

**Department of Computer Science Fall 2023**

**CSC 203 – Operating Systems**

**Lab #10**

**Objective:**

**Semaphore Mechanism in Operating Systems**

To Implementation of a Classical problem (Producer-Consumer) using semaphores.

| Name of Student | Maha Rehan |
| --- | --- |
| Student ID | 22sp-018-se |
| Date of Lab Conducted |  |

**Objective:** Implementation of a Classical problem (Producer-Consumer) using semaphores.

**THEORY**

Semaphore is a simply a variable. This variable is used to solve critical section problem and to achieve process synchronization in the multi-processing environment.

The two most common kinds of semaphores are counting semaphores and binary semaphores. Counting semaphore can take non-negative integer values and Binary semaphore can take the value 0 & 1. only. A semaphore can only be accessed using the following operations: wait() and signal().

**wait(Semaphore s)**

**{**

**while (s==0); */\* wait until s>0 \*/***

**s=s-1;**

**}**

**signal(Semaphore s)**

**{**

**s=s+1;**

**}**

*[****In python, acquire() and release() provide wait() and signal() functionality, respectively]***

**Python’s Simple Lock** *using class threading.Lock*

A simple [mutual exclusion lock](http://faculty.salina.k-state.edu/tim/ossg/glossary.html#term-mutual-exclusion) used to limit access to one thread. This is a [semaphore w](http://faculty.salina.k-state.edu/tim/ossg/glossary.html#term-semaphore)ith s = 1. **acquire()**

Obtain a lock. The process is blocked until the lock is available.

**release()**

Release the lock and if another thread is waiting for the lock, wake up that thread.

**Python’s Semaphore**

*using class threading.Semaphore(s)*

**acquire()**

Obtain a semaphore. The process is blocked until the semaphore is available.

**release()**

Release the semaphore and if another thread is waiting for the semaphore, wake up that thread.

**Python: Producer-Consumer Solution using Semaphores**

The mutex semaphore provides mutual exclusion for accesses to the buffer pool and is initialized to the value 1. The empty and full semaphores count the number of empty and full buffers. The semaphore empty is initialized to the value n (n =5 in this example); the semaphore full is initialized to the value 0.

import threading import random import time

buf = []

empty = threading.Semaphore(5)

full = threading.Semaphore(0)

mutex = threading.Lock()

**def producer():**

nums = range(5)

global buf

num = random.choice(nums) empty.acquire() mutex.acquire() # added buf.append(num) print("Produced", num, buf) mutex.release() # added full.release()

time.sleep(1)

