Threading in C++

Dr Ian Cornelius

Hello

Hello (1)

Learning Outcomes

1. Understand the concept of threading in C++
2. Demonstrate their knowledge on the use of threading in C++

Threading

Threading (1)

- **Recap**:
  - *Thread* refers to a basic unit of CPU utilisation
    - a separate process that has its own instructions and data
    - it may also represent a process that is part of a parallel program
      - although it may also represent an independent program
  - They share their code, data and other operating system resources with other threads belonging to the same process
  - A traditional process will have a single thread of control
    - if a process has multiple threads of control, then it has the ability to perform more than one task at a time
- Threading on C++ uses the following libraries:
  - `thread`: cross-platform compatible, limited functionality
  - `pthread.h`: Linux and MacOS compatible

`thread` Library

`thread` Library (1)

Creating a Thread

- The `thread` library is required and can be imported using the `#include` declaration
- The function `thread()` is used to create a thread

```
#include <thread>
std::thread(func, arg);
```

- Accepts two parameters:
  - `func`: the function that will be threaded
  - `arg`: the arguments that need to be passed through to the threaded function
- The `join()` function is used to wait for a thread to finish
  - must be called exactly one for each thread
  - must be called before a thread is *destroyed*

`thread` Library (2)

Example: Creating and Terminating a Thread

```cpp
#include <thread>
#include <iostream>
void threaded_function(const std::string& text) {
    std::cout << std::endl << std::endl << "[Thread ID: " << std::this_thread::get_id() << "] " << text;
}
int main() {
    std::thread t = std::thread(threaded_function, "Threading is fun in 5062CEM!");
    std::thread t2 = std::thread(threaded_function, "Threading is also fun in 5069CEM!");
    t.join();
    t2.join();
    return 0;
}
```

```
[Thread ID: 140511885125312] Threading is also fun in 5069CEM!

[Thread ID: 140511893518016] Threading is fun in 5062CEM!
```

`thread` Library (3)

`this_thread` Namespace

- The namespace `this_thread` can be used access properties for a given thread
  - `get_id()`: returns the ID of the running thread
  - `sleep_until()`: sleeps for a given amount of time using `time_point`
  - `sleep_for()`: sleeps for a given amount of time using `duration`

`thread` Library (4)

Killing a Thread

- There is no official method of killing a thread using the `thread` library
- If necessary, the developer will provide their own solution

`pthread` Library

`pthread` Library (1)

Creating a Thread

- The `pthread.h` library is required and can be imported using the `#include` declaration
- The function `pthread_create()` is used to create a thread

```
#include <pthread.h>
pthread_create(thread, attr, start_routine, arg);
```

- Accepts four parameters:
  - `thread`: an opaque and unique identifier for a new thread returned by the subroutine
  - `attr`: an opaque attribute object that can be used to set the thread attributes
  - `start_routine`: the C++ routine that will execute the thread once it has been created
  - `arg`: a single argument that may be passed to the `start_routine`
- Terminating a thread is achieved by calling the `pthread_exit()` function
  - often called when the thread has completed its work
  - used when the thread is no longer required to exist

```
#include <pthread.h>
pthread_exit(status);
```

- Accepts a single parameter:
  - `status`: normally provided a `nullptr`

`pthread` Library (2)

Example: Creating and Terminating a Thread

```
#include <iostream>
#include <pthread.h>
#include <mutex>
#define NUM_THREADS 2
struct arg_struct {
    char* arg1;
};
void* threaded_function(void* arguments) {
    auto* args = (struct arg_struct*) arguments;
    static std::mutex lock;
    std::lock_guard guard{lock};
    std::cout << "[Thread ID: " << pthread_self() << "] " << args->arg1 << std::endl;
    pthread_exit(nullptr);
```

```
[Thread ID: 139934411257536] Threading with 5062CEM is fun!
[Thread ID: 139934402864832] Threading with 5062CEM is fun!
```

`pthread` Library (3)

Joining Threads

- Joining multiple threads together is achieved with the `pthread_join()` function
- Call the thread until the thread terminates

```
pthread_join(threadId, returnStatus);
```

- Accepts two parameters:
  - `threadId`: the ID of the thread to be joined
  - `returnStatus`: a pointer to the location of the threads exit status

`pthread` Library (4)

Example: Joining Threads

- The attribute state for joining needs to be declared
- Threads can be joined together using the `pthread_join()`

```
#include <iostream>
#include <mutex>
#include <pthread.h>
#define NUM_THREADS 4
void* threaded_function(void* id) {
    static std::mutex lock;
    std::lock_guard guard{lock};
    std::cout << "Hello from Thread ID -> " << (long) id << std::
    pthread_exit(nullptr);
}
int main() {
```

```
Creating Thread -> 0
Creating Thread -> 1
Creating Thread -> 2
Creating Thread -> 3
Hello from Thread ID -> 1
Hello from Thread ID -> 3
Hello from Thread ID -> 2
Hello from Thread ID -> 0
```

`pthread` Library (5)

Detaching Threads

- Detaching threads from each other is achieved with the `pthread_detach()` function

```
pthread_detach(threadId, returnStatus);
```

- Accepts one parameter:
  - `thread`: the unique ID of the thread to be joined

Goodbye

Goodbye (1)

Questions and Support

- Questions? Post them on the **Community Page** on Aula
- Additional Support? Visit the Module Support Page
- Contact Details:
  - Dr Ian Cornelius, ab6459@coventry.ac.uk