Generelt om operativsystemer

Hva er problemet?

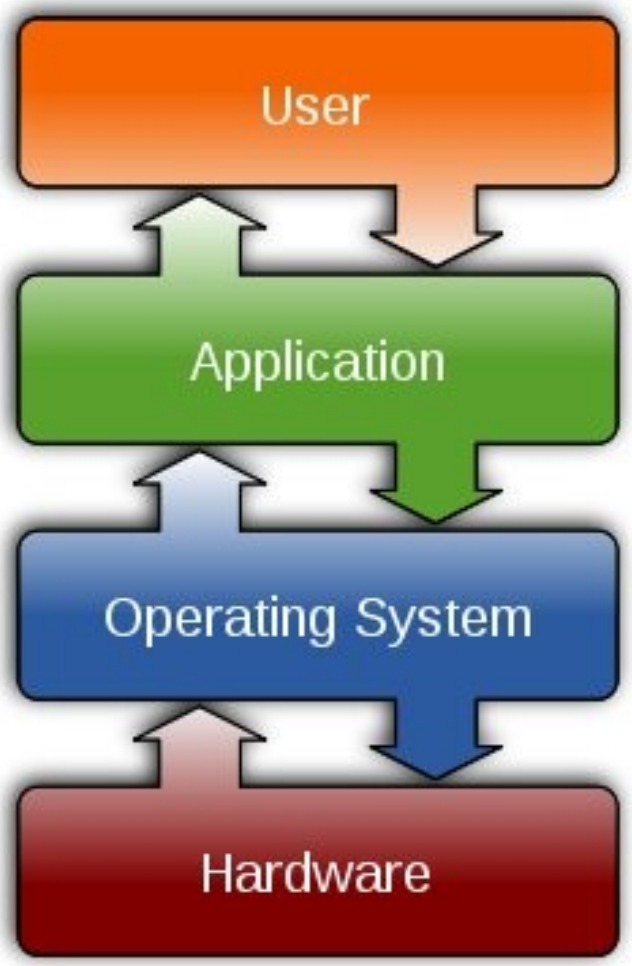
- Styring av maskinvare og ressurser tilknyttet en datamaskin er komplisert, detaljert og vanskelig
- Maskinvare, komponenter og programvare endres og forbedres raskt og ofte
- Vi må ha et system som håndterer dette for oss, slik at maskinen blir enklere å bruke og alltid er oppdatert og fungerer korrekt

Løsning: Et operativsystem (OS)

- Et OS er en samling med *programvare* som håndterer styringen av datamaskinen
- Forenkler bruken av datamaskinen for både brukere og applikasjoner
- OS er oftest ferdig installert på datamaskinen
- Lett å oppdatere manuelt eller automatisk

Operativsystemet gir en lagdeling

- OS er et mellomlag som skjuler kompleksiteten og dynamikken i maskinen
- OS tilbyr grensesnitt mot brukere og applikasjoner
- Figuren er en forenkling, men viser prinsippet for bruken av OS



Hovedoppgavene til et OS

- *Styring* av maskinvare
- *Deling* av maskinens ressurser
- *Abstraksjon* vekk fra detaljer om maskinvare og systemprogramvare

Designprinsipp for OS

- All maskinvare, komponenter og tilknyttede ressurser håndteres *internt* i OS-et
- Brukere tilbys et grensesnitt (grafisk eller tegnbasert) mot maskinen i form av *kommandoer* som kan utføres
- Applikasjoner tilbys grensesnitt mot datamaskinen i form av et API (Application Programming Interface) med *metoder* som kan kalles fra applikasjonens programkode

De viktigste deloppgavene til et OS

- Håndtering av prosesser
- Håndtering av minnet (RAM)
- Filhåndtering
- Håndtering av nettverk, komponenter og utstyr
- Sikkerhets- og feilhåndtering
- Interprosesskommunikasjon

Prosesshåndtering*

- Prosess:
  - Program som kjører/eksekverer
- OS må kunne håndtere mange prosesser “samtidig”, inkludert interaktiv dialog med brukere
- Håndteringen av prosesser må være rask, effektiv og “rettferdig”

* Kapittel 4 i læreboken

Noen deloppgaver i prosesshåndtering

- Opprette og fjerne prosesser
- Starte og stoppe prosesser
- Synkronisere prosesser
- Fordele mikroprosessorenes (CPU’enes) regnekraft mellom de aktive prosessene
- Prioritere prosesser som ønskes utført raskere/ mer effektivt enn andre

Minnehåndtering*

- Minne, aka:
  - RAM
  - Hukommelse
  - Hurtiglager
- Brukes for lagring av *både* data og programkode
- Minne er alltid en *begrenset* ressurs
- Må *deles* mellom alle prosessene på maskinen

