

ThreadPool

Generated by Doxygen 1.9.4

1 Data Structure Index	1
1.1 Data Structures	1
2 File Index	3
2.1 File List	3
3 Data Structure Documentation	5
3.1 ThreadPool Class Reference	5
3.1.1 Detailed Description	6
3.1.2 Constructor & Destructor Documentation	6
3.1.2.1 ThreadPool() [1/2]	6
3.1.2.2 ThreadPool() [2/2]	6
3.1.2.3 ~ThreadPool()	7
3.1.3 Member Function Documentation	8
3.1.3.1 finish()	8
3.1.3.2 status()	9
3.1.3.3 submit()	9
3.1.3.4 wait()	10
3.1.4 Field Documentation	11
3.1.4.1 cv	11
3.1.4.2 cv2	11
3.1.4.3 exit_flag	11
3.1.4.4 mtx	11
3.1.4.5 mtx2	11
3.1.4.6 num_threads	12
3.1.4.7 q	12
3.1.4.8 state	12
3.1.4.9 th	12
4 File Documentation	13
4.1 example.cpp File Reference	13
4.1.1 Function Documentation	13
4.1.1.1 main()	13
4.2 example.cpp	14
4.3 test.hpp File Reference	15
4.3.1 Function Documentation	15
4.3.1.1 function1()	16
4.3.1.2 function2()	16
4.3.1.3 test1()	16
4.3.1.4 test2()	17
4.3.1.5 test3()	17
4.3.1.6 test4() [1/2]	17
4.3.1.7 test4() [2/2]	17

4.4 test.hpp	18
4.5 ThreadPool.hpp File Reference	18
4.5.1 Detailed Description	19
4.6 ThreadPool.hpp	19
4.7 test.cpp File Reference	21
4.7.1 Function Documentation	22
4.7.1.1 function1()	22
4.7.1.2 function2()	22
4.7.1.3 test1()	23
4.7.1.4 test2()	23
4.7.1.5 test4()	23
4.8 test.cpp	24
4.9 ThreadPool.cpp File Reference	24
4.9.1 Detailed Description	24
4.10 ThreadPool.cpp	25
Index	27

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

ThreadPool	Manages a pool of worker threads to execute tasks concurrently	5
----------------------------	--	---

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

example.cpp	13
test.hpp	15
ThreadPool.hpp		
	Declaration of a simple ThreadPool class	18
test.cpp	21
ThreadPool.cpp		
	Implementation of the ThreadPool class	24

Chapter 3

Data Structure Documentation

3.1 ThreadPool Class Reference

Manages a pool of worker threads to execute tasks concurrently.

```
#include <ThreadPool.hpp>
```

Public Member Functions

- `ThreadPool ()=delete`
Deleted default constructor.
- `ThreadPool (unsigned int num_threads)`
Constructs a `ThreadPool` with a fixed number of threads.
- `template<class Func , class... Args>`
`void submit (Func &&f, Args &&... args)`
Submits a task to the thread pool.
- `void finish (bool secure=true)`
Stops the thread pool.
- `bool status ()`
Returns the current state of the thread pool.
- `void wait ()`
Waits until the task queue becomes empty.
- `~ThreadPool ()`
Destructor.

Private Attributes

- `thread * th`
- `mutex mtx`
- `mutex mtx2`
- `queue< function< void()> > q`
- `unsigned int num_threads`
- `atomic< bool > exit_flag`
- `atomic< bool > state`
- `condition_variable cv`
- `condition_variable cv2`

3.1.1 Detailed Description

Manages a pool of worker threads to execute tasks concurrently.

The [ThreadPool](#) maintains a queue of tasks. Worker threads continuously fetch and execute tasks from this queue until the pool is stopped.

Note

Tasks have no return value.

Warning

This implementation uses busy waiting and does not use condition variables.

Definition at line 36 of file [ThreadPool.hpp](#).

3.1.2 Constructor & Destructor Documentation

3.1.2.1 ThreadPool() [1/2]

```
ThreadPool::ThreadPool ( ) [delete]
```

Deleted default constructor.

Forces the user to specify the number of threads.

