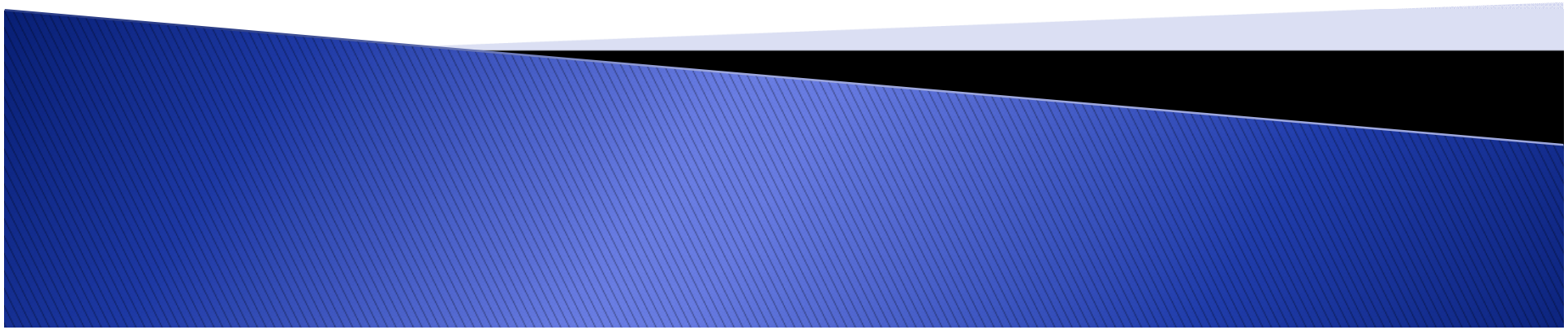
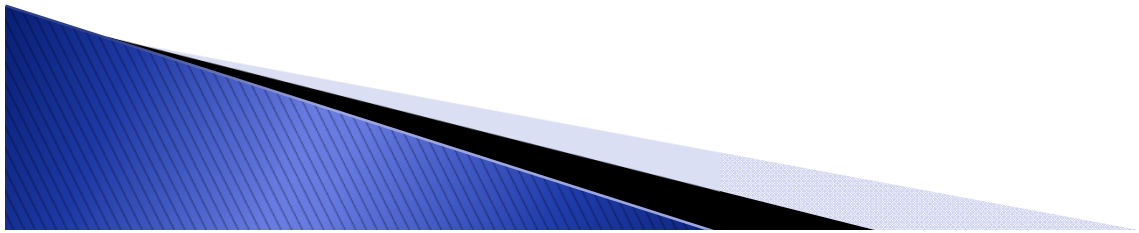
Multithreading in C++

Anton Örn Ívarsson
Pablo Albiol



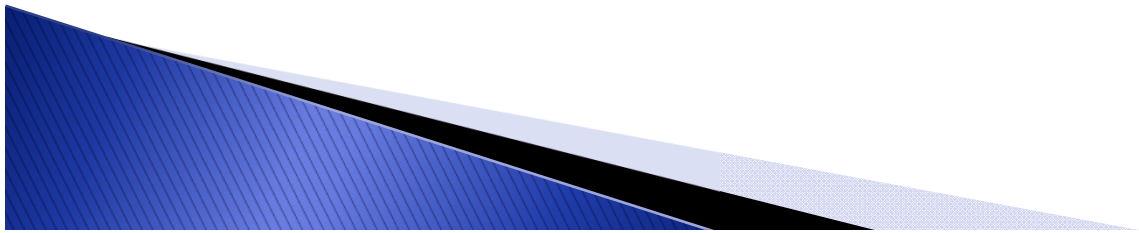
Overview

- Introduction to multithreading
  - General examples of applications.
- What is multithreading?
- How to multithread?
  - Libraries.
  - Functions.
- Mutex variables.
- Condition variables.

