Exercise 13 – Multitasking and asyncio

Objective

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

Questions

1. In the **labs** or (on Linux) your home 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.

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.

    Note: IDLE, and some other IDEs, does not display output from the child processes run by the multiprocessing module. So, run your code from the command-line.

Word prefixes are also called *stems*. We have written a program, **stems.py**, that 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*** to obtain a longer runtime and demonstrate multiprocessing.

This time could be better used by splitting the task between cores. Using the **multiprocessing** module will require the stem search to be moved to a function. Make sure that all 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.

Solutions

Question 1

```
import subprocess
import os
import sys

#(a)
proc = subprocess.run([sys.executable, 'client.py', 'words'])
print('Child exited with', proc.returncode)

#(b)
proc = subprocess.run([sys.executable, 'client.py', 'words'],
        stdout=subprocess.PIPE, stderr=subprocess.PIPE)

if proc.stderr != None:
   print('error:', proc.stderr.decode())

print('output:', proc.stdout.decode())
```

Question 2

The timings will obviously vary depending on the machine:

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)')
        return
    if __name__ == '__main__':
        mytimer.start_timer()
        stems = {}
        for row in open('words', 'r'):
            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)')
    return

        if __name__ == '__main__':
            mytimer.start_timer()
            stems = {}
            for row in open('words', 'r'):
                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)')
            return

if __name__ == '__main__':
    mytimer.start_timer()
    stems = {}
    for row in open('words', 'r'):
        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')
```