

---

## BAB I

### PYTHON MULTITHREADED PROGRAMMING

Menjalankan beberapa *thread* mirip dengan menjalankan beberapa program yang berbeda secara bersamaan, namun dengan manfaat berikut :

- Beberapa *thread* dalam proses berbagi ruang data yang sama dengan benang induk dan karena dapat saling berbagi informasi atau berkomunikasi satu sama lain dengan lebih muda daripada jika prosesnya terpisah
- *thread* terkadang disebut proses ringan dan tidak membutuhkan banyak memori atas, mereka lebih murah daripada proses.

Sebuah *thread* memiliki permulaan, urutan eksekusi dan sebuah kesimpulan. Ini memiliki pointer perintah yang melacak dari mana dalam konteksnya saat ini berjalan.

- 
- Hal ini dapat dilakukan sebelum *pre-empted (interrupted)*
  - Untuk sementara dapat ditunda sementara *thread* lainnya yang sedang berjalan ini disebut unggul.

### 1.1 Memulai *Thread* Baru

Untuk melakukan *thread* lain, perlu memanggil metode berikut yang tersedia di modul *thread* :

```
Thread.start_new_thread (function, args [, kwargs] )
```

Pemanggilan metode ini memungkinkan cara cepat dan tepat untuk membuat *thread* baru di linux dan window.

Pemanggilan metode segera kembali dan anak *thread* dimulai dan fungsi pemanggilan dengan daftar *args* telah berlalu. Saat fungsi kembali ujung *thread* akan berakhir.

---

Disini, *args* adalah tuple argumen. Gunakan tuple kosong untuk memanggil fungsi tanpa melewati argumen. *Kwargs* adalah kamus opsional argumen kata kunci.

Contoh :

```
#!/usr/bin/python
```

```
Import thread
```

```
Import time
```

```
# Define a function for the thread
```

```
Def print _time (threadName, delay):
```

```
    Count = 0
```

```
    While count <5:
```

```
        Time.sleep(delay)
```

```
        Count +=1
```

```
        Print " %s : %s " % (threadName, time.ctime(time.time()))
```

```
# Create two thread as follows
```

```
try:
```

```
thread.start _new _thread(print _time, ( "Thread-1 ", 2, ))
```

```
thread.start _new _thread(print _time, ( "Thread-2 ", 4,))
```

```
except:
```

```
    print "Error: unable to start thread "
```

```
while 1:
```

```
pass
```

---

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut :

Thread-1 : Thu Jan 22 15:42:17 2009

Thread-1 : Thu Jan 22 15:42:19 2009

Thread-2 : Thu Jan 22 15:42:19 2009

Thread-1 : Thu Jan 22 15:42:21 2009

Thread-2 : Thu Jan 22 15:42:23 2009

Thread-1 : Thu Jan 22 15:42:23 2009

Thread-1 : Thu Jan 22 15:42:23 2009

Thread-1 : Thu Jan 22 15:42:25 2009

Thread-2 : Thu Jan 22 15:42:27 2009

Thread-2 : Thu Jan 22 15:42:31 2009

Thread-2 : Thu Jan 22 15:42:35 2009

Meskipun sangat efektif untuk benang tingkat rendah, namun modul *thread* sangat terbatas dibandingkan dengan modul yang baru.

---

## 1.2 Modul Threading

Modul *threading* yang lebih baru disertakan dengan Python 2.4 memberikan jauh lebih kuat, dukungan tingkat tinggi untuk *thread* dari modul *thread* dibahas pada bagian sebelumnya.