3.1.2.2 ThreadPool() [2/2]

```
ThreadPool::ThreadPool ( unsigned int num_threads ) [explicit]
```

Constructs a [ThreadPool](#) with a fixed number of threads.

Constructs and starts the worker threads.

Parameters

<code>num_threads</code>	Number of worker threads to create.
--------------------------	-------------------------------------

Each worker thread runs a loop that retrieves tasks from the queue and executes them until termination is requested. Worker thread function.

Continuously checks the task queue and executes tasks while the pool is active and the exit flag is not set.

Definition at line 15 of file [ThreadPool.cpp](#).

```

00016     : num_threads(num_threads) {
00017
00018     /**
00019      * @brief Worker thread function.
00020      *
00021      * Continuously checks the task queue and executes tasks
00022      * while the pool is active and the exit flag is not set.
00023     */
00024     function<void(void)> f = [&] () {
00025
00026         function<void()> task;
00027
00028         unique_lock<mutex> lock(this->mtx);
00029
00030         while (!this->exit_flag.load() && this->state.load()) {
00031             cv.wait(lock, [&] () {
00032                 return !this->q.empty() || this->exit_flag.load() || !this->state.load();
00033             });
00034
00035             if (this->exit_flag.load() || !this->state.load()) {
00036                 lock.unlock();
00037                 break;
00038             }
00039             task = move(this->q.front());
00040             this->q.pop();
00041
00042             lock.unlock();
00043
00044             task();
00045         }
00046     };
00047
00048     this->exit_flag = false;
00049
00050     this->th = new (nothrow) thread[num_threads];
00051
00052     if (this->th != nullptr) {
00053         this->state = true;
00054
00055         try {
00056             for (unsigned int i = 0; i < this->num_threads; i++) {
00057                 this->th[i] = thread(f);
00058             }
00059         } catch (exception&) {
00060             this->state = false;
00061             // cout << "The system is unable to start a new thread" << endl;
00062         }
00063     } else {
00064         this->state = false;
00065     }
00066 }
00067 }
```

3.1.2.3 ~ThreadPool()

ThreadPool::~ThreadPool ()

Destructor.

Stops the pool and releases all allocated resources.

Ensures that the pool is stopped and memory is released.

Definition at line 105 of file [ThreadPool.cpp](#).

```

00105     {
00106     this->finish();
00107     try{
00108         delete[] this->th;
00109     }catch(...){
00110
00111     }
00112 }
```

Here is the call graph for this function:



3.1.3 Member Function Documentation

3.1.3.1 finish()

```
void ThreadPool::finish (
    bool secure = true )
```

Stops the thread pool.

Stops the thread pool and optionally joins threads.

Sets the exit flag and optionally waits for all threads to finish.

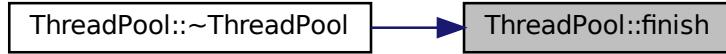
Parameters

<code>secure</code>	If true, joins all worker threads.
<code>secure</code>	If true, waits for all threads to finish execution.

Definition at line 83 of file [ThreadPool.cpp](#).

```
00083     this->exit_flag = true;
00084
00085     if (secure && this->state.load()) {
00086         for (unsigned int i = 0; i < this->num_threads; i++) {
00087             if (this->th[i].joinable()) {
00088                 try {
00089                     this->th[i].join();
00090                 } catch (...) {
00091                     // cout << "Error joining thread " << i << endl;
00092                 }
00093             }
00094         }
00095     }
00096     this->state = false;
00097 }
00098 }
```

Here is the caller graph for this function:



3.1.3.2 status()

```
bool ThreadPool::status ( )
```

Returns the current state of the thread pool.

Checks whether the thread pool is running.

Returns

True if the pool is running, false otherwise.

True if the pool is active, false otherwise.

Definition at line 74 of file [ThreadPool.cpp](#).

```
00074           {  
00075     return this->state;  
00076 }
```

3.1.3.3 submit()

```
template<class Func , class... Args>  
void ThreadPool::submit (   
    Func && f,  
    Args &&... args )
```

Submits a task to the thread pool.

Adds a new task to the task queue.

The task is stored in the internal queue and will be executed by one of the worker threads.

