Multithreading

VGP131-OOP II

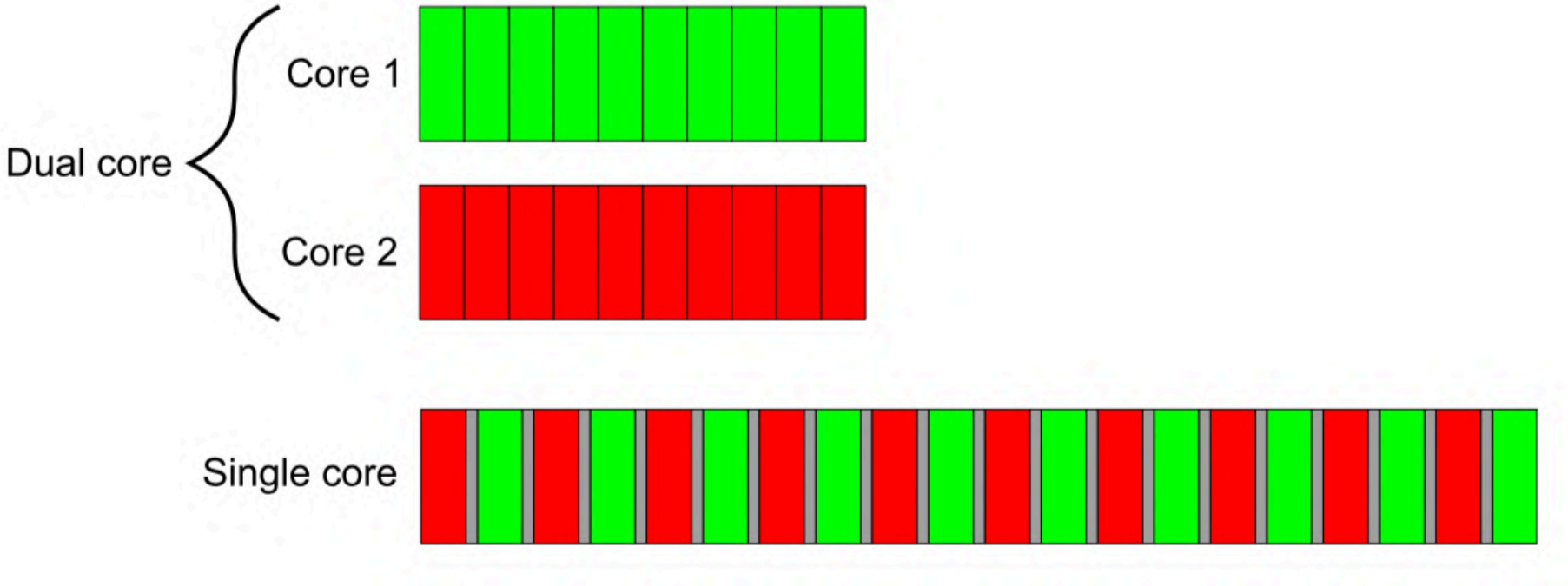
Instructor: Ivaldo Tributino

Concurrency in computer systems

- Concurrency in terms of computers means a single system performing multiple independent activities in parallel, rather than sequentially, or one after the other.
- Task switching: Computers with a single processing unit or core, can really only perform one task at a time, but it can switch between tasks many times per second that appear to be happening simultaneously.
- Whether they have multiple processors or multiple cores within a processor (or both), these computers are capable of genuinely running more than one task in parallel. We call this *hardware concurrency*.

Hardware Threads

- The figure shows an idealized scenario of a computer with precisely two tasks to do. On a dual-core machine, each task can execute on its own core. On a single-core machine doing task switching, the chunks from each task are interleaved.



CONCURRENCY WITH MULTIPLE THREADS

Threads are much like lightweight processes: each thread runs independently of the others, and each thread may run a different sequence of instructions.

All threads in a process share the same address space, and most of the data can be accessed directly from all threads – global variables remain global, and pointers or references to objects or data can be passed around among threads.

Although it's often possible to share memory among processes, this is complicated to set up and often hard to manage, because memory addresses of the same data aren't necessarily the same in different processes.

Getting started

Every C++ program has at least one thread, which is started by the C++ runtime: the thread running main().

Imperative that you ensure that the thread is correctly joined or detached, even in the presence of exceptions.

If you need to wait for a thread to complete, you can do this by calling join( ) on the associated std::thread instance.

Once you've called join(), the std::thread object is no longer joinable, and joinable( ) will return false.

Creating Threads

libraries.h | main.cpp | Makefile

main.cpp > function1()

```
12  void function1()
13  {
14      for(unsigned i =0; i<5; i++){
15          for(unsigned j=0; j<5; j++){
16              cout << '1';
17          }
18          cout << '\n' ;
19      }
20  }
21
22  void function2()
23  {
24      for(unsigned i =0; i<5; i++){
25          for(unsigned j=0; j<5; j++){
26              cout << '2';
27          }
28          cout << '\n' ;
29      }
30  }
31
32  int main(){
33
34      //************************************************
35      //         ---- Creating Threads -----
36      //************************************************
37
38      function1();
39      function2();
```

The function1( ) will be run first and then function2( ).

