcpy(buffer, "Feil ved aapning av fil.\n");
29     ant=strlen(buffer);
30
31     write(2, buffer, ant);
32     return 1;
33
34   }
35
```

```
36    strcpy(buffer,"Fil aapen. Trykk enter.\n");
37    ant=strlen(buffer);
38
39    write(1, buffer, ant);
40    read(0, NULL, 1);
41
42    close(fd);
43
44    strcpy(buffer, "Fil lukket.\nTrykk enter.\n");
45    ant=strlen(buffer);
46
47    write(1, buffer, ant);
48    read(0, NULL, 1);
49
50    unlink("testfil");
51
52    return 0;
53 }
```

Virtual File System

-



Opprettelse av filsystem

-

Opprettelse av filsystem

Ved opprettelse av et filsystem, blir det skrevet noen datastrukturer til disken.

Kommando for å skrive filsystem til disken:

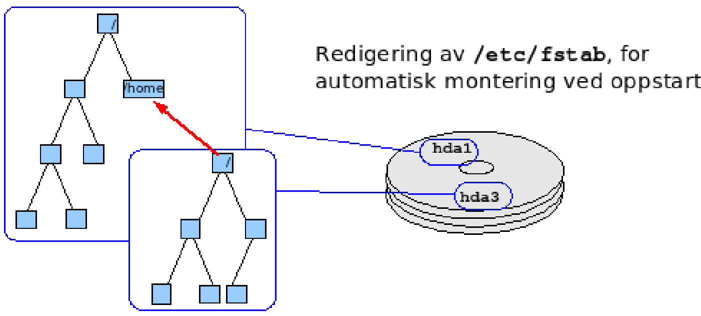
- **`mkfs(8)`**

Filen **`/proc/filsystems`** inneholder en liste over hvilke filsystemer den kjørende kjernen har støtte for.

Montering av filsystem

•

Montering av filsystemer



Eksempel:
`mount /dev/hda3 /home`

5.4 Oppgaver

- Noen av oppgavene under krever at du har root-tilgang. Du kan f.eks. bruke lab-maskinene eller egen PC. Det er også mulig å løse oppgavene på debbie.
- Dersom du ønsker å gjørre oppgaven på debbie.hive.no, kan du starte en virtuell maskin med kommanden: 'vstart VMNAVN', hvor VMNAVN

er navnet du gir til den virtuelle maskinen. Denne kommandoen starter en virtuell maskin og et konsoll-vindu hvor du automatisk logges inn som root. Se vstart(1). Tallet i parentes etter kommandoenavnet angir seksjonen i manualen. For å lese manualen: 'man 1 vstart'. For mer informasjon om den virtuelle maskinen, for de spesielt interesserte, se: http://netkit.org.

- I oppgavene under finner du mange referanser til kommandoer med et tall i parentes bak. F.eks. slik: cp(1). Tallet i parentes etter kommandoenavnene angir seksjonen i manualen. Seksjonsnummeret skrives før kommandonavnet. F.eks. slik: 'man 1 cp'.

5.1 Lenker

- For å løse denne oppgaven skal du bruke kommandoene: ln(1), rm(1). I tillegg trenger du å bruke en editor av fritt valg (emacs(1), vim(1), nano(1), mcedit(1), ...).

5.1.1 Harde lenker

- a) Opprett en fil med navn A og med teksten "Dette er fil A".
- b) Lag en hardlink med navn B, til filen A.
- c) Slett deretter A.
- d) Hva har skjedd med B?
- e) Lag en ny fil med navn A og med teksten "Dette er NYE fil A".
- f) Hva er nå skjedd med B?

5.1.2 Symbolske lenker

- a) Opprett en fil med navn A og med teksten "Dette er fil A".
- b) Lag en softlink med navn B, til filen A.
- c) Slett deretter A.
- d) Hva har skjedd med B?
- e) Lag en ny fil med navn A og med teksten "Dette er NYE fil A".
- f) Hva er nå skjedd med B?

5.2 Navnelister

- I denne oppgaven trenger du å bruke programmene cut(1), sort(1), uniq(1) og filen /etc/passwd.
- a) Lag en kommandolinje («pipeline») som skriver en alfabetisk sortert liste over brukernavn på et UNIX/Linux-system til en fil ved navn brukernavn.txt. Prøv den på debbie.
- b) Lag en kommandolinje («pipeline») som skriver en alfabetisk sortert liste over alle brukernes etternavn. Forsøk å unngå duplikate etternavn. Prøv den på debbie.

5.3 Systemkall

- Rediger eksemplet 'openeksempel.c' slik at programmet lukker standard utgang før testfil åpnes. Du finner eksemplet i fila /home/tn/osyda50/eksempler/05/openeksempel.c på debbie,
- Sørg også for at testfil ikke slettes før programmet terminerer.
- Kompiler og kjør programmet. Ser du noen forskjell på utskriften?
- Hva er innholdet i testfil?
- Kan du forklare hva som er skjedd?

5.4 Montering

- For å gjøre denne oppgaven må du ha root-tilgang. Du kan f.eks. bruke lab-maskinene, egen pc eller virtuell-maskin på debbie.
- I denne oppgaven trenger du å bruke dd(1), zero(4), mkfs(8), mount(8), cp(1) og umount(8)

a) Lag en fil med størrelse 3 MB

- Du kan bruke kopieringsprogrammet dd(1). Bruken av dd er litt uortodoks. Det stammer fra stammer fra JCL (Job Control Language).
- Det er fin artikkel om 'dd', med eksempler, på wikipedia http://en.wikipedia.org/wiki/Dd_(Unix).
- Du kan f.eks. fylle en fil med nuller ved å kopiere fra spesialfilen /dev/zero – zero(4).
- b) Lag et filsystem av valgfri type på filen v.h.a mkfs(8).

c) Monter filsystemet og kopier noen filer over på det.

- Du må vanligvis være 'root' for å kunne montere et filsystem.
- Til slutt avmonterer du filsystemet.
- d) Legg inn en linje filen /etc/fstab, slik at filsystemet monteres ved oppstart.
- e) Gjør en om start av systemet med kommandoen reboot(8), og kontroller at monteringen ble utført korrekt ved oppstart.

Kapittel 6

06 - Filsystem-sikkerhet og brukere i Linux

6.1 Løsningsforslag

5.1.1 Harde lenker

Oppgave

- a) Opprett en fil med navn A og med teksten "Dette er fil A".
- b) Lag en hardlink med navn B, til filen A.
- c) Slett deretter A.
- d) Hva har skjedd med B?
- e) Lag en ny fil med navn A og med teksten "Dette er NYE fil A".
- f) Hva er nå skjedd med B?

Løsning

- losninger/05/5.1.1.txt

`http://debbie.hive.no/osyda50/losninger/05/5.1.1.txt`

5.1.2 Symbolske lenker

Oppgave

- a) Opprett en fil med navn A og med teksten "Dette er fil A".
- b) Lag en softlink med navn B, til filen A.
- c) Slett deretter A.
- d) Hva har skjedd med B?
- e) Lag en ny fil med navn A og med teksten "Dette er NYE fil A".
- f) Hva er nå skjedd med B?

Løsning

- losninger/05/5.1.2.txt

`http://debbie.hive.no/osyda50/losninger/05/5.1.2.txt`

5.2 Navnelister

Oppgave

- a) Lag en kommandolinje («pipeline») som skriver en alfabetisk sortert liste over brukernavn på et UNIX/Linux-system til en fil ved navn brukernavn.txt. Prøv den på debbie.

- b) Lag en kommandolinje («pipeline») som skriver en alfabetisk sortert liste over alle brukernes etternavn. Forsøk å unngå duplikate etternavn. Prøv den på debbie.

Løsning

- cut -f1 /etc/passwd -d: | sort > brukernavn.txt

5.3 Systemkall

Oppgave

- Rediger eksemplet 'openeksempel.c' slik at programmet lukker standard utgang før testfil åpnes. Du finner eksemplet i fila /home/tn/osyda50/eksempler/05/openeksempel.c på debbie,

- Sørg også for at testfil ikke slettes før programmet terminerer.

- Kompiler og kjør programmet. Ser du noen forskjell på utskriften?

- Hva er innholdet i testfil?

- Kan du forklare hva som er skjedd?

Løsning

- losninger/05/5.3.c

Listing 6.1: losninger/05/5.3.c

```
1  #include <sys/types.h>
2  #include <fcntl.h>
3  #include <unistd.h>
4  #include <string.h>
5  #include <sys/stat.h>
6
7  /*
8
9   Demonstrasjon av systemkall:
10
11  - open(2)
12  - write(2)
```

```
13  - read(2)
14  - close(2)
15
16 */
17
18 int main()
19 {
20
21   char buffer[100];
22   int fd, ant;
23
24   // Endring 1: Lagt til linjen under.
25   close(1);
26
27   fd = open("testfil", O_RDWR | O_EXCL | O_CREAT, S_IRUSR);
28
29   if (fd < 0){
30
31     strcpy(buffer, "Feil ved aapning av fil.\n");
32     ant=strlen(buffer);
33
34     write(2, buffer, ant);
35     return 1;
36
37   }
38
39
40   strcpy(buffer,"Fil aapen. Trykk enter.\n");
41   ant=strlen(buffer);
42
43   write(1, buffer, ant);
44   read(0, NULL, 1);
45
46   close(fd);
47
48   strcpy(buffer, "Fil lukket.\nTrykk enter.\n");
49   ant=strlen(buffer);
50
51   write(1, buffer, ant);
52   read(0, NULL, 1);
53
54   // Endring 2: Kommentert bort linjen under
55   //  unlink("testfil");
56
57   return 0;
58 }
```

- Når standard ut lukkes, blir fildeskriptor nr. 1 ledig i tabellen over åpne filer. Kallet til open() returnerer laveste ledige fildeskriptor, nemlig fildeskriptor 1. Utskriften vil dermed havne i fila, istedet for på konsollet.

5.4 Montering

a)

Oppgave

- Lag en fil med størrelse 3 MB

Løsning

- dd if=/dev/zero of=/filsystem count=3 bs=1M

b)

Oppgave

- Lag et filsystem av valgfri type på filen v.h.a mkfs(8).

Løsning

- mkfs.ext3 /filsystem

c)

Oppgave

- Monter filsystemet og kopier noen filer over på det.
- Du må vanligvis være 'root' for å kunne montere et filsystem.
- Til slutt avmonterer du filsystemet.

Løsning

- mkdir /monteringspunkt
- mount -o loop /filsystem /monteringspunkt
- cp /etc/passwd /etc/shadow monteringspunkt/
- umount /monteringspunkt

d)

Oppgave

- Legg inn en linje filen /etc/fstab, slik at filsystemet monteres ved oppstart.

Løsning

- echo /filsystem /monteringspunkt/ ext3 loop 0 0 » /etc/fstab

e)

Oppgave

- Gjør en om start av systemet med kommandoen reboot(8), og kontroller at monteringen ble utført korrekt ved oppstart.

Kommentar

- På den virtuelle maskinen som kan startes med vstart(1), blir ikke kommandoen 'mount -a' kjørt ved oppstart, slik den vanligvis blir. Denne må derfor kjøres manuelt (eller legges til en av oppstartsfilene) etter omstart.

6.2 Filsystem-sikkerhet

ACM

•

Access Control Matrix

- Generell modell av system for tilgangskontroll
  - mengden av *rettigheter*: R.
  - mengden av *objekter* som skal beskyttes: O
  - mengden av handlende objekter – *subjekter*: S
  - matrisen A med elementer a[s,o]
  - når $s \in S$, $o \in O$ gir $a[s,o] \subseteq R$
  - *eks.*
    - $R=\{r,w,x\}$
    - $S=\{s_1,s_2\}$
    - $O=\{o_1,o_2\}$

A=

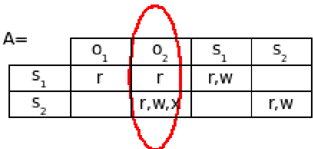
	o_1	o_2	s_1	s_2
s_1	r	r	r,w	
s_2		r,w,x		r,w

ACL

•

Access Control Lists

- Elementer *med innhold* i en *kolonne* i matrisen tilsvarer en ACL



A=

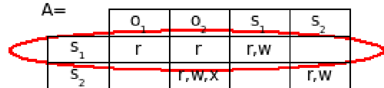
	o_1	o_2	s_1	s_2
s_1	r	r	r,w	
s_2		r,w,x		r,w

Capability Lists

•

Capability Lists

- Elementer *med innhold* i en *rad* i matrisen tilsvarer en *Capability List*.



ACL i linux

•

ACL i Linux

- trad. UNIX tilnærming – minimal ACL

		User	Group	Others
		Eieren	Gruppemedlemmer	Alle Brukere
R	Lese	0/1	0/1	0/1
W	Skrive	0/1	0/1	0/1
X	Utføre	0/1	0/1	0/1

0 – 7 0 – 7 0 – 7
rwx rwx rwx
U G O

Filrettigheter: Notasjon

•

Notasjon av filrettigheter i UNIX/Linux

Eksempel:

```
-rw-rw-r--  1 tn tn   20K jan  5 15:16 foreles1.sxi
```

Binær notasjon:	110	110	100
Otal notasjon:	6	6	4

6.3 Brukere i Linux

Brukerkonto

•

Brukerkonto

- En brukerkonto består av filer, resursser og informasjon som tilhører en bruker.
- Tilhørighet til en bruker, angis med et bruker-nummer (UID)

Brukerdatabasen /etc/passwd

•

"Brukerdatabasen"

brukernavn
kryptert passord (dersom ikke skyggepassord er installert)
brukernummer (UID)
gruppenummer (GID)
GECOS (fulltnavn, kontor nummer og bygning)
sti til hjemmekatalog
sti til innloggings-skall

```
tippmann:x:1117:100:Lars Kjærland,02HINT-B,,:/home/tippmann:/bin/bash
laktose:x:1118:100:Herbjørn Portaasen,02HINT-B,,:/home/laktose:/bin/bash
filip:x:1119:100:Filip Kandal Gjersøe,02HINT-B,,:/home/filip:/bin/bash
kenjo:x:1120:100:Kenneth Johannessen,03HINT-B,,:/home/kenjo:/bin/bash
krset:x:1121:100:Kristian Syse Setsaas,02HINT-B,,:/home/krset:/bin/bash
janterje:x:1122:100:Jan Terje Nilsen,02HINT-B,,:/home/janterje:/bin/bash
```

utdrag fra filen **/etc/passwd**

Eksempler på programmer som slår opp i **/etc/passwd**:

- **finger**
- **ls**

Skyggepassord - /etc/shadow

-

Skyggepassord

brukernavn
kryptert passord
dato for siste endring av passord
minumum antall dager mellom passordendringer
maximum antall dager mellom passordendringer
antall dager før utløpsdato, som advarsel skal gis
antall dager etter utløpsdato, som kontoen blir deaktivert
kontoens utløpsdato
reservert for fremtidig bruk

```
terjef:mXTveXqIac6teX:12288:0:99999:7:0::
truls:troInJZAoZeTg:12288:0:99999:7:0::
johob:jo/KNvll8E:12288:0:99999:7:0::
ifreska:ifUFS7v3schXuMmE:12288:0:99999:7:0::
manou:magzAUa2jtKzI:12288:0:99999:7:0::
bravoman:brZ5rmGcPeDQ:12288:0:99999:7:0::
```

utdrag fra filen **/etc/shadow**

Opprettelse av brukere

-

Opprette brukere

Følgende utføres ved opprettelse av en ny bruker:

- Ny linje i filen **/etc/passwd**
- Endringer i filen **/etc/group**
- (Ny linje i filen **/etc/shadow**)
- Opprette hjemmekatalog
- Kopiere innholdet i **/etc/skel** til hjemmekatalogen
- Endre eierskap og rettigheter på hjemmekatalogen (med innhold).
- Eventuelle andre tilpasninger

Eksempler på kommanoder som oppretter ny bruker:
- **adduser(8)**
- **useradd(8)**

Deaktivering av brukere

-

Deaktivere brukere

- Deakivere passordet – umulig for brukern å logge inn v.h.a passord
  - sette et utropstegn foran det krypterte passordet
  - **passwd -l**
- Fjerne mulighetene for brukeren å få et shell når han logger inn
  - endre stien til oppstarts-skallet i **/etc/passwd** til et program som ikke er et shell. F.eks. /bin/false

Fjerne brukere

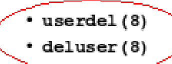
-

Fjerne brukere

Dette må gjøres for å fjerne en brukerkonto:

- Alle brukerens filer må fjernes
- Alle brukerens prosesser må stoppes
- Evt. annet.

Eksempler på programmer som kan fjerne brukerkonti:



- Underveisevaluering

6.4 Oppgaver

6.1 Brukerhåndtering

- For å gjøre denne oppgaven må du ha root-tilgang. Du kan f.eks. bruke lab-maksinene, egen pc eller virtuell-maskin på debbie.

a)

- Lag en brukerkonto for Dolly Duck med adduser(8)

b)

- Du trenger å bruke en editor, mkdir(1), cp(1) og chown(1), for å gjøre denne oppgaven.

- Lag en brukerkonto for Skrue McDuck manuelt ved å gjøre alle trinnene beskrevet tidligere i notatene. Du skal altså redigere /etc/passwd med en editor o.s.v.

c)

- Deaktiver onkel Skrues konto, og kontroller at han ikke kan logge inn.

d)

- Slett Dollys konto med deluser(8)

e)

- Slett Skrues konto v.h.a. rm(8) og en editor.

6.2 Hemmelig passordfil

- For å gjøre denne oppgaven må du ha root-tilgang. Du kan f.eks. bruke lab-maskinene, egen pc eller virtuell-maskin på debbie.
- Du bestemmer deg for at det er for usikkert at ordinære brukere kan lese brukerdatabasen (/etc/passwd) i linux-systemet ditt. Derfor endrer du rettighetene slik at det bare er mulig for brukeren root å lese den. Noter opprinnelige rettigheter. Gjør endringen v.h.a. kommandoen chmod(1).
- Logg inn som en ordinær bruker. Merker du noe forskjell? Kan du forklare hva som har skjedd? Endre rettighetene tilbake til det opprinnelige – som du noterte.

Kapittel 7

07-prosesser

7.1 Løsningsforlag

6.1 (tidligere 5.5) Brukerhåndtering

- For å gjøre denne oppgaven må du ha root-tilgang. Du kan f.eks. bruke lab-maksinene, egen pc eller virtuell-maskin på debbie.

a)

Oppgave

- Lag en brukerkonto for Dolly Duck med adduser(8)

Løsningsforlag

- losninger/06/6.1.a.txt

Listing 7.1: losninger/06/6.1.a.txt

```
Script started on Tue 28 Sep 2010 06:33:33 AM UTC
test1:~# adduser dolly
Adding user `dolly' ...
Adding new group `dolly' (1001) ...
Adding new user `dolly' (1001) with group `dolly' ...
Creating home directory `/home/dolly' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for dolly
Enter the new value, or press ENTER for the default
	Full Name []: Dolly Duck
	Room Number []:
	Work Phone []:
	Home Phone []:
	Other []:
Is the information correct? [Y/n]
test1:~# exit

Script done on Tue 28 Sep 2010 06:34:26 AM UTC
```

b)

Oppgave

- Lag en brukerkonto for Skrue McDuck manuelt ved å gjøre alle trinnene beskrevet tidligere i notatene. Du skal altså redigere /etc/passwd med en editor o.s.v.

Løsningsforslag

- losninger/06/6.1.b.sh

Listing 7.2: losninger/06/6.1.b.sh

