Notater OS

*Resource Sharing* can happen in two ways:

1. **Space-sharing**: Sharing the available space in memory (RAM) with other programs.
2. **Time-sharing**: Only one program can compute in the CPU. The OS must schedule this time fairly and securely.

*Resource isolation*: Two programs must not access the same resource at the same time. E.g. a program cannot read a different programs data and instructions in memory.

*Resource sharing*: If two programs want to communicate with each other, i.e. exchanging data, a common space in memory is allocated. The OS needs mechanisms to handle this securely.

*Program*: Program code that lies completely “dead” on a file, i.e. the recipe for a process.

*Process*: A programs code under execution, i.e. the activity of the recipe.

* When a process wants to use a resource, it needs to ask the OS kernel. If the resource is not immediately available, the process is suspended, and executed later.
* When a process is finished executing, it gives away its resources.

The three states of a process:

1. Waiting for IO resources
2. Waiting for CPU resources
3. Executing on CPU resource

Components of a process:

* **Program** code that is going to be executed. I.e. instructions on the runnable file.
* **Data** the program code use when running.
* **Resources** that the program demands. E.g. a file that has data the program needs.
* **Status** to the process when running. This is called a process descriptor?

*Sequential processing*: Inefficient if processes needs to wait for IO resources. The CPU resources should be used during this wait.

*Parallell processing*: The basis for the use of parallel processing is that processes will sometimes/often have to wait for IO resources. Also, the memory (RAM) needs to have space for all the resources for this to be efficient.

Instruction cycles: All the processing that happens from when an instruction is requested from memory, to the next instruction being requested.

*Fetch cycle*

*Execution cycle*

CPU

RAM

CPU

*Instruction*

*1. CPU request instruction*

*2. RAM transfer instruction to CPU*

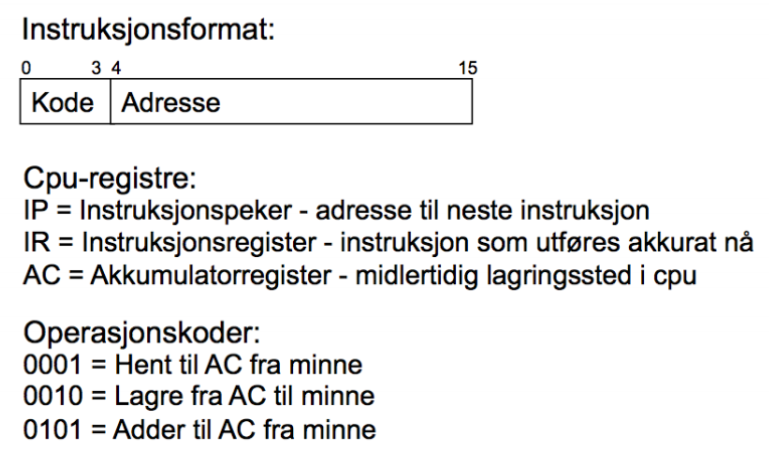
*3. CPU executes instruction*

*Goes back to 1.*

*Next instruction*

*Instruction*

How instructions are structured and executed in CPU.



*IO instructions:* The program instructions for the IO modules is located within the driver for the IO device. The CPU will normally request data from IO modules, but sometimes it is useful to go directly from memory to IO module. This is called *direct memory access* (DMA).

**Interrupts**

A mechanism necessary to take advantage of the available resources efficiently. I.e. interrupts will make the CPU usage more efficient and secure.

* While a process is waiting for IO the CPU will work on other processes, but when the IO is finished, it needs to send an interrupt to the CPU. When this interrupt occurs, the CPU knows the process that was originally waiting for IO, is now ready to execute.
* Normal actions, such as moving the mouse or update screen.
* Parallell processing (multitasking). Each process should get some CPU time, one process should not have control over the CPU over longer periods of time. Will make the system slow and “sluggish” if not implemented. It could also be a vulnerability for the OS if a process will never finish to execute or wait for IO.

# Linux system calls

**open**

This system call is used to open a file with a specified pathname. If the file does not exist it could be created by open() (if it is specified in the flags)

**read**

Read is used to read input from the user/keyboard. If you have a variable name after the read command, the input will be stored in that variable for later use.

**close**

Is used to close a file descriptor. A file descriptor is a number that identifies a specific open file in the OS. These numbers are stored in a global file table. By using this command with a number, we can remove a file descriptor so it does not refer to any file, and may be reused.

**write**

This system call is used to communicate with other users by copying lines form your terminal to theirs.

**fork**

Used to create a new process by copying a different process, also referred to as the “parent”. This new “child” process gets its own place in memory, but it will have the exact same data until further changes are made.

**exec**

This system call is used to replace a program in a process with a different program. I will also start running the process from the beginning.

**wait**

Is used to wait for a process has changed status. E.g. from un-ready to ready or running.

*These three commands are often used together in this order: fork – exec – wait. Fork will create a new process, exec will replace the program code and start running the new process. Wait will make the parent wait for the child to finish.*

# Ressurser

Ressursabstraksjon - menneskeliggjør oppgaven å programmere

OS er laget slik at det skal fungere for et bredt spekter av lignende maskinvare. En av oppgavene til OS er å tilby funksjoner som skal gjøre jobben enklere for programmerere. OS tilbyr en *abstrakt modell* av maskinvaren, en modell som *glatter over forskjeller mellom maskinvare av samme type.*

Fordelen med dette er at man kan utvikle programmer uten å ta store hensyn til hvilken underliggende maskinvare som finnes. F.eks. programmereren trenger ikke å vite hvordan en harddisk fungerer, de bruker funksjonskall også tar OS eller IO moduler av seg denne jobben. Ulempen er at spesialegenskaper som den aktuelle maskinvaren har blir utelatt.

