**Coding Platform:** gedit & nano , for user level code & terminal for compilation

**Operating System:** Ubuntu

**Version:** 18.10

## Dining Philosopher’s Problem

## Project Description

In computer science, the dining philosopher’s problem is an example problem often used in concurrent algorithm design to illustrate synchronization issues and techniques for resolving them. The dining-philosophers problem is considered a classic synchronization problem neither because of its practical importance nor because computer scientists dislike philosophers but because it is an example of a large class of concurrency-control problems. It was originally formulated in 1965 by Edsger Dijkstra as a student exam exercise, presented in terms of computers competing for access to tape drive peripherals. Soon after, Tony Hoare gave the problem its present formulation

## 2. Problem Statement

Five silent philosophers sit at a round table around a bowl of spaghetti. Chopsticks are placed between each pair of adjacent philosophers. Each philosopher must alternately think and eat. However, a philosopher can only eat spaghetti when they have both left and right chopsticks. Each chopstick can be held by only one philosopher and so a philosopher can use the chopstick only if it is not being used by another philosopher. After an individual philosopher finishes eating, they need to put down both forks so that the forks become available to others. A philosopher can take the fork on their right or the one on their left as they become available, but cannot start eating before getting both chopsticks. Eating is not limited by the remaining amounts of spaghetti or stomach space; an infinite supply and an infinite demand are assumed. The problem is how to design a discipline of behavior (a concurrent algorithm) such that no philosopher will starve; i.e., each can forever continue to alternate between eating and thinking, assuming that no philosopher can know when others may want to eat or think.