```
1 #!/bin/sh
2 echo "skrue:x:1002:1002:Skrue_Mc_Duck,,,:/home/skrue:/bin/bash" >>
      /etc/passwd
3 echo "skrue:x:1002" >> /etc/group
4 echo "skrue:*:14880:0:99999:7:::" >> /etc/shadow
5 cp -r /etc/skel/ /home/skrue
6 chown -R skrue.skrue /home/skrue
7 passwd skrue
```

c)

Oppgave

- Deaktiver onkel Skrues konto, og kontroller at han ikke kan logge inn.

Løsningsforslag

- Kommandoen 'passwd -l skrue' vil kun sørge for at skrue ikke kan logge inn med passord. Men dersom Skrue kan autentiseres på andre måter (f.eks. med offentlig nøkkel), vil han allikevel kunne logge inn. Ved å gi kommandoen 'chage -E0 skrue', blir utløpsdatoen satt til dag 0 i år 0 ("etter UNIX"), d.v.s 1. jan. 1970. Skrues konto der dermed deaktivert (forutsatt at klokka på maskinen går riktig).

d)

Oppgave

- Slett Dollys konto med deluser(8)

Løsningsforslag

- deluser –remove-all-files dolly

e)

Oppgave

- Slett Skrues konto v.h.a. rm(8) og en editor.

Løsningsforslag

- Med en editor sletter du linjene, som gjelder skrues konto, i filene passwd(5), group(5) og shadow(5).
- For å slette skues hjemmekatalog kan du bruke kommandoen 'rm -r /home/skrue'.
- Dersom du er usikker på om skrue har andre filer rundt omkring, som bør slettes, kan du f.eks. bruke programmet find(1) for finne dem. Kommandoen 'find / -nouser' finner eierløse filer.

Kommentar

- Bruk helst editoren vipw(8) for å redigere filene /etc/passwd, /etc/group og /etc/shadow. Denne sørger for å låse filen som redigeres, slik at ingen andre gjør endringer (f.eks. endrer passord), mens du redigerer.
- Default editor for vipw(8) er vi(1). Ved å sette miljøvariabelen $EDITOR til stien til din favoritt-editor vil vipw(8) bruke denne i stedet.

Eksempel:

- På 'netkit virtuell maskin', som kan startes med vstart(1) på debbie, finner du den brukervennlinge editoren mcedit(1). Denne har et menybasert og ganske intuitivt grensesnitt. Du kan bruke både mus og funksjonstaster.
- For å finne denne stien bruker du kommandoen 'which mcedit'. Kommandoen 'export EDITOR=$(which mcedit) setter variabelen i miljøet med korrekt sti.

6.2 (tidligere 5.6) Hemmelig passordfil

Oppgave

- Du bestemmer deg for at det er for usikkert at ordinære brukere kan lese brukerdatabasen (/etc/passwd) i linux-systemet ditt. Derfor endrer du rettighetene slik at det bare er mulig for brukeren root å lese den. Noter opprinnelige rettigheter. Gjør endringen v.h.a. kommandoen chmod(1).
- Logg inn som en ordinær bruker. Merker du noe forskjell? Kan du forklare hva som har skjedd? Endre rettighetene tilbake til det opprinnelige – som du noterte.

Løsning

- losninger/06/6.2.txt

Listing 7.3: losninger/06/6.2.txt

```
1 Script started on Tue 28 Sep 2010 09:37:23 AM UTC
2 test1:/hosthome/osyda50/losninger/06# stat /etc/passwd
3   File: '/etc/passwd'
```

```
4   ␣␣Size:␣1637␣␣␣␣␣␣␣␣Blocks:␣4␣␣␣␣␣␣␣␣␣␣IO␣Block:␣1024␣␣␣regular␣
      file
5   Device:␣6201h/25089d␣␣Inode:␣54957␣␣␣␣␣␣␣Links:␣1
6   Access:␣(0644/-rw-r--r--)␣␣Uid:␣(␣␣␣␣0/␣␣␣␣root)␣␣␣Gid:␣(␣␣␣␣0/␣␣␣␣
      root)
7   Access:␣2010-09-28␣09:37:23.000000000␣+0000
8   Modify:␣2010-09-28␣09:32:41.000000000␣+0000
9   Change:␣2010-09-28␣09:37:12.000000000␣+0000
10  test1:/hosthome/osyda50/losninger/06#␣chmod␣600␣/etc/passwd
11  test1:/hosthome/osyda50/losninger/06#␣stat␣/etc/passwd
12  ␣␣File:␣'/etc/passwd'
13    Size: 1637         Blocks: 4          IO Block: 1024   regular
      file
14  Device: 6201h/25089d  Inode: 54957       Links: 1
15  Access: (0600/-rw-------)  Uid: (    0/    root)   Gid: (    0/
      root)
16  Access: 2010-09-28 09:37:27.000000000 +0000
17  Modify: 2010-09-28 09:32:41.000000000 +0000
18  Change: 2010-09-28 09:37:35.000000000 +0000
19  test1:/hosthome/osyda50/losninger/06# login
20  test1 login: dolly
21  Password:
22  Last login: Tue Sep 28 09:35:14 UTC 2010 on tty0
23  I have no name!@test1: I have no name!@test1:~$ ls /home -l
24  total 3
25  drwxr-xr-x 2 1001 dolly    1024 2010-09-28 09:35 dolly
26  drwxr-xr-x 2  110 nogroup 1024 2008-12-06 16:50 mftp
27  drwxr-xr-x 2 1000 guest    1024 2008-12-06 16:53 guest
28  I have no name!@test1: ~I have no name!@test1:~$ logout
29  test1:/hosthome/osyda50/losninger/06#
```

7.2 Om prosesser

Problemet ved utviklingen av "batch"-, "time share" og "real time"-OS

- feilaktig synkronisering
- feil ved gjensidig utelukkelse
- uberegnelig utfall av progamkjøring
- vranglåser

Mot en løsning

- Problemene imøtegåes med systematisk måte å kontrollere programutførelsene
- Begrepet "process" ble først brukt under utvikling av Multics på 60-tallet

Begrepet Prosess - ulike definisjoner

- Et program under utførelse
- En instans av et program som kjører på en datamaskin
- Entitet som kan tildeles og kjøres på en prosessor

En enhet av aktivitet karakterisert av

- en en enkel sekvensiell tråd av utførelse
- en tilstand
- tilhørende system resursser
- «an address space with one or more threads executing within that address space, and the required system resources for those threads.» [Single UNIX Specification, Versjon1 (UNIX95) og Versjon 2 (UNIX98)]

Prosess består av

- et kjørbart program
- tilhørende data

kjøre-kontekst

- info nødveding for å kontrollere prosessen
- f.eks. ulike prosessor-registre, prioritet, tid-brukt, etc.

7.3 Representasjon/beskrivelse av prosesser

- Operativsystemets representasjon av en prosess kan kalles et "Process Image"

Process Image kan deles inn i

Adresserom - for programkjøring

data

- data og program som skal kunne modifiseres
- i "userspace"

program

- maskinkoden som skal utføres
- "read only"

stakk

- LIFO (minst en) for lagring ved prosedyrer og systemkall

Process Control Block (PCB) - for prosesshåntering

Prosess-identifikasjon

- ID for prosessen, brukeren, foreldreprosess, etc.

Eksempel fra Linux-kjernen:

•

`struct task_struct {...pid_t pid;...struct task_struct *parent;     /* parent process */..`

Finne prosess-identifikator

•

`pid_t getpid(void);pid_t getppid(void);`

- getpid(2)

Eksempler

- getpid-eksempel.c

Listing 7.4: eksempler/07/getpid-eksempel.c

```
#include <unistd.h>
#include <stdio.h>

/*

 Programmet skriver ut sin egen
 prosess-identifikator

*/

int main(void)
{
  int pid;

  pid = getpid();
  printf("Jeg er prosess nr. %d\n", pid);
  return 0;

}
```

- echo $$

Prosessor-tilstand

- Verdien på ulike registre

Informasjon for prosess-kontroll

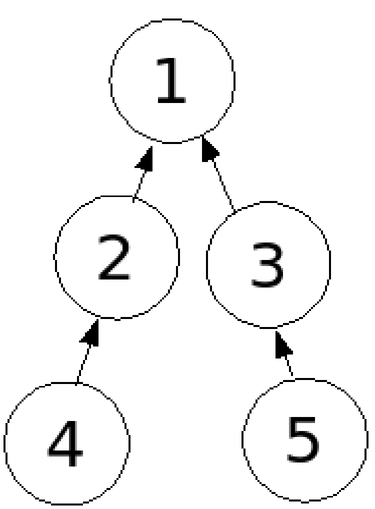
- Skedulering og tilstandsinformasjon
- Strukturering
- Interprosess-kommunikasjon
- Privilegier
- Minnehåndtering
- Tilknytning til resursser
- Måling av resurss-bruk
- task_struct

`http://lxr.linux.no/#linux+v2.6.31/include/linux/sched.h#L1166`

Vi ser på prosesser i Linux

Prosess-treet i Linux

Prosessene er organsiert i en trestruktur.

- 

Hver prosess har en foreldreprosess (bortsett fra den første, init)

-

```
struct task_struct { ... struct task_struct *parent; ...}
```

- (task_struct)

http://lxr.linux.no/#linux+v2.6.31/include/linux/sched.h#L1166

Eksempel på kommandoer som viser prosess-treet:

- pstree(1)
- ps –forest

Programmer for overvåking av prosesser:

- ps(1)
- pstree(1)
- top(1)

Et spesialfilsystem som gir et "filaktig" grensesnitt til prosessenes ressurser:

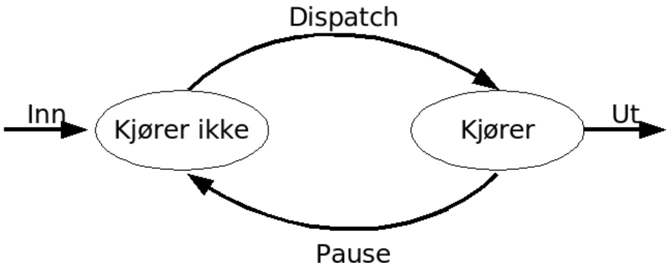
- /proc

7.4 Noen tilstands-modeller

modell med to tilstander

Kun en prosess kan kjøre om gangen (pr. prosessor)

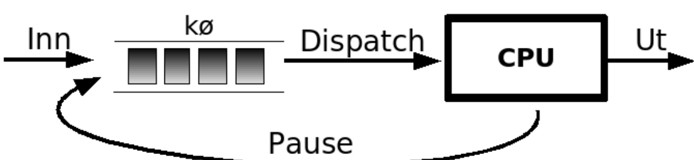
figur

- 

- Kjørende (running) - en prosess pr. prosessor
- Klar (ready) - kan kjøre umiddelbart dersom valgt av "dispatcher"

Kø i modell med to tilstander

figur

- 

- En kø av ventende prosesser

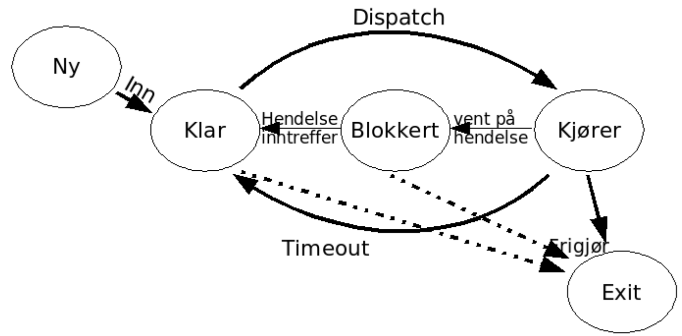
Elementene i køen kan f.eks. være

- pekere til PCB'er
- sammenlenkede PCB'er

modell med fem tilstander

figur

-

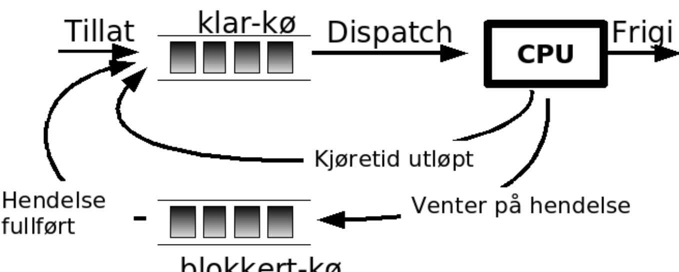


- Ikke alle prosesser er klare til å kjøre til enhver tid.
- Blokkert (blocked) venter på hendelse (f.eks fullføring av I/O-operasjon)

Fem tilstander – to køer

figur

-

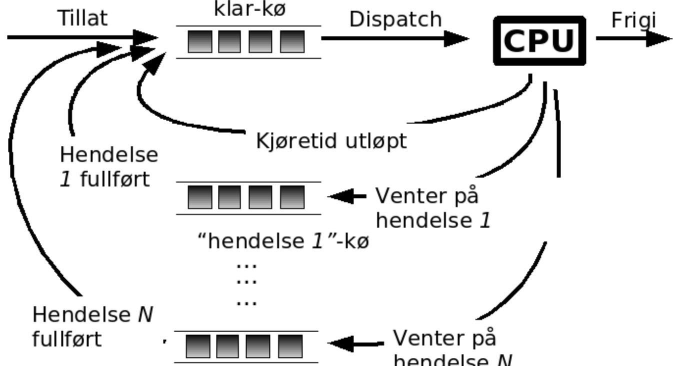


- En kø for alle prosesser som er klare til å kjøre
- En kø for alle prosesser som venter på en hendelse

Fem tilstander – mange køer

figur

-



- En kø for hver hendelse

modell med "suspended" prosesser

- Plass til håndtering av flere prosesser
- "Swapping" - prosessen flyttes helt eller delvis over i sekundærminne
- suspend
- Vi ser på figur "3.9 Process State Transition Diagram with Suspend State"

`http://debbie.hive.no/osyda50/figurer/3.9.png`

prosess-/tråd-modellen i Linux

- Vi ser på figur "4.18 Linux Process/Thread Model"

`http://debbie.hive.no/osyda50/figurer/4.18.png`

-

```
struct task_struct {volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */...
```

- (task_struct)

`http://lxr.linux.no/#linux+v2.6.31/include/linux/sched.h#L1166`

7.5 Kontroll av prosesser

Kjøremodus

For å beskytte OS'et har de fleste to modus

- kernel/system/control mode: Full tilgang
- user mode: Begrenset tilgang
- Modus bestemmes vanligvis av et bit et register i CPU.

Modus settes typisk til kernel mode når:

- Et avbrudd hender
- En prosess utfører et systemkall

Modus settes typisk til user mode når:

- OS-rutine er fullført og kjøring av bruker-prosess fortsetter

Oppstart av prosesser

- Prosessen får tildelt unik ID og oppføring i prosesstabellen
- Tildeling av minne (til alle deler av "process image")
- Initialisering av PCB
- Lenke prosessen inn i riktige strukturer
- evt. annet ...

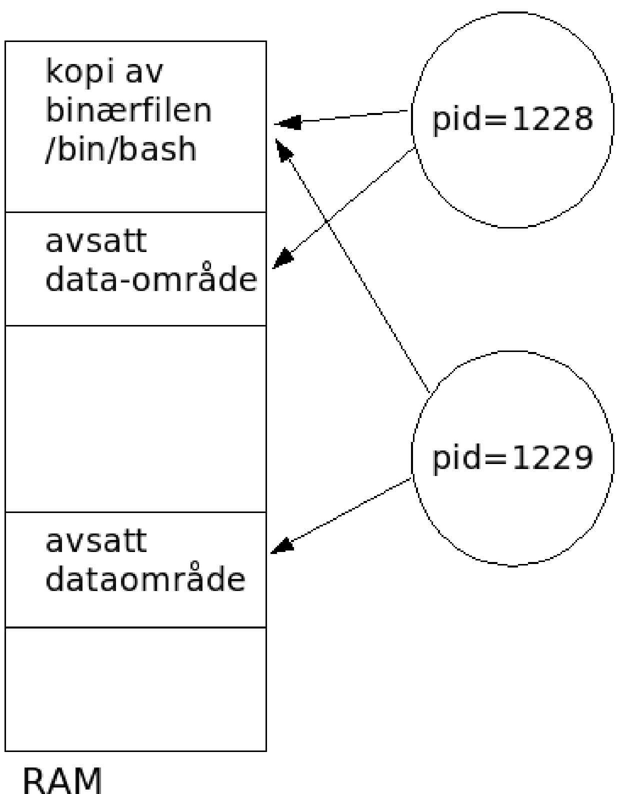
Eksempel: Oppstart av program i UNIX

Fase 1: fork(2)

- Systemkallet fork() oppretter en ny prosess med nye minne-områder for data avsatt.

  figur

-



- Innholdet i minne-områdene for data blir kopiert (egentlig copy on write).
- Fildeskriptor-tabellen blir kopiert – d.v.s. at barne-prosessen arver de åpne filene.

fork(2)

-

```
#include <sys/types.h>
#include <unistd.h>

pid_t fork(void);
```

Eks.:

- eksempler/07/fork-eksempel_I.c

Listing 7.5: eksempler/07/fork-eksempel_I.c

```
1 #include <sys/types.h>
2 #include <unistd.h>
3 #include <stdio.h>
4
```

```
5  /*
6     Demonstrasjon av systemkallet fork()
7
8  */
9
10 int main(void)
11 {
12    fork();
13    printf("Jeg er en prosess nr. %d\n", getpid());
14    return 0;
15 }
```

- eksempler/07/fork-eksempel_II.c

Listing 7.6: eksempler/07/fork-eksempel_II.c

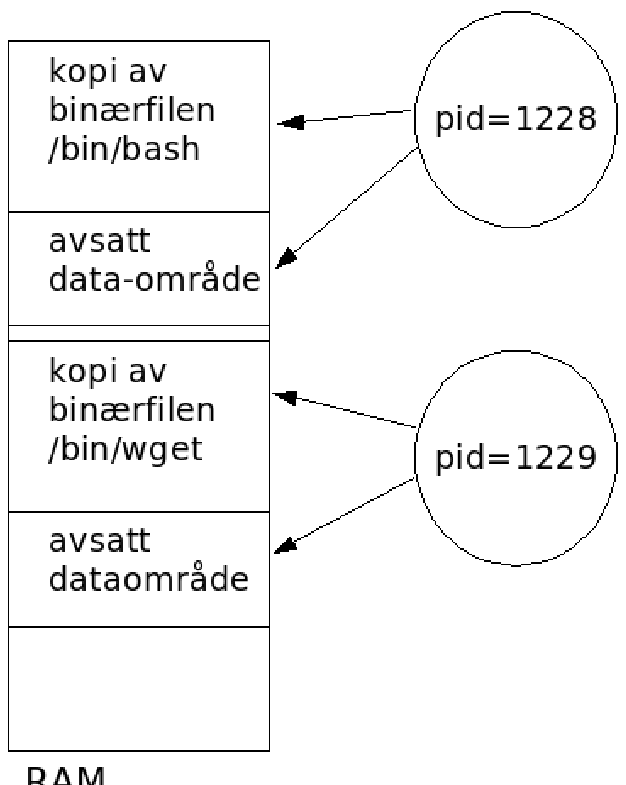
```
1  #include <unistd.h>
2  #include <stdio.h>
3
4  /*
5     Viser hvordan vi kan skille mellom foreldre
6     og barn etter en fork.
7
8  */
9
10 int main(void)
11 {
12    int forkretur;
13    forkretur = fork();
14
15    if (forkretur == 0)
16      printf("\nBarnet sier:\t");
17    else
18      printf("\nForeldren til %d, sier:\t", forkretur);
19
20    printf("Jeg er en prosess nr. %d\n", getpid());
21    return 0;
22 }
```

Fase 2: exec(3)

- exec() endrer innholdeti prosessens minne-område.
- Et nytt program blir lastet.Data-området blir initiert,bortsett fra miljøet sombeholdes som det er.

figur

-



•

```
#include <unistd.h>

extern char **environ;

extern char **environ;
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const  char  *path,  const  char  *arg, ..., char *const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
```

Eksempler

- eksempler/07/exec-eksempel_I.c

Listing 7.7: eksempler/07/exec-eksempel_I.c

```
1 #include <unistd.h>
2
3 /*
4   Demonstrasjon av systemkallet execl()
```

```
5
6 */
7
8 int main(void)
9 {
10
11   execl("/bin/ps","ps","-l", NULL);
12   return 0;
13
14 }
```

- eksempler/07/exec-eksempel_II.c

Listing 7.8: eksempler/07/exec-eksempel_II.c

