Homework 07: Concurrency

Due Date: Apr 13, 2020, 08:00am

Firstname Lastname: Yunya Wang

E-mail: yw4509@nyu.edu

Enter your solutions and submit this notebook

Problem 1 (60 Points)

Let us consider the Gamma function, or the Euler integral of the second kind:

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\, dt,$$

and in this HW we consider real $x > 0$.

(Here is more on the Gamma function https://en.wikipedia.org/wiki/Gamma_function (https://en.wikipedia.org/wiki/Gamma_function) . It is not needed for this HW assignment.)

1.1 (Points 15):

Write a function (in the cell below) that sequentially calculates the given Gamma integral.

In [111]:
```
import numpy as np
import math
import time
```

In [2]:
```
def calculate_gamma(x, bound_1, bound_2, number_of_steps):
    # sequential version to calculate Gamma(x):
    # where we approximate the given integral,
    # like this a discrete sum in number_of_steps
    # equidistant points on the interval [bound_1, bound_2]
    area=0
    step = (bound_2-bound_1)/number_of_steps
    bound = np.arange(bound_1,bound_2+step,step)
    for i in range(len(bound)-1):
        value1 = (bound[i]**(x-1))*math.exp(-bound[i])
        value2 = (bound[i+1]**(x-1))*math.exp(-bound[i+1])
        if value2>=value1:
            height=value1
        height = value2
        area+=height*step
    return area
```

1.2 (Points 5)

Evaluate, $\Gamma(6)$ by using `calculate_gamma(x, bound_1, bound_2, number_of_steps)` and the error of this computation.

As arguments, use `x=6, bound_1=0, bound_2=1000, number_of_steps=10_000_000` . We know that $\Gamma(x) = x!$, so $\Gamma(6) = 5! = 120$.

In [3]:
```
st = time.time()
approx = calculate_gamma(6, 0, 1000, 10_000_000)
et = time.time()
print(f'Result is {approx}, Finished in time {(et-st):0.4f} seconds')
```

```
Result is 119.99999999994274, Finished in time 38.0628 seconds
```

In [4]:
```
error = approx - 120
print(f'error is {error}')
```

```
error is -5.725553364754887e-11
```

Write two functions to calculate $\Gamma(x)$ by using:

1.3.1 (Points 15) threading with N=4 threads;

1.3.2 (Points 15) multiprocessing with N=4 processes.

1.3.3 (Points 10) Compare the times of the three versions and write a short explanation of what you are observing.

How does the answer change when N=8 and why?

In [112]:
```
from multiprocessing import cpu_count
print('number of CPU cores:', cpu_count())
```

```
number of CPU cores: 4
```

1.3.1

In [113]:
```
from queue import Queue
from threading import Thread
from threading import Lock
import logging

logging.basicConfig(format='(%(threadName)-9s) %(message)s',)
```

In [131]:
```
bound_1=0
bound_2 = 1000
number_of_steps = 10_000_000
step = (bound_2-bound_1)/number_of_steps
bound = np.arange(bound_1,bound_2+step,step)

#we further divide the bound per 1000 steps
chunks = [bound[i:i+1000+1] for i in range(0,len(bound)-1000,1000)]
print(len(chunks))
```

```
10000
```

In [158]:
```
y = 0
def calculate_gamma_thread(x,chunk):
    while True:
        global y
        chunk = q.get()
        for i in range(len(chunk)-1):
            with Lock():
                value1 = (chunk[i]**(x-1))*math.exp(-chunk[i])
                value2 = (chunk[i+1]**(x-1))*math.exp(-chunk[i+1])
                if value2>=value1:
                    height=value1
                height = value2
                y+=height*step
        q.task_done()
```

In [159]:
```
st = time.time()
q=Queue()
num_threads=4
for i in range(num_threads):
    worker = Thread(target=calculate_gamma_thread,
                    args=(6,q,))
    worker.setDaemon(True)
    worker.start()

for chunk in chunks:
    q.put(chunk)

q.join()
et = time.time()

print(f'results ran by 4 threads {y} ,Finished in {(et - st):.4f} second
s')
```

```
results ran by 4 threads 119.73463439161083 ,Finished in 51.2276 second
s
```

1.3.2

In [126]:
```
from multiprocessing.pool import Pool
```

In [127]:
```
def calculate_gamma_pool(x,chunk):
    area=0
    for i in range(len(chunk)-1):
        value1 = (chunk[i]**(x-1))*math.exp(-chunk[i])
        value2 = (chunk[i+1]**(x-1))*math.exp(-chunk[i+1])
        if value2>=value1:
            height=value1
        height = value2
        area+=height*step
    return area
```

