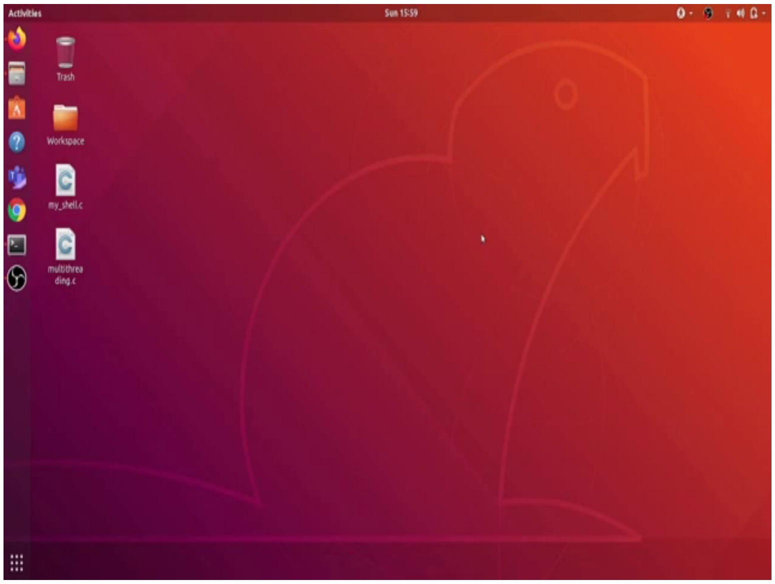
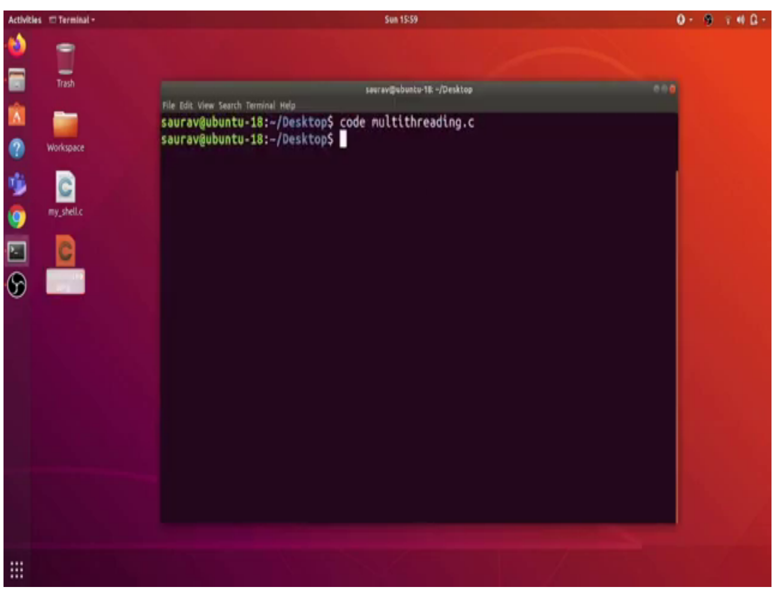
Design and Engineering of Computer Systems
Professor Mythilli Vutukuru
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Multithreading

(Refer Slide Time: 00:15)



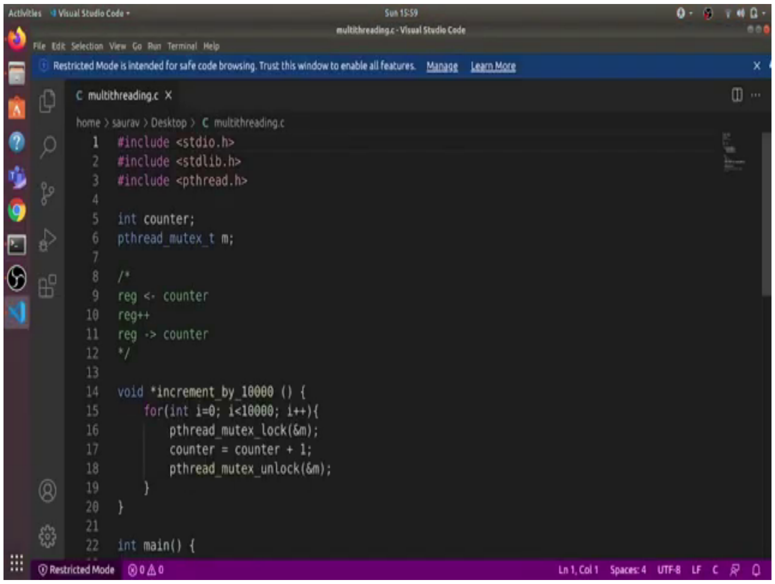
Hi everyone, in this video, we will learn about multi-threading, we will write the C programme, which spawns multiple threads and which access is a global counter in both the threads.

(Refer Slide Time: 00:33)



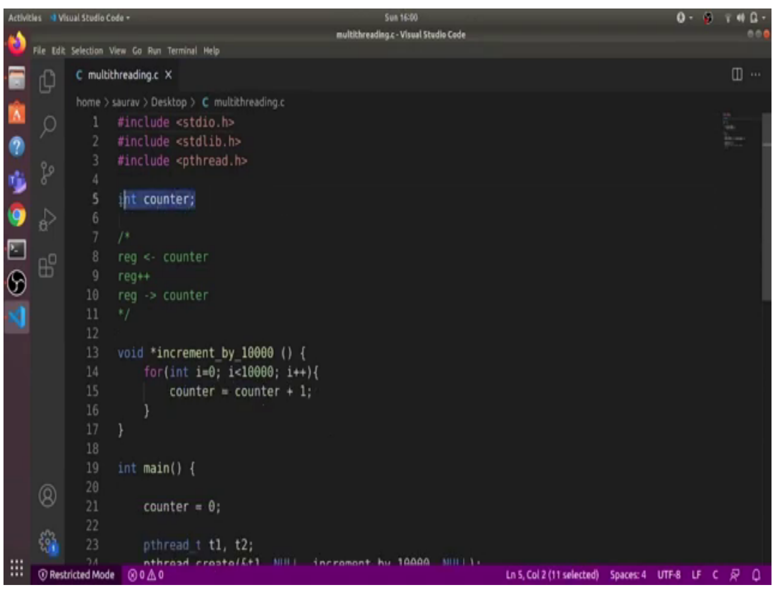
So, I have written this multi-threading in dot C programme. Let us, open it in visual code.

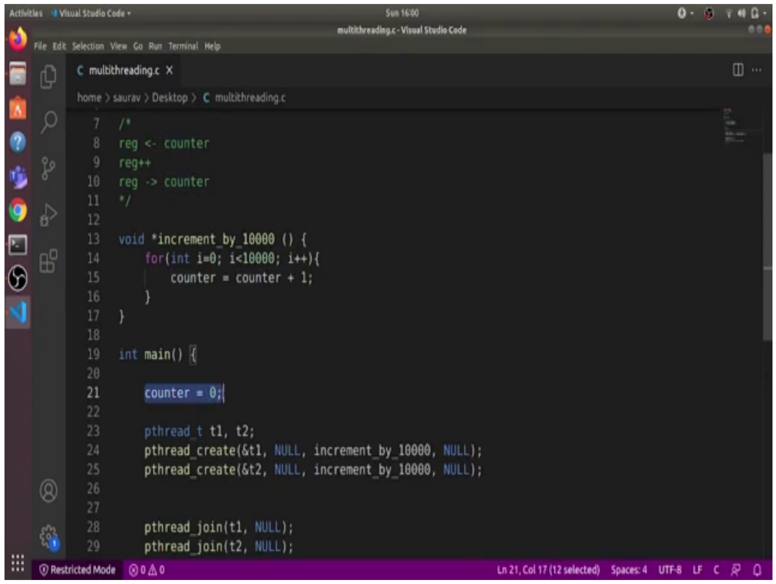
(Refer Slide Time: 00:37)

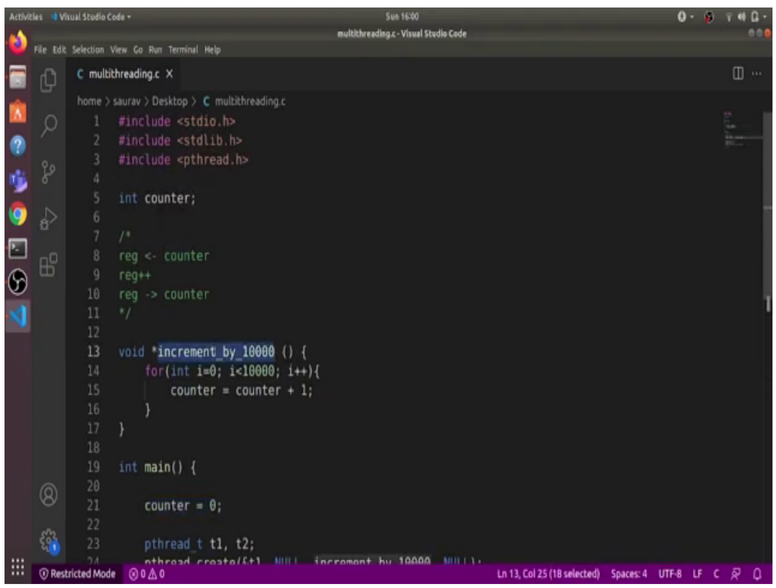


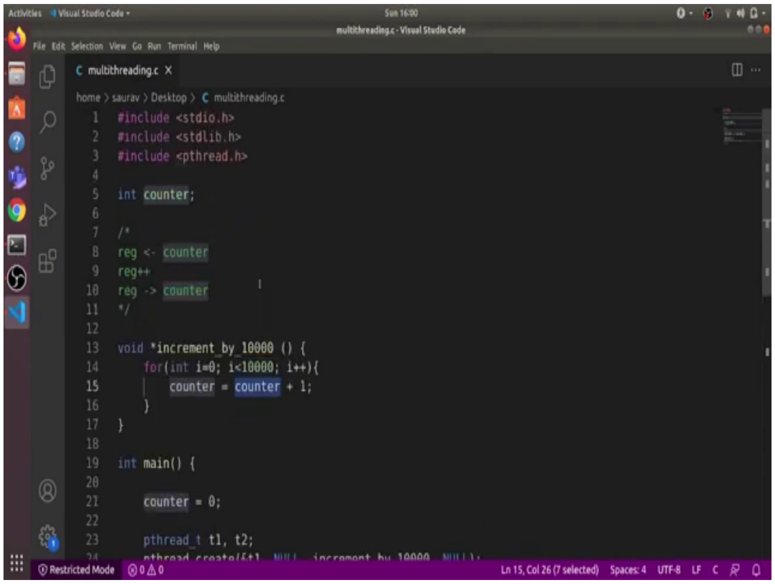
So, this is the multi-threading dot C programme.