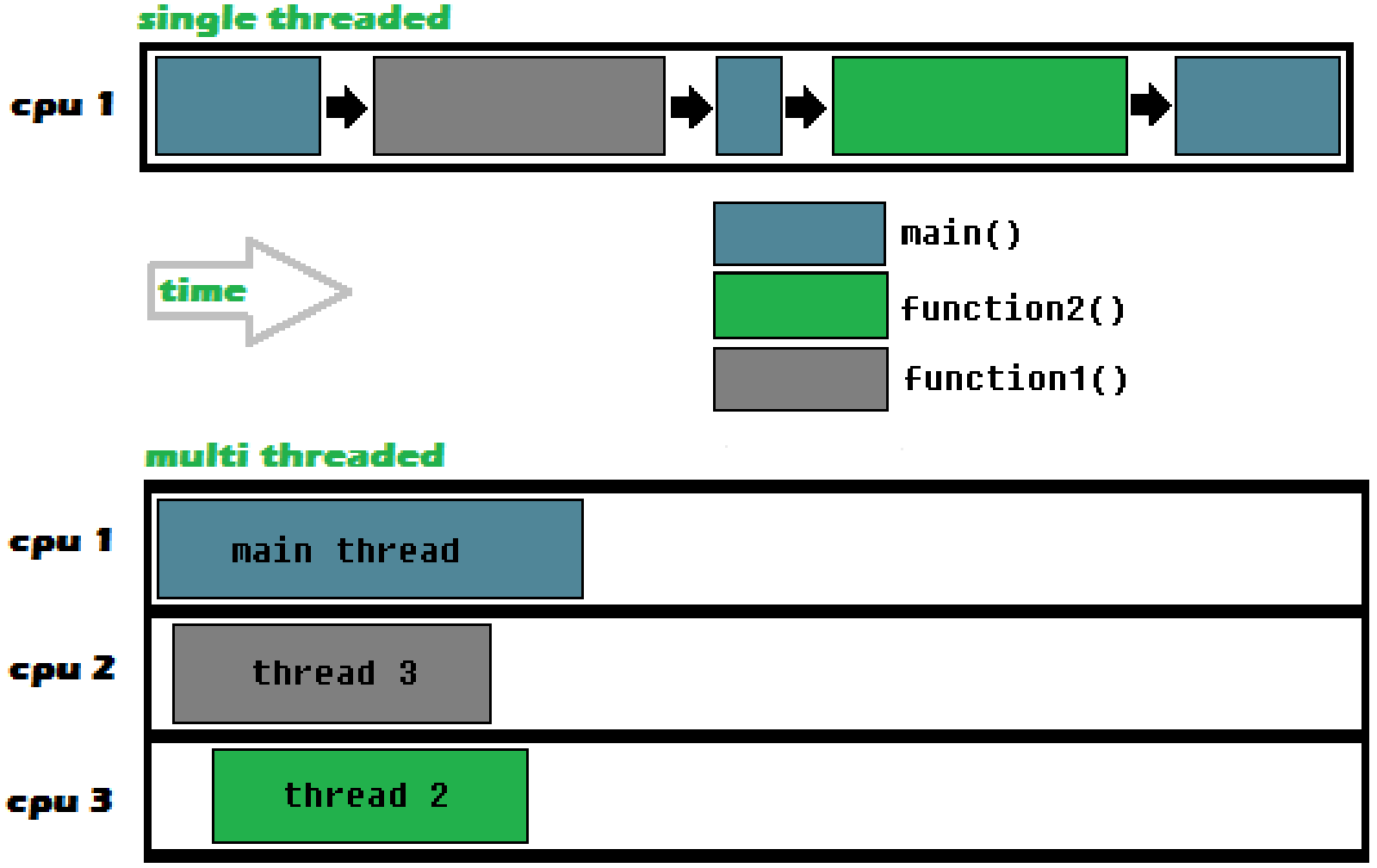


What is multithreading?

- MS Windows Task manager
  - Multiple processes.
    - Each process contains threads $\geq$ 1.
- Internet browser
  - Multiple pages open.
  - Multiple things happening on each pages.
- Almost every well functioning program.
- How does this work?