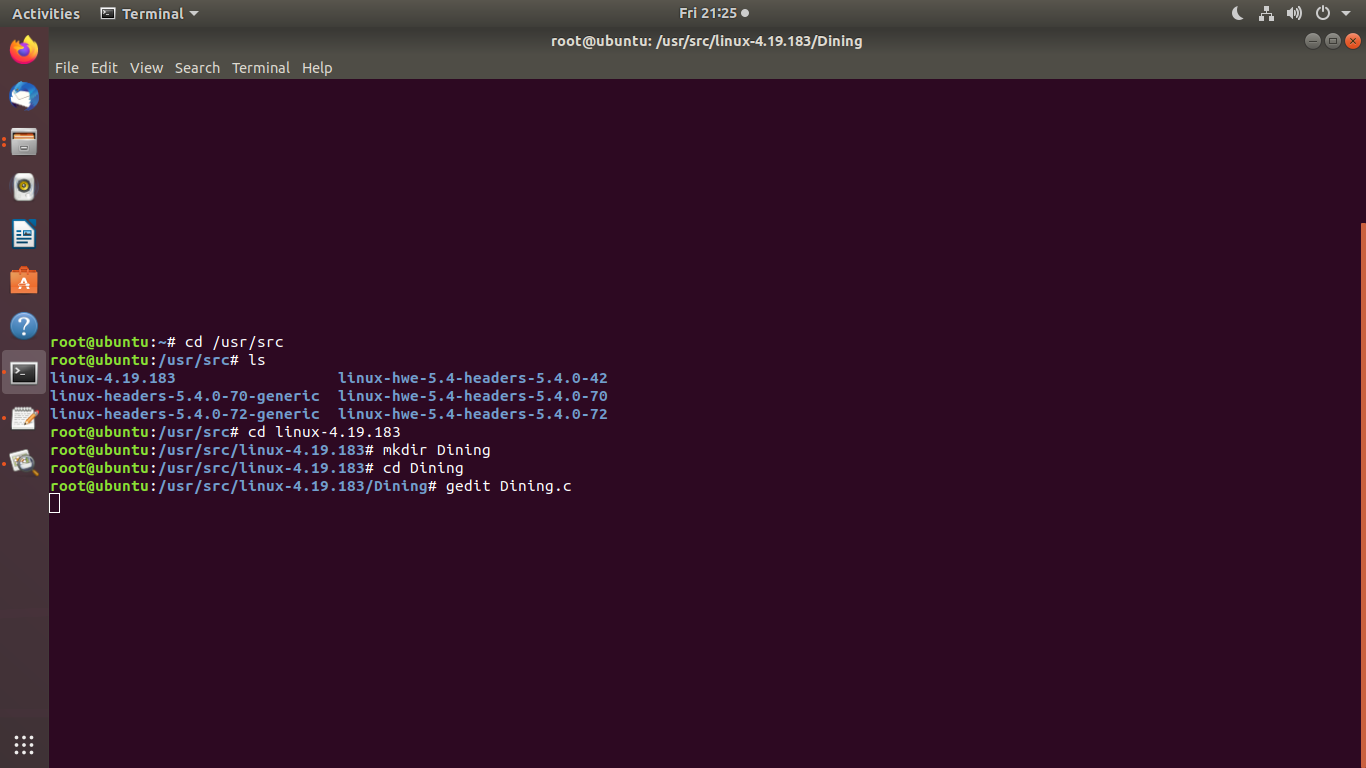
## Operating System Implementation

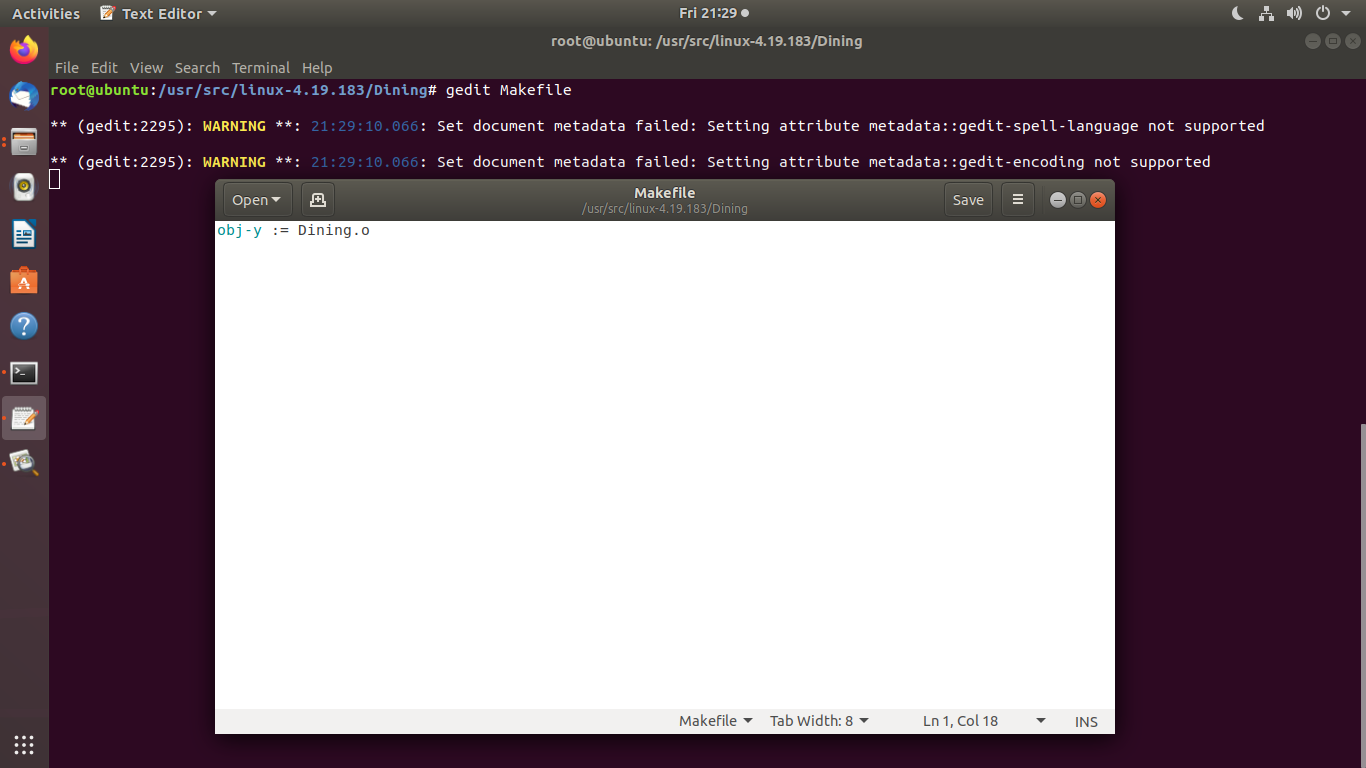
We will utilize Semaphore and Threads to exhibit and take care of the issue. One straightforward arrangement is to address every chopstick with a semaphore. A scholar attempts to get a chopstick by executing a wait () procedure on that semaphore. At the point when philosopher delivers his chopsticks by executing the signal () procedure on the fitting semaphores, our code will be depend on kernel level and there is semaphore

