C programming questions:

Q. What will the following program line print on the terminal

printf(“ The numbers are: %d, %f, %c, %s\n”, -50, 3.145, ‘z’, “String”);

Solution: The numbers are -50, 3.145, z, String

Q. Consider the following C code:

```c
#include <stdio.h>

int main() {

    int x = 1, y = 2, z = 3;
    int result = 0;

    do {
        x += y;
        y *= z;
        z--;
        result++;

    } while (z > 0 && x < 10);

    printf("Result: %d\n", result);
    printf("x: %d, y: %d, z: %d\n", x, y, z);

    return 0;
}
```

Determine the output that would be produced by this program. Explain your reasoning for each part of the output. If the program does not compile or produces an error, explain why. If the program runs indefinitely, explain why that happens. If there's more than one possible output, provide all possible outputs and explain the conditions under which each would occur.

Answer: result: 3
x: 21, y: 12, z: 0

The key to understanding this program is to carefully track the values of x, y, and z inside the do-while loop.

Q. What is the output of the following code snippet?

#include <stdio.h>

```
int main() {
    int i = 5;
    int *p = &i;
    printf("%d", *p++);
    printf("%d", i);
    return 0;
}
```

a) 55
b) 56
c) 65
d) 66

Correct answer: c) 65

Q. Read the following carefully and answer the question 5

```
#include <stdio.h>

int main() {

int numbers[] = {1, 2, 3, 4, 5};
int *ptr;
ptr = numbers;
*(ptr + 2) = 10;

for (int i = 0; i < 5; ++i) {
printf("%d ", numbers[i]);
}
printf("\n");

return 0;
}
```

What is the output of this code?

a) 1 2 3 4 5
b) 1 12 3 4 5
c) 1 2 10 4 5
d) 10 2 3 4 5

Answer: c

Please refer to the lecture 3 slides.

Q. Read the following carefully and answer the question 6:

```
#include <stdio.h>

int increment(int* number){
   *number = *number + 1;
   return 0;
}

int main() {

int number = 5;

increment(&number);

printf("%d\n", number);

return 0;
}
```

What is the output of this program?

a) 5
b) 6
c) 0
d) The program does not work (it has errors).

Answer: b

Please refer to the lecture 3 slides.

Q. What is the output of the following code snippet?

```
#include <stdio.h>
int main() {
   int x = 5;
   int y = 2;
   float result = x / y;
   printf("%f", result);
   return 0;
```

}
a) 2.500000
b) 2.000000
c) 2
d) Compiler error
Correct answer: b) 2.000000

Q. What is the output of the following code snippet?

```
#include <stdio.h>
int main() {
    float x = 5;
    float y = 2;
    float result = x / y;
    printf("%f", result);
    return 0;
}
```

a) 2.500000
b) 2.000000
c) 2
d) Compiler error
Correct answer: a) 2.500000

Q. Which header file is typically included to use delay functions in C programming for embedded systems?
a) <stdlib.h>
b) <unistd.h>
c) <time.h>
d) <stdint.h>
Correct answer: c) <time.h>

Q. A water level sensor is connected to a water tank. The sensor outputs 5V(HIGH) under normal conditions, and gives a 0V(LOW) signal when the water level exceeds a certain threshold. You are to complete the following code that detects this change using interrupts. More specifically, you have to fill in the missing part of the following code that specifies what type of interrupt will be used in this case(rising, falling, both).

```
void configure_interrupt(int gpio_number){ configure_gpio_input(gpio_number); // set gpio as input
char InterruptEdge[40];
sprintf(InterruptEdge, "/sys/class/gpio/gpio%d/edge", gpio_number);
```

FILE* fp = fopen(InterruptEdge, "w");
fwrite("falling", sizeof(char), 7, fp);
fclose(fp);}

Solution: missing part highlighted in green.

