Title: Comparison of Programming Paradigms: Synchronous, Multithreading, Multiprocessing, and Asynchronous

1. Introduction

Implementing four different program execution paradigms. Synchronous, Multithreading, Multiprocessing and Asynchronous. Comparing each execution time and analysing each paradigms.

2. Selected Program

We have selected four paradigms to analyse performing IO operation reading the text and converting all letters into capital letters.

Asynchronous:

```
import time

def process_data_synchronously(data):
process_data = ""
for char in data:
time.sleep(0.1)
process_data += char.upper()
return process_data

with open('input_file.txt', 'r') as input_file:
data = input_file.read()

start_time = time.time()
processed_data = process_data_synchronously(data)
end_time = time.time()

with open('input_file.txt' , 'w') as output_file:
output_file.write(processed_data)

execution_time = end_time - start_time
```

```
print(f"Execution time of Synchronous program is:
{execution_time}")
```

Multithreading:

```
import threading
import time

def process_data_thread(data, result):
processed_data = ""
for char in data:
time.sleep(0.1)
processed_data += char.upper()
result.append(processed_data)

def process_data_concurrently(data):
chunk_size = len(data) // NUM_THREADS
threads = []
results = []
start_index = 0

# Create threads to process data chunks concurrently
for _ in range(NUM_THREADS):
end_index = start_index + chunk_size
thread_data = data[start_index:end_index]
result = []
thread = threading.Thread(target=process_data_thread,
args=(thread_data, result))
threads.append(thread)
results.append(result)
thread.start()
```

```
start_index = end_index

# Wait for all threads to complete
for thread in threads:
thread.join()

# Concatenate processed data from all threads
processed_data = ""
for result in results:
processed_data += result[0]

return processed_data

NUM_THREADS = 4

with open('input_file.txt', 'r') as input_file:
data = input_file.read()

start_time = time.time()
processed_data = process_data_concurrently(data)
end_time = time.time()

with open('output_file.txt', 'w') as output_file:
output_file.write(processed_data)

execution_time = end_time - start_time
print(f"Execution time of Multithreading program is: {execution_time}")
```

Multiprocessing:

```
from multiprocessing import Process, Queue
import time

def process_data_process(data, result):
processed_data = ""
for char in data:
time.sleep(0.1) # Simulate processing time
processed_data += char.upper()
result.put(processed_data)

def process_data_multiprocess(data):
chunk_size = len(data) // NUM_PROCESSES
processes = []
results = Queue()
start_index = 0

# Create processes to process data chunks concurrently
for _ in range(NUM_PROCESSES):
end_index = start_index + chunk_size
process_data = data[start_index:end_index]
process = Process(target=process_data_process,
args=(process_data, results))
processes.append(process)
process.start()
start_index = end_index

# Wait for all processes to complete
for process in processes:
process.join()
```

```
# Concatenate processed data from all processes
processed_data = ""
while not results.empty():
processed_data += results.get()

return processed_data

NUM_PROCESSES = 4

with open('input_file.txt', 'r') as input_file:
data = input_file.read()

start_time = time.time()
processed_data = process_data_multiprocess(data)
end_time = time.time()

with open('output_file.txt', 'w') as output_file:
output_file.write(processed_data)

execution_time = end_time - start_time
print(f"Execution time of Multiprocessing program is:
{execution_time}")
```

Asynchronous:

```
import asyncio
import time

async def process_data_async(data):
processed_data = ""
for char in data:
```

```
await asyncio.sleep(0.1)  # Simulate processing time
processed_data += char.upper()
return processed_data

async def main():
with open('input_file.txt', 'r') as input_file:
data = input_file.read()

start_time = time.time()
processed_data = await process_data_async(data)
end_time = time.time()

with open('output_file.txt', 'w') as output_file:
output_file.write(processed_data)

execution_time = end_time - start_time
print(f"Execution time of Asynchronous program is:
{execution_time}")

# Run the asynchronous code within the existing event
loop
await main()
```

3. **Implementation**
   Implemented each programming paradigm to achieve the same functionality. Code snippets for each are given above.

4. **Execution and Recording**
   Executed each program in the same machine and execution time was recorded in each programming paradigm.

5. **Analysis and comparison**
   Comparison was done between all four programming paradigms to identify performance. Respective weakness and strength is measured.

6. Presentation

Execution time taken by each is shown in table below:

Paradigm	Program Execution (ms)
Synchronous	2.5034420490264893
Multithreading	0.6311659812927246
Multiprocessing	0.07044696807861328
Asynchronous	2.427222728729248

7. Conclusion

Provided valuable insight into performance characteristics of different programming paradigm.