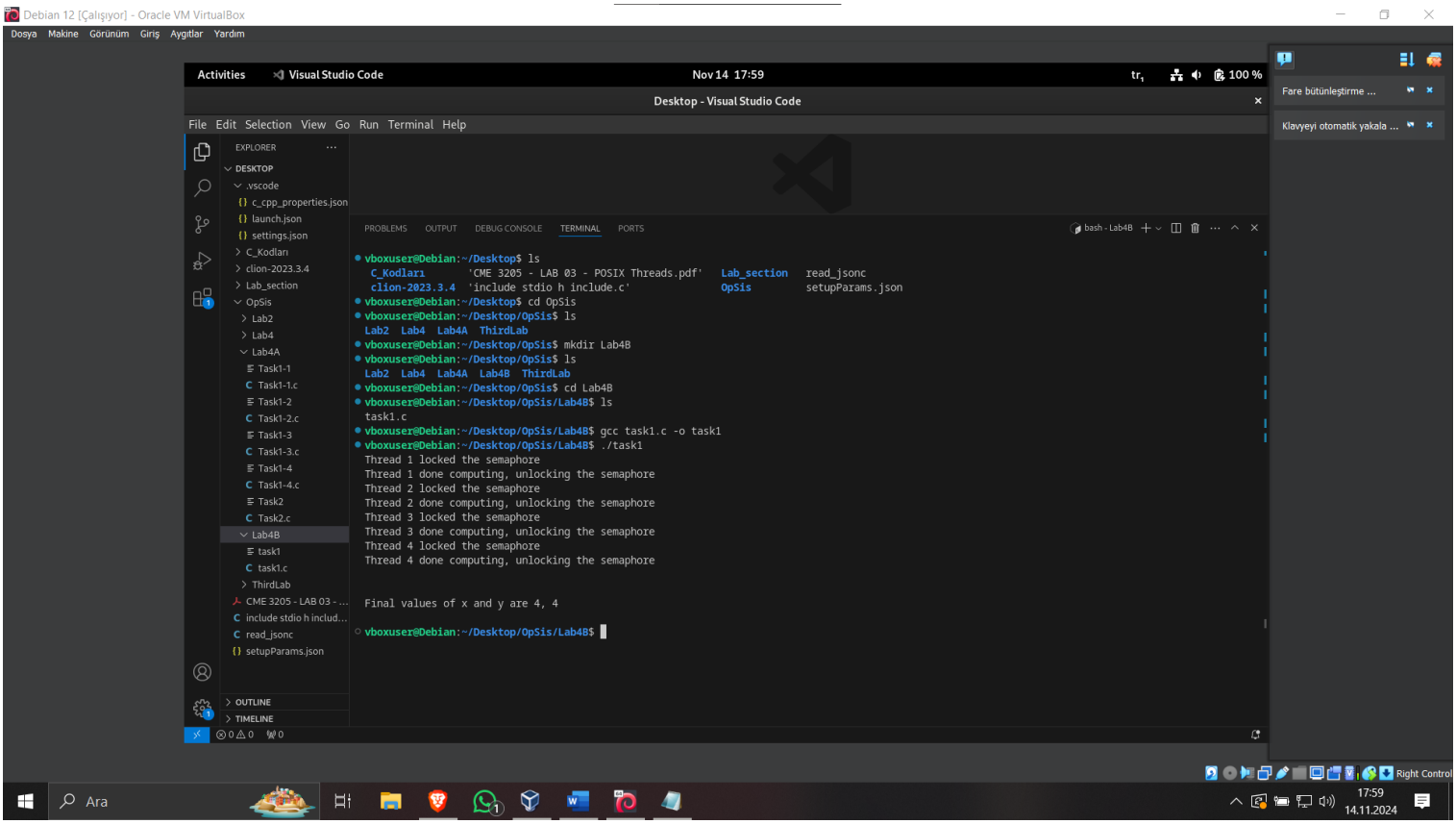
# 2021510010 – Çağrı AYDIN – Lab 04 Part B

## TASK 1:

ScreenShot:

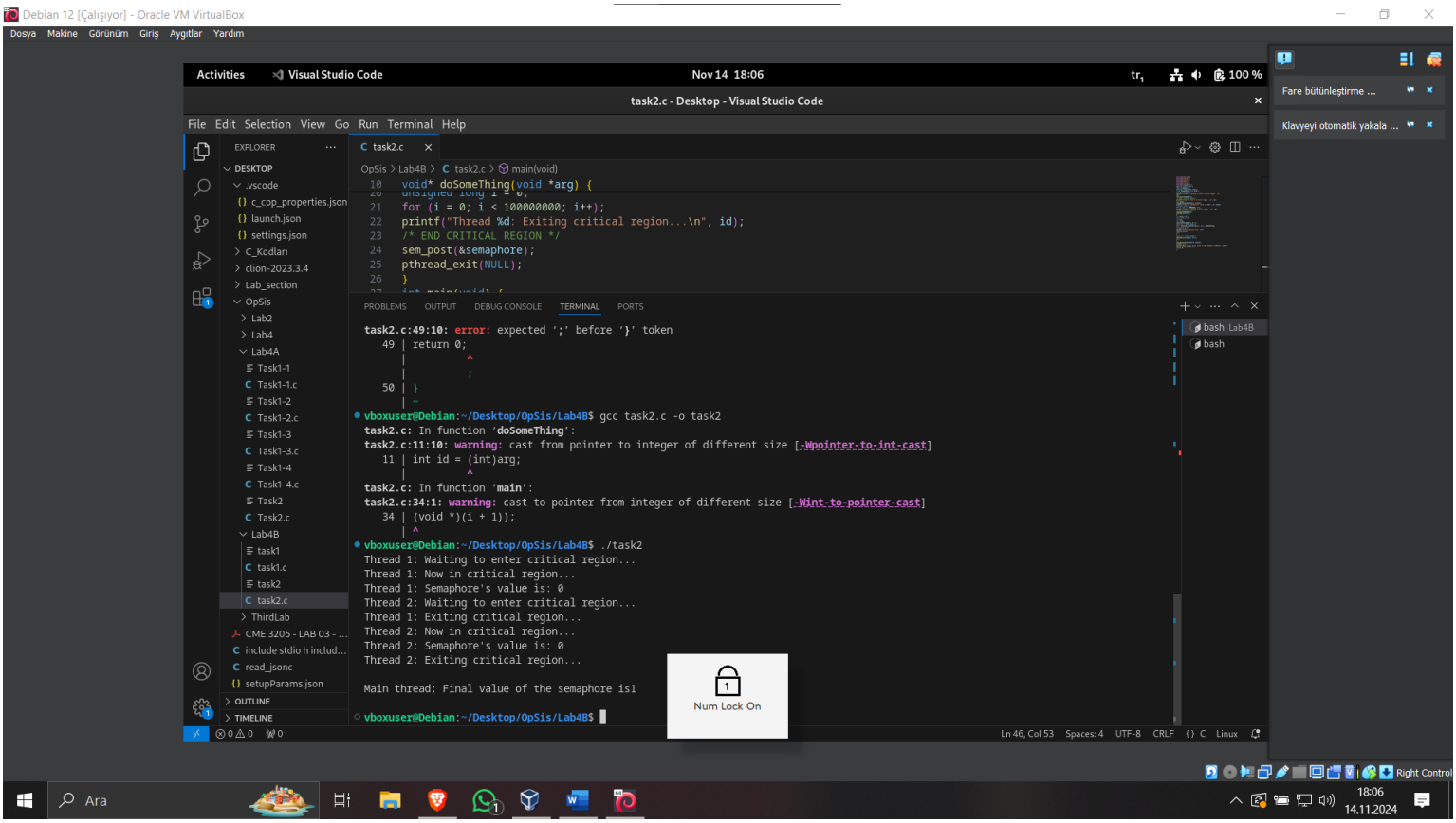



New Code Below:

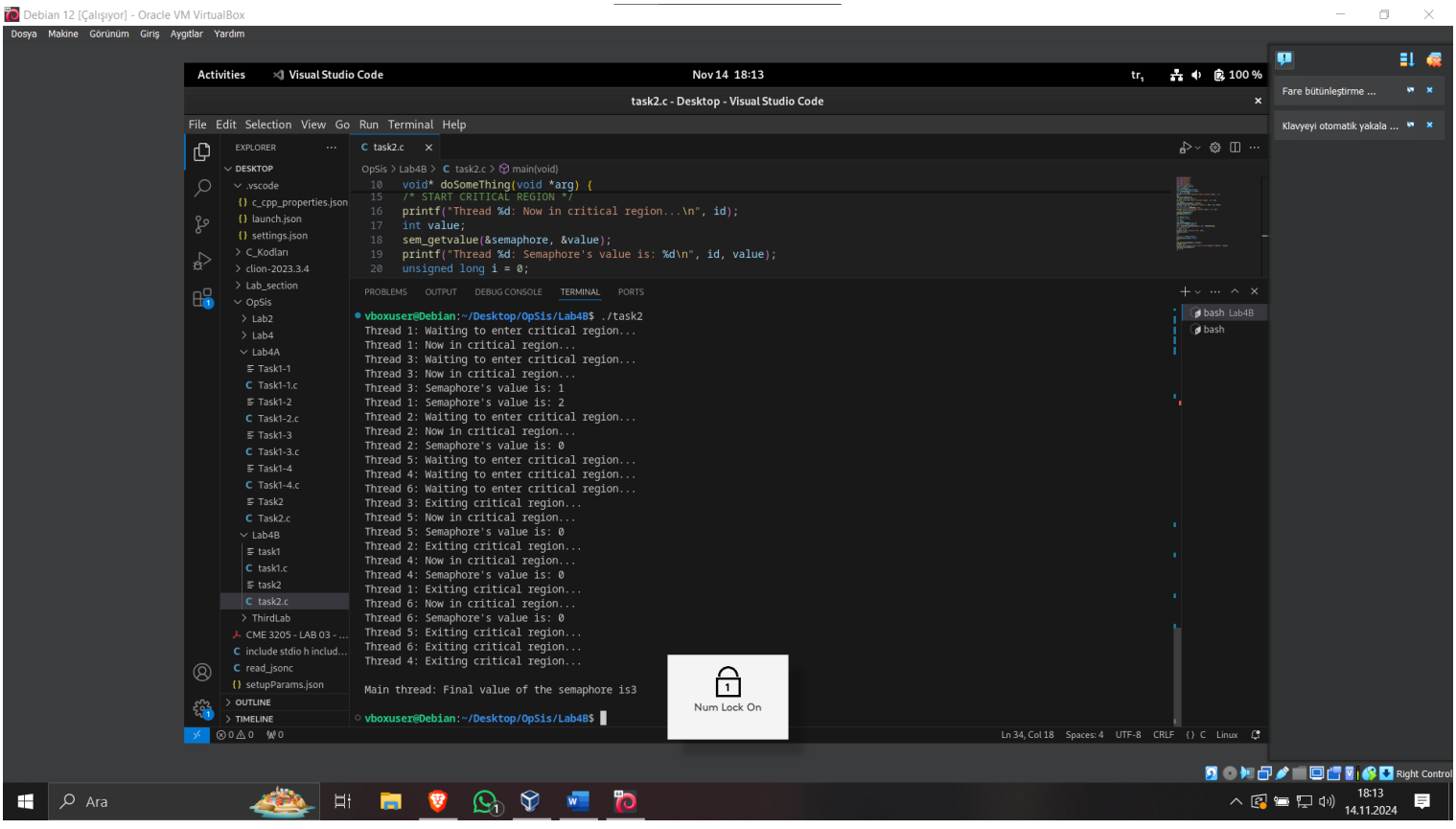
```
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <semaphore.h>
#include <stdint.h> // For intptr_t

#define THREAD_COUNT 4
pthread_t tid[THREAD_COUNT];
int x = 0, y = 0;
sem_t semaphore;

void* doSomeThing(void *arg) {
   int id = (int)(intptr_t)arg;
   sem_wait(&semaphore);
   printf("Thread %d locked the semaphore\n", id);
   x = y + 1;
   unsigned long i = 0;
   for (i = 0; i < 100000000; i++);
   y = x;
   printf("Thread %d done computing, unlocking the semaphore\n", id);
   sem_post(&semaphore);
   pthread_exit(NULL);
}

int main(void) {
   int i = 0, j = 0;
   int err;

   // Initialize the binary semaphore with 1 (allowing one thread at a time)
   if (sem_init(&semaphore, 0, 1) != 0) {
      printf("Semaphore init failed!\n");
      return 1;
   }

   // Create threads
   while (i < THREAD_COUNT) {
      err = pthread_create(&(tid[i]), NULL, &doSomeThing, (void *)(intptr_t)(i + 1));
      if (err != 0)
         printf("Can't create thread :[%s]!\n", strerror(err));
      i++;
   }

   // Wait for threads to finish
   while (j < THREAD_COUNT) {
      pthread_join(tid[j], NULL);
      j++;
   }

   printf("\n\nFinal values of x and y are %d, %d\n\n", x, y);

   // Destroy the semaphore
   sem_destroy(&semaphore);

   return 0;
}
```

# Task 2:

Thread Count is 2 and Semaphore is 1:




Thread Count is 6 and Semaphore is 3:



When we change the thread count to 6 and the semaphore value to 3, the first three threads can enter the critical region at the same time because the semaphore allows up to three threads. The other threads will have to wait until one of the first three threads finishes and releases the semaphore.