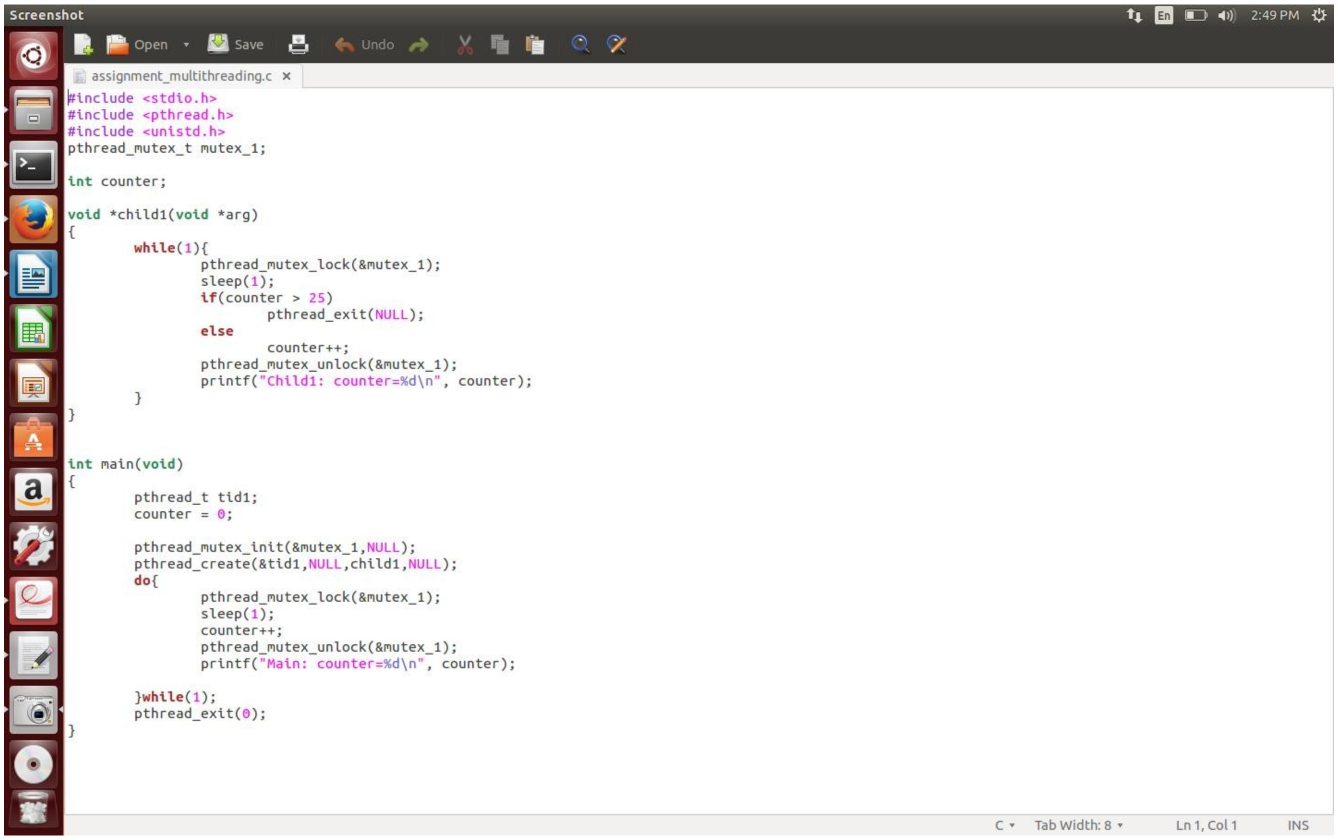
INTRODUCTION TO OPERATING SYSTEMS (ASSIGNMENT-2) VIVEK REDDY(vrp263)

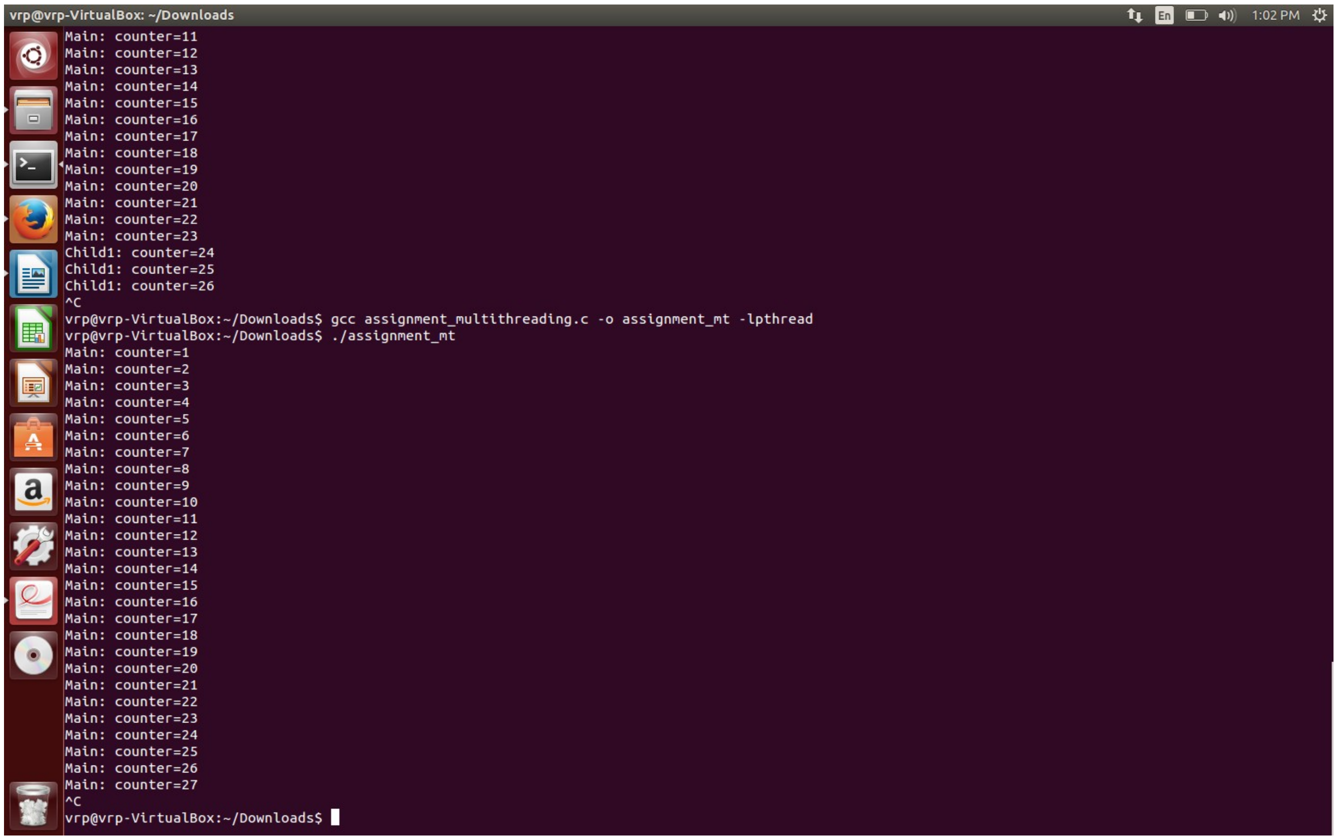
1. **This program creates a new thread. Both the main thread and the new thread then increment the variable counter infinitely. After incrementing the value of counter, each thread prints a message showing the value of the variable. One of the threads exits when it finds that the counter value has exceeded 25. Run the program and explain the output. Why do the print statements stop appearing after a certain point in the program? Explain.**