Template Parameters

<i>Func</i>	Callable object type.
<i>Args</i>	Argument types for the callable.

Parameters

<i>f</i>	Function to execute.
<i>args</i>	Arguments passed to the function.

Locks the mutex, pushes the task into the queue, and then unlocks the mutex.

Definition at line 136 of file [ThreadPool.hpp](#).

```
00136
00137
00138     {
00139         lock_guard<mutex> lock(this->mtx);
00140
00141
00142         this->q.push(
00143             move(
00144                 [&] () {
00145                     f(args...);
00146                 }
00147             );
00148         );
00149     }
00150
00151     this->cv.notify_one();
00152 }
```

Here is the caller graph for this function:



3.1.3.4 wait()

```
void ThreadPool::wait ( )
```

Waits until the task queue becomes empty.

Waits until all queued tasks have been processed.

Note

This function performs a busy wait.

Warning

This function uses busy waiting and may waste CPU time.

Definition at line 119 of file [ThreadPool.cpp](#).

```
00119
00120     unique_lock<mutex> lock(this->mtx2);
00121
00122     this->cv2.wait(lock, [&] () {
00123         return this->q.empty();
00124     });
00125 }
00126 }
```

3.1.4 Field Documentation

3.1.4.1 cv

```
condition_variable ThreadPool::cv [private]
```

Condition variable used to stop the threads until a new process is pushed

Definition at line 62 of file [ThreadPool.hpp](#).

3.1.4.2 cv2

```
condition_variable ThreadPool::cv2 [private]
```

Condition variable used to wait until all the queue finish

Definition at line 65 of file [ThreadPool.hpp](#).

3.1.4.3 exit_flag

```
atomic<bool> ThreadPool::exit_flag [private]
```

Flag indicating when threads should terminate

Definition at line 56 of file [ThreadPool.hpp](#).

3.1.4.4 mtx

```
mutex ThreadPool::mtx [private]
```

Mutex protecting access to the task queue

Definition at line 44 of file [ThreadPool.hpp](#).

3.1.4.5 mtx2

```
mutex ThreadPool::mtx2 [private]
```

Mutex protecting used to wait until all the process finish

Definition at line 47 of file [ThreadPool.hpp](#).

3.1.4.6 num_threads

```
unsigned int ThreadPool::num_threads [private]
```

Number of threads managed by the pool

Definition at line 53 of file [ThreadPool.hpp](#).

3.1.4.7 q

```
queue<function<void()> > ThreadPool::q [private]
```

Queue containing pending tasks

Definition at line 50 of file [ThreadPool.hpp](#).

3.1.4.8 state

```
atomic<bool> ThreadPool::state [private]
```

Indicates whether the thread pool is active

Definition at line 59 of file [ThreadPool.hpp](#).

3.1.4.9 th

```
thread* ThreadPool::th [private]
```

Pointer to the array of worker threads

Definition at line 41 of file [ThreadPool.hpp](#).

The documentation for this class was generated from the following files:

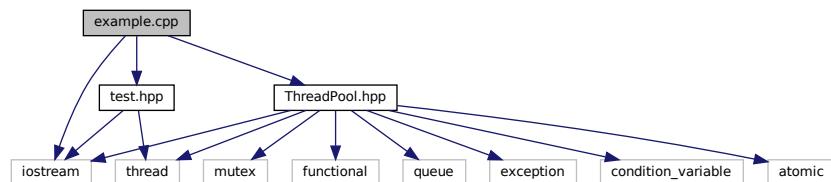
- [ThreadPool.hpp](#)
- [ThreadPool.cpp](#)

Chapter 4

File Documentation

4.1 example.cpp File Reference

```
#include <iostream>
#include "ThreadPool.hpp"
#include "test.hpp"
Include dependency graph for example.cpp:
```



Functions

- int [main](#) (int argc, char *argv[])