(Refer Slide Time: 00:42)



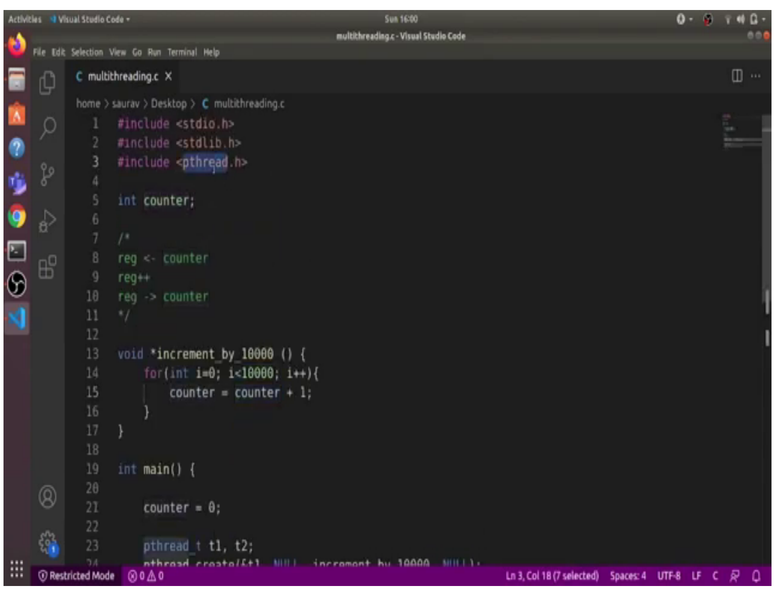

multithreading.c
home > saurav > Desktop > multithreading.c
/*
reg <- counter
reg++
reg -> counter
*/
void *increment_by_10000 () {
for(int i=0; i<10000; i++){
counter = counter + 1;
}
}
int main() {
counter = 0;
pthread_t t1, t2;
pthread_create(&t1, NULL, increment_by_10000, NULL);
pthread_create(&t2, NULL, increment_by_10000, NULL);
pthread_join(t1, NULL);
pthread_join(t2, NULL);
Restricted Mode
Ln 21, Col 17 (12 selected) Spaces: 4 UTF-8 LF C

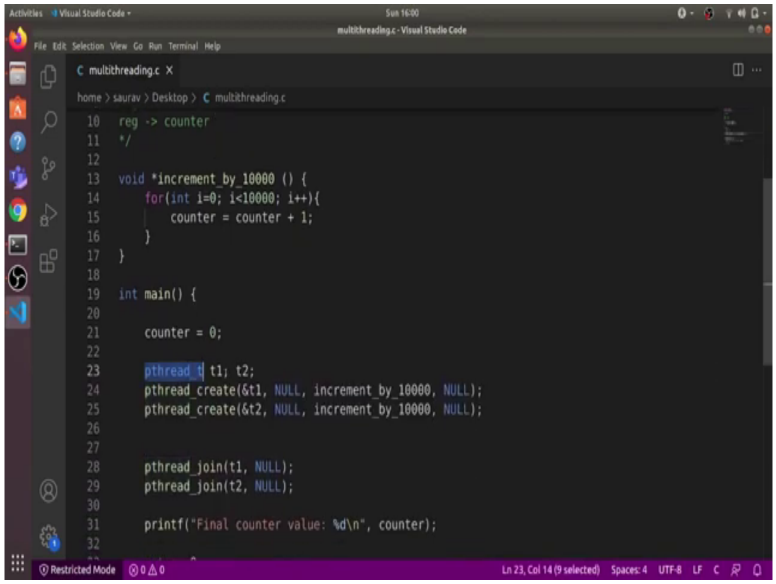

multithreading.c
home > saurav > Desktop > multithreading.c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
int counter;
/*
reg <- counter
reg++
reg -> counter
*/
void *increment_by_10000 () {
for(int i=0; i<10000; i++){
counter = counter + 1;
}
}
int main() {
counter = 0;
pthread_t t1, t2;
Restricted Mode
Ln 13, Col 25 (18 selected) Spaces: 4 UTF-8 LF C

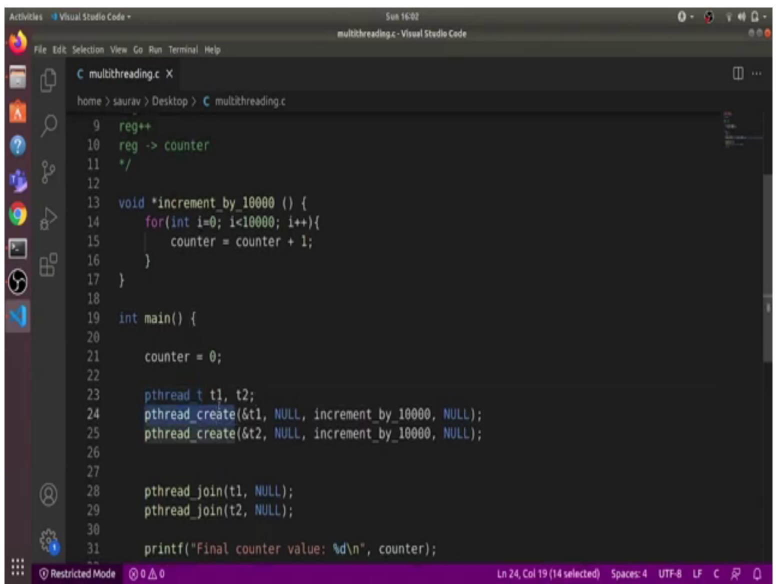


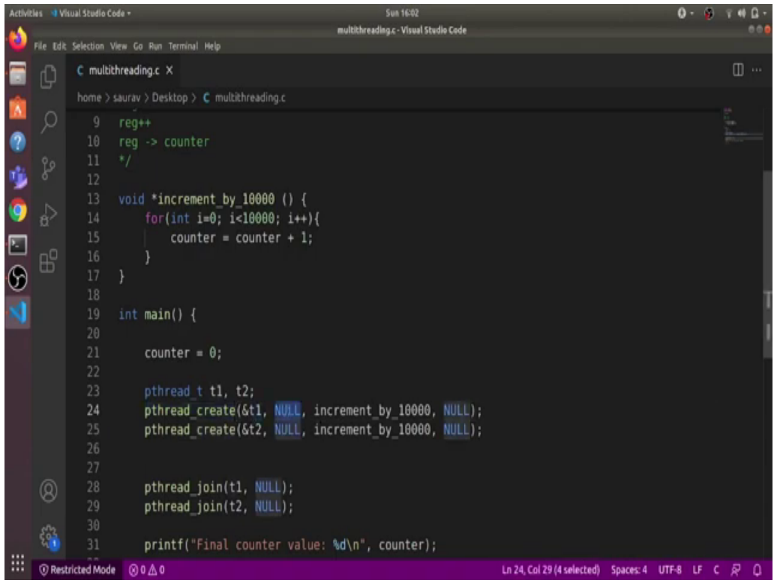
Let us, go over the code. We have this global counter variable and in the main function we initialise this counter variable to 0, then there is this increment by 10,000 function, which increments this counter variable by 10,000. So, it runs a loop for 10,000 times and each time it adds one to the counter variable.

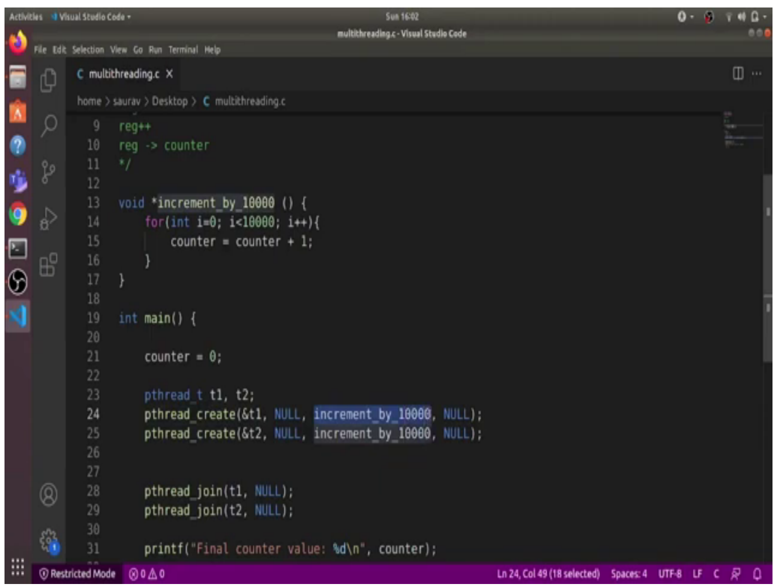
(Refer Slide Time: 01:03)

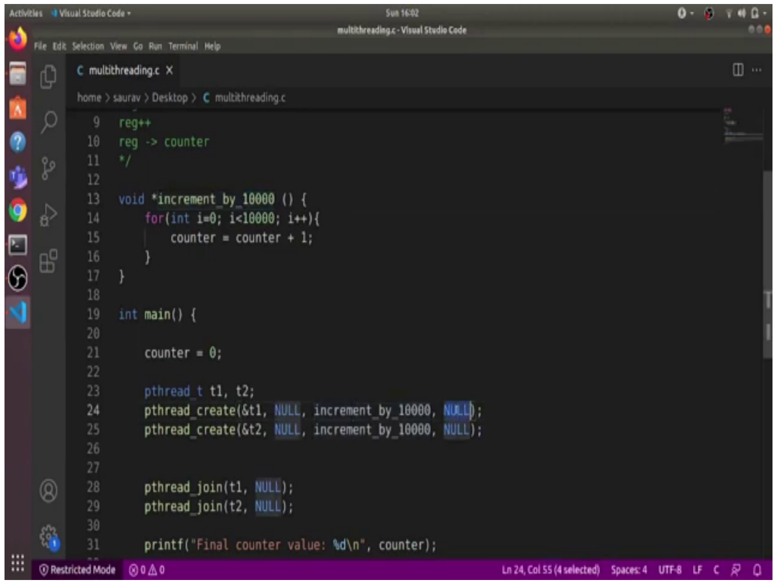



Activities
Visual Studio Code
multithreading.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
multithreading.c
home > saurav > Desktop > multithreading.c
reg -> counter
*/
void *increment_by_10000 () {
for(int i=0; i<10000; i++){
counter = counter + 1;
}
}
int main() {
counter = 0;
pthread_t t1; t2;
pthread_create(&t1, NULL, increment_by_10000, NULL);
pthread_create(&t2, NULL, increment_by_10000, NULL);
pthread_join(t1, NULL);
pthread_join(t2, NULL);
printf("Final counter value: %d\n", counter);
Restricted Mode
Ln 23, Col 14 (9 selected) Spaces: 4 UTF-8 LF C