Q. Alice's C code is not running as expected. Explain to her the bug in each of the following:

```c
#include <stdio.h>

int main(int argc, char* argv[]){
    i = 0;
    while(i<10){
        i++;
        printf("this is line number %d\n", i);
    }
    return 0;
}
```

```c
#include <stdio.h>

int main(int argc, char* argv[]){
    int myArray[] = {7, 4, 4, 6, 11, 1};
    for(int i=0; i<7; i++){
        printf("index %d has value %d\n", i, myArray[i]);
    }
    return 0;
}
```

Embedded Linux Sample Questions:

Q. What is linux Kernel and what core functionalities does it provide

Solution: Linux kernel is the main component of a Linux operating system (OS) and is the core interface between a computer's hardware and its processes. It communicates between the 2, managing resources.
Core functionalities:

1. Memory management: Keep track of how much memory is used to store what, and where
2. Process management: Determine which processes can use the central processing unit (CPU), when, and for how long
3. Device drivers: Act as mediator/interpreter between the hardware and processes
4. System calls and security: Receive requests for service from the processes

Q. What is the usage of Linux Filesystem

Solution: Linux Filesystem is used to handle the data management of the storage in Linux OS. Most significant purpose of this is to manage the user data. It helps to arrange the file on the disk storage. This includes storing, retrieving and updating data. Some file systems accept data for storage as a stream of bytes which are collected and stored in a manner efficient for the media. It manages the file name, file size, creation date, and much more information about a file.

Q. Which Linux command helps navigate directories

Solution: Change directory(cd) command.
The cd command will allow you to change directories. When you open a terminal you will be in your home directory. To move around the file system you will use cd. Examples:

- To navigate into the root directory, use "cd /"
- To navigate to your home directory, use "cd" or "cd ~"
- To navigate up one directory level, use "cd .."
- To navigate to the previous directory (or back), use "cd -"
- To navigate through multiple levels of directory at once, specify the full directory path that you want to go to. For example, use, "cd /var/www" to go directly to the /www subdirectory of /var/. As another example, "cd ~/Desktop" will move you to the Desktop subdirectory inside your home directory.

Q. Which Linux command is used to change file permission to ‘executable (x)’ for all users

Solution: To change file and directory permissions, chmod (change mode) is used. The owner of a file can change the permissions for user (u), group (g), or others (o) by adding (+) or subtracting (-) the read, write, and execute permissions.

For permitting access to all the user:

```
chmod a+x
```

Q. List two advantages of shell scripts

Solution:

- To automate the frequently performed operations.
- To run a sequence of commands as a single command.

Q. What is the primary purpose of makefile system

Solution: A makefile is basically a script that guides the make utility to choose the appropriate program files that are to be compiled and linked together. The purpose of a makefile system is to easily build an executable that might take many commands to create.

Q. How to provide an input to a C program from the terminal

Solution: scanf() function can be used to provide an input to a C program from the terminal
Example code:

```
int a;
printf("Enter the value:");
scanf("%d", &a);
```

Output: Enter the value: 2

Q. Write a sample code that opens a file at path “E:\cprogram\newprogram.txt” with write permission, writes integer values from 1 to 10 to the file, then closes the file

```
#include <stdio.h>

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

 // DECLARE a FILE * variable
  FILE *infile;

// open file
infile = fopen("E:\cprogram\newprogram.txt","w");
```

```
// checking file
   if((infile == NULL) {
      printf("Can't open file \n");
      exit(1);
   }
// putting values in to the text file
for(int n=0;n<10;n++) {
      fprintf(infile,,"%d\n", n+1 );
    }
// close file
   fclose(infile);
}
```

Q. What is the output of "top" Linux command

Solution: Top command shows all running processes
First line indicates

- current time
- uptime of the machine
- users sessions logged in
- average load on the system (load average: 8.02, 8.67, 9.01) the 3 values refer to the last minute, five minutes and 15 minutes.