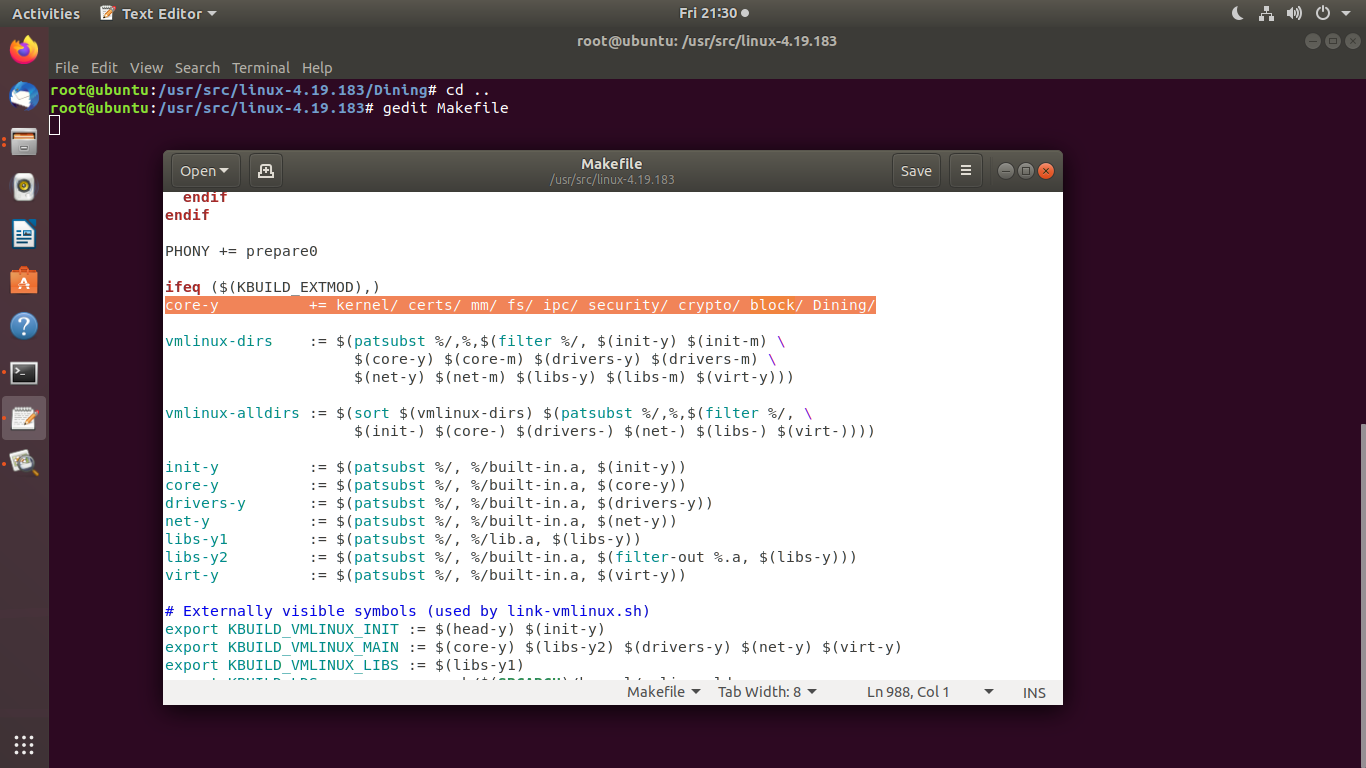
## Project Outcome

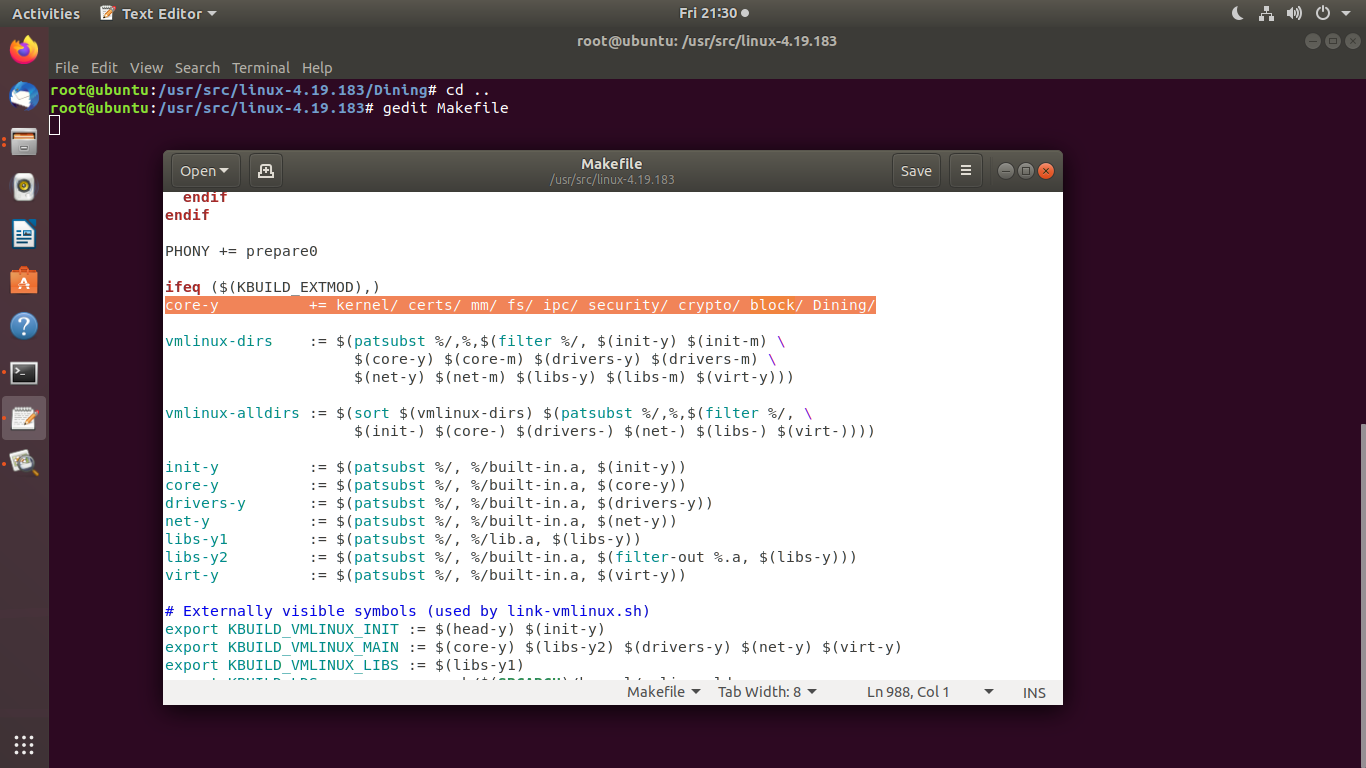
Getting rid of deadlock by properly synchronizing. Meeting need to allocate several resources among several processes in a deadlock-free and starvation-free manner. The main Objective of this project was to learn the use of semaphore and to understand the concept of deadlock and way to resolve this problem.