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
(base) Ivaldo:Week10 admin$ ./main
11111
11111
11111
11111
11111
22222
22222
22222
22222
22222
(base) Ivaldo:Week10 admin$
```

bash
bash

master*+ ⊗ 0 ⚠ 0 Live Share Ln 31, Col 1 Spaces: 4 UTF-8 LF C++ Mac

Creating Threads

main.cpp ● | library_1.h | library_2.h | thread_guard.h | threadsafe_queue.h | Polygon.cpp | Polygon.h

main.cpp > main()

```cpp
int main(){

    //*************************************************
    //         ----- Creating Threads -----
    //*************************************************

    // // function1();
    // // function2();


    std::thread t1(function1);
    std::thread t2(function2);

    t1.join();
    t2.join();

    if(!t1.joinable()){
        cout << "thread object is no longer joinable" << endl;
    }
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

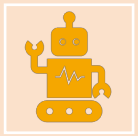
```
2222
111
11111
11111
11111
11111
2
22222
22222
22222
thread object is no longer joinable
(base) Ivaldo:Week10 admin$
```

performing two tasks simultaneously

Live Share | Ln 123, Col 1 | Spaces: 4 | UTF-8 | LF | C++ | Mac

Initializing thread with an object

In the previous example, we used regular function for the thread task. However, we can use any callable object such as lambda functions, functor and function pointer.



A functor is an object of a class that overloads operator **( ) –** function call operator.

Passing arguments to a thread function

- Passing arguments to the callable object or function is fundamentally as simple as passing additional arguments to the std::thread constructor. But it's important to bear in mind that by default the arguments are *copied* into internal storage, where they can be accessed by the newly created thread of execution, even if the corresponding parameter in the function is expecting a reference.
- To pass the message by reference, we must use **std::ref(data) –** A reference to the data will be passed instead of a reference to a copy of the data.
- We can use **move** to pass the parameter without copying and not sharing memory between the threads.

Using Functors

main.cpp | Polygon.h | Polygon.cpp | thread_guard.h | thread_safe_stack.h | library.h

main.cpp > main()

```
57      }
58
59      //************************************************
60      // - Initializing thread with an object -
61      //************************************************
62      cout << "--- Initializing thread with an object ---" << '\n';
63
64      {
65          Polygon square(4);
66          std::thread t3(square, 8);
67
68          if(t3.joinable()) t3.join();
69      }
70      cout << "- Using std::ref ---" << '\n';
71
72      {
73          Polygon square(4);
74          std::thread t3(std::ref(square), 8);
75
76          if(t3.joinable()) t3.join();
77      }
78
79      // cout << "--- USING REF AND RAII ---" << '\n';
80      // {
81      //     Polygon square(4);
82      //     {
83      //         std::thread t3(std::ref(square), 8);
84      //         thread_guard tg(t3);
85
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

```
--- Initializing thread with an object ---
Constructor Invoked
Copy Constructor Invoked
Copy Constructor Invoked
Polygon destroyed
Polygon destroyed
Polygon destroyed
- Using std::ref ---
Constructor Invoked
Polygon destroyed
(base) ivaldo:Week10 admin$
```

bash
bash

master*+ | 0 ⚠ 0 | Live Share | Ln 70, Col 30 | Spaces: 4 | UTF-8 | LF | C++ | Mac

Using RAII to wait for a thread to complete

- Resource Acquisition Is Initialization (RAII), holding a resource is a class invariant, and is tied to object lifetime: resource allocation (or acquisition) is done during object creation (specifically initialization), by the constructor, while resource deallocation (release) is done during object destruction (specifically finalization), by the destructor. In other words, resource acquisition must succeed for initialization to succeed.

- RAII idiom and provide a class that does the join() in its destructor, as in the following listing.

Thread Guard

libraries.h | main.cpp | thread_guard.h | Makefile

thread_guard.h > ...

```cpp

#include "libraries.h"

class thread_guard
{
    std::thread& _t;

public:
    //Explicit, that is, it cannot be used for implicit conversions and copy-initialization.
    explicit thread_guard(std::thread& t): _t(t) {}

    ~thread_guard()
    {
        if(_t.joinable()) _t.join();
        cout << "thread object was destroyed." << endl;
    }
    // to ensure that they're not automatically provided by the compiler.
    thread_guard(thread_guard const&)=delete;

    thread_guard& operator=(thread_guard const&)=delete;

};
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
11111
thread object is no longer joinable
(base) Ivaldo:Week10 admin$
```

bash
bash

master*+ 0 0 Live Share Ln 22, Col 3 Spaces: 4 UTF-8 LF C++ Mac

REF & RAII

main.cpp | library_1.h | library_2.h | thread_guard.h | threadsafe_queue.h | Polygon.cpp | Polygon.h

main.cpp > main()

```
125     cout << "thread object is no longer joinable" << endl;
126 }
127
128 //*************************************************
129 // - Initializing thread with an object -
130 //*************************************************
131 cout << "--- Initializing thread with an object ---" << endl;
132
133 {
134     Polygon square(4);
135     std::thread t3(square, 8);
136
137     if(t3.joinable()) t3.join();
138 }
139
140
141 cout << "--- USING REF AND RAII ---" << endl;
142 {
143     Polygon square(4);
144     {
145         std::thread t3(std::ref(square), 8);
146         thread_guard tg(t3);
147
148     }
149     cout << "--------------" << endl;
150 }
151
152
153
154
155
156
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

bash

```
--- USING REF AND RAII ---
Constructor Invoked
thread object was destroyed.
--------------
Polygon destroyed
(base) Ivaldo:Week10 admin$
```

0 ⚠ 0 | Live Share | Ln 146, Col 28 | Spaces: 4 | UTF-8 | LF | C++ | Mac

Transferring ownership of a thread

- We transfer the ownership of the thread by moving it.
- Putting std::thread objects in a std::vector is a step toward automating the management of those threads: rather than creating separate variables for those threads and joining with them directly, they can be treated as a group.

Std :: move

main.cpp | Polygon.h | Polygon.cpp | thread_guard.h | thread_safe_stack.h | library.h

main.cpp > main()

```cpp
//*************************************************
//   - Transferring ownership of a thread  -
//*************************************************
cout << "--- Transferring ownership of a thread ---" << '\n';
{
    std::thread t_1(function1);
    std::thread t_2 = std::move(t_1);
    if(!t_1.joinable()){
        cout << "thread object is no longer joinable" << '\n';
    }
    if(!t_2.joinable()){
        cout << "thread object is no longer joinable" << '\n';
    };
    thread_guard g(t_2);
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
--- Transferring ownership of a thread ---
thread object is no longer joinable
11111
11111
11111
11111
11111
thread object was destroyed.
(base) Ivaldo:Week10 admin$
```

bash
bash

master*+ 0 0 Live Share Ln 107, Col 1 Spaces: 4 UTF-8 LF C++ Mac

How Many Threads Should We Create?

- One feature of the C++ Standard Library that helps here is std::thread::hardware_concurrency( ). This function returns an indication of the number of threads that can truly run concurrently for a given execution of a program. On a multicore system it might be the number of CPU cores.

How many Threads?

main.cpp | Polygon.h | Polygon.cpp | thread_guard.h | thread_safe_stack.h | library.h

main.cpp > main()

```
91      {
92          std::thread t_1(function1);
93          std::thread t_2 = std::move(t_1);
94          if(!t_1.joinable()){
95              cout << "thread object is no longer joinable" << '\n';
96          }
97          if(!t_2.joinable()){
98              cout << "thread object is no longer joinable" << '\n';
99          };
100         thread_guard g(t_2);
101     }
102
103     //*************************************************
104     //   - std::thread::hardware_ concurrency()  -
105     //*************************************************
106     cout << "--- How Many Threads Should We Create? ---" << '\n';
107
108     {
109     unsigned int c = std::thread::hardware_concurrency();
110      cout << "Number of cores: " << c << '\n';;
111     }
112
113
114
115
116
117
118
119
120
121
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

