6G5Z1005 Computer Networks and Operating Systems 2015/16

Term 1
Operating Systems Component Handbook

Lecturer: Emma Norling

Additional staff: Soufiene Djahel

TABLE OF CONTENTS

1 Staff Contact Details

Lecturer: Emma Norling
Office: John Dalton E128
Email: e.norling@mmu.ac.uk
Telephone: 0161 247 3884

Additional Staff: Soufiene Djahel
Office: John Dalton E114
Email: s.djahel@mmu.ac.uk

Both staff members advertise scheduled office hours via their Moodle profile (click on the appropriate name at the top of the unit Moodle area).

Unit Leader: Mohammad Hammoudeh
Office: John Dalton E131
Email: m.hammoudeh@mmu.ac.uk
Telephone: 0161 247 2845

Dr Hammoudeh will be teaching on the networks component of this unit, in term 2, but also has overall responsibility for the unit. If you have an issue that cannot be resolved by this term's teaching staff, you may wish to raise it with him.

2 Unit Learning Outcomes

The unit learning outcomes are:

1. Demonstrate knowledge and correct application of the principles of concurrency.
2. Explain the principles of operation, and the evolution of operating system structural organisation with reference to computer and processor architecture.
3. Explain the architectural principles of computer networks, and contrast different approaches to organising networks.
4. Demonstrate the ability to properly configure network components including switches, routers and services (such as FTP, DNS, DHCP).
5. Design, implement, and maintain local area networks.

The last three of these will be covered in the networks component of this unit, in term two. In other words, by the end of this term, you should be able to demonstrate knowledge and correct application of the principles of concurrency and explain the principles of operation, and the evolution of operating system structural organisation with reference to computer and processor architecture.

3 Delivery

Each week, with the exception of tutor week (week 7), there is one one-hour timetabled "lecture" and one two-hour timetabled lab class for every student.

3.1 Lectures

The material for this unit will be delivered in a "flipped" form. Each week, you will be expected to watch a small set of podcasts, which are accessible through Moodle. You should also read the relevant section(s) from the textbook. The lecture will then be used for discussion and clarification of the topics covered in the podcasts, with some exercises that will both test you for your understanding and prepare you for the assessment tasks.

This is a new mode of delivery this year, developed in response to student feedback. It gives you the flexibility to watch the podcasts when and where you like, and as often as you want. The discussion session (timetabled lecture) is an essential part of the programme. Even if you feel you understood the topics covered in the week's podcast perfectly, other students may raise questions that you hadn't even considered, and the exercises will help to prepare you for your assessment.

The podcasts that you should watch in any given week are given in that week's Moodle area. The topics covered in the podcasts will be discussed in the following week's lecture slot. You *must* make sure that you watch the podcasts and read the appropriate textbook sections in order to benefit fully from the discussion.

Week	**Discussion Topic** **(You need to cover the work *before* the week specified.)**
1	Overview – standard lecture, no preparation required.
2	File management
3	Processes and threads
4	Concurrency
5	Deadlocks and starvation
6	Process scheduling
7	Tutor week – no scheduled classes
8	Memory management
9	Virtual memory
10	I/O management
11	Security
12	Review & future studies

3.2 Lab classes

Lab classes will be used for practical exercises to reinforce your understanding of the material and also to develop skills that will be used in your further studies and beyond. (In particular, you will be learning to use a Linux workspace, if you have not already done so.) In some weeks the lab classes will be used for in-class tests that form part of your assessment portfolio, and two of the lab classes are dedicated to work on your programming task (also part of your assessment portfolio).

4 Expectations

4.1 What you should expect of staff

- Staff will treat you courteously and respectfully at all times.
- Staff will support you to achieve the unit learning outcomes.
    - Discussion sessions give you the opportunity to clarify points.