```
1 #include <unistd.h>
2 #include <stdio.h>
3
4 /*
5   Demonstrasjon av systemkallet execl()
6   i kombinasjon med fork.
7
8 */
9
10 int main(void)
11 {
12
13   int forkretur;
14
15   forkretur = fork();
16
17   if (forkretur == 0)
18     execl("/bin/ps","ps","-l", NULL);
19
20   printf("Jeg er foreldren (%d).\n", getpid());
21   return 0;
22
23 }
```

Bytte av prosess (Process switch)

Når kan kjørende prosess byttes ut?

Når OS får kontroll, kan det bytte prosess

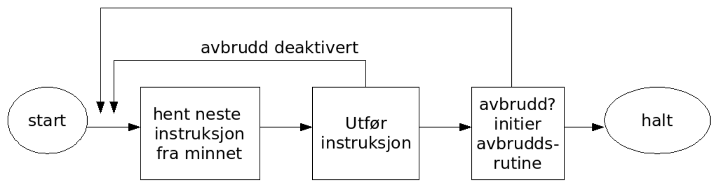
Avbrudd (ekstern hendelse)

- tidsavbrudd
- I/O-avbrudd
- “page fault”
- “trap” - feil/unntak
- systemkall (intern hendelse)

Bytte av modus (Mode switch)

Husk figur fra kap. 1 – fetch execute m/avbrudd

-



Hvis en avbrudd er skjedd

- Programtelleren settes til starten av avbruddsrutinen.
- Modus settes til kernel mode.
- Informasjonen som trengs for å gjenoppta kjøringen av prosessen, prosessens kontekst, lagres i PCB (i delen kalt prosessortilstand)

Bytte av tilstand

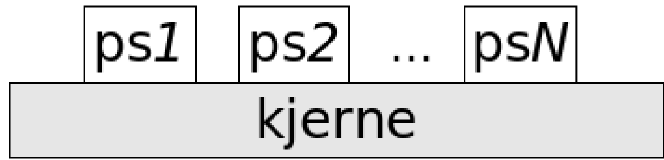
Når en prosess forlater tilstanden "kjørende":

- Prosessor-kontekst lagres
- Oppdater PCB
- Flytt PCB til passende kø
- Velg ny prosess for kjøring
- Oppdater PCB til valgt prosess
- Oppdater strukturer for minnehåndtering
- Gjennopprett prosessortilstand

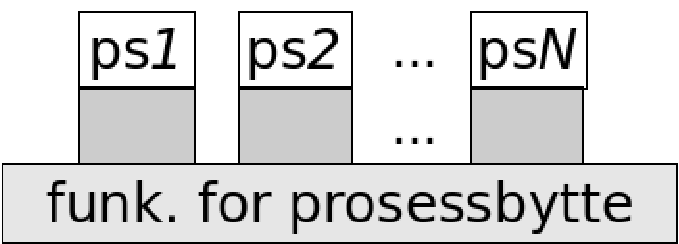
7.6 Alternativer for operativsystemets kjøring

Separat kjerne

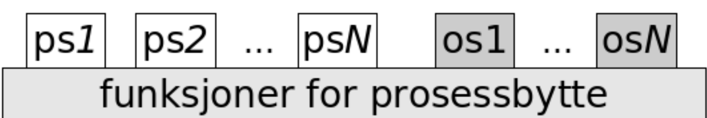
-



OS-funksjoner kjører i brukerprosessene

- 

OS-funksjoner kjører som separate prosesser

- 

7.7 Oppgaver

7.1

- Hva tror du blir utskriften av dette programmet?
-

```
int main() {
fork();
fork();
write(1,"A",1);
return 0;
}
```

- Kompiler og kjør programmet. Ble utskriften som forventet?

7.2

- a) Ta utgangspunkt i eksemplet open_eksempel.c fra forrige tema. Endre det slik at det ”forker” etter at filen er åpnet, men før kallet til read().
- b) Kjør programmet og inspiser fildeskriptor-tabellen til både foreldreprosessen og barnet mens de har filen åpen. Er det noen forskjell i prosessenes fildeskriptor-tabeller?
- c) Kan du finne hvilken tilstand prosessenene fra b) er i. Bruk ps(1) eller top(1).

7.3

- Lag et C-program som går i en løkke og bruker fork(2) og exec(3) for å kjøre følgende kommandoer:

-

```
cut -f1 -d: /etc/passwd
cut -f2 -d: /etc/passwd
...
cut -f7 -d: /etc/passwd
```

7.4

- I filsystemet /proc finnes en katalog for hver prosess på systemet. Inne i hver av disse katalogene, ligger finner du informasjon om prosessene.Dette oppgaven går ut på at du skal lage et shell-script som tar en prosess-identifikator (PID) som argument og skriver ut hvilken tilstand prosessen med oppgitt PID befinner seg i. Skriptet skal slå opp denne informasjonen i filsystemet /proc.Til denne oppgaven trenger du å bruke en editor, kommadoen cut(1) eller grep(1) og filsystemet proc(5).

7.5 - obligatorisk

- Lag et c-program som gjentatte ganger leser inn en kommando fra "standard-input" og kjører denne kommandoen i en egen prosess. Du trenger å bruke systemkallenene fork(2) og exec(3) for å få til dette.

Kapittel 8

08-Tråder og SMP

8.1 Løsningsforslag

7.3

Oppgave

- Lag et C-program som går i en løkke og bruker fork(2) og exec(3) for å kjøre følgende kommandoer:
-

```
cut -f1 -d: /etc/passwd
cut -f2 -d: /etc/passwd
...
cut -f7 -d: /etc/passwd
```

Løsning

- losninger/07/7.3.c

Listing 8.1: losninger/07/7.3.c

```c
#include <unistd.h>

int main (){

  char s[]="-f1";
  int i;

  for (i=1; i<=7; i++){

    if ( fork() == 0 )
      execl("/usr/bin/cut", "cut", s, "-d:", "/etc/passwd", NULL );

    s[2]++;

  }

  return 0;

}
```

7.4

Oppgave

- I filsystemet /proc finnes en katalog for hver prosess på systemet. Inne i hver av disse katalogene, ligger finner du informasjon om prosessene.Dette oppgaven går ut på at du skal lage et shell-script som tar en prosess-identifikator (PID) som argument og skriver ut hvilken tilstand prosessen med oppgitt PID befinner seg i. Skriptet skal slå opp denne informasjonen i filsystemet /proc.Til denne oppgaven trenger du å bruke en editor, kommadoen cut(1) eller grep(1) og filsystemet proc(5).

Løsning

- losninger/07/7.4.sh

Listing 8.2: losninger/07/7.4.sh

```
1 #!/bin/bash
2
3 grep State /proc/$1/status
4 #cut -f3 -d" " /proc/$1/stat
```

8.2 Tråder

Vi repeterer: To definisjoner av prosess:

En enhet av aktivitet karakterisert av

- en en enkel sekvensiell tråd av utførelse
- en tilstand
- tilhørende system resursser
- «an address space with one or more threads executing within that address space, and the required system resources for those threads.»

Om prosesser og tråder

Begrepet 'prosess' kan deles i to uavhengige deler

- eierskap til resursser
- skedulering/kjøring

Vi beholder prossess-begrepet og knytter det til den første delen.

"prosess/task"

- enhet for eierskap (til resursser)
- enhet for beskyttelse (beskyttelse-funksjon i OS)

Til den andre delen knytter vi begrepet 'tråd'.

"tråd/lettvekts-prosess"

- enhet for kjørings/skedulering
- kjøring følger en kjørings-sti/"execution path/"trace"
- Vi ser på figur 1.26 "Nested Procedures"

`http://debbie.hive.no/osyda50/figurer/figur_1.26.png`

- kjørings-sti kan være sammenflettet med andre prosesskjøringer

Ulike varianter

- Vi ser på figur 4.1 "Threads and Processes"

`http://debbie.hive.no/osyda50/figurer/figur_4.1.png`

Prosess-håndtering

- Vi ser på figur 4.2 "Single Threaded and Multithreaded Process Models"

`http://debbie.hive.no/osyda50/figurer/figur_4.2.png`

- eksempler/08/lokal_vs_global_prosess.c

Listing 8.3: eksempler/08/lokal_vs_global_prosess.c

```
1  #include <unistd.h>
2  #include <stdio.h>
3
4  void *funksjon();
5  int global;
6
7  int main() {
8
9    fork();
10   funksjon();
11
12   return 0;
13
14 }
15
16 void *funksjon(){
17
18   int lokal = 0;
19
20   printf("%d:Global: %d\tLokal: %d. Skriv nye:\n", getpid(), global
       , lokal );
21   scanf("%d %d", &global, &lokal);
22   printf("%d:Nye verdier:\tGlobal: %d\tLokal: %d\n", getpid(),
       global, lokal );
23
24 }
```

- eksempler/08/lokal_vs_global_traad.c

Listing 8.4: eksempler/08/lokal_vs_global_traad.c

```
1  #include <unistd.h>
2  #include <stdio.h>
3  #include <pthread.h>
4
5  /*
6    Husk aa ta med argumentene "-lpthread" og
7    "-D_REENTRANT" ved kompilering.
8  */
9
10 void *funksjon();
11 int global;
12
13 int main() {
14
15   pthread_t traad;
16
17   pthread_create(&traad, NULL, funksjon, NULL );
18   pthread_join(traad, NULL);
19
20   pthread_create(&traad, NULL, funksjon, NULL );
21   pthread_join(traad, NULL);
22
23   return 0;
24
25 }
26
27 void *funksjon(){
28
29   int lokal = 0;
30
31   printf("%d:Global: %d\tLokal: %d. Skriv nye:\n", getpid(), global
        , lokal );
32   scanf("%d %d", &global, &lokal);
33   printf("%d:Nye verdier:\tGlobal: %d\tLokal: %d\n", getpid(),
        global, lokal );
34
35 }
```

Hvorfor tråder?

- I noen tilfeller: Raskere programutførelse
- Oppleves raskere – ting gjøres i «bakgrunnen» - bruker opplever ikke venting.
- Asynkrone hendelser (eks. sikkerhetkopiering).
- Lettere programutvikling – modularisering
- (De tre første punktene vil også gjelde prosesser som kjøres parallelt/konkurrerende)
- Vi ser på figur_4.3

`http://debbie.hive.no/osyda50/figurer/figur_4.3.png`

Fordeler med tråder fremfor prosesser

- raskere å opprette en ny tråd i en eksisterende prosess, enn å opprette en ny prosess.
- raskere å terminere en tråd enn en prosess
- raskere å veksle mellom tråder enn prosesser
- raskere kommunikasjon mellom tråder (sparer to modus-bytte) - trenger ikke involvere kjernen

Handlinger som involverer alle tråder i en prosess

- suspensjon - veksle hele prosessens adresserom over på sekundærminne
- terminering av prosess

Trådenes tilstand

- Kjørende
- Klar
- Blokkert
- IKKE suspendert – kun hele prosesser suspenderes

Operasjoner knyttet til endring av tråders tilstand

- ”spawn” - opprette
- blokkere - Dersom hele prosessen blokkeres mens en tråd venter på en hendelse? ... da forsvinner noe av vitsen med tråder
- avblokkere
- avslutt
- Vi ser på figur 4.4 ”Multithreading Example on a Uniprocessor”

`http://debbie.hive.no/osyda50/figurer/figur_4.4.png`

Tråder på ”user level” eller ”kernel level”

User Level Threads (heretter også kalt brukertråder)

- Vi ser på figur ”4.6 (a)” fra pensumboka.

`http://debbie.hive.no/osyda50/figurer/figur_4.6.png`

- brukerapplikasjonene håndterer trådene
- kjernen ikke involvert
- tråd-bibliotek (”threads library”)

ulemper med brukertråder

- Ved blokkerende systemkall blokkeres hele prosessen - og dermed alle trådene
- Kan ikke utnytte flerprosesskjøring ("multiprosessing")

Brukertrådenes ulemper kan omgås

implementere applikasjonen v.h.a. flere prosesser

- løser begge problemene
- mister fordelene med tråder

"jacketing"

- løser blokkeringsproblemet
- gjør om til ikke-blokkerende
- egen tråd som kontrollerer om ressurs er ledig før systemkall utføres

fordeler med brukertråder

- ved veksling sparer to bytte av modus.
- muligheter for applikasjonspesifikk skedulering.
- uavhengig av OS-funksjonaltitet - kan brukes på alle OS.

Kernel Level Threads (heretter også kalt kjernetråder)

- Vi ser på figur "4.6 (b)" fra pensumboka.

`http://debbie.hive.no/osyda50/figurer/figur_4.6.png`

- kjernen håndterer trådene
- API mot trådfunksjonalitet i kjernen
- Eks. Windows

Fordeler med kjernetråder

trådene er skedulerings-enhet

- ved blokkerende systemkall blokkeres kun tråden
- har dermed ingen av "user level"-trådens ulemper
- kjerne-rutiner kan også være flertrådkjørende

Ulemper ved bytte av kjernetråder

- tråd-veksling medfører modus-bytte
- Vi ser på tabell 4.1 "Thread and Process Latencies" (mikro-sekunder).

`http://debbie.hive.no/osyda50/figurer/04-tabeller-0.png`

Kombinert

- Vi ser på figur "4.6 (c)" fra pensumboka.

`http://debbie.hive.no/osyda50/figurer/figur_4.6.png`

- Opprettelse av tråder skjer i "user space"
- "Bruker-trådene" i en prosess kan knyttes til like mange (el. færre) "kjerne-tråder"
- Hovedtyngden av trådenes skedulering og synkronisering gjøres i "user space"
- Kan unngå ulempene fra begge de to variantene
- Eks. Solaris

Tråder i Linux

- eldre versjoner - kun "brukertråder" (typisk pthreads)
- nyere versjoner: Lignende Solaris
- brukertråder knyttes til kjernetråder
- flere brukertråder som utgjør en brukerprosess kobles til flere kjerneprosesser
- deler samme gruppe ID
- unngår kontekst switch
- clone(2) brukes i implementasjon av tråder

Eksempel på inspeksjon av tråder

- eksempler/08/traadtest.c

Listing 8.5: eksempler/08/traadtest.c

```
1  #include <stdio.h>
2  #include <pthread.h>
3
4  /*
5
6    Husk aa ta med argumentene "-lpthread" og
7    "-D_REENTRANT" ved kompilering.
```

```
8
9  */
10
11 void *traadtest(){
12
13     printf("Trykk enter.");
14   getchar();
15   return NULL;
16 }
17
18 int main(){
19
20   pthread_t traad;
21   pthread_create(&traad, NULL, traadtest, NULL);
22   pthread_join(traad, NULL);
23   return 0;
24
25 }
```

- pstree
- ps -L

8.3 SMP (09.1)

Parallellisering

- Hva menes med parallellisering?

Hvorfor parallellisere?

Bedre ytelse

Demonstrasjon av bedre ytelse

- eksempler/09/seriell_vs_parallell.sh

Listing 8.6: eksempler/09/seriell_vs_parallell.sh

```
1  #!/bin/sh
2
3  # Skript som kompilerer og maaler kjoeretiden paa eksemplene:
4  # parallell.c og seriell.c
5
6  # 1. Kjoer testen paa maskin med en prosessor
7  # 2. Kjoer testen paa maskin med flere prosessorer.
8  # 3. Sammenligne tiden
9
10 # av Thomas Nordli <tn@hive.no>
11
12
13 if [ ! -e parallell ]; then
14     gcc parallell.c -o ./parallell
15 fi
16
17 if [ ! -e seriell ]; then
18     gcc seriell.c  -o ./seriell
19 fi
```

```
20
21 for PRG in ./parallell ./seriell; do
22     /usr/bin/time -f "%C: %E" $PRG
23 done
```

- eksempler/09/parallell.c

Listing 8.7: eksempler/09/parallell.c

```
1  #include <sys/types.h>
2  #include <sys/wait.h>
3  #include <unistd.h>
4  #include <stdio.h>
5  #include <time.h>
6
7  #define N 10000
8  #define M 8
9
10 int main(){
11
12   int i,j,k;
13   double x;
14
15   // Starter M antall CPU-intensive prosesser
16   for (i=0; i<M; i++){
17
18     if (!fork()){
19
20       // Gjoer mange unyttige beregninger
21       for (j=0; j<N; j++)
22   for (k=0; k<N; k++)
23     x=x*j;
24       return 0;
25     }
26   }
27
28   for (i=0; i<M; i++)
29     wait(NULL);
30
31   return 0;
32
33 }
```

- eksempler/09/seriell.c

Listing 8.8: eksempler/09/seriell.c

```
1  #include <sys/types.h>
2  #include <sys/wait.h>
3  #include <unistd.h>
4  #include <stdio.h>
5  #include <time.h>
6
7  #define N 10000
8  #define M 8
9
10 int main(){
11
12   int i,j,k;
13   double x;
```

```
14
15    // Starter M antall CPU-intensive prosesser
16    for (i=0; i<M; i++){
17
18      if (!fork()){
19
20        // Gjoer mange unyttige beregninger
21        for (j=0; j<N; j++)
22    for (k=0; k<N; k++)
23      x=x*j;
24        return 0;
25      }
26      else
27        wait(NULL);
28    }
29
30    return 0;
31 }
```

(bedre pålitelighet)

- Hvordan kan parallellisering bidra til bedre pålitelighet?

Alternative for parallell-prosessering

Flynn's taksonomi

- SISD - Single Instruction - Single Datum
- SIMD - Single Instruction - Multiple Data
- MISD - Multiple Instructions - Single Datum
- MIMD - Multiple Instrucions -Multiple Data
- https://computing.llnl.gov/tutorials/parallel_comp/#Flynn

`https://computing.llnl.gov/tutorials/parallel_comp/#Flynn`

- Vi ser på figur "4.8 Parallell Processor Architechtures" i pensumboka.

`http://debbie.hive.no/osyda50/figurer/4.8.png`

SMP (Symmetric Multiprocessing) - symmetrisk flerprosesskjøring

Hver prosessor har

- kontroll-enhet, aritmetisk-logisk enhet (ALU), registre
- tilgang til
- felles primærminne via delt buss
- felles I/O-enheter via delt buss

- mulighet til å kommunisere med andre prosessorer v.h.a delt minne (og direkte signaler)
- vanligvis minst ett nivå med lokalt mellomlager ("cache")
- cache coherence problem: Hva hvis data i lokalt mellomlager endres?
- figur "4.9 i pensumboka"

`http://debbie.hive.no/osyda50/figurer/4.9.png`

Arkitektur

- kjernen kan kjøres på enhver prosessor
- prosessorene selv-skedulerer
- brukere/programmerere skal ikke behøve å ta hensyn til hvor mange prosessorer det er i systemet

Konsekvenser for OS

- skedulering - to prosessorer må ikke velge å kjøre samme prosess
- kjerne-rutiner må kunne kjøres på flere prosessorer samtidig (må være reentrant)
- synkronisering av tilgang til resursser (f.eks. ved bruk av låser)

minnehåndtering

- "multiported memories"
- prosessorenes mekanismer for sideveksling må synkroniseres
- Hvor mange CPU'er er det på debbie.hive.no?

Eksempler

- eksempler/09/min_cpu.c

Listing 8.9: eksempler/09/min_cpu.c

```
1  #include <utmpx.h>
2  #include <stdio.h>
3
4  #define N 10
5
6  int main(){
7
8    int i;
9
10   for (i=0; i<N; i++)
11
12     if (!fork()){
13
```

```
14        printf("Pid:%d\tCpu:%d\n", getpid(), sched_getcpu() );
15        return 0;
16      }
17
18    for (i=0; i<N; i++)
19      wait();
20
21    return 0;
22
23  }
```

8.4 Oppgaver

8.1

- a) Endre programmet eksemplet openeksempel.c slik at filen åpnes i en egen tråd.
- b) Inspiser prosessens fildeskriptortabell, mens programmet kjører.

8.2

- Nettleseren Google Chrome er implementert slik at hver «tab» kjører som en egen prosess. Finn ut hvorfor dette valget er gjort.
- Tips: Les tegneserien.

`http://www.google.com/googlebooks/chrome/`

8.3

a)

- Lag et program som starter en ny prosess og venter til barnet har terminert, før det selv terminerer. Bruk fork() for å starte ny prosess og wait() for å få foreldreprosessen til å vente på barnet. Barneprosessen skal kun terminere så fort som mulig.

b)

- Utvid programmet i a) slik at det starter og venter 10000 ganger.

c)

- Ta en kopi av programmet i b) og endre kopien, slik at den starter og venter på en tråd i stedet for en prosess. Bruk pthread_create() for å starte tråden og pthread_join() for å vente på at tråden er ferdig-kjørt. Pass på at også tråden returnerer så fort som mulig.

d)

- Med en null-fork menes å opprette, skedulere, utføre og fullføre en prosess/tråd som starter en null-prosedyre. Programmene i b) og d) kan brukes til å lage et gjennomsnitt av hvor lang tid en null-fork av en prosess og en tråd tar.Mål og beregn gjennomsnittelig tid for nullfork av både tråder og prosesser. Bruk clock() og konstanten CLOCKS_PER_SEC.

Eksempel på tidtaking:

-

```
clock_t start, stopp;
double sekunder;
start = clock();
...
stopp = clock();
'sekunder= (double) (stopp-start)/CLOCKS_PER_SEC;
```

9.1

- Logg inn på student.hive.no eller en annen maskin dere har tilgang til og finn ut hvor mange prosessorer er tilgjengelige for prosesser/tråder på student.hive.no. Bruk f.eks. 'top' eller pseudofilstystemet '/proc'. Er det flere eller færre enn på debbie.hive.no?

9.2

- Finn feilen i skriptet under:
- oppgaver/09/finn_feilen_1.sh

Listing 8.10: oppgaver/09/finn_feilen_1.sh

