UNIVERSITY OF MORATUWA

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

B.Sc. Engineering
2012 Intake Semester 7 Examination

CS4532 CONCURRENT PROGRAMMING

Time allowed: 2 Hours October 2016

ADDITIONAL MATERIAL: *None*

INSTRUCTIONS TO CANDIDATES:

1. This paper consists of **five (5)** questions in **six (6)** pages.
2. Answer any **four (4)** questions.
3. Start answering each of the main questions on a new page.
4. The maximum attainable mark for each question is given in brackets.
5. This examination accounts for 50% of the module assessment.
6. This is a closed book examination.

   NB: It is an offence to be in possession of unauthorized material during the examination.
7. Only calculators approved by the Faculty of Engineering are permitted.
8. Assume reasonable values for any data not given in or with the examination paper. Clearly state such assumptions made on the script.
9. In case of any doubt as to the interpretation of the wording of a question, make suitable assumptions and clearly state them on the script.
10. This paper should be answered only in English.

Question 1 (25 marks)

(i) State whether the following statements are TRUE or FALSE. Give one sentence justification for your answer.

Write the corresponding sub-question number and the answer in your answer book.

a) Message passing can be used to solve mutual exclusion problems. [2]

b) A program that utilizes busy waiting to provide mutual exclusion typically takes more time to execute than a program that utilizes a mutex. [2]

c) It is a good practice to read the counter value of a semaphore and then decide what action to take. [2]

d) Once a deadlock occurs, by rolling back to a previous safe state, we can guarantee that the deadlock will not occur again. [2]

e) Under the ideal conditions, a k-stage pipeline can obtain a speedup of n while processing an input vector of n elements. [2]

f) If $N >> P$, it is possible to achieve asymptotically linear speedup while adding N numbers using P processors. [2]

g) A call to MPI *Barrier*() blocks the calling process until all the processes in the communicator have reached this routine. Therefore, it may introduce delays in a program. [2]

h) While using the token-based algorithm to solve a distributed mutual exclusion problem, it is easier to create a new token, if the previous token is known to be lost. [2]

(ii) One of your customers can use up to 4 processors for 40% of his/her applications. Customer wants to improve the overall performance of applications. How much overall speed up will the customer gain if

a) customer increases the number of processors from 1 to 4? [3]

b) customer use 2 processors, but add features that will allow the applications to use them for 80% of execution? [3]

c) From ii(a) and ii(b) above, which technique will you recommend to the customer? Briefly Discuss. [3]

Question 2 (25 marks)

(i) Consider the following program with 2 threads.

```
int x;
```

Thread 1

```
...
X = 3
x += 1
...
```

Thread 2

```
...
X = 0
X *= 2
...
```

a) Provide 4 possible values for *x* once the execution of 2 threads is over. [4]

b) Give a semaphore-based solution to make sure that *x* will be set only to 7.

You are not allowed to change value of *x*, only order of execution can be controlled. [4]

(ii) In multithreaded programs, there is often a division of labor between threads. In one common pattern, some threads are producers and some are consumers. Producers create items of some kind and add them to a data structure; consumers remove the items and process them.

Provide the pseudocode for the producer and the consumer of the finite (bounded) buffer problem. [10]

(iii) The following is an implementation for *get_forks*(*i*) and *put_forks*(*i*) methods in Dining Philosophers Problem. Note that functions *right*(*i*) and *left*(*i*) return the right and left forks respectively while fork is an array of semaphores where the initial value of each semaphore is 1. The size of the fork array is 5.

```
def get_forks(i):
        fork[right(i)].wait()
        fork[left(i)].wait()
```

```
def put_forks(i):
        fork[right(i)].signal()
        fork[left(i)].signal()
```

a) What is the problem with the above solution? Discuss. [3]

b) Provide a solution to the problem you have identified above by limiting the number of philosophers at the table. Discuss how it will solve the problem you identified in iii(a). [4]

Question 3 (25 marks)

(i) In a *shared bathroom problem*, there are two classes of threads, called male and female. There is a single bathroom resource that must be used in the following way:

1. Mutual exclusion – persons of opposite sex may not occupy the bathroom simultaneously.

2. Starvation-freedom – everyone who needs to use the bathroom eventually enters.

The protocol is implemented via the following four procedures: *enterMale*() delays the caller until it is ok for a male to enter the bathroom, *leaveMale*() is called when a male leaves the bathroom, while *enterFemale*() and *leaveFemale*() do the same for females. For example,

```
enterMale();
teeth.brush(toothpaste);
leaveMale();
```

Propose a suitable solution to the shared bathroom problem.