The second row provides task information:

- Total Processes running
- Running Processes
- Sleeping Processes
- Stopped Processes
- Processes waiting to be stopped from the parent process (*Zombie Process*)

Third row indicates how the cpu is used.
Fourth and Fifth Row : Memory usage

Q. A) Bob just finished coding his first program for his BeagleBoneBlack on his native machine and needs to transfer the file to his board for testing. He opens his terminal and changes directory to the one containing his source code, "mycode.c" and there are no other files. Write the entire command needed to transfer the file to debian's home directory on his board.

B) Bob uses the above command and then uses SSH to enter the BeagleBone's shell. Assuming he is already in the debian home directory, write the entire command needed to compile his code into a program called "hello_world."

C) Bob is still new to C programming and has been fixing lots of mistakes in his code using the nano code editor. He is frustrated with having to type the above command repeatedly. Write a basic makefile implementation he can use to speed up this process.

Beaglebone Sample Questions:

Q. Which of the following GPIO numbers corresponds to pin GPIO1_28 on the BeagleBone Black?

A) GPIO 28
B) GPIO 60
C) GPIO 61
D) GPIO 92

Answer: B

Given GPIOX_Y, GPIO number = 32*X + Y = 32*1 + 28 = 60

Q. Why almost every pin on beaglebone offer many functionalities e.g, P8_13 can be used both as PWM and GPIO, P9_21 can be configured as GPIO or UART or SPI or PWM

Solution: Pins on beaglebone have multiplexer mode (mmode), because of device trees based Pin Configuration.

Q. Using a BeagleBone Black, write a shell script to toggle an LED connected to one of the GPIO pins. Assume the LED is connected to pin P8_8. Provide the script to accomplish this task.

Answer:

```bash
#!/bin/bash

# Define the GPIO pin for the LED
LED_GPIO="P8_8"

# Export the GPIO pin
echo $LED_GPIO > /sys/class/gpio/export

# Set the direction of the pin to output
echo "out" > /sys/class/gpio/gpio$LED_GPIO/direction

# Infinite loop to toggle the LED
while true
do
    # Turn the LED on
    echo 1 > /sys/class/gpio/gpio$LED_GPIO/value
    sleep 1

    # Turn the LED off
    echo 0 > /sys/class/gpio/gpio$LED_GPIO/value
    sleep 1
done
```

Q. You are working on configuring interrupts on a GPIO pin in a C program. Below is a function that configures an interrupt on a specified GPIO pin. The function initially configures the interrupt on the falling edge. Your task is to modify the function to configure the interrupt on both rising and falling edges. Modify the following code to achieve this.

```c
#include <stdio.h>

void configure_interrupt(int gpio_number){
    // set gpio as input
    configure_gpio_input(gpio_number);

    // configure interrupts using edge file
    char InterruptEdge[40];
    sprintf(InterruptEdge, "/sys/class/gpio/gpio%d/edge", gpio_number);

    // configure interrupt on falling edge
    FILE* fp = fopen(InterruptEdge, "w");

    // configure interrupt on both edges
    fwrite("falling", sizeof(char), 4, fp);
    fclose(fp);
}
```

Answer: *fwrite("both", sizeof(char), 4, fp);*

Q. Write a C program to monitor a push button connected to pin P9_12 of the BeagleBone Black. Configure the pin as an input with an internal pull-up resistor enabled. The program should continuously monitor the button state and print a message whenever the button is pressed. Assume that the button press is detected when the pin reads 0 due to the pull-up resistor.

Q. What are different kind of timers on beaglebone
Beaglebone black has multiple timers.

Solution:

a. Dual-mode(DM) timers. DM timer is a 32 bit timer that contains a free running upward counter with auto reload capability on overflow. DMTimer can be configured in three modes of operation.

b. DMtimer 1ms
   - Implemented using the DMtimer_1ms software module,
   - Generates an accurate 1 ms tick using a 32.768 khz clock.
   - Can be used to schedule wake-ups

c. Real Time Clock Subsystem timer, clock that keeps track of the current time of day.

d. Watchdog timer : Electronic timer that is used to detect and recover from computer malfunctions.