```
1  #!/bin/sh
2
3  # Dette skriptet skulle gjort foelgende:
4  #
5  # - kompilert filen ~tn/osyda50/eksempler/04/openeksempel.c
6  # - kjoert det kjoerbare programmet (./openeks)
7  # - slettet det kjoerbare programmet (./openeks)
8  #
9  # men innheholder EN feil.
10 #
11 # Finn, forklar og rett opp feilen.
12 #
13 # Hilsen Thomas <tn@hive.no>
14
15 gcc -o ./openeks ~tn/osyda50/eksempler/04/openeksempel.c &
16 ./openeks
17 rm ./openeks
```

9.3

- a) Skriv om parallell.c og seriell.c, slil at de bruker tråder i stedet for prosesser.
- b) Skriv om skriptet 'eksempler/09/seriell_vs_parallell.sh', som er gjennomgått over, slik at det måler kjøretiden på programmene du skrev i a).
- c) Kjør seriell_vs_parallell.sh på en maskin med flere prosessorer (f.eks. debbie.hive.no). Legg merke til eventuelle forskjeller i kjøretid. Hva betyr de?
- d) Hvis du har tilgang til en enkeltprosessor-maskin, så forsøk å kjøre testen på denne. Ble det noen forskjeller fra målingene i c)? Kan du forklare dem?

Kapittel 9

10 Samtidig behandling

9.1 Løsningsforslag

8.3

a)

- Lag et program som starter en ny prosess og venter til barnet har terminert, før det selv terminerer. Bruk fork() for å starte ny prosess og wait() for å få foreldreprosessen til å vente på barnet. Barneprosessen skal kun terminere så fort som mulig.

b)

- Utvid programmet i a) slik at det starter og venter 10000 ganger.

c)

- Ta en kopi av programmet i b) og endre kopien, slik at den starter og venter på en tråd i stedet for en prosess. Bruk pthread_create() for å starte tråden og pthread_join() for å vente på at tråden er ferdig-kjørt. Pass på at også tråden returnerer så fort som mulig.

d)

- Med en null-fork menes å opprette, skedulere, utføre og fullføre en prosess/tråd som starter en null-prosedyre. Programmene i b) og d) kan brukes til å lage et gjennomsnitt av hvor lang tid en null-fork av en prosess og en tråd tar.Mål og beregn gjennomsnittelig tid for nullfork av både tråder og prosesser. Bruk clock() og konstanten CLOCKS_PER_SEC.

Løsning

- losninger/08/8.3.null_fork.c

Listing 9.1: losninger/08/8.3.null_fork.c

```
1 #include <unistd.h>
```

```
2  #include <sys/types.h>
3  #include <sys/wait.h>
4  #include <stdio.h>
5  #include <time.h>
6
7  #define N 100000
8
9  int main(){
10
11   clock_t start, stopp;
12   int cpid;
13   int i=0;
14
15   start = clock();
16   while (i<N){
17
18     cpid=fork();
19
20     if ( cpid < 0 )
21       continue;
22
23     if (cpid){
24       wait(NULL);
25       i++;
26     }
27
28     else
29       return 1;
30
31   }
32   stopp = clock();
33   printf("Det tok %f sekunder i snitt.\n", ((double) (stopp-start))
         /CLOCKS_PER_SEC/N );
34
35   return 0;
36
37 }
```

- losninger/08/8.3.null_traad.c

Listing 9.2: losninger/08/8.3.null_traad.c

```
1  #include <stdio.h>
2  #include <time.h>
3  #include <pthread.h>
4
5  #define N 100000
6
7  /*
8
9    Husk aa ta med argumentene "-lpthread" og
10   "-D_REENTRANT" ved kompilering.
11
12 */
13
14 void *nulltraad(){
15   return NULL;
16 }
17
18 int main(){
19
```

```
20   clock_t start, stopp;
21   pthread_t traad;
22   int i;
23
24   start = clock();
25   for (i=0;i<N;i++){
26
27     pthread_create(&traad, NULL, nulltraad, NULL);
28     pthread_join(traad, NULL);
29
30   }
31   stopp = clock();
32
33   printf("Det tok %f sekunder i snitt.\n", ((double) (stopp-start))
        /CLOCKS_PER_SEC/N );
34
35   return 0;
36
37 }
```

9.2 Del 1 (09.2)

Om ”concurrency”

Hva er ”concurrency”

- Flere tråder/prosesser kjører konkurerende.
- Sammeflettede (interleaving) eller overlappende prosesser/tråder har det samme problemet:

Relativ utføringshastighet er uforutsigbar. Den er bestemt av

- andre prosessers kjøring
- skedulerings-rutiner
- avbrudds-rutiner

Hvor finner vi ”concurrency”

systemer

- multiprogramming
- multiprocessing
- distributed processing
- Hva er menes med disse?

kontekster

- flere applikasjoner
- innenfor samme applikasjon
- i selve operativsystemkjernen

Kappløp (race condition)

- Vi har et kappløp når flere prosesser/tråder leser og skriver data, på en måte som gjøre at resultatet er avhengig av rekkefølgen instruksjonene utføres på.

eks: transaksjoner mellom konti:

Uten tråder:

- eksempler/09/balanse_0.c

Listing 9.3: eksempler/09/balanse_0.c

```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <unistd.h>
4  #include <stdlib.h>
5  #include <time.h>
6
7  #define T 1000   // Totalsummen i spillet
8  #define N 5      // Antall konti
9
10 /*
11
12   Programmet simulerer flytting av penger mellom N antall bankkonti
        .
13   For hver 10.millioner transaksjoner, gjoeres en saldoutskrift av
        alle kontiene.
14
15   Her er det ingen samtidig kjoering som skaper problemer.
16
17   Thomas Nordli <tn@hive.no>
18
19 */
20
21 void opprett_konti();
22 void skriv_saldo();
23
24 int konto[N];
25
26 int main(){
27
28   int til, fra, belop;
29   unsigned int i;
30
31   srand( time(NULL)); // initierer slumtallgenerator
32   opprett_konti();
33   skriv_saldo();
34
35   while(1){
36
37     if ( 0 == i%10000000 )
38       skriv_saldo();
39     i++;
40     fra=rand()%N;
41     til=rand()%N;
42     belop=rand() % ( (konto[fra]/10) ); // aldri mer enn 10% av
          saldo
43
44     konto[fra]-=belop;
```

```
45        konto[til]+=belop;
46
47    }
48
49    return 0;
50
51 }
52
53
54 void skriv_saldo(){
55
56    int sum,i;
57
58    for (i=0, sum=0; i<N; i++){
59
60       printf("konto %d:\t%d\n", i, konto[i]);
61       sum+=konto[i];
62
63    }
64
65    printf("----------------\nTotalt:\t%d\n================\n",sum);
66
67 }
68
69 void opprett_konti(){
70
71    int i;
72
73    for (i=0;i<N;i++)
74       konto[i]=T/N;
75
76 }
```

Ikke-samtidige tråder:

- eksempler/09/balanse_1.c

Listing 9.4: eksempler/09/balanse_1.c

```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <unistd.h>
4  #include <stdlib.h>
5  #include <time.h>
6
7  /*
8
9     Programmet simulerer flytting av penger mellom N antall bankkonti
         .
10    Flytting av penger gjoeres i M antall traader,
11    som kjoeres sekvensielt.
12    Hver av de traadene avslutter etter ca. ett sekund.
13
14    Hovedtraaden venter til alle traadene er terminert,
15    og skriver saa saldoene til slutt.
16    Her er det ingen problemer foraarsaket av samtidig kjoering,
17    da traaden kjoeres etter hverandre - den andre starter ikke
18    foer den foerste er ferdig etc.
19
20    Husk aa ta med argumentene "-lpthread" og
21    "-D_REENTRANT" ved kompilering.
```

```
22
23   Thomas Nordli <tn@hive.no>
24
25 */
26
27 #define T 1000   // Totalsummen i spillet
28 #define N 5      // Antall konti
29 #define M 3      // Antall traader
30
31 void opprett_konti();
32 void *flytt_penger();
33 void skriv_saldo();
34
35
36 int konto[N];
37
38 int main(){
39
40   pthread_t spekulant;
41   int i;
42
43   srand( time(NULL)); // Initierer slumtallgenerator
44   opprett_konti();
45
46   for (i=0; i<M; i++){
47     pthread_create(&spekulant, NULL, flytt_penger, NULL);
48     pthread_join(spekulant, NULL);
49
50   }
51   skriv_saldo();
52
53
54   return 0;
55
56 }
57
58
59 void skriv_saldo(){
60
61   int sum,i;
62
63   for (i=0, sum=0; i<N; i++){
64
65     printf("konto %d:\t%d\n", i, konto[i]);
66     sum+=konto[i];
67
68   }
69
70   printf("---------------\nTotalt:\t%d\n===============\n",sum);
71
72 }
73
74 void opprett_konti(){
75
76   int i;
77
78   for (i=0;i<N;i++)
79     konto[i]=T/N;
80
81 }
82
83 void *flytt_penger(){
```

```
84
85     int til, fra, belop;
86     clock_t start;
87
88     start = clock()/CLOCKS_PER_SEC;
89
90     while(1){
91
92       if ((clock()/CLOCKS_PER_SEC)-start > 1 ) // ikke kjoer mer enn
           1s
93         break;
94
95       fra=rand()%N; // tilfeldig konto
96       til=rand()%N; // tilfeldig konto
97       belop=rand() % ( (konto[fra]/10) );  // aldri mer enn 10% av
           saldo
98
99       konto[fra]-=belop;
100      konto[til]+=belop;
101
102    }
103
104    return NULL;
105
106  }
```

Samtidige tråder:

- eksempler/09/balanse_2.c

Listing 9.5: eksempler/09/balanse_2.c

```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <unistd.h>
4  #include <stdlib.h>
5  #include <time.h>
6
7  /*
8
9    Programmet simulerer flytting av penger mellom N antall bankkonti
       .
10   Flytting av penger gjoeres i M antall traader,
11   som kjoeres samtidig (concurrent).
12
13   Hovedtraaden skriver saldoer hvert sekund.
14
15   Her er det problemer foraarsaket av samtidig kjoering.
16   Vi har et kapploep (race condition).
17
18   Husk aa ta med argumentene "-lpthread" og
19   "-D_REENTRANT" ved kompilering.
20
21   Thomas Nordli <tn@hive.no>
22
23 */
24
25 #define T 1000  // Totalsummen i spillet
26 #define N 5     // Antall konti
27 #define S 10    // Antall spekulanter som flytter penger
28
```

```
29  void opprett_spekulanter();
30  void opprett_konti();
31  void *flytt_penger();
32  void skriv_saldo();
33
34  pthread_t spekulant[S];
35  int konto[N];
36
37  int main(){
38
39    srand( time(NULL)); // Initierer slumtallgenerator
40    opprett_konti();
41    opprett_spekulanter();
42
43    while(1){
44
45      sleep(1);
46      skriv_saldo();
47
48    }
49
50    return 0;
51
52  }
53
54
55  void skriv_saldo(){
56
57    int sum,i;
58
59    for (i=0, sum=0; i<N; i++){
60
61      printf("konto %d:\t%d\n", i, konto[i]);
62      sum+=konto[i];
63
64    }
65
66    printf("--------------\nTotalt:\t%d\n==============\n",sum);
67
68  }
69
70  void opprett_konti(){
71
72    int i;
73
74    for (i=0;i<N;i++)
75      konto[i]=T/N;
76    skriv_saldo();
77
78  }
79
80  void *flytt_penger(){
81
82    int til, fra, belop;
83
84    while(1){
85
86      fra=rand()%N;
87      til=rand()%N;
88      belop=rand() % ( (konto[fra]/10) );
89
90      konto[fra]-=belop;
```

```
91        konto[til]+=belop;
92
93     }
94
95     return NULL;
96
97  }
98
99  void opprett_spekulanter(){
100
101    int i;
102    for (i=0; i<S; i++)
103      pthread_create(&spekulant[i], NULL, flytt_penger, NULL);
104
105 }
```

Gjensidig utelukkelse

Basis krav for "concurrency"'

- muligheten til å håndheve gjensidig utelukkelse

d.v.s:

- å kunne ekskludere alle andre prosesser/tråder fra en bestemt handlemåte mens en prosess/tråd er gitt muligheten til handlemåten

Gjensidig utelukkelse medfører problemer

vranglås (deadlock)

Eks.

- Prosesser: P1 og P2
- Resursser: R1 og R2
- P1 har R1 og venter på R2
- P2 har R2 og venter på R1

utsulting (starvation)

Eks.

- Prosesser: P1, P2 og P3
- Resurss: R1
- P1 og P2 veksler på å bruke R1
- P3 sulter ut

Typer av prosess-interaksjon

Konkuranse

- Prosessene er uavhengige.
- Prosessenes kJøretids-fordeling kan påvirkes.

Mulige kontroll-utfordringer

- gjensidig utelukkelse
- vranglås
- utsulting

Samarbeid ved deling

- Resultat av prosessene kan avhenge hverandres handlinger.
- Prosessenes kJøretids-fordeling kan påvirkes.

Mulige kontroll-utfordringer

- gjensidig utelukkelse
- vranglås
- utsulting
- data-koherens

Samarbeid ved kommunikasjon

- Resultat av prosessene kan avhenge av mottatt informasjon fra andre prosesser.
- Prosessenes kjøretids-fordeling kan påvirkes.

Mulige kontroll-utfordringer

- vranglås
- utsulting
- (Se også tabell_5.2.png fra pensumboka.)

`http://debbie.hive.no/osyda50/figurer/tabell_5.2.png`

Semaforer

- Semaforer er en mekanisme for å begrense tilgangen til resursser.

Tre varianter av semaforer

1. Generell eller tellende semafor (general semaphore / counting semaphore)

Et heltall x med tre operasjoner: init

- Setter verdien av semaforen til et ikke-negativt heltall.
- Flere prosesser/tråder kan gis adgang ved å initialisere semaforen med ønskelig antall – større enn en.

P (semWait)

- Dekrementerer verdien av semaforen.
- Hvis verdien blir mindre enn null. Blokkeres tråden/prosessen og legges i kø.
-

```
x--;if (x<0) {  block();  enqueue();}
```

V (semSignal)

- Inkrementerer verdien av semaforen.
- Hvis verdien blir mindre eller lik null, hentes en tråd/prosess fra køen og "avblokkeres".
-

```
x++;if (x<=0) {  p=dequeue();  unblock(p);}
```

- P og V må være atomiske operasjoner – de skal kjøre uavbrutt

Eksempler på bruk

- Vi ser på eksemplet i figur 5.7 i pensumboka

`http://debbie.hive.no/osyda50/figurer/5.7.png`

- eksempler/09/balanse_3.c

Listing 9.6: eksempler/09/balanse_3.c

```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <unistd.h>
4  #include <stdlib.h>
5  #include <time.h>
6
7
8  /*
9
10   Programmet simulerer flytting av penger mellom N antall bankkonti
     .
```

```
11     Flytting av penger gjoeres i M antall traader,
12     som kjoeres samtidig (concurrent).
13
14     Hovedtraaden skriver saldoer hvert sekund.
15
16     Her er de kritiske omraadene beskyttet av en semafor.
17     Vi har dermed intqet kapploep (race condition).
18
19     Husk aa ta med argumentene "-lpthread" og
20     "-D_REENTRANT" ved kompilering.
21
22     Thomas Nordli <tn@hive.no>
23
24   */
25
26   #define T 1000   // Totalsummen i spillet
27   #define N 5      // Antall konti
28   #define S 10     // Antall spekulanter som flytter penger
29
30   void opprett_spekulanter();
31   void opprett_konti();
32   void *flytt_penger();
33   void skriv_saldo();
34
35   pthread_mutex_t laas;
36   pthread_t spekulant[S];
37   int konto[N];
38
39   int main(){
40
41     pthread_mutex_init(&laas, NULL); // Initierer mutex med "default-
         verdier"
42     srand( time(NULL));               // Initierer slumtallgenerator
43
44     opprett_konti();
45     opprett_spekulanter();
46
47     while(1){
48
49       sleep(1);
50       pthread_mutex_lock(&laas);
51       skriv_saldo();                // Kritisk omraade
52       pthread_mutex_unlock(&laas);
53     }
54
55     return 0;
56
57   }
58
59
60   void skriv_saldo(){
61
62     int sum,i;
63
64     for (i=0, sum=0; i<N; i++){
65
66       printf("konto %d:\t%d\n", i, konto[i]);
67       sum+=konto[i];
68
69     }
70
71     printf("----------------\nTotalt:\t%d\n================\n",sum);
```

