TUNKU ABDUL RAHMAN UNIVERSITY OF MANAGEMENT AND TECHNOLOGY

FACULTY OF COMPUTING AND INFORMATION TECHNOLOGY

ACADEMIC YEAR 2023/2024

MAY/JUNE EXAMINATION

BMCS3003 DISTRIBUTED SYSTEMS AND PARALLEL COMPUTING

SATURDAY, 8 JUNE 2024 TIME: 9.00 AM – 11.00 AM (2 HOURS)

BACHELOR OF COMPUTER SCIENCE (HONOURS) IN DATA SCIENCE
BACHELOR OF COMPUTER SCIENCE (HONOURS) IN INTERACTIVE SOFTWARE TECHNOLOGY

BACHELOR OF SOFTWARE ENGINEERING (HONOURS)

Instructions to Candidates:

Answer **ALL** questions. All questions carry equal marks.

Question 1

a) Explain the roles of **'servers'** and **'clients'** in a client-server network. (2 marks)

b) Explain the term **peer-to-peer network**. (3 marks)

c) Give **ONE (1)** advantage and **ONE (1)** disadvantage of a client-server network over peer-to-peer networks. (4 marks)

d) Explain the way files are stored and shared in a peer-to-peer network. Also explain why peer-to-peer scales better and more resilient to failure than client-server strategy. (8 marks)

e) List **FOUR (4)** challenges of distributed systems and discuss each of them. (8 marks)

[Total: 25 marks]

Question 2

a) Define **Mutex** and explain its purpose and how it works in a multi-threaded environment? (3 marks)

b) Compare and contrast between a **Binary Semaphore** and a **Mutex**? (4 marks)

c) Assuming `sem` is a binary semaphore object with an initial count of 1, provide C++ code to demonstrate the following programming mistakes and explain what detrimental effect(s) might possibly happen with multiple threads executing them concurrently.

(i) Perform down operation on the `sem` object twice before entering the critical section. Then perform an up operation only once. (2 marks)

(ii) Perform an up operation, enter the critical section, and finally perform a down operation. (2 marks)

(iii) Perform a down operation, enter the critical section, but no up operation is done. (2 marks)

(iv) Perform up operation, enter the critical section, but no down operation is done. (2 marks)

d) A fixed-buffer-size **pipe** can be implemented using producer-consumer algorithm. Write a C++ code for a pipe with a buffer size of 10. Assume the following semaphore objects are given:

```
Semaphore full_count (0);
Semaphore empty_count(10);
Semaphore mut(1);
```

To perform a down or up operation on a semaphore object, use the `down()` and `up()` methods. To add into and get message from the buffer, do like the following:

```
buffer_add(msg);       // msg is the message object to add
msg = buffer_get();    // msg will hold the message retrieved
```

Implement the following two pipe functions using producer-consumer algorithm:

```
void pipe_send_msg(Msg &msg);
Msg &pipe_receive_msg();
```

(10 marks)

[Total: 25 marks]

Question 3

a) Explain the purpose and working principle of the **Banker's Algorithm** in the context of resource allocation and deadlock avoidance. (2 marks)

b) Consider a system with **four processes (P0, P1, P2, and P3)** and **three resource types (A, B, and C)**. The maximum demand and current allocation for each process are as follows:

- **Maximum Demand (Max):**
  - P0: A=4, B=1, C=1
  - P1: A=2, B=6, C=2
  - P2: A=5, B=3, C=1
  - P3: A=1, B=4, C=3

- **Current Allocation (Allocation):**
  - P0: A=1, B=0, C=0
  - P1: A=1, B=3, C=2
  - P2: A=2, B=1, C=1
  - P3: A=1, B=4, C=1

Initially, the total resources are A=8, B=10, and C=7.

(i) Calculate the **Available** resources. (2 marks)

(ii) Calculate the **Need** matrix. (4 marks)

(iii) Determine if there is a safe sequence of resource allocation for the processes by deriving it. You must show your working. (10 marks)

c) Answer the following data dependency questions.

(i) Explain what Anti-Dependency (write-after-read) is all about and its implications. Give an example. (3 marks)

(ii) Explain what Output Dependency (write-after-write) is and its implications. Provide an example. (3 marks)

(iii) Suggest a way how Question 3 c) (i) and Question 3 c) (ii) be solved to allow parallel executions. (1 mark)

[Total: 25 marks]

Question 4

Write a CUDA C++ kernel that takes an array **A** of size **N** and computes the following for each element in parallel:

(1) First, compute the square of the element itself.
(2) Then add it to the squares of its left and right neighbors (if they exist).

Your kernel should store the results back in array **A**. The kernel proto-type is given below:

```
__global__ void computeSquaresKernel(double* A, int N);
```

a) Since the array **A** is used as input and also output and its data is changed in parallel by threads, the kernel needs to create a temporary shared variable, say **sharedArray**, for the thread block to temporarily store the computed values of (1) and (2). Explain why this is important and is needed. Provide a scenario to help your explanation if needed. (3 marks)

b) Show how the **sharedArray** should be declared. (2 marks)

c) Write the CUDA kernel for `computeSquaresKernel(.)`. Remember to check for neighbour existence before squaring its value and synchronise the threads whenever necessary. (9 marks)

d) Implement the following function

```
void computeSquaresAndNeighbors(double* hostArray, int arraySize);
```

to dynamically allocate the device memory, copy the data to and fro, invoke the kernel, and finally do the necessary clean up using the following CUDA functions:

```
cudaError_t cudaMalloc (void** devPtr, size_t size);
cudaError_t cudaFree(void* devPtr);
cudaError_t cudaMemcpy(void* dst, const void* src,       \
                       size_t count, cudaMemcpyKind kind);
```

where `cudaMemcpyKind` type value can be either

`cudaMemcpyHostToDevice` or `cudaMemcpyDeviceToHost`

Set the number of threads in a block to 256 and dynamically compute the grid size. Remember to pass the appropriate size for the dynamic memory allocation of the temporary shared variable through the kernel execution configuration. (11 marks)

[Total: 25 marks]