* Lite om dette i læreboken, legger ut ekstra lærestoff på kursets nettsider

Noen krav til minnehåndtering i et OS

- God fordeling av minnet mellom prosessene, når alle prosesser samlet krever mer minne enn tilgjengelig
- Fungere både statisk (ved oppstart av prosess) og dynamisk (mens prosessen kjører)
- Beskytte den delen av minnet som er reservert for én prosess mot lesing og endring fra andre prosesser
- Dele minne mellom prosesser hvis dette er ønskelig
- Legge lite brukte minneområder midlertidig ut på disk for å frigjøre plass til andre prosesser

Utstyrshåndtering*

- OS leveres med *drivere*:
  - Programkode for å håndtere “faste komponenter” (CPU, RAM...) og “plug-ins” (grafikk, nettverk, printere...)
- OS tilbyr et Service Provider Interface (SPI):
  - Håndterer nytt/ukjent utstyr ved å installere medfølgende *drivere* i OS-et
- OS tilbyr applikasjonene et API for utstyrshåndtering:
  - Enhetlig grensesnitt der detaljer i driverne skjules

* Litt mer om utstyrshåndtering kommer senere i kurset

Noen krav til et godt konstruert OS

- Effektivitet
- Vedlikehold
- Korrekthet
- Standardisering
- Grensenitt og abstraksjon

Effektivitet

- Et OS gjør *ikke* noe produktivt arbeid i seg selv.
- OS må utføre oppgavene sine effektivt, slik at det meste av maskinens ressurser brukes til *applikasjonene*
- Alle OS er *optimaliserte* mht. effektivitet
- OS inneholder derfor mange svært kompliserte og meget raske algoritmer for håndtering av datamaskinen, f.eks.
  - Scheduling: Deling av noen få CPU'er på mange prosesser
  - Paging: Flytting av data og kode mellom RAM og disk

Vedlikehold

- Et OS er et meget stort programsystem:
  - Ca. 50 millioner kodelinjer i Windows 7 (“guesstimate”)
  - Ca. 15 millioner linjer fordelt på 35 000 filer i Linux-*kjernen*
- Stort antall utviklere involvert
  - Koordineringskostnader og -tid må minimeres
- Et OS må derfor konstrueres *modulært*:
  - Nøvendig med en god og logisk oppdeling i moduler
  - Moduler bør kunne vedlikeholdes *uavhengig* av hverandre

Korrekthet

- Vi *må* kunne stole på at OS-et fungerer korrekt:
  - Feil i OS kan gjøre maskinen “ubrukelig”
  - Må kunne håndtere tilfeldighet og uforutsigbarhet
  - Må ha gode mekanismer for feilhåndtering
- Krever:
  - Solid teoretisk fundament for implementasjonen
  - God modularisering internt i koden
  - Godt og sikkert grensesnitt mot applikasjonene
  - Gode og omfattende rutiner for testing og feilretting

Standardisering

- Tjenestene som et OS tilbyr *må*(?) følge standarder for samhandling med annen programvare *
- OS-designere må velge gode standardløsninger for bl.a:
  - Hvilke tjenester skal tilbys og hvordan skal de brukes?
  - Implementasjon av tjenestene i høynivå API for f.eks. Java
  - Hvordan skal filene på maskinen organiseres?
  - Hvilke dataformater (tegnsett og tall) skal støttes?
  - Hvordan skal eksekverbare programmer lagres?

* Historisk har Linux vært gode på standardisering, mens Microsoft har vært mer proprietært

Grensesnitt og abstraksjon

- OS må tilby et godt grensesnitt, både mot brukere og utviklere av applikasjoner, som:
  - Følger kjente og vanlige standarder
  - Er stabilt og korrekt
  - Skjuler den interne virkemåte av OS-et
  - Ivaretar kompabilitet selv om maskinvare og systemprogrammer/drivere endres

Grensesnitt og abstraksjon i OS: Analogi til OOP og Java-objekter

- Grensesnittet mot et OS kan sammenlignes med bruk av klasser i objektorientert programmering:
  - Klasseobjekter i Java tilbyr et grensesnitt i form av `public` metoder som kan kalles “fra utsiden”
  - Dataene som lagres i klassen, og selve implementasjonen av metodene, inkapsles/skjules i den interne Java-koden for klasseobjektet
  - Klasseobjektet defineres for brukerne med et grensensnitt som beskriver *funksjonaliteten*

