DE MONTFORT UNIVERSITY LEICESTER

Faculty of Computing, Engineering & Media (CEM)

Coursework Brief 2021/2022

Module name:	Concurrent and Parallel Algorithms		
Module code:	CTEC2910		
Title of the Assignment:	Coursework		
This coursework item is: (delete as appropriate)	Summative		
This summative coursework will be marked anonymously: (delete as appropriate)	Yes		
The learning outcomes that are assessed by this coursework are: 1. Analyse specific programming language and library support for the development of concurrent programs 2. Apply standard concurrent algorithm design patterns			
This coursework is: (delete as appropriate)	Individual		
If other or mixed ... explain here:			
This coursework constitutes 60% **of the overall module mark.**			
Date Set:	Week 18, Tuesday 1 February 2022		
Date & Time Due:	**Week 25, Monday 21 March 2022, 14:00**		
Your marked coursework and feedback will be available to you on: If for any reason this is not forthcoming by the due date your module leader will let you know why and when it can be expected. The Associate Professor Student Experience (cemstudentexperience@dmu.ac.uk) should be informed of any issues relating to the return of marked coursework and feedback. Note that you should normally receive feedback on your coursework by **no later than 20 University working days after the formal hand-in date,** provided that you have met the submission deadline.	21/04/2022		

Please turn over ….

When completed you are required to submit your coursework via:

Turnitin through the Blackboard shell

If you need any support or advice on completing this coursework please visit the Student Matters tab on the Faculty of Technology Blackboard page.

Late submission of coursework policy: Late submissions will be processed in accordance with current University regulations which state:
*"the time period during which a student may submit a piece of work late without authorisation and have the work capped at 40% [50% at PG level] if passed is **14 calendar days**. Work submitted unauthorised more than 14 calendar days after the original submission date will receive a mark of 0%. These regulations apply to a student's first attempt at coursework. Work submitted late without authorisation which constitutes reassessment of a previously failed piece of coursework will always receive a mark of 0%."*

Academic Offences and Bad Academic Practices:

These include plagiarism, cheating, collusion, copying work and reuse of your own work, poor referencing or the passing off of somebody else's ideas as your own. If you are in any doubt about what constitutes an academic offence or bad academic practice you must check with your tutor. Further information and details of how DSU can support you, if needed, is available at:

http://www.dmu.ac.uk/dmu-students/the-student-gateway/academic-support-office/academic-offences.aspx and

http://www.dmu.ac.uk/dmu-students/the-student-gateway/academic-support-office/bad-academic-practice.aspx

Tasks to be undertaken: Question 1, 2

Deliverables to be submitted for assessment:

Single Word or PDF document. DO NOT submit any code files or any other supplementary material.

How the work will be marked: Anonymously according to the marking grid.

Module leader/tutor name:	Vasileios Germanos
Contact details:	Email: vasileios.germanos@dmu.ac.uk

Assignment

About this assessment

This assignment has TWO (2) parts with a total 100 marks, which will constitute 60% of your final mark for the module. Answer both parts.

Objectives

The objective of this assessment is for you to demonstrate your understanding of issues related to the design and use of concurrent algorithms.

Submission

Submit your work as a single Word document or pdf document through the portal available on Blackboard. Write your answers to the questions in the Word or pdf document.

DO NOT submit any code, either as separate files or within your document. Any code submitted will not be marked.

Marking

Please refer to the marking grid provided.

Anonymity

The assignment will be marked anonymously. You will compromise this anonymity if you put your name on the work. Please include your p-number on your assignment.

Question 1 (40 marks)

A cake shop provides a stream of cakes for customers. Cakes are made by a chef in the kitchen and placed on a conveyor belt. The conveyor belt carries the cakes to where the customers are waiting to buy them.

This scenario has been simulated using two processes: the *chef* and the *customer*, and a shared conveyor belt implemented as an array called *conveyor* with infinite size, where each space in the array can hold one cake. In this scenario, there are multiple chefs.

The pseudo-code for the chef is as follows.

```
1. while(true){
2.    sushi = makeSushi(); // Create a sushi
3.    wait(empty);
4.    conveyor[in] = sushi; // Put the sushi on the conveyor belt
5.    in = (in + 1) mod n;
6. }
```

This code should be familiar to you as it is similar to the example of the Producer-Consumer problem. Only a few sentences are required for each of the questions below.

(a) (20 marks) The pseudo-code for the **chef** has an error(s). Propose the solution by giving the correct code.

The scenario states that the chef makes cakes and not sushi. Therefore I am changing the variable sushi=makeSushi() to cake=makecake()

```
1.    while(true){
      2.    cake = makeCake(); // Create a cake not a sushi
      3.    wait(prodMutex);// locks the critical section so only 1
      chef can input the data into the specified conveyor slot
      4.    conveyor[in] = cake; // Put the cake on the conveyor belt
      5.    in = (in + 1); // no need for mod n. Conveyor size is
      infinite
      6.    signal(prodMutex); unlocks the critical section so many
      chefs can access the slot again. Even though it has a cake
      7.    signal(Ready); // signals the customer that their food is
      ready
      8.    }
```

(b) (20 marks) Now assume that there is a **single chef** and a **single customer**, with a conveyor belt of **finite** size. Give the pseudo-code for this scenario, both for the **chef** and **customer**.

```
Chef:
1.   int in =0
2.   While(true){
     3.   wait(empty);
     4.   cake=makeCake();
     5.   conveyor[in]= cake;
     6.   in=(in+1) MOD n; // n is buffer length
     7.   signal(full)
     8.   }
Consumer
1.   int out =0;
2.   while(true){
     3.   wait(full);
     4.   cake = conveyor[out];
     5.   conveyor[out]=null;
     6.   out=(out+1) MOD n // n is the buffer length
     7.   signal(empty);
     8.   eat(cake);
```

Question 2 (60 marks)

There is a self-service passport control that has three scanning devices, so only three passengers can scan their passport at the same time. The code for each passenger is given below. It uses a semaphore *passport*, to represent whether there is space left in the passport control for more passengers. S1, S2, and S3 are labels identifying the line code.

Passenger code:
S1: wait(passport);
S2: scan();
S3: signal(passport);

(a) (40 marks) There are five passengers, A, B, C, D, and E that want to scan their passport. Each passenger operates according to the above code. Your task is to give an execution trace that contains both of the following:

- The trace should show what happens when there are less than three passengers in the passport control and one or more of them leave.
- The trace should show what happens when there are three passengers in the passport control and another passenger, or several passengers want to scan their passport.

The trace must start with an empty passport control and show all the steps of each of the passengers as they enter/leave.

You must show the value of *passport* after each *wait* or *signal* step. Ensure that you state the initial value of *passport*.

If a wait operation executes, indicate whether the process succeeds or is placed in the queue. If a signal operation executes, indicate whether the value is changed, or a sleeping process is woken up.

Each line of your trace should have the following format:
Statement executed including which Passenger process (e.g., M.S1 means that the passenger named M executed statement S1); the value of the *passport* semaphore (e.g., passport=2); whether the wait succeeded or was placed in the queue/ whether the signal changed the semaphore value or a sleeping process woke up.
For example: M.S1; passport=2; wait succeeded.

A.S1; passport=2; wait succeeded.
A.S2;passport=2;
A.S3;passport=3;
B.S1;passport=2; wait succeeded.
C.S1;passport=1; wait succeeded.
D.S1;passport=0; wait succeeded.
E.S1;passport=0;wait failed, E is put into queue
B.S2; passport=0;
C.S2; passport=0;
D.S2; passport=0;
B.S3; passport=0;Signal woke up E
E.S2; passport=0;
C.S3; passport=1;
D.S3; passport=2;
E.S3; passport=3;

(b) (20 marks) The code has now been modified. Each time a passenger uses a scanning device, an integer *totalScanned* is incremented. The variable is initially 0. The new code is given below:

Passenger code:
S1: wait(passport);
S2: totalScanned = totalScanned + 1;
S3: scan();
S4: signal(passport);

The code given has a problem. The passengers have found that sometimes the *totalScanned* value does not reflect the actual number of passengers who have entered the passport control.

- Explain, in words, how this situation could occur.
- Show a trace that demonstrates the problem occurring. Note that you are not being asked to solve the problem.

This output would occur when a passenger has done step S2 but not S3. Within the time the passenger does S2 to S3 the totalScanned could be off by either 1,2 or 3 places.

In this example there are 3 people, A,B and C

A.S1; passport=2; totalScanned=0;wait succeeded.
B.S1;passport=1; totalScanned=0;wait succeeded.
C.S1;passport=0; totalScanned=0;wait succeeded.
A.S2; passport=0; totalScanned=1;
B.S2;passport=0; totalScanned=2;

C.S2;passport=0; totalScanned=3;
///

At this point, people haven't scanned their passports yet but the total scanned passports are 3 higher than they should be.

///

A.S3; passport=0; totalScanned=3;
B.S3; passport=0; totalScanned=3;
C.S3; passport=0; totalScanned=3;
A.S4; passport=1; totalScanned=3;
B.S4; passport=2; totalScanned=3;
C.S4; passport=3; totalScanned=3;

Mark Range	Questions 1a and 1b	Question 2a	Question 2b
80-100%	An excellent explanation, demonstrating a thorough knowledge of the principles.	A clear, correct trace that correctly demonstrates the scenario, with correct values.	A clear, correct trace that correctly demonstrates how the problem occurs and a correct explanation.
60-79%	An explanation that demonstrates a good understanding of the principles, with mostly correct points made.	A trace that is mostly correct, which demonstrates the scenario but may have some minor errors in the values.	A trace that generally demonstrates how the problem occurs but may have some errors. A correct or almost correct explanation.
50-59%	An explanation that demonstrates a generally good understanding of the principles.	A trace that has some errors but generally demonstrates the scenario and has generally correct values.	A correct or almost correct explanation. The trace given may not be correct, but shows some understanding of the problem.
40-49%	An explanation that demonstrates a basic understanding of the principles.	A trace that shows some basic understanding of the concepts but has some incorrect values.	An explanation that shows some understanding of the problem. The trace may be incorrect.
20-39%	A weak explanation of the concepts, demonstrating very little understanding of the principles.	A trace that shows only a very weak understanding of the concepts, with incorrect values.	A weak explanation and an incorrect trace, demonstrating a very weak understanding of the problem.
0-19%	An incorrect or extremely weak explanation of the concepts, demonstrating no understanding of the principles.	No trace or a trace that is not at all correct and does not demonstrate any understanding of the concepts.	The explanation and trace given are both incorrect and demonstrate no understanding of the problem.