```
72
73 }
74
75 void opprett_konti(){
76
77   int i;
78
79   for (i=0;i<N;i++)
80     konto[i]=T/N;
81 }
82
83 void *flytt_penger(){
84
85   int til, fra, belop;
86
87   while(1){
88
89     fra=rand()%N;
90     til=rand()%N;
91     belop=rand() % ( (konto[fra]/10) );
92
93     pthread_mutex_lock(&laas);
94     konto[fra]-=belop;           // kritisk ..
95     konto[til]+=belop;           // omraade
96     pthread_mutex_unlock(&laas);
97   }
98
99   return NULL;
100
101 }
102
103 void opprett_spekulanter(){
104
105   int i;
106   for (i=0; i<S; i++)
107     pthread_create(&spekulant[i], NULL, flytt_penger, NULL);
108
109 }
```

2. Binær semafor

- Kan kun ha verdien verdien 0 eller 1.

3. mutex

- Brukes som synonym til binær semafor

eller med tilleggsbegrensning:

- samme tråd/prosess som "låste" må "låse opp".

Sterk eller svak semafor (strong or weak semaphore)

- Dersom semaforens kø håndteres som en FIFO, kalles den en sterk semafor.
- Dersom semaforen ikke har definert noen køorden, kalles den en svak metafor.
- Dette gjelder både generelle og binære semaforer.

9.3 Del 2 (10)

Produsent-konsument-problemet (semaforer nå igjen...)

Problemformulering

- Data lagres i et lager
- Produsent putter data inn i lageret
- Konsument henter data ut fra lageret

To varianter

- 1. lager med ubegrenset størrelse (urealistisk)
- 2. lager med begrenset størrelse ("Bounded Buffer Problem")

Eksempel med semaforer

- produsent-konsument_1.c

`http://debbie.hive.no/osyda50/eksempler/10/produsent-konsument_1.c`

- produsent-konsument_2.c

`http://debbie.hive.no/osyda50/eksempler/10/produsent-konsument_2.c`

- produsent-konsument_3.c

`http://debbie.hive.no/osyda50/eksempler/10/produsent-konsument_3.c`

Vranglås

Definisjon av vranglås

- Permanent blokkering av et sett prosesser som enten konkurerer om systemresursser eller kommuniserer med hverandre.

Eksempel:

- Vi ser på figur 6.1 på side 263 i Stallings.

`http://debbie.hive.no/osyda50/figurer/6.1.png`

Hvordan kan vranglås oppstå?

Kriterier for vranglås

- 1. Gjensidig utelukkelse
- 2. Hold og vent
- 3. Ingen "preemption"
- 4. Sirkulær venting
- Vranglås kan oppstå dersom de tre første kriteriene er tilstede:
- Dersom en vranglås oppstår, er i tillegg det fjerde kriterium oppfylt.

Resurss-allokerings-graf

- Prosess er sirkel.
- Resurss er firkant
- En prikk inne i firkanten for hver instans av resurssen.
- Pil fra Prosess til resurss: En resurssforespørsel som ikke er innvilget ennå
- Pil fra resurss-instans (prikk) til prosess: Prosessen holder instansen
- Sykler i grafen betyr at vi har sirkulær venting.
- vi ser eksempler på resurss-allokerings-grafer(figur 6.5)

`http://debbie.hive.no/osyda50/figurer/6.5.png`

- Hvordan ser resurss-allokerings-grafen ut for figurene 6.1 a) og 6.1 b) ?

Håndtering av vranglås

Forebygging (prevention)

Indirekte - fjerne et av de tre første punkt i kriteriene Gjensidig utelukkelse

- Dette må vi ha

Hold og vent:

- Prosess venter (blokkert) til den får tilgang til alle resursene den kommer til å trenge på en gang.
- Unødvendig mye venting.
- Ofte vanskelig å vite på forhånd hvilke resusrsser en prosess kommer til å trenge.

"Preemption"

- Tvinge prossesser til å gi slipp på resurss.
- Dette kan gjøres med resursser hvis tilstand lett kan lagres og gjenopprettes (som f.eks. en prosessor)

Direkte - hindre det fjerde punktet

- hindre det fjerde punktet, sirkulær venting, ved å definere en linjær rekkefølge av resursser slik at:
- dersom en prosess har en resurss, kan den kun få tilgang til resursser som kommer etter i rekkfølgen

Unngåelse (avoidance)

- Resurss-tilgang gis kun dersom det ikke vil oppstå sirkulær venting. Om det vil oppstå sirkulær venting, avgjøres ved tidspunkt for (1) forespørsel om prosessopprettelse eller (2) ved resursforespørsel fra en eksisterende prosess.
- Tillater de tre nødvendige betingelsene for vranglås.

Begrensninger

- Prosessens fremtidige forespørsler må være kjent på forhånd.
- Det må ikke være noen krav til synkronisering av prosessene. De må være uavhengige.
- Antall allokerbare resursser må være bestemt.
- Ingen prosess må avslutte mens den holder resursser.

Unngår

- ”preemption”.
- tilbakerulling (rollback)

Oppdagelse (detection)

- Algoritme for å oppdage vranglås

Måter å løse opp vranglås:

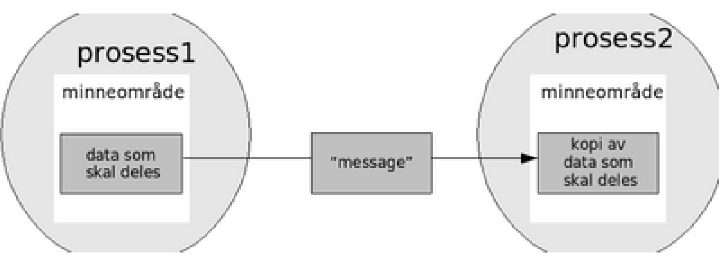
- Avslutte alle de fastlåste prosessene
- Tilbakeføre de fastlåste prosessene til en tidligere tilstand (”rollback”) og starte dem igjen.
- Avslutt en og en prosess i låsen, inntil låsen løses opp.
- ”Preempt” en og en resurss i låsen, inntil låsen løses opp.

“Message Passing”

Krav til samhanding av prosesser

- synkronisering - for gjensidig utelukkelse
- kommunikasjon - dele informasjon

"Message Passing" håndterer både synkronisering og kommunikasjon

- 

Primitiver

- send(destination, message)
- recieve(source, message)
- "mailboxes" og "ports" (se figur 5.18 s. 242 i Stallings)

`http://debbie.hive.no/osyda50/figurer/5.18.png`

- Typisk format med hode og kropp (se figur 5.19 s. 243 i Stallings)

`http://debbie.hive.no/osyda50/figurer/5.19.png`

- Passer til distribuerte systemer

Eksempler

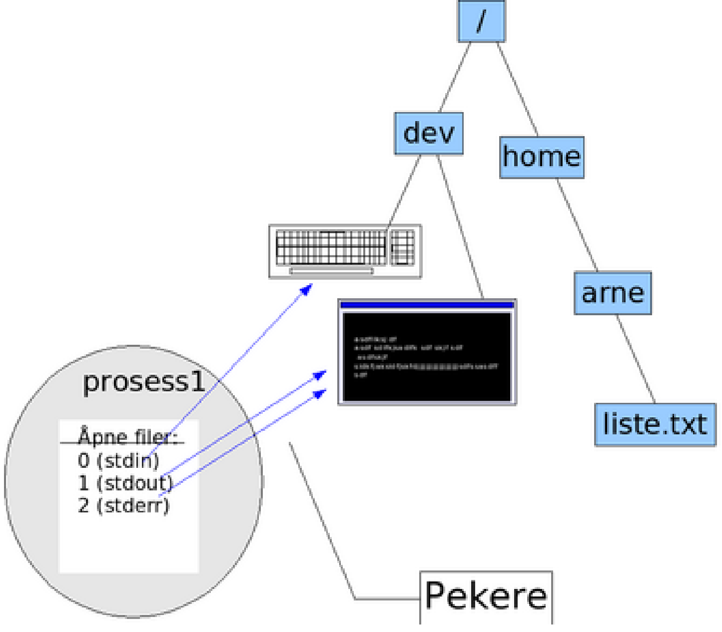
Eksempler fra UNIX

- message queues
- rør (pipes)
- navgitte rør (FIFOs/named pipes)

De to siste ser vi nærmere på...

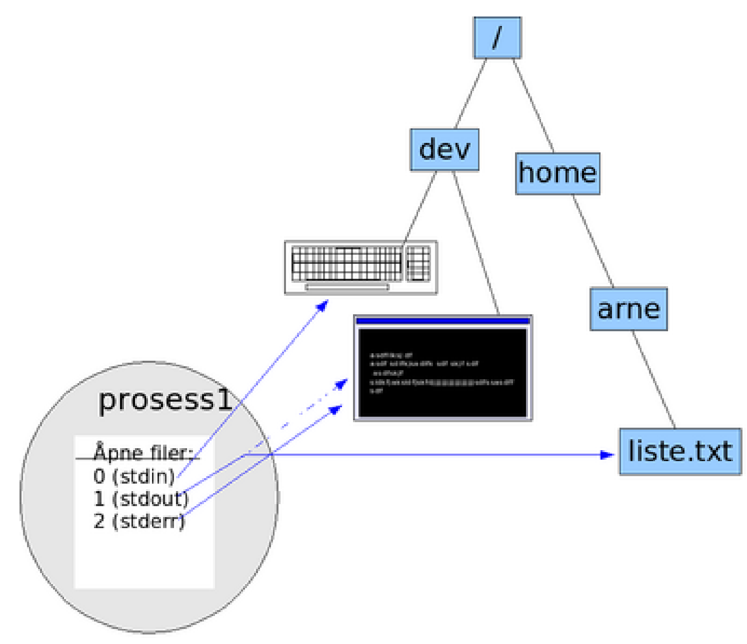
Rør og FIFO'er Rør («pipes»)
Redirigering (nå igjen...)
Standard inn og ut

-



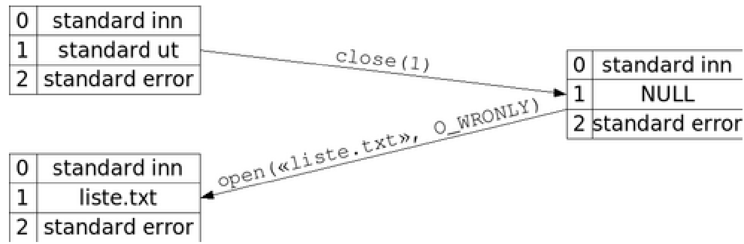
Redirigering til fil

•



Redirigering v.h.a. close() og open()
figur

•

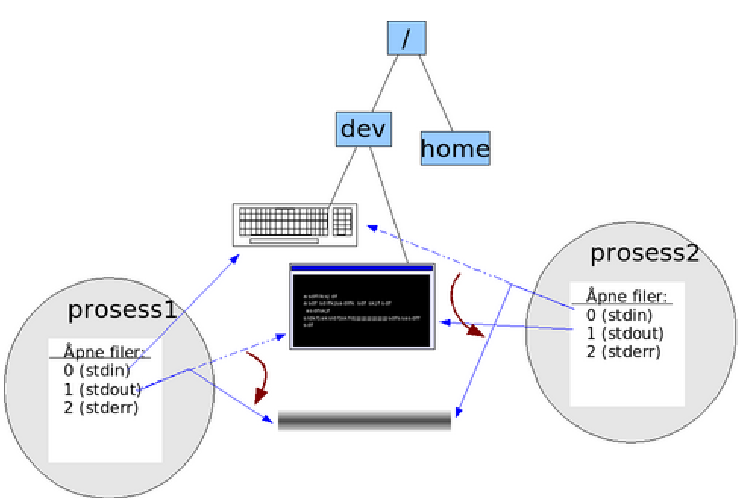


- Når en fil åpnes, får den tildelt den ledige fildeskriptoren med lavest verdi.
- Eksempel: redirigering.c

`http://debbie.hive.no/osyda50/eksempler/09/redirigering.c`

Redirigering til annen prosess v.h.a.
figur

-



rør («pipes»)
Duplisering av fildeskriptor

-

```
#include <unistd.h>int dup(int oldfd);int dup2(int oldfd, int newfd);
```

- Manualside: dup(2)

Opprettelse av rør

-

```
#include <unistd.h>int pipe(int filedes[2]);
```

- Manualside: pipe(2)

Eksempler

- pipe-eksempel_I.c - med dup()

`http://debbie.hive.no/osyda50/eksempler/10/pipe-eksempel_I.c`

- pipe-eksempel_II.c – med dup2()

`http://debbie.hive.no/osyda50/eksempler/10/pipe-eksempel_II.c`

Navngitte rør («FIFOs/named pipes»)

Filtyper i Unix

-

1. Vanlige filer
2. Kataloger
3. Karakter-enhetsfiler
4. Blokk-enhetsfiler
5. Symbolske lenker
6. "Local domain sockets"
7. **Navngitte rør (FIFO's)**

Hvorfor er så mye forskjellig representert som filer?

Bruk av navngitt rør på kommandolinjen

For å opprette en navgitt rør fra kommandolinjen:

- mkfifo navngitt_ror

Eksempel:

- Skrive og lese til fila fra to ulike konsollvinduer:
- lese: cat navngitt_ror
- skrive: cat > navngitt_ror

Opprettelse av node i et filsystem

-

```
#include <sys/types.h>#include <sys/stat.h>#include <fcntl.h>#include <unistd.h>int mknod(
```

- mode: Kombinasjon (bitvis eller) av rettighetsmaske og filtype.
- dev: NULL hvis ikke filtypen er blokk-enhet/karakter-enhet
- Manualside: mknod(2)

Eksempler

- navngitt_pipe.c

`http://debbie.hive.no/osyda50/eksempler/10/navngitt_pipe.c`

- pipe-tjener.c

`http://debbie.hive.no/osyda50/eksempler/10/pipe-tjener.c`

9.4 Oppgaver

9.4

- a) Forklar hva som forårsaker feilen i programmet finn_feilen_2.c

Listing 9.7: oppgaver/09/finn_feilen_2.c

```
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <unistd.h>
4  #include <stdlib.h>
5  #include <time.h>
6
7  /*
8
9    Programmet simulerer flytting av penger mellom N antall bankkonti
       .
10   Flytting av penger gjoeres i en egen traad.
11
12   Hovedtraaden skriver saldoer hvert sekund.
13   Her er det et problem foraarsaket av samtidig kjoering av de to
       traadene.
14   Problemet foerer til at summen av kontiene saldo ikke beregnes
       riktig.
15
16   a) Kan du forklare hvordan feilen oppstaar?
17   b) Kan du fikse feilen? (f.eks. med semaforer)
18
19   Husk aa ta med argumentene "-lpthread" og
20   "-D_REENTRANT" ved kompilering.
21
22   Thomas Nordli <tn@hive.no>
23
24 */
25
26 #define T 1000   // Totalsummen i spillet
27 #define N 5      // Antall konti
28
29
30 void opprett_konti();
31 void *flytt_penger();
32 void skriv_saldo();
33
34 pthread_t spekulant;
35 int konto[N];
36
37 int main(){
38
39   srand( time(NULL)); // Initierer slumtallgenerator
40   opprett_konti();
41   pthread_create(&spekulant, NULL, flytt_penger, NULL);
42   skriv_saldo();
43
44   while(1){
45
46     sleep(1);
47     skriv_saldo();
48
49   }
50
51   return 0;
```

```
52
53 }
54
55
56 void skriv_saldo(){
57
58   int sum,i;
59
60   for (i=0, sum=0; i<N; i++){
61
62     printf("konto %d:\t%d\n", i, konto[i]);
63     sum+=konto[i];
64
65   }
66
67   printf("---------------\nTotalt:\t%d\n===============\n",sum);
68
69 }
70
71 void opprett_konti(){
72
73   int i;
74
75   for (i=0;i<N;i++)
76     konto[i]=T/N;
77
78 }
79
80 void *flytt_penger(){
81
82   int til, fra, belop;
83
84   while(1){
85
86     fra=rand()%N;
87     til=rand()%N;
88     belop=rand() % ( (konto[fra]/10) );   // aldri mer enn 10% av
         saldo
89
90     konto[fra]-=belop;
91     konto[til]+=belop;
92
93   }
94
95   return NULL;
96
97 }
```

- b) Løs problemet v.h.a. semaforer.

10.1

- Observer og forklar hva som skjer når du gjør følgende:

a)

Lag en FIFO i arbeidskatalogen med:

- mkfifo fifo-fil.

b)

Åpne FIFO'n i en grafisk nettleser på samme maskin som du lagde FIFO'n:

- firefox fifo-fil &

c)

I et konsoll-vindu, skriv:

- echo Hallo > fifo-fil

10.2

- Eksemplene produsent-konsument_1.c, produsent-konsument_2.c og produsent-konsument_3.c gjør ingenting.

a)

- Endre programmene slik at de jobber mot en tabell (array) av typen char.
- La produsenten skrive bokstaver inn i tabellen.
- La konsumenten hente bokstavene ut av tabellen og skrive dem ut på standard utgang ("stdout").

b)

- Skriv om programmet i a) slik at du i stedet for semaforer og char[], bruker et navngitt rør.

10.3

- Lag et program med to tråder og to resursser, slik at betingelsene for vranglås er tilstede. Som resurrser kan dere bruke enkle variabler f.eks. en 'char' eller 'int'. Bruk semaforer til gjensidig utelukkelse.

10.4

- Eksemplet pipe-tjener.c har kun enveis-kommunikasjon. Gjør den toveis ved å:
- Opprett en FIFO til, og rediriger utskriften fra pipe-tjeneren inn i den.
- Lag en klient til pipe-tjeneren.

Kapittel 10

11 - minne og signaler

10.1 Løsninger

10.2

Oppgave

- Eksemplene produsent-konsument_1.c, produsent-konsument_2.c og produsent-konsument_3.c gjør ingenting.

a)

- Endre programmene slik at de jobber mot en tabell (array) av typen char.
- La produsenten skrive bokstaver inn i tabellen.
- La konsumenten hente bokstavene ut av tabellen og skrive dem ut på standard utgang ("stdout").

Løsningsforslag på oppgave 10.2a)

- sirkeltabell.c

Listing 10.1: losninger/10/sirkeltabell.c

```
1  #include <unistd.h>
2  #include <fcntl.h>
3
4  int main (){
5
6    char abc[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
7
8    int a_len = sizeof(abc);
9    int b = 0;
10   int i;
11
12   for (i=0; i< 5*a_len; i++) {
13
14     write( 1, abc+b, 1 );
15     b++;
16     b %= a_len;
```

```
17    }
18
19    write( 1, "\n", 1);
20
21    return 0;
22  }
```

- 10.2.a.c

Listing 10.2: losninger/10/10.2.a.c

```
1   #include <semaphore.h>
2   #include <pthread.h>
3   #include <unistd.h>
4   #include <stdio.h>
5
6   #define LAGERSTR 6
7
8   /*
9     Husk -pthread ved kompilering med gcc'
10   */
11
12
13  void *produser();
14  void *konsumer();
15
16  pthread_mutex_t laas= PTHREAD_MUTEX_INITIALIZER;
17
18  sem_t ledige_plasser;
19  sem_t klare_data;
20
21  char lager[LAGERSTR];
22
23  int inn = 0;
24  int ut  = 0;
25
26  int main (){
27
28    pthread_t konsument;
29
30    sem_init(&ledige_plasser, 0, LAGERSTR );
31    sem_init(&klare_data, 0, 0 );
32
33    pthread_create(&konsument, NULL, konsumer, NULL);
34
35    // produsent
36
37    while(1){
38
39      sem_wait(&ledige_plasser);
40      pthread_mutex_lock(&laas);
41
42      if ( 1 > read(0, lager+inn, 1) )
43        break;
44
45      inn++;
46      inn%=LAGERSTR;
47
48      pthread_mutex_unlock(&laas);
49      sem_post(&klare_data);
50
```

```
51    }
52
53    sleep(1);
54    return 0;
55
56  }
57
58  void *konsumer(){
59
60    while(1){
61
62      sem_wait(&klare_data);
63      pthread_mutex_lock(&laas);
64
65      printf("%c", lager[ut]);
66      //write(1, lager+ut, 1);
67
68      ut++;
69      ut%=LAGERSTR;
70
71      pthread_mutex_unlock(&laas);
72      sem_post(&ledige_plasser);
73    }
74  }
```

10.2 Repetisjon: Pipes

- eksempler/11/pipe_illustrated.c

Listing 10.3: eksempler/11/pipe_illustrated.c

```
1   #include <sys/types.h>
2   #include <sys/wait.h>
3   #include <unistd.h>
4   #include <stdio.h>
5
6   void skriv_fdtab(char *overskrift);
7
8   int main () {
9
10    /* Bruker funksjonen skriv_fdtab()
11       til aa vise fildeskriptortabellen
12       til en prosessene underveis i
13       eksemplet 'pipe-eksempel_I.c'
14
15       To do:
16       Programmet trenger synkronisering
17       av utskrift til konsollet.
18     */
19
20    int fda[2];
21
22    skriv_fdtab("Ved oppstart");
23    pipe(fda);
24    skriv_fdtab("Etter pipe(fd)");
25
26
27
28    if(fork() == 0) {
29
```

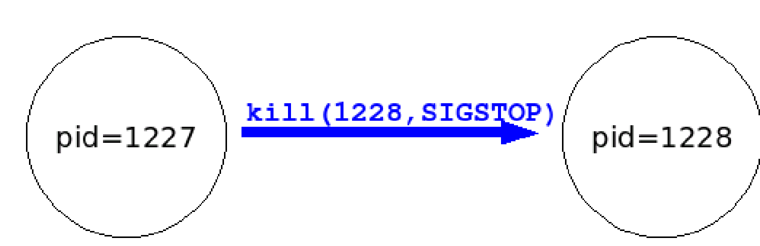
```
30      // Stenger standard utgang
31      close(1); skriv_fdtab("Etter close(1)");
32
33      // Dupliserer skrive-enden (inngangen) av roeret
34      dup(fda[1]); skriv_fdtab("Etter dup(fda[1])");
35
36      // stenger lese-enden (utgangen) av roeret
37      close(fda[0]); skriv_fdtab("Etter close(fda[0])");
38
39      // stenger skrive-enden (inngangen) av roeret
40      close(fda[1]); skriv_fdtab("Etter close(fda[1])");
41
42      execl("/bin/ls", "ls", NULL);
43
44    }
45
46    else {
47
48      // stenger standard inngang
49      close(0);  skriv_fdtab("Etter close(0)");
50
51      // dupliserer lese-enden (utgangen) av roeret
52      dup(fda[0]); skriv_fdtab("Etter dup(fda[0])");
53
54      // stenger lese-enden (utgangen) av roeret
55      close(fda[0]); skriv_fdtab("Etter close(fda[0])");
56
57      // stenger skrive-enden (inngangen) av roeret
58      close(fda[1]); skriv_fdtab("Etter close(fda[1])");
59
60      execl("/usr/bin/rev", "rev", NULL);
61
62    }
63
64    return 1;
65
66  }
67
68
69  void skriv_fdtab(char *overskrift) {
70
71    /*Funksjonen skriver ut fil-deskriptor-tabellen
72      til prosessen som kaller den
73     */
74
75    char sti[50];
76    int pid = getpid();
77
78    sprintf(sti, "/proc/%d/fd", pid);
79
80    if ( fork() == 0 ){
81      dup2(2,1); // Soerger for at utskriften kommer paa konsollet
82      printf("\n\n%s(%d)\n--------\n", overskrift,pid);
83      fflush(NULL);
84      sleep(1);
85      execl("/bin/ls","ls","-l", sti, NULL);
86
87    }
88
89    wait(NULL);
90
91  }
```

10.3 Signaler

- Prosesser kan kommunisere v.h.a. signaler.

Signaler i linux

figur

- 

-

```
struct task_struct {
 ...
 struct signal_struct *signal;
 ...
}
```

UNIX-programmer for å sende signaler:

- kill(1)
- killall(1)
- Oversikt over de ulike signalene finnes i manualen, seksjon 7: man 7 signal

Sende et signal til en prosess

-

```
#include <types.h>
#include <signal.h>

int kill(pid_t pid, int sig);
```

- kill(2)
- Eksempel: kill-eksempel.c

Listing 10.4: eksempler/11/kill-eksempel.c

```
1 #include <stdio.h>
2 #include <signal.h>
3 #include <unistd.h>
4
5 /*
6   Demonstrasjon av kill(2)
```

```
7
8  */
9
10 int main(void)
11 {
12
13   int forkretur, den_andres_pid, i;
14   char *tekst;
15
16   forkretur=fork();
17
18   if(forkretur==0){
19     tekst="BARN";
20     den_andres_pid = getppid();
21   }
22
23   else{
24     tekst="FORELDRE";
25     den_andres_pid = forkretur;
26   }
27
28   for (i = 0; i < 10; i++){
29     printf("%s\n",tekst);
30     sleep(1);
31     kill(den_andres_pid, SIGCONT);
32     kill(getpid(), SIGSTOP);
33   }
34
35   return 0;
36 }
```

Oppfange signaler

