|  |
| --- |
| CSC413/CSE315  **Design of Operating System**  Programming Assignment – Autumn 2024 |

**Due on: 9th January 2025**

**Please read the following information very carefully.**

* The assignment is divided into 2 sections: a **mandatory** section and an **optional** section. The mandatory section is worth 100 marks. *The optional section is not mandatory!!!!*
* Ensure you submit on time. Late submissions may be penalized.
* **Answer** every question to the **best of your abilities** and **do not resort to plagiarism** as you will be asked to present your answers at **viva**. Failure to show competence according to your answers will be marked as plagiarism and dealt with accordingly.
* The optional section contains difficult and tough questions which most may not be able to answer properly. **Only attempt these questions after you are done with the mandatory set.**  
  That being said, successfully answering any optional question will merit you much praise and significant consideration in the grading for this course.
* Submissions are individual. *However, you may participate in a group to solve the optional section problems. Clearly state all students who worked in the group to merit this bonus.*
* Attempt to **answer every question in** the **mandatory** section in its own code file/folder.
* Ensure your deliverables are in a **zip** file with the following prefix.  
  “<Section Number>\_<ID>\_<Course Name>\_<Student Name>.zip”  
  ex: 3\_2120117\_CSC413-Ezhar Uddin.zip   
  ex: 1\_2091012\_CSE315\_Rezwan\_Ahmed.zip   
  This helps with automation. Your submission may not be graded if this prefix does not match.
* **WARNING: No code submission will be accepted inside a PDF or Word document**. Ensure all your code exists as code/project files that can be easily executed/run on demand.
* If needed, any textual justifications or descriptions of how you solved a problem can be accepted, either through code comments or if necessary, a separate PDF or Word document. But the questions will be asked primarily in viva.
* For **viva**, ensure you have a **laptop** that is at the very least, capable of running all the applications in the mandatory section. You may need the following:
  + An IDE capable of compiling Java 8/JavaFX [ex: IntelliJ/NetBeans]
  + Linux Desktop Environment + Build tools [ex: Ubuntu/Red Hat]
  + Windows Desktop Environment + An IDE capable of compiling WIN32 applications [ex: CodeBlocks]

If you do not have this, you do not need to appear for the viva.

* All students who worked in a group to solve an optional problem must be present at the viva together.
* You have to appear at the viva only for the questions which you have answered successfully. If you did not submit the programming assignment you do not need to appear for the viva.

# Mandatory Section

You should answer every question in this section. Submit the code/projects via ZIP SUBMISSION and prepare talking points for the questions in the bulleted lists.

1. **Socket Communication:** Produce a networking-capable application in Java that is capable of free one-way socket communication: **[25]**

There are two separate programs to produce, a client and server application. The client program will receive arbitrary input from the command line argument. The client program's task is to send an arbitrary text message to the server. Once it successfully sends this message to the server, it should close immediately, signalling that its task was complete. All common exceptions must be handled. Here is an example of the output expected.

|  |
| --- |
| $ ./client\_msg Hello Everyone!  Connecting to server 127.0.0.72:58291...  Message Sent!  $ |

The server program’s task is to receive any and every client that may connect to this server and successfully obtain the message that they sent. Then it should display this message in a nice and clean format, ensuring that clients are differentiated.

|  |
| --- |
| $ ./server\_msg  Server hosted at 127.0.0.72:58291!  [127.0.0.8] Hello Everyone!  [127.0.0.8] aajsdfjkasdfjnqawnern234nksjndf32 1jk23kj12 12n3kj12jk3kj123jk1jk4jk124j12j3njk12jk3njkndjkfasfj12jk3j1k23jdsjkfjkasd jkasnfjk asjdf jk123j4j sadjf sdkfnjka23jn jkzsdjf kajsndf  [127.51.32.3] test |

You should have to deal with the following challenges:

* How you managed to code the programs in Java
* Be capable of running both applications and demonstrating expected behaviour.
* Explain the behaviour observed
* Produce valid output every time the program is run.