Q. Locate all XTAL oscillators on beaglebone

Solution: Beaglebone black has three XTAL oscillators:

- A 24 MHz oscillator for AM3358
- A 25 MHz oscillator for Ethernet
- A 32kHz RTC

Q. Give an example of System clock in beaglebone

Solution:
System Clock:

- CLOCK_REALTIME - System-wide clock, visible to all processes running on the system

Q. How to get beaglebone kernel version, see network configuration, and get CPU information

Solution:

-Kernel version: uname command can be used.

-Network configuration: The hostnamectl command is typically used to display information about the system's network configuration. It also displays the kernel version.

-CPU information: You can simply view the information of your system CPU by viewing the contents of the /proc/cpuinfo file with the help of cat command like below:

```
cat /proc/cpuinfo
```

Q. Name two boot modes for BBB. Which one is the default boot mode

Solution:

a. eMMC boot(MMC1)
b. SD boot(MMC0)

Default boot mode is eMMC boot.

Q. How to configure a beaglebone pin to PWM mode

Solution: config-pin <pin_number> pwm

Q. How many ADC interfaces are available in Beaglebone Black and what is the length of ADC input

Solution: There are 7 ADC pins available with 12 bit input.

Q. What is the purpose of a pull up resistor for a GPIO input? Explain how a GPIO input pin with a pull up resistor can be used to detect a button press.

Q. How many timer pins are available on the beaglebone P8 and P9 pin headers

Solution: 4

Q. The filesystem path for beaglebone GPIO is, /sys/class/gpio. Find the path for beaglebone PTP clock (/dev/ptp/). How many PTP devices (directories) did you find? Enter the directory

for PTP device (ptp0)

Solution: The path to PTP clock is /dev
You will find one ptp device: ptp0

Qualitative Question Samples from Lectures:

Q. What is the difference between microprocessor and microcontroller

Solution:

- Microprocessor – (the physical processor chip) : composed of control unit, register, arithmetic and logic units
- Microcontroller: Central core of microprocessor but limited capabilities in regards to registers, memory size, and speed.

Q. Name different components of a microcontroller

Solution:

- On board memory
- Several Timers
- I/O configurable ports
- In implementation, may or may not have a keyboard, rather a keypad/switches for input or other types of control, often does not have monitor

Q. What is pulse width modulation? What do period and duty cycle represent?

Q. What is difference between Monotonic and Raw_Monotonic clock in Linux
Solution: Monotonic clock gives the system approximation of seconds from an arbitrary epoch whereas raw_monotonic clock advances time as counted by local oscillator

Q. What is a POSIX clock
Solution: POSIX is a standard for implementing and representing time sources. In contrast to the hardware clock, which is selected by the kernel and implemented across the system; the POSIX clock can be selected by each application, without affecting other applications in the system.

Q. What is a bootloader

Solution: A piece of code that runs before any operating system is running, usually each operating system has a set of bootloaders specific for it. It Contain several ways to boot the OS kernel and commands for debugging and/or modifying the kernel environment

Q. What are the differences between polling and interrupts

Solution: The main difference between interrupt and polling is that in interrupt, the device notifies the CPU that it requires attention while, in polling, the CPU continuously checks the status of the devices to find whether they require attention. In brief, an interrupt is asynchronous whereas polling is synchronous

Q. A CPU sharing technique in which "Two or multiple programs take turns to use the CPU, and programs YIELD control voluntarily to let other programs use the CPU." is known as:

a) Subroutines
b) Coroutines
c) Threads
d) Events

Solution: b

Please refer to lecture 10

Q. What is the difference between Concurrency and Parallelism

Solution: Concurrency is about dealing with lots of things at once. Parallelism is about doing lots of things at once.