**def consumer():**

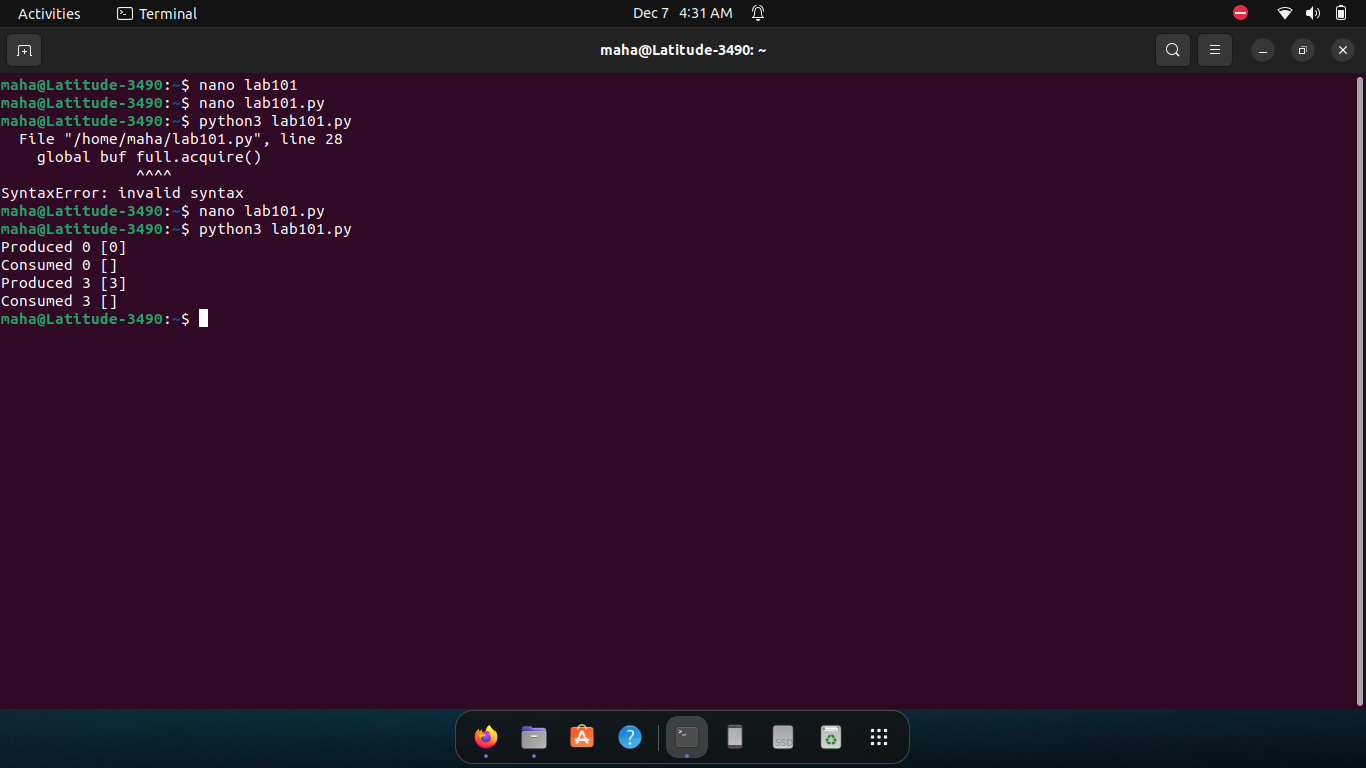
global buf full.acquire()

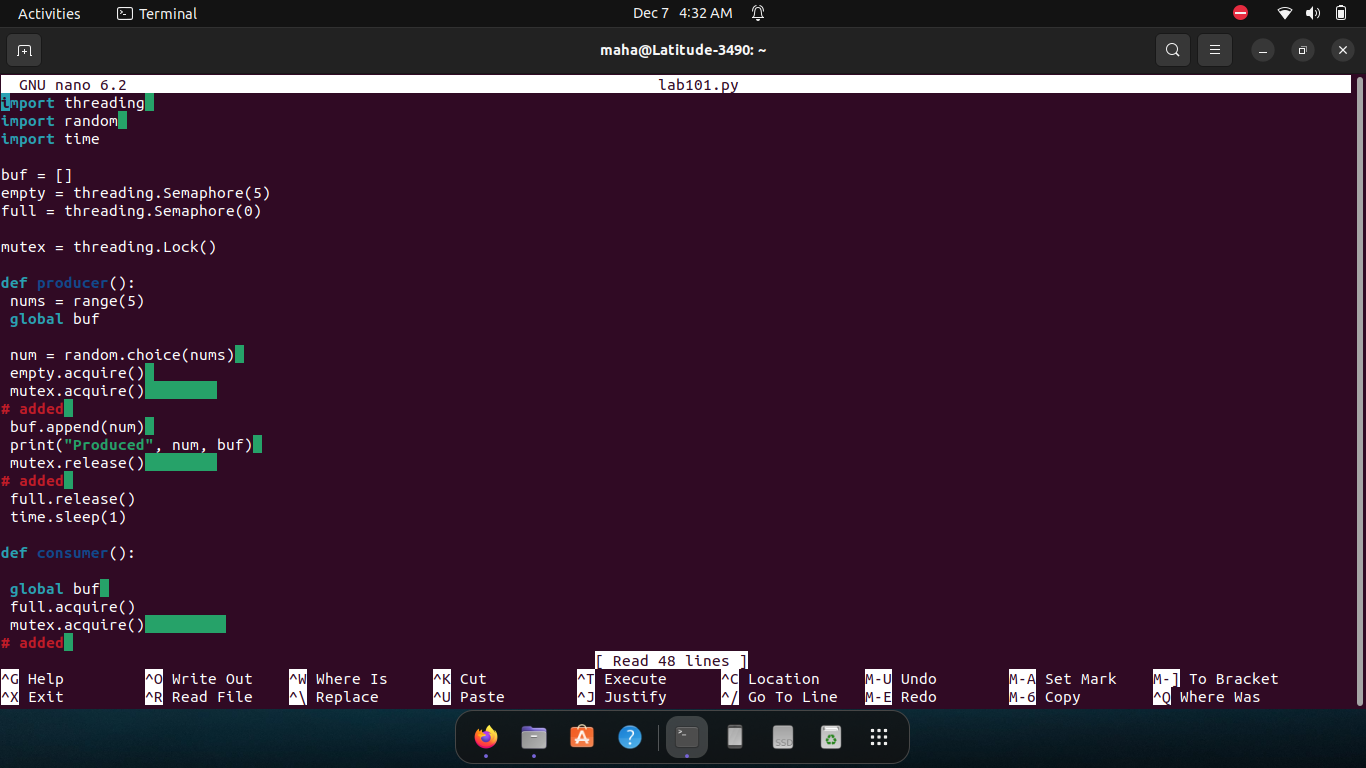
mutex.acquire() # added num = buf.pop(0) print("Consumed", num, buf) mutex.release() # added empty.release()