In [128]:
```
bound_1=0
bound_2 = 1000
number_of_steps = 10_000_000
step = (bound_2-bound_1)/number_of_steps
bound = np.arange(bound_1,bound_2+step,step)

#we further divide the bound per 1000 steps
chunks = [bound[i:i+1000+1] for i in range(0,len(bound)-1000,1000)]
print(len(chunks))
```

```
10000
```

In [161]:
```
st = time.time()
with Pool(4) as p:
    results = p.starmap(calculate_gamma_pool,[(6,chunk) for chunk in chu
nks])
et = time.time()
print(f'reults ran by 4 threads {sum(results)} ,Finished in {(et - st):.
4f} seconds')
```

```
reults ran by 4 threads 119.99999999999991 ,Finished in 18.3949 seconds
```

In [162]:
```
st = time.time()
with Pool(8) as p:
    results = p.starmap(calculate_gamma_pool,[(6,chunk) for chunk in chu
nks])
et = time.time()
print(f'reults rand by 8 threads {sum(results)} ,Finished in {(et - st):
.4f} seconds')
```

```
reults rand by 8 threads 119.99999999999991 ,Finished in 18.3475 second
s
```

1.3.3 (Points 10) Compare the times of the three versions and write a short explanation of what you are observing.

How does the answer change when N=8 and why?

The reuturning result is the same but the time varies between the three.

The fastest is the multiproccessing with Pool. It finished in 18.3949 secondswith 4 threads and 18.3475 seconds with 8 threads.

The second fastest is the original one with no threading or pool. It takes 38 seconds.

The slowest is the thresding one with almost 51.2276 seconds. It is mainly due to we need to the part q.join()

After we change to 8 threads, the time doesn't change much for the pool but increased for the threading one. Because the with more threads, it takes longer time to join.

Problem 2 (40 points)

Website uptime is the time that a website or web service is available to the users over a given period.

The task is to build an application that checks the uptime of websites.

- The application should go over a list of website URLs and checks if those websites are up.
- Instead of performing a classic HTTP GET request, it performs a HEAD request so that it does not affect traffic significantly.
- If the HTTP status is in the danger ranges (400+, 500+), a message is casted.

Here are some useful functions:

In [46]:
```
#### _website uptimer_ ####

import time
import logging
import requests

class WebsiteDownException(Exception):
    pass

def ping_website(address, timeout=20):
    """
    Check if a website is down. A website is considered down
    if either the status_code >= 400 or if the timeout expires

    Throw a WebsiteDownException if any of the website down conditions a
re met
    """
    try:
        response = requests.head(address, timeout=timeout)
        if response.status_code >= 400:
            logging.warning("Website %s returned status_code=%s" % (addr
ess, response.status_code))
            raise WebsiteDownException()
    except requests.exceptions.RequestException:
        logging.warning("Timeout expired for website %s" % address)
        raise WebsiteDownException()

def check_website(address):
    """
    Utility function: check if a website is down, if so, notify the user
    """
    try:
        ping_website(address)
    except WebsiteDownException:
        print('The websie ' + address + ' is down')
```

You need a website list to try our system out. Create your own list or use the following one.

In [52]:
```
WEBSITE_LIST = [
    'http://amazon.co.uk',
    'http://amazon.com',
    'http://facebook.com',
    'http://google.com',
    'http://google.fr',
    'http://google.es',
    'http://google.co.uk',
    'http://gmail.com',
    'http://stackoverflow.com',
    'http://github.com',
    'http://heroku.com',
    'http://really-cool-available-domain.com',
    'http://djangoproject.com',
    'http://rubyonrails.org',
    'http://basecamp.com',
    'http://trello.com',
    'http://shopify.com',
    'http://another-really-interesting-domain.co',
    'http://airbnb.com',
    'http://instagram.com',
    'http://snapchat.com',
    'http://youtube.com',
    'http://baidu.com',
    'http://yahoo.com',
    'http://live.com',
    'http://linkedin.com',
    'http://netflix.com',
    'http://wordpress.com',
    'http://bing.com',
]
```

A serial version of the *website uptimer* can be written as:

In [53]:
```
import time

start_time = time.time()

for address in WEBSITE_LIST:
    check_website(address)

end_time = time.time()

print("Time for Serial: %ssecs" % (end_time - start_time))
```

```
(MainThread) Timeout expired for website http://really-cool-available-d
omain.com

The websie http://really-cool-available-domain.com is down

(MainThread) Timeout expired for website http://another-really-interest
ing-domain.co

The websie http://another-really-interesting-domain.co is down

(MainThread) Website http://netflix.com returned status_code=405

The websie http://netflix.com is down
Time for Serial: 4.839232921600342secs
```

You should build two versions of the **website uptimer**, by using:

2.1 (Points 15) threading with N=4 threads;

2.2 (Points 15) multiprocessing with N=4 processes.