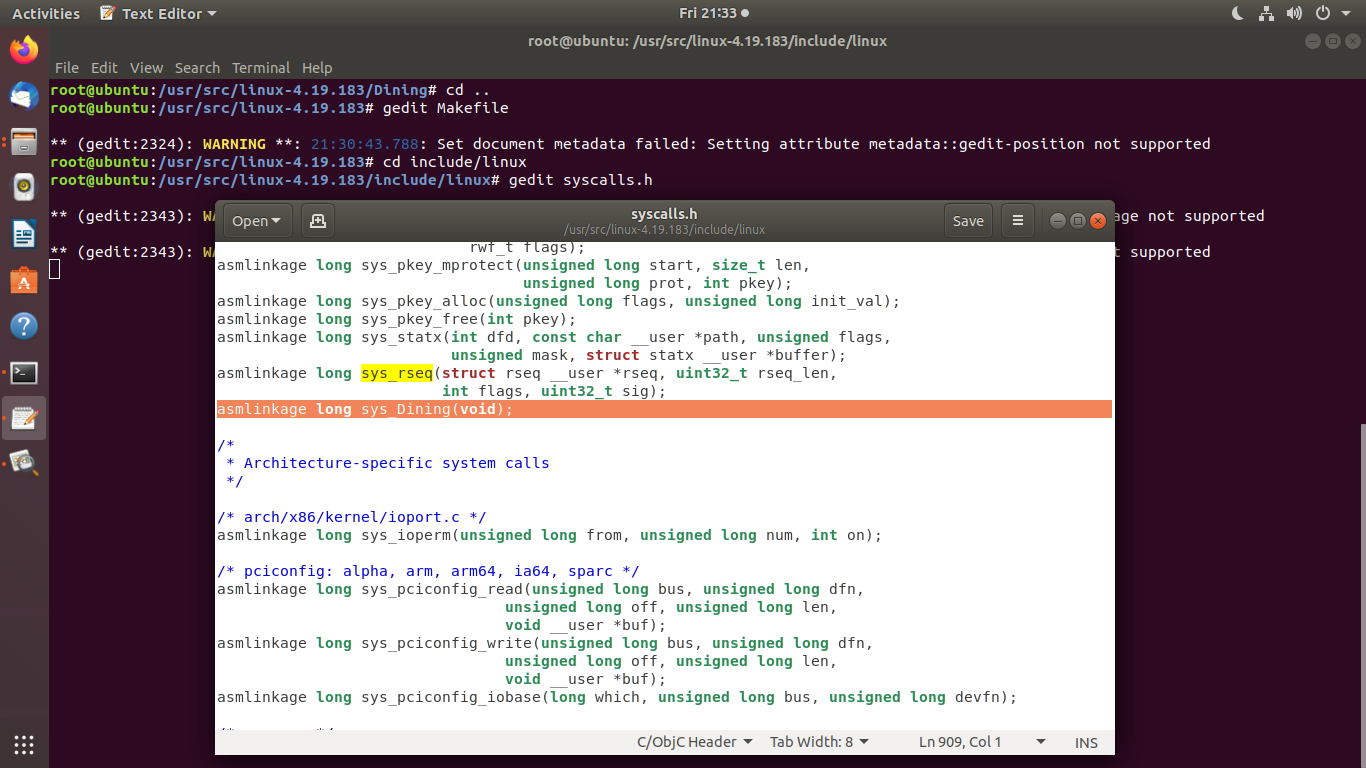
## Project Snippets