Activities
Visual Studio Code
multithreading.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
multithreading.c
home > saurav > Desktop > multithreading.c
reg++
reg -> counter
*/
void *increment_by_10000 () {
for(int i=0; i<10000; i++){
counter = counter + 1;
}
}
int main() {
counter = 0;
pthread_t t1, t2;
pthread_create(&t1, NULL, increment_by_10000, NULL);
pthread_create(&t2, NULL, increment_by_10000, NULL);
pthread_join(t1, NULL);
pthread_join(t2, NULL);
printf("Final counter value: %d\n", counter);
Restricted Mode
Ln 24, Col 19 (14 selected) Spaces: 4 UTF-8 LF C

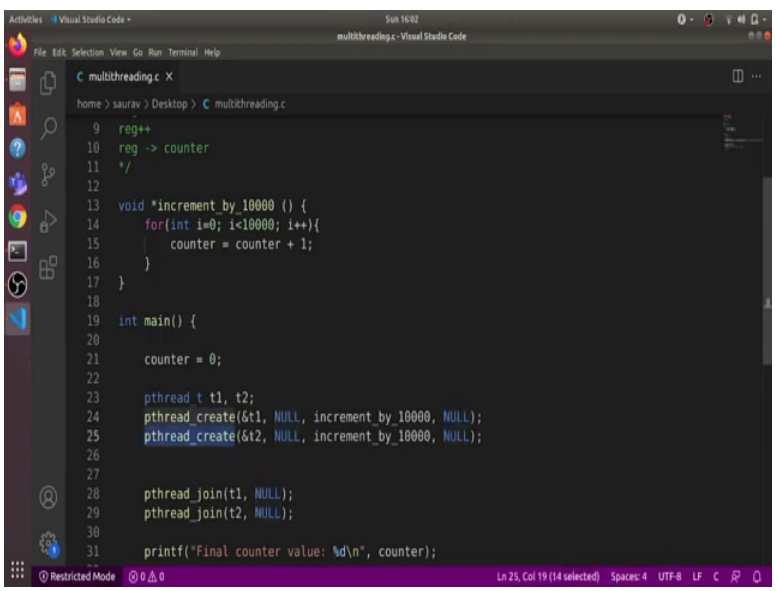

Activities
Visual Studio Code
Sun 16:02
multithreading.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
multithreading.c
home > saurav > Desktop > multithreading.c
reg++
reg -> counter
*/
void *increment_by_10000 () {
for(int i=0; i<10000; i++){
counter = counter + 1;
}
}
int main() {
counter = 0;
pthread_t t1, t2;
pthread_create(&t1, NULL, increment_by_10000, NULL);
pthread_create(&t2, NULL, increment_by_10000, NULL);
pthread_join(t1, NULL);
pthread_join(t2, NULL);
printf("Final counter value: %d\n", counter);
Restricted Mode
Ln 24, Col 29 (4 selected) Spaces: 4 UTF-8 LF C

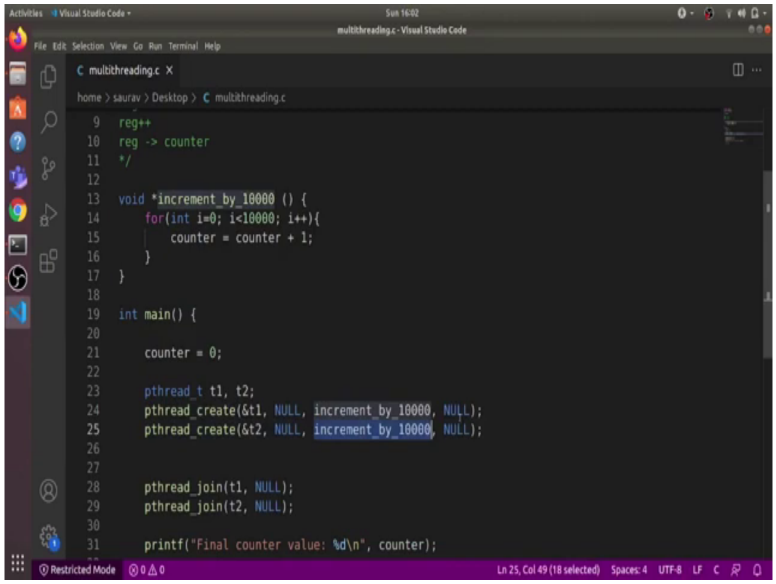

Activities
Visual Studio Code
Sun 16:02
multithreading.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
multithreading.c
home > saurav > Desktop > multithreading.c
reg++
reg -> counter
*/
void *increment_by_10000 () {
for(int i=0; i<10000; i++){
counter = counter + 1;
}
}
int main() {
counter = 0;
pthread_t t1, t2;
pthread_create(&t1, NULL, increment_by_10000, NULL);
pthread_create(&t2, NULL, increment_by_10000, NULL);
pthread_join(t1, NULL);
pthread_join(t2, NULL);
printf("Final counter value: %d\n", counter);
Restricted Mode
Ln 24, Col 49 (18 selected) Spaces: 4 UTF-8 LF C

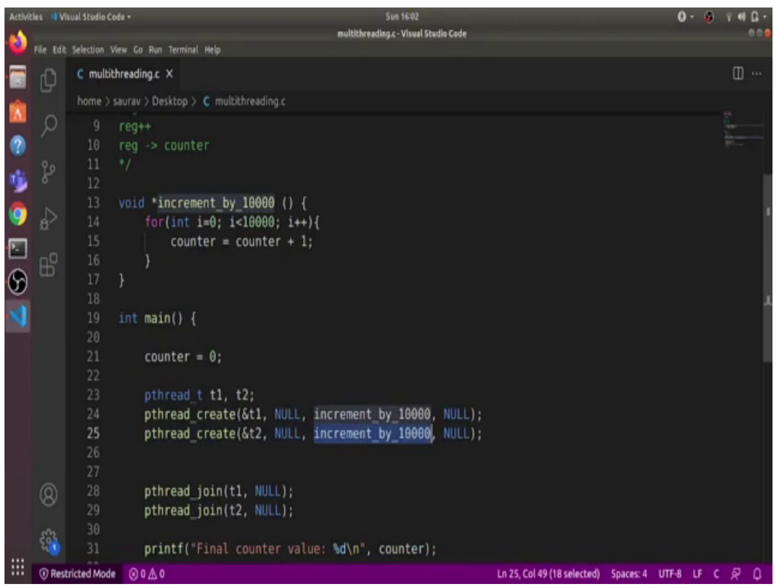


So here we use pthread library, which is the POSIX threads library, which allows us to create multiple threads, we first declare two threads, t1 and t2. So, we use the pthread_create function to create the threads. So, first you create t1 thread and we do not specify any attribute. And we give it increment by 10,000 function so that this thread runs this increment by 10,000 function. And because increment by 10,000, does not take any argument, so we specified null here.

(Refer Slide Time: 01:34)

