In [1]:
```
import threading
var=0
def f1():
    global var
    var=var+1
def f2(Lobj):
    for v in range(1000000):
        Lobj.acquire() # lock
        f1() # CS
        Lobj.release() # unlock
def f3():
    global var
    var=0
    Lobj=threading.Lock()
    th1=threading.Thread(target=f2,args=(Lobj,))
    th2=threading.Thread(target=f2,args=(Lobj,))
    th3=threading.Thread(target=f2,args=(Lobj,))
    th1.start()
    th2.start()
    th3.start()
    th1.join()
    th2.join()
    th3.join()

if __name__ == '__main__':
    for v in range(10):
        f3()
        print("var value:{}".format(var))
```

```
var value:3000000
var value:3000000
var value:3000000
var value:3000000
var value:3000000
var value:3000000
var value:3000000
var value:3000000
var value:3000000
var value:3000000
```

In [4]:
```
import threading
var=0
def f1():
    global var
    var=var+1
def f2(Lobj):
    for v in range(1000000):
        Lobj.acquire() # lock
        f1() # CS
        Lobj.release() # unlock
def f3():
    global var
    var=0
    Lobj=threading.Lock()
    th1=threading.Thread(target=f2,args=(Lobj,))
    th2=threading.Thread(target=f2,args=(Lobj,))

    th1.start()
    th2.start()
    th1.join()
    th2.join()

for v in range(10):
    f3()
    print("var value:{}".format(var))
```

```
var value:21366458
var value:2000000
var value:2000000
var value:2000000
var value:2000000
var value:2000000
var value:2000000
var value:2000000
var value:2000000
var value:2000000
```

In [ ]:
```
## Thread - creation - Template

import threading #  STEP 1
def thread_name():
    # code block - execution block

for v in range(N): # N - times
    tobj1=threading.Thread(target=thread_name,args=()) # STEP 2
    tobj1.start() # STEP 3
    tobj1.join() # STEP 4

#############################################################################
import threading #  STEP 1

def thread_name(Lock):
    # code block - execution block
    Lock.acquire() # STEP 6
        # crititcal section
        #  .....
        #  fx()
        #
    Lock.release() # STEP 7

Lock=threading.Lock() # STEP 5

for v in range(N): # N - times
    tobj1=threading.Thread(target=thread_name,args=(Lock,)) # STEP 2
    tobj1.start() # STEP 3
    tobj1.join() # STEP 4
```

In [6]:
```
import threading
class Box(threading.Thread):
    def __init__(self,name):
        super(Box,self).__init__() # calling parent class constructor
        self.name=name

    def run(self):
        print("this is execution block")
        print("My code block - operation")
        print("Exit from Block")

obj1=Box("Thread1")
obj2=Box("Thread2")
obj1.start()
obj2.start()

obj1.join()
obj2.join()
print("Exit from main thread")
```

```
this is execution block
My code block - operation
Exit from Block
this is execution block
My code block - operation
Exit from Block
Exit from main thread
```

In [7]:
```
import threading
class Box(threading.Thread):
    def __init__(self,name):
        super(Box,self).__init__() # calling parent class constructor
        self.name=name

    def run(self):
        print("this is execution block")
        tobj.acquire() # lock
        print("My code block - operation")
        tobj.release() # unlock
        print("Exit from Block")

tobj=threading.Lock()
obj1=Box("Thread1")
obj2=Box("Thread2")
obj1.start()
obj2.start()

obj1.join()
obj2.join()
print("Exit from main thread")
```

```
this is execution block
My code block - operation
Exit from Block
this is execution block
My code block - operation
Exit from Block
Exit from main thread
```

In [ ]:
```
import threading
class Box(threading.Thread): # inheritance
    def __init__(self,name):
        super(Box,self).__init__() # calling parent class constructor
        self.name=name

    def run(self):
        print("this is execution block")
        tobj.acquire() # lock
        fx() # nested call
        tobj.release() # unlock
        print("Exit from Block")

def fx():
    print("This thread execution")
    print("critical section")

tobj=threading.Lock()
obj1=Box("Thread1")
obj2=Box("Thread2")
obj1.start()
obj2.start()

obj1.join()
obj2.join()
print("Exit from main thread")
```

In [ ]:
```
# Cpython - GIL - realparallel execution code ?
# --------
#  multiprocessing - depends on the H/W (CPU arch)
#
# process <-->  process
#   |            |
#  cpu1          cpu2


# import multiprocessing
# multiprocess.Process<==class
# start() ; run() ; join(); is_alive();
#

# current namespace - module - like forking ->(fork()) newprocess(childprocess)
```