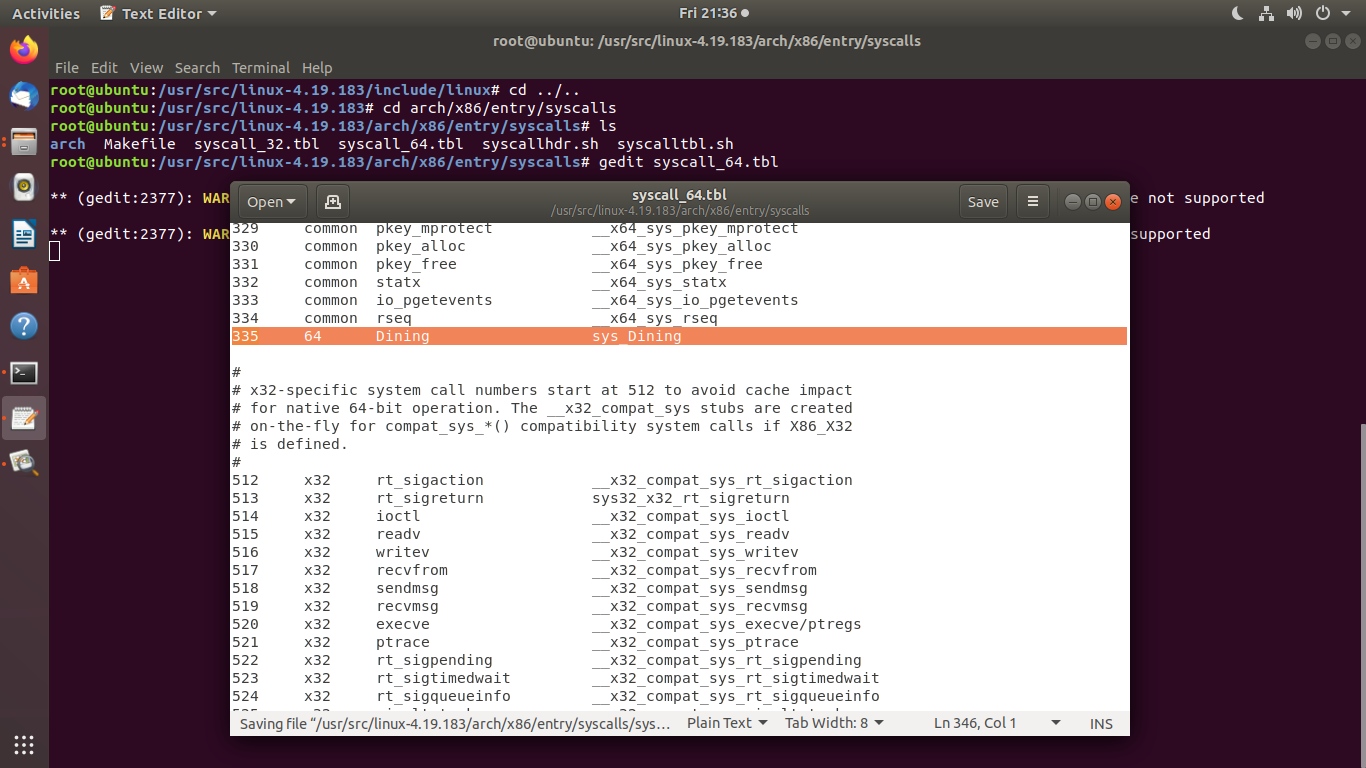


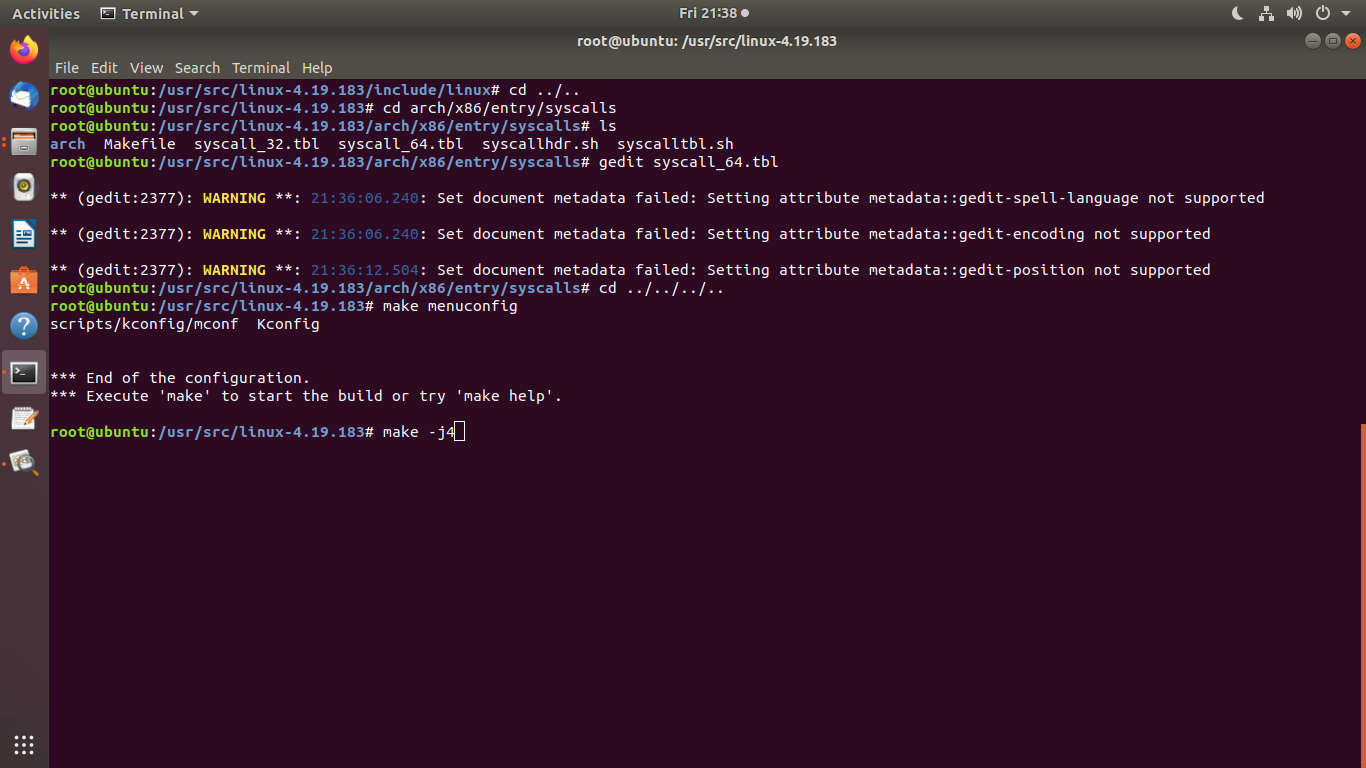