Ressursdeling – fordeling av goder

Ulike ressurser som er tilgjengelige må fordeles av OS. OS vil ha den komplette oversikten over hvilke ressurser som er tilgjengelige og hvem som etterspør de. Uten dette kan f.eks. flere programmer ønske å skrive ut på en printer, da kan noen linjer komme fra det ene programmet og de neste fra et annet. Man deler gjerne mellom to ulike ressursfordelinger:

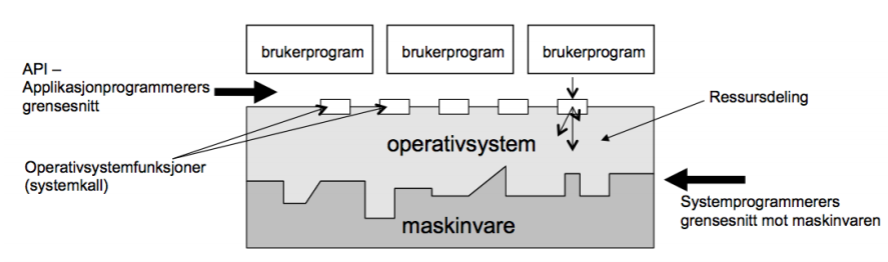
* Plass-deling: f.eks. deling av plass i minne
* Tids-deling: f.eks. som i en-kjernet CPU, hvert program får f.eks. 100ms hver på å kjøre.

Ressursisolering – sikkerhet

OS må garantere at ingen andre programmer har tilgang til den samme ressurser på samme tid. Ressursisolering skal garantere at det ikke foregår uautorisert tilgang til ressurser. F.eks. dataområdet til et program i minnet skal ikke kunne leses av andre.

Felles bruk av ressurser – kommunikasjon

Hvis to programmer ønsker å kommunisere, altså utveksle data, vil dette foregå via felles minneområde der begge programmene har tilgang til det samme minneområdet. Man må sikre at dette foregår på en kontrollert og sikker måte, derfor trengs det både mekanismer for ressursisolering og for felles bruk av samme ressurs i OS.



# Oppgaver til OS

OS har mange forskjellige oppgaver, men fire oppgaver er sentralt:

1. Utstyrsadministrasjon
2. Prosess- og ressursadministrasjon
3. Minneadministrasjon
4. Filadministrasjon
5. Utstyrsadministrasjon

Ressursabstraksjon er viktig, OS skal se på alle utstyrsenheter med et standardisert grensesnitt. For at dette skal fungere ligger det et driverprogram mellom OS og utstyrsenheten. Utstyrsadministrasjon dreier seg om hvordan OS håndterer allokering, isolering og deling av utstyrsenhet.

1. Prosess – og ressursadministrasjon

En prosess er en viktig del av et system, den har ingen hensikt uten ressurser som f.eks. CPU, minne, etc. Prosessadministrasjon innebærer funksjoner for å opprette nye prosesser, avslutte prosesser, sette prosesser i ventekø, sette prosesser i kjørende tilstand etc. OS skiller mellom 1. kjøring av prosesser på CPU, og 2. allokering av ressurser til prosessen.

*«Prosessen er en enhet for ressurstildeling og tråder er en enhet for tildeling av kjøretid (dvs. CPU).»*

1. Minneadministrasjon

Minneadministrasjon må sørge for at prosessene ikke ødelegger for hverandre. Når man kjører flere prosesser samtidig må man kunne plassere prosesser på vilkårlig plass i minnet og samtidig ha kontroll over alle prosessene. Minneadministrasjon må også kunne håndtere deling av et felles minneområde mellom prosesser når det er behov for det.

Dette kan gjøres på forskjellig vis. Sidedelt- og segmentdelt minneadministrasjon er viktige begreper. Moderne OS bruker fordelene til begge i en minneadministrasjon med sidedelte segmenter.

1. Filadministrasjon

Filer representerer lagringsenheten til datamaskinen. Filene er en abstraksjon av lagringsenheten. Disse kan opprettes, kopieres, slettes, navngis, sette tilgangsrettigheter på etc.

# Kjernemodus og brukermodus

Alle prosesser kan befinne seg i to forskjellige moduser: *brukermodus* og *kjernemodus*.

En prosess opererer i brukermodus når den utfører instruksjoner mot sine egne dataområder i minnet. Derimot så snart prosessen trenger å utføre IO operasjoner (lese fra tastatur, fra disk, skjerm, etc.) må den utføre et systemkall.

Prosessen vil da gå over i kjernemodus, da opererer programmet instruksjoner mot OS dataområder. Kjernen er lokalisert i et eget område i minnet. Denne modusen er fullstendig styrt og kontrollert av OS, dette gjør at prosessene ikke kan utføre ulovlige operasjoner, som f.eks. lese fra andre sine områder på disken, skrive over viktige systemområder, endre viktige systemparametre etc.