# YouTube Link: https://www.youtube.com/watch?v=iYJNmuD4McE

# A lock or mutex is a sychronization mechanism for enforcing

# limits on access to a resource in an environment where there

# are many threads of execution.

# More on locks:

# https://en.wikipedia.org/wiki/Lock\_(computer\_science)

import time

from multiprocessing import Process, Lock, Value

def add\_500\_no\_mp(total):

for i in range(100):

time.sleep(0.01)

total += 5

return total

def sub\_500\_no\_mp(total):

for i in range(100):

time.sleep(0.01)

total -= 5

return total

def add\_500\_no\_lock(total):

for i in range(100):

time.sleep(0.01)

total.value += 5

def sub\_500\_no\_lock(total):

for i in range(100):

time.sleep(0.01)

total.value -= 5

def add\_500\_lock(total, lock):

for i in range(100):

time.sleep(0.01)

lock.acquire()

total.value += 5

lock.release()

def sub\_500\_lock(total, lock):

for i in range(100):

time.sleep(0.01)

lock.acquire()

total.value -= 5

lock.release()

#if \_\_name\_\_ == '\_\_main\_\_':

#

# total = 500

# print(total)

# total = add\_500\_no\_mp(total)

# print(total)

# total = sub\_500\_no\_mp(total)

# print(total)

#if \_\_name\_\_ == '\_\_main\_\_':

#

# total = Value('i', 500)

# add\_proc = Process(target=add\_500\_no\_lock, args=(total,))

# sub\_proc = Process(target=sub\_500\_no\_lock, args=(total,))

#

# add\_proc.start()

# sub\_proc.start()

#

# add\_proc.join()

# sub\_proc.join()

# print(total.value)

if \_\_name\_\_ == '\_\_main\_\_':

total = Value('i', 500)

lock = Lock()

add\_proc = Process(target=add\_500\_lock, args=(total, lock))

sub\_proc = Process(target=sub\_500\_lock, args=(total, lock))

add\_proc.start()

sub\_proc.start()

add\_proc.join()

sub\_proc.join()

print(total.value)

import time

from multiprocessing import Process, Value, Lock

def func(val, lock):

for i in range(500):

time.sleep(0.01)

with lock:

val.value += 1

if \_\_name\_\_ == '\_\_main\_\_':

v = Value('i', 0)

lock = Lock()

procs=Process(target=func, args=(v, lock))

procs.start()

procs.join()

print(v.value)

import time

from multiprocessing import Process, Value

def func(val):

for i in range(500):

time.sleep(0.01)

val.value += 1

if \_\_name\_\_ == '\_\_main\_\_':

v = Value('i', 0)

procs = Process(target=func, args=(v,))

procs.start()

procs.join()

print(v.value)

'''

43

When you use Value you get a ctypes object in shared memory that by default is synchronized using RLock. When you use Manager you get a SynManager object that controls a server process which allows object values to be manipulated by other processes. You can create multiple proxies using the same manager; there is no need to create a new manager in your loop:

Use Manager to create multiple shared objects, including dicts and lists.

Use Manager to share data across computers on a network.

Use Value or Array when it is not necessary to share information across a network and the types in ctypes are sufficient for your needs.

Value is faster than Manager.

'''

import threading

def worker1(v):

#with v.get\_lock():

for i in range(1000000):

ctypes\_int.value += 1

def worker2(v):

#with v.get\_lock():

for i in range(1000000):

ctypes\_int.value += 2

ctypes\_int = threading.Value("i", 0)

print(ctypes\_int.value)

process1 = threading.Thread(

target=worker1, args=[ctypes\_int])

process2 = threading.Thread(

target=worker2, args=[ctypes\_int])

process1.start()

process2.start()

process1.join()

process2.join()

print (ctypes\_int.value)