    - Staff will be available during lab sessions for further clarification (except when in-class tests are scheduled).
    - Staff have nominated contact hours, when you can visit them in their offices for further assistance.
    - Staff will respond to email during normal working hours. If you do not receive a response within two working days, you should assume that your email has gone astray.
- Staff will provide materials in a timely manner.
    - All relevant podcasts will be available at least one week before the scheduled discussion of the topic, and remain available for the duration of the unit.
    - Details of the portfolio assessment are available from the start of term (in this document).
    - Details of laboratory exercises will be available at least one week in advance of the session.
- Formative feedback will be provided for your assessment elements.
    - In the case of in-class tests, you will receive feedback from the first sitting of your test before attempting the second.
    - For your programming task, you should seek feedback from the staff member in your laboratory class.
    - For the reflective writing, the mid-term portfolio element will provide you with formative feedback. Your final reflective report will be an extension of this midterm report, and should incorporate changes based upon the feedback received.

4.2 What Staff Expect of You

- You will treat staff courteously and respectfully at all times.
- If you have a problem with any aspect of the unit, you will raise it first with the teaching staff.
- You will attend your scheduled classes, having adequately prepared, each week.
- You will demonstrate your competency through the elements of the assessment portfolio.

5 Assessment

Assessed coursework 1 (ACW1) covers this term's material. It is a competency portfolio of work, which includes in-class tests, a programming task, and reflective writing. ACW1 is marked out of 100 and is worth 50% of your overall mark for this unit. As a reminder, the learning outcomes assessed by this portfolio are:

1. Demonstrate knowledge and correct application of the principles of concurrency.
2. Explain the principles of operation, and the evolution of operating system structural organisation with reference to computer and processor architecture.

The breakdown of work in this portfolio is:

Portfolio element	**Marks available**	**Schedule**
In-class test #1 Covering these topics: Overview, file management, processes and threads.	20	First sitting during your lab class, week 4. Second sitting during your lab class, week 5.
Mid-term reflection	Formative feedback only	To be submitted via Moodle by 10am Monday, week 8.
Concurrent programming task	30	To be submitted via Moodle by 10am Monday, week 9. Lab classes in weeks 6&8 will be devoted to this task.
In-class test #2 Covering these topics: Concurrency (including deadlocks and starvation), processor scheduling, memory, I/O management, security.	30	First sitting during your lab class, week 11. Second sitting during your lab class, week 12.
Reflective report	20	To be submitted via Moodle by 23:59 Friday, week 12.
Total	**100**	

5.1 In-Class Tests

In-class tests will be in the form of Moodle quizzes, with a variety of question types including multiple choice and short answer. The quizzes that are presented during the weekly discussion sessions will help to prepare you for these tests. For each of the in-class tests, you will have *two* opportunities to sit the test. The mark you will receive for each test will be the best of the two tries. For example, if you got 15/20 for you first sitting of test 1, 18/20 for your second sitting of test 1, 21/30 for your first sitting of test 2 and 17/30 for your second sitting of test 2, you would receive 18/20 for test 1 and 21/30 for test 2.

If you are satisfied with the mark you achieved on your first sitting of the test, there is no need to re-sit it. However unless you achieved a perfect score, you are strongly advised to have a second attempt. You will not see identical questions in your second attempt, but they will be drawn at random from the same test bank, covering the same topics.

5.2 Mid-Term Reflection

The midterm reflection is a brief report (300-500 words) reflecting on what you have learnt so far in this unit, and how this is relevant to your degree. Although it is for formative feedback only, it is an essential element of your portfolio, providing you with feedback that should help you produce a better final report.

You will need to make it clear in your report which degree you are undertaking. Your report does not need to include every detail of what you have learnt so far (hopefully you would need significantly more than 500 words for this), but rather highlight what you have felt was particularly important or unexpected in your learning. You should reflect upon how this knowledge is relevant to the further studies in your degree and/or the career that you wish to pursue.

You will receive formative feedback for this element of your portfolio, and indicative grading via the rubric shown on page 11. The same rubric will be used for your final reflective report, which is part of your summative assessment, so you should consider the feedback carefully and use it to guide the writing of your final reflection.

5.3 Concurrent Programming Task

The aim of the programming task is for you to demonstrate your knowledge of concurrent programming in Java. The detailed specifications of this task start on page 9. The marking rubric shown on page 12 will be used to grade the concurrent programming task.