```
11111
11111
11111
11111
thread object was destroyed.
--- How Many Threads Should We Create? ---
Number of cores: 4
(base) Ivaldo:Week10 admin$
```

bash
bash

master*+ | 0 ⚠ 0 | Live Share | Ln 106, Col 45 | Spaces: 4 | UTF-8 | LF | C++ | Mac

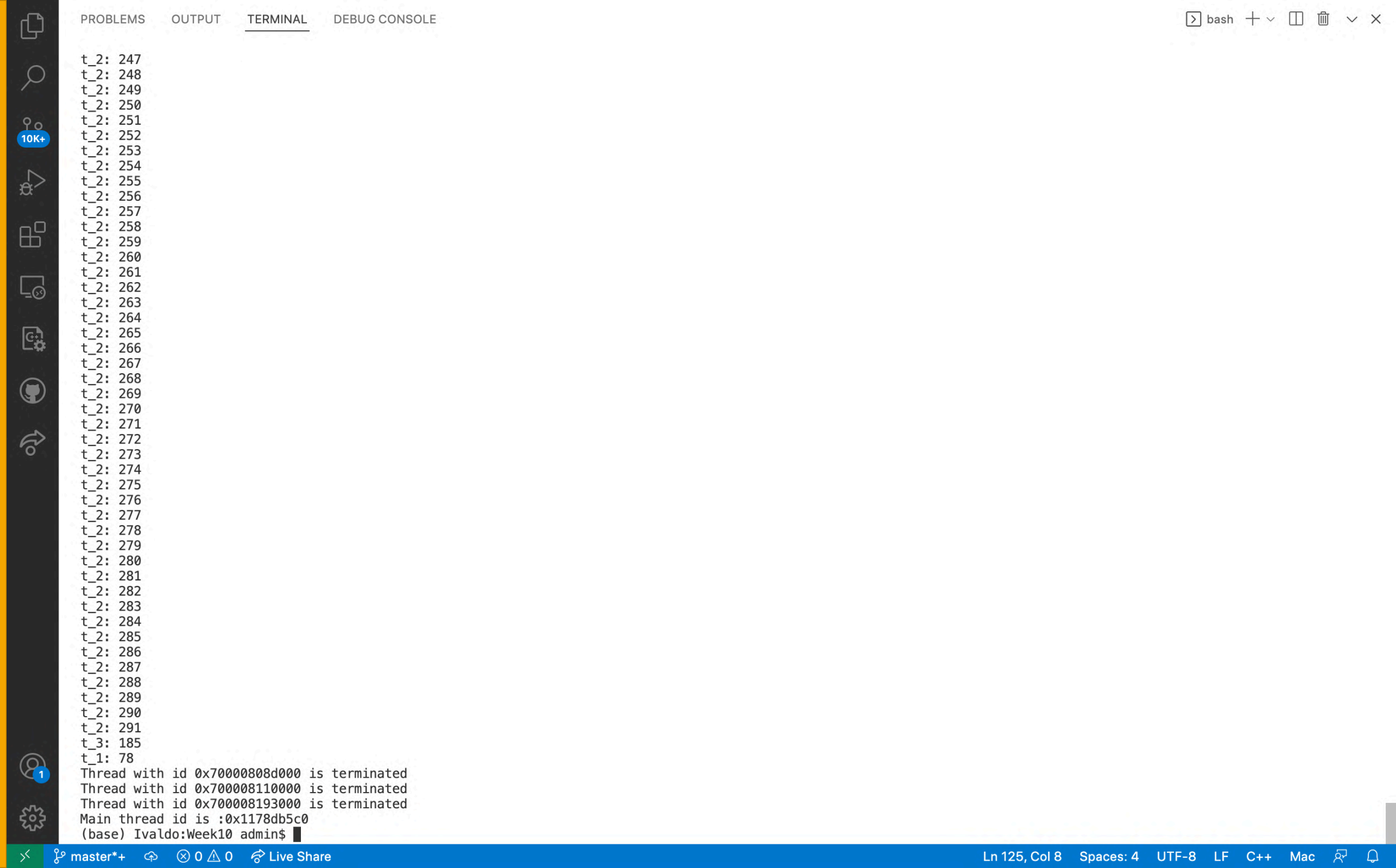
Thread id

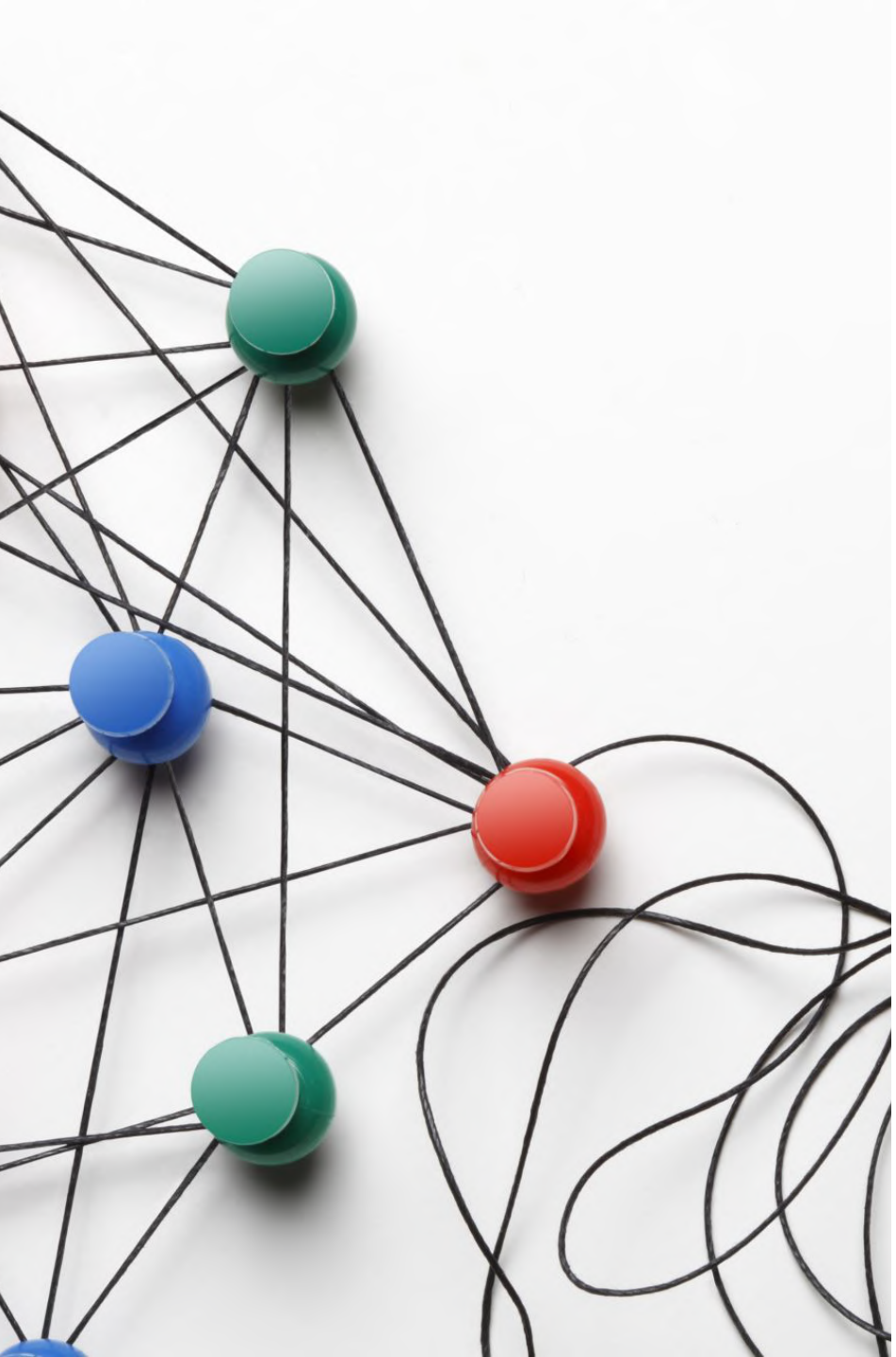
- Every thread has its unique identifier;
- Object of type std::thread::id;
- We obtained id calling the get_id();
- Provide a total ordering for all distinct values. This allows them to be used as keys in associative containers, or sorted, or compared in any other way that you as a programmer may see fit;
- The Standard Library also provides std::hash<std::thread::id> so that values of type std::thread::id can be used as keys in the new unordered associative containers too.

Threads ID?