The *threading* modul mengekspos semua metode dari *thread* dan menyediakan beberapa metode tambahan :

- **threading.activeCount()**

Mengembalikan jumlah objek *thread* yang aktif

- **threading.currentThread()**

Mengembalikan jumlah objek *thread* dalam kontrol benang pemanggil

- **threading.enumerate()**

Mengembalikan daftar semua benda *thread* yang sedang aktif

Selain metode, modul *threading* memiliki *thread* kelas yang mengimplementasikan *threading*. Metode yang disediakan oleh *thread* kelas adalah sebagai berikut :

- **run()**

Metode adalah titik masuk untuk *thread*

- **start()**

Metode dimulai *thread* dengan memanggil metode run

- **join([time])**

Menunggu benang untuk mengakhiri

- **isAlive()**

Metode memeriksa apakah *thread* masih mengeksekusi

- **getName()**

Metode mengembalikan nama *thread*

- **setName()**

Metode menetapkan nama *thread*

## 1.3 Membuat Thread Menggunakan Threading Modul

Untuk melaksanakan *thread* baru menggunakan *threading* harus melakukan hal berikut :

- Mendefinisikan subclass dari *thread* kelas
- Menimpa `_init_ (self [args])` metode untuk menambahkan argumen tambahan

- 
- Menimpa `run(self[args])` metode untuk menerapkan apa *thread* harus dilakukan ketika mulai

---

Setelah membuat baru *thread* subclass, dapat membuah sebuah instance dari itu dan kemudian memulai *thread* baru dengan menerapkan *start()*, yang ada gilirannya panggilan *run()* metode.

Contoh :

```
#!/usr/bin/python

import threading
import time

exitFlag = 0

class myThread (threading.Thread):
    def __init__(self, threadID, name, counter) :
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
    def run(self) :
        print "Starting " + self.name
        print _time(self.name, self.counter, 5)
        print "Exiting "+ self.name
    def print_time(threadName, delay, counter):
        while counter:
            if exitFlag:
                threadName.exit()
            time.sleep(delay)
            print " %s: %s " % (threadName, time.ctime(time.time()))
            counter -= 1

# Create new threads
thread1 = myThread(1, "Thread-1 ", 1)
thread2 = myThread(2, "Thread-2 ", 2)

# Start new threads
thread1.start()
thread2.start()
print "Exiting Main Thread "
```

Ketika kode diatas dijalankan, menghasilkan hasil sebagai berikut:

```
Starting Thread-1
Starting Thread-2
Exiting Main Thread
Thread-1 : Thu Mar 21 09:10:03 2013
Thread-1 : Thu Mar 21 09:10:04 2013
```

---

Thread-2 : Thu Mar 21 09:10:04 2013  
Thread-1 : Thu Mar 21 09:10:05 2013  
Thread-2 : Thu Mar 21 09:10:06 2013  
Thread-1 : Thu Mar 21 09:10:07 2013  
Exiting Thread-1  
Thread-2 : Thu Mar 21 09:10:08 2013  
Thread-2 : Thu Mar 21 09:10:10 2013  
Thread-2 : Thu Mar 21 09:10:12 2013  
Exiting Thread=2

### 1.4 Sinkronisasi *Thread*

*Threading* modul disediakan dengan Python termasuk sederhana untuk menerapkan mekanisme bahwa memungkinkan untuk menyinkronkan *thread* penguncian. Sebuah kunci baru dibuat dengan memanggil *lock()* metode yang mengembalikan kunci baru.

The *acquire (blocking)* metode objek kunci baru digunakan untuk memaksa *thread* untuk menjalankan serempak. Opsional *blocking* parameter memungkinkan untuk mengontrol apakah *thread* menunggu untuk mendapatkan kunci.

Jika *blocking* diatur ke 0, *thread* segera kembali dengan nilai 0 jika kunci tidak dapat diperoleh dan dengan 1 jika kunci dikuisisi. Jika pemblokiran diatur ke 1, blok dan menunggu kunci yang akan dirilis.

The *release()* metode objek kunci baru digunakan untuk melepaskan kunci ketika tidak lagi diperlukan.