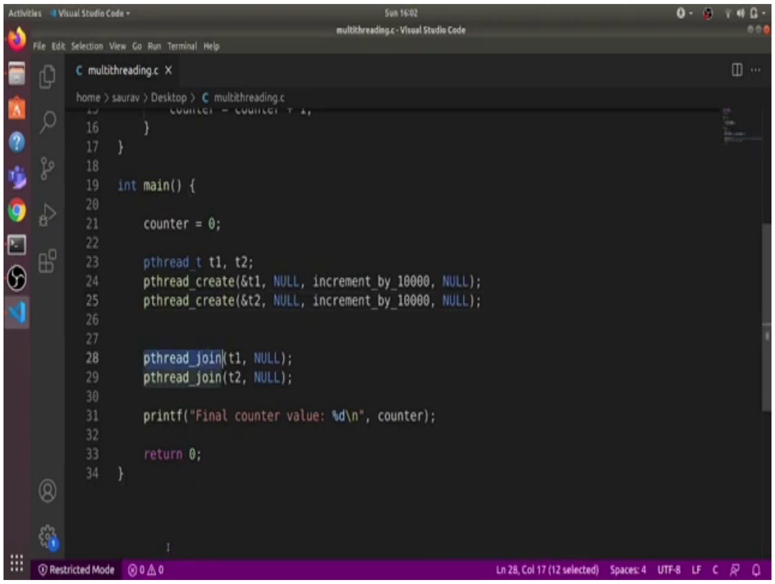


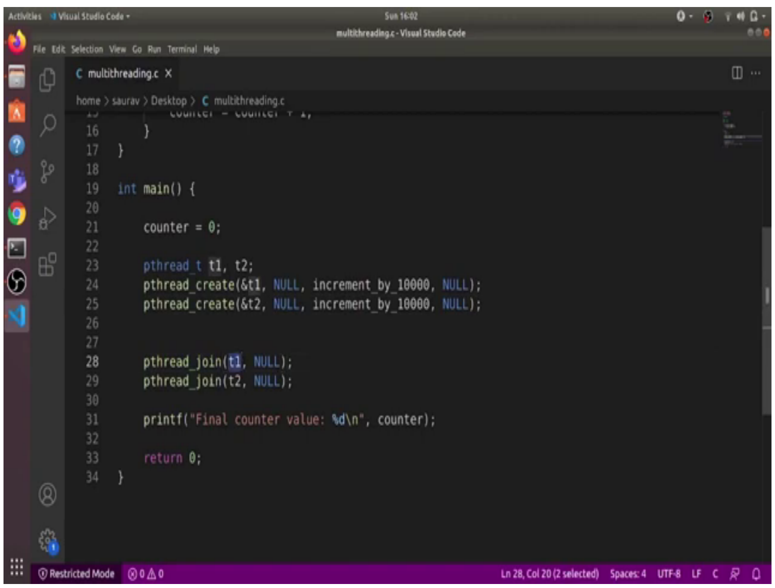


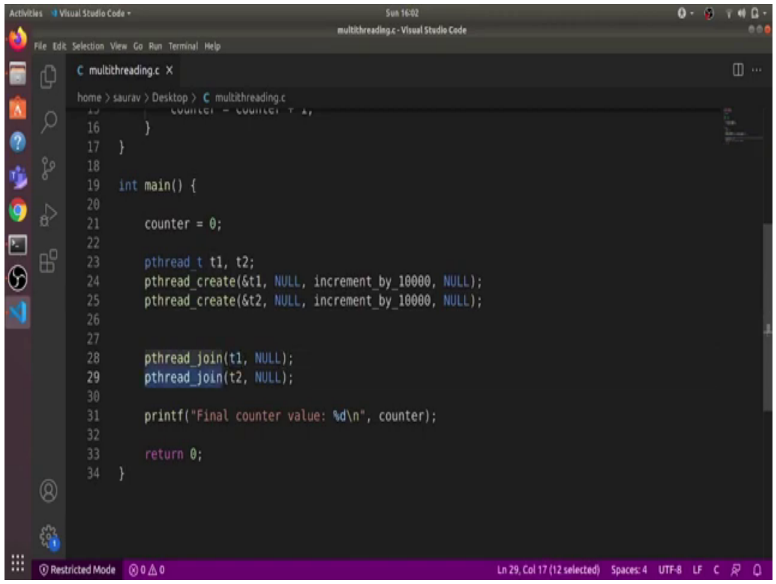


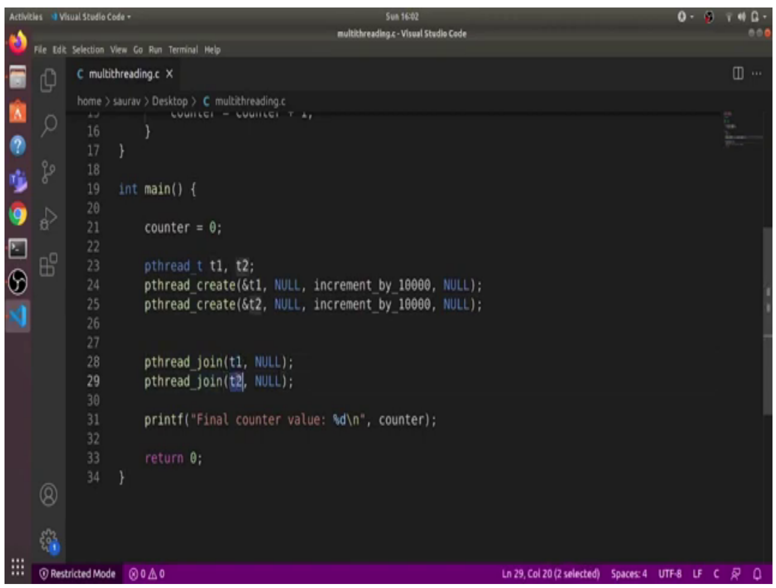
Then we created t2 thread again using the same increment were 10,000 function and again, we do not specify any arguments.

(Refer Slide Time: 01:43)



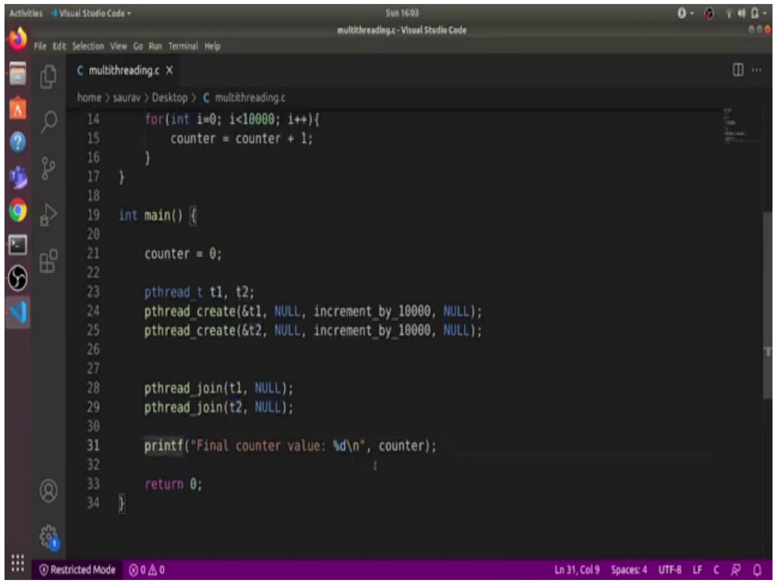


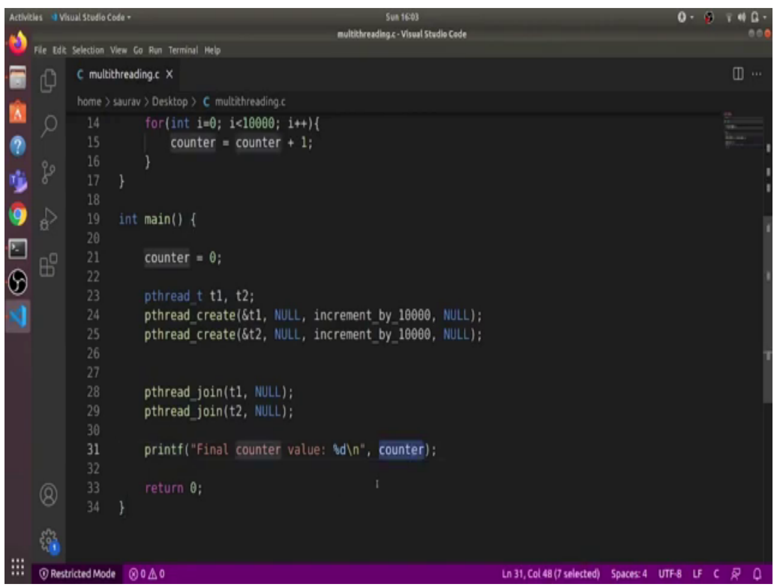




Then we use this pthread_joint function so that the parent thread waits for thread t1 to complete. And similarly, it waits for the thread t2 complete.

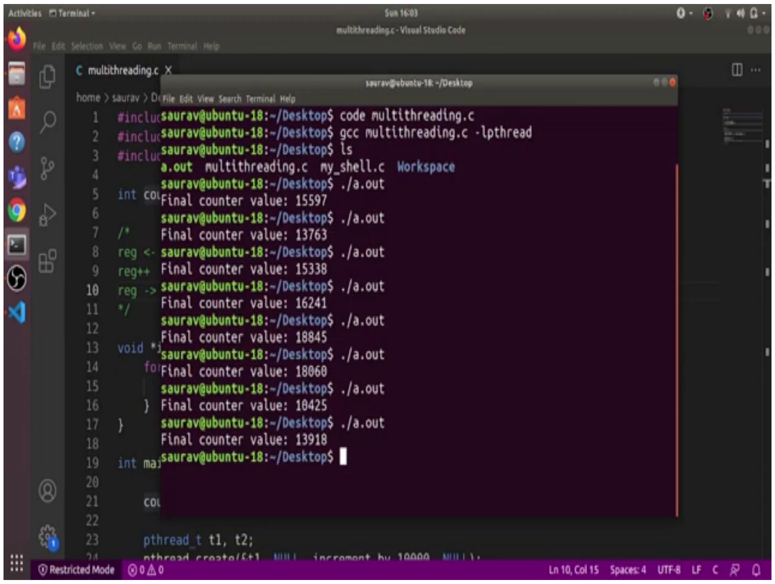
(Refer Slide Time: 01:57)





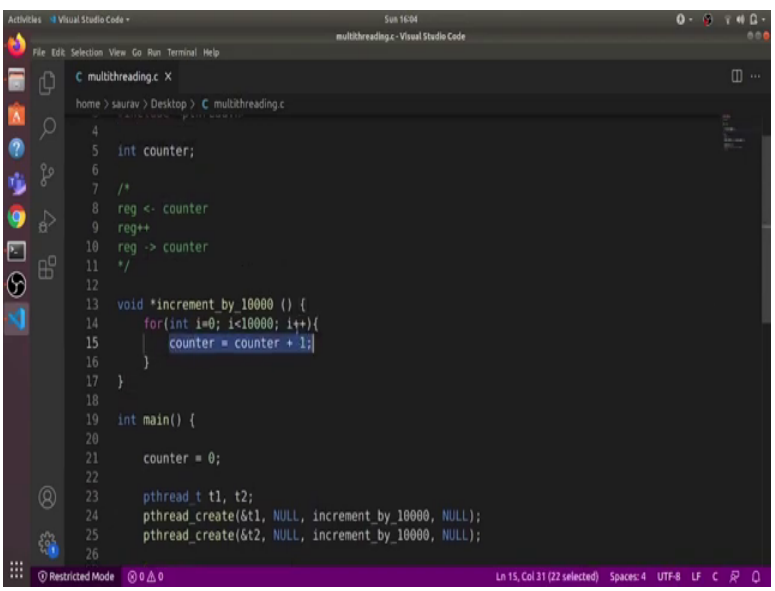
So, once both threads have finished execution, then the parent thread prints the final counter value. So, we expect that because we are incrementing, this counter by 10000 two times. So, the final counter value should be 20,000.

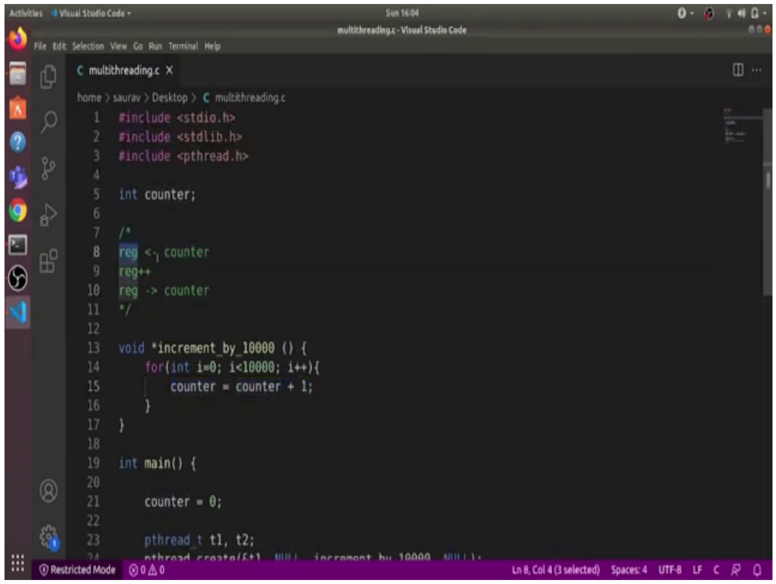
(Refer Slide Time: 02:12)