-

```
#include <signal.h>

int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
struct sigaction {
 ...
 void (*sa_handler)(int);
 sigset_t sa_mask;
 int sa_flags;
 ...
}
```

- sigaction(2)
- Eksempel: signal-eksempel.c

Listing 10.5: eksempler/11/signal-eksempel.c

```
1  #include <stdio.h>
2  #include <signal.h>
3  #include <unistd.h>
4
5  /*
6    Demonstrasjon av hvordan utfoere
```

```
7     egendefinert rutine ved mottak
8     av signal.
9
10  */
11
12  void signalrutine(int sig)
13  {
14    printf("/\\");
15  }
16
17  int main(void)
18  {
19
20    struct sigaction handling;
21
22    // Skriver ut egen pid
23    printf("%d\n", getpid());
24
25    // Initierer sigaction-struktur
26    handling.sa_handler = signalrutine;
27    (void) sigemptyset(&handling.sa_mask);
28    handling.sa_flags = 0;
29
30    // Aktiverer ny signalrutine
31    sigaction(SIGINT, &handling, 0);
32
33    while(1){
34      printf(".");
35      fflush(NULL);
36      sleep(1);
37    }
38
39    return 0;
40
41  }
```

Vente på terminering av en prosess

-

```
#include <types.h>
#include <wait.h>

pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
```

- wait(2)

Eksempler

- konkurranse.c

Listing 10.6: eksempler/11/konkurranse.c

```
1  #include <sys/types.h>
2  #include <unistd.h>
3  #include <stdio.h>
4
5  /*
```

```
6     Demonstrasjon av konkurerrende
7     kjoering av to prosesser.
8
9   */
10
11  int main(void)
12  {
13
14    int forkretur, i, j;
15    char *tekst;
16
17    forkretur=fork();
18
19    if(forkretur==0)
20      tekst="BARN";
21    else
22      tekst="FORELDRE";
23
24    for (i = 0; i < 20; i++) {
25      for (j = 0; j < 9999999; j++)
26        ;
27      printf("%s\n",tekst);
28      fflush(NULL);
29    }
30
31    return 0;
32
33  }
```

- wait-eksempel_I.c

Listing 10.7: eksempler/11/wait-eksempel_I.c

```
1   #include <sys/types.h>
2   #include <unistd.h>
3   #include <stdio.h>
4   #include <wait.h>
5
6   /*
7     Demonstrasjon av kjoering av to prosesser,
8     hvor den ene venter paa den andres terminering.
9
10  */
11
12  int main(void)
13  {
14
15    int forkretur, i, j;
16    char *tekst;
17
18    forkretur=fork();
19
20    if(forkretur==0)
21      tekst="BARN";
22    else{
23      tekst="FORELDRE";
24      (void) wait(NULL);
25    }
26
27    for (i = 0; i < 20; i++){
28      for (j = 0; j < 9999999; j++)
```

```
29          ;
30        printf("%s\n",tekst);
31        fflush(NULL);
32      }
33
34      return 0;
35
36  }
```

10.4 Minnehåndtering

krav til minnebehandling

- figur 7.1 s. 313 illustrerer addreseringskravene til en prosess.

`http://debbie.hive.no/osyda50/figurer/7.1.png`

”relocation”

- prosessene bør kunne eksistere i varierende posisjon i minnet
- Vi ser på figur 7.8

`http://debbie.hive.no/osyda50/figurer/7.8.png`

- Logisk adresse
- Relativ adresse
- Base- og ”bounds”-register
- Absolutt adresse

protection

- prosessen må kun lese og skrive i tildelt område av minnet.

deling

- minnebehandlings-systemet bør gjøre det mulig for prosesser å dele minne.

logisk organisering

- De fleste dataprogram er delt inn i ulike deler avhengig funksjonalitet. Noen deler skal f.eks ikke være skrivbare. Noen skal være kjørbare, etc.

Segmentering er et eksempel på en en slik logisk organisering av minnet tildelt prosesser.

- En prosess deles i ulike segmenter.
- Segmentene kan ha ulik størrelse
- Segmentene lastes inn i minnet.

Eks:

- pmap $$

fysisk organisering

minst to nivåer:

- primærminne
- sekundærminne
- minnehierarkiet

partisjonering av minnet

statisk partisjonering

- Vi ser på figur 7.2 side 317

`http://debbie.hive.no/osyda50/figurer/7.2.png`

- En partisjon pr. prosess
- Antall partisjoner er fast
- Intern fragmentering

dynamisk partisjonering

- Vi ser på figur 7.4 side 320

`http://debbie.hive.no/osyda50/figurer/7.4.png`

- Partisjonene opprettes ved behov
- Ekstern fragmentering
- Størrelsen tilpasses prosessenes behov
- ”Compactiton”

Noen bibliotekskall for minnehåndtering

void *malloc(size_t size);

- allokerer minne for size antall byte
- minnet blir ikke initialisert!

returnerer

- peker til allokert område, eller
- NULL hvis minne ikke ble allokert

void *calloc(size_t nmemb, size_t size);

- allokerer minne for en array med nmemb antall elementer av størrelse size.
- minnet blir initialisert med nuller.

returnerer

- peker til allokert område, eller
- NULL hvis minne ikke ble allokert

void free(void *ptr);

- frigjør minneområde som ptr peker på.
- ptr må være returverdien fra et tidligere oppkall til malloc(),calloc()eller realloc().
- returnerer ingen verdi

Eksempler

- eksempel: malloc_eks.c

Listing 10.8: eksempler/11/malloc_eks.c

```
1  #include <unistd.h>
2  #include <stdlib.h>
3
4  int main(){
5
6    char s1[]="1234567890";
7    char s2[]="abc";
8    char nl[]="\n";
9
10   char *t2, *t1;
11   int i;
12
13   t1 = malloc(10);
14   for (i=0; i<10; i++)
15     t1[i] = s1[i];
16   write(1, t1, 10);
17   free(t1);
18   write(1,nl,1);
19
20   t2 = malloc(10);
21   for (i=0; i<3; i++)
22     t2[i] = s2[i];
23   write(1, t2, 10);
24   free(t2);
25   write(1,nl,1);
26
27   return 0;
28
29 }
```

- eksempel: calloc_eks.c

Listing 10.9: eksempler/11/calloc_eks.c

```
#include <unistd.h>
#include <stdlib.h>

int main(){

  char s1[]="1234567890";
  char s2[]="abc";
  char nl[]="\n";

  char *t1, *t2;
  int i;

  t1 = calloc(10,1);
  for (i=0; i<10; i++)
    t1[i] = s1[i];
  write(1, t1, 10);
  free(t1);
  write(1,nl,1);

  t2 = calloc(1,10);
  for (i=0; i<3; i++)
    t2[i] = s2[i];
  write(1, t2, 10);
  free(t2);
  write(1,nl,1);

  return 0;

}
```

Vanlige feil i forbindelse med frigjøring av minne

- bruk av frigjort minne v.h.a. «dangling pointer»
- frigjøring av samme minneområde flere ganger
- forsøk på frigjøring av minne, v.h.a. en peker som ikke er returverdi fra malloc().
- bruk av minne som er utenfor allokert «chunk»

unnlate å frigjøre minne

- minnelekasje
- trashing

buffer overflow

- Vi ser på eksempel spill.c

Listing 10.10: eksempler/11/spill.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

```

```
5  void gratulerer(){printf("gratulerer\n");}
6  void kondolerer(){printf("kondolerer\n");}
7
8  void spill(int gjetning, int terning){
9
10   if(gjetning == terning)
11     gratulerer();
12   else
13     kondolerer();
14 }
15
16
17 int main(){
18
19   int terning, gjetning;
20   static void (*peker)(int, int);
21   static char buffer[2];
22   peker=spill;
23
24   // "Saar et tilfeldig froe"
25   srand(time(NULL));
26
27   // Kaster en terning
28   terning = rand()%12+1;
29
30   printf("Terningene er kastet. Gjett tallet (1-12)\n");
31   scanf("%s",buffer);
32   printf("Du skrev %s, og det ble lagret i %p.\n", buffer, buffer);
33   printf("Pekeren inneholder %p, og ble lagret i %p.\n", peker, &
       peker);
34   gjetning = atoi(buffer);
35   peker(gjetning, terning);
36
37   return 0;
38
39 }
```

- Vi kompilerer spillet:
-

```
gcc spill.c -o spill
```

- Vi gjetter tallet 2:
-

```
echo 2 | ./spill
```

- Sjansen for å vinne er 1/12.
- Vi oversvømmer bufferet:
-

```
echo -e "AAAAAAAAAAAA"  | ./spill
```

- Hvor kommer tallet 41 fra?
- man ascii

- Vi kan angi ascii-verdien til 'echo':
-

```
echo -e "\x41"
echo -e "\x00\x01\x02\x03\x04\x05\x06\x08" | ./spill
```

- Nå kan vi se at vi kan endre innholdet i 'peker' til det vi måtte ønske.
- Vi finner adressen til funksjonen gratulerer:
-

```
nm spill | grep gratulerer
```

- Nå kan vi vinne hver eneste gang.

10.5 Oppgaver

11.1

- I denne oppgaven skal du lage et program som simulerer terningkast uten å bruke noen randomfunksjon.

a)

- Endre programmet losninger/10/sirkeltabell.c slik at det skriver tallene 1 til 6 i stedet for alfabetet.

b)

- Endre programmet i a) slik at det skriver tallene fra 1 til 6, i en evig løkke.

c)

- Endre programmet i b) slik at løkken forsatt går i evig løkke, men at utskrift av et tall kun kommer når prosessen får mottar et signal. Et tall for hvert signal. Velg selv hvilket signal du ønsker å benytte.

11.2

- Lag et program som dramatisk endrer minnebruken under kjøring. Inspiser under kjøring med f.eks. top.

11.3

Demoner (tjenere) er ofte implementert slik at de leser sine konfigurasjonsfiler ved oppstart og ved mottak av signalet SIGHUP. Lag et program som simulerer dette. Ved å implementere en funksjon som:

- 1. kalles ved oppstart
- 2. kalles ved mottak av signalet SIGHUP
- 3. åpner en fil med navn oppgave3.ini
- 4. leser et heltall fra filen, og lagrer det i en variabel.

11.4

a)

- Utnytt sårbarheten i eksempler/11/spill.c til å vinne hver gang, slik det ble demonstrert i forelesningen

b)

- Utnytt sårbarheten i spill.c til å tape hver gang.

Kapittel 11

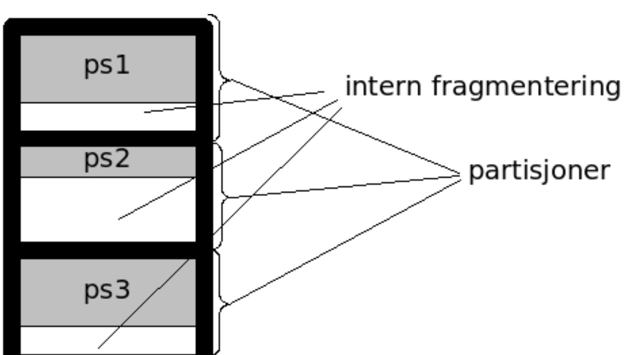
12 Sideveksling, virtuelt minne og sikkerhetstrusler

11.1 Kort repetisjon fra forrige gang

Partisjonering av minnet

Statisk partisjonering

figur

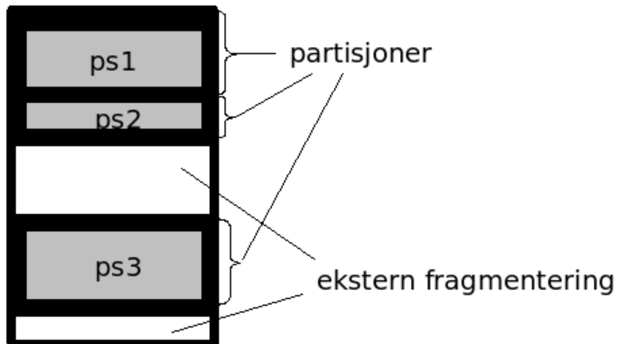
- 

- En partisjon pr. prosess
- Antall partisjoner er fast
- Intern fragmentering

Dynamisk partisjonering

figur

-



- Partisjonene opprettes ved behov
- Størrelsen tilpasses prosessenes behov
- Ekstern fragmentering
- "Compactiton"

Segmentering

- En prosess deles i ulike segmenter.
- Segmentene kan ha ulik størrelse
- Segmentene lastes inn i minnet.
- Programmerer/kompilator kan kontrollere
- Ulike rettigheter (lese/skrive/kjøre)

Typiske inndeling i segmenter:

- tekst-segment
- data-segment
- stakk

Eks:

- pmap $$

11.2 Sideveksling (paging)

- minnet delt i rammer
- rammene har samme størrelse
- prosessen deles i sider (pages)
- sidene har samme størrelse som rammene
- Vi får ingen ekstern fragmentering
- Vi får litt intern fragmentering

Vi ser på figurer

- 7.9

`http://debbie.hive.no/osyda50/figurer/7.9.png`

- 7.11(b)

`http://debbie.hive.no/osyda50/figurer/7.11.png`

- 7.12(a)

`http://debbie.hive.no/osyda50/figurer/7.12.png`

11.3 Virtuelt minne

Forutsetninger

Hvis

- oversettelse fra logisk til fysisk adresse skjer dynamisk ved ”runtime” (av MMU).
- og en prosess kan deles opp i mindre biter

så

- trenger ikke hele prosessen være i minnet samtidig under kjøring
- Delene i minnet utgjør ”resident set”
- ”principle of locality”

Rutine for minnetilgangsfeil (Memory Access Fault):

- referanse til minne utefor ”resident set”
- avbrudd indikerer minnetilgangsfeil
- prosess settes i blokkert tilstand
- OS sender I/O-read for å hente siden som mangler
- OS kjører i gang annen prosess

Når I/O-operasjonen er fullført

- sendes avbrudd
- og prosess-tilstanden endres til ”klar”

Fordeler

- prosesser kan være større enn primærminnet

flere prosesser i minnet om gangen

- større sannynlighet for at det finnes prosesser klare til kjøring
- -> høyere grad av prosessor-utnyttelse

«Thrashing»

- Hvis det fysiske minnet er fullt: For at en side skal kunne hentes inn, må enn annen settes ut.
- Det hender at siden som settes ut, snart må hentes inn igjen.
- Hvis dette skjer for ofte - kaller vi det for "thrashing"
- Hva kan vi gjøre?

Virtuelt minne (vm) m/sideveksling

Vi ser på figurer fra boka

- 8.2(a) - adresseformat

`http://debbie.hive.no/osyda50/figurer/8.2.png`

- 8.3 – oversettelse av adresser

`http://debbie.hive.no/osyda50/figurer/8.3.png`

Translation Lookaside Buffer

- Page Table er stor – får ikke plass i registre

En egen cache for sidetabellen:

- Translation Lookaside Buffer

Vi ser på figurer fra boka:

- 8.7 – eks. på hw-implementasjon

`http://debbie.hive.no/osyda50/figurer/8.7.png`

- 8.8 - flytdiagram

`http://debbie.hive.no/osyda50/figurer/8.8.png`

- 8.9 - «associative mapping»

`http://debbie.hive.no/osyda50/figurer/8.9.png`

VM m/sideveksling og segmentering

Vi ser på figur fra boka

- 8.12 – Adresseoversettelse ved segmentering

`http://debbie.hive.no/osyda50/figurer/8.12.png`

- 8.13 – Adresseoversettelse ved sideveksling kombinert med segmentering

`http://debbie.hive.no/osyda50/figurer/8.13.png`

Opprettelse av veksleområde i Linux

Opprettelse av veksleområde

- Veksleområde kan være en fil eller en partisjon.

Følgende må gjøres for å sette opp et nytt veksleområde:

- Klargjøre området (Eks.: mkswap /dev/hda4)
- Aktivere området (Eks.: swapon /dev/hda4)
- Deaktivere området (Eks.: swapoff /dev/hda4)
- Aktivering av veksleområder ved oppstart
- For at aktivisering skal skje automatisk ved oppstart, må det legges til en ny linje i filen /etc/fstab.

eks. på /etc/fstab

-

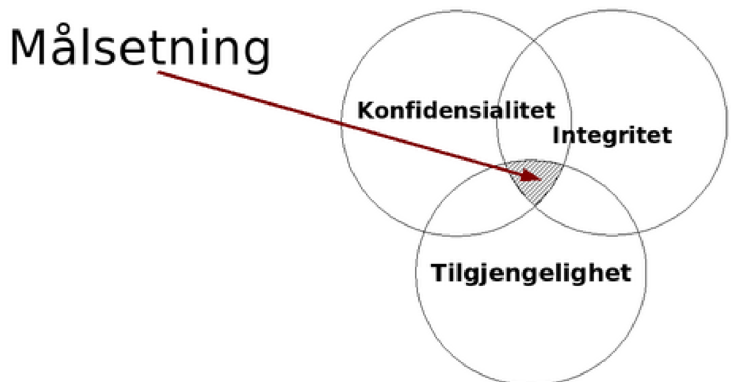
```
# /etc/fstab: filesystem table. #
# filesystem  mountpoint  type  options  dump  pass
/dev/hda1   /        ext3   defaults,errors=remount-ro  0  1
/dev/hda2   none     swap   sw                          0  0
proc        /proc    proc   defaults                    0  0
/dev/fd0    /floppy  vfat   defaults,user,noauto,showexec,umask=022  0  0
/dev/cdrom  /cdrom   iso9660  defaults,ro,user,noexec,noauto        0  0
/dev/cdrom1 /cdrom1  iso9660  defaults,ro,user,noexec,noauto        0  0
/dev/dvd    /dvd     iso9660  defaults,ro,user,noexec,noauto        0  0
/dev/cdaudio /cdaudio  iso9660  defaults,ro,user,noexec,noauto      0  0
```

11.4 Sikkerhetstrusler

Konsepter

tre hovedmål

-



- konfidensialitet
- integritet
- tilgjengelighet

noen forfekter to "ekstramål"

- autentisitet
- "accountability"

Trusselbildet

Trusler

Trusler og angrepsformer

- Vi ser på tabell 14.1

`http://debbie.hive.no/osyda50/figurer/t14.1.png`

Sikkerhetssystemets ramme

- Vi ser på figur 14.2

`http://debbie.hive.no/osyda50/figurer/14.2.png`

Trusler og "assets"

- Vi ser på tabell 14.2

`http://debbie.hive.no/osyda50/figurer/t14.2.png`

Inntrengere

AND80's klassifisering

- masquerader
- misfeasor
- clandestine user

RADC04's klassifisering

- hacker
- kriminell organisasjon
- intern trussel

Ondsinnet kode - (malware)

parasittisk

- virus
- logisk bombe
- bakdør

selvstendig

- ormer

"bots"

- DDOS (Distributed Denial of Service) - distritibuert tjenestenektangrep
- spam
- sniffing
- keylogging
- spredning av malware
- spreding av adware
- manipulering av avstemminger/spill

Trojansk hest

- Program gjør noe annet enn det det tilsynelatende gjør.

Eksempel:

- I Ken Thompsons 'Turing Award 1984'-tale beskriver han hvordan en bakdør plasseres i login-programmet til UNIX, uten at det kan spores i noe kildekode. Hverken i login-programmets kildekode eller kompilatorens kildekode.

`http://cm.bell-labs.com/who/ken/trust.html`

mobil kode

- plattform-uavhengighet
- f.eks. makrovirus/skript

rootkit

- opprettholde root-tilgang
- endrer systemets funksjonalitet
- godt skjult
- eks.: figur 14.6

`http://debbie.hive.no/osyda50/figurer/14.6.png`

11.5 Oppgaver

12.1

- Lag et program 'minneallokasjon.c' som leser inn et heltall fra standard inn. Programmet skal så allokere det antall bytes som brukeren skrev inn. Tips: malloc(3)

12.2

a)

- Start en virtuell maskin på debbie med kommandoen: vstart minnetest.

b)

- Finn ut hvor mye virtuelt minne den virtuelle maskinen har. Ikke steng den virtuelle maskinen ennå, for du skal bruke den mer. Tips: free(1), top(1).

c)

- Du finner din "debbie-hjemmekatalog" i katalogen '/hosthome' på den virtuelle maskinen 'minnetest'. På den virtutelle maskinen 'minnetest' skal du finne og kjøre programmet du lagde i Oppgave 1. Oppgi et antall bytes som er større en det virtuelle minnet du fant i b). Hva skjer?

12.3

a)

- I denne oppgaven skal du øke det virtuelle minnet til den virtuelle maskinen 'minnetest' v.h.a.en såkalt 'vekslefil'.
-

```
dd if=/dev/zero of=/vekslefil bs=1024 count=40
```

-

```
mkswap /vekslefilswapon /vekslefil
```

e)

- Prøv nå å gjenta det du gjorde i Oppgave 2 c) med samme antall bytes.

f)

- Gi kommandoen 'halt' i konsollet på den virtuelle maskinen for å ta den ned. Eller 'vhalt minnetest' i konsollet til verskmaskinen (debbie).