1. **WIN32 Programming**: Write a small C/C++ Windows application that utilizes the WIN32 API. **[25]**

The program is a simple command-line Windows executable that constantly prints the position of the mouse cursor, in terms of screen coordinates x and y. You are free to dictate the rate at which the cursor position updates but ensure that the mouse position continues to update even when the application loses focus. You must only use the WIN32 API to obtain this information. Refer to the picture for understanding.

A screenshot of a computer

Description automatically generated

The program must be a native Windows C/C++ application: using other libraries to obtain the mouse position like JavaFX, SDL or GTK is not allowed. You should also be able to describe:

* How you installed and used the Windows SDK
* How you solved the implementation
* Documentation/resources you needed to refer to

1. **Multithreading:** Write a C/C++ application that employs multiple threads to perform a Monte Carlo simulation. **[25]**

The program should approximate the result of an integration . It should achieve this by using the [Monte Carlo method](https://en.wikipedia.org/wiki/Monte_Carlo_method). Here is a summary:

|  |
| --- |
| **Monte Carlo Algorithm:**   1. Generate a random point in the rectangular region bounded by and maximum value of in the interval 2. Check if this point falls above or below the curve  * If y-coordinate of x-coordinate of   + then this is below the curve * If y-coordinate of x-coordinate of   + then this is above the curve  1. If a set of points are sampled from a continuous and uniform probability distribution, then   And   1. Therefore, generate a lot of random points and use them like so   Area below curve |

|  |  |
| --- | --- |
| A graph of a function  Description automatically generated | The picture shows an intuitive understanding of how the Monte Carlo Method approximates the area under a curve. Here, the method is being used to approximate by determining the area under |

The program should evaluate the definite integral of some function (defined in code) from to by spawning 8 threads and having each of those threads generate random numbers to speed up the computation. These numbers are used as points for the Monte Carlo Algorithm which then classify the point as being over or under the curve. After all work is done, the main thread should then print the estimated area.

The arguments to the program should look like this

|  |
| --- |
| ./monte\_carlo <x\_lowerbound> <x\_upperbound> <max\_y> <num\_points\_to\_generate> |

As an example, given the function was defined as

|  |
| --- |
| ./monte\_carlo 0 1 4 1000000000 Area: 3.141590912 |

Should be approximately equal to .

Your job is to:

* Explain how you solved the problem.
* Perform the required task successfully using multithreading.
* Produce valid output every time the program is run.
* Be capable of handling various kinds of input.
* Be capable of correctly integrating various kinds of 2-dimensional functions i.e. .

## **The Producer – Consumer Problem [25]**

In Section 7.1.1 (Book), we presented a semaphore-based solution to the producer– consumer problem using a bounded buffer. In this project, you will design a programming solution to the bounded-buffer problem using the producer and consumer processes shown in book Figures 7.1 and 7.2. The solution presented in Section 7.1.1 uses three semaphores: empty and full, which count the number of empty and full slots in the buffer, and mutex, which is a binary (or mutualexclusion) semaphore that protects the actual insertion or removal of items in the buffer. For this project, you will use standard counting semaphores for empty and full and a mutex lock, rather than a binary semaphore, to represent mutex. The producer and consumer—running as separate threads —will move items to and from a buffer that is synchronized with the empty, full, and mutex structures. You can solve this problem using either Pthreads or Java API.

## **The Buffer**

Internally, the buffer will consist of a fixed-size array of type buffer item (which will be defined using a typedef). The array of buffer item objects will be manipulated as a circular queue. The definition of buffer item, along with the size of the buffer, can be stored in a header file such as the following:

/\* buffer.h \*/

typedef int buffer item;

#define BUFFER SIZE 5

The buffer will be manipulated with two functions, insert item() and remove item(), which are called by the producer and consumer threads, respectively. A skeleton outlining these functions appears in Figure D.1. The insert item() and remove item() functions will synchronize the producer and consumer using the algorithms outlined in Figure 7.1 and Figure 7.2. The buffer will also require an initialization function that initializes the mutual- exclusion object mutex along with the empty and full semaphores.