Contoh:

```
#!/usr/bin/python
```

```
import threading
```

```
import time
```

```
class myThread (threading.Thread):  
    def __init__(self, threadID, name, counter):  
        threading.Thread.__init__(self)  
        self.threadID = threadID  
        self.name = name  
        self.counter = counter  
    def run(self)  
        print "Starting "+ self.name  
        # Get lock to synchronize threads  
        ThreadLock.acquire()  
        print _time(self.name, self.counter, 3)  
        # Free lock to release next thread  
        ThreadLock.release()
```

```
Def print _time(threadName, delay, counter):  
    while counter:
```



---

```

time.sleep(delay)
print " %s: %s " % (threadName, time.ctime(time.time()))
counter -= 1
threadLock = threading.Lock()
threads = []

# Create new threads
thread1 = myThread(1, "Thread-1,1 ")
thread2 = myThread(2, "Thread-2,2 ")

# Start new Threads
thread1.start()
thread2.start()

# Add threads to thread list
threads.append(thread1)
threads.append(thread2)

# Wait for all threads to complete
for t in threads:
    t.join()
print "Exiting Main thread "
```

Bila kode diatas dieksekusi, maka menghasilkan sebagai berikut :

```

Starting Thread-1
Starting Thread-2
Thread-1: Thu Mar 21 09:11:28 2013
Thread-1: Thu Mar 21 09:11:29 2013
Thread-1: Thu Mar 21 09:11:30 2013
Thread-2: Thu Mar 21 09:11:32 2013
Thread-2: Thu Mar 21 09:11:34 2013
Thread-2: Thu Mar 21 09:11:36 2013
Exiting Main Thread
```

### 1.5 Multithreaded Antrian Prioritas

The queue modul memungkinkan untuk membuat objek antrian baru yang dapat menampung jumlah tertentu item. Ada metode berikut untuk mengontrol antrian :

- **get()**

Menghapus dan mengembalikan item dari antrian

- **put()**

Menambahkan item ke antrian

- **qsize()**

---

Mengembalikan jumlah item yang saat ini dalam antrian

- **empty()**

Mengembalikan benar jika antrian kosong jika tidak, salah

- **full()**

Mengembalikan benar jika antrian penuh jika tidak, salah

Contoh:

```
#!/usr/bin/python
```

```
import Queue
import threading
import time
```

```
exitFlag = 0
```

```
class myThread (threading.Thread):
    def __init__(self, threadID, name, q):
        threading.Thread.__init__(self)
        self.name = name
        self.q = q
    def run(self):
        print "Starting "+ self.name
        process_data(self.name, self.q)
        print "Exiting "+ self.name
```

```
def process_data(threadName, q):
    while not exitFlag:
        queueLock.acquire()
        if not workQueue.empty():
            data = q.get()
            queueLock.release()
            print "%s processing %s" % (threadName, data)
        else:
            queueLock.release()
            time.sleep(1)
```

```
threadList = [ "Thread-1 ", "Thread-2 ", "Thread-3 "]
nameList = [ "One ", "Two ", "Three ", "Four ", "Five "]
queueLock = threading.Lock()
workLock = Queue.Queue(10)
threads = []
```

---

```
threadID = 1

# Create new threads
For tName in threadList:
    thread = myThread(threadID, tName, workQueue)
    thread.start()
    thread.append(thread)
    threadID +=1

# Fill the queue
queueLock.acquire()
for word in nameList:
    workQueue.put(word)
queueLock.release()

# Wait for queue to empty
while not workQueue.empty():
    pass

# Notify threads it's time to exit
exitFlag = 1

# Wait for all threads to complete
For t in threads:
    t.join()
print "Exiting Main Thread "
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut:

```
Starting Thread-1
Starting Thread-2
Starting Thread-3
Thread-1 processing One
Thread-2 processing Two
Thread-3 processing Three
Thread-1 processing Four
Thread-2 processing Five
Exiting Thread-3
Exiting Thread-1
Exiting Thread-2
Exiting Main Thread
```