4.1.1 Function Documentation

4.1.1.1 main()

```
int main (
    int argc,
    char * argv[] )
```

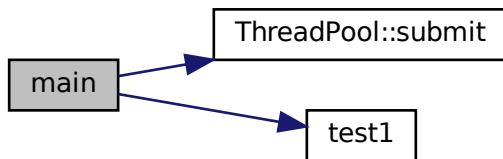
Definition at line 9 of file [example.cpp](#).

```
00009
00010 {
```

```

00011     ThreadPool thp{10};
00012
00013     for(int i=0;i<20;i++){
00014         thp.submit(test1,i);
00015     }
00016
00017
00018     thp.wait();
00019
00020     if(thp.status()){
00021         thp.finish();
00022     }
00023
00024     if(!thp.status()){
00025         cout<<"Cerrado correctamente"\;
00026     }
00027
00028     return 0;
00029
00030 }
```

Here is the call graph for this function:



4.2 example.cpp

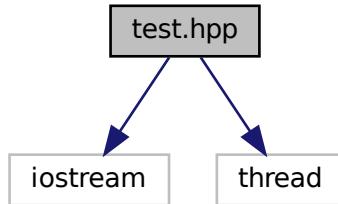
[Go to the documentation of this file.](#)

```

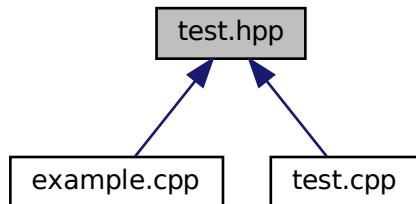
00001 #include <iostream>
00002 #include "ThreadPool.hpp"
00003 #include "test.hpp"
00004
00005
00006 using namespace std;
00007
00008
00009 int main(int argc, char * argv[]){
00010
00011     ThreadPool thp{10};
00012
00013     for(int i=0;i<20;i++){
00014         thp.submit(test1,i);
00015     }
00016
00017
00018     thp.wait();
00019
00020     if(thp.status()){
00021         thp.finish();
00022     }
00023
00024     if(!thp.status()){
00025         cout<<"Cerrado correctamente"\;
00026     }
00027
00028     return 0;
00029
00030 }
```

4.3 test.hpp File Reference

```
#include <iostream>
#include <thread>
Include dependency graph for test.hpp:
```



This graph shows which files directly or indirectly include this file:



Functions

- void **test1** (int)
- void **test2** (int, int)
- template<class T >
void **test3** (T x, T y)
- template<class... Args>
void **test4** (int x, Args... args)
- void **test4** ()
- void **function1** (int x, int y, int z)
- template<class... Args>
void **function2** (void(*funcion)(int, int, int), Args... args)

4.3.1 Function Documentation

4.3.1.1 function1()

```
void function1 (
    int x,
    int y,
    int z )
```

Definition at line 27 of file test.cpp.

```
00027
00028     cout<<x+y+z<<endl;
00029
00030 }
```

4.3.1.2 function2()

```
template<class... Args>
void function2 (
    void(*)(int, int, int) funcion,
    Args... args )
```

Definition at line 33 of file test.cpp.

```
00033
00034
00035     cout<<"function2 "<<endl;
00036     funcion(2,3,4);
00037     funcion(args...);
00038 }
```

4.3.1.3 test1()

```
void test1 (
    int i )
```

Definition at line 7 of file test.cpp.

```
00007
00008     cout<<this_thread::get_id()<<" Proceso dentro de test1 "<<i<<endl;
00009 }
```

Here is the caller graph for this function:



4.3.1.4 test2()

```
void test2 (
    int x,
    int y )
```

Definition at line 12 of file [test.cpp](#).

```
00012     {
00013         cout<<this_thread::get_id()<<" Proceso dentro de test2 con suma "<<x+y<<endl;
00014     }
```

4.3.1.5 test3()

```
template<class T >
void test3 (
    T x,
    T y )
```

Definition at line 13 of file [test.hpp](#).

```
00013     {
00014         cout<<this_thread::get_id()<<" Proceso dentro de test3 con suma "<<x+y<<endl;
00015     }
```

4.3.1.6 test4() [1/2]

```
void test4 ( )
```

Definition at line 22 of file [test.cpp](#).

```
00022     {
00023         cout<<this_thread::get_id()<<" Proceso dentro de test4 sin argumentos"<<endl;
00024     }
```