Q. How to find the optimal sampling rate for sampling analog inputs

Solution: Use Nyquist theorem

Q. Why do we clean sensor data before processing it

Solution: To eliminate noise from system, environmental and human errors

Q. What are the causes of missing sensor data

Solution: sensor defect, network disconnection, corrupt storage

Q. Why critical sections inside a thread should be kept small

Solution: So the CPU can context switch to other threads instead of being locked to a single thread

Q. An embedded system has two sensors (temperature and proximity sensor), two actuators (thermostat, screen), and one processor. Temperature readings are sampled periodically.

A window of temperature samples are processed, based on which a thermostat actuator is controlled. The proximity sensor only generates a reading when it detects motion in vicinity. As soon as the motion is detected, the screen is turned on for a duration of 10 seconds. Screen turns off in the absence of movement. Present a multi-threading approach and an event-driven approach to develop this embedded application. List drawbacks of each approach and explain which will be the optimal approach. You also have the option to come up with a hybrid approach (combination of both multi-threading and event-driven approaches).

Solution: Refer to your lectures and readings on CPU Sharing

Q. A sensor data has arrived and caused an interrupt. At the same time, a divide by zero exception occurred in the main program. Which interrupt is handled first

Solution: Trap sources (such as divide by zero exception) have higher interrupt priorities than external interrupt sources (sensor interrupt)

Q. To poll a GPIO pin on BBB, we use "epoll" Linux API and monitor events of interest. Which events should we look for when GPIO pin value changes: EPOLLIN and/or EPOLLOUT and/or EPOLLET

Solution: Each of the three flags indicate the following about the status of the file being monitored:

EPOLLIN: signals file is available for read operations

EPOLLOUT: signals file is available for write operations

EPOLLET: only raise the above signal when the read value in the file has changed

We know that in Linux, interrupts are handled by the GPIO driver, which writes GPIO values to the file at: /sys/class/gpio/gpio<number>/value. We are interested in reading from the GPIO file, hence we use the flag EPOLLIN. Interrupts are either configured with a rising or falling edge, which means whenever an interrupt occurs, the value available in GPIO file changes either from 0 to 1 or vice versa. Hence, we use the EPOLLET alongwith EPOLLIN, since we are only interested in detecting the change in value in the GPIO file.

Q. Fill in the missing parts of the following code to write a function that doubles the value of its input number, and execute it as a thread.

```
#include <stdio.h>
#include <stdlib.h>

#include <unistd.h>
```

```
#include <pthread.h>

threadFunction(void *var)
{
        int temp = (int*) var*2;
        sleep(1);
        printf("Doubled value %d is:\n", *temp);
        return NULL;

}

int main()  {
int input = 5;

Pthread_t thread_id;
printf("Before Thread\n");
pthread_create( %thread_id, NULL,threadFunction, (void*)(&data)) ;
pthread_join(thread_id, NULL);
printf("After Thread\n");
pthread_exit(0);

}
```

Solution: missing parts highlighted in green.

Q. What will happen when the user presses Ctrl+C in the following code?

```
#include <stdio.h>
#include <signal.h>
void signal_handler(int signal) {
   printf("Received signal: %d\n", signal);
}
int main() {
   signal(SIGINT, signal_handler);
   printf("Press Ctrl+C to trigger the signal...\n");
   while(1) {}
   return 0;
}
```

a) The program terminates
b) The signal handler function is executed
c) Both a and b happen

d) Nothing happens, the program keeps running
Solution: The program terminates and the signal handler function is executed

Q. What is the output of this program?

```
#include <stdio.h>
#include <pthread.h>
void* thread_function(void* arg) {
   printf("Thread says: %s\n", (char*) arg);
   pthread_exit(NULL);
}
int main() {
   pthread_t thread_id;
   char* message = "Hello, world!";
   pthread_create(&thread_id, NULL, thread_function, (void*) message);
   pthread_join(thread_id, NULL);
   return 0;
}
```

a) Thread says: Hello, world!
b) Hello, world!
c) The program will not compile
d) The program will run but will not output anything
Solution: Thread says: Hello, world!

