

OPERATING SYSTEM

DINNING PHILOSOPHER

PROJECT REPORT

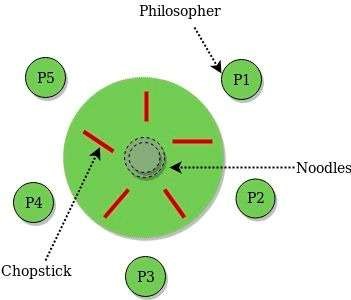
GROUP MEMBERS: STUDENT ID:

HAIRAM NASEEM 21K-3062

HUZAIFA ZULFIQAR 21K-3010

OMER SHOAIB 21K-3066

# Title: Dining Philosophers Problem in C



Introduction:

The Dining Philosophers problem is a classic synchronization problem in computer science that illustrates challenges with concurrency and resource allocation. The problem is described as a group of philosophers who sit at a circular table, and in front of each philosopher is a plate of spaghetti and a fork. The philosophers spend their time alternating between thinking and eating, but they require two forks to eat. The challenge is to prevent deadlock and starvation while allowing each philosopher to eat.

Goals:

Our goal was to create a program that simulates the dining philosophers problem and demonstrates the use of semaphores to avoid deadlock and starvation.

Methodology:

In this project, we implemented the Dining Philosophers problem in C language using pthreads and semaphores. We used the following steps to implement the solution:

Define the state of each philosopher (thinking, hungry, or eating) using an enum.

Create a thread for each philosopher and a semaphore for each fork and a mutex semaphore.

Implement the grab\_forks() function, which attempts to pick up two forks and move the philosopher to the eating state. If a fork is not available, the philosopher waits.

Implement the put\_down\_forks() function, which releases the two forks and moves the philosopher back to the thinking state.

In the philosopher function, loop indefinitely, alternating between thinking, grabbing forks, eating, and putting down forks.

In main, initialize the semaphores, ensure the last philosopher picks up the right fork first to avoid deadlock, and create and join threads for each philosopher.

Problem Encountered:

One of the problems encountered in the implementation of the dining philosophers problem is the possibility of deadlock. Deadlock occurs when each philosopher holds one fork and is waiting for the other fork to become available, creating a situation where no philosopher can eat.

One possible solution to this problem is to ensure that at least one philosopher picks up their right fork first. This helps prevent a scenario where each philosopher is holding their left fork and waiting for their right fork to become available, which can lead to a deadlock.

In the given implementation, this solution is applied by adding the following line of code before the threads are created:

sem\_wait(&forks[NUM\_PHILOSOPHERS - 1]);

This ensures that the last philosopher picks up their right fork first before the other philosophers, thereby avoiding the possibility of a deadlock.

Another issue that can arise is the possibility of starvation, where a philosopher is constantly blocked from acquiring both forks and is therefore unable to eat. One solution to this problem is to implement a fairness policy that ensures that all philosophers have an equal chance of acquiring forks. This can be achieved by introducing a queue that orders the philosophers based on the time they have been waiting for forks.

However, this solution is not implemented in the given code.

Results:

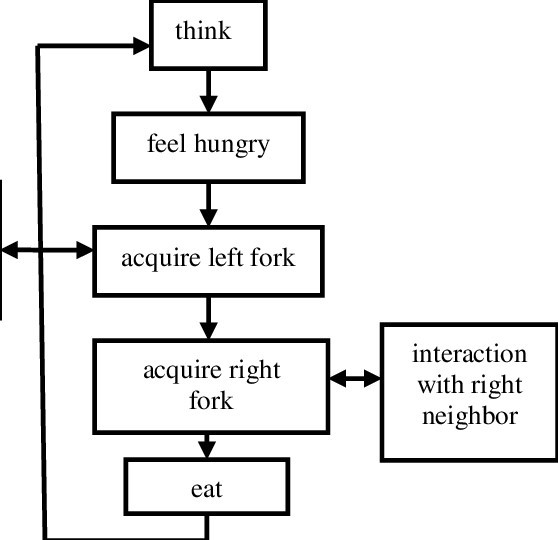
After running the program, we observed that each philosopher was able to take turns eating without deadlocking or starving. We also observed that the output was printed correctly, indicating the state of each philosopher and which forks were picked up or put down.

Evaluation:

Overall, we believe that our implementation of the Dining Philosophers problem in C language using pthreads and semaphores was successful. We were able to prevent deadlock and starvation and allow each philosopher to eat. However, we did encounter a potential issue with the implementation, which is that the semaphores for each fork were initialized with a value of 1, which could lead to multiple philosophers grabbing the same fork at the same time. A potential improvement would be to use binary semaphores (initialized with a value of 0) instead to ensure exclusive access to each fork.

Conclusion:

In conclusion, the Dining Philosophers problem is an interesting challenge that illustrates the complexities of concurrency and resource allocation. We successfully implemented a solution to the problem in C language using pthreads and semaphores. While the implementation worked correctly, there were potential areas for improvement that could increase the efficiency and reliability of the program.

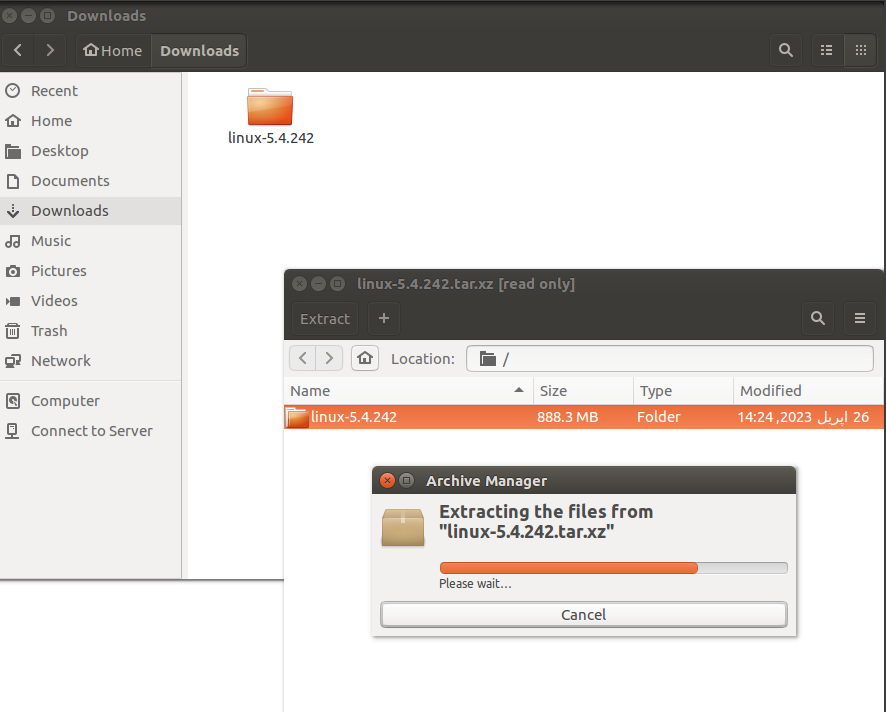


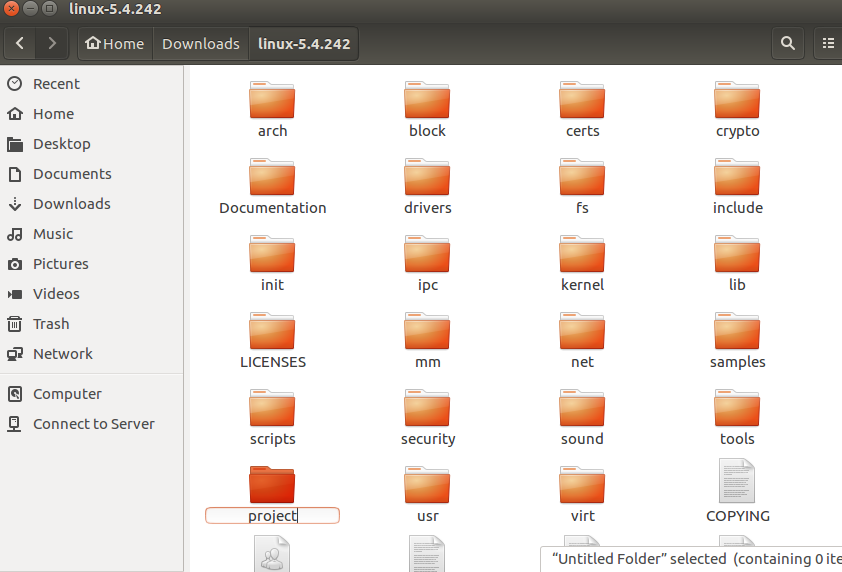
**Steps of System Call:**

1. Downloading a kernel:

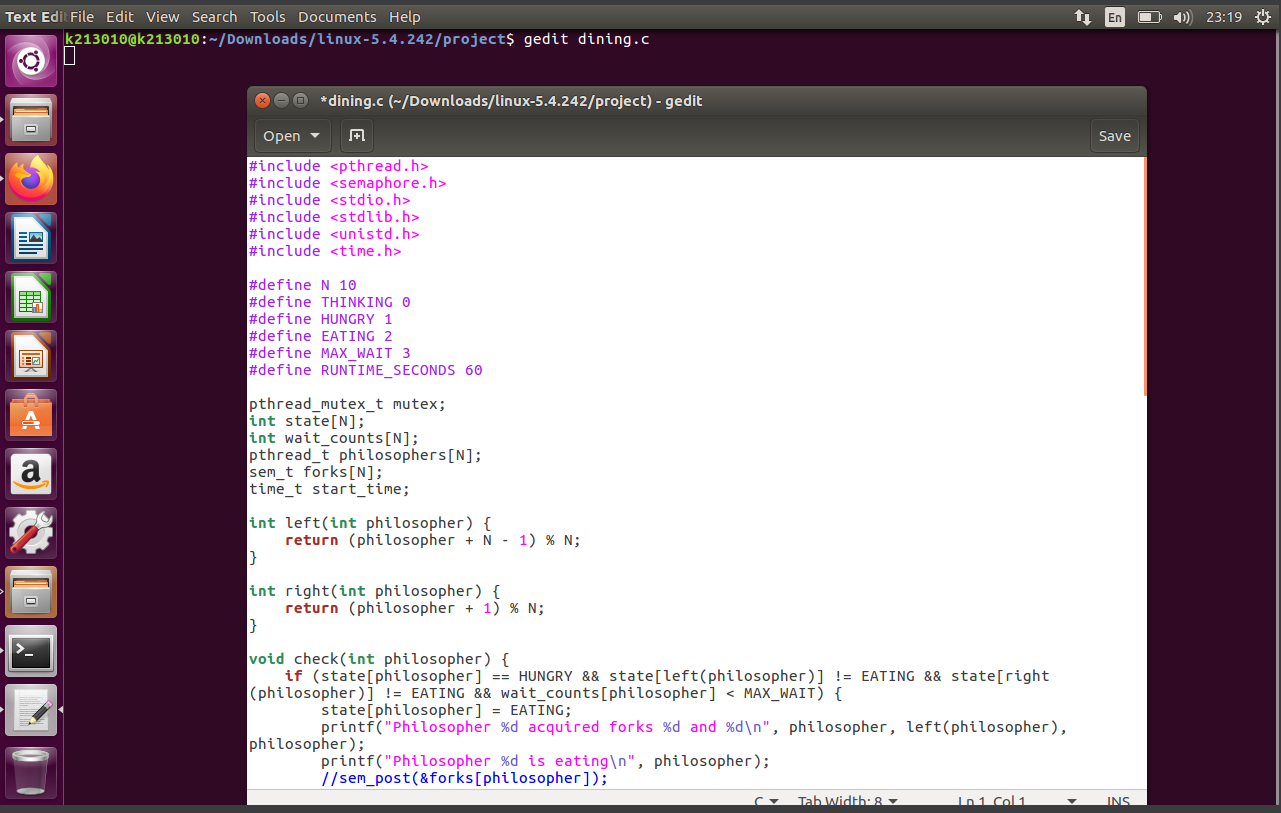


