# Disposition 1 – Programs in relation to the OS and Kernel

What is a program? It is our executable file so is this a process? Not until we execute it. The OS is the layer between the user application and the hardware. The OS is privileged and can execute privileged code while a user application is restricted.

## Processes and threads

A thread lives inside a processes but a processes in itself has at least one thread the main thread, that is because a thread is really just a strand of execution.

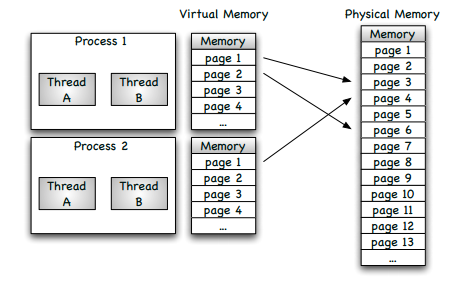
A process has its own virtual memory space; no two processes can access each other’s memories that would result in an error. A thread does not have its own memory space it uses the processes memory space.

Proces:

* Er en instans af et program der er ved at blive afviklet
* Har sit eget memory space
* Kan kun snakke med andre processer gennem mekanismer kontrolleret af OS (IPC)
* Processer kan føde andre processer som kan afvikle det samme eller andre programmer
* Må føde tråde i sit eget memory space

Tråde:

* Streng af eksekvering
* Alle processer har mindst 1 tråd
* Tråde fra den samme proces deler memory space
* Tråde kan gøre skade på hinanden



* Fortæl at hver proces har virtuel hukommelse!
* Som er forskellig fra fysisk hukommelse

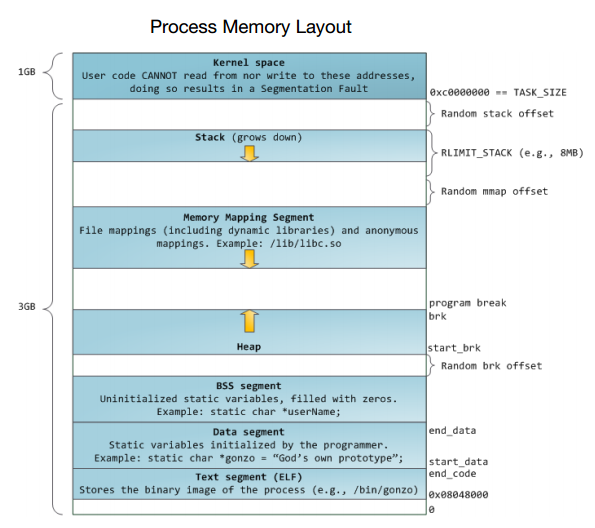
## Virtual Memory

Der findes en memory mapping unit MMU som linux bruger til at finde adresser ud fra de page tabels linux vedligeholder.

* Har delt hukommelsen op i pages oversætter en logisk page til en fysisk page
* Fordele: Vi skal ikke bekymre os om hvor meget fysisk hukommelse der er, vi får isoleret vores processer så de ikke kan læse hinandens hukommelse, det minimere risikoen for at fejl i en proces får der til at ske fejl i en anden proces.
* Kan skrive pages til harddisken for at lave plads til andre processer hvis en proces ikke laver noget med sin hukommelse i ram.
* Ovenstående kaldes paging/swapping.

## Process anatomy

* For en 32 bits process er hukommelsen en 4 GB blok der kan se således ud:

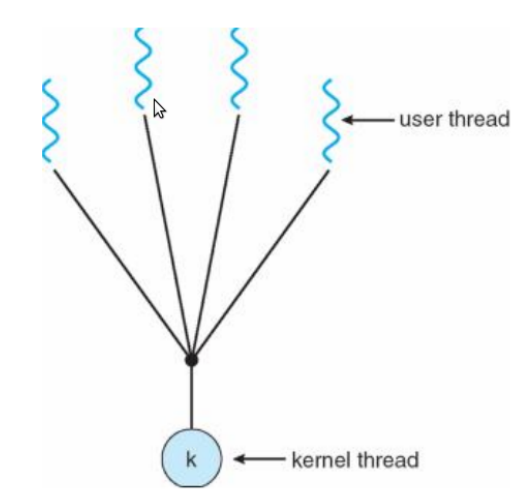


* **Kernel space:** her bor alle adresser fra kernel det er grundet til dette er at når virtuel hukommelse er slået til gælder det også for kernel derfor er alle proces nød til at have det segment med i deres hukommelses blok de kan dog ikke skrive til det.
* **Stack:** Her ligger alle variable for funktioner osv. Der bliver lagt en ny frame på stacken når en funktion bliver kaldt.
* **Memory mapping segment:** Her bliver indholdet af filer mappet direkte til IO det er godt hvis vi er ude efter preformance. Derfor bliver det også brugt til dynamiske libraries.
* **Heap:** Dette er til runtime merory allocation. Malloc og new.
* **BBS Segment:** Gemmer uninitialized globale variable dette område er anonymt
* **Data Segment:** Gemmer inintialized globale variable En streng vil her blive gemt som pointer og selve indholdet vil blive gemt I text segmented.
* **Text Segment:** Her bor den kode der skal eksekveres