```cpp
//*************************************************
//              ---    Thread ID ---
//*************************************************
cout << "--- Thread ID ---" << '\n';
using SC = std::chrono::steady_clock;
auto dealine = SC::now() + std::chrono::milliseconds(1);

// Accesses to atomic objects can establish synchronization between threads.
std::atomic<int> counter = 0;

//create 3 different threads
std::thread t_1([&](){
    while(SC::now()<dealine)
        printf("t_1: %d\n", ++ counter);

});
std::thread t_2([&](){
    while(SC::now()<dealine)
        printf("t_2: %d\n", ++ counter);

});
std::thread t_3([&](){
    while(SC::now()<dealine)
        printf("t_3: %d\n", ++ counter);

});
//get id of threads t_1, t_2 and t_3
std::thread::id id1 = t_1.get_id();
std::thread::id id2 = t_2.get_id();
std::thread::id id3 = t_3.get_id();

if(t_1.joinable())
{
    t_1.join();
    cout << "Thread with id " << id1 << " is terminated" << '\n';
}
if (t_2.joinable())
{
    t_2.join();
    cout << "Thread with id " << id2 << " is terminated" << '\n';
```

Threads ID?


PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
bash
t_2: 247
t_2: 248
t_2: 249
t_2: 250
t_2: 251
t_2: 252
t_2: 253
t_2: 254
t_2: 255
t_2: 256
t_2: 257
t_2: 258
t_2: 259
t_2: 260
t_2: 261
t_2: 262
t_2: 263
t_2: 264
t_2: 265
t_2: 266
t_2: 267
t_2: 268
t_2: 269
t_2: 270
t_2: 271
t_2: 272
t_2: 273
t_2: 274
t_2: 275
t_2: 276
t_2: 277
t_2: 278
t_2: 279
t_2: 280
t_2: 281
t_2: 282
t_2: 283
t_2: 284
t_2: 285
t_2: 286
t_2: 287
t_2: 288
t_2: 289
t_2: 290
t_2: 291
t_3: 185
t_1: 78
Thread with id 0x70000808d000 is terminated
Thread with id 0x700008110000 is terminated
Thread with id 0x700008193000 is terminated
Main thread id is :0x1178db5c0
(base) Ivaldo:Week10 admin$
master*+
0 0
Live Share
Ln 125, Col 8
Spaces: 4
UTF-8
LF
C++
Mac



Sharing data between threads

- If you’re sharing data between threads, you need to have rules for which thread can access which bit of data when, and how any updates are communicated to the other threads that care about that data.
- Incorrect use of shared data is one of the biggest causes of concurrency-related bugs.
- The most basic mechanism for protecting shared data provided by the C++ Standard is the *mutex*.

Using mutexes in C++

- In C++, you create a mutex by constructing an instance of std::mutex, lock it with a call to the member function lock(), and unlock it with a call to the member function unlock().
- The Standard C++ Library provides the std::lock_guard class template, which implements that RAII idiom for a mutex; it locks the supplied mutex on construction and unlocks it on destruction, thus ensuring a locked mutex is always correctly unlocked.

Mutex

main.cpp | Polygon.h | Polygon.cpp | thread_guard.h

main.cpp > main()

```
144
145     //************************************************
146     //         -- Using mutexes in C++ --
147     //************************************************
148     cout << "--- Using mutexes in C++ ---" << '\n';
149
150     vector<int> v;
151
152     std::thread thr_push(push,std::ref(v));
153     std::thread thr_pop(pop,std::ref(v));
154
155     if (thr_push.joinable()) thr_push.join();
156     if (thr_pop.joinable()) thr_pop.join();
157
158
159
160
161
162
163
164
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
--- Using mutexes in C++ ---
Push 0
Push 1
Push 2
Pop 1
Push 3
Push 4
Push 5
Pop 4
Push 6
Push 7
Push 8
Pop 7
Push 9
Pop 9
Pop 8
Pop 6
Pop 5
Pop 3
Pop 2
(base) Ivaldo:Week10 admin$
```

