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					

		Learning Objectives					
		L1	L2	L3	L4	L5	L6
Questions	Q1				✓		
	Q2		✓	✓			
	Q3	✓				✓	✓
	Q4				✓		

Questions

Q1 (25 Points) Match the terms and the definitions in the table given below.
A) Race Condition **B)** Semaphore **C)** Queue **D)** Deadlock **E)** Pipe

Term	Definition
	It creates a pair of connection objects that can be used to send messages between two processes. It returns a pair of connection objects for two processes.
	It is a synchronization object that controls access by multiple processes/threads to a common resource in a parallel programming environment.
	It is a data structure which is thread-safe FIFO (first-in, first-out). It can be used to pass messages between processes in a parallel program.
	It occurs when two or more threads can access shared data and they try to change it at the same time. As a result, the values of variables may be unpredictable.
	It is a situation where two or more threads are blocked forever, waiting for resources that will never be available.

Q2 (25 Points) Import the required modules and write function_1 and function_2 to pass all the tests given below.

```python
def dummy_function_1(dummy_list: list[int]) -> None:
    if not threading.current_thread() == threading.main_thread():
        dummy_list.append(1)

def dummy_function_2(dummy_list: list[int]) -> None:
    if not threading.current_thread() == threading.main_thread():
        dummy_list.append(2)

def dummy_function_3(dummy_list: list[int]) -> None:
    if not threading.current_thread() == threading.main_thread():
        dummy_list.append(3)
```

```python
def test_imported_modules():
    list_of_modules = ["multiprocessing", "asyncio", "threading"]
    for module in list_of_modules:
        assert module in dir(sys.modules[__name__]), f"{module} is not imported"

def test_the_type_of_functions():
    assert callable(function_1), "function_1 is not a function"
    assert asyncio.iscoroutinefunction(function_2), "function_2 is not a coroutine"

def test_function_1():
    assert isinstance(function_1(), int), "function_1 does not return an int"
    assert function_1() == multiprocessing.cpu_count(), "function_1 does not return the correct value"

def test_function_2():
    dummy_list = []
    functions = [dummy_function_1, dummy_function_2, dummy_function_3]
    asyncio.run(function_2(functions, dummy_list))
    assert len(dummy_list) == 3, "function_2 does not run the functions in parallel"
    assert dummy_list == [1, 2, 3], "function_2 does not run the functions in the list"
```

Q3 (25 Points) Complete the **main** function to satisfy these requirements:

- Create a list of **CheckPrime** objects, each with a different number (**n**) to check for primality and start them as separate processes.
- It is not allowed that the number of processes exceeds the number of cores in the host PC.
- Balance the load between the cores equally, so it is desired that one core waits another.

```
class CheckPrime(multiprocessing.Process):
    def __init__(self, q: multiprocessing.Queue, n: int):
        super().__init__()
        self.q = q
        self.n = n

    @staticmethod
    def is_prime(n: int) -> bool:
        return n > 1 and all(n % i for i in range(2, n))

    def run(self):
        print(f"Checking {self.n}")
        if self.is_prime(self.n):
            self.q.put(self.n)

def main():
    processes = []
    started_processes = []
    prime_queue = multiprocessing.Queue()
    n = 50
```

```
    primes = []
    while not prime_queue.empty():
        primes.append(prime_queue.get())
    primes.sort()
    print(primes)
```

Q4 (25 Points) The simple code given below, which generates two processes, gives an unpredictable output. Even if the usage of 'lock=True' argument does not solve the problem. Rewrite the functions plus_one_by_one and minus_one_by_one to solve the problem.

```
from multiprocessing import Process, Value

def plus_one_by_one(n: Value, times: int):
    for i in range(times):
        n.value += 1

def minus_one_by_one(n: Value, times: int):
    for i in range(times):
        n.value -= 1

if __name__ == '__main__':
    number = Value('i', 0, lock=True)
    p1 = Process(
        target=plus_one_by_one, args=(number, 1000000)
    )
    p2 = Process(
        target=minus_one_by_one, args=(number, 1000000)
    )
    p1.start()
    p2.start()
    p1.join()
    p2.join()
    print(number.value)
```

You have 90 minutes, gl hf.
Assoc. Prof. Dr. Bora Canbula