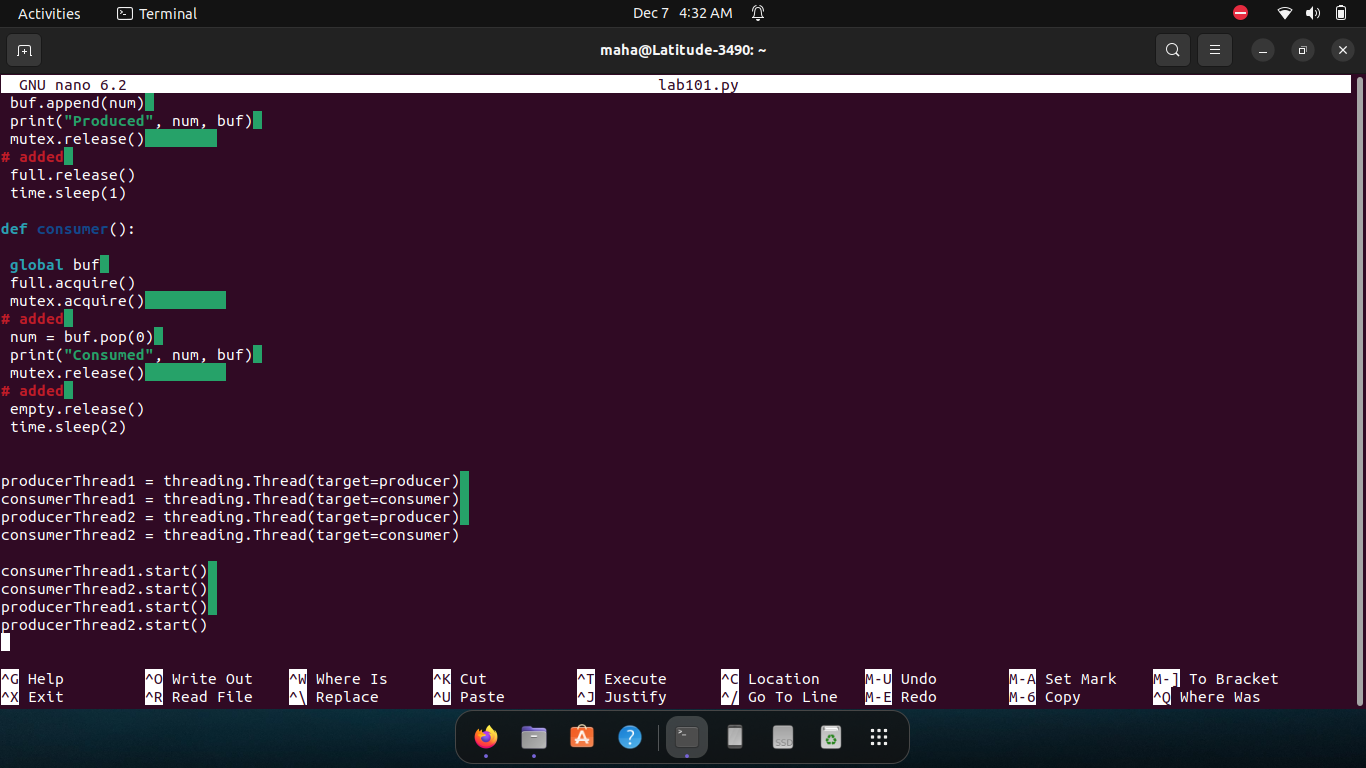
time.sleep(2)