In [9]:
```
# file:ab.py                                    import ab
# -------------                                 ab.f1() -><object>
def f1():                                      # myobj=ab.f1()
    class Box:                                 # myobj.port
       port=1234
    obj=Box()
    return obj

myobj=f1()
myobj.port
```

Out[9]:
```
1234
```

In [ ]:
```
# python (DS+function)  -------------------   DataBase(SQL)
#  'insert into ..'           <DBI-module>                  ....
#                                              |select *from table
#  ()[][()]<iterator>  -------------------------
```

In [ ]:

```
D:\DB2>python
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 18 2019, 23:46:00) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import sqlite3
>>> sqlite3
<module 'sqlite3' from 'C:\\Users\\Karthikeyan\\AppData\\Local\\Programs\\Python
\\Python37-32\\lib\\sqlite3\\__init__.py'>
>>>
>>> sqlite3.connect("emp.db")
<sqlite3.Connection object at 0x00681CA0>
>>> db1=sqlite3.connect("emp.db")
>>> db1.cursor()
<sqlite3.Cursor object at 0x0064AC60>
>>> sth=db1.cursor()
>>> sth.execute("create table emp(eid INT,ename TEXT)")
<sqlite3.Cursor object at 0x006A00A0>
>>> sth.execute("insert into emp values(101,'Arun')")
<sqlite3.Cursor object at 0x006A00A0>
>>> sth.execute("insert into emp values(102,'Vijay')")
<sqlite3.Cursor object at 0x006A00A0>
>>>
>>> eid=103
>>> ename="Anu"
>>>
>>> sth.execute("insert into emp values(?,?)",(eid,ename))
<sqlite3.Cursor object at 0x006A00A0>
>>> sth.execute("insert into emp values(334,'xerox')")
<sqlite3.Cursor object at 0x006A00A0>
>>>
>>> sth.execute("select *from emp")
<sqlite3.Cursor object at 0x006A00A0>
>>> sth.fetchone()
(101, 'Arun')
>>> sth.fetchone()
(102, 'Vijay')
>>> sth.fetchone()
(103, 'Anu')
>>> sth.fetchone()
(334, 'xerox')
>>> sth.fetchone()
>>>
>>> sth.execute("select *from emp")
<sqlite3.Cursor object at 0x006A00A0>
>>> sth.fetchall()
[(101, 'Arun'), (102, 'Vijay'), (103, 'Anu'), (334, 'xerox')]
>>>
>>>
>>> sth.execute("select *from emp")
<sqlite3.Cursor object at 0x006A00A0>
>>>
>>> next(sth)
(101, 'Arun')
>>> next(sth)
(102, 'Vijay')
>>> next(sth)
```

```
(103, 'Anu')
>>> next(sth)
(334, 'xerox')
>>> next(sth)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>>
>>> sth.execute("select *from emp")
<sqlite3.Cursor object at 0x006A00A0>
>>>
>>> for var in sth:
...     print(var)
...
(101, 'Arun')
(102, 'Vijay')
(103, 'Anu')
(334, 'xerox')
>>> with open("emp.txt","w") as WH:
...     for var in sth.execute("select *from emp"):
...             WH.write("{}\t{}\n".format(var[0],var[1]))
...
9
10
8
10
>>> with open("emp.txt") as FH:
...     print(FH.read())
...
101     Arun
102     Vijay
103     Anu
334     xerox

>>> with open("D:\\emp.csv") as F:
...     print(F.read())
...
ram,sales,pune,1000
ashi,prod,bglore,2345
xerox,sales,chennai,45900
yahoo,prod,pune,32450
anu,HR,hyd,4560
biju,prod,bglore,4567
vijay,hr,chennai,3453
theeb,sales,hyd,5678
nithin,prod,pune,1236
>>>

import sqlite3
con=sqlite3.connect("Emp1.db")

cur=con.cursor()
cur.execute("create table EMP1(ename TEXT,edept TEXT,ecity TEXT,cost INT)")

with open("D:\\emp.csv") as FH:
    for var in FH:
        n,d,c,cost=var.split(",")
```

```
        cur.execute("insert into EMP values(?,?,?,?)",(n,d,c,cost))

for v in cur.execute("select *from EMP"):
    print(v)
```