4.3.1.7 test4() [2/2]

```
template<class... Args>
void test4 (
    int x,
    Args... args )
```

Definition at line 18 of file [test.hpp](#).

```
00018     {
00019         cout<<this_thread::get_id()<<" Proceso dentro de test4 con argumento "<<x<<endl;
00020     }
```

4.4 test.hpp

[Go to the documentation of this file.](#)

```

00001 #ifndef TEST_HPP
00002 #define TEST_HPP
00003
00004 #include <iostream>
00005 #include <thread>
00006
00007 using namespace std;
00008
00009 void test1(int);
00010 void test2(int,int);
00011
00012 template <class T>
00013 void test3(T x,T y){
00014     cout<<this_thread::get_id()<<" Proceso dentro de test3 con suma "<<x+y<<endl;
00015 }
00016
00017 template <class... Args>
00018 void test4(int x,Args... args){
00019     cout<<this_thread::get_id()<<" Proceso dentro de test4 con argumento "<<x<<endl;
00020 }
00021
00022 void test4();
00023
00024
00025 void function1(int x,int y,int z);
00026
00027 template <class... Args>
00028 void function2(void (*funcion)(int,int,int),Args... args);
00029
00030
00031 #endif

```

4.5 ThreadPool.hpp File Reference

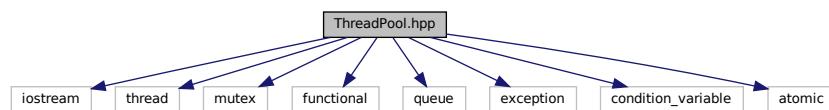
Declaration of a simple [ThreadPool](#) class.

```

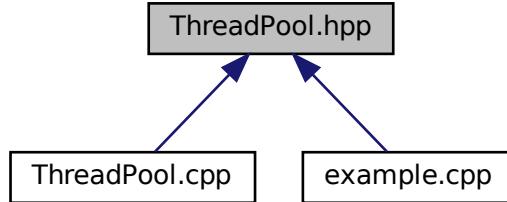
#include <iostream>
#include <thread>
#include <mutex>
#include <functional>
#include <queue>
#include <exception>
#include <condition_variable>
#include <atomic>

```

Include dependency graph for ThreadPool.hpp:



This graph shows which files directly or indirectly include this file:



Data Structures

- class [ThreadPool](#)
Manages a pool of worker threads to execute tasks concurrently.

4.5.1 Detailed Description

Declaration of a simple [ThreadPool](#) class.

Author

qwert-asci

This file contains the declaration of a basic thread pool that executes tasks using a fixed number of worker threads.

Definition in file [ThreadPool.hpp](#).

4.6 ThreadPool.hpp

[Go to the documentation of this file.](#)