The main() function will initialize the buffer and create the separate producer and consumer threads. Once it has created the producer and consumer threads, the main() function will sleep for a period of time and, upon awakening, will terminate the application. The main() function will be passed three parameters on the command line:

* 1. How long to sleep before terminating
  2. The number of producer threads
  3. The number of consumer threads A skeleton for this function appears in Figure D.2.

#include "buffer.h"

/\* the buffer \*/

buffer item buffer[BUFFER SIZE];

int insert item(buffer item item) *{*

/\* insert item into buffer return 0 if successful, otherwise

return -1 indicating an error condition \*/ *}*

int remove item(buffer item \*item) *{*

/\* remove an object from buffer placing it in item return 0 if successful, otherwise

return -1 indicating an error condition \*/ *}*

**Figure D.1** Outline of buffer operations.

## **The Producer and Consumer Threads**

The producer thread will alternate between sleeping for a random period of time and inserting a random integer into the buffer. Random numbers will be produced using the rand() function, which produces random integers between 0 and RAND MAX. The consumer will also sleep for a random period of time and, upon awakening, will attempt to remove an item from the buffer.

An outline of the producer and consumer threads appears in Figure D.3

#include "buffer.h"

int main(int argc, char \*argv[]) *{*

/\* 1. Get command line arguments argv[1],argv[2],argv[3] \*/

/\* 2. Initialize buffer \*/

/\* 3. Create producer thread(s) \*/

/\* 4. Create consumer thread(s) \*/

/\* 5. Sleep \*/

/\* 6. Exit \*/

*}*

**Figure D.2** Outline of skeleton program.

#include *<*stdlib.h*>* /\* required for rand() \*/ #include "buffer.h"

void \*producer(void \*param) *{*

buffer item item; while (true) *{*

/\* sleep for a random period of time \*/ sleep(...);

/\* generate a random number \*/ item = rand();

if (insert item(item)) fprintf("report error condition"); else

printf("producer produced %d∖n",item);

*}*

void \*consumer(void \*param) *{*

buffer item item; while (true) *{*

/\* sleep for a random period of time \*/ sleep(...);

if (remove item(&item)) fprintf("report error condition"); else

printf("consumer consumed %d∖n",item);

*}*

**Figure D.3** An outline of the producer and consumer threads.

# Optional Section

Only attempt this section if you have completed the Mandatory Section. Answering any one problem is sufficient but feel free to solve more if you are capable. You can use any and every resource available to you to solve these problems, short of plagiarism.

1. **Introduction to Linux Kernel Programming:** In this project, you will learn how to create a kernel module and load it into the Linux kernel.

This project will require the use of a Linux OS and possibly the source code of the kernel you are using. Although you may use any text editor to write these C programs, you will have to use the terminal application to compile the programs, and you will have to enter commands on the command line to manage the modules in the kernel.

As you’ll discover, the advantage of developing kernel modules is that it is a relatively easy method of interacting with the kernel, thus allowing you to write programs that directly invoke kernel functions. It is important for you to keep in mind that you are indeed writing kernel code that directly interacts with the kernel. That normally means that any errors in the code could crash the system!