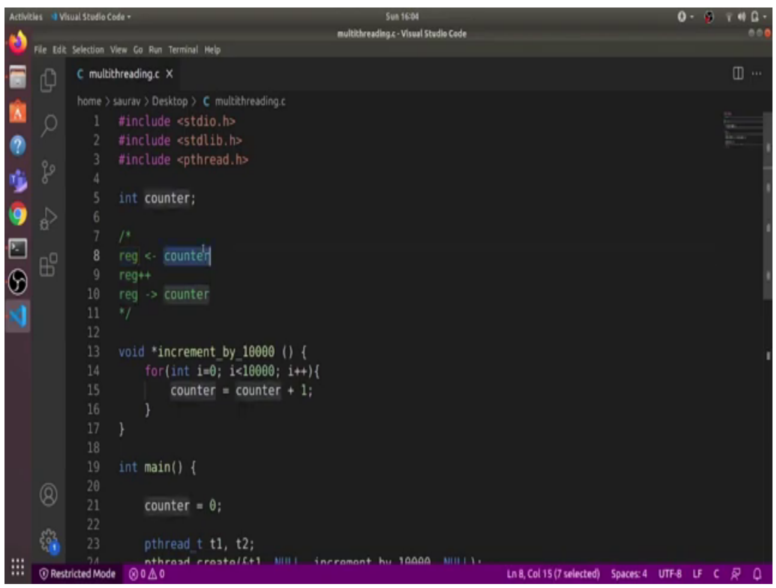


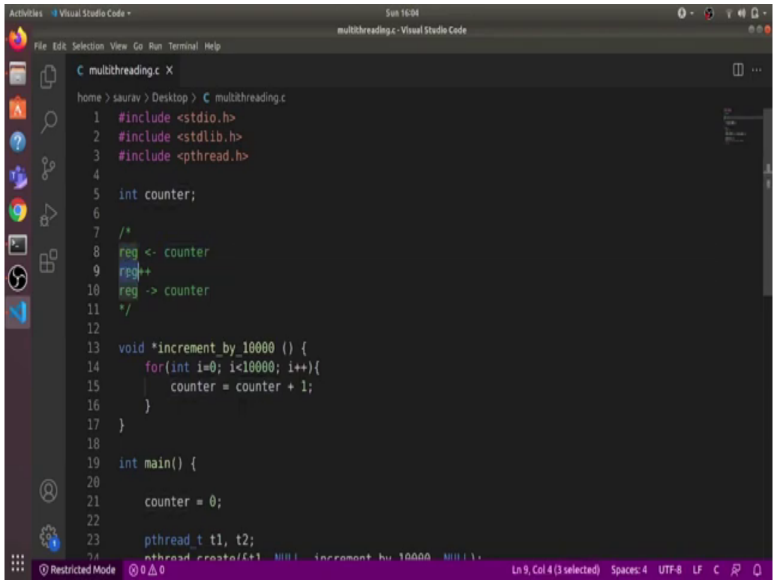
So, let us compile this programme and run it. So, I will open that terminal and compile it using gcc. Because we are using the pthread library, so we need to specify that while compiling. So this has created this a dot out file. Let us, run this a dot out. So, you can see that the counter value is less than 20,000. Let us, run it a couple of more times. So, every time we get some different value and which is less than 20,000. So, why is that so? This is because there is a race condition in the code.

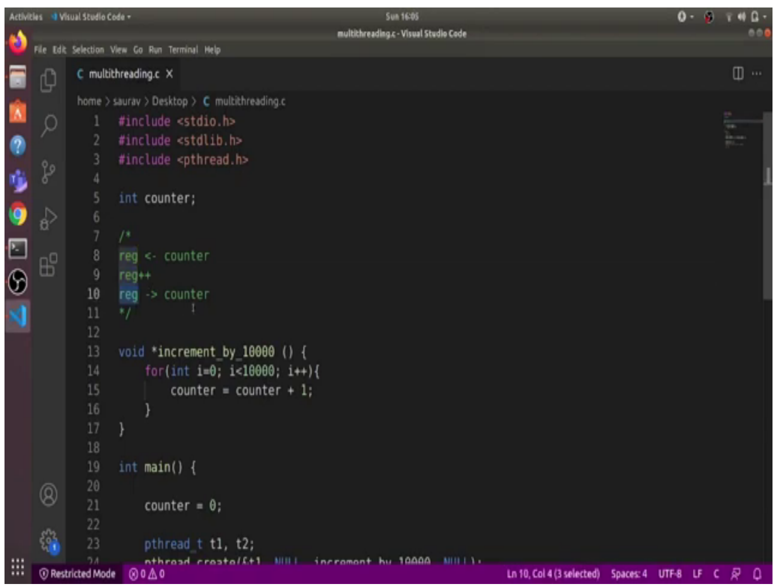
(Refer Slide Time: 02:44)

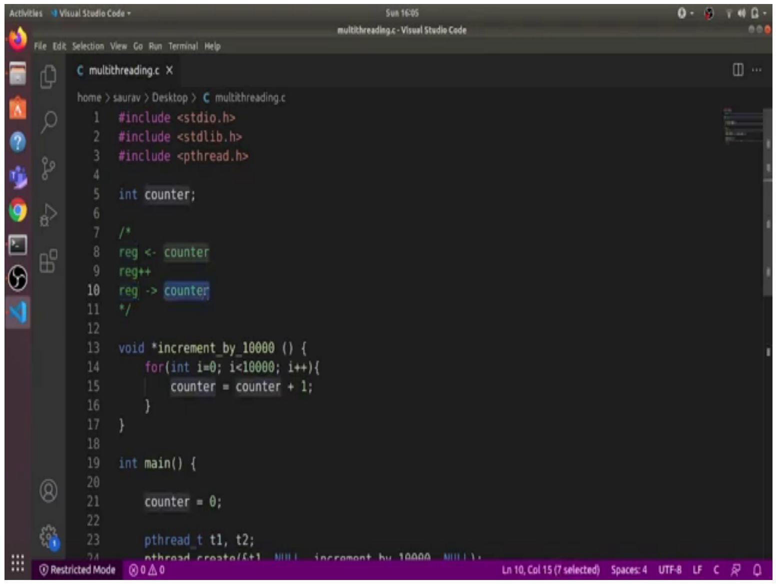



Activities
Visual Studio Code
Sun 16:04
multithreading.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
multithreading.c
home > saurav > Desktop > multithreading.c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
int counter;
/*
reg <- counter
reg++
reg -> counter
*/
void *increment_by_10000 () {
for(int i=0; i<10000; i++){
counter = counter + 1;
}
}
int main() {
counter = 0;
pthread_t t1, t2;
Restricted Mode
Ln 8, Col 4 (3 selected) Spaces: 4 UTF-8 LF C


Activities
Visual Studio Code
Sun 16:04
multithreading.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
multithreading.c
home > saurav > Desktop > multithreading.c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
int counter;
/*
reg <- counter
reg++
reg -> counter
*/
void *increment_by_10000 () {
for(int i=0; i<10000; i++){
counter = counter + 1;
}
}
int main() {
counter = 0;
pthread_t t1, t2;
Restricted Mode
Ln 8, Col 15 (7 selected) Spaces: 4 UTF-8 LF C

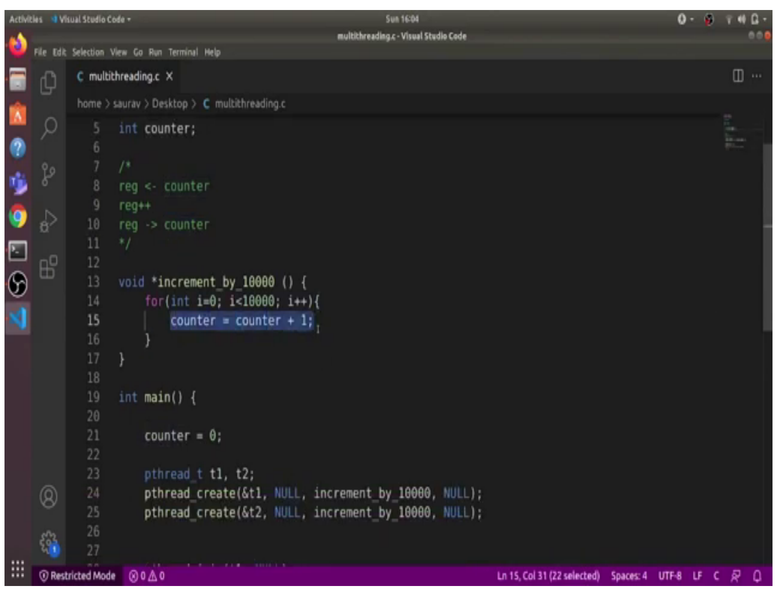

Activities
Visual Studio Code
Sun 16:04
multithreading.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
multithreading.c
home > saurav > Desktop > multithreading.c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
int counter;
/*
reg <- counter
reg++
reg -> counter
*/
void *increment_by_10000 () {
for(int i=0; i<10000; i++){
counter = counter + 1;
}
}
int main() {
counter = 0;
pthread_t t1, t2;
Restricted Mode
Ln 9, Col 4 (3 selected)
Spaces: 4
UTF-8
LF
C

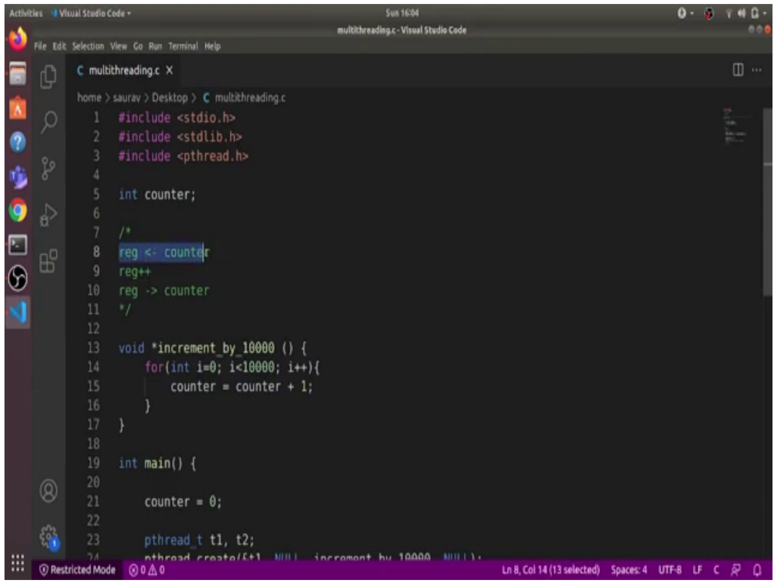

Activities
Visual Studio Code
Sun 16:05
multithreading.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
multithreading.c
home > saurav > Desktop > multithreading.c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
int counter;
/*
reg <- counter
reg++
reg -> counter
*/
void *increment_by_10000 () {
for(int i=0; i<10000; i++){
counter = counter + 1;
}
}
int main() {
counter = 0;
pthread_t t1, t2;
Restricted Mode
Ln 10, Col 4 (3 selected)
Spaces: 4
UTF-8
LF
C