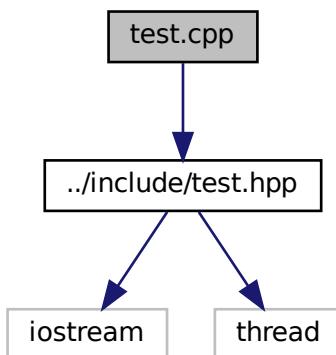
```
00001
00002 /**
00003 * @file ThreadPool.hpp
00004 * @author qwert-asci
00005 * @brief Declaration of a simple ThreadPool class.
00006 *
00007 * This file contains the declaration of a basic thread pool
00008 * that executes tasks using a fixed number of worker threads.
00009 */
00010
00011 #ifndef THREADPOOL_HPP
00012 #define THREADPOOL_HPP
00013
00014 #include <iostream>
00015 #include <thread>
00016 #include <mutex>
00017 #include <functional>
00018 #include <queue>
00019 #include <exception>
00020 #include <condition_variable>
00021 #include <atomic>
00022
```

```
00023 using namespace std;
00024
00025 /**
00026 * @class ThreadPool
00027 * @brief Manages a pool of worker threads to execute tasks concurrently.
00028 *
00029 * The ThreadPool maintains a queue of tasks. Worker threads continuously
00030 * fetch and execute tasks from this queue until the pool is stopped.
00031 *
00032 * @note Tasks have no return value.
00033 * @warning This implementation uses busy waiting and does not use
00034 *          condition variables.
00035 */
00036 class ThreadPool {
00037
00038 private:
00039
00040     /** Pointer to the array of worker threads */
00041     thread* th;
00042
00043     /** Mutex protecting access to the task queue */
00044     mutex mtx;
00045
00046     /** Mutex protecting used to wait until all the process finish */
00047     mutex mtx2;
00048
00049     /** Queue containing pending tasks */
00050     queue<function<void()> q;
00051
00052     /** Number of threads managed by the pool */
00053     unsigned int num_threads;
00054
00055     /** Flag indicating when threads should terminate */
00056     atomic<bool> exit_flag;
00057
00058     /** Indicates whether the thread pool is active */
00059     atomic<bool> state;
00060
00061     /** Condition variable used to stop the threads until a new process is pushed */
00062     condition_variable cv;
00063
00064     /** Condition variable used to wait until all the queue finish */
00065     condition_variable cv2;
00066
00067 public:
00068
00069 /**
00070 * @brief Deleted default constructor.
00071 *
00072 * Forces the user to specify the number of threads.
00073 */
00074     ThreadPool() = delete;
00075
00076 /**
00077 * @brief Constructs a ThreadPool with a fixed number of threads.
00078 *
00079 * @param num_threads Number of worker threads to create.
00080 */
00081     explicit ThreadPool(unsigned int num_threads);
00082
00083 /**
00084 * @brief Submits a task to the thread pool.
00085 *
00086 * The task is stored in the internal queue and will be executed
00087 * by one of the worker threads.
00088 *
00089 * @tparam Func Callable object type.
00090 * @tparam Args Argument types for the callable.
00091 * @param f Function to execute.
00092 * @param args Arguments passed to the function.
00093 *
00094 */
00095     template <class Func, class... Args>
00096     void submit(Func&& f, Args&&... args);
00097
00098 /**
00099 * @brief Stops the thread pool.
00100 *
00101 * Sets the exit flag and optionally waits for all threads to finish.
00102 *
00103 * @param secure If true, joins all worker threads.
00104 */
00105     void finish(bool secure = true);
00106
00107 /**
00108 * @brief Returns the current state of the thread pool.
00109 *
```

```
00110 * @return True if the pool is running, false otherwise.
00111 */
00112     bool status();
00113
00114 /**
00115 * @brief Waits until the task queue becomes empty.
00116 *
00117 * @note This function performs a busy wait.
00118 */
00119     void wait();
00120
00121 /**
00122 * @brief Destructor.
00123 *
00124 * Stops the pool and releases all allocated resources.
00125 */
00126     ~ThreadPool();
00127 };
00128
00129 /**
00130 * @brief Adds a new task to the task queue.
00131 *
00132 * Locks the mutex, pushes the task into the queue,
00133 * and then unlocks the mutex.
00134 */
00135 template <class Func, class... Args>
00136 void ThreadPool::submit(Func&& f, Args&&... args) {
00137
00138     {
00139         lock_guard<mutex> lock(this->mtx);
00140
00141         this->q.push(
00142             move(
00143                 [&] () {
00144                     f(args...);
00145                 }
00146             );
00147         );
00148     }
00149 }
00150
00151     this->cv.notify_one();
00152 }
00153
00154 #endif
```

4.7 test.cpp File Reference

```
#include "../include/test.hpp"
Include dependency graph for test.cpp:
```



Functions

- void **test1** (int i)
- void **test2** (int x, int y)
- void **test4** ()
- void **function1** (int x, int y, int z)
- template<class... Args>
void **function2** (void(*funcion)(int, int, int), Args... args)

4.7.1 Function Documentation

4.7.1.1 **function1()**

```
void function1 (
    int x,
    int y,
    int z )
```

Definition at line 27 of file [test.cpp](#).

```
00027
00028     cout<<x+y+z<<endl;
00029
00030 }
```

4.7.1.2 **function2()**

```
template<class... Args>
void function2 (
    void(*)(int, int, int) funcion,
    Args... args )
```