Q. Answer the following questions regarding threads
A) A critical section describes what?
B) What primitive is used to synchronize access to shared resources and how it is used?
C) What is a context switch and when does it occur?
D) What CFLAG is needed to compile a multi-threaded application with gcc?
E) It is impossible to know when a thread will be pre-empted: TRUE FALSE
F) Critical sections should be as large as possible to maximize thread utilization: TRUE FALSE
G) Each thread has its own stack: TRUE FALSE
H) When a new thread is instantiated, the creator waits for it to finish executing: TRUE FALSE
I) Threads will always run in the order they were created: TRUE FALSE

Q. Fill in the missing parts of the following code to write a function that double the value of its input number, and execute it as a thread.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

 #include <pthread.h>

threadFunction(void *var)
{
        int temp = (int*) var*2;
        sleep(1);
        printf("Doubled value %d is:\n", *temp);
        return NULL;

}

int main()  {
int input = 5;

Pthread_t thread_id;
printf("Before Thread\n");
pthread_create( %thread_id, NULL,threadFunction, (void*)(&data)) ;
pthread_join(thread_id, NULL);
printf("After Thread\n");
pthread_exit(0);

}
```

Solution: missing parts highlighted in green.

Q. What is the purpose of an interrupt in embedded systems?
a) To halt the execution of the program
b) To allow the processor to respond to asynchronous events
c) To increase the clock speed of the processor
d) To execute a specific function periodically
Correct answer: b) To allow the processor to respond to asynchronous events

Q. Write a C program that creates two threads. The first thread should print all even numbers between 1 and 10, while the second thread should print all odd numbers

between 1 and 10. Ensure that the main thread waits for both threads to finish execution before printing "Done."

Answer:

```c
#include <stdio.h>
#include <pthread.h>

// Function prototypes
void *printEven(void *arg);
void *printOdd(void *arg);

int main() {
    pthread_t evenThread, oddThread;

    // Create threads
    pthread_create(&evenThread, NULL, printEven, NULL);
    pthread_create(&oddThread, NULL, printOdd, NULL);

    // Wait for threads to finish
    pthread_join(evenThread, NULL);
    pthread_join(oddThread, NULL);

    printf("Done.\n");

    return 0;
}

// Function to print even numbers
void *printEven(void *arg) {
    printf("Even numbers: ");
    for (int i = 2; i <= 10; i += 2) {
        printf("%d ", i);
    }
    printf("\n");
    pthread_exit(NULL);
}

// Function to print odd numbers
void *printOdd(void *arg) {
    printf("Odd numbers: ");
    for (int i = 1; i <= 9; i += 2) {
        printf("%d ", i);
    }
    printf("\n");
    pthread_exit(NULL);
}
```

Q. The following program has two threads where both threads print numbers from 0 to 49. What is the order of execution of threads, and how many times thread 1 and thread 2 will be preempted?

```c
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <unistd.h>
#include <pthread.h>
void* threadFuncOne(void *var){
	pthread_t thread_id = pthread_self(); // this
	for(int i=0; i<50; i++){
		printf("Printing %d inside thread with ID: %u\n", i, thread_id);
	}
	return NULL;
}
void* threadFuncTwo(void *var){
	pthread_t thread_id = pthread_self(); // this
	for(int j=0; j<50; j++){
		printf("Printing %d inside thread with ID: %u\n", j, thread_id);
	}
	return NULL;
}
int main() {
	int data = 6;
	pthread_t thread_id, thread_id2;
	pthread_create(&(thread_id), NULL, threadFuncOne, (void*)(&data) );
	pthread_create(&(thread_id2), NULL, threadFuncTwo, (void*)(&data) );
	pthread_join(thread_id, NULL);
	pthread_join(thread_id2, NULL);
	pthread_exit(0);
}
```

Solution: The order of execution is determined by the OS and we are not aware of it. The OS can preempt the threads many times and we cannot know the number of times OS preempts these threads.

Q. Read the following carefully and answer the question 2:

Consider the following two functions, each of the two functions are being executed in separate threads, and share the variable buffer:

```c
// defining buffer size
#define SIZE 26

void* inputThread(void* input) {


    while(1){
```

```
      // wait before next write until the buffer has space
      pthread_mutex_lock(&lock);  // acquires the lock; i.e. marks the
start of critical section
      while (count == SIZE) {
      }
        count++;
        // receive input from the user
        printf("Type the next character:");
        buffer[putIndex] = getch();
        putIndex++;
        // if the buffer has reached its end, the next character will be
written at the start of buffer
        if (putIndex == SIZE) {
            putIndex = 0;
        }
      pthread_mutex_unlock(&lock);  // releasing the lock; marking the
end of critical section
    }
}


void* outputThread(void* input) {

    while(1){
        // wait if the buffer is empty
      pthread_mutex_lock(&lock);  // acquires the lock; i.e. marks the
start of critical section
      while (count < SIZE) {
      }
        // read the character from buffer, update variables and print
the character to terminal
        printf("Character at index %d received: ", getIndex);

        for(int i=SIZE; i >0; i--){
          count--;
          c = buffer[getIndex];
          getIndex++;
          //printf("%d\n", count);
          printf("%c", c);
          // return index to zero after reading the last character in
the buffer
```

```
          if (getIndex == SIZE) {
            printf("\n");
            getIndex = 0;
          }
        }
      pthread_mutex_unlock(&lock);  // releasing the lock; marking the
end of critical section

    }
}
```

Question 2:

Here the inputThread should receive characters from the user until the buffer is filled (26 characters are received), and then outputThread should print out all the characters in the buffer. In the current state the two threads may not work properly. What changes should be made for the two threads to work properly?

Answer:

Both threads start by acquiring the lock first, so whenever each thread starts it will acquire the lock and only then it can move forward otherwise it will be stuck at the **pthread_mutex_lock(&lock)** call (Remember that only one thread can hold the lock at a time, which means only one thread will move past the **pthread_mutex_lock(&lock)** call at a time).

There are two scenarios when the program starts:

a) **The first execution of the inputThread starts before the outputThread ever starts.**
   "inputThread" has the following while loop at the start,

```
while (count == SIZE) {}
```

   When the buffer gets filled, the thread will get stuck at this loop while it has also acquired lock for itself. It waits for outputThread to empty the buffer. But the outputThread cannot empty the thread because it cannot acquire the lock. So, we need to modify the loop as below:

```
while (count == SIZE) {
            pthread_mutex_unlock(&lock);
            pthread_mutex_lock(&lock);
      }
```

Now, while waiting for the outputThread to empty the buffer, we continuously release and acquire the lock for inputThread. This gives an opportunity to outputThread to acquire the lock while inputThread has called **pthread_mutex_lock(&lock)** but before inputThread calls **pthread_mutex_unlock(&lock).** Once outputThread has acquired the lock, it can empty the thread.

b) **The first execution of the outputThread starts before the inputThread ever starts.**
"outputThread" has the following while loop at the start:

```
while (count < SIZE) {}
```

The loop waits for the input buffer to get filled while it has acquired a lock for outputThread using **pthread_mutex_lock(&lock)** at the start of this function. So, inputThread cannot fill in the buffer since it will be stuck at **pthread_mutex_lock(&lock)** at its start waiting for the lock. So, we will change the while loop to the following:

```
while (count < SIZE) {
            pthread_mutex_unlock(&lock);
            pthread_mutex_lock(&lock);
      }
```

With this modified code, while we wait for the buffer to be filled by inputThread, we continuously release and re-acquire the lock. This presents an opportunity for inputThread to acquire lock in between **pthread_mutex_lock(&lock)** and **pthread_mutex_unlock(&lock)** calls by the outputThread and fill the buffer.