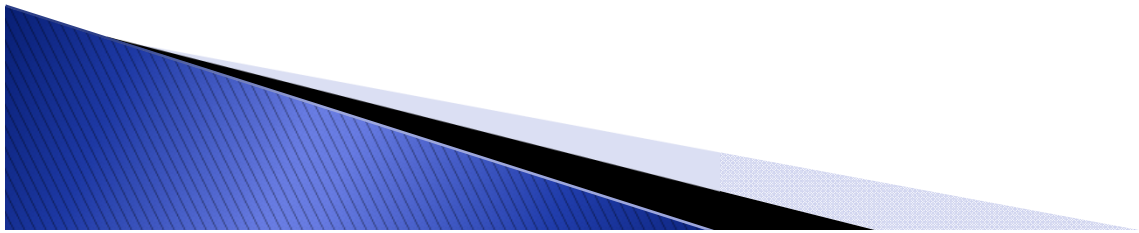


What is multithreading?



How to multithread?

- Libraries:
  - Linux:
    - POSIX threads
    - #include <pthread.h>
  - Windows:
    - Win32 threads.
      - #include <windows.h>
  - Wraps:
    - BOOST library
  - C++11
    - Supports multithreading through std.

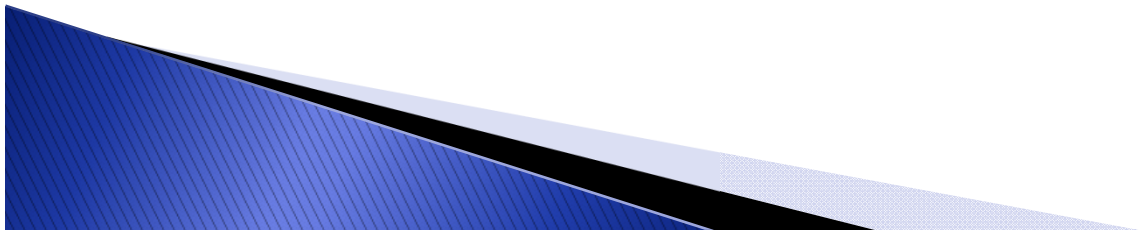


How to multithread?

- pthread_t t1, t2,…
- Create a thread:
  - pthread_create(*thread, *attr, void *(*start_routine)(void*), void *arg);
    - **thread** : The unique identifier for the thread. This identifier has to be of type pthread_t.
    - **attr** : Object which you can create for the thread with specific attributes for the thread. Can be NULL if you want to use the default attributes. Enough for most applications.
    - **start_routine** : The function that the thread has to execute.
    - **arg** : The function argument. If you don't want to pass an argument, set it to NULL.
    - **returns** : 0 on success, some error code on failure.