producerThread1 = threading.Thread(target=producer) consumerThread1 = threading.Thread(target=consumer) producerThread2 = threading.Thread(target=producer) consumerThread2 = threading.Thread(target=consumer)

consumerThread1.start() consumerThread2.start() producerThread1.start() producerThread2.start()

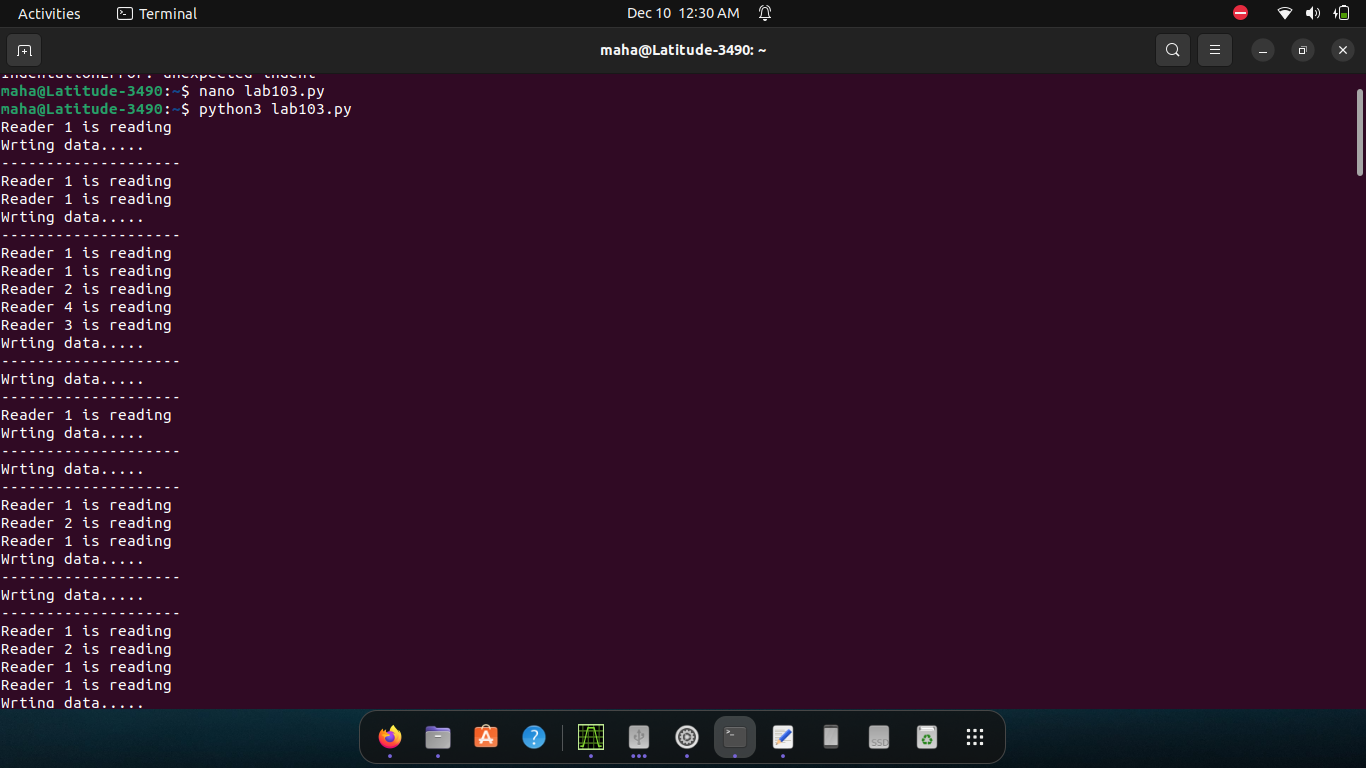


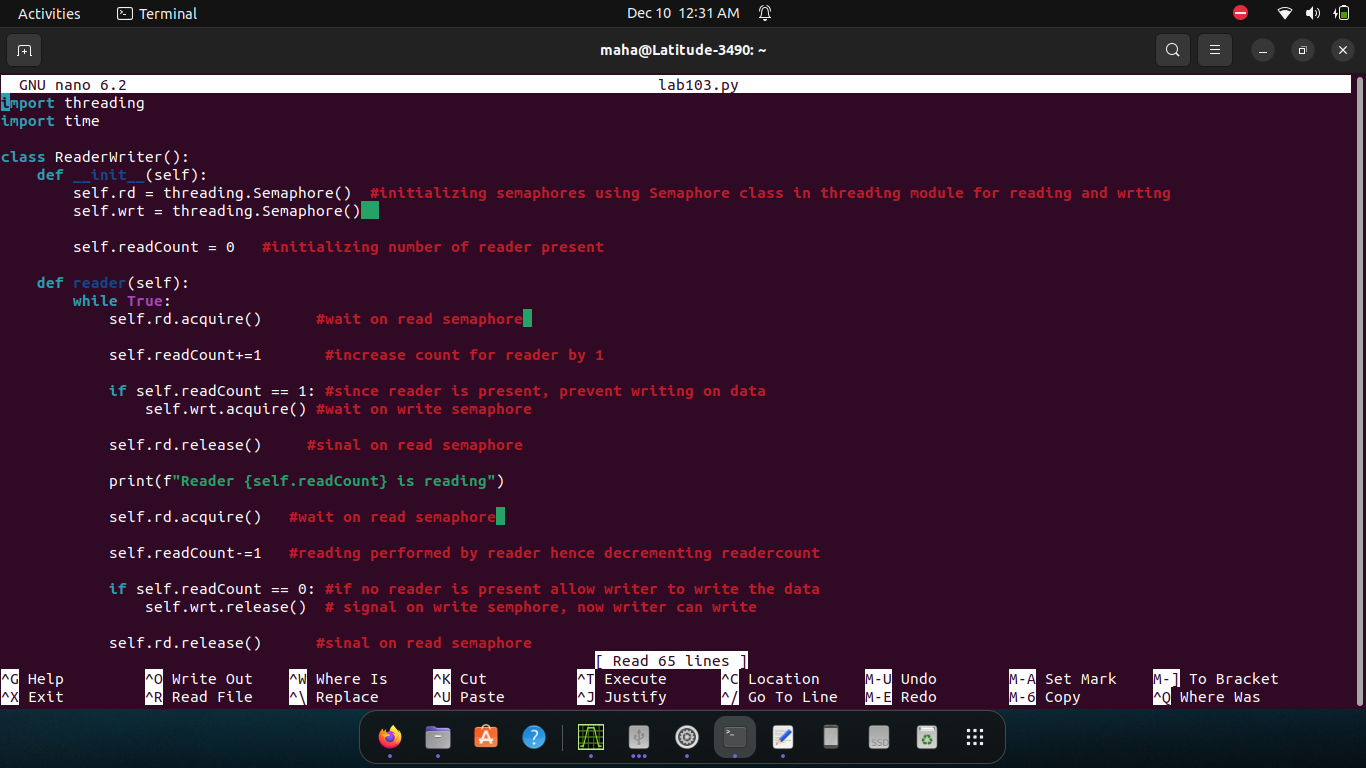


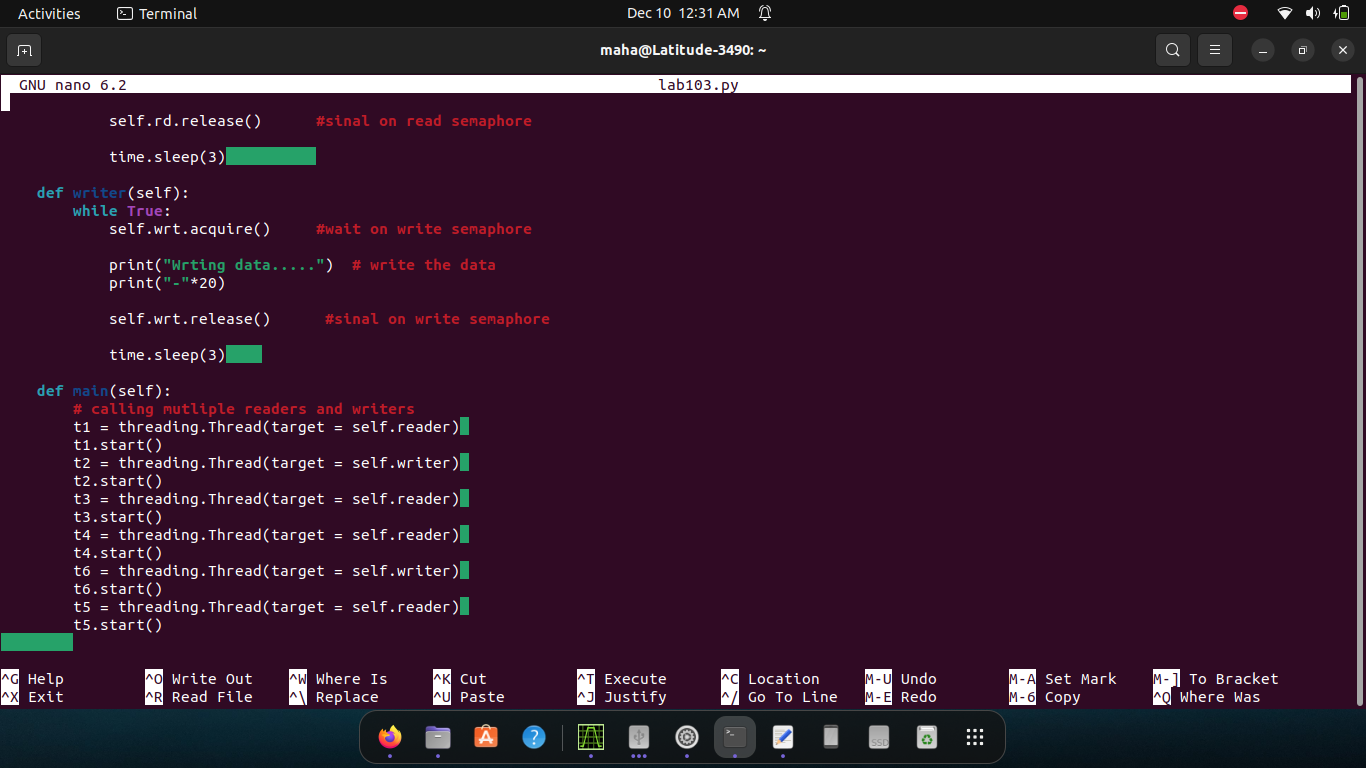


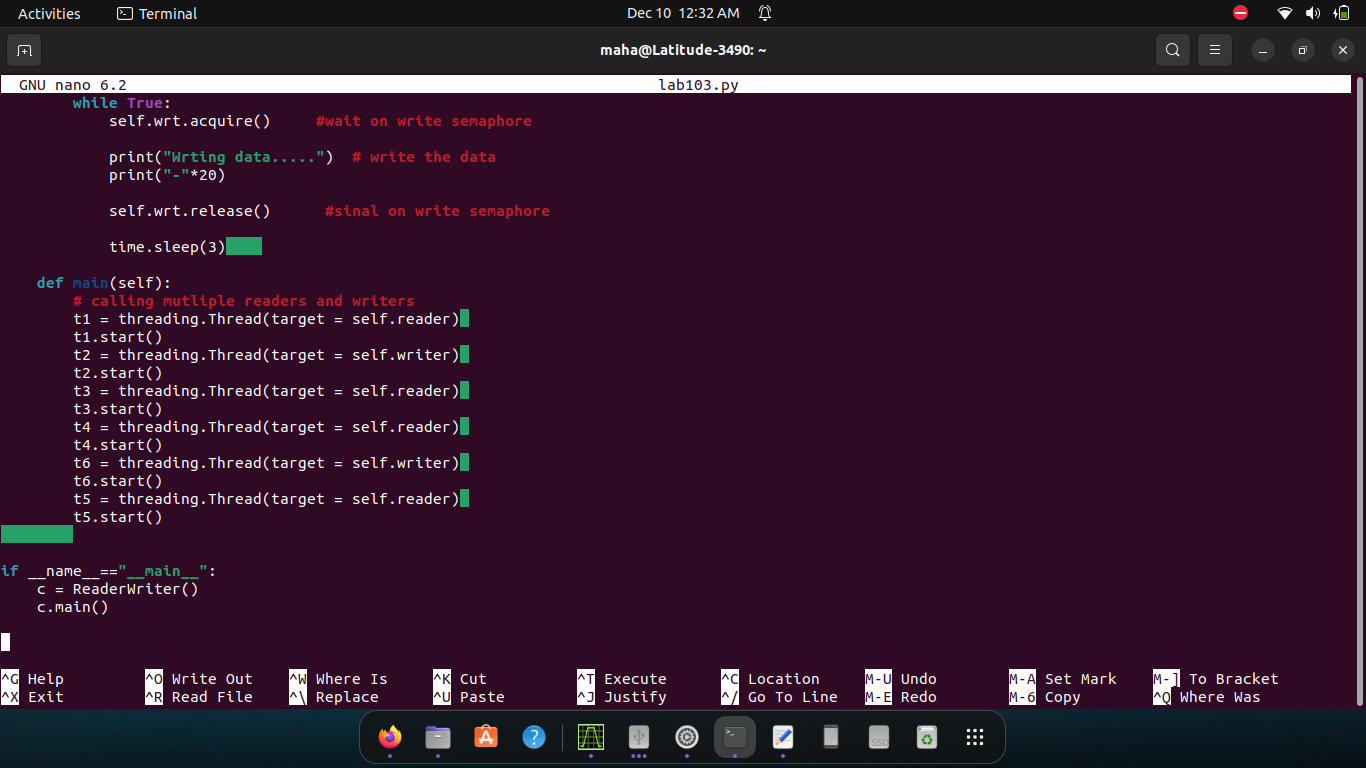
**Lab Exercise(s):**

1. Write a python program that demonstrates the synchronization of Readers and Writer Problem using semaphores.









1. Write a python program that demonstrates the synchronization of Consumer Producer Bounded Buffer

Problem using semaphore.