```
std::mutex m;
void push(vector<int>& v)
{
    // m.lock();
    for (int i = 0; i < 10; ++i)
    {
      cout << "Push " << i << '\n';
      std::this_thread::sleep_for(std::chrono::seconds(1));
      v.push_back(i);
    }
    // m.unlock();

}
void pop(vector<int>& v)
{
    // m.lock();
    for (int i = 0; i < 10; ++i)
    {
        if(v.size()!=0)
        {
            int val = v.back();
            v.pop_back();
            cout << "Pop " << val << '\n';
        }
        std::this_thread::sleep_for(std::chrono::seconds(3));
    }
    // m.unlock();
}
```

Mutex

main.cpp Polygon.h Polygon.cpp thread_gu

main.cpp > ...

```
144
145     //*************************************************
146     //         -- Using mutexes in C++ --
147     //*************************************************
148     cout << "--- Using mutexes in C++ ---" << '\n';
149
150     vector<int> v;
151
152     std::thread thr_push(push,std::ref(v));
153     std::thread thr_pop(pop,std::ref(v));
154
155     if (thr_push.joinable()) thr_push.join();
156     if (thr_pop.joinable()) thr_pop.join();
157
158
159
160
161
162
163
164
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
--- Using mutexes in C++ ---
Push 0
Push 1
Push 2
Push 3
Push 4
Push 5
Push 6
Push 7
Push 8
Push 9
Pop 9
Pop 8
Pop 7
Pop 6
Pop 5
Pop 4
Pop 3
Pop 2
Pop 1
Pop 0
(base) Ivaldo:Week10 admin$
```

```
void push(vector<int>& v)
{
    m.lock();
    for (int i = 0; i < 10; ++i)
    {
      cout << "Push " << i << '\n';
      std::this_thread::sleep_for(std::chrono::seconds(1));
      v.push_back(i);
    }
    m.unlock();

}
void pop(vector<int>& v)
{
    m.lock();
    for (int i = 0; i < 10; ++i)
    {
        if(v.size()!=0)
        {
            int val = v.back();
            v.pop_back();
            cout << "Pop " << val << '\n';
        }
        std::this_thread::sleep_for(std::chrono::seconds(3));
    }
    m.unlock();
}
```

master*+ 0 0 Live Share Ln 312, Col 1 Spaces: 4 UTF-8 LF C++ Mac

Mutex

main.cpp | Polygon.h | Polygon.cpp | thread_guard.h | thread_safe_stack.h | library.h

main.cpp > main()

```cpp
    if (t_2.joinable())
    {
        t_2.join();
        cout << "Thread with id " << id2 << " is terminated" << '\n';
    }
    if (t_3.joinable())
    {
        t_3.join();
        cout << "Thread with id " << id3 << " is terminated" << '\n';
    }

    //*********************************************
    //          -- Using mutexes in C++ --
    //*********************************************
    cout << "--- Using mutexes in C++ ---" << '\n';

    vector<std::thread> threads;

    char characters[] = {'+', '-', '*', '/'};
    for(unsigned i=0;i < 4 ;++i)
    {
        threads.push_back(std::thread(function3,characters[i]));
    }
    for_each(threads.begin(),threads.end(),std::mem_fn(&std::thread::join)); // std::mem_fn invokes the member function pointed by pm.
```

```cpp
void function3(char c)
{
    // std::lock_guard<std::mutex> guard(m);
    std::this_thread::sleep_for(std::chrono::seconds(2));
    for(unsigned i =0; i<200; i++){
        cout << c << " ";
    }
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
--- Using mutexes in C++ ---
** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * / / / / / / / / / / / / / / / / / / / / - - - - - - - - -
- - - - - - - -  / / / / / /-+ + + +  - - - - - - - - - - - - - - - - / / / / / / / / /  + + + + + + + + + /- - - - - - - - - - - - - - - - - - - + / /
 / /- + + + + + + + +  - - -  / / / / / / / / / / / / / / -+ + + + / - - -+ / / / / / / / / / /   + + + + +/- - - - - - - - - -   / / / / / / / / / -+ + +
+/ - - - - - - - -  / / / + + + + + + + + + + + + -/ / / / /+ - - - -   + + + + + + + + + + + + + + + + + + + + + + +-/ / / /   - - - - /+ + + - / / /
+ - - - - - - - - - - - - - - - - - - - - - - - -   + + + -/ / / /+ - - - - - - - - -  + + + + + + + + +  / / / / / / / / / / / /-  / /+ + + + + +
 /+  +- - - - -  +-  +-  +-/ / / / - + + + +  -+  -+  -+  -+ - - - - - - - - - -   - + -/+ + + + +  + + /- - + - - - + + + + + +/   / +-  +/ - + / / / /
 - - - - - / + + + +  /+-   + + + + / / + + + + + + + + +/ - - -  + + + + + /-  - -/+   + + + + / / /+- - - - / /  + + + + + + -/  + + + + +  /+- - -  +
/ - - - + - - - -/  -/  -/  -/  / / / / / / / / / / /-  + + + + / - +/ + + + + +  - - + /- + +   + + + + +- - + + + + +  / -+ + + +  -/ /  + + + / / / / /
(base) Ivaldo:Week10 admin$
```

bash
bash

master*+ ⊗ 0 ⚠ 0 Live Share Ln 171, Col 5 Spaces: 4 UTF-8 LF C++ Mac

Mutex

main.cpp | Polygon.h | Polygon.cpp | thread_guard.h | thread_safe_stack.h | library.h

main.cpp > mtx