1. **Kernel Modules Overview**

The first part of this project involves following a series of steps for creating and inserting a module into the Linux kernel. You can list all kernel modules that are currently loaded by entering the command

|  |
| --- |
| $ lsmod |

This command will list the current kernel modules in three columns: name, size, and where the module is being used.

|  |
| --- |
| /\* simple.c: a file that demonstrates Linux modules \*/  #include <linux/init.h>  #include <linux/kernel.h>  #include <linux/module.h>  /\* This function is called when the module is loaded. \*/  int simple\_init(void)  {  printk(KERN\_INFO "Loading Kernel Module∖n"); return 0;  }  /\* This function is called when the module is removed. \*/  void simple\_exit(void)  {  printk(KERN\_INFO "Removing Kernel Module∖n");  }  /\* Macros for registering module entry and exit points. \*/  module\_init(simple\_init);  module\_exit(simple\_exit);  MODULE\_LICENSE("GPL");  MODULE\_DESCRIPTION("Simple Module");  MODULE\_AUTHOR("SGG"); |

The program above is a very basic kernel module that prints appropriate messages when it is loaded and unloaded. The function simple\_init() is the module entry point, which represents the function that is invoked when the module is loaded into the kernel. Similarly, the simple\_exit() function is the module exit point— the function that is called when the module is removed from the kernel.

The module entry point function must return an integer value, with 0 representing success and any other value representing failure. The module exit point function returns void. Neither the module entry point nor the module exit point are passed any parameters. The two following macros are used for registering the module entry and exit points with the kernel:

|  |
| --- |
| module init(simple\_init);  module exit(simple\_exit); |

Notice in the figure how the module entry and exit point functions make calls to the printk() function. printk() is the kernel equivalent of printf(), but its output is sent to a kernel log buffer whose contents can be read by the dmesg command. One difference between printf() and printk() is that printk() allows us to specify a priority flag, whose values are given in the <linux/printk.h> include file. In this instance, the priority is KERN\_INFO, which is defined as an informational message.

The final lines — MODULE\_LICENSE(), MODULE\_DESCRIPTION(), and MODULE\_AUTHOR()— represent details regarding the software license, description of the module, and author. For our purposes, we do not require this information, but we include it because it is standard practice in developing kernel modules and compilation will fail without a valid license.

This kernel module *simple.c* is compiled using the Makefile accompanying the source code with this project. To compile the module, enter the following on the command line:

|  |
| --- |
| $ make |

The compilation produces several files. The file *simple.ko* represents the compiled kernel module. The following step illustrates inserting this module into the Linux kernel.

1. **Loading and Removing Kernel Modules**

Kernel modules are loaded using the insmod command, which is run as follows:

|  |
| --- |
| $ sudo insmod simple.ko |

To check whether the module has loaded, enter the lsmod command and search for the module *simple*. Recall that the module entry point is invoked when the module is inserted into the kernel. To check the contents of this message in the kernel log buffer, enter the command

|  |
| --- |
| $ sudo dmesg |

You should see the message "Loading Module." Removing the kernel module involves invoking the rmmod command (notice that the ‘.ko’ suffix is unnecessary):

|  |
| --- |
| $ sudo rmmod simple |

Be sure to check with the dmesg command to ensure the module has been removed. Because the kernel log buffer can fill up quickly, it often makes sense to clear the buffer periodically. This can be accomplished as follows:

|  |
| --- |
| $ sudo dmesg -c |

Proceed through the steps described above to create the kernel module and to load and unload the module. Be sure to check the contents of the kernel log buffer using dmesg to ensure that you have followed the steps properly.

As kernel modules are running within the kernel, it is possible to obtain values and call functions that are available only in the kernel and not to regular user applications. For example, the Linux include file <linux/hash.h> defines several hashing functions for use within the kernel. This file also defines the constant value GOLDEN\_RATIO\_PRIME (which is defined as an unsigned long).

This value can be printed out as follows:

|  |
| --- |
| printk(KERN\_INFO "%lu\n", GOLDEN\_RATIO\_PRIME); |

As another example, the include file <linux/gcd.h> defines the following function.

|  |
| --- |
| unsigned long gcd(unsigned long a, unsigned b); |

which returns the greatest common divisor of the parameters a and b.

1. **Submission**

Once you can correctly load and unload a module, write a kernel module to complete the following additional steps:

* Print out the value of GOLDEN\_RATIO\_PRIME in the module\_init() function.
* Print out the greatest common divisor of 3,700 and 24 in the module\_exit() function.