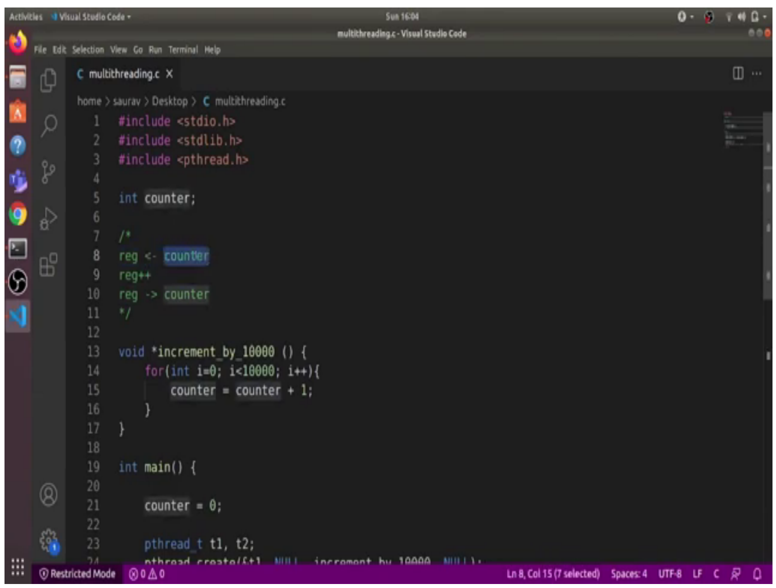


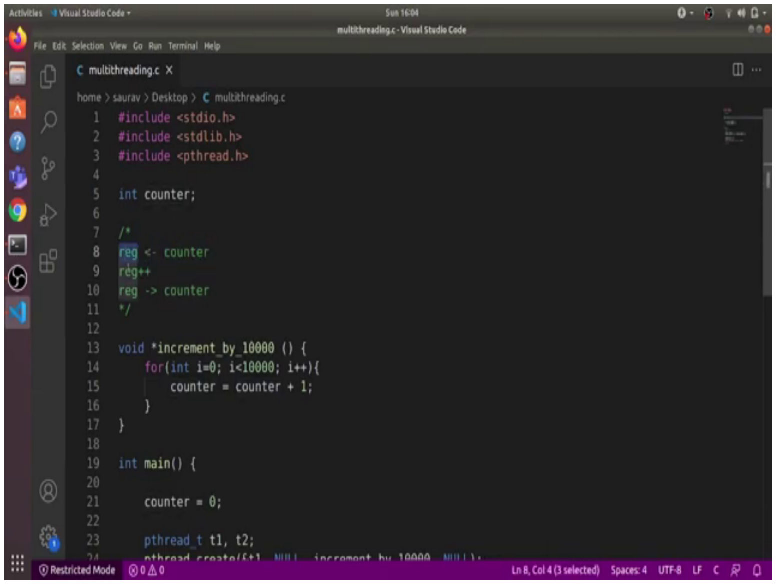
And what do I mean by race condition. So, there is this line in the code which mentions counter = counter + 1. So, when it is translated to assembly, then there are three instructions, it first loads this counter value in a register, then it increments the value in the register by 1. And finally it again stores back the register value to the counter variable.

(Refer Slide Time: 03:12)



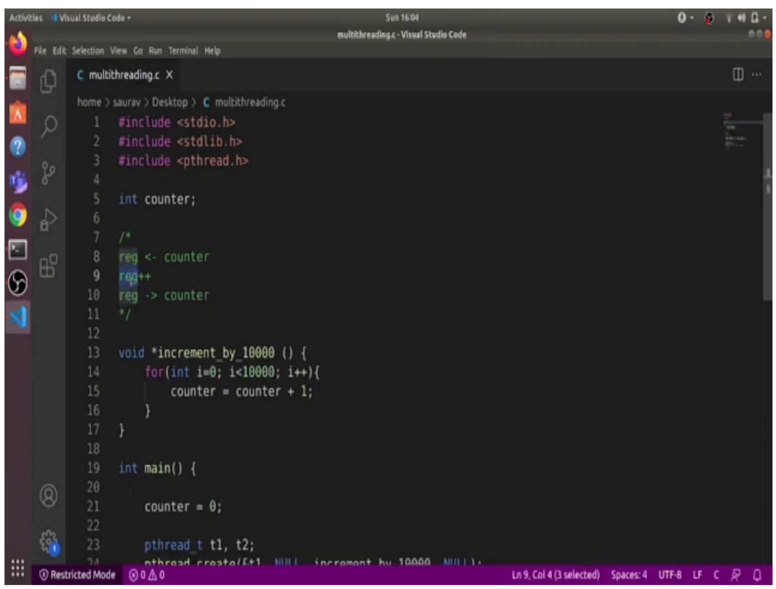

Activities
Visual Studio Code
Sun 16:04
multithreading.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
multithreading.c
home > saurav > Desktop > multithreading.c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
int counter;
/*
reg <- counter
reg++
reg -> counter
*/
void *increment_by_10000 () {
for(int i=0; i<10000; i++){
counter = counter + 1;
}
}
int main() {
counter = 0;
pthread_t t1, t2;
Restricted Mode
Ln 8, Col 14 (13 selected)
Spaces: 4
UTF-8
LF
C

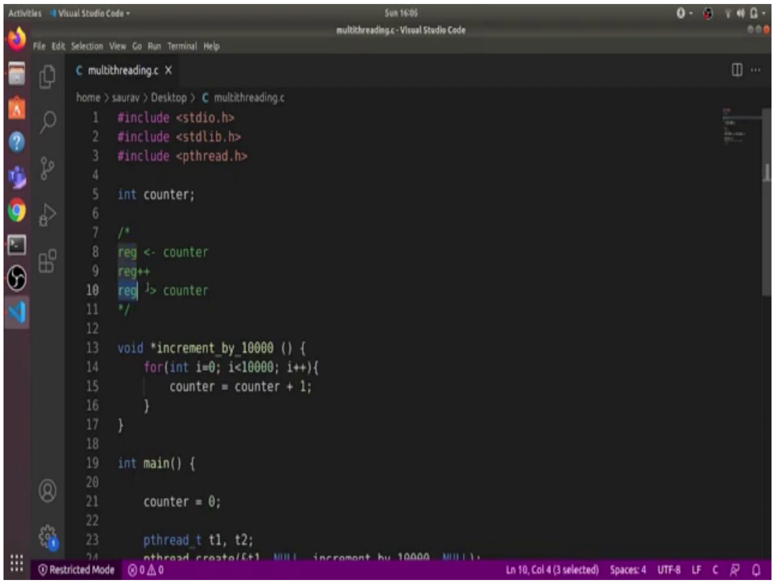

Activities
Visual Studio Code
Sun 16:04
multithreading.c - Visual Studio Code
File Edit Selection View Go Run Terminal Help
multithreading.c
home > saurav > Desktop > multithreading.c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
int counter;
/*
reg <- counter
reg++
reg -> counter
*/
void *increment_by_10000 () {
for(int i=0; i<10000; i++){
counter = counter + 1;
}
}
int main() {
counter = 0;
pthread_t t1, t2;
Restricted Mode
Ln 8, Col 15 (7 selected)
Spaces: 4
UTF-8
LF
C

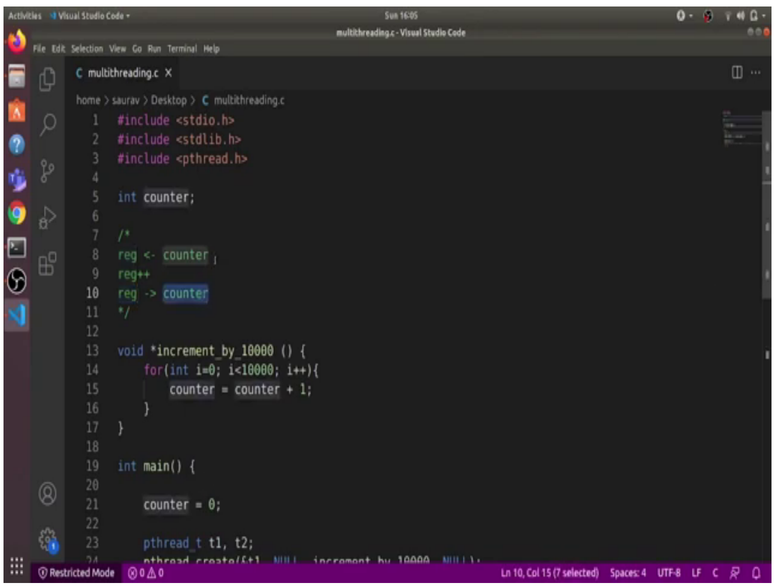


Because there are two threads which are accessing this part of code concurrently. So, it might happen that first thread execute this instruction, it loads the counter value in the register.

(Refer Slide Time: 03:19)

