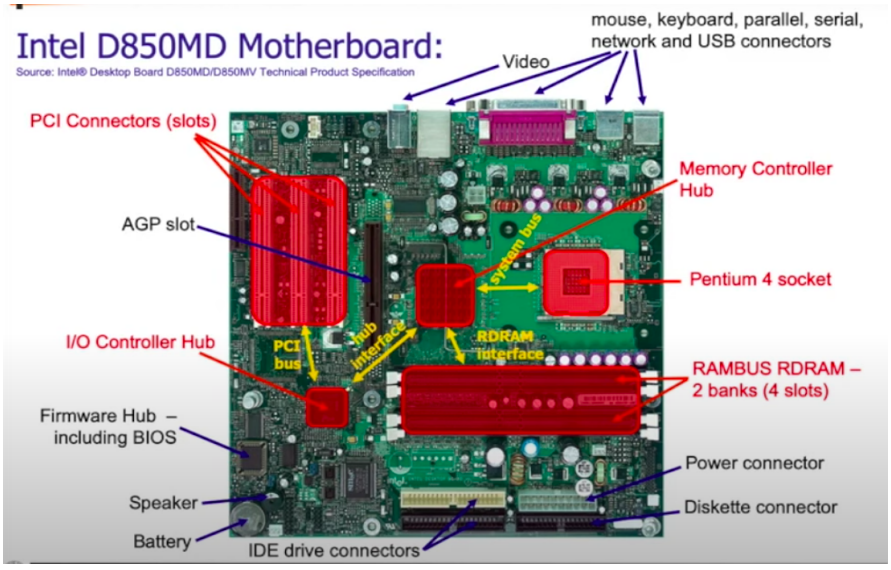
Hvordan kjøres programmer innen et OS?

Hardware

Vi har diverse hardware-deler som omfatter følgende:

- **CPU** som er ansvarlig for å knytte sammen de fysiske delene i maskinen og utføre instrukser i maskinkode.
- **Minneenheter** som oppbevarer instrukser som skal utføres i CPU´en(Cache(s), RAM, ROM, Flash)
- **Input/Output** enheter som igangsetter diverse instrukser(Nettverkskort, disker, cd, tastatur, musepeker).
- **Enheter** som knytter sammen minneenheter og I/O-enheter sammen(interconnects, busses)

Eksempel av et Intel D850MD Motherboard som viser hvordan de overnevnte hardware-komponentene er knyttet sammen:



Vi ser hvordan “busser” og “minneenheter” knytter sammen diverse komponenter. Blant annet er det “System bus” som knytter “Memory Controller Hub” og “CPU(Pentium 4 socket).

Et typisk eksempel på hvordan instrukser utføres i et Intel 32-bit arkitektur:

Adresseområde

- Vi har et adresseområder med 64 gb hvor hver prosess har et adresseområde på 4 gb.

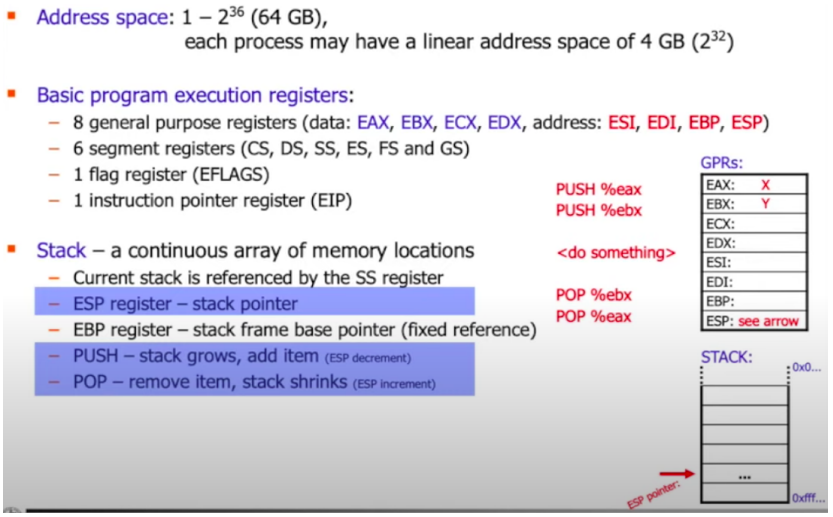
Programmerings-eksekveringsregistere:

- Informasjonen under er kun ment for å vise hvilke registere som er ansvarlig for adresseminne og data.
- 8 generelle registere(Data: EAX, EBX, ECX, EDX, Adresse: ESI, EDI, EBP, ESP)
- 6 segment registere (CS, DS, SS, ES, FS og GS)

- 1 flag register (EFLAGS)
- 1 instruction pointer register (EIP)

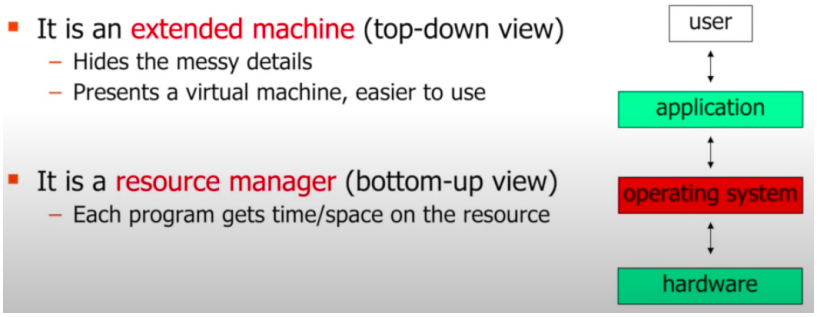
Stack

- Vi har også en stack som er det samme som en array hvor hver element i arrayet tilsvarer minnelokasjoner
- Vi har også diverse metoder i stacken som PUSH(legge til i minne) og POP(fjerne fra minne)



Hva er et OS?

Et operativsystem er en samling programmer/verktøy/funksjoner som opptrer som et mellomlag mellom hardware og brukerne/applikasjonene. Dvs at man skal tilby et høyt nivå interface til lavt nivå hardware enheter som CPU, minne og I/O enheter. Samtidig tilby tjenester og fasiliteter slik at brukere(oss) kan benytte av disse hardware enhetene på en effektiv, hensiktsmessig og trygg måte.



- **Extended Machine**

  Operativsystemet sørger for at ressursene som benyttes av de ulike hardware-enhetene, blir utnyttet effektivt. Dette fører til at vi unngår tilfeller hvor å utføre en gitt instruks i minne, blir utrolig komplisert ettersom OS forenkler arbeidet.
- **Resource Manager**

  Vi vet at OS har utrolig mye å gjøre ettersom det er mange hardware komponenter som utfører en gitt prosess i en gitt tidsramme. Dermed kan vi se på OS som at den har oversikt over alle ressursene.

Operativsystem kategorier

- Single-user, single-task:
  - Sjeldent brukt, hensikten er at OS kan kun utføre en gitt tjeneste for en gitt bruker, hvor bruker tilsvarer en pc eller mobil
- Single-user, multi-tasking:
  - OS er lagd slik at en gitt bruker(En pc eller smarttelefon) utfører flere tjenester.
- Multi-user, multi-tasking:
  - Dette er knyttet mot servere som i UIO som tillater mange brukere til å utføre flere tjenester.
- Distribuert OS:
  - Hvis ressursene er fordelt i flere maskiner, så må OS´et være bygd slik at det kan håndtere diverse ressurser som er distribuert forskjellig.

Hvorfor skjønne OS?

En meget god grunn til å skjønne OS, er at det hjelper med å forstå de tekniske begrensningene som finnes i hardware-komponentene. For da kan ting bli riktig og effektivt, for la oss foreslå følgende scenario:

La oss si at en virksomhet har et system de har lyst til å lansere men den virket ikke som den skulle. Dette kan skyldes av at man ikke forstår hvordan OS er satt opp for som nevnt tidligere så har OS som ansvar å fordele ressursbruken for gitte hardware-komponenter. Hvis man ikke forstår hvordan ressursbruken fordeles så kan det ende opp med at systemet eller funksjonen ikke gikk som planlagt.

OS komponenter og tjenester

Vi har diverse komponenter i OS som er ansvarlige for å utføre de tjenestene som OS er ansvarlig for.

Synlige til bruker

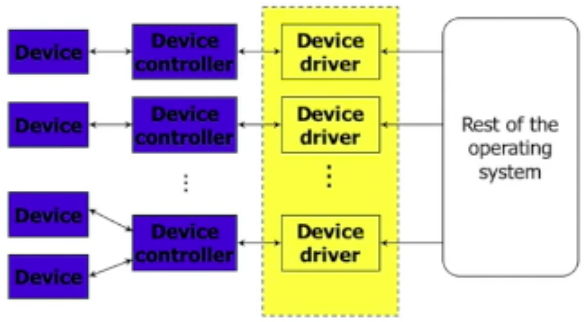
- Bruker interface(Shell)
    - Gjør at bruker kan interagere med OS og benytte av maskinens ressurser
- Fil system
    - Et system for bruker slik at bruker kan lage, slette, modifisere eller manipulere filer
- Enhet håndtering/Device management
    - Kontrollere systemets periferenheter som tastatur, disk, mus osv.

De som ikke er synlig

- Prosessor håndtering/Processor management
    - Tilføre en mekanisme for systemet slik at man kan utnytte prossene som kjøres i maskinen mer effektivt og kontrollerbart i forhold til maskinens begrensning når det kommer til CPU og klokkefrekvens.
- Minnehåndtering/Memory management
    - Det å håndtere minne godt, som vil si at vi skal til enhver tid ha plass til å kunne sette av til instrukser av maskiner i prosesser.
- Communication services/Kommunikasjonstjenester
    - Mekanisme for å vite hvordan de ulike hardware-komponentene skal samhandle f.eks hvordan brukerkomponenter og de lav-nivå komponentene skal kommunisere med hverandre.

“Device Management”

OS må være i stand til å kontrolle periferenheter som disks, tastatur, nettverkskort, skjerm, høyttaler, musepeker osv. Det er veldig mange enheter som OS´et må håndtere og derfor så har vi gitte prinsipper vi følger for å håndtere disse enhetene. Slikt kan vi se hvordan periferenhetene er satt opp i OS:



Håndtering av disse enhetene kan foretas på to måter gjennom registre som kontrollerer disse enhetene:

Port I/O

- Spesial instrukser som snakker med minne til en gitt enhet

Memory mapped I/O

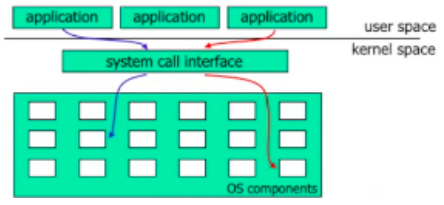
- Registeret som blir mappet inn i et minne

Device driver

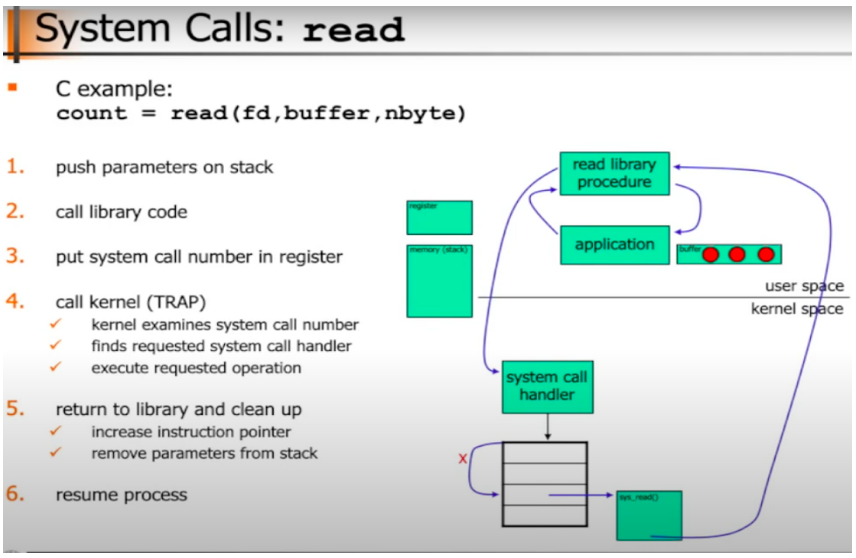
- En driver som kjører inne i kjernen/kernel
- Gjør delen beskrevet ovenfor hvor man benytter driverne til å kommunisere med de overnevnte enhetene slått sammen til “Device controller”(se bildet ovenfor).

System calls

Interface mellom OS og brukere er definert som “system calls”. Å lage et “system call” er lik som en prosedyre/funksjon i et programmeringspråk, men “system call” vil gå inn i kernel.



Parameterne i “system call” kan gjerne være et periferenhet hvor vi da ønsker å modifisere den enheten i kernel. Eksempel blir vist under over hvordan dette fungerer



Interrupt Program Execution

Noen ganger så har vi instrukser som vi ønsker å avbryte. Disse gjøres gjennom “Interrupts” og “Exception”.