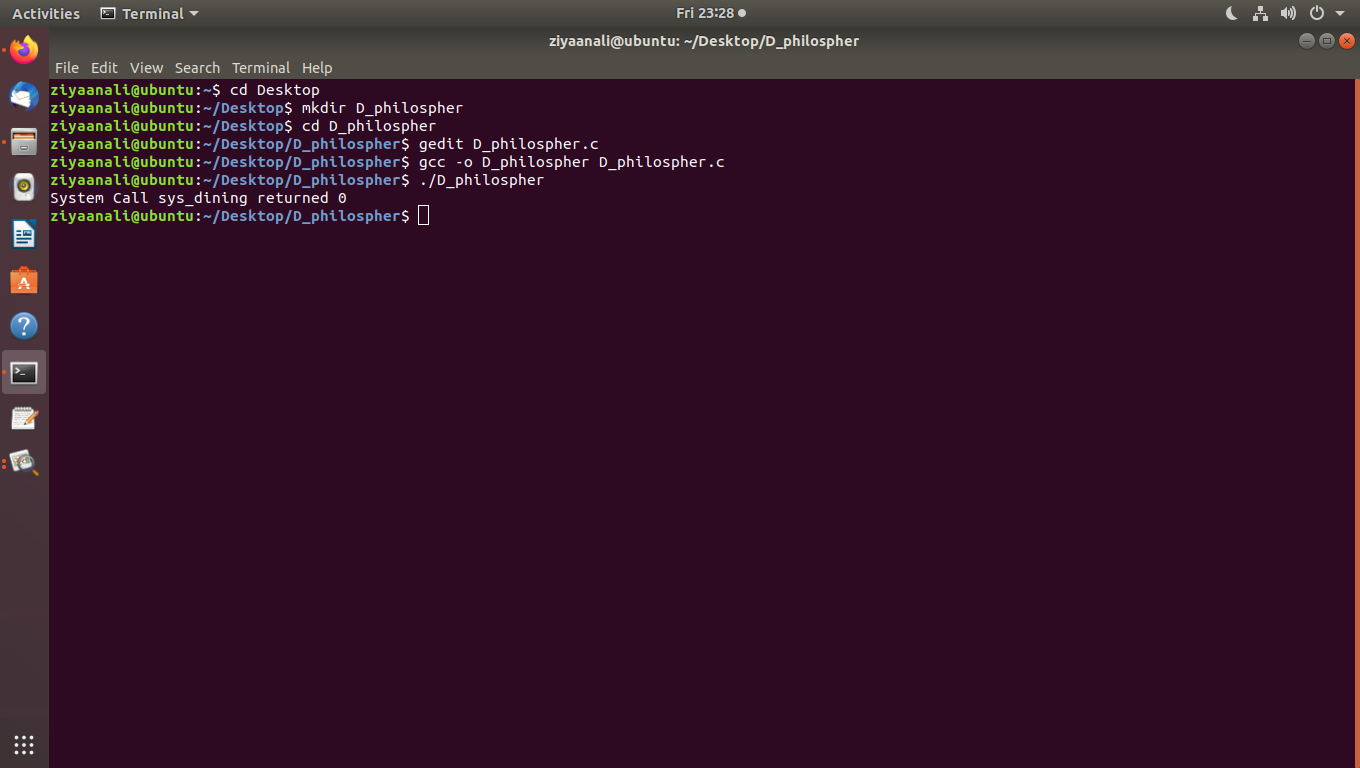


