ssts

Stefano Lusardi

Mar 15, 2023

QUICK START

Small & Simple Task Scheduler for C++17

ssTs is a time-based *Task Scheduler*, written in modern C++.

Header only, with no external dependencies.

ssTs features:

- a ready to use, general purpose *Thread Pool* implementation.
- a *Task Scheduler* APIs to run workloads at given time points.

ssTs requires a C++17 compiler. Currently the project is built and tested on the following platforms:

- Windows, MSVC >= 2017, Clang >= 9.0
- Linux, GCC >= 7.5, Clang >= 8.0
- MacOS, GCC >= 8.4, Clang >= 10.0

CHAPTER ONE

LICENSING

This software is licensed under the MIT license. See the LICENSE file for details.

1.1 Getting Started

The fastest way to get started with **ssTs** library is to include the three header files directly in your project.

1.2 Basic Usage

1.3 Install

1.4 Examples

1.5 Tests

1.6 task

Defined in `ssts/task.hpp`

class **`task`**

Move-only callable object.

This class represents a callable object. Can be initialized with any invocable type that supports operator(). Internally the class implements a type-erasure idiom to accept any callable signature without exposing it to the outside.

Public Functions

template<typename **FunctionType**>
inline explicit **task**(*FunctionType* &&f)

Default constructor.

Creates a task instance with the given callable object. The callable object can be e.g. a lambda function, a functor, a free function or a class method bound to an object.

Parameters
`f` – Callable parameterless object wrapped within this task instance.

inline **task**(*task* &&other) noexcept

Move constructor.

Move constructs a task instance to this.

Parameters
`other` – task object.

inline void **operator()**()

operator().

Invokes a task.

inline void **invoke**()

invoke().

Invokes a task. Explicit overload of operator().

1.7 task_pool

class **task_pool**

Task Pool that can run any callable object.

This class is general purpose thread pool that can launch task asyncronously. It is possible to get an asyncronous result of a task execution.

Public Functions

inline explicit **task_pool**(const unsigned int num_threads = std::thread::hardware_concurrency())

Constructor.

Creates a *ssts::task_pool* instance with the given number of threads.

Parameters
`num_threads` – Number of threads that will be used in the underlying *ssts::task_pool*.

inline **~task_pool**()

Destructor.

Destructs this after all joinable threads are terminated.

inline void **stop**()

Stop all threads.

Stop thread pool and join all joinable threads.

template<typename **FunctionType**>
inline auto **run**(*FunctionType* &&f, const std::optional<size_t> &task_hash = std::nullopt)

Run a callable object asynchronously.

Enqueue a new task with the given callable object. The enqueued task will run as soon as a thread is available. Returns the result of the asynchronous computation.

Template Parameters

FunctionType – Types of the callable object.

Parameters

f – Callable object.

Returns

std::future task result

1.8 task_scheduler

class **task_scheduler**

Task Scheduler that can launch tasks on based several time-based policies.

This class is used to manage a queue of tasks using a fixed number of threads. The actual task execution is delgated to an internal *ssts::task_pool* object.

Public Functions

inline explicit **task_scheduler**(const unsigned int num_threads = std::thread::hardware_concurrency())

Constructor.

Creates a *ssts::task_scheduler* instance. The number of threads to be used by the *ssts::task_pool* defaults to the number of threads supported by the platform.

Parameters

num_threads – Number of threads that will be used in the underlying *ssts::task_pool*.

inline **~task_scheduler**()

Destructor.

Destructs this. If the *task_scheduler* is running its tasks are stopped first.

inline void **`start`**()

Start running tasks.

This function starts the *task_scheduler* worker thread. The function is guaranteed to return after the scheduler thread is started.

inline size_t **`size`**()

Get the number of scheduled tasks.

This function return the number of tasks that are currently scheduled for execution (both enabled and disabled).

Returns
Number of tasks to be run.

inline void **`stop`**()

Stop all running tasks.

This function stops the *task_scheduler* execution and stops all the running tasks.

inline bool **`is_duplicate_allowed`**() const

Check if duplicated tasks are allowed.

Duplicated tasks are created with the same task_id. If a task has been started without a task_id it is not possible to check if it has duplicates. In case duplicates are not allowed task insertion will be silently rejected for same task_id.

Returns
bool indicating if duplicated tasks are allowed.

inline void **`set_duplicate_allowed`**(bool is_allowed)

Enable or disable duplicated tasks.

Duplicated tasks are created with the same task_id. If a task has been started without a task_id it is not possible to check if it has duplicates. In case duplicates are not allowed task insertion will be silently rejected for same task_id.

inline bool **`is_scheduled`**(const std::string &task_id)

Check if a task is scheduled.

If a task has been started without a task_id it is not possible to query its status. In case a task_id is not found this function return false. If a task is no longer scheduled it must be added using one of the following APIs: ssts::task_scheduler::in, ssts::task_scheduler::at, ssts::task_scheduler::every.

Parameters
`task_id` – task_id to check.

Returns
bool indicating if the task is currently scheduled.

inline bool **`is_enabled`**(const std::string &task_id)

Check if a task is enabled.

If a task has been started without a task_id it is not possible to query its status. In case a task_id is not found this function return false. By default new tasks are enabled. A task can be enabled or disabled by calling *ssts::task_scheduler::set_enabled*.

Parameters
`task_id` – task_id to check.

Returns
bool indicating if the task is currently enabled.

inline bool `set_enabled`(const std::string &task_id, bool is_enabled)

Enable or disable task.

If a task has been started without a task_id it is not possible to update its status. In case a task_id is not found this function return false. It is possible to check if a task is enabled or disabled by calling *ssts::task_scheduler::is_enabled*.

Parameters

- `task_id` – task_id to enable or disable.
- `is_enabled` – true enables, false disables the given task_id.

Returns
bool indicating if the task is currently enabled.

inline bool `remove_task`(const std::string &task_id)

Remove a task.

If a task has been started without a task_id it is not possible to remove it. In case a task_id is not found this function return false. It is possible to check if a task is scheduled by calling *ssts::task_scheduler::is_scheduled*.

Parameters
`task_id` – task_id to remove.

Returns
bool indicating if the task has been properly removed.

inline bool `update_interval`(const std::string &task_id, ssts::clock::duration interval)

Update a task interval.

If a task is not recursive (i.e. has not been started with every() APIs) or the task has not been assigned a task_id, it is not possible to update it. In case of any failure (task_id not found or task non recursive) this function return false.

Parameters

- `task_id` – task_id to update.
- `interval` – new task interval to set.

Returns
bool indicating if the task has been properly updated.

INDEX