2.3 (Points 10)

Compare the times of the three versions and write a short explanation of what you are observing.

How does the answer change when N=8 and why?

2.1

In [91]:
```
def check_website_thread(address):
    while True:
        address = q.get()
        try:
            ping_website(address)
        except WebsiteDownException:
            print('The websie ' + address + ' is down')
        q.task_done()
```

In [98]:
```
st = time.time()
q = Queue()
for i in range(4):
    worker = Thread(target=check_website_thread,
                    args=(q,))
    worker.setDaemon(True)
    worker.start()

for link in WEBSITE_LIST:
    q.put(link)

q.join()
et = time.time()
print("Time for 4 threads: %ssecs" % (et - st))
```

```
(Thread-232) Timeout expired for website http://really-cool-available-d
omain.com
(Thread-232) Timeout expired for website http://another-really-interest
ing-domain.co

The websie http://really-cool-available-domain.com is down
The websie http://another-really-interesting-domain.co is down

(Thread-229) Website http://netflix.com returned status_code=405

The websie http://netflix.com is down
Time for 4 threads: 2.5213460922241 21secs
```

In [99]:
```
st = time.time()
q = Queue()
for i in range(8):
    worker = Thread(target=check_website_thread,
                    args=(q,))
    worker.setDaemon(True)
    worker.start()

for link in WEBSITE_LIST:
    q.put(link)

q.join()
et = time.time()
print("Time for 8 threads: %ssecs" % (et - st))
```

```
(Thread-234) Timeout expired for website http://really-cool-available-d
omain.com
(Thread-234) Timeout expired for website http://another-really-interest
ing-domain.co

The websie http://really-cool-available-domain.com is down
The websie http://another-really-interesting-domain.co is down

(Thread-234) Website http://netflix.com returned status_code=405

The websie http://netflix.com is down

(Thread-233) Timeout expired for website http://baidu.com

The websie http://baidu.com is down
Time for 8 threads: 2.814148187637329secs
```

2.2

In [94]:
```
start_time = time.time()

with Pool(4) as p:
    p.map(check_website,WEBSITE_LIST)

end_time = time.time()

print("Time for Pool(4): %ssecs" % (end_time - start_time))
```

```
(MainThread) Timeout expired for website http://really-cool-available-d
omain.com

The websie http://really-cool-available-domain.com is down

(MainThread) Timeout expired for website http://another-really-interest
ing-domain.co

The websie http://another-really-interesting-domain.co is down

(MainThread) Website http://netflix.com returned status_code=405

The websie http://netflix.com is down
Time for Pool(4): 1.3936619758605957secs
```

In [95]:
```
start_time = time.time()

with Pool(8) as p:
    p.map(check_website,WEBSITE_LIST)

end_time = time.time()

print("Time for Pool(8): %ssecs" % (end_time - start_time))
```

```
(MainThread) Timeout expired for website http://really-cool-available-d
omain.com

The websie http://really-cool-available-domain.com is down

(MainThread) Timeout expired for website http://another-really-interest
ing-domain.co

The websie http://another-really-interesting-domain.co is down

(MainThread) Website http://netflix.com returned status_code=405

The websie http://netflix.com is down
Time for Pool(8): 1.245030164718628secs
```

2.3

The result is the same but the time varies between three of them.

Original time:4.839232921600342secs

Threading4:2.521346092224121secs

Threading8:2.814148187637329secs

Pool4:1.1.3936619758605957secs

Pool8:1.245030164718628secs

The fastest is the Pool. When we increase threads from 4 to 8, the processing time decreased a lot as now we further split them into 8 running paralle at the same time.

The second fastest is Threading. Threading 4 runs faster than threading 8 mainly due to q.join() part. In threading 8, which causes the main thread to wait for the queue to finish processing all the tasks.

The slowest is the original one

In [ ]:

In [ ]: