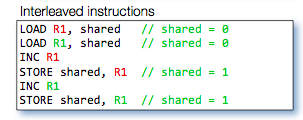
# ISU: 2. Synchronization and protection

## Data integrity - Concurrency challenge

* Shared data problem
* 
* Shared data må ikke blive tilgået af flere (tråde) på samme tid!

## Mutex & Semaphore

* Beskytter shared data ved at ”tage en nøgle/token” – ligesom når man går på toilettet
* Skal slippes igen når data er færdigbehandlet
* Kø til at tage mutex – FIFO
* Prioritet

**Mutex**

* Kun holderen af mutex’en kan slippe den
* Kræver to operationer – lock og unlock

**Semaphorer**

* Andre kan slippe semaphoren, selvom de ikke har den
* Kræver to operationer – take og release
* Kan have værdi større end én – counting semaphore og binary semaphore

## Mutex & Conditionals

* Signaleringsmekanisme brugt med mutex
* For at tjekke om en betingelse er opfyldt – f.eks. park-a-lot
* Take lock 🡪 check conditional (og bool) 🡪 if false, release lock and wait until signaled
  + Spurious wakeups! Dobletjek

🡪 Flytte afhængighed fra programmøren til en template class f.eks. scoped locking idiom

* Højere abstraktionsniveau
* Contructor 🡪 lock
* Destructor 🡪 unlock
* Dette fjerne også problemet med oprydning og undgår deadlocks ved fejl

## Producer / Consumer problem

* Producer putter noget i en fuld buffer?
* Consumer læser noget fra en tom buffer?
* 🡪 Kan løses med semaphorer 🡪 venter på semaphore til enten ikke tom eller ikke fuld

2 Semaphorer med størrelse på buffer:

* Producer tager emptySlotsLeft(-) og slipper usedSlotsLeft(+) – Skal vente på at empty > 0
* Consumer tager usedSlotsLeft(-) og slipper emptySlotsLeft(+) – Skal vente på at used > 0

## Dinning Philosophers

* Multiple mutexes

## Dead locks

* Situation hvor to (eller flere) tråde venter på at den anden skal frigive en ressource
  + Det kommer aldrig til at ske
* For at opnå deadlock:
  + Mutex beskyttet
  + En tråd der venter på en mutex, skal allerede holde en anden mutex
  + Non pre-emtive system
  + Cirkulær betingelse: t0 holder a og venter på b, t1 holder b og venter på a.

**Dning pholosophers:**

* En filosof skal have to gafler – først tager han den til venstre og derefter den til højre
* 1 tager a
* 2 tager b
* 3 tager c
* 4 tager d
* 1 forsøger at tage d – men må vente
* 2 forsøger at tage a – men må vente
* 3 forsøger at tage b – men må vente
* 4 forsøger at tage c – men må vente
* 🡪 Deadlock

Forkert brug af semaphores og mutexes kan lede til deadlock – f.eks. hvis aldrig initieret korrekt eller while(1) i en lås – eller fejl

Løsning på deadlock:

* Fjern en af de 4 parametre
* Dining philosophers: bryd den cirkulære betingelse – 1+3 starter med højre, 2+4: venstre

### Problemer

* Priority inversion – Det sker måske aldrig, måske lige pludseligt – you never know
* HP1 tråd bruger lås, HP2 tråd bliver sat til at vente, LP tråd tilgår lås lige idet den slippes. LP tråd får lås, hvorved HP2 tråd kommer til at vente længere tid 🡪 priority inversion
* Kan løses vha. mutexes og priority
* Semaphorer understøtter ikke prioriteter

Re-entrant????????????????