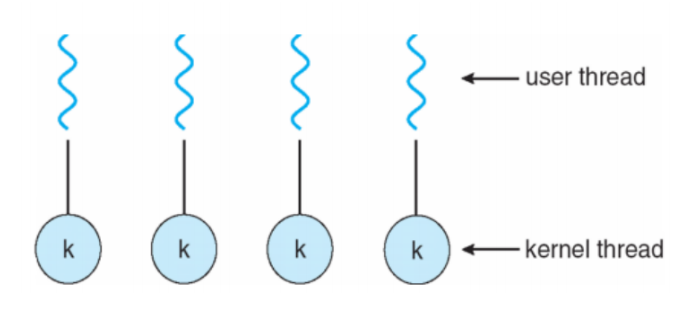
## Threading Model

There are three models:

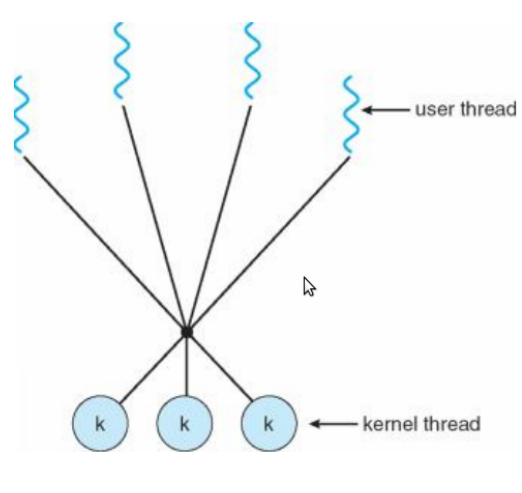
1. User level threading
   1. It is a simple implementation the kernel need not know about threads
   2. It is easy to switch context, the kernel need not be involved
   3. Cannot handle multiple cores



1. Kernel level threading
   1. Kernel needs to be thread aware
   2. User thread = kernel thread that the scheduler can control
   3. Efficient on multiple cores.



1. Hybrid level threading
   1. Complex implementation
   2. Needs good coordination between user and kernel land scheduler.

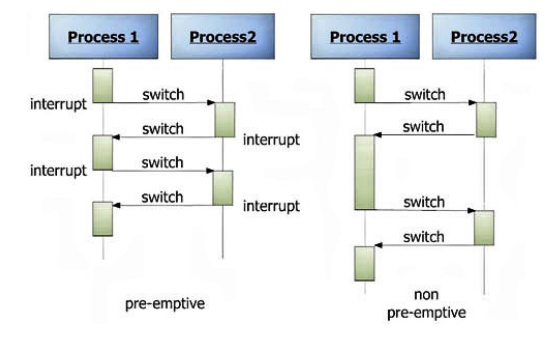


## Thread execution and scheduling

Hvordan styrers det så hvilke trade der bliver afviklet? Det styrer operativ systemet

Der er to strategier:

* **Premeptive**: Tasks kan kan afbrydes til enhvertid
* **Nonpreemptive**: Taks skal selv yielde cpu’en



Der kan også være forskellige prioriteter på trådene hvor det sår er schedulerens opgave at sørge for de tråde med højst prioritet kører først.

### Cache

* Hvis noget data bruges ofte og skal bruges hurtigt kan det skrives til CPUens cache
* Cachen er som regel meget mindre end fx RAM men til gengæld meget hurtigere. I cahcen bliver ofte anvendt hukommelse gemt så det hurtigt kan hentes frem

### Scheduler

* Schedulerens opgave er at sørge for at alle processor og deres trade får en fair del af den CPU tid der er til rådighed
* Det gør den alt afhængig af tråd prioriteter, opgavens arbejdsbyrde mm.
* Scheduleren virker i kernel-space og derfor skal evt. user-space tråde mappes ned til kernen alt afhængig af hvilken tråd model man vælger