```cpp
    //*************************************************
    //          -- Using mutexes in C++ --
    //*************************************************
    cout << "--- Using mutexes in C++ ---" << '\n';

    vector<std::thread> threads;

    char characters[] = {'+', '-', '*', '/'};
    for(unsigned i=0;i < 4 ;++i)
    {
        threads.push_back(std::thread(function3,characters[i]));
    }
    for_each(threads.begin(),threads.end(),std::mem_fn(&std::thread::join)); // std::mem_fn invokes the member function pointed by pm.
```

```cpp
void function3(char c)
{
    std::lock_guard<std::mutex> guard(m);
    std::this_thread::sleep_for(std::chrono::seconds(2));
    for(unsigned i =0; i<200; i++){
        cout << c << " ";
    }
}
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

```
--- Using mutexes in C++ ---
+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
 + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - - - - - - - - - - - - - - - - - * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * / / / / / / / / / / / /
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
 / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / (base) Ivaldo:Week10 admin$
```

bash
bash

master*+ | 0 ⚠ 0 | Live Share | Ln 282, Col 1 (17 selected) | Spaces: 4 | UTF-8 | LF | C++ | Mac

Waiting for an event or other condition

If one thread is waiting for a second thread to complete the task, it has several options:

- First, it could just keep checking a flag in shared data (protected by a mutex) and have the second thread set the flag when it completes the task.
- A second option is to have the waiting thread sleep for small periods between the checks using the std::this_thread::sleep_for().
- The third, and preferred, option is to use the facilities from the C++ Standard Library to wait for the event itself. The most basic mechanism for waiting for an event to be triggered by another thread - std::condition_variable.

Waiting for an event or other condition


```cpp
//*************************************************
//  - Waiting for an event or other condition  -
//*************************************************
cout << "--- Waiting for an event or other condition ---" << '\n';

// SOURCE : https://www.tutorialcup.com/cplusplus/multithreading.htm
cout << "The id of current thread is " << std::this_thread::get_id() << '\n';

//get current time
time_t timet = system_clock::to_time_t(system_clock::now());

//convert it to tm struct
struct tm * time = localtime(&timet);
cout << "Current time: " << std::put_time(time, "%X") << '\n';

std::cout << "Waiting for the next minute to begin...\n";
time->tm_min++, time->tm_sec = 0;
//sleep until next minute is not reached
std::this_thread::sleep_until(system_clock::from_time_t(mktime(time)));
cout << std::put_time(time, "%X") << " reached!\n";

//sleep for 5 seconds
std::this_thread::sleep_for(std::chrono::seconds(5));

//get current time
timet = system_clock::to_time_t(system_clock::now());

// convert it to tm struct
time = std::localtime(&timet);
cout << "Current time: " << std::put_time(time, "%X") << '\n';
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
--- Waiting for an event or other condition ---
The id of current thread is 0x11171d5c0
Current time: 12:54:28
Waiting for the next minute to begin...
12:55:00 reached!
Current time: 12:55:05
(base) Ivaldo:Week10 admin$
```

bash
bash

Waiting for an event or other condition

main.cpp ● | Makefile | library.h | Polygon.h | Polygon.cpp

main.cpp > main()

```
216   cout << "--- By condition " << '\n';
217
218   std::thread thr_print(print_name);
219   std::thread thr_input(input_name);
220
221   if(thr_print.joinable()) thr_print.join();
222   if(thr_input.joinable()) thr_input.join();
223
224
225
226   void print_name () {
227     std::unique_lock<std::mutex> lck(m);
228     while (!ready) cv.wait(lck);
229     string name;
230     std::cin >> name;
231     std::cout << "My name is: " << name << '\n';
232   }
233
234   void input_name() {
235     std::unique_lock<std::mutex> lck(m);
236     cout << "Enter your name: ";
237     ready = true;
238     cv.notify_one();
239   }
240
241
242
243
244
245
246
247
248
249
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

bash

```
--- By condition
Enter your name: Ivaldo
My name is: Ivaldo
(base) Ivaldo:Week10 admin$
```

master*+ ⊗ 0 ⚠ 0 Live Share Ln 250, Col 1 Spaces: 4 UTF-8 LF C++ Mac

Asynchronous task: std::future

Tasks work very similar to threads, but the main difference is that they can return a value.

You use std::async to start an *asynchronous task* for which you don't need the result right away.

std::async returns a std::future object, which will eventually hold the return value of the function.

When you need the value, you just call get() on the future, and the thread blocks until the future is *ready* and then returns the value.

std::async function template declared in the <future> header.

Using std::future

$$\begin{bmatrix} A_x & A_y & A_z \end{bmatrix} \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} = A_x B_x + A_y B_y + A_z B_z = \vec{A} \cdot \vec{B}$$

