# Disposition 2 – Syncronisation and protection

## Data integrity – Concurrency challenge

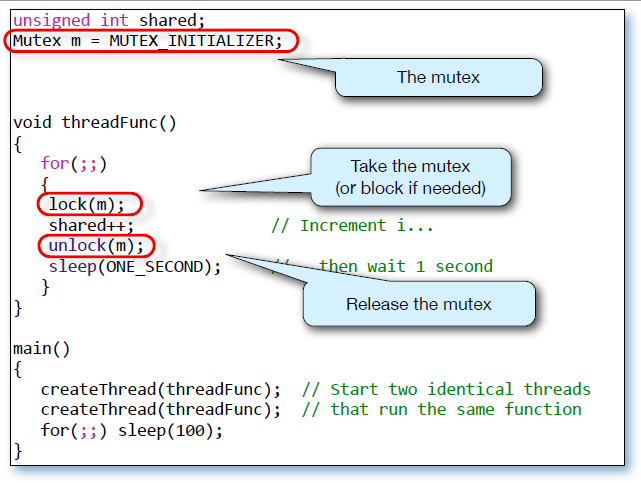
Hvordan sikre vi at tråde ikke kommer op at slås over noget data? Det er vi nød til at tænke over da tråde bor i samme proces og derved kan dele data.

Vi bruger preemptive scheduling i linux derved kan vi blive afbrudt midt i en opgave. Så som at tælle en variabel op. Med ++ operatoren

## Mutex & Semaphore

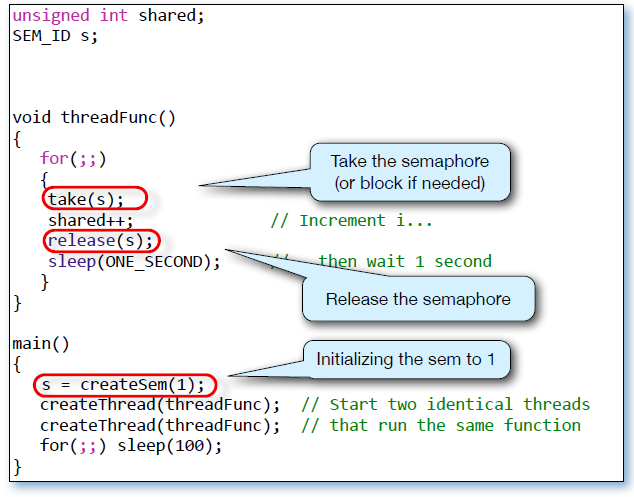
En mutex er en låse mekanisme en tråd kan ”låse” en mutex og arbejde på en delt resurse og derefter ”låse den op” igen. Det kræver dog at alle tråde der arbejde på denne resurse bruger den samme mutex!

* Kun den tråd der låser kan låse mutex op igen!
* Andre der prøver at låse mutex bliver blokkeret.



En semaphore er noget lignende bortset fra at den tæller og ikke blot låser

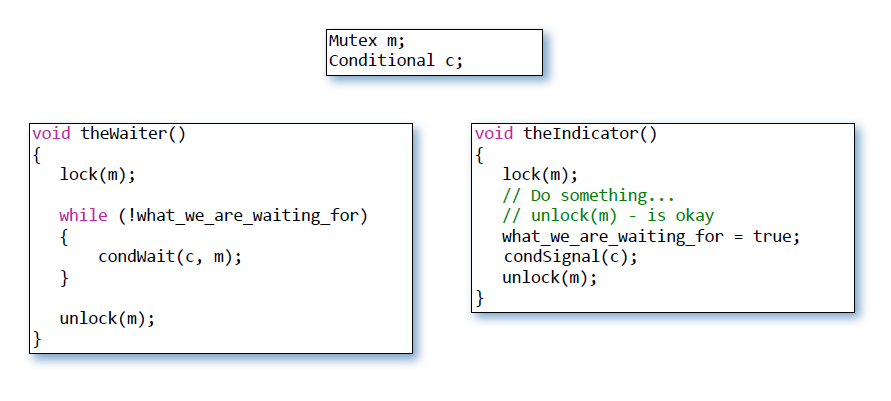
* Kan initieres med en værdi
* Tæller ned med take funktion
* Tæller op med post funktion
* Hvis semaphoren er 0 venter tråden ellers giver den los
* Kan bruges som signalerings mekanisme



## Mutex & Conditionals

Conditionals er en signalerings mekanisme der bruges sammen med mutex således at en tråd kan vente på en anden tråd