Brukergrensesnitt i OS: Shell vs. GUI

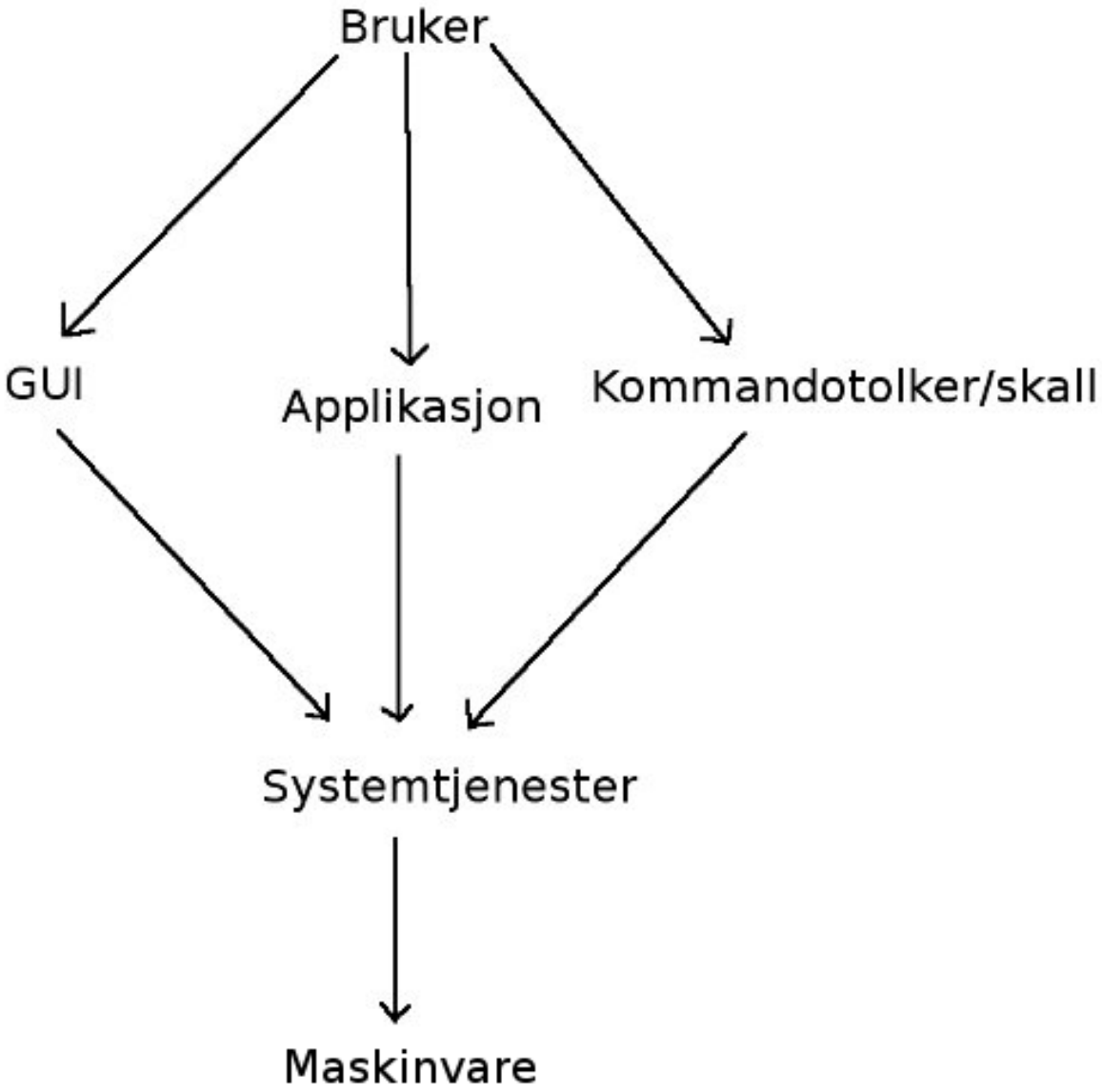
- Shell (aka kommandotolker eller skall):
  - Tekstbasert linje-for-linje som “i gamle dager”
  - Krever mer av bruker, mindre av maskin/nettverk
  - Er en egen programmeringsomgivelse
  - Effektivt til håndtering av store datamengder
  - Uunnværlig til systemadministrasjon
- GUI (Graphical User Interface):
  - Enklere å forstå, krever mye mer ressurser enn shell
  - Visualisering/grafisk fremstilling av filer og systemtilstand
  - Peke-klikke-dille-mikke databehandling, lite effektivt

OS og systemtjenester

- *Systemtjenestene* er tjenester som OS-et kan utføre på kommando, f.eks å:
  - Skrive data til disk
  - Reservere en del av minnet
- Systemtjenester utføres av OS-et når en applikasjon gjør et *systemkall* ved å kalle en metode fra OS-ets API
- Eksempel: Når metoden `write` kalles, vil den utføre en systemtjeneste som skriver data til en fil. OS-et sørger for at dette gjøres på riktig måte uten at applikasjonen trenger å kjenne detaljene om hardware og drivere.

Kjernen i et OS

- Alle systemtjenestene som tilbys til applikasjoner er samlet i det som kalles OS-ets *kjerne* (kernel)
- Kjernen oppdateres oftest automatisk ved behov (ny maskinvare, forbedringer av OS-algoritmer etc.)
- Kjernen er *tilpasset* maskinvaren
- Kjernen til et OS er oftest skrevet i C eller direkte i maskinkode for effektivitet


Bruker
GUI
Applikasjon
Kommandotolker/skall
Systemtjenester
Maskinvare