Kapittel 12

13 - Sikkerhetstiltak (og litt mer om trusler)

12.1 Autentisering

Definisjon

- "The process of verifying an identity claimed by or for a system entity"
- (ref.: ietf.org/rfc/rfc2828)

http://ietf.org/rfc/rfc2828

To faser

- Identifisering
- Verifisering

Generelle metoder

noe du vet

Eksempel: Passord

- Figur 15.1

http://debbie.hive.no/osyda50/figurer/15.1.png

hashing

- DES
- MD5
- Blowfish

salt

- like passord fremstår ikke
- ”dictionary attack” blir vanskeligere

Passord i Linux

- På linux brukes krypteringsalgoritmen enveis-DES (eller MD5). At den er enveis, vil si at algoritmen ikke kan dekryptere det den har kryptert.
- Brukerenes passord blir lagret kryptert i en fil (/etc/passwd el. /etc/shadow).
- For å autentisere en bruker, blir det passordet brukeren taster inn, kryptert med samme algoritme, og sammenlignet med det krypterte passordet i filen.
- Det finnes programmer ved å benytte seg av ordlister og kjente krypteringsalgoritmer (og mye prossessortid), som klarer å finne enkle passord. F.eks. Crack og John the Ripper.
- crypt()

`http://manpages.debian.net/cgi-bin/man.cgi?query=crypt&apropos=0&sektion=3&manpath=Debian+5.0+lenny&format=html&locale=en`

noe du har

- «Token» = berettigelsestegn

Eksempler på ”tokens”

minnekort

- krever leser
- ofte kombinert med passord/PIN

eks.

- minibankkort
- nøkkelkort

smartkort grensesnitt
elektronisk (krever leser)

- elektriske kontakpunkter
- trådløs forbindelse
- manuell (taster og display)

algoritme

- statisk
- dynamisk passordgenerator
- ”challenge-response”

noe du er/gjør

Eksempler på biometri

- Ansiktsgjenkjenning
- Fingeravtrykk
- håndgeometri
- retina-mønster
- Iris
- Underskrift/signatur
- Stemme

12.2 Tilgangskontroll

Tre kategorier av adgangskontroll-policies:

DAC

- Figur 15.4

`http://debbie.hive.no/osyda50/figurer/15.4.png`

- Dicretionary Access Control - brukerbestemt tilgangskontroll
- Tilgang til objekter er basert på subjektets identitet og autoriseringsregler.
- Subjektet kan ha rett til å gi rettigheter videre til andre - derav ordet 'discretionary'.

Eks. filrettigheter i UNIX

fig.: ACL i linux

-

ACL i Linux

- trad. UNIX tilnærming – minimal ACL

		User	Group	Others
		Eieren	Gruppemedlemmer	Alle Brukere
R	Lese	0/1	0/1	0/1
W	Skrive	0/1	0/1	0/1
X	Utføre	0/1	0/1	0/1

0 – 7 0 – 7 0 – 7

rwx rwx rwx

U G O

fig.: Notasjon

- Notasjon av filrettigheter i UNIX/Linux

  Eksempel:

  -rw-rw-r-- 1 tn tn 20K jan 5 15:16 foreles1.sxi

  Binær notasjon: 110 110 100

  Otal notasjon: 6 6 4

MAC

- Mandatory Access Control
- Objekter er klassifisert
- Subjekter er klarert
- Tilgang er basert på subjektets klarereingsnivå og objektets klasse.
- Subjekter kan ikke gi rettigheter videre - derav ordet 'mandatory'
- Reglene settes sentralt.
- Eks.: SELinux (utviklet av NSA)

`http://www.nsa.gov/public_info/press_room/2001/se-linux.shtml`

RBAC

- Role Based Access Control - rollebasert tilgangskontroll.
- Figur 15.7

`http://debbie.hive.no/osyda50/figurer/15.7.png`

12.3 IDS - Intrusion Detection System

”Host-based IDS”

- Programmer for integritetskontroll beregner tall ut fra innholdet til filer, sjekk-summer (en: checksum), slik at tallet blir annerledes dersom innholdet er endret.
- Tallene lagres i en database sammen med filattributter.
- Programmet kjøres periodisk, f.eks. som en ”cron”-jobb, og kan dermed varsle endringer i konfigurasjonfiler og viktige systemprogrammer.
- Tips: Det kan være lurt å kjøre programmet fra en medium som er skrivebeskyttet. (f.eks.cdrom)

Programmer som utfører integritets-kontroll:

- aide
- tripwire
- osiris

Data kilde

Host Based

- logger
- systemtilstand

Eks.: Integritets-kontroll

- Programmer for integritetskontroll beregner tall ut fra innholdet til filer, sjekk-summer (en: checksum), slik at tallet blir annerledes dersom innholdet er endret.
- Tallene lagres i en database sammen med filattributter.
- Programmet kjøres periodisk, f.eks. som en ”cron”-jobb, og kan dermed varsle endringer i konfigurasjonfiler og viktige systemprogrammer.

Tips:

- Det kan være lurt å kjøre programmet fra en medium som er skrivebeskyttet. (f.eks.cdrom)

Programmer som utfører integritets-kontroll:

- aide
- tripwire
- osiris

Network Based

- nettverkstrafikk
- eks.: snort

`http://www.snort.org/`

Analyse metode

- Anomalideteksjon
- Signaturbasert

12.4 Antivirus

- Signaturer (må oppdateres)

Vanskelig å forhindre virusinfeksjon...

- detektere infeksjon
- identifisere viruset
- fjerne viruset

Generic Decryption

- polymorfe virus må dekryptere seg selv for å aktiveres
- Virusene kjøres i en et beskyttet virtuelt miljø - en sandkasse (sandboxing)
- Dette kan være ”real time”. Det må dermed gå veldig fort.
- Utfordring: Hvor lange skal viruset få kjøre?

GD-scanner består av:

- CPU-emulator
- signatur-scanner
- kontrollmodul for emulering

12.5 antivirus

- signaturer (må oppdateres)
- Sandboxing
- Generic Decryption

12.6 forsvar mot buffer overflyt-angrep

compile time

programmeringsspråk

- eks.: java bytecode enviroment

kodeteknikker

- OpenBSD

`http://openbsd.org/`

biblioteker/språktillegg

- eks.:libsafe

`http://www.research.avayalabs.com/gcm/usa/en-us/initiatives/all/nsr.htm&Filter=ProjectTitle:Libsafe&Wrapper=LabsProjectDetails&View=LabsProjectDetails`

beskyttelse av stakken

- eks.: Stackguard (inserts canary)

runtime

- noexec-bit i MMU for stack/heap

randomisert minne-plassering

- stack
- standard bibliotek-rutiner

guard pages

- mellom minneområder
- MMU-flag - illegal adresse

12.7 Fysiske tiltak

Plassering

- sett f.eks. ikke en maskinen i kjeller som trues av oversvømmelse

Fysiske beskyttelse

- spenningsvern
- låser, vegger, skap, dører, etc...

Velkjent sårbarhet i vanlige sylinderlåser

- upload.wikimedia.org > Wikipedia > Commons > 6 > 6e > Pin tumbler with key

http://upload.wikimedia.org/wikipedia/commons/6/6e/Pin_tumbler_with_key.svg

- upload.wikimedia.org > Wikipedia > Commons > E > E8 > Newtons cradle animation book

http://upload.wikimedia.org/wikipedia/commons/e/e8/Newtons_cradle_animation_book.gif

- upload.wikimedia.org > Wikipedia > En > 6 > 68 > Snap gun

http://upload.wikimedia.org/wikipedia/en/6/68/Snap_gun.jpg

- youtube.com > Watch ? ...

http://www.youtube.com/watch?v=7xkkS2p7SuQ&feature=fvw

12.8 Redundans

- Redundans styrker systemets robusthet:
- Hindrer avbrudd
- Forbedrer tilgjengeligheten

Eksempler:

- UPS
- Raid
- Speiling
- Sikkerhetskopier

12.9 BIOS

- Oppstartspassord – systemet starter ikke opp igjen etter eventuelt strømbrudd.
- Deaktivere oppstart fra diskett/cd og sette passord på BIOS-oppsett-programmet.
- BIOS kan/kunne resettes ved å ta ut batteriet på hovedkortet eller med en bryter på hovedkortet.

12.10 Sikkerhet på linux

Hva er spesielt med Linux?

- In a World Without Walls and Fences, Who Needs Windows and Gates?
- Åpen kildekode; en sikkerhetsrisiko eller sikkerhetsgaranti?
- Hvorfor finnes det (nesten) ikke linux-virus?

Enbruker-modus

Enbruker-modus i krisesituasjoner - f. eks. ved brudd på sikkerheten:

- init 1 (på skallets kommandolinjen)
- linux single (oppstartslasterens kommandolinje)

Konsollet

- Låsing av display/virtuellt konsoll
- Eksempler:
- xlock/xtrlock - låser X-display. Brukes kun ved bruk av display manager (f.eks. xdm el. kdm)
- vlock låser alle eller noen av de virtuelle konsollene – ubrukelig fra X.

Oppdage fysiske brudd på sikkerhet

- Er systemet startet om - uten at det var du selv som gjorde det: Bruk f.eks. uptime(1) til å finne opptid..
- kontrollere kabinettet
- kontrollerer logg-filer
- logg-filer kan lagres på en sentral syslog-tjener - Det finnes syslog-varianter som sender loggene kryptert men vanligvis sendes dataene ukryptert.

Trusselen fra lokale brukere

- Det er ofte lettere for en inntrenger å få tilgang til en brukerkonto enn en root-konto.
- Når en inntrenger har en brukerkonto, er det lettere for ham/henne å skaffe seg tilgang til root-kontoen.

Beskyttelse mot angrep fra lokale brukere.

Opprettelse av nye brukere

- Gi ikke nye brukere fler rettigheter enn det de trenger.
- Fjern inaktive konti
- Bruk samme brukeridentifikatorer for brukere som har konti på flere maskiner - letter administrasjon og lesing av logger.
- La aldri flere personer dele en konto.

Brukeren root

- Opptre som root, kun når du trenger det.
- Testkjør kompliserte kommandoer før de faktisk utfører sin oppgave.

Vær forsiktig med søkestien (PATH) til root, den bør:

- være kort
- aldri inneholde aktiv katalog (.)
- aldri inneholde kataloger som er skrivbare for brukerne.
- Vær forsiktig med hva som legges i /etc/securetty
- Ikke forhast deg, når du opererer som root
- Vær utrolig forsiktig med hvem du gir root-tilgang
- sudo(8) kan brukes for å gi brukere rettigheter til visse komandoer som root

Instillinger i /etc/fstab

Man kan i /etc/fstab sette restriksjoner på monterte enheter/partisjoner

- nosuid - Forhindrer SUID/SGID -programmer å kjøre
- nodev - Forhinder opprettelse av blokk-enhets-filer
- noexec - Forhindrer programmer å kjøre
- Dersom man f.eks. har en egen partisjon for /home, så kan man med fordel sette nodev og nosuid på denne.

Eksempler:

- Linux Security Howto foreslår nodev og noexec på /var dersom den er på en egen partisjon.
- På fjernmonterte områder (NFS) bør man bruke nodev og nosuid (i noen tilfeller også noexec)

12.11 (Ekstra for spesielt interesserte) Et tenkt tilfelle: Innbrudd i fire akter

1. akt: Rekognosering

Port-scannere

- Tyven forsøker forgjeves å finne åpne dører, ved å portscanne målet.
- Port-scannere er programmer som prøver å koble seg til alle portene på en maskin eller et helt nettverk, og forsøker å bestemme hvilke tjenester som er tilgjengelige. Slike programmer gjør det lettere å finne maskiner som kjører tjenester med kjente svakheter/hull.
- Eksempler på nettverks-scannere:
- nmap(1) og nessus
- Nessus
- Grafisk grensesnitt (GTK)
- Plugin oppsett for port-scannings tester
- Hjemmeside: http://www.nessus.org

`http://www.nessus.org`

Mot-tiltak

Hold systemet ajour

Eksempel: APT (brukt i f.eks. i debian og ubuntu)

- Advanced Packaging Tool

Hjelpemiddel for å:

- laste ned programvare
- håndtere programvare-avhengigheter
- installere/avinstallere
- oppgradere
- holde systemet ajour

/etc/apt/source.list

-

```
# See sources.list(5) for more information, especialy
# Remember that you can only use http, ftp or file URIs
# CDROMs are managed through the apt-cdrom tool.
# Security updates for "stable"
deb http://security.debian.org stable/updates main contrib non-free
deb http://security.debian.org testing/updates main contrib non-free

# Stable
deb http://ftp2.de.debian.org/pub/debian stable main contrib non-free
```

Bruk av APT

- debian.org > Doc > Manuals > Apt-howto

`http://www.debian.org/doc/manuals/apt-howto/`
Eksempel på håndtering av OpenOffice
Søk etter programpakker:

- apt-cache search openoffice

Vis info om en programpakke:

- apt-cache show openoffice.org
- apt-cache showpkg openoffice.org

Installasjon av programpakke:

- apt-get install openoffice.org

Avinstallasjon av programpakke:

- apt-get remove openoffice.org

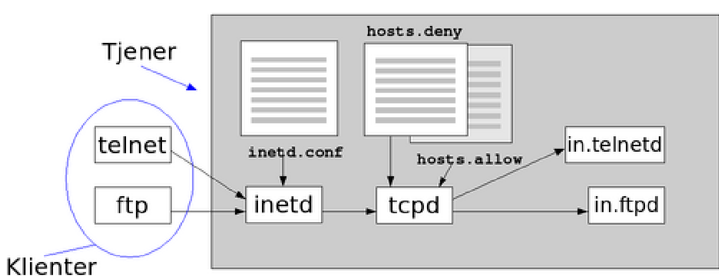
Oppgradering av alle installerte pakker:

- apt-get update
- apt-get upgrade

System-tjenester

- Tyven finner noen åpne porter, og undersøker om programmene som lytter på disse har noen kjente svakheter.
- Jo flere tjenester som kjører (og lytter), jo større sjanse er det for at tyven finner et hull. Kjør derfor ikke flere tjenester på systemet enn det er behov for. Kommandoen under gir en oversikt over tjenester som lytter på porter: netstat -tap
- Kommenter ut de tjenestene du ikke vil tilby, i filen /etc/inetd.conf og start om inetd().
- Fjern programpakker du vet at du ikke har bruk for.
- Sjekk hvilke tjenester som startes ved oppstart.

inetd ogTCP_wrappers figur

- 

- Tidligere benyttet de fleste linux-distribusjoner seg av super-demonen inetd() og TCP_wrappers, tcpd(), for å håndtere forespørsler mot mange av tjenestene.
- Innholdet i konfigurasjonsfilen inetd.conf bestemmer hvilke tjenester super-demonen skal representere, mens filene /etc/host.allow og /etc/hosts.deny hvem som får tilgang. ikke.
- En ny variant extended internet daemon xinetd(), er også i vanlig bruk. Denne kombinerer funksjonaliteten i inetd(), tcpd() og portmapd().

/etc/inted.conf

- navn på tjeneste (/etc/services)
- socket type (stream for TCP, dgram for UDP, raw for IP)
- protokoll (/etc/protocol)
- wait/nowait[.max] (wait for UDP og nowait for TCP)
- bruker[.gruppe] (/etc/passwd)
- stien til tjenerprogram (1. arg til exec())
- argumenter til tjenerprogram (resten av arg. til exec())

Eksempel:

-

```
#ftp streamtcpnowaitroot /usr/sbin/tcpd /usr/sbin/in.ftpdssh streamtcpnowaitroot /usr/sbin
```

Brannmurer

- Brannmurer er typisk installert på maskiner som er koblet til to nettverk, f.eks Internettet og det lokale nettverket. En brannmur konfigureres ved å lage regler for hva som sendes fra det ene nettverket til det andre.
- Selv om man har en brannmur på nettverket, er det viktig å sikre maskinene på innsiden.
- Det er også lurt at hver maskin kjører en brannmur, som kontrollerer inngående og utgående trafikk.
- Fra og med linux-kjerne 2.0, kan brannmur-kode kompileres inn i kjernen.
- Ved hjelp av programmene under kan man, i farta, endre reglene.

Implementasjoner i linux

- ipfwadm (for 2.0 kjerner)
- ipchains (for 2.2 kjerner)
- iptables (fra 2.4 kjerner)
- Er det noe vits i å kjøre en brannmur på maskinen dersom alle unødvendige tjenester er skrudd av?

Å oppdage port-scanning

- Det finnes verktøy for å oppdage port-scannere, men ved bruk av tcp_wrappers og rutinemessig kontroll av logger, vil det oppdages.
- Det finnes dog såkalte "stealth scanners", som ikke setter spor i loggene.
- Hva er vitsen med å oppdage port-scanning?

2. akt: Spionering

Pakke-sniffere

- Tyven har ikke mistet motet, og har tatt seg inn i bygningen hvor målet står. Han har klart å skaffe seg fysisk tilgang på en maskin i et åpent kontorlandskap.
- Han setter ethernet-porten i promiskiøst modus og starter en pakke-sniffer. Pakke-snifferen lytter på ethernetporten etter brukernavn og passord og som lagrer sine funn i en logg.
- Tyven trekker seg tilbake og håper at loggen etter en stund vil inneholde mange loginnavn og passord for konti på flere maskiner på nettverket.
- Tjenester som sender informasjonen ukryptert som sårbare for slik avlytting.
- I mange tilfeller trenger ikke inntrengeren engang bryte seg inn på en maskin i nettet; han kan ta med seg sin egen, og koble den til det lokale nettverket.
- Dersom nettverket er trådløst, kan det holde å ta med pc'en i bilen og parkere utenfor bygningen der det trådløse nettverket er.

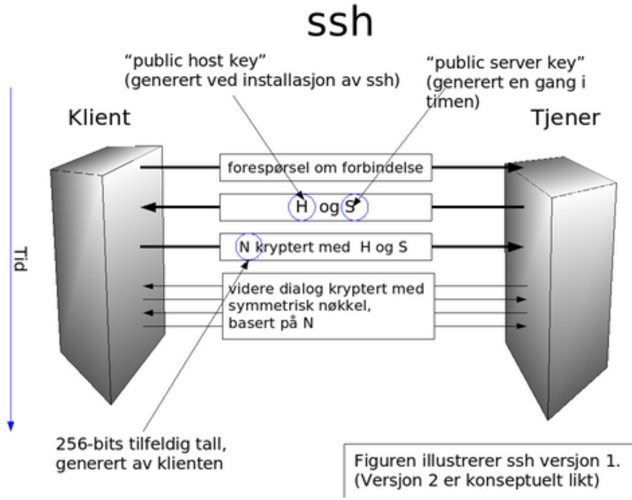
Eksempel på pakkesniffere:

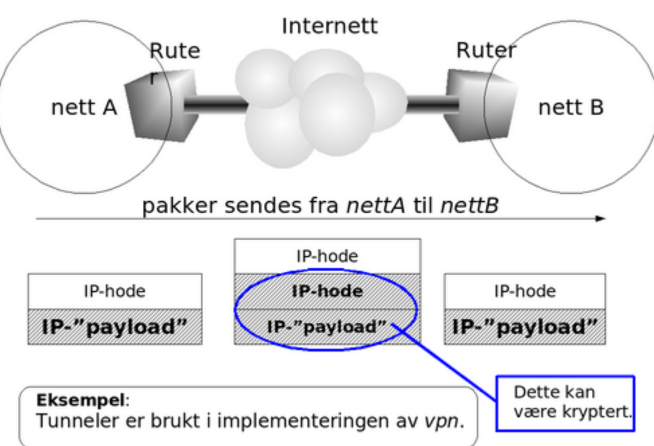
- wireshark(1)
- tcpdump(8)

Mottiltak

- Kryptering av internettforbindelser

ssh

- 

- 

3. akt Inntrengning og utkastelse

Tyven kommer seg inn og blir kastet ut igjen

- Systemadministratoren oppdager inntrengeren og tar kontakt med den rettmessige eieren av kontoen som er opphavet til "uregelmessighetene", v.h.a. et alternativt medium (f.eks. telefon eller personlig oppmøte).

- Systemadministratoren vurderer: Om hun skal koble systemet fra nettverket stenge for tilgang fra stedet brukeren logget seg inn fra v.h.a. tcp_wrappers/pakkefilter.

- Hun deaktiverer brukerens konto/endre passord og dreper eller stopper brukerens prossesser.
- Hun følger nøye med en stund etterpå, for å se om angrepet fortsetter via annen konto og/eller fra et annet sted.