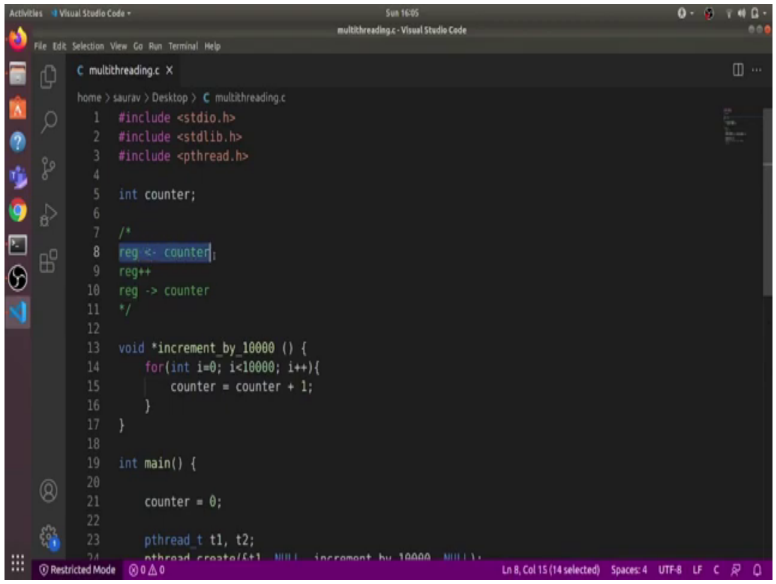


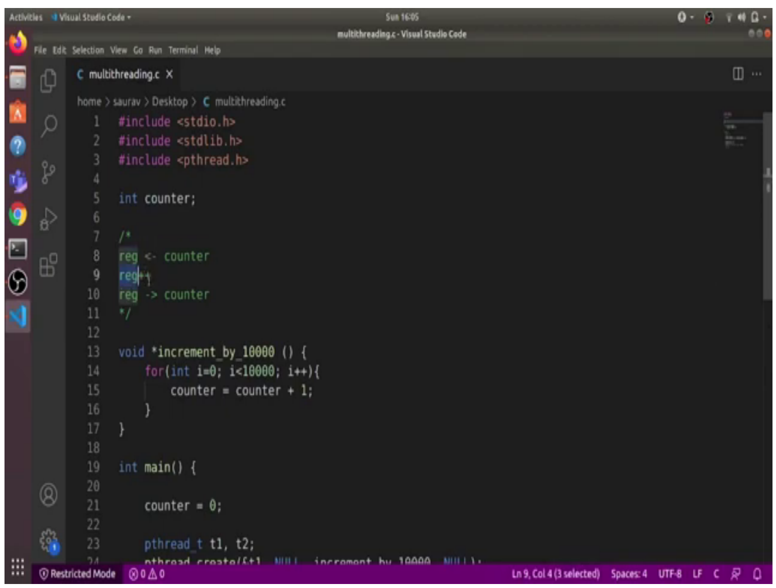


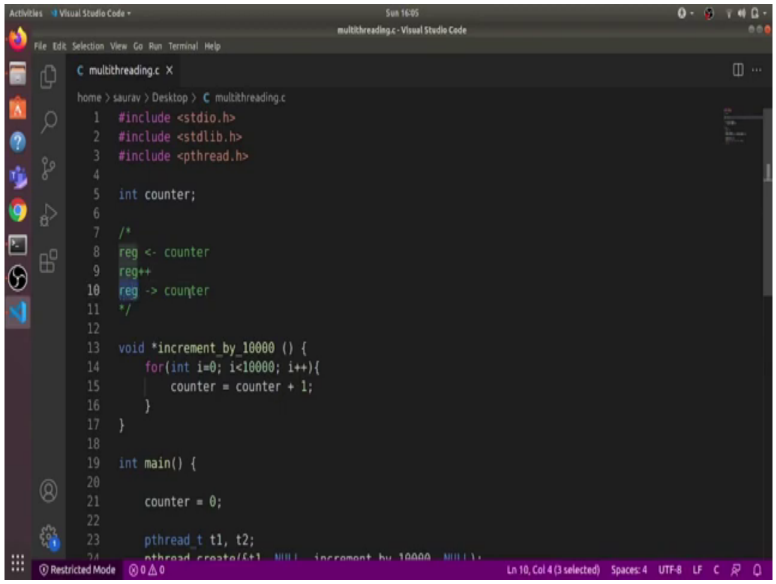


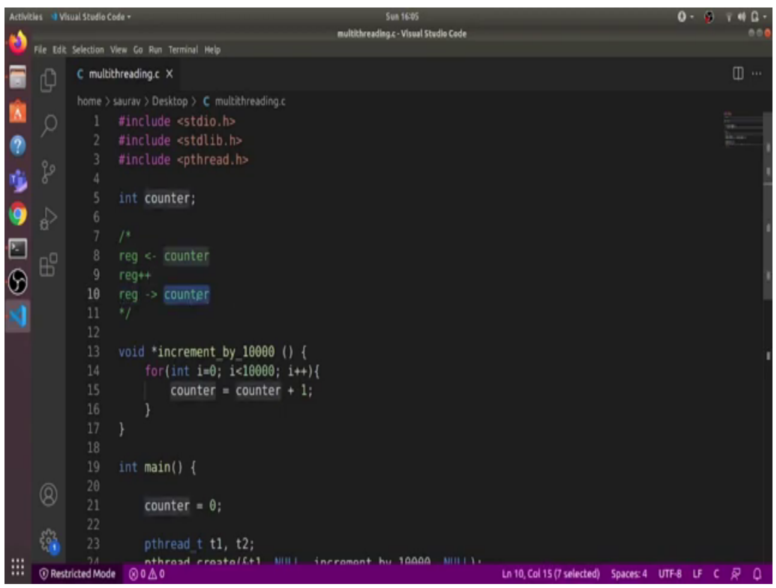
It increments the register value by 1. But before it stores back the counter value back to the counter variable.

(Refer Slide Time: 03:30)



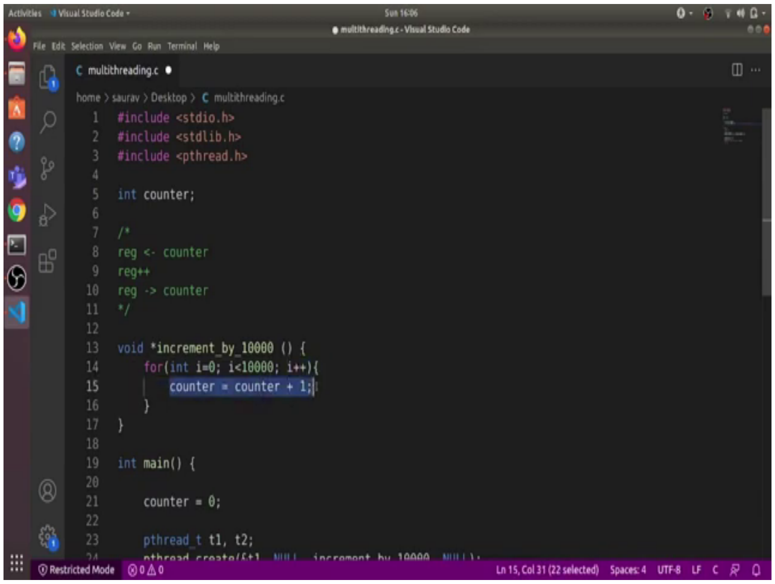






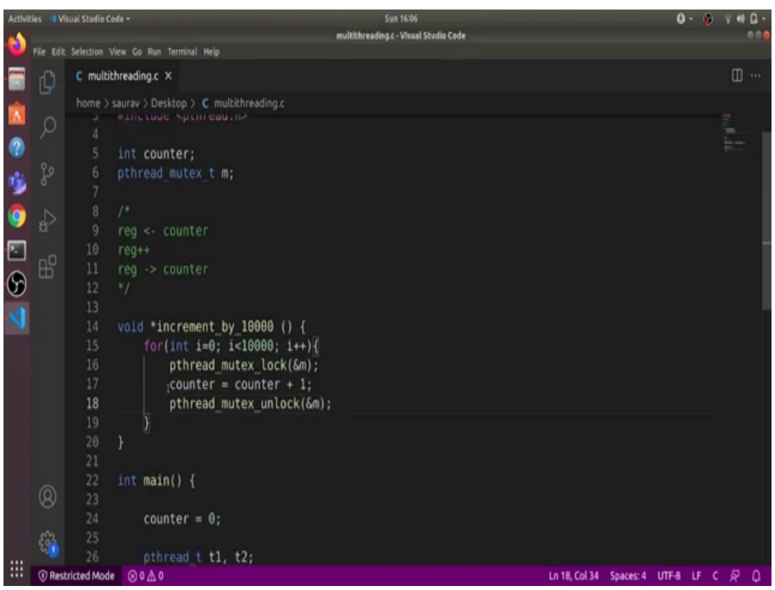
The another thread executes this instruction. So, it takes in the previous value, it also incremented by 1. And finally, both the threads store the same value to the counter variable. In essence, instead of increasing the counter value by two the execution just in visit the counter value by 1. So, how do we avoid this reg condition?

(Refer Slide Time: 03:55)



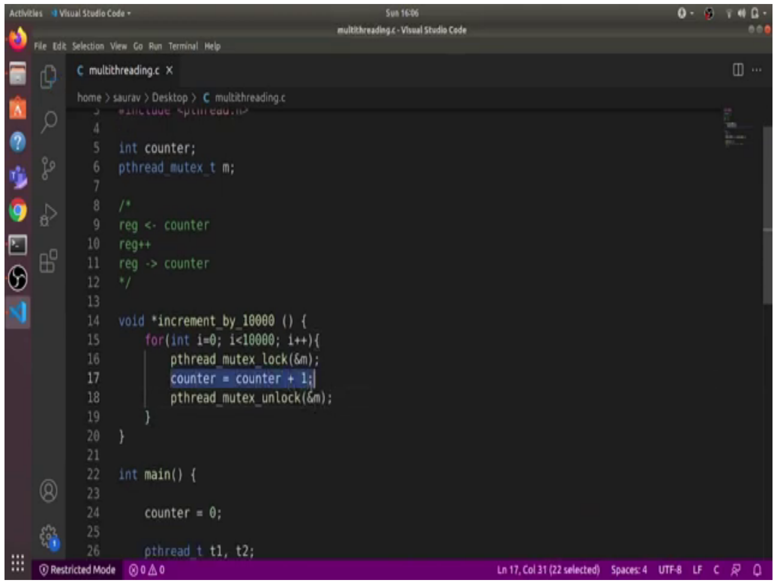
So, you can get rid of this race condition using locks, we will lock the critical section of the code which is this line so that only one thread can execute it at a time. So first, we will declare a lock pthread_mutex_t and we will call this lock m. So, this mutex stands for mutual exclusion, because we want mutual exclusion for this part of code.

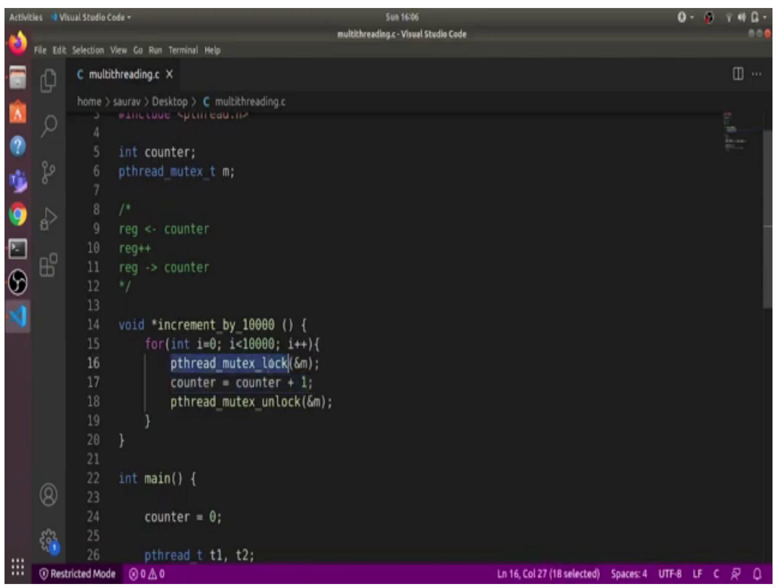
(Refer Slide Time: 04:16)