Definition at line 33 of file [test.cpp](#).

```
00033
00034
00035     cout<<"function2 "<<endl;
00036     funcion(2,3,4);
00037     funcion(args...);
00038 }
```

4.7.1.3 test1()

```
void test1 (
    int i )
```

Definition at line 7 of file [test.cpp](#).

```
00007      {
00008          cout<<this_thread::get_id()<<" Proceso dentro de test1 "<<i<<endl;
00009      }
```

Here is the caller graph for this function:



4.7.1.4 test2()

```
void test2 (
    int x,
    int y )
```

Definition at line 12 of file [test.cpp](#).

```
00012      {
00013          cout<<this_thread::get_id()<<" Proceso dentro de test2 con suma "<<x+y<<endl;
00014      }
```

4.7.1.5 test4()

```
void test4 ( )
```

Definition at line 22 of file [test.cpp](#).

```
00022      {
00023          cout<<this_thread::get_id()<<" Proceso dentro de test4 sin argumentos"<<endl;
00024      }
```

4.8 test.cpp

[Go to the documentation of this file.](#)

```

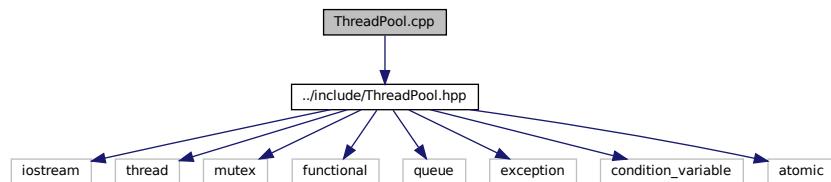
00001 #include "../include/test.hpp"
00002
00003
00004
00005 using namespace std;
00006
00007 void test1(int i){
00008     cout<<this_thread::get_id()<<" Proceso dentro de test1 "«i«endl;
00009 }
00010
00011
00012 void test2(int x,int y){
00013     cout<<this_thread::get_id()<<" Proceso dentro de test2 con suma "«x+y«endl;
00014 }
00015
00016
00017
00018
00019
00020
00021
00022 void test4(){
00023     cout<<this_thread::get_id()<<" Proceso dentro de test4 sin argumentos"«endl;
00024 }
00025
00026
00027 void function1(int x,int y,int z){
00028     cout<<x+y+z<<endl;
00029 }
00030
00031
00032 template <class... Args>
00033 void function2(void (*funcion)(int,int,int),Args... args){
00034
00035     cout<<"function2 "«endl;
00036     funcion(2,3,4);
00037     funcion(args...);
00038 }
```

4.9 ThreadPool.cpp File Reference

Implementation of the [ThreadPool](#) class.

```
#include "../include/ThreadPool.hpp"
```

Include dependency graph for ThreadPool.cpp:



4.9.1 Detailed Description

Implementation of the [ThreadPool](#) class.

Definition in file [ThreadPool.cpp](#).

4.10 ThreadPool.cpp

[Go to the documentation of this file.](#)

```

00001 /**
00002 * @file ThreadPool.cpp
00003 * @brief Implementation of the ThreadPool class.
00004 */
00005
00006 #include "../include/ThreadPool.hpp"
00007
00008 /**
00009 * @brief Constructs and starts the worker threads.
00010 *
00011 * Each worker thread runs a loop that retrieves tasks
00012 * from the queue and executes them until termination
00013 * is requested.
00014 */
00015 ThreadPool::ThreadPool(unsigned int num_threads)
00016     : num_threads(num_threads) {
00017
00018 /**
00019 * @brief Worker thread function.
00020 *
00021 * Continuously checks the task queue and executes tasks
00022 * while the pool is active and the exit flag is not set.
00023 */
00024     function<void(void)> f = [&] () {
00025
00026         function<void()> task;
00027
00028         unique_lock<mutex> lock(this->mtx);
00029
00030         while (!this->exit_flag.load() && this->state.load()) {
00031             cv.wait(lock, [&] () {
00032                 return !this->q.empty() || this->exit_flag.load() || !this->state.load();
00033             });
00034
00035             if (this->exit_flag.load() || !this->state.load()){
00036                 lock.unlock();
00037                 break;
00038             }
00039             task = move(this->q.front());
00040             this->q.pop();
00041
00042             lock.unlock();
00043
00044             task();
00045
00046         };
00047     };
00048
00049     this->exit_flag = false;
00050
00051     this->th = new (nothrow) thread[num_threads];
00052
00053     if (this->th != nullptr) {
00054         this->state = true;
00055
00056         try {
00057             for (unsigned int i = 0; i < this->num_threads; i++) {
00058                 this->th[i] = thread(f);
00059             }
00060         } catch (exception&) {
00061             this->state = false;
00062             // cout << "The system is unable to start a new thread" << endl;
00063         }
00064     } else {
00065         this->state = false;
00066     }
00067 }
00068
00069 /**
00070 * @brief Checks whether the thread pool is running.
00071 *
00072 * @return True if the pool is active, false otherwise.
00073 */
00074 bool ThreadPool::status() {
00075     return this->state;
00076 }
00077
00078 /**
00079 * @brief Stops the thread pool and optionally joins threads.
00080 *
00081 * @param secure If true, waits for all threads to finish execution.
00082 */