You may use a suitable combination of locks, mutexes, semaphores, conditional variables, etc. [17]

(ii) Static or dynamic load balancing is essential in most systems to increase the utilization and quality of service. What type of load balancing would you recommend for the following problems? Justify your recommendation.

a) Processing Twitter messages to find hashtags (hashtag is used to mark keywords or topics). [4]

b) While cracking 1,000 password-protected word documents found from a suspected terrorist's laptop. When attempted password is invalid, the respective word processing software introduces a random delay (between 1 to 120 seconds) before allowing the next password to be tried.

Assume brute-force approach is used to crack passwords. [4]

Question 4 (25 marks)

(i) Suppose you are given one million random integers. You need to find the minimum number among all the given integers.

a) Outline a CUDA kernel to find the minim number given an array of numbers. [8]

b) Show how you would launch the kernel and any other code required to find the minimum among all the one million random integers. [4]

(ii) Following is a variant of the *monkeys on rocks problem*.

There are monkeys and gorillas living on two very high rocks. The northern monkeys live on the northern rock that provides water but no food. Conversely, the southern gorillas live on the southern rock that provides food but no water. However, both the monkeys and gorillas have to eat and to drink! There is a small rope between the two rocks. The rope can carry up to five monkeys and one gorilla at a time. Concurrent crossing in both directions is not possible.

a) One of your classmates said "*This problem can be solved with the solution to readers and writers problem*". Do you agree or disagree with this statement? Discuss. [4]

b) Provide a pseudocode to solve this problem. [9]

Question 5 (25 marks)

(i) Suppose we want to generate the first 10,000 values of the geometric sequence. Given a number n, the rule is $x_n = 2^n$. In General we can write a geometric sequence as:

$$\{a,\ ar,\ ar^2,\ ar^3,\ \ldots\ \}$$

For example,

$$2,\ 4,\ 8,\ 16,\ 32,\ 64,\ 128,\ 256,\ \ldots$$

a) Outline an MPI program (using pseudo code) that can be used to generate the first 10,000 values of the geometric sequence. Once the list is generated, it should be saved to a file at process 0.

Use relevant MPI functions that are given in the Appendix. Note that it is impractical to create 10,000 concurrent processes/threads. [16]

b) Comment on the performance of your program and its ability to full utilize all the computational nodes available in the MPI cluster. [4]

(ii) Using a suitable example, explain how linear ordering of resources helps to prevent deadlocks. [5]

Appendix – MPI Functions

```
. . .
#include <mpi.h>
. . .
int main(int argc, char* argv[]) {
   . . .
   /* No MPI calls before this */
   MPI_Init(&argc, &argv);
   . . .
   MPI_Finalize();
   /* No MPI calls after this */
   . . .
   return 0;
}
```

```
int MPI_Init(int *argc, char **argv)
int MPI_Comm_size(MPI_Comm comm, int *size)
int MPI_Comm_rank(MPI_Comm comm, int *rank)
int MPI_Finalize()
int MPI_Send (void *buf,int count, MPI_Datatype datatype, int dest, int
            tag, MPI_Comm comm)
int MPI_Recv (void *buf,int count, MPI_Datatype datatype, int source, int
            tag, MPI_Comm comm, MPI_Status *status)
int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype
            datatype, MPI_Op op, int root, MPI_Comm comm)
int MPI_Allgather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void
            *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)
int MPI_Allreduce (void *sendbuf, void *recvbuf, int count, MPI_Datatype
            datatype, MPI_Op op, MPI_Comm comm)
int MPI_Bcast( void *buffer, int count, MPI_Datatype datatype, int root,
            MPI_Comm comm)
int MPI_Gather(void *sendbuf, int sendcnt, MPI_Datatype sendtype, void
            *recvbuf, int recvcnt, MPI_Datatype recvtype, int root,
            MPI_Comm comm)
int MPI_Scatter(void *sendbuf, int sendcnt, MPI_Datatype sendtype, void
            *recvbuf, int recvcnt, MPI_Datatype recvtype, int root,
            MPI_Comm comm)
```

Operation Value	Meaning
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical and
MPI_BAND	Bitwise and
MPI_LOR	Logical or
MPI_BOR	Bitwise or
MPI_LXOR	Logical exclusive or
MPI_BXOR	Bitwise exclusive or
MPI_MAXLOC	Maximum and location of maximum
MPI_MINLOC	Minimum and location of minimum

---------- END OF THE PAPER ----------