Interrupts

- Interrupts er elektroniske signaler som resulterer i at man tvinger å avslutte en gitt instruks og la de andre instruksene får mulighet til å bli eksekvert.
- Disse forårsakes av asynkroniske hendelser, dvs hendelser som blir forårsaket utenifra f.eks at nettverkspakker blir motatt av prosessoren. Disse hendelsene kan være disk operasjoner, utgåtte tidsklokker som bestemmer hvor lenge instruksen skal eksekveres, kommende nettverkspakker osv.
- Disse “Interrupts” kan bli avbrutt eller masket ut.

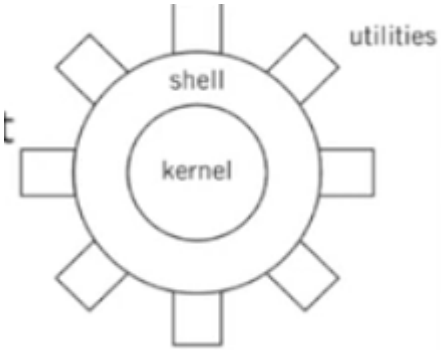
Exceptions

- En annen måte for en prosess å avbryte et program på, er gjennom exceptions og gjelder for synkroniske hendelser
- Disse utføres når en prosessor detekterer en predefinert kondisjon imens den utfører en instruks.
- TRAPS: Prosessoren møter en kondisjon som “exception handler” kan håndtere (overflow, møter på et “break” point i stacken som utføres i et system call)
- FAULTS: Prosessoren møter en feil som “exception handler” kan rette opp(Division by zero, wrong data format)
- ABORTS: Terminerer prosessen på grunn av feil i hardware som hindrer prosessoren i å utføre.

Hvordan skal “OS” organiseres?

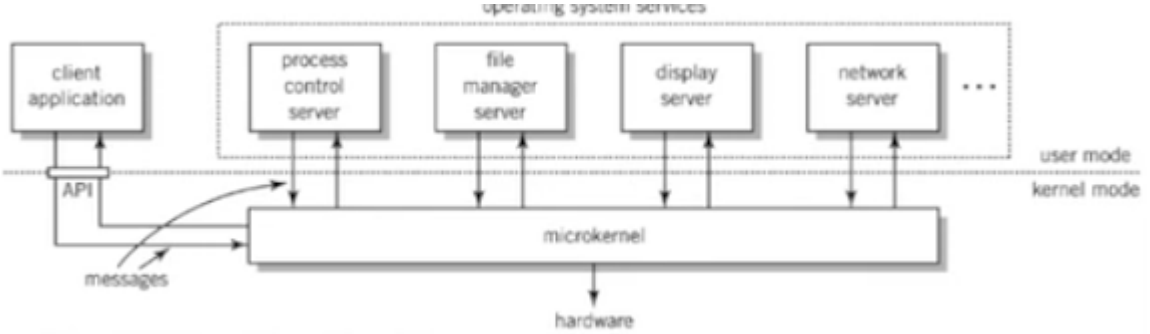
Nå som vi har lært hva “OS”-er, så må vi da ha kjennskap til måter vi kan dele “OS” på. Årsaken er at, hvordan skal vi sette opp “OS” slik at den håndterer alle funksjoner, system calls, osv. Heldigvis finnes det to ulike måter vi kan dele “OS”-kjernen på hvor vi har **monolitiske kjerner** og **mikrokjerner**.

Monolittiske kjerner



En monolittisk kjerne er en “OS-kjerne” som er bygd opp av en rekke funksjoner lenket til et stort program. Fordelen er at håndtering av system-kall vil være effektivt. Ulempen ved en slik oppsett, er at om det gjøres en feil i programmet, så vil hele programmet krasje. Dermed vil “OS-kjernen” også krasje, så det krever godt implementerte funksjoner og håndtering av feilkoder for å unngå dette.

Mikrokjerne



Hovedfunksjonaliteten lagres i en mikrokjernen som ligger på toppen av hardwaren. Videre vil alle andre “OS-tjenester” kjøre gjennom denne mikrokjernen som egne komponenter og enheter. Dette kan sees i bildet ovenfor hvor vi ser at gitte tjenester er i hver sin ramme. Fordelen er at vi ser hardware-komponenter er strukturert og man holder oversikt over ting. Ulempen er at det er mye som må gås igjennom for å utføre et gitt arbeid. F.eks, la oss si at vi ønsker å lytte til en videofil, så må vi utføre et systemkall i mikrokjernen. Mikrokjernene håndterer dette, men den vil da la ett av komponentene vist ovenfor, håndtere system kallet også vil tilstanden til systemkallet bli passert fram og tilbake.