# Threads 02

|  |  |
| --- | --- |
| **Læringsmål** | Du kan:   * **1Pf5:** anvende centrale metoder og teknikker til at designe og konstruere programmer som samarbejdende processer/tråde * **1Tf1:** anvende centrale metoder og redskaber til synkronisering af processer og tråde |
| **Forventet læringsudbytte [SOLO]** | Programmeringssporet:   * [Multistrukturel] Du kan nævne flere kendetegn omkring tråde og race condition |
| **Din forberedelse** | Programmeringssporet:   * Benyt [Threading in C#](http://www.albahari.info/threading/threading.pdf) (PDF) fra tidligere forberedelse (Ex53)   + Læs om ”Synchronization Essentials” (side 19 – 21)   + Læs om ”Locking” (side 21 til 27) * [Threading with Monitor in C#](https://www.c-sharpcorner.com/UploadFile/1d42da/threading-with-monitor-in-C-Sharp/) * [Threading with Semaphore in C#](https://www.c-sharpcorner.com/UploadFile/1d42da/threading-with-semaphore-in-C-Sharp/) * [Interlocked](https://www.dotnetperls.com/interlocked) * [Protecting shared resources from concurrent access in multithreading](https://www.youtube.com/watch?v=pjzplzR8E3U) (video: 11:24) * [C# syntactic sugar - the lock statement](https://possiblythemostboringblogever.blogspot.com/2013/03/c-syntactic-sugar-lock-statement.html) * [Difference between Monitor and lock in C#](https://www.youtube.com/watch?v=N9jJKPVxD7c) (video: 3:54) * [Producer–consumer problem](https://en.wikipedia.org/wiki/Producer%E2%80%93consumer_problem) * Genlæs / gense materiale fra den sidste opgave |

I dagens opgave skal du arbejde med synkronisering mellem tråde. Det kan være svært at overskue de forskellige muligheder, men husk at det primært handler om forskellige måder at styre, hvordan mange tråde kan enes om at bruge en fælles ressource og bl.a. undgå ’race condition’, dvs. hvor to eller flere tråde kan tilgå/ændre fælles data samtidigt og skabe en utilsigtet effekt.

# Dagens ord

“Sometimes it is the people no one can imagine anything of   
who do the things no one can imagine”  
[Alan Turing](https://en.wikipedia.org/wiki/Alan_Turing)

# Øvelse 1: Terminologi

Del teamet op i to mindre grupper, og brug **Ordet rundt** til at reflektere over begreberne ”Producer”, ”Consumer”, ”Interlocking”, “Monitor” og “Semaphore”. Sørg for, at alle får mulighed for at tale.

*Tidsramme: 20 minutter*

# Klargøring

Udfør følgende:

* Hent kodedokumentet ’Ex54-Threads02.Kode.docx’ i Git materialefolderen.

Dokumentet indeholder nogle kodeudsnit, som du skal bruge i de følgende øvelser.

# Øvelse 2: Delt ressourceproblem

Find ’Kode udsnit 1’ i kodedokumentet.

Programmet simulerer kørsel af registreringer fra et antal vejrstationer (WEATHERSTATIONS). Hver vejrstation, repræsenteret af en tråd, har en sensor, som foretager et antal målinger (MEASURES) af en given værdi (VALUE). For at gøre kørslen deterministisk er værdien det samme hver gang (1.0). Værdierne summeres, så for hver kørsel burde summen være (WEATHERSTATIONS \* MEASURES \* VALUE). Hvis *accum* ikke har denne værdi for en kørsel udskrives forskellen.

Udfør følgende:

* Opret et nyt konsolprojekt
* Kopier kodeudsnittet (’Kode udsnit 1’) over til VS og kør det
  + Forklar resultatet[[1]](#footnote-1)
* Løs problemet ved hjælp af låsning
  + Du har løst problemet, når der blot står: *”Press a key …”*

**Hint:** Overvej nøje, *hvor* problemet opstår, og *hvorfor*.

# Øvelse 3: Interlocked

Problemet i øvelse 2 var, at en anden tråd kunne ”bryde ind” og manipulere accum fra accum blev læst (og gemt i temp) til accum blev tildelt sin nye værdi. Denne delte ressource skulle trådsikres, for at undgå synkroniseringsproblemet, ved hjælp af en lås.

Nu skal du se lidt længere nede i kodedokumentet. Se på ‘kodeudsnit 2’. Programmet viser (forhåbentligt), at operationerne X++ og X-- heller ikke er trådsikre.

To tråde (*adder* og *subtractor*) manipulerer *Number* samme antal gange. Den samlede effekt burde altså være at *Number* er lig med nul efter kørslen. Forhåbentlig viser en kørsel af programmet, at dette ikke er tilfældet[[2]](#footnote-2).  
En ofte anvendt ”løsning” på denne problematik er at benytte en lås i definitionen af get og set:

public Int64 Number {get { lock (key) { return \_number; } } set { lock (key) { \_number = value; } } }

En hurtig test viser dog, at det heller ikke er trådsikkert.

Udfør følgende:

* Forklar, hvorfor “løsningen” ikke er trådsikker
* Løs problemet vha. klassen *InterLocked*. (se forberedelsen til i dag).

# Øvelse 4: Producer/Consumer

I denne øvelse skal du arbejde med tråde og producer/consumer-problematikken.

## Øvelse 4.1: Forberedelse

* Opret en ny konsolapplikation i Visual Studio IDE
* Brug koden fra ‘Kode udsnit 3’ i kodedokumentet til at implementere de nødvendige klasser:
  + Program.cs
  + Producer.cs
  + Consumer.cs
  + Buffer.cs
  + Car.cs
* Sørg for at indsætte de korrekte namespaces i din kode, så den kan kompileres (bygges ikke køres) uden fejl
* Inspicer koden

## Øvelse 4.2: Trådsikring af Put()- og Get()-metoderne i Buffer-klassen

I denne øvelse skal du gøre Put()- og Get-metoderne i Buffer-klassen trådsikre ved hjælp af Monitor-klassen. Buffer-klassen har allerede en bufferLock, som du kan bruge.

Krav til implementeringen:

* Du skal aflåse den kritiske sektion, som er tilgangen til køen bufferData, ved hjælp af Monitor.Enter() og Monitor.Exit().
* Du skal benytte Monitor.Wait, når køen er tom og Get() bliver kaldt fra en Consumer, og benytter Monitor.Pulse() for at vække en ventende tråd, når en ny Car er blvet tilføjet.
* Du skal benytte Monitor.Wait(), når køen er fuld og Put() bliver kaldt fra en Producer, og benytte Monitor.Pulse() for at vække en ventende tråd, når Car er blevet fjernet.
* Du må ikke bruge ”Lock”
* Du skal benytte IsFull() og IsEmpty() metoderne

1. Hvis du ikke oplever fejl, så fjern kommenteringen af Thread.Sleep(1) i Sensor() og mindsk antallet af vejrstationer til 10. [↑](#footnote-ref-1)
2. Hvis ikke du oplever fejl, så prøv at skift fra ”Debug” til ”Release”. Hjælper det ikke, må du stole på, at koden ikke er trådsikker. [↑](#footnote-ref-2)