Når innbruddet allerede har hendt

Stenge hullet

- Forsøk å avgjøre hvordan innbruddet har skjedd
- stenge hullet

Reparere det som er skadet

- Forsøk å avgjøre hvilken skade som har skjedd, og reparere det.
- Ofte er det nødvendig å reinstallere all programvare (f.eks. hvis inntrengeren har fått root-tilgang).

Sikkerhetskopier

- Vurder å tilbakeføre sikkerhetskopier for å gjenopprette endrede/skadede filer.
- Kontrollere integriteten til filene på sikkerhetskopien før tilbakeføring.

Oppspore inntrengeren

- Ta kontakt med systemadministratoren på det systemet inntrengeren kom fra.
- Forklar hva som skjedde. Utdrag av logger etc.

4. og siste akt: Hevnen

DoS-angrep

- Tyven tyven blir forbannet og bestemmer seg for å ødelegge, ved å gjøre nett-tjenestene til offeret helt eller tilnærmet ubrukelig. Han vil gjøre det med et såkalt DoS-angrep (Denial of Service) .
- For å gjøre det vanskelig å stanse angrepet, vil han forsøke seg på et distribuert DoS-angrep. D.v.s. at angrepet kommer fra mange maskiner samtidig. Dette realiseres f.eks. med tids-innstilte eller fjernstyrte trojanske hester.

12.12 Oppgaver

Eksempler på kommandoer som kan brukes i oppgave 1 og 2:

Opprette en fil med navn og passord for brukeren arne:

- htpasswd -c passordfil arne

Legge til navn og passord i en eksisterende fil:

- htpasswd passordfil beate

Endre passord til navn som allerede eksisterer i fil:

- htpasswd passordfil arne

Teste alle passord i en passordfil:

- /usr/sbin/john passordfil

Starte virtuell maskin:

- vstart virtuellmaskin

13.1

- a) Opprett en fil med noen krypterte passord v.h.a. programmet htpasswd(1). Lag noen passord du tror er enkle å gjette, og noen vanskelige.

b)

- Forsøk så om programmet john(8), klarer å knekke passordene.
- Programmet john ligger i katalogen /usr/sbin, som normalt bare ligger i PATH-variablen til 'root'. For å kjøre programmet må dere derfor enten legge til /usr/sbin i PATH variabelen eller oppgi full sti til programmet.

13.2

- Start en virtuell maskin på debbie. Opprett noen brukere med kommandoen 'adduser'. La brukerne få passord av ulik kompleksitet - noen enkle og noen kompliserte. Du skal nå forsøke å finne passordene til disse brukerne.
- John er ikke installert på den virtuelle maskinen, så filen med de passord-hashene (/etc/shadow) må kopieres til din hjemmekatalog på debbie (/host-home), og kjøre john på debbie.
- Tips: bruk opsjonen –format=md5 til programmet john(8).

13.3

a) Lag et C-program som:

- leser et passord fra STDIN
- Velger salt-verdi
- genererer passord-hash
- lagrer hashen og saltet i en fil
- tips: bruk crypt(3)

b) Lag et C-program som

- leser passord fra STDIN
- henter salt fra filen i a)
- genererer passord-hash
- sammenligner generert hash
- med hashen lagret i fila

13.4

- Finn ut hvilke porter som er åpne på matiksi.hive.no og debbie.hive.no. Vær forsiktig med portscanning. Det vil av noen oppfattes som fiendtlig aktivitet. Tips: nmap(1).

13.5

- I denne oppgaven skal du bli root på labmaskin på C258 uten å bruke rootpassordet
- a) Start maskinen og velg med piltastene linjen hvor det står "CentOS"
- b) Trykk 'a' for å redigere argumentene til kjernen. Legg til argumentet '1' eller 'single' og trykk enter.
- c) Skriv 'whoami' når du er ferdig med å "boote"
- d) Hva kunne vi gjort for å forhindre at noe slikt kunne skjedd?

13.6

- Ekstraoppgave: Forsøk å sette opp et software-raid på en virtuell maskin (eller lab-maskin)

Kapittel 13

Løsningsforslag på eksamen høst 2008

1.

Forklar hvordan DMA (Direct Memory Access) bidrar til å øke graden av utnyttelse i et datasystem.

Svar:

Når I/O-enheter kan lese og skrive til minne uten å involvere CPU'en, vil CPU'en kunne jobbe parallelt med I/O-enhetene.

2.

a)

Hva ligger i begrepet kjøretidsdeling (Time sharing)?

Svar:

Flere brukere (eller prosesser/tråder) kan bruke datasystemet samtidig – kjøretiden fordeles mellom brukerne (prosessene/trådene).

b)

Hvordan realiseres det?

Svar:

Jobb (el. prosess/tråd) blir avbrutt ved faste tidsintervaller og lagt i kø. Ny jobb velges ut av køen. Intervallet (og køen) er, idéelt sett, så kort at brukerne ikke oppfatter denne vekslingen.

c)

Beskriv bakgrunnen for at denne teknikken ble utviklet.

Svar:

P.g.a. høye maskinvare-kostnader, måtte mange dele samme datasystem. Brukerne ønsket å interagere direkte med maskinen (ikke via operatør).

3.

a)

Hva menes med et 'shell' i UNIX/Linux?

Svar:

Brukergrensesnitt mot datasystemet. Vanligvis kommandolinjebaserte (Command Line Interface - CLI). Program som leser, tolker og utfører kommandoer.

b)

Nevn et slikt 'shell'.

Svar:

bash (Bourne Again Shell)

c)

Hva menes med et 'shell script'?

Svar:

En fil som er ment å leses av et shell Hver linje i fila leses, tolkes og utføres av shellet.

d)

Skriv et 'shell script' som - lager en katalog - kopierer en fil fra foreldrekatalogen (parent directory) inn i den nyopprettede katalogen

Du kan selv bestemme hva filen og katalogen skal hete.

Svar:

```
1 #/bin/sh
2 mkdir katalog
3 cp ../fil katalog/
```

e)

Hva må til for å kjøre skriptet i d)?

Svar:

Jeg antar at filen heter d.sh.

bash d.sh

eller

bash < d.sh

eller

chmod +x d.sh

./d.s

4.

a)

Hva er en 'prosess'?

En enhet av aktivitet i et datasystem karakterisert av

- en eller flere sekvensielle tråder av utførelse

- en tilstand
- tilhørende system resursser

b)

Hva er en 'tråd'?

Sekvensiell utførelse av instruksjoner med tilhørende minne og tilstand i tillegg til minnet og tilstanden den deler med resten av prosessen den tilhører.

c)

Skriv et C-program som starter en annen prosess. Den nye prosessen skal kjøre et annet program. Du kan selv bestemme hvilket program som skal startes.

Svar:

```
1 if (fork() == 0)
2     execlp(``ls'',''ls'',NULL);
```

5.

a)

Hva er Symmetrisk Multiprocessing? Tegn og forklar.

I et SMP-system finnes flere prosessorer som deler minne, buss og I/O-enheter. Prosesser/tråder kan kjøres på en hvilken som helst av prosessorenene. Hver prosessor selv-skedulerer.

b)

Hva er et Cluster? Tegn og forklar.

Selvstendige datasystemer (noder) som fungerer som et datasystem. Hver node har sitt eget sett med CPU'er, minne, busser og I/O-enheter. Nodene kommuniserer f.eks. ved hjelp av standard nettverkskomponenter.

c)

Sammenlign systemene i a) og b).

Svar:

Cluster skalererer veldig bra i motsetning til SMP. SMP er mer moden teknologi.

SMP: raskere kommuniksajon mellom prosesser/tråder, enn på ulike noder i et cluster, siden kan dele minne (og buss).

6.

Hva menes med 'concurrency'?

Svar:

Prosesser/tråder veksler på å kjøre eller kjører helt/delvis parallelt, innenfor et gitt tidsintervall. Det er uforutsigbart når et bytte skjer, slik at selv om rekkefølgen av instruksjonenene i de konkurerrende prosessene hver for seg er forutsigbar, vil rekkefølgen på instruksjonene til alle prosessene *sett som ett system* ikke være forutsigbar.

7.

For å kunne håndtere 'concurrency', må vi kunne håndheve gjensidig utelukkelse (mutual exclution). Innføring av gjensidig utelukkelse medfører to nye problemstillinger.

a)

Nevn disse problemstillingene.

Svar:

- Vranglås (deadlock)
- Utsulting (Starvation)

b)

Beskriv en av disse problemstillingene.

Svar:

Ved utsulting vil noen prosesser bytte på å bruke en resurss, på en slik måte at en eller flere prosesser som trenger tilgang til resurssen, aldri får tilgang. At en prosess aldri får tilgang kalles utsulting.

c)

Gi et eksempel på problemstillingen du valgte i b).

Svar:

Prosess 01 og prosess 02 bytter på å ha ekslusiv tilgang til ressurs R. En prosess 003 har lavere prioritetet, trenger også tilgang. Den er blokkert mens den venter på tilgang, men vil aldri få tilgang fordi den har lavere prioritet i fordelingsmekanismen (scheduler).

8.

a)

Beskriv produsent/konsument-problemet.

Svar:

Dette er et synkroniseringsproblem som oppstår når en produsent og en konsument ikke produserer og konsumerer i samme takt.

b)

Skisser en løsning på problemet i a)

Svar:

En løsning på problemet er å bruke en kø, slik at produsenten blokkeres når køen er full, og konsumenten blokkeres når køen er tom.

9.

a)

Hva menes med sideveksling (paging)?

Svar:

Minnet deles rammer på lik størrelse. Prosessenes minneområdet deles i sider av samme størrelse. Operativsystemet kan flytte (veksle) sidene mellom primærminnet og sekundærminnet etter behov.

b)

Hva menes med virtuelt minne?

Svar:

Et adresserbart minneormåde som (vanligvis) er større enn det fysiske minnet.

Implementert med en mekanisme som dynamisk frigjør plass i primærminnet ved å flytte data til sekundærminnet ved behov. Mekanismen henter også dataene tilbake til primær-minnet når de blir adressert.

c)

Forklar hvordan sideveksling brukes i forbindelse virtuelt minne.

Svar:

Prosessene er delt i sider. Alle sidene til en prosess trenger ikke være i minnet samtidig. Dette gjør at vi kan ha flere prosesser i minnet samtidig som er klare til å kjøre (og utnytte maskinen bedre). Prosessene kan også være større enn det fysiske minnet. Når sideveksling brukes i forbindelse med virtuelt minne, vil systemet flytte en side av gangen mellom primærminnet og sekundærminnet. Når en minnecelle adresseres med en virtuell adresse som befinner seg i en side som ikke er i primærminnet, skjer en såkalt “page fault”. Den adresserende prosessen blokkeres mens siden bringes inn til primærminnet.

10.

a)

Hva menes med autentisering?

Svar:

Kan beskrives som en to-trinns-prosess:

- Identifisering: Brukeren hevder en identitet.
- Verifisering: Systemet utfører en kontrollrutine for å avgjøre om brukeren kan anses å tilsvare oppgitt identitet.

b)

Beskriv hvordan autentisering tradisjonelt gjøres i UNIX/Linux.

Svar:

Brukernavn og passord-hash produserers og lagres i en tabularisk fil (/etc/shadow el./etc/passwd), sammen med en tilfeldig valgt verdi kalt 'salt' som brukes i hashfunksjonen. Filen inneholder altså linjer med (brukernavn, salt, hashverdi)

Ved autentisering identifiserer brukeren seg med et brukernavn. Systemet søker opp brukernavnet i fila, og henter ut tilhørende salt og hash (dersom brukernavnet finnes i fila).

For å verifisere identiteten oppgir brukeren et passord. Dette passordet brukes sammen med saltet fra fila for å produsere en ny hash-verdi. Denne hashverdien vil være identisk med den som ble hentet fra fila, dersom passordet er korrekt.

Lykke til :)

Kapittel 14

Løsningsforslag på eksamen høst 2009

1.(10%)

a)

Beskriv kort hva som menes med *avbrudd (interrupt)* i forbindelse med datasystemer.

Svar

Prosessor kan avbrytes med elektrisk signal slik at neste instruksjon som utføres fra en forhåndsbestemt minneadresse knyttet til signalet.

b)

Forklar kort hvordan avbrudd kan bidra til å øke graden av utnyttelse i et datasystem.

Svar

Siden I/O-enehetene selv kan "si i fra" v.h.a. avbrudd-mekanismen når I/O-operasjonen er ferdig, kan maskinen jobbe med noe annet istedenfor å ikke gjøre noe mens den venter på I/O-enheten. Mindre dødtid gir mer arbeid pr. tidsenhet og dermed høyere utnyttelsesgrad.

Svar

2. (15%)

a)

Beskriv kort hva som menes med kjøretidsdeling (time sharing)?

Svar

Systemet deler tiden inn i faste tidsintervaller, og rullerer på kjøring av av brukere/prosesser/jobber/tråder

b)

Hvordan skapes kjøretidsdeling?

Svar

En klokke sender et avbrudd-signal Ved utløp av hvert tidsintervall. Den kjørende bruker/jobb/prosess/tråd blir blokkert og lagt i kø. En ny bruker/jobb/-prosess/tråd blir valgt til å kjøre på prosessoren.

c)

Nevn minst en gevinst/fordel og en kostnad/ulempe ved å innføre kjøretidsdeling i et system.

Svar

Fordel: Lavere responstid.

Ulempe: Lavere utnyttelsesgrad i forhold til batch-systemer med multiprosessing (p.g.a. mer tid til veklsing mellom brukere/jobber/prosesser/tråder).

3. (15%)

a)

Beskriv kort hva som menes med kappløp (race condition)?

Svar

To eller flere tråder/prosesser leser/skriver til samme resurss i løpet av samme tidsrom (konkurrerende/samtidig), på en slik måte at utfallet av kjøringen blir uforutsigbar.

b)

Hvordan kan en slik situasjon oppstå.

Svar

Dette kan oppstå når vekslingen mellom kjøring av trådene/prosessene er avhengig av hendelser som ikke er forutsigbare.

c)

Gi et eksempel og skisser en prinsipiell løsning.

Svar

I et system med kjøretidsdeling har trådene A og B begge tilgang variabelen C. Både A og B skal utføre: C=compute(C).

Verdien av C vil dermed avhenge om når vekslingen skjer. Dersom en av prosessene blir avbrutt av klokka etter at *compute()* har lest verdien av C, men før beregningene ferdige. Dette er f.eks. avhengig av hvor mange andre prosesser/tråder som til en hver tid kjører på systemet.

Dette kan håndteres med mekanismer for eksklusiv tilgang (gjensidig utelukkelse) til den delte resurssen, som i eksemplet her er representert med variablen C, slik at trådene kan fullføre operasjoner på delte resursser uten at andre tråder endrer resurssenes tilstand.

4. (10%)

Under ser du innholdet i en fil.

```
#!/bin/sh
echo a > b
mkdir c
cd c
cp ../b .
```

a)

Hva slags fil er dette? Gi en kort begrunnelse.

Svar

Dette er et shell-skript fordi den første som gir stien till shellet og resten er kommandoer som kan gis til et shell.

b)

Forklar kort hensikten med hver enkelt linje i fila.

Svar

1. Stien til programmet som skal tolke linjene under.
2. Åpne filen b og skriv tegnet a i filen (evt. gammelt filinnhold forsvinner).
3. Lag katalogen c som en underkatalog til arbeidskatalogen.
4. Endre arbeidskatalogen til den nyopprettede katalogen.
5. Kopier filen b fra foreldrekatalogen til arbeidskatalogen.

5. (20%)

Et dataprogram inneholder bl.a. følgende kode:

```
#include <stdlib.h>
#define N 4

void compute(int a){

  // gjoer noen beregninger her
}

void f1(){

  int i;

  for (i=0; i<N; i++){

    if (fork()==0 ){
      compute(i);
      exit(0); // terminerer prosessen
    }
  }

  for (i=0; i<N; i++)
```

```
22        wait(NULL);
23  }
24
25  void f2(){
26
27     int i;
28     for (i=0; i<N; i++){
29
30        if (fork()==0){
31
32           compute(i);
33           exit(0); // terminerer prosessen
34        }
35
36        wait(NULL);
37     }
38  }
39
40  int main(){
41
42     f1();
43     f2();
44     return 0;
45  }
```

Anta at koden kjører på en system med *symetrisk multiprosessering* (SMP). Operativsystemet har *4 prosessorer* tilgjengelige.

a)

Gi en kort overordnet forklaring på hva koden gjør.

Svar

Mainfunksjonen starter to funksjoner *f1()* og *f2()*. Begge disse starter fire nye prosesser. Hver av de nye prosessene kjører funksjonen *compute()*. Deretter terminerer de.

f1()

Funksjonen *f1()* starter fire prosesser. Når alle fire er startet, venter den til alle er terminert før den returnerer.

f2()

Funksjonen *f2()* starter fire prosesser. Etter hver oppstart av en prosess, venter til den nyoppstartede prosessen er terminert før neste prosess startes.

b)

Vil kjøretiden til f1() bli kortere, lengre eller like lang dersom koden hadde kjørt på et system med kun en enkel prosessor. Gi en kort begrunnelse for svaret.

Svar

Kjøretiden blir sannsynligvis blir lengre fordi de fire prosessene ikke lenger har anledning til å kjøre parallellt.

c)

Vil kjøretiden til f2() bli kortere, lengre eller like lang dersom koden hadde kjørt på et system med kun en enkel prosessor. Gi en kort begrunnelse for svaret.

Svar

Prosessene som startes i *f2* vil aldri kjøres parallell, siden den ny prosess ikke startes før den pågående er fullført. Antall prosessorer vil dermed ikke påvirke funksjonens kjøretid. D.v.s. at kjøretiden blir tilnærmet lik.

d)

Vil kjøretiden til main() bli kortere, lengre eller like lang dersom koden hadde kjørt på et system med *8 prosessorer*. Begrunn svaret.

Svar

Siden *f2()* starter etter at *f1()* og dens barneprosesser, er fullført, og den opprinnelige prosessen ikke gjør stort annet enn å vente, vil det kun være fire prosesser som kan jobbe parallellt selv om antall prosessorer skulle øke. Det vil derfor ikke gi noen kortere kjøretid selv om antall prosessorer øker.

6. (20%)

a)

Hva menes med *vranglås* (dead lock)?

Svar

Med vranglås i det datasystem menes at to eller flere prosesser/tråder/jobber aldri kommer videre i sin programkjøring på grunn av at at de venter på hverandre i det som kalles *sirkulær venting*.

b)

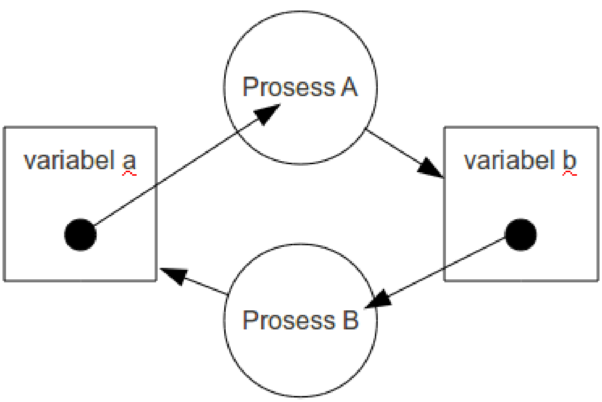
Gi et praktisk eksempel på en vranglås i et datasystem.

Prosess A har en lås på variablene a og venter på tilgang på variablen b. Prosess B har en lås på variablene b og venter på tilgang på variablen a.

c)

Beskriv eksemplet du ga i b) som en *resurss-allokerings-graf* (Resource Allocation Graph)

Svar



d)

Foreslå kort hvordan hvordan vranglåsen i b) kunne vært unngått.

Svar

A og *B* kunne endres slik at de sørget for tilgang til begge variablene i samme rekkefølge, før de begynte å jobbe med dem.

7. (10%)

a)

Gi en kort beskrivelse av og et enkelt eksempel på av hva produsent/konsument-problemet går ut på.

Svar

Problemet går ut på å få til synkronisering i en situasjon hvor produksjon ikke skjer i samme takt som konsum.

b)

Beskriv en måte dette løses på i praksis. Bruk gjerne eksempel i C-kode eller shell-skript.

Svar

Dette kan løses ved at det som produserers plasseres i en kø, og det som skal konsumeres hentes ut av køen. Når køen er tom, må konsumenten vente. Når

køen er full må produsenten vente. I kommandolinjen `cat /etc/passwd | grep Hansen` vil *cat* være produsenten og *grep* være konsumenten. Produsentens utskrift sendes inn i en *pipe*, som er implementert ved hjelp av en kø. Konsumenten leser fra køen.