Manisa Celal Bayar University – Department of Computer Engineering
CSE 3237 Parallel Programming – Final Exam

Name and Surname	
Student Id	
Signature	

Question	1	2	3	4	Total
Score					

You have 75 minutes, gl hf.

Assoc. Prof. Dr. Bora CANBULA

QUESTIONS

Q1 (25 Points) Mark the statements given below as True or False.

T/F	Statement
F	Global Interpreter Lock exists for preventing threads from raising a race condition.
T	Processes are not affected from Global Interpreter Lock.
F	Duplex Pipe is a pair of threads which are running concurrently.
F	Embarrassingly Parallel is a kind of problem that impossible to parallelize its solution.
T	The main difference between a function and a coroutine that coroutines are awaitable.

Q2 (25 Points) Write an **asynchronous context manager** in Python with the name AsyncFileReader, which reads the whole content of a file asynchronously. You can find some codes below to read a file in synchronous fashion.

```
def read_file(filename):
    file = open(filename, mode="r")
    content = file.read()
    file.close()
    return content

def sync_main():
    content = read_file("test.txt")
    print(content)

if __name__ == "__main__":
    sync_main()
```

aiofiles module provides a very useful awaitable (aiofiles.open), which can be awaited with the same arguments as in the example above.

```
import asyncio
import aiofiles

class AsyncFileReader:
    def __init__(self, filename):
        self.filename = filename

    async def __aenter__(self):
        self.file = await \
            aiofiles.open(self.filename, mode="r")
        return self.file

    async def __aexit__(self, exc_type, exc, tb):
        await self.file.close()
```

```
async def async_main():
    async with AsyncFileReader("test.txt") as file:
        content = await file.read()
        print(content)

if __name__ == "__main__":
    sync_main()
    asyncio.run(async_main())
```

QUESTIONS VS PÇB MATRIX

	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Q1	✓	✓																										
Q2	✓	✓					✓	✓																				
Q3	✓	✓					✓	✓																				
Q4	✓	✓	✓	✓			✓	✓																				

Q3 (25 Points) Write a Python class with the name FileWatcher, which will be a **daemon thread**. This thread will watch a local file, check for changes for every n seconds, and print "File has changed" to the screen if the file has changed after the last check. You can use the functions given below.

```
import time
import os
import threading

def sleep_for_a_while(n):
    time.sleep(n)

def last_modification_time(path):
    return os.path.getmtime(path)


class FileWatcher(threading.Thread):
    def __init__(self, path, interval=1):
        super().__init__()
        self.path = path
        self.interval = interval
        self.last_mod_time = \
            last_modification_time(path)
        self.daemon = True

    def run(self):
        while True:
            if last_modification_time(self.path) != \
                   self.last_mod_time:
                print("File has changed")
                self.last_mod_time = \
                   last_modification_time(self.path)
            sleep_for_a_while(self.interval)


if __name__ == "__main__":
    watcher = FileWatcher(path="test.txt", n=1)
    watcher.start()
    while True:
        sleep_for_a_while(1)
        print("Main thread is running")
```

Q4 (25 Points) The value of Euler's number (e) can be estimated as given below:

Random Numbers of n sets	s
0.67 0.60	2
0.53 0.55	2
0.19 0.59 0.60	3
...	
...	
...	
0.93 0.34	2
0.53 0.04 0.34 0.80	4
0.79 0.17 0.65	3
Number of random numbers that are needed before the sum of the random numbers exceeds 1	136

Estimated Value of Euler's Number
= 136/50 = 2.720000

Write a Python program to employ the method given above, which will satisfy the following features:

- Code must continue to run until the accuracy reaches 1.E-06 (exact value is math.e).
- Code should create 8 processes.
- The communications must be through simplex pipe(s).
- Keep it simple: Don't use classes, define and use only functions.

```
import random, multiprocessing, os, math

def estimate_e(n, pipe):
    count = 0
    for i in range(n):
        sum = 0
        while sum <= 1:
            sum += random.random()
            count += 1
    pipe.send(count)

def accuracy(estimated_value):
    return abs(estimated_value - math.e)

if __name__ == "__main__":
    BATCH_SIZE = 100000
    NUM_BATCHES = os.cpu_count()
    total = 0
    number_of_iterations = 0
    while True:
        pipes = []
        processes = []
        for i in range(NUM_BATCHES):
            recv_end, send_end = multiprocessing.Pipe(False)
            pipes.append(recv_end)
            process = multiprocessing.Process(
                target=estimate_e, args=(BATCH_SIZE, send_end)
            )
            processes.append(process)
            process.start()
        for pipe in pipes:
            total += pipe.recv()
        for process in processes:
            process.join()
        number_of_iterations += NUM_BATCHES * BATCH_SIZE
        estimated_value = total / number_of_iterations
        print(f"Estimated value of e is {estimated_value} with {accuracy(estimated_value)}")
        if accuracy(estimated_value) < 1.0e-6:
            break
```