# RTOS

## Doel

Het doel van dit onderzoek is om er achter te komen hoe RTOS precies in elkaar zit en hoe wij hier mee kunnen communiceren

## Hypothese

Hoe werkt RTOS?

## Werkwijze

Powerpoints door kijken en er uit halen wat het meest nuttige is

## Resultaten

Een klasse voor de communicatie via de seriële poort

* + LibSerial

Een RTOS in de vorm van een bibliotheek van klassen

* + met coöperatieve multitasking (task)
  + concurrency mechanismen (pool, flag, …)
  + timer
  + clock
  + de taak blokkeert zichzelf (**suspend()**, **sleep()**, **release()**en **wait()**, **read()**en **write()**op een concurrency mechanisme);
  + de taak deblokkeert een taak met een hogere prioriteit (**resume()**en **set()**, **signal()**, **read()**en **write()**op een concurrency mechanisme);
  + het RTOS constateert dat een **clock** of **timer** vuurt waarop een taak met hogere prioriteit staat te wachten (dit kan alleen optreden als zijeffect van een van de vorige calls).
* Het RTOS vereist het gebruik van de compiler vlag **-std=c++11**
  + zie ook de bijgeleverde make file
  + Dit vereist de gcc/g++ versie 4.9.x compiler

aanwezig in jessie-distributie van Raspian



## Conclusie

Het onderzoek heeft naar voren gebracht dat en gevaren zijn met deadlocks als de communicatie tussen de websocket en RTOS niet goed zijn ingesteld zijn.

## Evaluatie

Tijdens het maken van de software die met het RTOS moet praten moet er goed opgelet worden dat de taken zich zelf vrijgeven gezien RTOS met coöperatieve multitasking werkt.