UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science

COS 122 Operating Systems

Practical 3

Due: 2017-09-18 @ 14:00 PM

September 4, 2017

PLAGIARISM POLICY

UNIVERSITY OF PRETORIA

The Department of Computer Science considers plagiarism as a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent) and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to `http://www.ais.up.ac.za/plagiarism/index.htm` (from the main page of the University of Pretoria site, follow the *Library* quick link, and then click the *Plagiarism* link). If you have any form of question regarding this, please ask one of the lecturers, to avoid any misunderstanding. Also note that the OOP principle of code re-use does not mean that you should copy and adapt code to suit your solution.

Instructions

In this module it is advised that you have a Virtual Machine(VM) to do all your practicals due to the nature of this course it is easy to mistakenly destroy your Operating System. For all practicals you must make use of a Linux VM.

Follow the Upload and Demo instructions carefully at the end of the practical.

- All written answers must be in either txt or pdf format no other formats will be marked.
- Only upload your source code, makefile, and supporting documents. Do not upload large, binary, or OS files. Keep your uploaded archive small.
- Upload your work in a tar or gz file before 14:00 on the 18th of September 2017.
- Bring your VM on a flash disk to your booked practical demonstration session in the week starting on the 19th of September 2017, so you may be marked.
- On non-demo days, there will still be teaching assistants available in the labs to help you.
- Make use of a Makefile for your program.
- You makefile **must** have these commands:
  - # make
  - # make run

Upload Instructions

As it would be impractical to upload your Virtual Machine to the CS website for the practicals, you are required to submit your code and answers to the written questions:

- Upload your work in one tar or gz file to the Practical 3 assignment slot on the COS 122 course website before 14:00 on Monday 18th September 2017. No late submissions will be accepted.
- You have to demo what you uploaded.
- All written answers must be in either txt or pdf format no other formats will be marked.
- Every file you submit should contain your name, surname, and student number at the top of the file in comments.
- Only upload your source code, makefile and supporting documents (written answers). Do not upload large binary, or OS files. Keep your uploaded archive small.

- Part of this practical will be marked by Fitchfork. Make sure you follow file naming conventions. Fitchfork slots will open next week.
- There will Fitchfork upload slots for individual tasks and a slot for the final upload containing all your files.
- **Failure to upload your work will result in 0 marks being awarded for your practical.**

Demo Instructions

At the demo, we will check your uploaded work. Only students who have uploaded on time will be allowed to demo. **You may only demo during the slot where you are booked**. You should be able to demo the practical within Virtual Box.

- This practical must be marked by a tutor or teaching assistant during your booked practical demonstration sessions. Fitchfork uploads will also count a small percentage of the final mark.
- During the demo, you will have to show and explain your code to the tutor. The tutor will ask you questions with regards to the code and the practical in general. Make sure to complete the practical yourself, that you come prepared, and that you understand all of the questions you had to answer in writing.
- You have to be there at the start of the session until you get marked. If you come just before the session ends you will not get marked.
- You have a limited time to demo approximately 5 to 8 minutes. Please ensure that your VM is running before you get marked to speed up the process.
- If you are getting marked next and you are not ready to demo you will receive 0 marks for this practical.
- If your program doesn't compile due to syntax errors you will get 0 marks.
- If your program receives a runtime error you will lose marks.
- You have to be prepared to answer any question(s) the teaching assistant or tutor may ask you. You should not have to waste time looking for the answer.
- Make sure that you sign the attendance register for the session and ensure that the teaching assistant has entered your mark on the mark sheet correctly.
- **Failure to demo your work will result in 0 marks being awarded for this practical.**

Total: [25 Marks]

Task 1 - Deadlock and Mutual Exclusion [25 Marks]

Task 1(a) Deadlock - 6 marks

This task does not have a Fitchfork component. You only have to submit this task as part of your final submission.

In this task, you are required to implement methods in the given file **deadlock.cpp** downloaded from the CS website. Your need to implement these using threads, you may use high level **C++ threads** or **pthreads** for this purpose.
Methods to implement are defined below:

1. writeToFile(int threadNumber) - this simply writes data to a file called "deadlock.txt", the data that must be written consists of the following:
   - The first 10 iterations of the product of the **thread number** and **i** separated by a single space.
   - Where **i** is a loop variable from 1 to 10.
   - Example: if thread number is 2 the data that should be written to the file is: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20].

2. lock() - implement any type of locking mechanism of your choice. You may use a library for this.

3. unlock() - implement a unlocking mechanism with relation to the way you implemented the **lock()** function.

4. main(...) - You should have one command line argument that accepts the number of threads to create.
   - For each thread created you need to display a message as well as the thread ID.
   - Your threads should call the **threadFunction()** function.