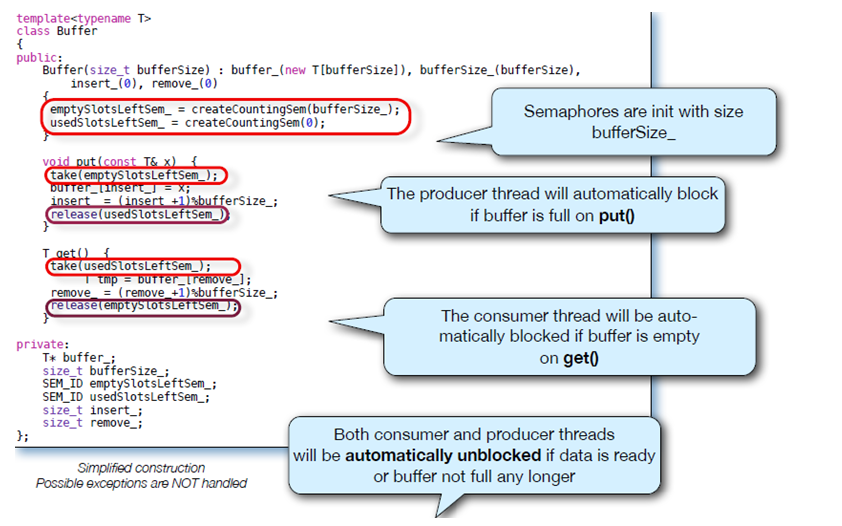
* Har altid et while loop rundt om conditional kaldet for at tjekke om data er klar da vi kan vågne uden dette er tilfældet.
* Eksempel? Flowchart?



## Producer / Consumer problem

Hvis en tråd er producent og den afleverer sit produkt i en buffer og der samtidigt er en forbruger så er vi nød til at sikre os mod at arrayet bliver henholdsvis fyldt/tømt.

* For situationen med 1 Producer og en consumer kan det løses med 2 semaphore
  + En der tæller hvor man items der er i bufferen
  + En der tæller hvor mange pladser der er tilbage i bufferen
  + Når der ligges data i bufferen tælles empty count en ned og fill count en op
  + Vice versa for når der fjernes data
  + VIRKER KUN MED 1 AF HVER!
* For flere consumere eller producenter skal der også bruges mutual exclution!
  + Efter semaphoren er talt ned tages mutex



## Dead locks

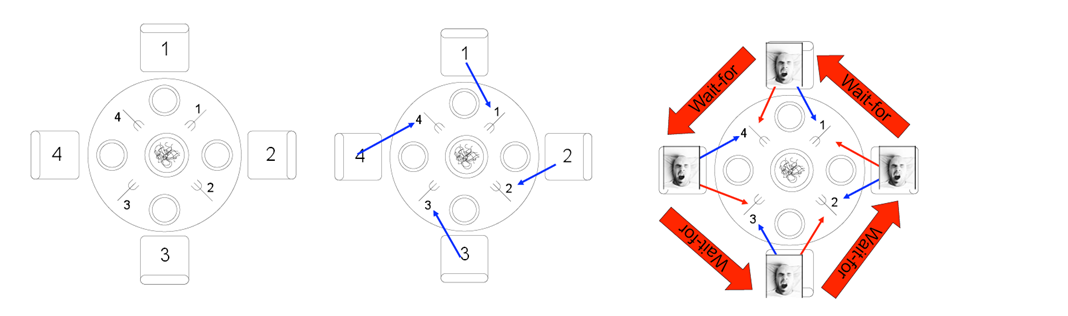
Der skal være fire ting opfyldt før der kan være en deadlocK:

1. Mutual exclution – resursen kan kun tilgåes af en adgangen
2. Hold and wait – Processer der allerede har taget en resurse beder om at få andre resurser
3. No preemption – Ingen resurse kan blive fratager sin ejer med magt
4. Circular wait condition – pi kan komme til at vente på en resurse pi+1 har.

## 

## Dinning Philosopher

Dinning Philosopher bruges til at illustrer en deadlock, der sidder fire philosopher rundt om et bord de skal spise men de har kun fire gafler og hver filosof skal bruge to gafler for at spise.



* Hvis alle filosofer samler deres højre gaffel op først kan dette give et deadock fordi ingen af dem vil kunne samle deres ventre gaffel op da den er i hånden på en anden filosof.
* Men hvis rækkefølgen som de samler gaflerne op i ændres til at 1 og 3 tager gaflen til højre først
* 2 og 4 tager gaflen til venstre først vil de ikke kunne gå i deadlock

