Hva er en prosess?

* Et eksempel for å illustrere hva en prosess er, kan vi sammenligne et dataprogram med ei bok. På samme måte som kjøringen av et program er en prosess, kan en også si at lesing av ei bok er en pågående aktivitet som kan kalles en prosess.

* Et annet eksempel er ei kakeoppskrift som er «helt død» så lenge den står i kokeboka, men når du begynner å bake setter du i gang prosessen kakebaking.
* En kan si at et program er en sekvens av instruksjoner. Kjøringen av en slik sekvens av instruksjoner er da en prosess.
* Når vi skal kjøre en kommando vil det skje en rekke aktiviteter i datamaskinen, hver aktivitet foregår som en egen prosess.
* Et program er ikke en prosess før den har knyttet til seg en del ressurser
* Hver prosess har sine spesialoppgaver, og for å utføre en større og mer sammensatt oppgave må prosessene samarbeide (kommunisere) med hverandre. Hver prosess operer innenfor sitt eget adresseområdet, og kun der. Dette er et viktig prinsipp i forhold til sikkerheten i systemet.
* Prosessene har egne dataområder, selv om de deler programkode.
* Vi skiller mellom prosesser som kjører i brukermodus og kjernemodus, før var det bare preemptive i brukermodus, men senere har dette også kommet til kjernemodus. Det betyr at prosesser som kjører i kjernemodus kan avbrytes og dermed kan andre kjøre mens en kjerneprosess må vente på et eller annet. Dette gjorde at Operativsystemet responderte mye raskere på hendelser i systemet.
* Et viktig poeng er at prosessen bare får lov å operere innenfor sitt eget adresseområde av sikkerhetsgrunner. Referanser til adresser utenfor adresserommet blir betraktet som brudd på sikkerheten. Operativsystemet vil da gripe inn og stoppe den aktuelle prosessen.

Oppbygging av prosesser:

* **Prosessdeskriptor/PCB:** Dette er hvor Operativsystemet gjemmer unna en del info om hver prosess, som trengs for å kunne starte opp prosessen igjen på en enkel måte. Infoen som lagres er feks:
  + **Innholdet av cpu-**registerene idet prosessen ble stoppet
  + **Adressen til neste instruksjon som skal utføres**
  + **Adresse til kode-området(program)**
  + **Adresse til data-området**
  + **Forbruk av cpu-tid så langt, etc.**

Når prosessen skal gjennopprettes fylles spu-registeret opp med data fra prosessdeskriptoren.

Hver gang det skiftes på hvilken prosess som kjører skjer det en slik lagring av statusinformasjon fra CPU til PCB for avtroppende prosess, og en innlegging av info fra PCB til CPU for påtroppende prosess.

* Et program som definerer hva prosessen skal gjøre.
* Et dataområde som prosessen operer på.
* Et sett av ressurser som prosessen trenger for å kunne kjøre.
* En PCB for å holde følge med progresjonen under kjøringen av prosessen.

Operativsystemets jobb:

* Den første jobben til Operativsystemet er å sette opp et miljø for prosesser og ressurser, og for oppretting av nye prosesser.
* **Enprosess-system:** Når Operativsystemet kun kan kjøre en sekvens av instrukser (prosess) om gangen. Dvs at for at en prosess må være avsluttet før den neste kan starte opp.
* **Flerprosess-system:** Flere sekvenser med instrukser tillates å kjøre samtidig i datamaskinen, med samtidig mener vi en slags tidsdeling av cpu-en der hver av prosessene som kjører på cpu.en skifter på å bruke cpu-en. (kvasiparallell).
  + **Merk:** selv om vi sier at Operativsystemet kjører flere samtidig prosessorer, så er det alikavle kun en prosess som kan kjøre om gangen, fordi skifte av prosesser skjer så fort at over tid kan det se ut som prosessene kjører samtidig.
* Det er operativsystemet sin oppgave å context switche. For å vite hvilke prosesser som få kjøre på cpu-en og hvor lenge den skal kjøre bruker Operativsystemet forskjellige algoritmer. For eksempel kan Operativsystemet la prosessen kjøre inntil den ikke har mer arbeid å utføre, eller til det kommer en ny prosess med høyere prioritet, eller til den finner ut at den kjørende prosessen har kjørt lenge nok å suspanderer den og henter inn en ny prosess.
  + **Context switching:** veksling mellom prosesser.
* Operativsystemet fordeler cpu-tid til prosesser – kun en prosess kan kjøre om gangen, forutsatt at det er en cpu i maskinen.
  + Om en prosess som kjører blir avbrutt vil Operativsystemet føre kontrollen tilbake til prosessen etter avbruddet er ferdig.
  + Ved hjelp av suspendering greier Operativsystemet å fordele cpu-tid til alle prosessene.
* Operativsystemet må holde rede på hvilke prosesser som har fått lov til å låne en ressurs, og hvilke prosesser som venter på hvilke ressurser. Til dette formålet finnes PCBer og ressursdeskriptorer.
* **Preemptive operasjonssystemer:** Operativsystemet har full kontroll på prosessene. Den bestemmer hvor lenge posessen får bruke cpuen, og fjerner prosesser dersom tidskvantet utløper og slipper nestemann til.
* **Non-preemptive operasjonssystemer/cooperative multitasking:** Her var det lagt inn en funksjon som sjekket om andre prosesser ønsket å bruke cpu-en. I såfall ville kjørende prosess overlate cpu-en til neste prosess. Dette hadde visse ulemper:
  + Om et program henger seg opp vil den ikke komme seg til delen der den spør om andre prosesser vil bruke cpuen og det vil derfor ikke skje noe prosesskifte og maskinen vil låse seg.
  + Det er programmereren sitt ansvar å lage programmer som er sammarbeidsvillig, dersom dette ikke gjøres vil et prograam ha cpu-en helt frem til det er ferdig kjørt uten at Operativsystemet kan gripe inn.

Livsløpet til prosessen:

* Når Operativsystemet starter opp for første gang lages det en første prosess (init-process). Dette skjer for at vi ikke har noen mekanismer for å opprette prosesser på det tidspunktet, så her er det snakk om å «håndlage» en prosess.
* For å lage nye prosesser så forker Operativsystemet denne init-prosessen, da lager prosessadministratoren nye prosesser ved behov ved fork-funksjonen, som lager en kopi av PCBen til en allerede eksisterende prosess.

For at vi skal kunne referere til PCBen, altså til prosessen følger det med en pid (prosess identifikasjon). Piden brukes bl.a når vi skal ta livet av en prosess.

Siden vi bruker init-processen som utgangspunkt for alle prosessene omtales dette for prosessers mor.

Etter forkingen erstatter prosessene programkoden sin til en programkode som er mer tilpasset den oppgaven som skal utføres. Prosessen er den samme men programmet er skiftet ut, piden er derfor lik.

