Multitasking

Objectives

To run external programs, in this case other Python scripts, using a variety of methods, first using the **subprocess** module and then using **multiprocessing**.

Reference Material

Chapter 13 Multitasking.

Question 1

In the labs directory you will find a simple Python program, **client.py**, which lists files to STDOUT. The name of the file is specified at the command line, and if it cannot be read then an error is returned, using exit.

a) Now call the Python program client.py from another, passing a filename. If you can't think of a file to list, use the current program, or use the 'words' file.

   Output an error message if, for some reason, the client.py fails. Test this by:
   passing a non-existent file name
   calling a non-existent program

b) Modify the calling program to use a pipe and capture its output in a list. Print out the number of lines returned by the client.py program. Test as before.

Question 2

The purpose of this exercise is to experiment with different scenarios using the multiprocessing module. This is best demonstrated using a multi-core machine, so you might first like to check if that is the case. If not then the exercise is still valid, but not quite so interesting.

Word prefixes are also called *stems*. We have written a program, **stems.py**, which reads the words file and generates the most popular stems of 2 to ***n*** characters long. It uses the "mytimer" module we created in a previous exercise, which you should make available.

Run the supplied **stems.py** program and note the time taken. You will note that no word exceeds 28 characters, so ***n*** could be 28, however we can increase the value of ***n*** in order to obtain a longer runtime and demonstrate multiprocessing.

This time could be better used by splitting the task between cores. Using the multitasking module will require the stem search to be moved to a function.

Make sure that all of the rest of the code is only executed in main (`if __name__ == '__main__':` test).

Scenarios:

a) ***n*** worker processes
This is where we split the task such that each stem length search runs in its own child process.

b) 2 worker processes ***n***/2 stem sizes each.
This assumes 2 CPU cores. It will require two processes to be launched explicitly, and each to be given a range of stem lengths to handle.

c) 2 worker processes using a queue.
This assumes 2 CPU cores. As in b), but instead of passing a range, pass the stem lengths through a queue. Make sure you have a protocol for the worker processes to detect that the queue has finished.

Note: There is a known Python bug, which can be ignored, where the multiprocessing module sometimes gives:
`Exception in thread QueueFeederThread (most likely raised during interpreter shutdown)`
This is fixed in Python 2.7 (allegedly).

If time allows…

Question 3

Recall the SharePrices.py program in the 05 Collections exercises. Your job in this exercise is to make the program multi-threaded.

If you did not complete the previous exercise, then take a copy from the `solutions/05 Collections` directory.

The `SharePrices` dictionary itself needs to be changed. The value is now a list, the first element is the sequence (thread) number, and the second is the share price. Initialise `SharePrices` as follows:

```
SharePrices = {'Global Motors'   :['0',50],
               'Big Blue Inc.'   :['0',50],
               'Gates Software'  :['0',50],
               'Banana Computers':['0',50]}
```

You will need two functions:

`SetStockPrices`

Takes one argument – the sequence number.-
Loop continually, setting the sequence number, and the share price (as before), in `SharePrices`.

`ReadStockPrices`

No arguments are required to this function.
Loop continually printing out the details of the `SharePrices` dictionary every two seconds.

Note that each time we print out `SharePrices` the sequence number on each line should be the same for each member.

For example:

```
1 Banana Computers   $01.30
1 Global Motors      $02.14
1 Big Blue Inc.      $01.08
1 Gates Software     $02.70

3 Banana Computers   $01.03
3 Global Motors      $09.89
3 Big Blue Inc.      $01.16
3 Gates Software     $01.11
```

is OK, but:

```
2 Banana Computers   $01.30
1 Global Motors      $02.14
3 Big Blue Inc.      $01.08
1 Gates Software     $02.70

1 Banana Computers   $01.03
3 Global Motors      $09.89
```

```
2 Big Blue Inc.      $01.16
3 Gates Software     $01.11
```

is not!

So you will have to apply a lock each time you want to read or write to `SharePrices`.

Run four threads for each function. The sequence number is passed into each `SetStockPrices` thread and should be between 0 and 3.

Question 4

Find the Python program `Fcopy.py`. It copies files from your machine to the Instructor's machine, timing the operation. Run this program first to get a benchmark timing.

a. Write a multi-threaded version, using Queues. Have two worker threads (you can experiment with other number of threads if you wish) which actually do the copy, the main thread will pass filenames to these threads using a queue.

   After the copy has been done, each worker thread should pass the new filename to an additional thread that will delete the file, using a second queue.

   Finally, don't forget to place a marker (like False) onto the queue to indicate the end of the list, and wait (join) for all threads to complete.

   Did the multithreaded version run quicker?

b. Convert your multithreaded program to use the multiprocessing module. Does that run quicker?

Solutions

Question 1

```
from subprocess import *
import os
import sys

(a)
proc = Popen([sys.executable, 'client.py', 'words'])
proc.wait()
print "Child exited with",proc.returncode

(b)
proc = Popen([sys.executable, 'client.py', 'words'],
             stdout=PIPE, stderr=PIPE)
(output, error) = proc.communicate()

if error != None:
    print "error:", error

print "output:", output
```

Question 2

Base time on a machine with dual-core 2.66Gz CPU: Process : 1.125 seconds

Time for scenario a) (n processes) Process : 71.641 seconds

b) Process : 4.969 seconds

c) Process : 4.875 seconds

```
a)
import mytimer
from multiprocessing import Process
##############################################################
def stem_search(stems, stem_size):
    best_stem = ""
    best_count = 0
    for (stem,count) in stems.items():
       if stem_size == len(stem) and count > best_count:
           best_stem  = stem
           best_count = count
    if best_stem:
        print "Most popular stem of size",stem_size,"is:", \
               best_stem,"(occurs",best_count,"times)"
##############################################################
if __name__ == '__main__':
    mytimer.start_timer()
    stems = {}
    for row in open ("words"):
        for count in range(1,len(row)):
            stem = row[0:count]
            if stem in stems:
                stems[stem] += 1
            else:
```

```
                stems[stem] = 1
    mytimer.end_timer('Load')

    # Process the stems
    mytimer.start_timer()
    n = 30
    for stem_size in range(2,n+1):
        proc = Process(target=stem_search,
                       args=(stems,stem_size))
        proc.start()
        processes.append(proc)
    for proc in processes:
        proc.join()
    mytimer.end_timer('Process')

b)
import mytimer
from multiprocessing import Process
##############################################################
def stem_search(stems, start, end):
    for stem_size in range(start,end):
        best_stem = ""
        best_count = 0
        for (stem,count) in stems.items():
            if stem_size == len(stem) and count > best_count:
                best_stem  = stem
                best_count = count

        if best_stem:
            print "Most popular stem of size", \
                  stem_size,"is:",             \
                  best_stem,"(occurs",best_count,"times)"
##############################################################
if __name__ == '__main__':
    mytimer.start_timer()
    stems = {}
    for row in open ("words"):
        for count in range(1,len(row)):
            stem = row[0:count]
            if stem in stems:
                stems[stem] += 1
            else:
                stems[stem] = 1
    mytimer.end_timer('Load')
    # Process the stems
    mytimer.start_timer()
    n = 30
    proc1 = Process(target=stem_search,
                    args=(stems,2,int(n/2)+1))
    proc1.start()
    proc2 = Process(target=stem_search,
                    args=(stems,int(n/2)+1,n+1))
    proc2.start()
    proc1.join()
    proc2.join()
    mytimer.end_timer('Process')

c)
```

```
import mytimer
from multiprocessing import Process, Queue
##############################################################
def stem_search(stems, queue):
    stem_size = 1
    while stem_size > 0:
        stem_size = queue.get()
        best_stem = ""
        best_count = 0

        for (stem,count) in stems.items():
            if stem_size == len(stem) and count > best_count:
                best_stem  = stem
                best_count = count
        if best_stem:
            print "Most popular stem of size", \
                   stem_size,"is:",            \
                   best_stem,"(occurs",best_count,"times)"
##############################################################
if __name__ == '__main__':
    mytimer.start_timer()
    stems = {}
    for row in open("words"):
        for count in range(1,len(row)):
            stem = row[0:count]
            if stem in stems:
                stems[stem] += 1
            else:
                stems[stem] = 1
    mytimer.end_timer('Load')
    mytimer.start_timer()
    n = 30
    queue = Queue()
    proc1 = Process(target=stem_search, args=(stems,queue))
    proc2 = Process(target=stem_search, args=(stems,queue))
    proc1.start()
    proc2.start()
    for stem_size in range(2,n):
        queue.put(stem_size)
    queue.put(0)
    queue.put(0)
    proc1.join()
    proc2.join()
    mytimer.end_timer('Process')
```