You are expected to be able to demonstrate the compilation, explain problems/challenges you faced, and show successful loading/unloading of the module on your system.

1. **CSE** **213/CSC305 Project Extension:** In your previous study of Object-Oriented Programming, you were asked to develop a real-world Java GUI application that sought to solve some IT/digital needs of some business or organization. Modify that project so it is capable of RMI and client-server communications.

Your task is to investigate and point out aspects in your project that could be serviced by some remote server instead of being implemented on the client.

Having a method implemented on the client makes it vulnerable to being exposed by reverse engineering or hacking: hence sensitive functions or functions that work with sensitive data are often implemented through remote APIs. Alternatively, a method may also be extremely computationally intensive, to the point where it is unfeasible to have it be processed client-side: such methods are also qualified to be RMI, and for such applications, the server is equipped with a lot of processing power to handle such calls on a large volume.

* 1. **Submission**

Once you have documented such use cases, implement working RMI or RPC systems to service some key methods. A significant amount of work must be done for this optional question to be considered completed. You are also required to explain

* What functions/actions have been implemented via remote execution methods
* Justifications for implementing said functions as RMI
* Demonstrate execution of RMI
  1. **Why this is necessary**

Through revisiting past projects, students can identify areas for improvement, address any shortcomings, and implement new techniques or technologies that they may have learned since the project's completion. Moreover, extending previous projects provides an opportunity for students to apply their knowledge in a practical context, reinforcing key concepts and fostering a deeper appreciation for the subject matter.

Working incrementally on your projects also allows you to showcase your growth and development over time to future interviewers in job interviews and gives you more prominence in your career as a developer. As a result, we feel it is beneficial if you complete this optional and strongly recommend you take it seriously.

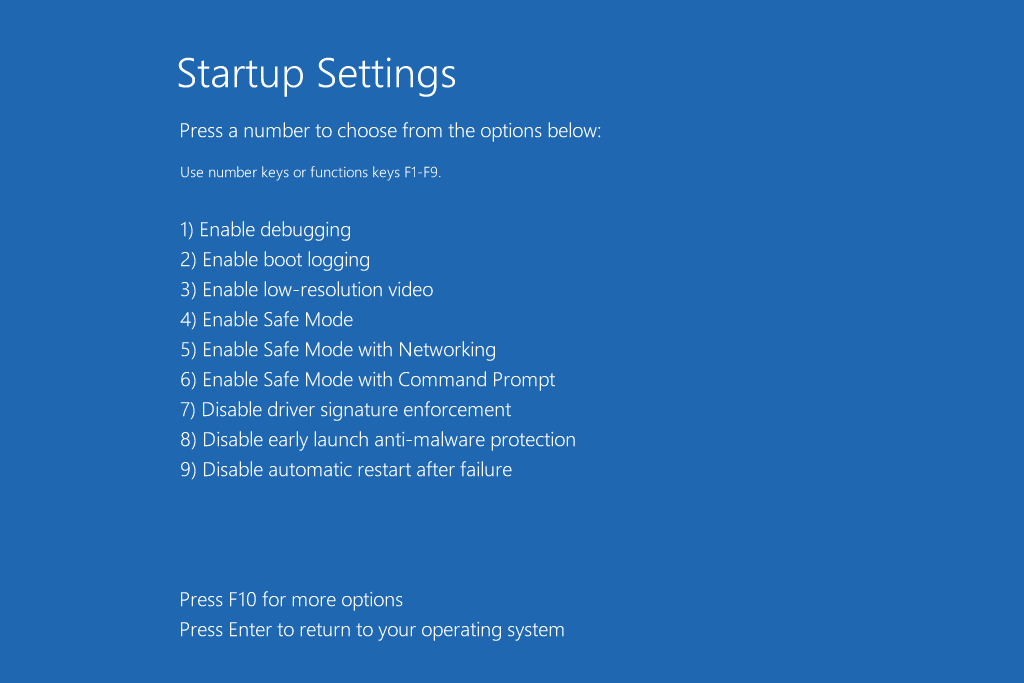
1. **WIN32 Driver Programming**: Demonstrate your ability to handle desktop applications programming and comprehend API and their documentation by writing a simple Windows driver.