* Selv om en prosess ikke bruker CPU-en eksisterer den fremdeles i datamaskinsystemet, noen venter feks på data fra brukeren, andre venter på datablokker som skal hentes fra disken. Disse prosessene ligger i en såkalt I/O-kø, der de venter på at etterspurt data skal bli tilgjengelig. Når de får tak i dataen flyttes de fra I/O-køen til CPU-køen der de ligger og venter på å få kjørt på CPU-en. De har altså fått tilgang til alle ressursene den trenger bortsett fra CPUen. CPUen er den siste resursen som blir tildelt.
* Prosessen som bruker CPU-en i øyeblikket må gi fra seg CPU-en før andre kan slippe til. Det skjer f.eks. når
  + Kjørende prosess skal utføre en i/o-operasjon
  + Prosess-administrator fjerner kjørende prosess fra CPU-en fordi den har kjørt lenge nok (suspenderer).
* **«Prosessens livsløp»:** 
  + Rett etter at en prosess er opprettet havner den i tilstanden **klar.** Det betyr at den er klar til å kjøre på CPU-en. Flere prosesser er i denne tilstander og danner en CPU-kø
  + Prosessen går deretter inn i **kjørende** tilstand, her blir den til et av følgende tre ting skjer:
    - Den tiden som er satt av til prosessen utløper, og prosessen må tilbake til klar-tilstanden, og en annen prosess får kjøre på CPU-en.
    - Prosessen ber om en ressurs, det tar tid å få tak i denne, og prosessen må dermed gi slipp på CPU-en og går over i en **blokkert** tilstand. Der blir den til ressursen er ledig
    - Prosessen er ferdig og avslutter.
  + Den siste tilstanden er blokkert, da er prosessen i ventekø der den venter på en gitt ressurs som den har etterspurt. Når ressursen er klar skifter prosessen tilstand fra blokkert til klar.
  + **OBS!** Prosessen kan ikke gå direkte fra blokkert til kjørende, den må alltid innom klar-tilstanden og CPU-køen der alle de andre prosessene som er klar befinner seg.
* Forde**len med å ha flere prosesser aktive samtidige er**:

1. at hver prosess kan konsentrere seg om en helt bestemt oppgave uten å bry seg om hva andre prosesser i systemet måtte holde på med. Feks om vi har en prosess for input og en prosess for kommandotolkning, der inputen lagrer hvert tegn fra et tastatur i en buffer og kommandotolkeren henter tegnene og tolker hvordan kommandoer som skal utføres. Du kan altså type-ahead, der input kan ta inn nye tegn uavhengig av om kommandotolkeren er klar.
2. Når Operativsystemet trenger å håndetere alle de forskjellige i/o-enhetene som er koblet til maskinen. Kommunikasjonen med i/o-enheter er som oftest asynkron med hensyn til kjøringen av sleve programmet.

Prosess-administrator:

* En prosess spesifiserer sitt ressursbehov til prosess-administratoren i Operativsystmet.
* **Prosess-administratoren:** sin oppgave er å sørge for at prosessen får de ressursene den spør etter, og at prosessen blir kjørt når det er dens tur. Den lager en modell av hver ressurs, den bruker denne modellen til å tilby prosessene ressurser. Den holder orden på prosessene som venter i kø. Grunnlaget for at prosess-administratoren kan gjøre jobben sin er:
  + **Ressurser og prosesser er representert**
  + **Metoder for scheduling**
  + **Synkronisering av prosesser**
  + **Metoder for å håntere vranglås**
* Å allokere ressurser til prosessen skjer etter PCB-en til en prosess er opprettet, og denne jobben gjør prosess-administratoren. Åpenbare ressurser som prosessen trenger er plass i minnet til programkode, data og stakk. (prosess-administratoren er en del av operativsystemet)
* Prosess-administratoren har et dataområde som representerer hver av ressursene, og kan derfor administrere disse i forhold til at flere prosesser spør om å få bruke disse samtidig.
* Det er prosessadministratoren som sørger for at prosessen skifter tilstand.

Ressurser:

* Pga det kun finnes et begrenset antall ressurser tilgjengelig må prosessene stilles i en ressurskø dersom resursene er i bruk når de etterspørres. Dette har ressurs-administratoren ansvaret for.
* Om en prosess trenger tilgang til andre ressurser under kjøring vil den etterspørre ressursen. Dette kan ta tid for ressursen kan være opptatt. Derfor vil det alltid skje et prosesskrift på cpu-en ved en slik sitasjon.
* Mens en av prosessene kjører på CPU-en vil alle andre prosesser befinne seg i en eller annen ressurskø. F.eks:
  + Prosessen bruker en i/o-enhet, og venter akkurat på at data fra i/o-enheten skal bli tilgjenglig.
  + Prosessen venter på at den forespurte ressursen skal bli ledig.
  + Prosessen har fått alle de ressursene den har spurt etter, og venter nå bare på cpu-ressursen for å kunne kjøre videre.
* Hver ressurs har en **ressursdeskriptor** (en datastruktur som inneholder data som beskriver ressursen). Ved hjelp av denne kan en ressurs allokeres til en prosess ved forespørsel. Når prosessen er ferdig med resurssen frigjøres den fra prosessen og allokeres til en annen prosess, eller den havner på ei liste av ledige ressurser.
* **Adresserom:** dette er en serie av minnelokasjoner som prosessen kan lese fra og skrive til. Den starter på adresse 0 og går opp til en max adresse, og inneholder programinstruksjonene til prosessen, samt dataområde og stakk. Videre er det snakk om adresser til registre i CPU og registre som representer ressurser. Alt dette utgjør prosessens adresserom. Størstedelen av adresserommet består av minnelokasjoner i primærminnet der programinstruksjoner, data og stakk befinner seg.
  + Når kildeprogrammet oversettes av en kompilator lagres den delen av adresserommet som utgjør minne-adressene og lenkes sammen med biblioteksfunksjonen som er brukt i programmet for å lage en kjørbar versjon.
* **Memory mapping:** er når vi aksesserer til noe går via ei adresse i minnet, f.eks registre i en i/o-kontoller går via ei adresse i minnet.

Tråder

* Vi ønsker flere prosesser som deler det samme adresseområdet. Da kan nemlig disse prosessene utvikle data, og dermed kunne kommunisere med hverandre via dataområdet innenfor det felles adresseområdet. Det er her begrepet tråder kommer inn.
* **Tråder**: separate kjørende programsekvenser som opererer innenfor samme adresseområde.
* Prosessen er viktig når det gjelder ressurser og ressursallokering. Hver prosess har knytet til seg et sett av ressurser til enhver tid. Det andre begrepet som man forbinder med prosesser, er selve kjøringen av programinstruksjonene til prosessen. Dette ses ofte på som en tråd. Tråden har en instruksjonsteller som hele tiden viser neste instruksjon som skal kjøres.
* Tråden kjører innafor en prosess, og har tilgang på alle de ressursene som prosessen eier, men kan allikevel behandles som en separat enhet når det gjelder scheduling. Flere tråder tilhørende den samme prosessen kan meget greit befinne seg i CPU-en.
* Prosessen tar seg av ressursene, men trådene tar seg av selve kjøringen av programkoden. Flere tråder kan kjøre samtidig på den samme programkode (på forskjellige steder i programkoden) og de har alle tilgang til prosessens ressurser.
* Tråden har hvert sitt område for å lagre unna status og lokale data.
* Til forskjell fra ulike prosesser, som ofte opererer i «fiendeland», så er trående innstilt på å samarbeide. Trådene må samarbeide, de er ikke ute etter å konkurrere med hverandre.
* Trådene har også forskjellige tilstander som kjørende, klar og blokkert. Trådene er uavhengig av hverandre i forhold til CPU-en.
* Hovedgrunnen for bruk av tråder er at applikasjonen vil kjøre flere aktiviteter samtidig, og at alle disse aktivitene har tilgang til det samme adresseområdet. Dette er til forskjell fra vanlige prosesser der hver prosess har sitt eget separate adresseområdet utilgjengelig for de andre prosessene.
* En annen viktig grunn for bruk av tråder er at de ikke har knyttet til seg noen ressurser, og dermed er det mye enklere og mindre krevende å opprette nye tråder (100 ganger raskere). Det vil ta all for lang tid å opprette nye prosesser for hver forespørsel, og å bare bruke en prosess utgår også siden den vil være utilgjengelig mens den hold på å betjene en av forespørselen.
* Tråder er svært kjappe å opprette, derfor er responstida kort. Med mange aktive tråder som betjener hver sin forespørsel vil cpu-tida bli mye mer effektivt utnyttet. Ventetida mens en diskoperasjon foregår brukes til å kjøre andre tråder.