Videregående softwareudvikling

Trådnings opgaver

Thread

C++ har siden C++11 haft klassen `std::thread` som kan bruges til at få vores programmer til at arbejde på flere tråde hvis systemet understøtter det. Vi bruger klassen ved at kalde constructoren og give den en callable samt de argumenter som callable skal bruge.

```
#include<thread>
std::thread thread_object(callable, args ...)
```

Når en thread er instantieret kalder den callable med argumenterne med det samme og eksisterer indtil callable returnerer.
Læs om thread Klassen: https://www.cplusplus.com/reference/thread/thread/

Opgave 1 - Bingo race

Skriv et program der instantierer 3 thread objekter (Tom, Claire og Jason) som hver især vil være den første til at råbe ”Bingo!” ”;

Giv hver af bingodeltagerne en af 3 forskellige callables der alle gør det samme. Hver callable skal printe ”<navn> raaber ”Bingo!” ”;
eks: `std::cout << "Tom Raaber \"Bingo\"" << std::endl;`

- Giv Tom en funciton pointer.
- Giv Claire en lambda.
- Giv Jason en functor/function object.

Kør funktionen et par gange og se hvem der vinder. Ser udskriften nogle gange underligt ud?

Sidder du fast I opgaven? Se eksemplet her:
https://www.geeksforgeeks.org/multithreading-in-cpp/

Mutex

En anden klasse der er god at kende er `std::mutex`. Ofte når vi arbejder med threads, kan der forekomme en race condition, som er når 2 threads prøver at tilgå den samme data på samme tid, og det kan have udefineret adfærd hvis det ikke tages højde for.

Der findes flere forskellige slags mutex som kan forskellige ting. En god idé er at tage et kig på lock_guard og unique_lock som er wrapper klasser til at håndtere en mutex.

Læs om mutex: https://www.cplusplus.com/reference/mutex/mutex/

Opgave 2 - Talebamse

Vores 3 bingo spillere spiller ikke længere bingo, men har svært ved ikke stadig at råbe i munden på hinanden.

Brug eksemplet fra før, men tilføj et `std::mutex` objekt "talebamse", som kan hjælpe med at holde styr på hvem der snakker.

Få deltagerne til at vente på at få bamsen før de får lov at sige noget.
Er der forskel på outputtet i forhold til sidste opgave?

Conditional_variable

Hvis vi har brug for at have en thread til at vente på en anden thread, kan det gøres ved brug af `std::conditional_variable`. For at vække den ventende thread, skal en anden thread både ændrer på en delt variabel og give den sovende thread besked. En thread kan sættes til at vente på en anden ved at skaffe en `std::unique_lock<std::mutex>` som bruger samme mutex som har til formål at beskytte den delte variabel, og derefter kalde conditional_variable functionen wait.

Læs om conditional_variable:
http://www.cplusplus.com/reference/condition_variable/condition_variable/

Opgave 3 - Asynkrone hooligans

En gruppe hooligans øver sig på deres kampråb. En leder styrer råbet. Hver hooligan der ikke er lederen får tildelt et bogstav fra strengen ”Go Team!”. Lederen kalder på hvert bogstav eks.

```
std::cout << "Give me a \"" << char << "\"" << std::endl;
```

Derefter svarer en tilfældig hooligan med sit bogstav, eks.

```
std::cout << char << "!" << std::endl;
```

Dette fortsætter til hver hooligan har sagt sit bogstav én gang. Når alle hooligans har sagt deres bogstav, så siger lederen.

```
std::cout << "What does that say?" << std::endl;
```

Alle hooligans siger så deres bogstav på samme tid.

Eks. Output:

```
Give me a "G"
G!
Give me a "o"
o!
Give me a " "
 !
Give me a "T"
T!
Give me a "e"
m!
Give me a "a"
!!
Give me a "m"
e!
Give me a "!"
a!
What does that say?
aT eomG!
```

Atomic

Atomic objektet er en template klasse som wrapper en variabel og sørger for at den aldrig kan forsage en race condition. Tilgang af atomic objektet sørger for at tilgang til den wrappede værdi altid vil blive udført sekventielt mellem threads.

Læs om Atomic: http://www.cplusplus.com/reference/atomic/atomic/

Opgave 4 - Sympatiske væddeløbsheste

Et ræs mellem 3 heste på en afstand af 10 meter afholdes. Hestene sættes i gang på samme tid. Når en hest er nået en meter, venter den på at alle andre heste i løbet er nået lige så langt som den selv er før den fortsætter. Hestene går ikke op i hvem der vinder.

Få hver hest til at printe ud hvor langt den er nået samt en form for id, så vi kan følge med i hvor langt hver hest er nået.

Eks. Output:

```
Starting Race!
Horse with id 1 start Running
Horse with id 1 has reached distance 1
Horse with id 2 start Running
Horse with id 2 has reached distance 1
Horse with id 0 start Running
Horse with id 0 has reached distance 1
Horse with id 0 has reached distance 2
Horse with id 2 has reached distance 2
Horse with id 1 has reached distance 2
Horse with id 1 has reached distance 3
Horse with id 2 has reached distance 3
Horse with id 0 has reached distance 3
Horse with id 0 has reached distance 4
Horse with id 2 has reached distance 4
Horse with id 1 has reached distance 4
Horse with id 1 has reached distance 5
Horse with id 2 has reached distance 5
Horse with id 0 has reached distance 5
Horse with id 0 has reached distance 6
Horse with id 2 has reached distance 6
Horse with id 1 has reached distance 6
Horse with id 1 has reached distance 7
Horse with id 2 has reached distance 7
Horse with id 0 has reached distance 7
Horse with id 0 has reached distance 8
Horse with id 2 has reached distance 8
```

```
Horse with id 1 has reached distance 8
Horse with id 1 has reached distance 9
Horse with id 2 has reached distance 9
Horse with id 0 has reached distance 9
Horse with id 0 has reached distance 10
Horse with id 2 has reached distance 10
Horse with id 1 has reached distance 10
```