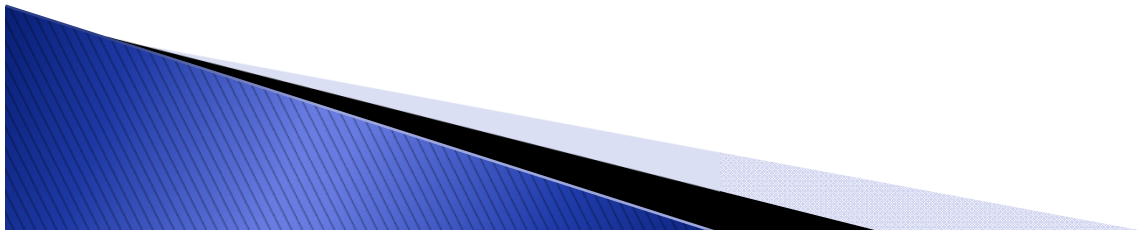
Examples

- Ex_NoThreads.cpp
- Ex_Threads.cpp



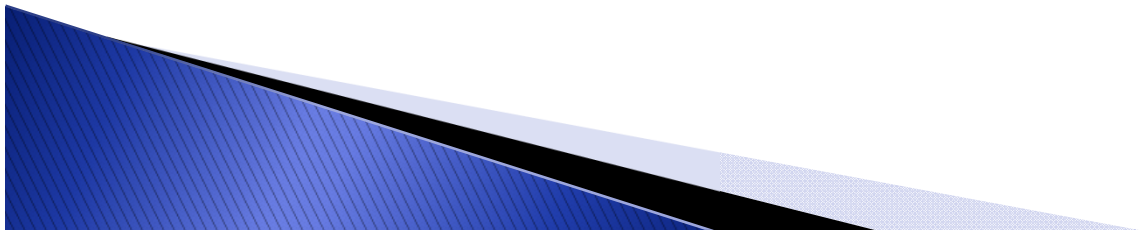
How to multithread?

- Terminate a thread
  - void pthread_exit(void *value_ptr);
    - **value_ptr** : The exit status of the thread. Can be set to NULL if you don't need to give it an exit status.



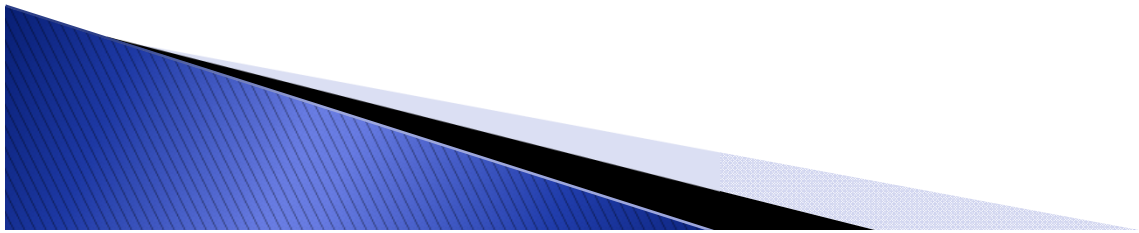
Example

- Ex2_threads.cpp



Joining threads

- Objective
  - Synchronizing treads. Wait for a thread to finish.
- How
  - Stopping execution of the code until a certain thread has been terminated.
  - Better than *pthread_exit.*

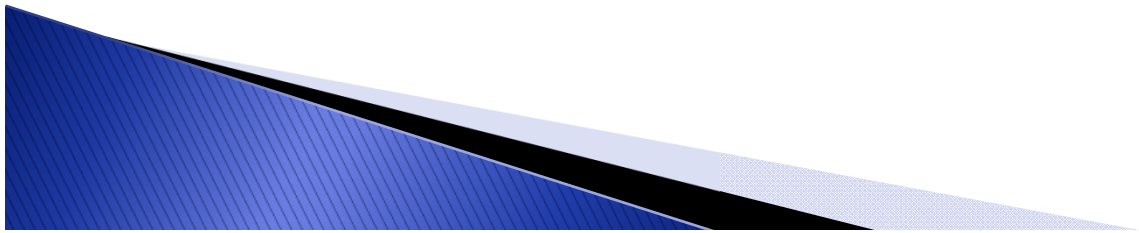


Joining threads

- Code

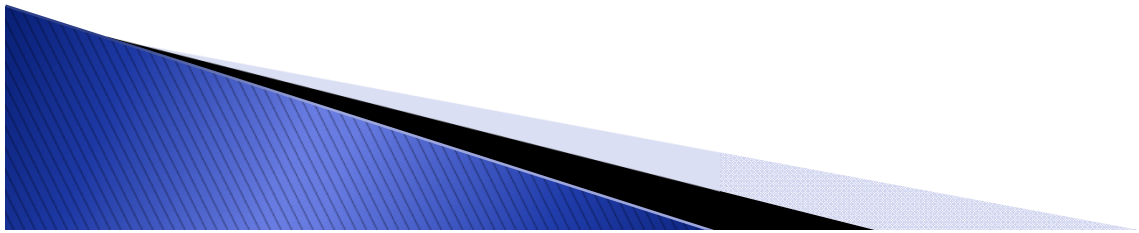
  *int pthread_join(pthread_t th, void **thread_return);*

- Parameters
  - th: Thread ID
  - Thread_return: Pointer to the value returned by the thread (by *pthread_exit(return_value)*).
- Example



Mutex variables

- Introduction
  - We want to share information (variables) between threads.
- Objective
  - Prevent threads from accessing a variable at the same time.
- How
  - Use Mutex variables. ”Semaphores”.
  - Mutexes don´t know which variables are controlling.



