Python Programming

Narendra Allam

Chapter 14

Multi-Threading

Topics Covering

- Concurrency
- Prallelism
- Memory Layout os a program - revisited
- Context switching / Time sharing
- Process vs Thread
- Multithreading
- multiprocessing
- Join
- Dead-lock
- live-lock
- lock
- Why python is not good for threading ?

Concurrency:
"Concurrency is the ability of performing more than one activity at the same time".

Parallelism:
"Parallelism is the concurrency which liverages multiple processing units".
Parallism is the subset of concurrency.

- Concurrency can be achievied, even we have single core, by time slicing and context switching.
- Concurrency is logical, Parallelism is physical.

Concurrency can be implemented using python 'threading' module

Process:
A program under execution is called process.
Thread:
A sequence of instructions in the execution flow or a thread of execution

Multi-threading

```
In [2]:
```

```
from threading import Thread
import time

def fun():
    for i in range(1, 11):
        print 'Child:', i
        time.sleep(1)

if __name__ == '__main__':
    thr = Thread(target=fun)
    thr.start()
    for i in range(11, 21):
        print 'Main: ', i
        time.sleep(1)
```

```
Child: 1
Main:  11
Child: Main: 2
 12
Child:Main:  3
 13
Child: 4
Main:  14
Child:Main:   5
15
Child: 6
Main:  16
Child: 7
Main:  17
Child: 8
Main:  18
Child: 9
Main:  19
Child: 10
Main:  20
```

Multi-processing

```
In [2]:
```

```python
from multiprocessing import Process
import time

def fun():
    for i in range(1, 11):
        print ('Child:', i)
        time.sleep(1)

if __name__ == '__main__':
    sub_process = Process(target=fun)
    sub_process.start()
    for i in range(10, 21):
        print ('Main: ', i)
        time.sleep(1)
```

```
Child: 1
Main:  10
Child: 2
Main:  11
Child: 3
Main:  12
Child: 4
Main:  13
Child: 5
Main:  14
Child: 6
Main:  15
Child: 7
Main:  16
Child: 8
Main:  17
Child: 9
Main:  18
Child: 10
Main:  19
Main:  20
```

Critical section:

A common resource being used by multiple threads, and if we are expecting the data consistancy, the section of code which is causing the resource inconsistant is called Critical section.

In [3]:

```
import threading
import time
lk = threading.Lock()

def fun(n):
    for i in range(1, n):
        lk.acquire()
        print 'Child: ', i
        lk.release()
        time.sleep(1)
    print 'Child EXITs'

if __name__ == '__main__':

    thr = threading.Thread(target=fun, args=(21,)) # passing parameters to thread
    thr.start()
    for i in range(21, 31):
        lk.acquire()
        print 'Main: ', i
        lk.release()
        time.sleep(1)
    print 'Main EXITs'
```

```
Child:  1
Main:  21
Child:  2
Main:  22
Child:  3
Main:  23
Child:  4
Main:  24
Child:  5
Main:  25
Child:  6
Main:  26
Child:  7
Main:  27
Child:  8
Main:  28
Child:  9
Main:  29
Child:  10
Main:  30
Child:  11
Main EXITs
Child:  12
Child:  13
Child:  14
Child:  15
Child:  16
Child:  17
Child:  18
Child:  19
Child:  20
Child EXITs
```

In [5]:

```python
import threading
import time
lk = threading.Lock()

data1 = []
data2 = []

def fun(n, main_thr=None):
    for i in range(1, n):

        lk.acquire()
        print 'Child: ' + str(i)
        data1.append(i)
        lk.release()

        time.sleep(1)
    print "Child thread is done!"

if __name__ == '__main__':
    # main_thr = threading.current_thread()
    child_thr = threading.Thread(target=fun, args=(11,))
    child_thr.start()
    for i in range(11, 16):

        lk.acquire()
        print 'Main: ' + str(i)
        data2.append(i)
        lk.release()

        time.sleep(1)

    print 'Main is done, and waiting for child to finish....'
    child_thr.join()
    print "Merging two lists...and summation..."
    print "sum = ", sum(data1 + data2)
    print "Main thread is done!"
```

```
Child: 1
Main: 11
Child: 2
Main: 12
Child: 3
Main: 13
Child: 4
Main: 14
Child: 5
Main: 15
Child: 6
Main is done, and waiting for child to finish....
Child: 7
Child: 8
Child: 9
Child: 10
Child thread is done!
Merging two lists...and summation...
sum =  120
Main thread is done!
```

Producer - Consumer

In [7]:

```
from threading import Thread, current_thread, Lock
import time
lock = Lock()
balance = 0

def deposite(start, stop, amount):
    for i in range(start, stop):
        global balance
        lock.acquire()
        balance += amount
        lock.release()
        print('Father deposited, balance= {} \n'.format(balance))
        time.sleep(1)

    print('depositer Ends')

def withdraw(start, stop, amount):
    for i in range(start, stop):
        global balance
        lock.acquire()
        if balance >= amount:
            print('bal = {}, thread={} '.format(balance, current_thread().name))
            balance -= amount
        else:
            print('bal = {}, Insufficient funds! for {}'.format(balance, current_thr
        lock.release()
        time.sleep(1)
    print('Withdrawer Ends')

def main():

    thr1 = Thread(name = 'Ramu', target=withdraw, args=(0, 10, 10))
    thr2 = Thread(name = 'Somu', target=withdraw, args=(0, 10, 10))
    thr3 = Thread(name = 'Father', target=deposite, args=(0, 10, 20))

    thr3.start()
    thr1.start()
    thr2.start()

    thr1.join()
    thr2.join()
    thr3.join()
    print 'EOD balance=', balance

if __name__ == '__main__':
    main()
```

```
Father deposited, balance= 20

bal = 20, thread=Ramu
bal = 10, thread=Somu
Father deposited, balance= 20
bal = 20, thread=Somu
bal = 10, thread=Ramu

bal = 0, Insufficient funds! for Somu
Father deposited, balance= 20

bal = 20, thread=Ramu
bal = 10, thread=Ramu
bal = 0, Insufficient funds! for Somu
Father deposited, balance= 20

bal = 20, thread=Ramu
Father deposited, balance= 30

bal = 30, thread=Somu
Father deposited, balance= 40
bal = 40, thread=Somu

bal = 30, thread=Ramu
bal = 20, thread=Somu
bal = 10, thread=Ramu
Father deposited, balance= 20

bal = 20, thread=Somu
Father deposited, balance= 30

bal = 30, thread=Ramu
bal = 20, thread=Somu
bal = 10, thread=Ramu
Father deposited, balance= 20

bal = 20, thread=Somu
Father deposited, balance= 30

bal = 30, thread=Ramu
Withdrawer Ends
Withdrawer Ends
depositer Ends
EOD balance= 20
```

Threading and Object Ortientation

In [8]:

```
from threading import Thread, Lock
import time

class Account(object):
    def __init__(self, _balance=0):
        self.lk = Lock()
        self.balance = _balance


    def withdraw(self, withdrawer, amount):
        self.lk.acquire()
        if self.balance >= amount:
            self.balance -= amount
            print '{} withdrawn money! bal {}'.format(withdrawer,
                                                      self.balance)
        else:
            print 'in sufficient funds! for {} bal {}'.format(withdrawer,
                                                              self.balance)
        self.lk.release()

    def deposite(self, depositer, amount):
        self.lk.acquire()
        self.balance += amount
        print '{} deposited money! bal {}'.format(depositer,
                                                  self.balance)
        self.lk.release()

class Depositer(Thread):
    def __init__(self, _name, n, account):
        super(Depositer, self).__init__(name=_name)
        self.n = n
        self.account = account

    def run(self):
        for i in range(self.n):
            self.account.deposite(self.getName(), 20)
            time.sleep(1)

class Withdrawer(Thread):
    def __init__(self, _name, n, account):
        super(Withdrawer, self).__init__(name=_name)
        self.n = n
        self.account = account

    def run(self):
        for i in range(self.n):
            self.account.withdraw(self.getName(), 10)
            time.sleep(1)

if __name__ == '__main__':
    account = Account()

    father   = Depositer('Father', 10, account)
    ramu = Withdrawer('Ramu', 10, account)
    somu = Withdrawer('Somu', 10, account)

    father.start()
    ramu.start()
    somu.start()
    father.join()
    ramu.join()
    somu.join()

    print 'Balance EOD:', account.balance
```

```
Father deposited money! bal 20
Ramu withdrawn money! bal 10
Somu withdrawn money! bal 0
Father deposited money! bal 20
Ramu withdrawn money! bal 10
Somu withdrawn money! bal 0
Father deposited money! bal 20
Somu withdrawn money! bal 10
Ramu withdrawn money! bal 0
Father deposited money! bal 20
Somu withdrawn money! bal 10
Ramu withdrawn money! bal 0
Father deposited money! bal 20
Somu withdrawn money! bal 10
Ramu withdrawn money! bal 0
in sufficient funds! for Somu bal 0
in sufficient funds! for Ramu bal 0
Father deposited money! bal 20
Father deposited money! bal 40
Somu withdrawn money! bal 30
Ramu withdrawn money! bal 20
Somu withdrawn money! bal 10
Father deposited money! bal 30
Ramu withdrawn money! bal 20
Somu withdrawn money! bal 10
Ramu withdrawn money! bal 0
Father deposited money! bal 20
Somu withdrawn money! bal 10
Father deposited money! bal 30
Ramu withdrawn money! bal 20
Balance EOD: 20
```