41012 Programming for Mechatronic Systems

Class: Week 7

Introduction

- Last week: Standard Template Library
  - Containers
  - Parallels to Pointers / Arrays
- This week:
  - Threading / Data Management
  - C++11 Threading
  - Some Design Considerations

Why Threads?

- Do we need threading?
  - Accessing data?
  - Synchronisation?
  - Time critical (nondetermenistic / stochastic)
- Process v Thread
  - A process is a collection of threads and the associated program
  - https://www.youtube.com/watch?v=O3EyzlZxx3g

Threads

- Pass into threads (function to run on, any arguments)

```
std::thread(increase_global,1000)
```

- They share the same footprint, and would have access to data of function that call them

C++11 Thraeds

- C++ Threads = OS Threads
  - POSIX
  - Shared Memory

- Encapulate traditionally used threading
  - pthreads
  - Boost Threads

C++11 <atomic>

- Atomic types are types that encapsulate a value whose access is guaranteed to not cause data races and can be used to synchronize memory accesses among different threads.

- http://www.cplusplus.com/reference/thread/thread/thread/

Mutexes

- Header with facilities that allow *mutual exclusion* (mutex) of concurrent execution of critical sections of code, allowing to explicitly avoid data races.
  - http://www.cplusplus.com/reference/mutex/mutex/

Can we have race conditions?

- Best practise
  - Avoid locking a mutex and calling a user function (need to be just before accessing data)
  - Avoid locking two mutexes (keep same order) / hierarchy
  - Depends on data, copies are not cheap but sometimes unavoidable

- https://youtu.be/_N0B5ua7oN8

Conditional variable

- A *condition variable* is an object able to block the calling thread until *notified* to resume.

- http://www.cplusplus.com/reference/condition_variable/condition_variable/
  - Allows to run threads in certain order at certain parts of thread

- Example: https://youtu.be/13dFggo4t_I