Mutex variables

- Code
  - Declaration

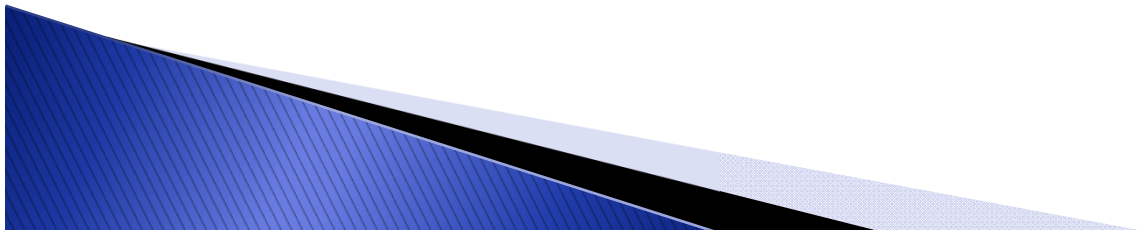
    pthread_mutex_t your_mutex_name = PTHREAD_MUTEX_INITIALIZER;
  - Lock/Unlock

    pthread_mutex_lock(your_mutex_name);

    …

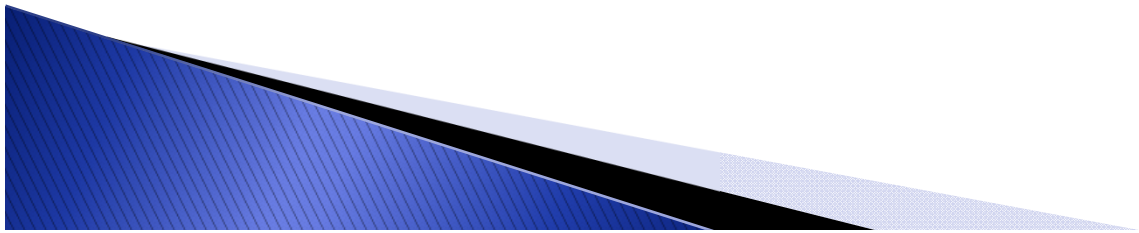
    …

    pthread_mutex_unlock(your_mutex_name);
- Example



Condition variables

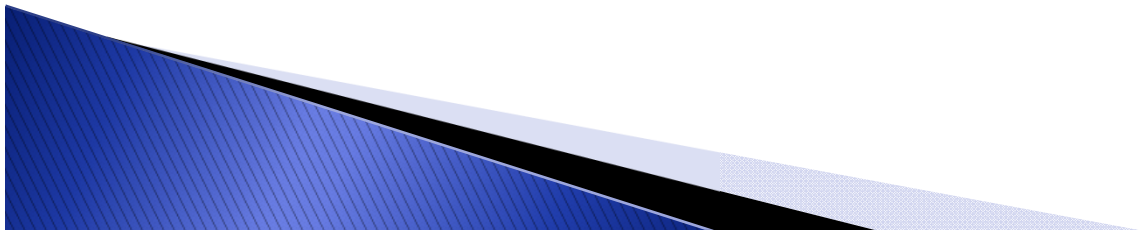
- Objective
  - Synchronizing treads depending on the value of certain variables.
- How
  - Mutexes are controlling the access to variables while condition variables control access to variables based on the value of variables.
  - A condition variable puts one thread on “wait” until it gets a signal from an other thread.
  - Condition variables are used together with mutexes.



Mutex variables

- Code
  - Declaration

    pthread_cond_t condition_var = PTHREAD_COND_INITIALIZER;
  - Wait
  - Signal
- Example



Conclussions and applications

- Powerful
- Mechatronics/robotics example