Threads

Every process starts with a main thread. We can create multiple threads along with it.

Syntax:

#include<thread> -- Library to be linked

thread <thread\_name>(func\_name) -- declare a thread with a call to func\_name()

thread <thread\_name>{func\_name} -- same as above

thread <thread\_name>{func\_name, arg1, arg2} -- threading by passing arguments

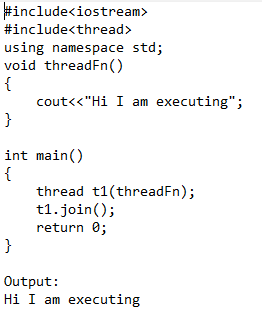
thread <thread\_name>{func\_name, ref(arg1)} -- Passing arguments by reference

<thread\_name>.join() -- main() func waits for the thread to complete and join.

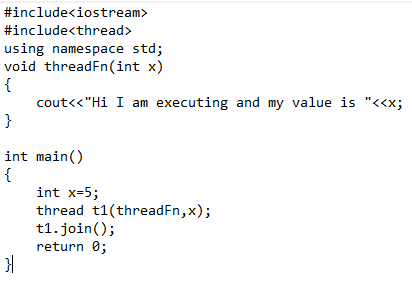
this\_thread::get\_id() -- returns the thread id.

this\_thread::sleep\_for(time) -- delay the execution for the sleep time

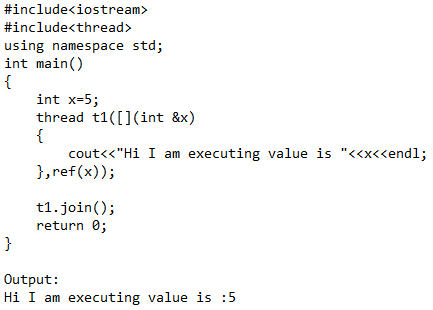
**Simple Demonstration**



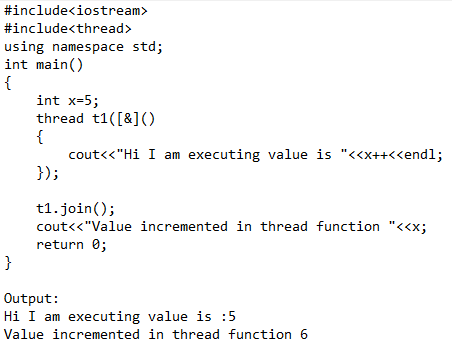
**Calling Thread Function by passing parameters**



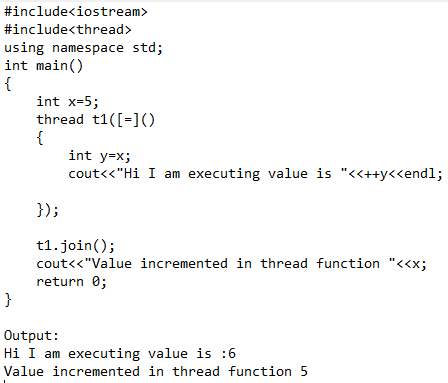
**Writing thread function in declaration**



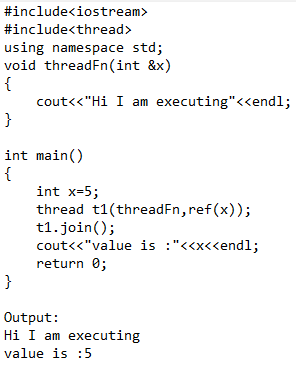
**Passing all local variables to the thread function as reference**



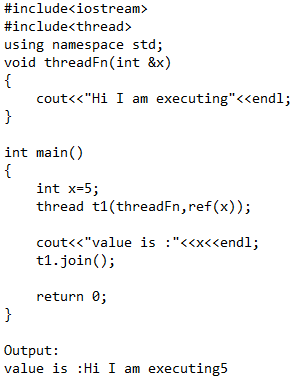
**Passing local variables but not as reference – cannot alter main function values**



**Using join – waits the thread to complete execution**



**Using join at incorrect places**

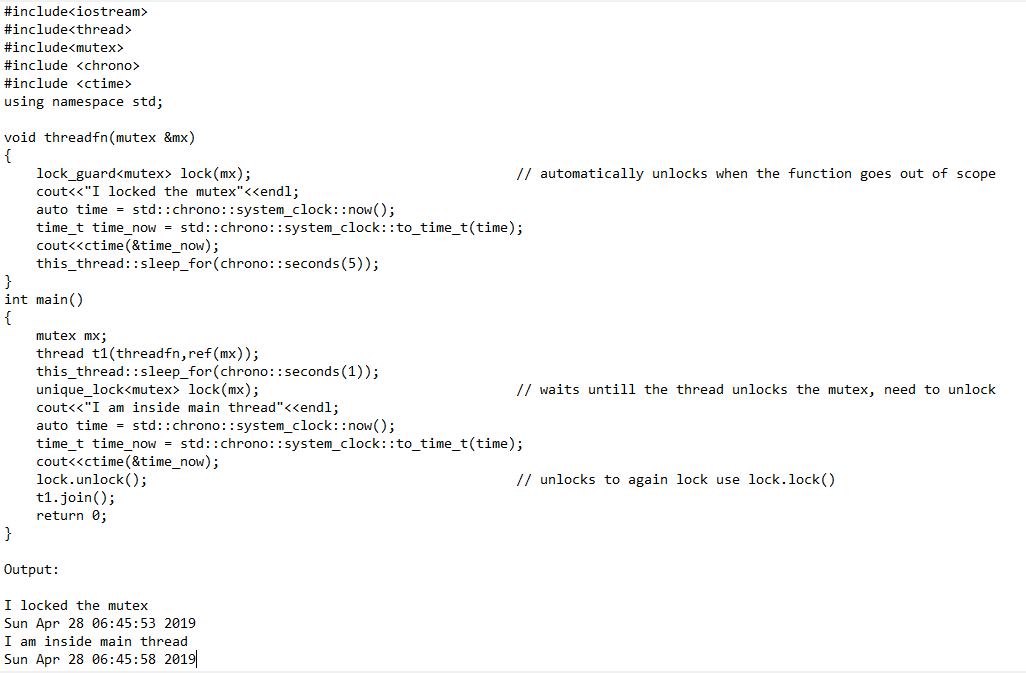


Packages supporting multithreading

Future

Promise

**Mutex lock**



Difference between lock\_guard and unique\_lock?

**lock\_guard** 🡪 automatically unlocks once the function goes out of scope. Cannot be used with condition variables.

unique\_lock 🡪 needs to be unlocked and can be locked and unlocked. Used with condition variables.

Mutex with condition variables

<https://www.youtube.com/watch?v=kdXGTGveme8>