   “The program has different output at each time we execute the program that is because we never know which thread will execute after the given if condition , because that will be in the hands of the OS scheduler that is linux scheduler and we do not have any control over which thread exits after the condition.

   The code given implements a mutex semaphore which instantiate and locks a global variable known as counter and when the thread is run it increments counter by 1, prints the counter value and unlocks the resource that is in this case the counter variable. The child thread unlike the main thread checks if the counter variable has exceeded 25 each time it is run and when this condition returns a that is true, the thread stops executing. After this, when the main thread tries to access the counter variable, thread gets blocked because child thread which ran prior, locked the mutex but did not unlock it. This is why the print statements stop executing and printing on the console”.



```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
pthread_mutex_t mutex_1;

int counter;

void *child1(void *arg)
{
        while(1){
                pthread_mutex_lock(&mutex_1);
                sleep(1);
                if(counter > 25)
                        pthread_exit(NULL);
                else
                        counter++;
                pthread_mutex_unlock(&mutex_1);
                printf("Child1: counter=%d\n", counter);
        }
}


int main(void)
{
        pthread_t tid1;
        counter = 0;

        pthread_mutex_init(&mutex_1,NULL);
        pthread_create(&tid1,NULL,child1,NULL);
        do{
                pthread_mutex_lock(&mutex_1);
                sleep(1);
                counter++;
                pthread_mutex_unlock(&mutex_1);
                printf("Main: counter=%d\n", counter);

        }while(1);
        pthread_exit(0);
}
```



```
vrp@vrp-VirtualBox: ~/Downloads
Main: counter=11
Main: counter=12
Main: counter=13
Main: counter=14
Main: counter=15
Main: counter=16
Main: counter=17
Main: counter=18
Main: counter=19
Main: counter=20
Main: counter=21
Main: counter=22
Main: counter=23
Child1: counter=24
Child1: counter=25
Child1: counter=26
^C
vrp@vrp-VirtualBox:~/Downloads$ gcc assignment_multithreading.c -o assignment_mt -lpthread
vrp@vrp-VirtualBox:~/Downloads$ ./assignment_mt
Main: counter=1
Main: counter=2
Main: counter=3
Main: counter=4
Main: counter=5
Main: counter=6
Main: counter=7
Main: counter=8
Main: counter=9
Main: counter=10
Main: counter=11
Main: counter=12
Main: counter=13
Main: counter=14
Main: counter=15
Main: counter=16
Main: counter=17
Main: counter=18
Main: counter=19
Main: counter=20
Main: counter=21
Main: counter=22
Main: counter=23
Main: counter=24
Main: counter=25
Main: counter=26
Main: counter=27
^C
vrp@vrp-VirtualBox:~/Downloads$ 
```

"Sometimes the counter variable will increase more than 25 by never running the child thread, but when a child thread is run, value of the counter variable is greater than 25, thread exits before unlocking the mutex and the program execution stops (Shown above)".

2) The corrected version of the code that fixed the problem encountered earlier.

```
          void *child1(void *arg)
                    {
                              while(1){
                                                  pthread_mutex_lock(&mutex_1);
                                                  sleep(1);
                                                  if(counter > 25){
                              pthread_mutex_unlock(&mutex_1); //code for unlocking the
                                                                                      resources
                              pthread_exit(NULL); //exiting the resources after unlocking
                    }
                              else
                                        counter++;
                                        pthread_mutex_unlock(&mutex_1);
                                        printf("Child1: counter=%d¥n", counter);
                              }
                    }
```

"Whenever the counter value goes over the value 25 and the child thread is run, the code enters the "if" condition block and unlocks the mutex before exiting the thread. So when the other thread wants to access the counter variable, it has free resources to do so. Since we are killing the child thread the Parent thread Locks the mutex, increments the counter and unlocks the mutex. The parent thread continues infinitely till it is killed or interrupted by an external factor".