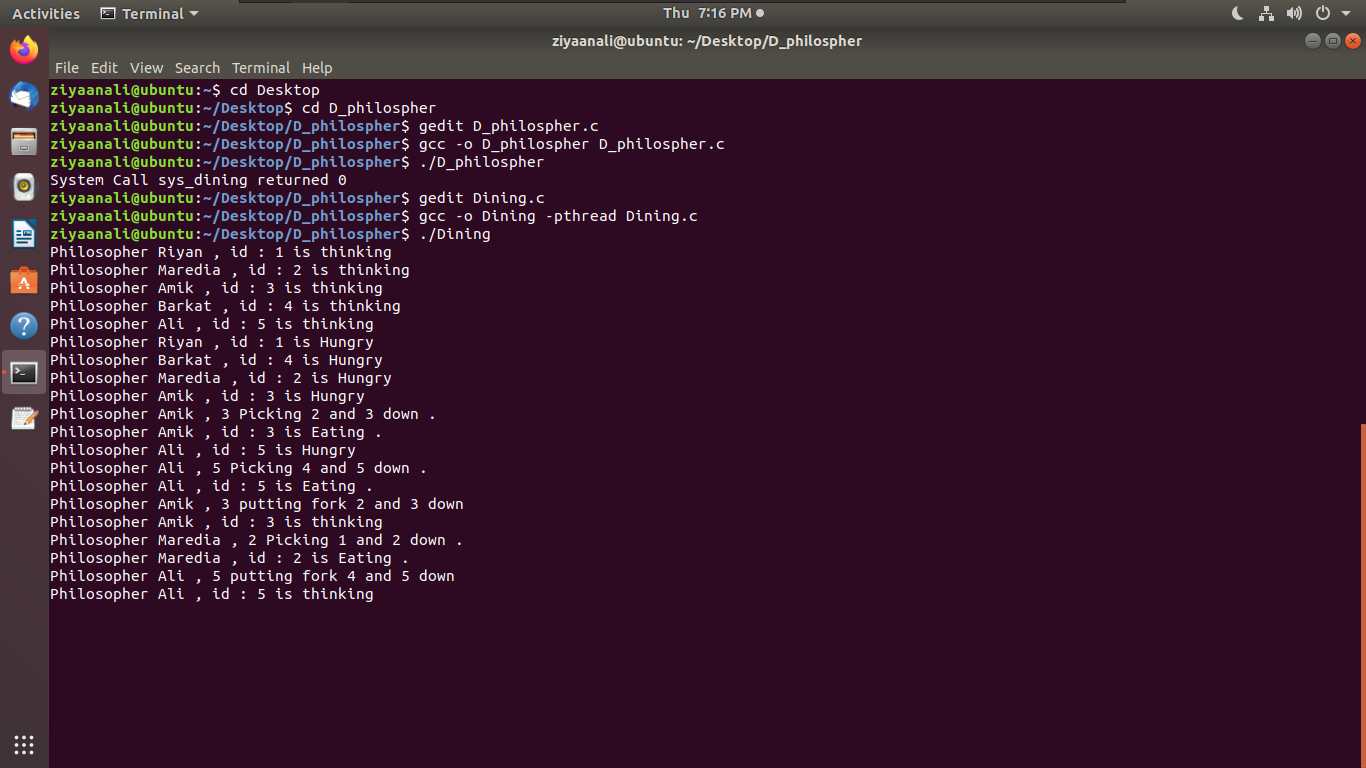












## Code File:

****