If time allows...
Question 3

```
from threading import Thread
from threading import Lock
import time
import random

SharePrices = {'Global Motors'   :['0',50],
               'Big Blue Inc.'   :['0',50],
               'Gates Software'  :['0',50],
               'Banana Computers':['0',50]}

csSharePrices = Lock()

##########################################################

def SetStockPrices(seq):
    # Updates stock prices with random price changes
    global SharePrices

    while True:
        csSharePrices.acquire()    # TODO
        for key,sp in SharePrices.items():
            SharePrices[key][0] = seq
            SharePrices[key][1] = max(1.0,
                                      sp[1] * ( 1 +
((random.random() - 0.5)/0.5) * 0.05))

        csSharePrices.release()    # TODO

##########################################################

def ReadStockPrices():

    global SharePrices

    while True:

        csSharePrices.acquire()    # TODO
        for key,sp in SharePrices.items():
            print("{} {:<18s} ${:05.2f}".\
                      format(sp[0],key,sp[1]))
        print()

        csSharePrices.release()    # TODO
        time.sleep(2)

##########################################################
```

```
if __name__ == '__main__':

    tids = []

    # start share price update thread
    for i in range(0,4):
        th_set =
            Thread(target=SetStockPrices,args=str(i))
        th_set.start()

    # Wait for request from client
    for i in range(0,4):

        th_st = Thread( target=ReadStockPrices )
        th_st.start()
        tids.append(th_st)

    for tid in tids:
        tid.join()
```

Question 4

Here is our multithreaded version (without the timing routines), which actually runs slightly slower than the single threaded version.

```
import platform
import os.path
import sys
import glob
from threading import Thread
from Queue import Queue

##########################################################

def RemoveThread(*args):
    TargetDir,queue = args

    while True:
        FName = queue.get()
        if not FName: break
        os.remove(TargetDir + FName)

def WorkerThread(*args):

    TargetDir,queue,rqueue = args

    while True:

        FName = queue.get()
        if not FName: break

        Data = open(FName,'rb').read()

        FName = os.path.basename(FName)
        fh = open(TargetDir + FName,'wb')
        fh.write(Data)
        fh.close()
```

```
        rqueue.put(FName)

##########################################################

OperSys,Host = platform.uname()[:2]
Source = './Bitmaps'

if not os.path.isdir(Source):
    sys.exit("Unable to access "+Source)

Source = Source + '/*'

if OperSys == 'Windows':
    TargetDir = '\\\\INSTRUCTOR\\Shared\\' + Host + '\\'
else:
    TargetDir = '/mnt/hgfs/\\\\INSTRUCTOR/' + Host + '/'

if not os.path.isdir(TargetDir):
    os.mkdir(TargetDir)

start_timer()
NumThreads = 2

for i in range(0,10):

    print 'Loop',i
    queue  = Queue()
    rqueue = Queue()

    Tids = []

    for i in range(0,NumThreads):
        Tids.append(Thread(target=WorkerThread,
                    args=(TargetDir,queue,rqueue)))

    rth = Thread(target=RemoveThread,
                 args=(TargetDir,rqueue))

    for th in Tids:
        th.start()

    rth.start()

    for FName in glob.iglob(Source):
        queue.put(FName)

    for th in Tids:
        queue.put(False)

    for th in Tids:
        th.join()

    rqueue.put(False)
    rth.join()

end_timer("Threaded:")
```

This is our multiprocessing version (without the timing routines), which runs considerably faster:

```
import platform
import os.path
import glob
import sys
from multiprocessing import Process, Queue
##########################################################

def RemoveProcess(*args):
    TargetDir,queue = args

    while True:
        FName = queue.get()
        if not FName: break
        os.remove(TargetDir + FName)


def WorkerProcess(*args):

    TargetDir,queue,rqueue = args

    while True:

        FName = queue.get()
        if not FName: break

        Data = open(FName,'rb').read()

        FName = os.path.basename(FName)
        fh = open(TargetDir + FName,'wb')
        fh.write(Data)
        fh.close()

        rqueue.put(FName)

##########################################################
if __name__ == '__main__':

    OperSys,Host = platform.uname()[:2]
    Source = './Bitmaps'

    if not os.path.isdir(Source):
        sys.exit("Unable to access "+Source)

    Source = Source + '/*'

    if OperSys == 'Windows':
        TargetDir = '\\\\INSTRUCTOR\\Shared\\'+Host+'\\'
    else:
        TargetDir = '/mnt/hgfs/\\\\INSTRUCTOR/'+Host+'/'

    if not os.path.isdir(TargetDir):
        os.mkdir(TargetDir)

    start_timer()
    NumProcs = 2
```

```
for i in range(0,10):
    print 'Loop',i
    queue  = Queue()
    rqueue = Queue()

    Pids = []

    for i in range(0,NumProcs):
        Pids.append(Process(target=WorkerProcess,
                    args=(TargetDir,queue,rqueue)))

    rth = Process(target=RemoveProcess,
                  args=(TargetDir,rqueue))

    for th in Pids:
        th.start()

    rth.start()

    for FName in glob.iglob(Source):
        queue.put(FName)

    for th in Pids:
        queue.put(False)

    for th in Pids:
        th.join()

    rqueue.put(False)
    rth.join()

end_timer("Multiprocessing:")
```