You should read the specification carefully, and make sure that your code conforms to the specifications. You should also develop the habit of documenting your code as you write it. As you are writing Java code for this task, it would be a good idea to use Javadoc. If you are not familiar with Javadoc, a gentle introduction is available at http://www.tutorialspoint.com/java/java_documentation.htm. A more general guide to code style and documentation has been prepared by staff at Princeton University, available at http://introcs.cs.princeton.edu/java/11style/, which all students are encouraged to read and consider.

There are two weeks of lab classes dedicated to working on this task, but you should also dedicate some time outside class to working on this task. During your scheduled classes, you should show your work to the staff member in the lab, who can provide formative assessment of your work.

5.4 Final Reflection

The final reflective report is a short report (approximately 500 words). In this report you are asked to reflect upon what you have learnt about operating systems, and how it is relevant to your degree. You should use your midterm report as a basis for this report, extending it to reflect on the subsequent topics studied, and adapting it to take account of the feedback you received on that earlier report.

You will need to make it clear in your report which degree you are undertaking. Your report does not need to include every detail of what you have learnt (hopefully you would need significantly more than 500 words for this), but rather highlight what you have felt was particularly important or unexpected in your learning. You should reflect upon how this knowledge is relevant to the further studies in your degree and/or the career that you wish to pursue.

As for your midterm report, this report will be graded according to the rubric on page 11.

CONCURRENT PROGRAMMING TASK - DETAILS

Many banking systems allow multiple cardholders access a single account (e.g. a business account). The consequence of this is that is possible for more than one person to attempt to withdraw or deposit money from a single account simultaneously. If the access to the account is implemented naïvely (ignoring the principles of concurrent programming), this can lead to race conditions, and the potential for the following fraud to be successful:

> Multiple cardholders may collude and attempt to carry out a timing attack. In such an attack, multiple cardholders withdraw money from the account simultaneously, with the aim of only one deduction to the account balance being made.

Your task is to write a program (detailed specifications below) that will simulate the operation of an ATM-enabled bank account with multiple linked cards. In other words, there will be several individuals who can access an account, each with his/her own card (). You program should demonstrate the principles of concurrent programming, making it impossible for the above fraud to be successful.

DETAILED SPECIFICATIONS

Your program should implement each access card as a separate thread, and the account as a monitor.

You program should take *two* command line arguments: the first is the number of access cards, and the second should be the starting balance. If the user does not supply these two arguments, **your program should exit and give an error message**. Do not use default values for these variables, or prompt the user for these values once the program has started.

Within each thread, you should use the following code to simulate a series of accesses to the account:

```
for (int i = 0; i < 20; i++) {
      if (Math.random() > 0.5) {
            account.withdraw((int)(Math.random()*10));
      } else {
            account.deposit((int)(Math.random()*10));
      }
      sleep(200);
}
System.out.println("THREAD "+ getId() + " " + localBalance);
```

(This code is the basis for each thread. You are expected to build upon it in you program, but should maintain the same random generation of withdrawals and deposits. If you choose to implement a runnable instead of extending the Thread class, change the line saying `sleep(200);` to `Thread.sleep(200);`.)

Only one card may access the account balance at any time. Furthermore, the account must always have a positive balance; if a thread attempts to withdraw and there are insufficient funds, it must block until sufficient funds become available. Note that this means there is a possibility of deadlock: if the account balance is 0 (or close to 0) and all cards request a withdrawal at this point, they may all enter the blocking state. You do not have to implement deadlock prevention, avoidance, detection or recovery; you should just accept that this may happen. (It can be avoided by starting with a small number of cards, such as 5, and a sufficiently high account balance, such as 1000.)

Program Execution

Your program should initialise the account (based on the balance specified on the command line), create and start the *n* threads (where *n* is the number of access cards, specified on the command line), keep a record of transactions, and when all threads are finished, print (to standard output) "**Complete**," followed by a statement of the transactions, in the format described below.

In addition, each thread, just before it terminates, should print out its net balance of transactions. (This is indicated by the `System.out.println("THREAD "+ getId() + " " + localBalance);` line of code overleaf.) The balance is the sum of all withdrawals, minus the sum of all deposits. Note that it acceptable for this to be negative; holders may deposit more than they withdraw.

