# Disposition04 – OSAPI

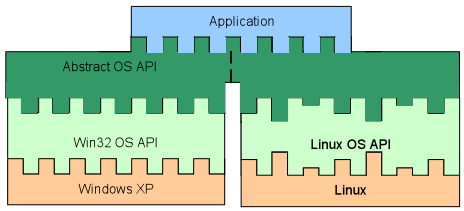
Hvad er et API – det star for application programming interface og er en specification af function som kan bruges til at programmere på en given enhed.

## The design philosophy - Why OO and OSApi?

Hvorfor object orienteret

* Det er nemt at arbejde med når man er vandt til det
* Det giver en ren kode
* Det er ikke så stort et gab mellem design og implemetion du kan tage et UML klasse diagram og straks ved du hvordan klassen skal kodes.
* Minimere fejl, nemmere at udvide

Hvorfor OSAPI?



* Forskellige OS har forskellige API’s til tråde mutex osv.
* OSAPI pakker kaldene ind således at vores program ikke skal kende til den OS specifikke implementering
* Programmet kan så genbruges på andre styresystemer ved at laver et OSAPI lag der passer til disse
* Vi skal kun kende en API at skrive vores programmer op imod efter at OSAPI laget er skrevet.

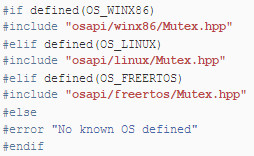
## Elaborate on the challenge of building it and its current design

### The pointer to implementation how and why

* Går ud på at gemme implementationen af en klasse ved at have et public interface der indeholder en pointer til implementationen.
* Funktionerne i det public interface er wrappere for funktionerne i implementationen
* Der allokeres ny plads til implementationen i constructoren via new.
* Det betyder at vi kan ændre implementatioen uden at det berører den der bruger klassen.
* Det kostet dog en pointer dereference at gøre dette og der er en smule mere arbejde i at implementere det.

### CPU / OS Architecture

* Da OSAPI kan virke på flere platforme skal vi kunne vælge den rigtige implementation når kompiler



* Linkeren kaldes så med –D flaget hvor det OS vi ønske at kompilere til så er specificeret
* Der skal her også bruges den rigtige toolchain alt efter hvilke platform der kompileres til.
* VI har brugt makefiler til dette.

## Effects on design/implementation

### MQ with pthreads vs MQ in OSAPI

* Essentielt er der ikke den store forskel udover at de nu ligger i OSAPIet.
* De bruges på samme måde
* Med pthreads var vi nød til at give køen med som parameter nu kan vi have den som member i den klasse der implementere tråden.
* Det er derfor i OO implementation nemmere at se hvad der hører til hvor -> koden er mere overskuelig.

### RAII

* Står for resource acquisition is initialization
* Det går ud på at vi hare en klasse som allokere en resource I sin konstrukter
* Hertil lader vi så destrukteren ryde op for os
* Nu kan vi så bruge det at c++ lader os have objekter på stakken det betyder at vi får nedlagt vores ressource igen
* Dette kunne være en mutex der låses/låses op.

### Threads before and now

Pthread:

* Der blev lavet en fri function som var trådens function
* Tråden blev erklæret og startet med pthreadcreate()
* Funktionen skulle returnere en void\* og tage en void\* som parameter.

OSAPI:

* Klasserne ThreadFunctor og Thread udgør tråd funktionaliteten i OSAPI
* Thread er kontrol klassen der starter og stopper trådene
* ThreadFunktor er den klasse alle tråde arver fra det er her run funktionenen bor (run er pure).
* En thread instans skal kende til en threadfunktor instans ellers kan der ikke startes en tråd
* Thread starter tråden ved at kalde den statiske funktion threadmapper. (er statisk for at overholde kravet om en fri funktion der tager og returnere void\*)
* ThreadMapper kalder run!

## UML Diagram to implementation – How?

