

Multithreading

- Steps to create a thread
- Some thread functions
- Locking critical sections and RLock vs Lock
- Event and Condition
- Queue based Synchronization

Run any Function on thread

- Import threading module:
`from threading import Thread`
- Create a thread:
`<thread object> = Thread(target = <target function>, args=(<args tuple>))`
- Start the Thread
`<thread object>.start()`
- Join the Thread (optional)
`<thread object>.join()`

trainer.cpp@gmail.com

The join function

- Join function allows waiting till a thread finishes

`<thread object>.join()`

- Join() is a blocking call
- Execution of the remaining code blocks till the thread on which join was called finishes.

trainer.cpp@gmail.com

Threading functions and sleep

- Current_thread()
- Get_ident()
- setName()
- getName()
- Import time
`time.sleep(<time in seconds>)`

trainer.cpp@gmail.com

Lock – Locking Critical Section

- `acquire()` - acquire a lock; blocks if lock already held by some other thread till any thread releases it
- `release()` - release the lock
- `locked()` - tell whether already locked or not
- Ex: Two threads increment the same value in parallel might cause invalid value updates

trainer.cpp@gmail.com

Using Context Manager with Lock

- `with <lock_object>:`
 # critical section of code
- Forgetting to release a lock from a thread causes all the other threads waiting for that lock to **block** infinitely.
- The **with** statement handles automatic acquisition and release of the lock object.

trainer.cpp@gmail.com

RLock vs Lock

- `lock_object = threading.Lock()`
 - `with lock_object :` `#lock acquired once`
 - `with lock_object :` `#same lock acquired again will block`
 - `# critical section of code`
- Simple Lock objects block if acquired again by the same thread.
- RLock is Re-entrant lock :
 - it doesn't block if acquired by the same thread again and again

trainer.cpp@gmail.com

Event and Condition for synchronization

- Event:
 - `set()`
 - `wait()`
 - `clear()`
- Condition: Provides locking along with Event based synchronization
 - Condition = Event + Lock
 - Multiple conditions can share a common lock.
 - `acquire()`
 - `release()`
 - `wait()`

trainer.cpp@gmail.com

Queue based synchronization

- The Queue class in python is thread safe
- The get and put methods to add and remove data from the queue are blocking calls.
- Available inside the queue module.

trainer.cpp@gmail.com