Statement Format

The statement should be in the following format:

```
Transaction   Withdrawal   Deposit  Balance
                                    1000
1(1)          40                    960
2(3)                       60       1020
3(2)                       20       1040
4(4)          90                    950
…             …            …        …
…             …            …        …
```

The number appearing in parentheses after the transaction number shows the ID of the thread responsible for the transaction.

Hint: you may wish to use String.format to achieve neatly formatted output.

6 Marking

6.1 Rubric Used to Assess Mid-Term Report and Final Reflective Report

Graduate Outcome	86-100%	70-85%	60-69%	50-59%	40-49%	35-39%	20-34%	0-19%
Apply skills of critical analysis to real world situations within a defined range of contexts	Links between theory and practice are evaluated with original insight.	Links between theory and practice are evaluated critically.	Links between theory and practice are evaluated.	Links between theory and practice are analysed.	Links between theory and practice are constructed.	An inadequate range of links between theory and practice is constructed.	Links between theory and practice are provided but are confused or wrong.	Links between theory and practice are absent, or confused and wrong.
Demonstrate a high degree of professionalism (e.g. initiative, creativity, motivation, professional practice and self-management.	Work is insightfully evaluated with respect to the characteristics of a professional.	Work is meticulously evaluated respect to the characteristics of a professional.	Work is rigorously evaluated with respect to the characteristics of a professional.	Work is thoroughly evaluated with respect to the characteristics of a professional.	Work is evaluated with respect to the characteristics of a professional.	There is evidence of partial identification of strengths and weaknesses of personal performance in relation to the characteristics of a professional.	There is evidence of a limited attempt to identify strengths and weaknesses of personal performance in relation to the characteristics of a professional.	There is little or no evidence of an attempt to identify strengths and weaknesses of personal performance in relation to the characteristics of a professional.
Express ideas effectively and communicate information appropriately and accurately	Work is communicated fluently.	Work is communicated convincingly.	Work is communicated effectively and thoroughly.	Work is communicated clearly and confidently.	Work is clearly communicated.	Communication of work is inadequate or confused.	Communication of work is inadequate and confused.	Communication of work is very difficult to understand, or absent.

6.2 Rubric Used to Assess Concurrent Programming Task

Graduate Outcome	86-100%	70-85%	60-69%	50-59%	40-49%	35-39	20-34%	0-19%
Code and documentation presented at an industry standard.	Code and documentation are written authoritatively, to a first-class industry standard.	Code and documentation are meticulously presented, approaching first-class industry standards.	Code and documentation are precisely presented, to a standard that would be acceptable in industry.	Code and documentation are clear and consistent, with minor gaps or need for improvement.	Code and documentation are of a satisfactory standard, with some room for improvement.	Code and documentation are inadequate, being incomplete, unclear or misleading	Code and/or documentation are incoherent, extremely limited, or unstructured.	Code and/or documentation are formless, lacking, or detrimental to the understanding of the program.
Threads used correctly.	Program demonstrates authoritative use of threads.	Program demonstrates sophisticated use of threads.	Program demonstrates the correct use of threads, including creation, starting and termination.	Threads used carefully, with minor errors in creation, starting and/or termination.	Adequate use of threads, with some gaps or errors.	Inadequate use of threads, with significant errors.	Erroneous use of threads, demonstrating a lack of understanding.	Threads not used.
Java monitors used correctly	Program demonstrates authoritative use of Java monitors.	Program demonstrates sophisticated use of Java monitors.	Program demonstrates the correct use of Java monitors, including prevention of race conditions and use of wait/notify.	Java monitors used confidently, with minor errors.	Satisfactory use of Java monitors, with some gaps or errors.	Java monitors not used correctly.	Erroneous use of Java monitors, or attempt to re-implement monitors.	Monitors not used.
Program performance and structure.	Program performs precisely as specified, showing correct implementation and meticulous consideration of the input and output requirements.	Program performs as specified, but with minor variation from the specified I/O requirements.	Program performs largely as specified, but with variation from the specified I/O requirements, and/or minor errors in calculations.	Some errors in program performance, such as incorrect input handling, incomplete or erroneous output, partially missing calculations.	Overall structure of program roughly correct, but performance significantly wrong.	Significant errors in program structure.	Extremely limited program performance, and little or no program structure.	Program does not compile.