```
long long getDotProduct(std::vector<int>& v,std::vector<int>& w){
  return std::inner_product(v.begin(),v.end(),w.begin(),0LL);
}

long long getDotProductFuture(std::vector<int>& v, std::vector<int>& w){

    auto future1 = std::async([&]{return std::inner_product(&v[0],&v[v.size()/4],&w[0],0LL);});
    auto future2 = std::async([&]{return std::inner_product(&v[v.size()/4],&v[v.size()/2],&w[v.size()/4],0LL);});
    auto future3 = std::async([&]{return std::inner_product(&v[v.size()/2],&v[v.size()*3/4],&w[v.size()/2],0LL);});
    auto future4 = std::async([&]{return std::inner_product(&v[v.size()*3/4],&v[v.size()],&w[v.size()*3/4],0LL);});

    return future1.get() + future2.get() + future3.get() + future4.get();
}
```

Using std::future

main.cpp | Polygon.h | Polygon.cpp | thread_guard.h | thread_safe_stack.h | library.h

main.cpp > NUM

```cpp
// get NUM random numbers from 0 .. 100
std::random_device seed;

// generator
std::mt19937 engine(seed()); // pseudo-random generator

// distribution
std::uniform_int_distribution<int> dist(0,100);

// fill the vectors
std::vector<int> v, w;
v.reserve(NUM);
w.reserve(NUM);
for (int i=0; i< NUM; ++i){
    v.push_back(dist(engine));
    w.push_back(dist(engine));
}

// measure the execution time
system_clock::time_point start = system_clock::now();
cout << "getDotProduc(v,w): " << getDotProduct(v,w) << '\n';
std::chrono::duration<double> dur  = system_clock::now() - start;
cout << "Total Time Taken = " << dur.count() << '\n';

// measure the execution time using Future
start = std::chrono::system_clock::now();
cout << "getDotProducFuture(v,w): " << getDotProductFuture(v,w) << '\n';
dur  = std::chrono::system_clock::now() - start;
cout << "Total Time Taken using Future = " << dur.count() << '\n';

return 0;
```

```cpp
long long getDotProduct(std::vector<int>& v,std::vector<int>& w){
  return std::inner_product(v.begin(),v.end(),w.begin(),0LL);
}

long long getDotProductFuture(std::vector<int>& v, std::vector<int>& w){

    auto future1 = std::async([&]{return std::inner_product(&v[0],&v[v.size()/4],&w[0],0LL);});
    auto future2 = std::async([&]{return std::inner_product(&v[v.size()/4],&v[v.size()/2],&w[v.size()/4],0LL);});
    auto future3 = std::async([&]{return std::inner_product(&v[v.size()/2],&v[v.size()*3/4],&w[v.size()/2],0LL);});
    auto future4 = std::async([&]{return std::inner_product(&v[v.size()*3/4],&v[v.size()],&w[v.size()*3/4],0LL);});

    return future1.get() + future2.get() + future3.get() + future4.get();
}
```

PROBLEMS | OUTPUT | TERMINAL | DEBUG CONSOLE

```
--- Using std::future ---
getDotProduc(v,w): 250004960777
Total Time Taken = 1.51819
getDotProducFuture(v,w): 250004960777
Total Time Taken using Future = 0.141271
(base) Ivaldo:Week10 admin$
```

bash
bash

master*+ | 0 ⚠ 0 | Live Share | Ln 17, Col 1 (33 selected) | Spaces: 4 | UTF-8 | LF | C++ | Mac

References:

1. Bogotobogo (2022/03/02) Thread Tutorials, https://www.bogotobogo.com/cplusplus/C11/1_C11_creating_thread.php
2. CS & Math Stuff(2022/03/03) Threads-Basics, https://oneraynyday.github.io/dev/2017/11/19/C++-Threads-Basics/
3. Hackernoon (2022/03/02) Learn C++ Multi-Threading in 5 Minutes, https://hackernoon.com/learn-c-multi-threading-in-5-minutes-8b881c92941f
4. Tutorialcup (2022/03/02) C++ Multithreading , https://www.tutorialcup.com/cplusplus/multithreading.htm
5. Baeldung (2022/03/02) The Difference Between Asynchronous And Multi-Threading, https://www.baeldung.com/cs/async-vs-multi-threading
6. MC++ (2022/03/02) Asynchronous Function Calls, https://www.modernescpp.com/index.php/asynchronous-function-calls
7. Anthony, Williams. "c++ concurrency in Action. Practical Multithrea-ding." (2001).
8. Youtube, Back to Basics: Concurrency - Arthur O'Dwyer - CppCon 2020. https://www.youtube.com/watch?v=F6Ipn7gCOsY

Appendix

- Libraries.h
- thread_guard.h
- Polygon.h
- Polygon.cpp
- main.cpp