Keep in mind WIN32 drivers are very similar to how Linux kernel modules work. No support will be provided for programming and compiling the driver: you must discover and find resources on the Internet that will help you to program a driver on your own. Once you have managed to do that,

1. **Running the Driver:**

The driver you have compiled should be something like driver.sys. However, getting any arbitrary driver to run on Windows is quite difficult compared to on Linux. This is because Windows employs [**driver signing**](https://learn.microsoft.com/en-us/windows-hardware/drivers/install/driver-signing) to ensure that unsafe or dangerous code in drivers cannot be freely executed. The kernel-mode code signing policy enforces that kernel-mode drivers must be signed for the driver to load. Getting your drivers signed is only possible if you are a large enough company which can enter into an agreement with Microsoft or go through a strenuous [approval process](https://learn.microsoft.com/en-us/windows-hardware/drivers/install/kernel-mode-code-signing-policy--windows-vista-and-later-) and have your drivers be thoroughly tested by [WHQL](https://learn.microsoft.com/en-us/windows-hardware/drivers/install/whql-release-signature). As amateur and niche OS programmers we want to avoid this hassle.

To allow your unsigned driver to execute, you must first restart your PC into **Startup Settings/Advanced Boot Options** by holding Shift while clicking on Power > Restart, or pressing F8 when your computer is booting. Then press F7 or 7 to **disable driver signature enforcement**.



This allows unsigned drivers to temporarily execute during that session. After this, we need to create and start a service that uses our driver (this is somewhat like an entry-point for our driver, code in our driver needs to first start executing through some service). Open the command prompt with administrative privileges and type in

|  |
| --- |
| **> sc**  DESCRIPTION:  SC is a command line program used for communicating with the  Service Control Manager and services. ... **> sc create myDriver binPath= C:\Path\to\Driver\driver.sys type= kernel**  [SC] CreateService SUCCESS **> sc start myDriver**  The driver will execute without any prompt. Check DebugView for output.  **> sc stop myDriver** This will unload the driver. Check DebugView for output (if you have handled it). **> sc delete myDriver** [SC] DeleteService SUCCESS |

1. **View output**

Windows is an operating system primarily built for its customers, the masses. To provide an easy, seamless, and user-friendly experience for most people, there are very few built-in tools that allow programmers to tinker around and manipulate the OS. There is no built-in analogue of the Linux Kernel log viewer dmesg. Hence to view the output of the driver we will need an additional software called [DebugView](https://learn.microsoft.com/en-us/sysinternals/downloads/debugview) from Sysinternals. Run DebugView as an administrator before you start the driver in the section above.



Then click on **Capture > Capture Kernel** and **Enable Verbose Kernel Output** to ensure you can see the output produced by your driver.

1. **Submission**

Once you can correctly load and unload a driver, write a driver and complete the following additional steps:

* Print out “Hello Kernel! <your name>” using DbgPrint() in the driver entry function that appears only in the kernel log output captured by DebugView.
* Print out “Goodbye Kernel!” when your driver is unloaded that appears only in the kernel log output captured by DebugView.
* [**EXTREMELY DIFFICULT**] Prove your driver is executing code in kernel mode by calling the kernel mode functions ZwCreateFile() and ZwWriteFile() and make some random file: show the NTSTATUS and IO\_STATUS\_BLOCK struct’s contents to show this file was written successfully. [You are considered successful even if your system crashes]

You are expected to be able to demonstrate the compilation, explain problems/challenges you faced, and show successful loading/unloading of the driver on your system.