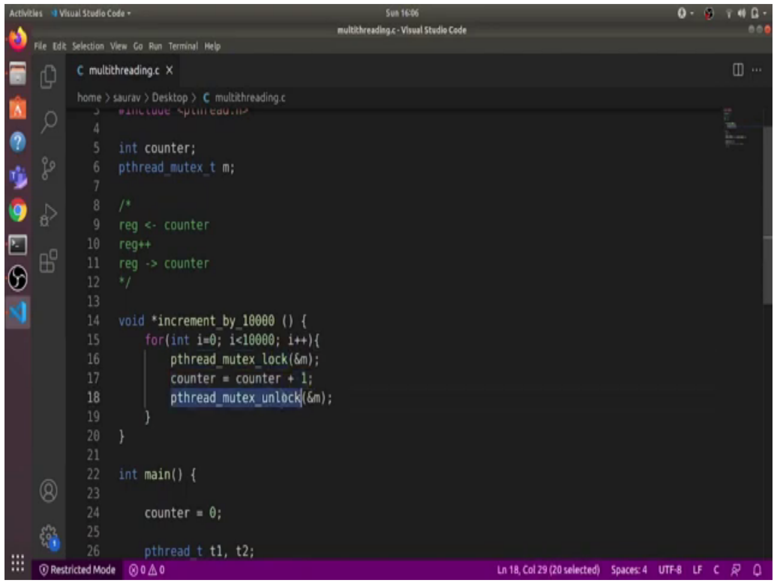


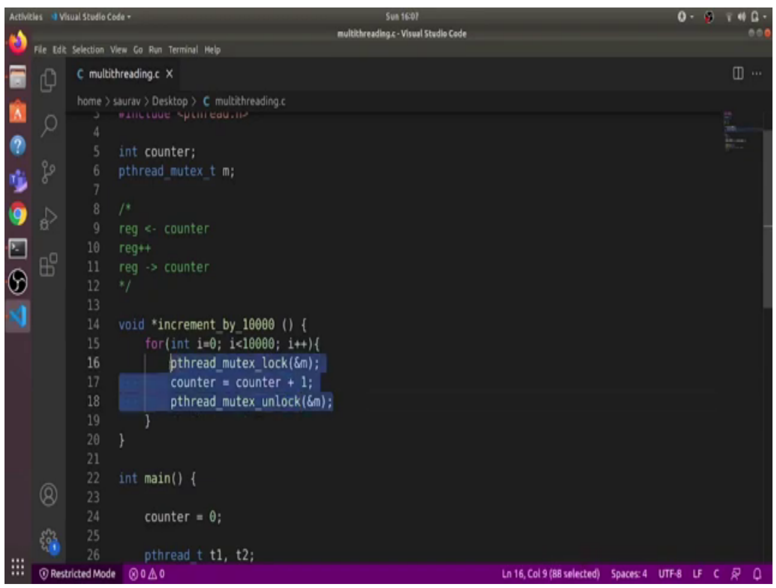
So, we acquire lock before entering the critical section, use pthread_mutex_lock and we give it a pointer to m. Then, after executing the counter = counter + 1, we unlock it so that another thread can access the lock and run this part of code. So, we use pthread_mutex_unlock(&m).

(Refer Slide Time: 04:46)



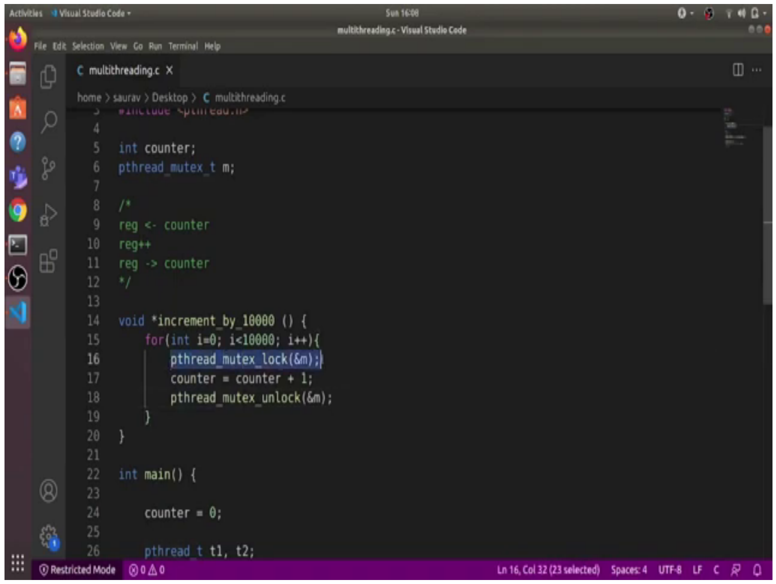


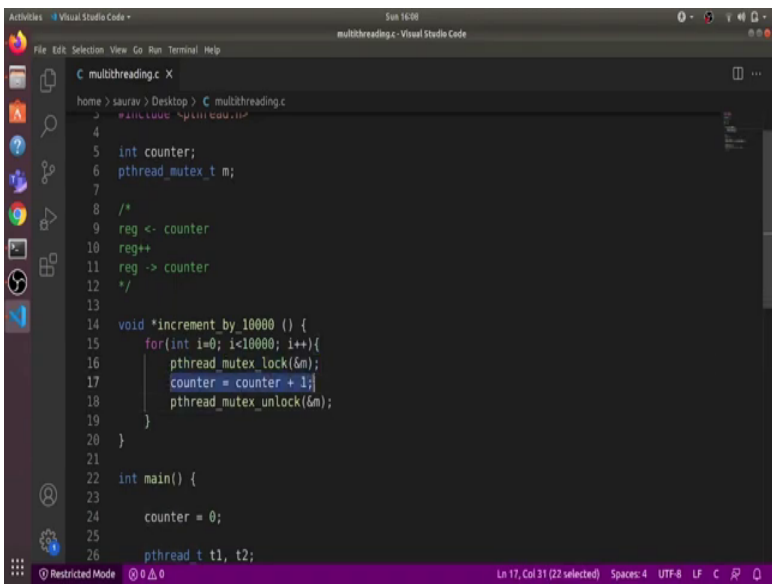


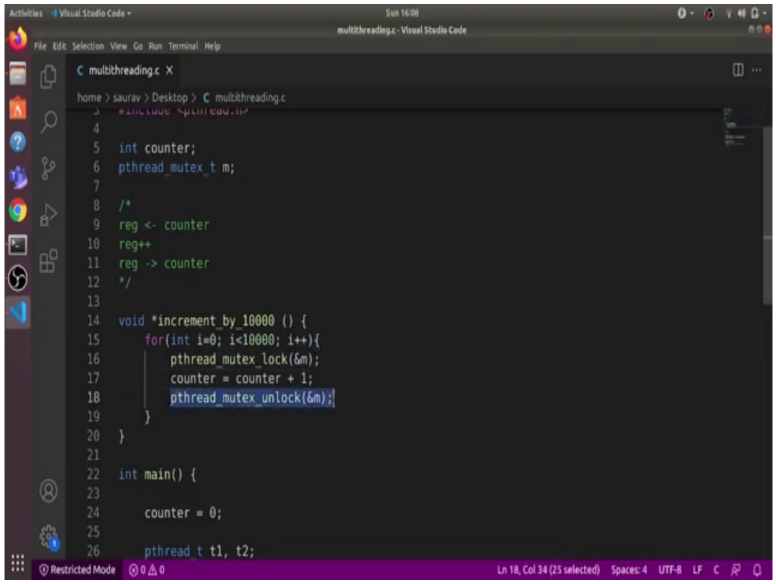


So, now before entering the critical section, each of the thread will have to acquire the lock and then they will release it after exiting from the critical section. So, only one thread will be able to execute this part of code.

(Refer Slide Time: 04:58)

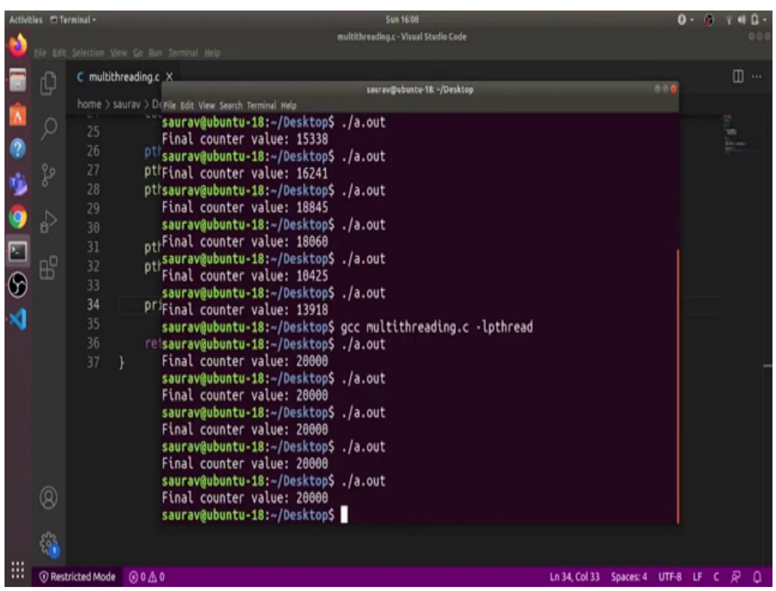






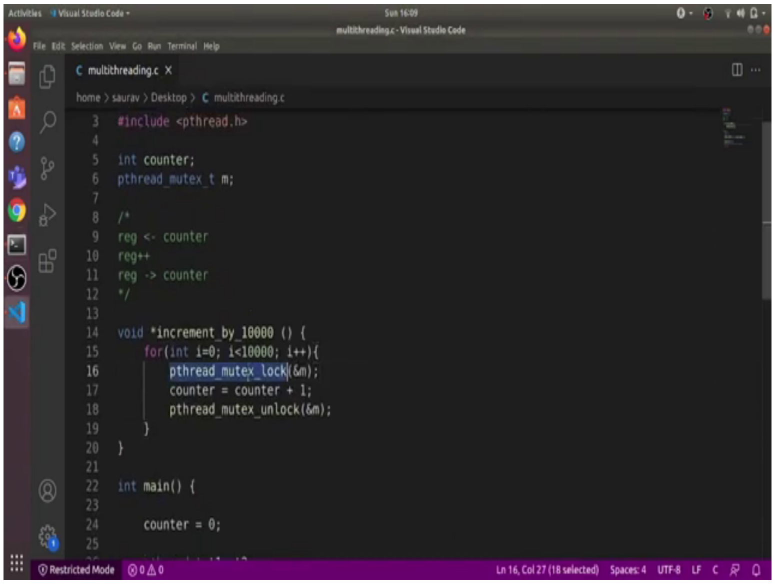
Now, what will happen is first third will acquire the look then when it is executing counter = counter + 1. Even if the second thread reaches this particular part of code, it will be unable to acquire the lock because first thread already has the lock and only when first thread releases the lock after this particular code, will the second thread be able to acquire it and execute counter = counter + 1. So, this is how we can get rid of the race condition.

(Refer Slide Time: 05:25)



So, let us compile and run it again. So, I will open a terminal and compile it again with the thread library. Alright, so now you see we are getting 20,000 every time we run the code.

(Refer Slide Time: 05:41)



So, it is very important to use this mutual exclusion locks whenever you are accessing a global variable in case of multi threading programme. So, that is it for this video. Thanks and have a nice day.