1. Extracting the kernel and making a new folder “project”:

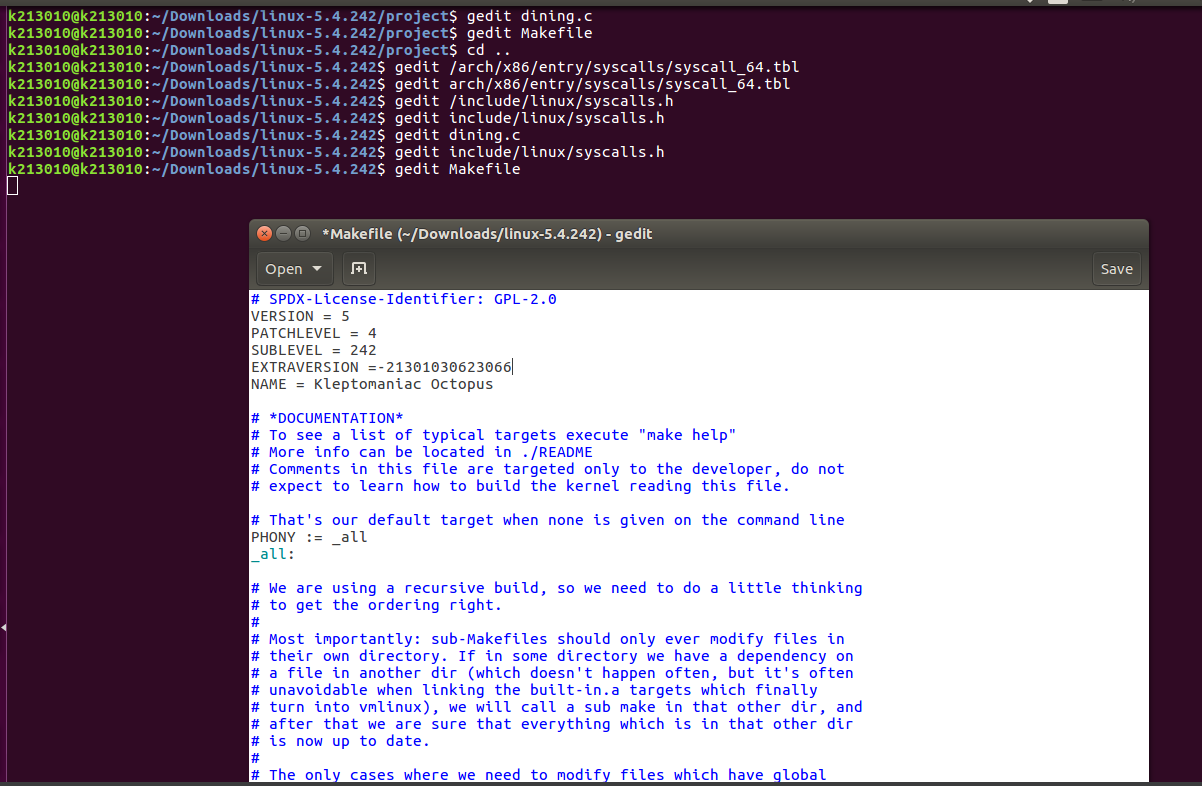


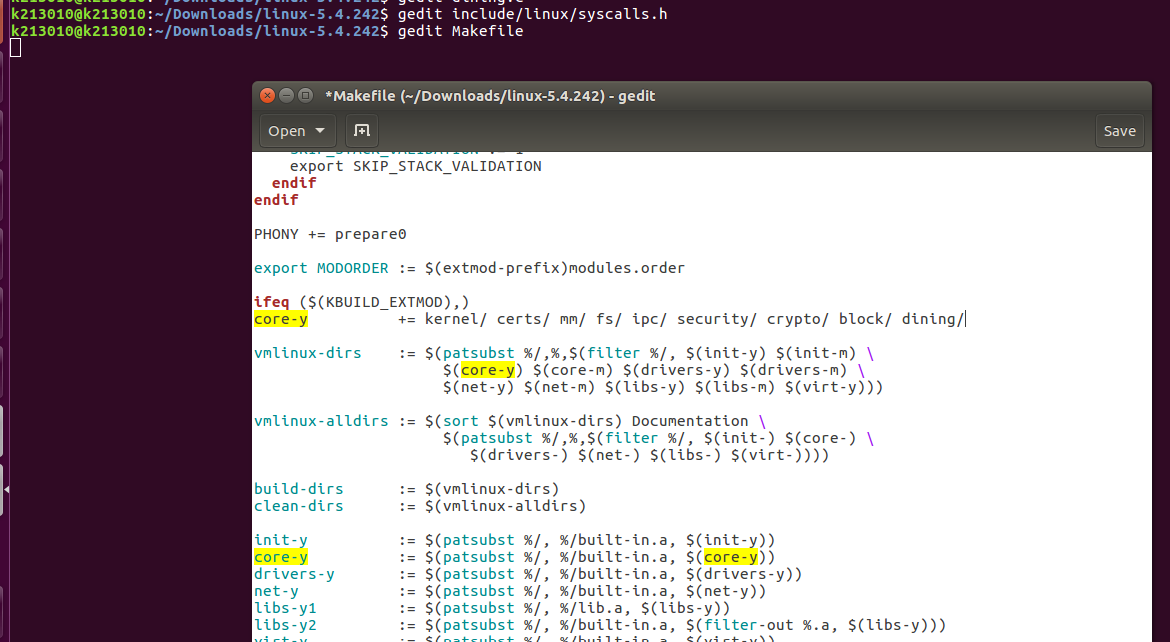


1. Adding Dining Philosopher code for the system call:

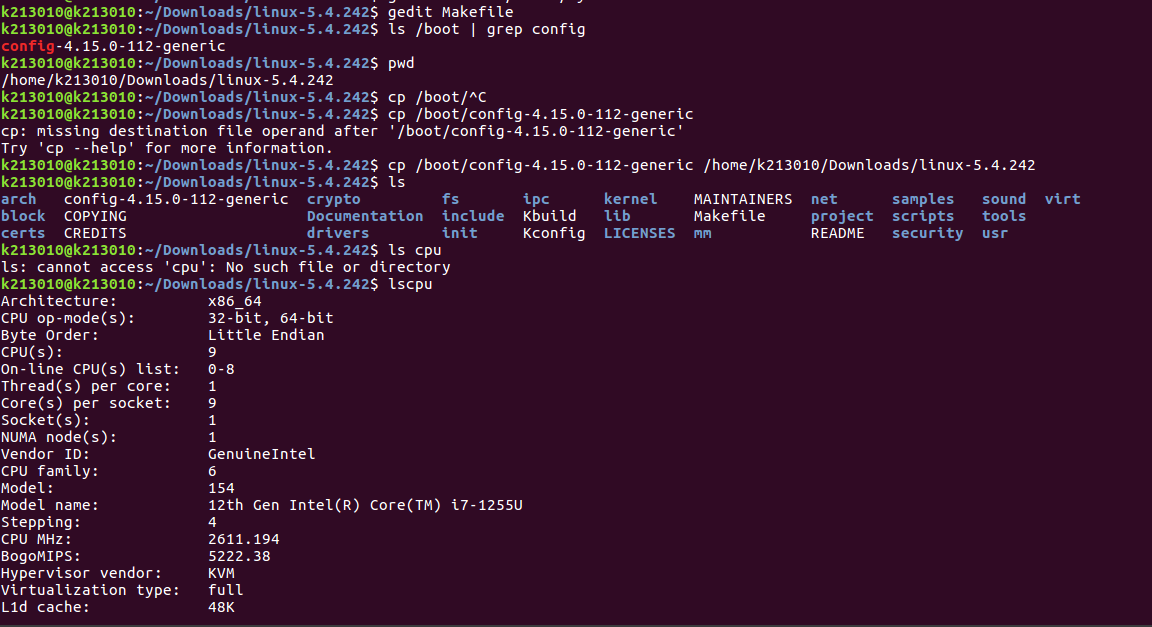


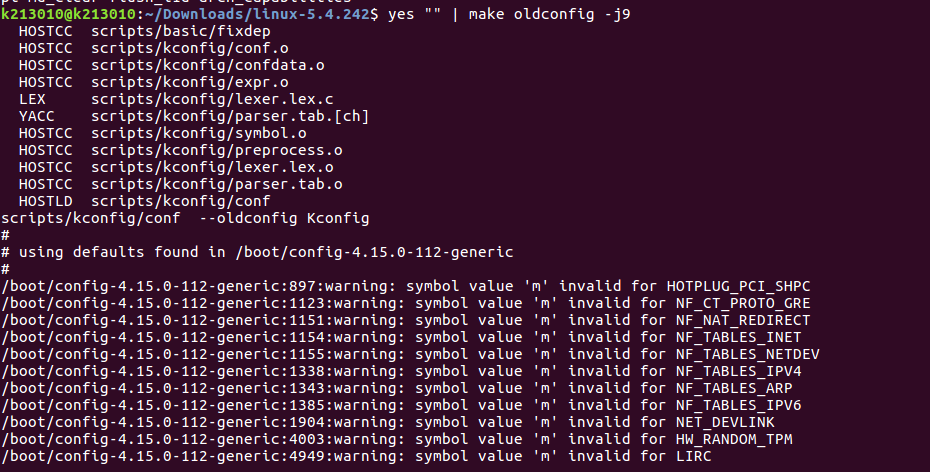
1. Changing version and adding the project folder in the kernel’s Makefile:





1. Creating a config file:





1. Compiling the kernel:

