# Prosesser og tråder

# Definisjon av prosess

- Enkel definisjon:
  - En prosess er et program som kjører på datamaskinen
- Mer presis definisjon:
  - En prosess er en samling av ressurser som er nødvendige for å utføre en oppgave som er beskrevet med programkode på en datamaskin

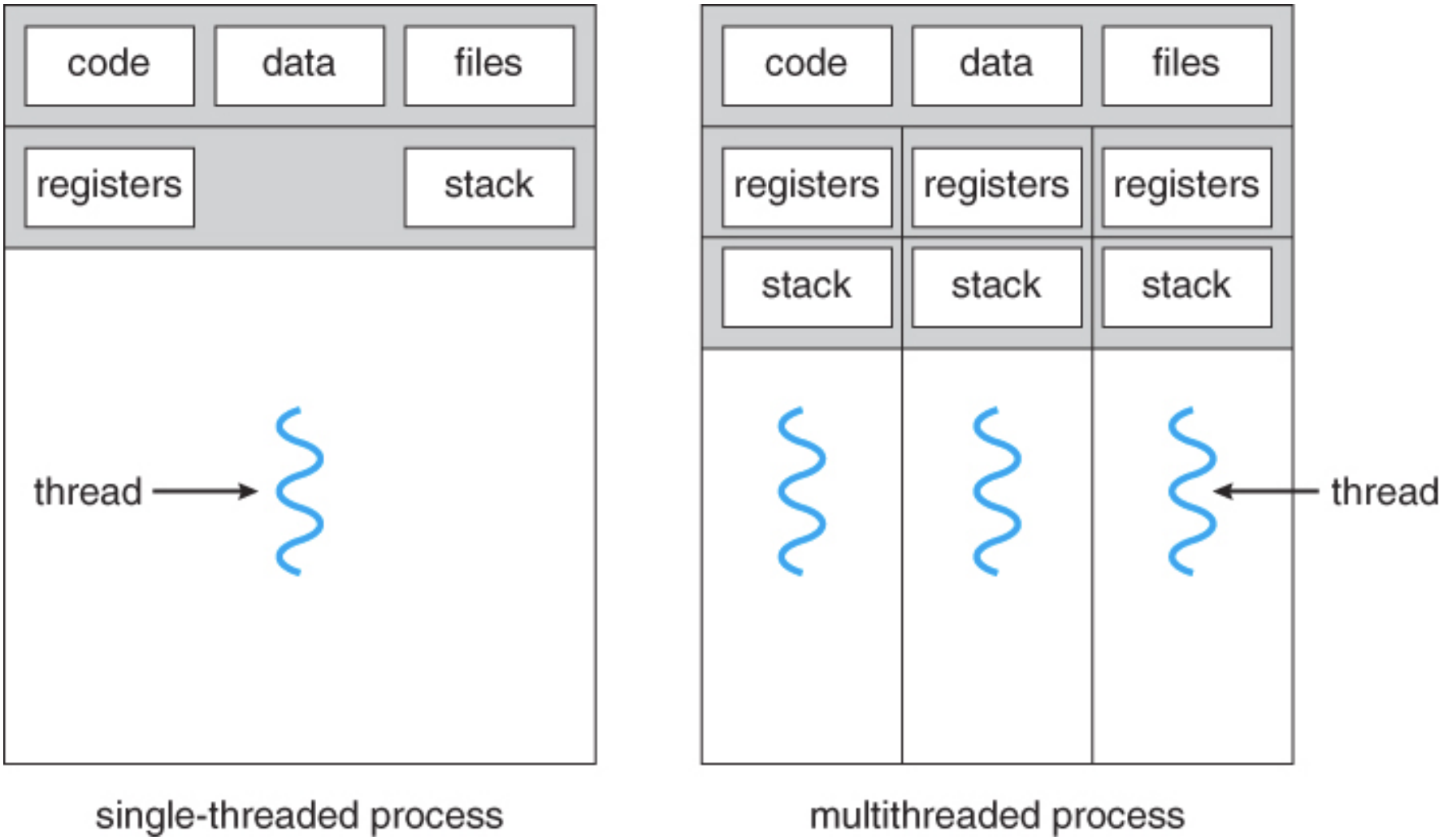
# Ressursene som utgjør en prosess

- Programkoden:
  - Ferdig kompilerte maskininstruksjoner som er lastet inn i RAM og er klar for kjøring
- Minneområdet (RAM) som er reservert for prosessen
- Filer og I/O-enheter som skal leses/skrives
- Kontrolldata som OS bruker for å styre prosessen:
  - Prioriteter
  - Rettigheter (til filer, minne, I/O-porter etc.)
- En eller flere *tråder* (“threads of execution”)

# Tråder / threads

- Tråd (“thread of execution”):
  - Utførende enhet som går gjennom programkoden og eksekverer maskininstruksjoner
- Tråder må ha egne interne kontrolldata* for å holde rede på bl.a.:
  - Hvor i programkoden tråden befinner seg
  - Hvilken maskininstruksjon som er den neste som skal utføres (eksekveringsstack)
  - Hvilke dataverdier som instruksjonen skal bruke

*: Tråddata lagres oftest i registre i CPU’en




- En prosess kan inneholde én eller flere tråder
- Tråder *kan* kjøre parallellt og uavhengig av hverandre, men mange problemer krever at trådene *synkroniseres*

# Analogi med et kjøkken

Kjøkken

- Formål: Lage mat
- Kokker
- Råvarer og utstyr
- Oppskrifter

Prosess

- Formål: Løse oppgave
- Tråder
- Minne, disk, kontrolldata
- Programkode

# Prosess vs. program *

- Et program er en *statisk* enhet:
  - Ligger lagret som kildekode eller maskinkode
  - Koden forandrer seg ikke når programmet utføres
- En prosess er en *dynamisk* enhet:
  - Endrer seg hele tiden under kjøring av programmet
  - Trådenes programteller, stack og registre oppdateres
  - Variablene som er lagret i RAM endrer verdier
  - Filer åpnes, skrives til, lukkes

*: Merk at begrepene “program”, “prosess” og “tråd”” ofte brukes om hverandre og upresist

# Tråd vs. prosess

- Tråder:
  - Sekvensiell “flyt av utførelse” inne i en prosess
  - Flere tråder kan kjøre parallelt (”samtidig”)
- Trådene er “lettvekts-prosesser”:
  - Mye raskere å opprette ny tråd enn ny prosess
  - Mye raskere kommunikasjon mellom samarbeidende tråder inne i en prosess, enn mellom ulike prosesser
- Parallelle tråder er utfordrende for programmereren:
  - Trådene deler minne, mye feil/inkonsistens mulig
  - Samarbeidende tråder må synkroniseres i koden

# Single- og multithread OS

- Tidlige OS: Prosesser med bare én “tråd”:
  - Begrepet “threads” fantes ikke, bare separate prosesser
  - Deling av data mellom prosesser er vanskelig
  - Prosesser med bare én tråd stopper når de må vente på en ressurs, f.eks. I/O eller en annen maskin på nettet
- OS med flere tråder i hver prosess:
  - Multi-thread OS (og Internett) vanlige for ~20 år siden
  - Flere tråder gjør at f.eks. nettlesere lett kan fortsette selv om en server ikke svarer
  - Både Windows og Linux er multi-thread

# Prosessenes tilstand

- En prosess har hele tiden en *tilstand*, som endres etter som programmet kjøres/tiden går
- Tilstanden til en prosess bestemmes av:
  - Dataene som ligger i RAM
  - Status for alle ressursene som er tilknyttet prosessen
  - Tilstanden til alle trådene som kjører inne i prosessen
- Hele tilstanden til en prosess må kunne *lagres* unna, f.eks. ved avbrudd (interrupt) i CPU for å gjøre I/O
- Prosessen må etterpå kunne hentes frem igjen og gjenskapes slik at den fortsetter å kjøre med *nøyaktig samme tilstand* som da den ble lagret

# Lagring av prosesstilstander i OS

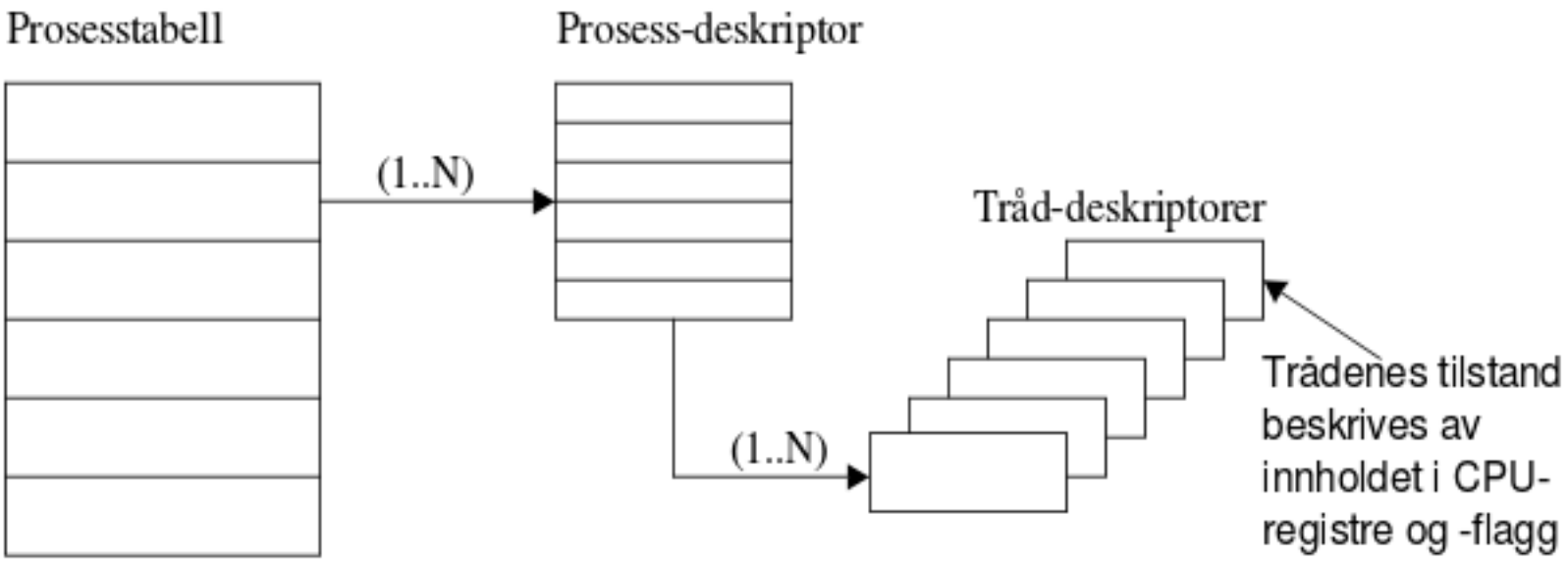
- Tilstanden til hver enkelt prosess lagres i en *prosess-deskriptor*, som bl.a. inneholder:
  - Prosessnavn og -eier
  - Prioritet og rettigheter
  - Informasjon om minneområde(r) som er tildelt
  - Trådliste
  - Informasjon om selve programmet for prosessen
  - Informasjon om *forelder*- og *barneprosesser*
- Deskriptorene for alle prosessene som finnes på maskinen lagres i *prosesstabellen* i OS-et

# Lagring av trådtilstander

- Tilstanden til hver av trådene i en prosess lagres i en *tråddeskriptor*, som bl.a. inneholder:
  - Instruksjonspeker, som inneholder adressen til instruksjonen i programkoden som tråden nå utfører
  - Innholdet i CPU-registre
  - Trådens *utføringsstatus*, som er en av tre mulige:

    | | |
    |---|---|
    | *ready* | klar til å kjøre i CPU, “venter på tur” |
    | *running* | kjører i CPU |
    | *blocked* | kan ikke kjøre, venter på f.eks. I/O |

- Tråddeskriptorene lagres lokalt i prosessene




*Figur 4-1: Prosesstabell, prosessdeskriptor og tråddeskriptor*

# Forelder- og barneprosesser

- *Alle* prosesser lages ved at en tråd i en prosess oppretter en ny prosess, gjennom et *systemkall* til Linux-kjernen *
- Prosessen som lages er *barn* av *forelder*-prosessen:
  - F.eks. blir alle prosesser som startes fra kommandolinjen barn av prosessen som kjører shellet
- OS lagrer alle barn-forelder forhold mellom prosessene
- En forelderprosess kan vanligvis styre, stoppe, starte og slette sine egne barneprosesser
- Tilsvarende slektskap finnes mellom *trådene* i en prosess

*: Med unntak av den aller første prosessen, `init`

# Spawn*: Opprettelse av en Linux-prosess

- I Linux brukes begrepet “spawning” om det som skjer når en forelderprosess oppretter en barneprosess
- Spawning gjøres av shellet eller av en annen applikasjon, ved å utføre et av disse systemkallene i Linux-kjernen:

  `fork()` `vfork()` `clone()`

- Systemkallene er API-funksjoner skrevet i språket C
- Alle systemkallene for spawning er dokumentert i `man`-systemet (section 2: system calls)

*: *spawn*: to produce; to give birth to; give rise to

# `fork`, `vfork`

- Oppretter en barneprosess som er en *kopi* av forelder-prosessen, inkludert programkoden, innholdet av RAM, registre, filer, prosess- og tråddata
- `fork`:
  - Barneprosessen får et eget minneområde i RAM
- `vfork`:
  - Barne- og forelderprosess deler minneområdet

# `clone` – for å lage tråder

- `clone`
  - Forelder og barn er kopier av hverandre
  - Deler både minneområde og de fleste andre ressurser knyttet til prosessen
  - Brukes til å opprette nye *tråder* i Linux

# `execve` – for å starte nye programmer

- De tre systemkallene for spawning oppretter alle *kopier* av forelderprosessen
- Systemkallene `exec`/`execve` *erstatter* nåværende prosess med en *ny* prosess
- Hele innholdet av prosessen som *kaller* `exec` slettes, og byttes ut med helt nye prosessdata
- Vanlig å først gjøre en `fork`, og deretter la den nye barneprosessen gjøre `exec` for å bytte ut seg selv med et nytt program som skal startes

# `init`

- Den aller første prosessen som startes i Linux etter booting
- `init` opprettes av Linux-kjernen, som igjen startes av ROM-kode
- `init` er *stamfaren* til alle andre prosesser på systemet
- `init` er “foreldreløs”, alle andre prosesser har en forelder

# Prosess-kjeder og -trær

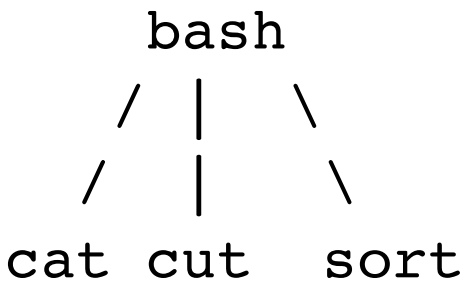
- Kan alltid dannes en kjede av prosesser fra nåværende prosess til forelder, til forelders forelder osv. helt opp til `init`
- En prosess kan spawne *flere* barn, alle prosessene som kjører på et Linux-system utgjør derfor et prosess-tre
- `pstree` – kommando for å se (deler av) prosesstreet

# Eksempel på prosess-tre

- Gir følgende kommando i Bash:

```
cat | cut -d: -f1 | sort
```

- Bash (shellet) er en egen prosess på maskinen
- Bash vil her starte tre nye prosesser, en for hver kommando
- Linux setter opp prosessene slik at de kommuniserer med hverandre via pipe-buffere
- Lokalt prosess-tre:



# Eksempel på prosess-kjede

- `init` starter GUI-systemet `gnome`
- `gnome` starter `gnome_terminal`, terminalvinduet
- `bash` kjører inne i dette terminalvinduet
- Deretter startes en `find`-kommando fra kommandolinjen:

```
init
  \__gnome
        \__gnome_terminal
                    \
                     \__bash
                         \ __find
```