Things to note:

1. You may **NOT** modify the function threadFunction() in this task.

2. Every time the program executes the contents of the file should be overwritten.

Your output should look something like this:

```
[~/Desktop/COS122/Practicals/2017/Practical3/code] => ./deadlock 5
Created Thread: 139799059453696
Created Thread: 139799051060992
Created Thread: 139798970889984
Created Thread: 139798962497280
Created Thread: 139798954104576
Something went wrong!
```

Task 1(b) Mutual Exclusion - 6 marks

This task does have a Fitchfork component. Make sure your deadlock is gone before submitting this task, since deadlocked programs will not return feedback and many such uploads can cause the server to slow down. You have to submit this task to the Fitchfork **and** the final upload slot. Fitchfork upload slots will open next week Monday (11 September), do not forget to upload.

In this task, you are required to implement methods in the given file **mutex.cpp** downloaded from the CS website. You may use the previous tasks code for this. Your need to implement these using threads, you may use high level **C++ threads** or **pthreads** for this. Things to note:

1. You may **NOT** modify the function threadFunction() in this task.
2. Every time the program executes the contents of the file should be overwritten.
3. The file that you write data to should be called "mutex.txt".
4. In this task there is an exception thrown in the threadFunction() you need to find a way to overcome this and unlock successfully in order to avoid the deadlock, without editing the threadFunction().

Your output should look something like this.

```
[~/Desktop/COS122/Practicals/2017/Practical3/code] => ./mutex 5
Created Thread: 140702068262656
Created Thread: 140701987763968
Created Thread: 140701979371264
Created Thread: 140701970978560
Something went wrong!
Created Thread: 140701962585856
Handled it!!
Something went wrong!
Handled it!!
Something went wrong!
Handled it!!
Something went wrong!
Handled it!!
Something went wrong!
Handled it!!
[~/Desktop/COS122/Practicals/2017/Practical3/code] => cat mutex.txt
 1 2 3 4 5 6 7 8 9 10
 5 10 15 20 25 30 35 40 45 50
 2 4 6 8 10 12 14 16 18 20
 3 6 9 12 15 18 21 24 27 30
 4 8 12 16 20 24 28 32 36 40
[~/Desktop/COS122/Practicals/2017/Practical3/code] =>
```

Task 1(c) File Locks - 8 marks

This task does have a Fitchfork component. Make sure your program works and that it does not hang before uploading. You have to submit this task to the Fitchfork

and the final upload slot. Fitchfork upload slots will open next week Monday (11 September), do not forget to upload.

You will notice in the previous task that the order in which the data written to the file is incorrect, this is due to the OS scheduling, sometimes threads may be scheduled sequentially but not always. It is good practice to never rely that the OS will schedule the threads sequentially thus you need to implement file locks in order to make sure that the order in which the threads are writing to the file is preserved. You are required to implement methods in the given file **filelock.cpp** downloaded from the CS website. You have to implement a file-locking mechanism for this.

Things to note:

1. You can make use of the code you have written in Task1(b).
2. You may **NOT** modify the function threadFunction() in this task either.
3. Every time the program executes the contents of the file should be overwritten.
4. The file that you write data too should be called "filelock.txt".
5. You need to perform significant testing and error handling.

Hint: make use of hidden files (`https://www.le.ac.uk/oerresources/bdra/unix/page_39.htm`) to check if the file is locked or not. Also make use of a counter variable like a semaphore in the hidden file to have the data written to the file in order.
Your output should look something like this.

```
[~/Desktop/COS122/Practicals/2017/Practical3/code] => ./filelock 5
Created Thread: 139853153793792
Something went wrong!
Handled it!!
Created Thread: 139853153793792
Something went wrong!
Handled it!!
Created Thread: 139853153793792
Something went wrong!
Handled it!!
Created Thread: 139853153793792
Something went wrong!
Handled it!!
Created Thread: 139853153793792
Something went wrong!
Handled it!!
[~/Desktop/COS122/Practicals/2017/Practical3/code] => cat filelock.txt
 1 2 3 4 5 6 7 8 9 10
 2 4 6 8 10 12 14 16 18 20
 3 6 9 12 15 18 21 24 27 30
 4 8 12 16 20 24 28 32 36 40
 5 10 15 20 25 30 35 40 45 50
[~/Desktop/COS122/Practicals/2017/Practical3/code] =>
```

Task 1(d) Review Questions - 5 marks

The following task requires you to write short answers to the questions below, you may be penalized for lengthy answers.

- How did deadlock occur in Task 1(a)?
- Deadlocks should not occur in practice, how should a programmer safeguard a system in order to prevent a deadlock situation to occur as seen in Task 1(a)?
- When writing data to a file which is a better type of lock mechanism to use?
- Give one advantage and one disadvantage of using file locks?

Uploads

Make sure you have uploaded task 1(b) and 1(c) to the Fitchfork slots. The Fitchfork uploaded archives should contain your source code files (use the given naming convention) and a makefile than can compile and run your code. Upload your practical with the files from all tasks to the final upload slot. This final upload will be used during the practical demo.

Total: [25 Marks]