```

```
00083 void ThreadPool::finish(bool secure) {
00084     this->exit_flag = true;
00085
00086     if (secure && this->state.load()) {
00087         for (unsigned int i = 0; i < this->num_threads; i++) {
00088             if (this->th[i].joinable()) {
00089                 try {
00090                     this->th[i].join();
00091                 } catch (...) {
00092                     // cout << "Error joining thread " << i << endl;
00093                 }
00094             }
00095         }
00096         this->state = false;
00097     }
00098 }
00099
00100 /**
00101 * @brief Destructor.
00102 *
00103 * Ensures that the pool is stopped and memory is released.
00104 */
00105 ThreadPool::~ThreadPool() {
00106     this->finish();
00107     try{
00108         delete[] this->th;
00109     }catch(...){
00110     }
00111 }
00112
00113
00114 /**
00115 * @brief Waits until all queued tasks have been processed.
00116 *
00117 * @warning This function uses busy waiting and may waste CPU time.
00118 */
00119 void ThreadPool::wait() {
00120     unique_lock<mutex> lock(this->mtx2);
00121
00122     this->cv2.wait(lock,[&](){
00123         return this->q.empty();
00124     });
00125 };
00126 }
```

Index

- ~ThreadPool
 - ThreadPool, 7
- cv
 - ThreadPool, 11
- cv2
 - ThreadPool, 11
- example.cpp, 13, 14
 - main, 13
- exit_flag
 - ThreadPool, 11
- finish
 - ThreadPool, 8
- function1
 - test.cpp, 22
 - test.hpp, 15
- function2
 - test.cpp, 22
 - test.hpp, 16
- main
 - example.cpp, 13
- mtx
 - ThreadPool, 11
- mtx2
 - ThreadPool, 11
- num_threads
 - ThreadPool, 11
- q
 - ThreadPool, 12
- state
 - ThreadPool, 12
- status
 - ThreadPool, 9
- submit
 - ThreadPool, 9
- test.cpp, 21, 24
 - function1, 22
 - function2, 22
 - test1, 22
 - test2, 23
 - test4, 23
- test.hpp, 15, 18
 - function1, 15
 - function2, 16
- test1, 16
- test2, 16
- test3, 17
- test4, 17
- test1
 - test.cpp, 22
 - test.hpp, 16
- test2
 - test.cpp, 23
 - test.hpp, 16
- test3
 - test.hpp, 17
- test4
 - test.cpp, 23
 - test.hpp, 17
- th
 - ThreadPool, 12
- ThreadPool, 5
 - ~ThreadPool, 7
 - cv, 11
 - cv2, 11
 - exit_flag, 11
 - finish, 8
 - mtx, 11
 - mtx2, 11
 - num_threads, 11
 - q, 12
 - state, 12
 - status, 9
 - submit, 9
 - th, 12
 - ThreadPool, 6
 - wait, 10
- ThreadPool.cpp, 24, 25
- ThreadPool.hpp, 18, 19
- wait
 - ThreadPool, 10