

**Project Report**

|  |  |
| --- | --- |
| **Name** | **Roll-No:** |
| Syed Abdul Rehman | 21K-3156 |
| Zulqarnain Warsi | 21K-4689 |
| Raja Ibrahim | 21K-4585 |

**Instructor:** Miss Nausheen Shoaib

**Table of Contents**

[Introduction: 3](#_Toc134817248)

[Objective: 3](#_Toc134817249)

[Background: 4](#_Toc134817250)

[Language: 4](#_Toc134817251)

[Platform: 4](#_Toc134817252)

[Methodology: 5](#_Toc134817253)

[Code: 6](#_Toc134817254)

[Results: 10](#_Toc134817255)

[Output: 10](#_Toc134817256)

[Conclusion: 12](#_Toc134817257)

**Title: Ice Cream Factory Problem (System Call)**

# 

# **Introduction:**

The Ice Cream Factory Problem is a challenge that requires coordination between production and delivery of ice cream from a factory to multiple retailers. It is an important problem to solve because it involves managing resources and scheduling efficiently to produce and deliver ice cream on time while reducing costs.

# **Objective:**

The objective of this code is to simulate the manufacturing process of the ice cream in an ice cream factory. The code uses semaphore and kernel threads to manage the various stages of manufacturing process such as boiling, adding sugar, flavors, coning and freezing finally wrapping it to be served. The objective is to show how semaphores and kernel threads could be used to manage the multi stages of its manufacture.

# **Background:**

The project implements a solution for an ice cream factory, program receives the data itself about the orders and its details Then it creates threads and uses extensive usage of semaphores to handle the concurrency.

# **Language:**

C language.

# **Platform:**

Linux/Ubuntu

# **Methodology:**

Our program uses system call implementation to kernel for this project. Project involves use of semaphores to deal with mutex and handling synchronization between threads. The system call approach to solving the Ice Cream Factory Problem involved designing and implementing a set of system calls which allowed the factory to manage the production and delivery of ice cream to retailers. It receives input for the number of orders, and the no. of boilers, sugar, flavor, cone, freezing, and wrapping required for each order. The CurrentOrder is a global variable that holds the order no. and the Num array is an integer array which have the order number for each order that is placed. Then the thread is created for each order, it defines a function called sys\_icecream() which is the system user can make to place order. Code initializes semaphores to control its manufacturing process. The Cust() func represents kernel thread for simulating cust order (placing in the waiting, picking from the counter and start manufacturing) uses up and down to inc and dec the value of semaphore. Furthermore, StartManufacture() manages the manufacture process creates child threads to handle all steps and waking up all the threads waiting for the response using wake\_up\_process(). In Manufacture() up and down are widely used to manage all processes of manufacturing correct usage of semaphores surely impose synchronization mechanism which will avoid deadlocks.

* Adding System call:

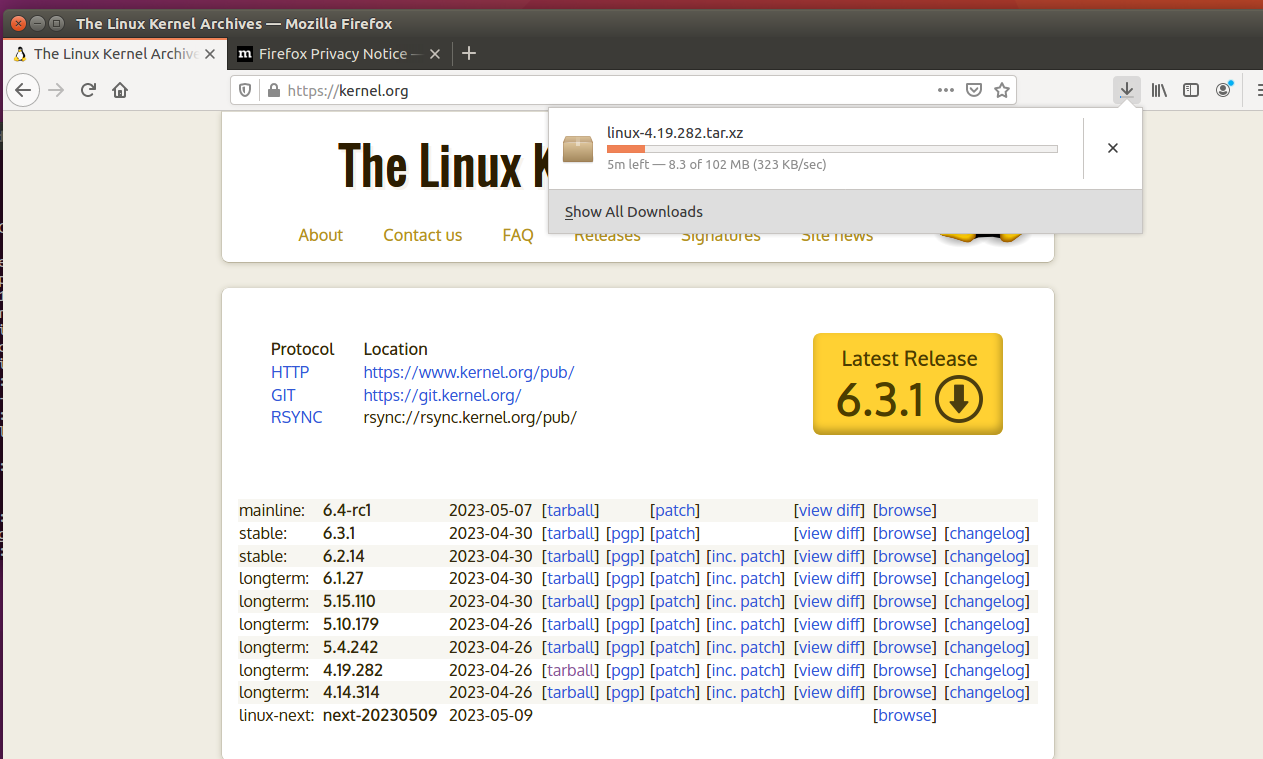
<https://youtu.be/qcsKGWshRZE>

Some important steps:

Running these commands:

sudo apt-get install gcc • sudo apt-get install libncurses5-dev • sudo apt-get install bison • sudo apt-get install flex • sudo apt install make • sudo apt-get install libssl-dev • sudo apt-get install libelf-dev • sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu $(lsb\_release -sc) main universe" • sudo apt-get update • sudo apt-get upgrade

Downloading the kernel

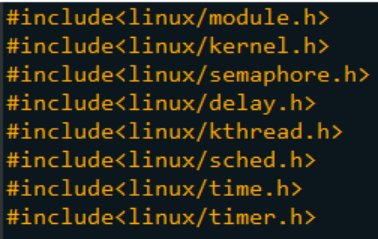


Extracting the kernel

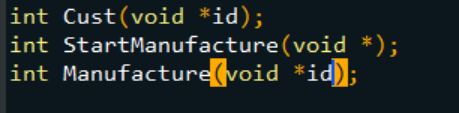
Making a new folder called as Icecream and then adding entire most important code of this project in it.

# **Code:**

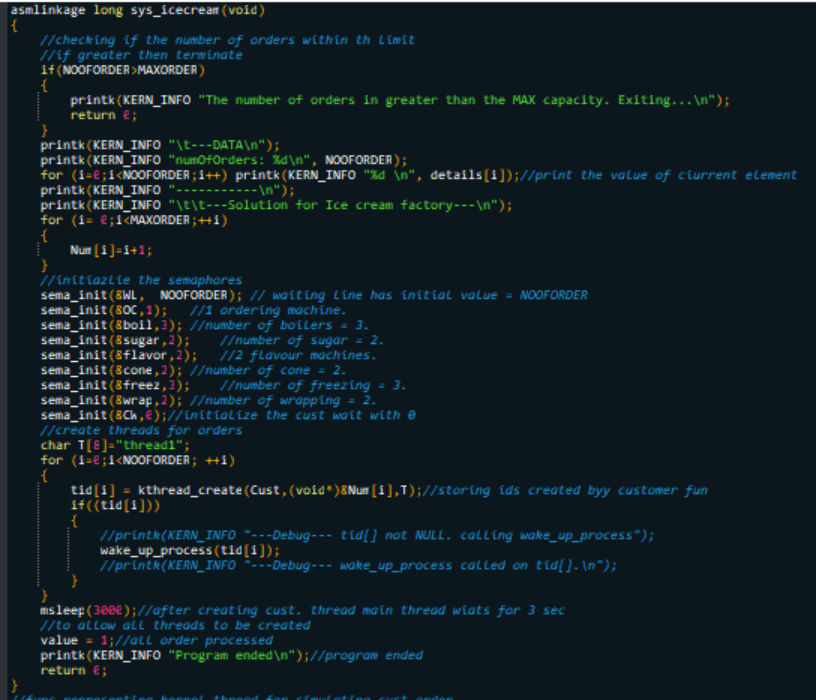
Important libraries:



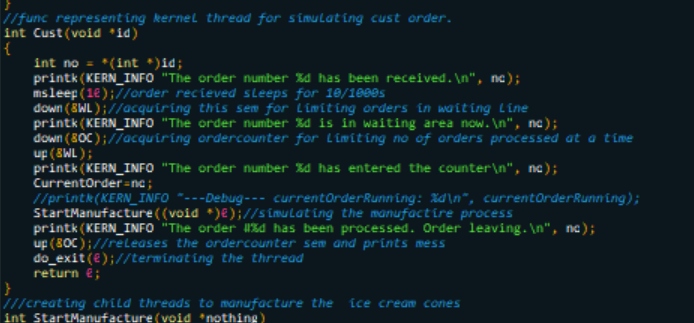
Necessary Functions:



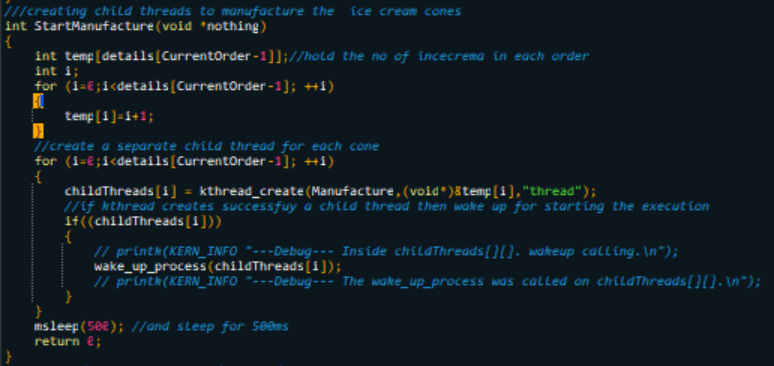
The real function:



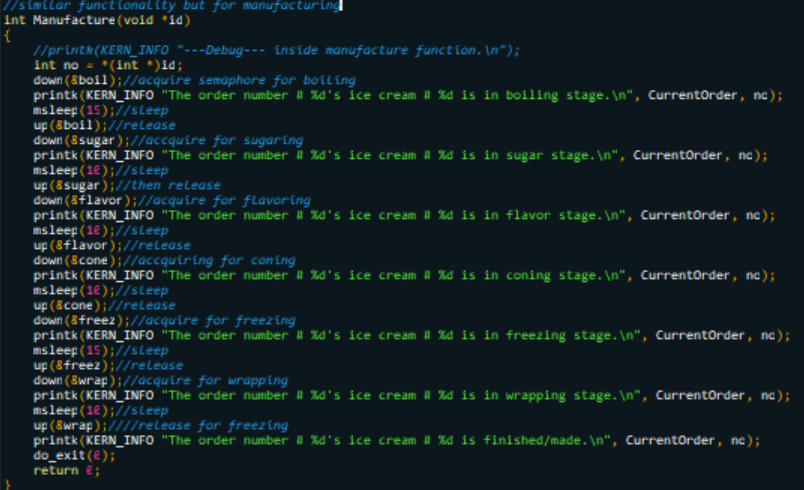
The Cust func:



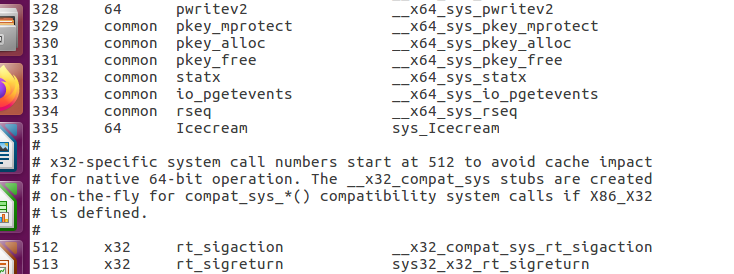
The StartManufacture func:



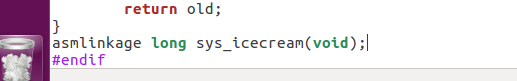
The Manufacture fun:



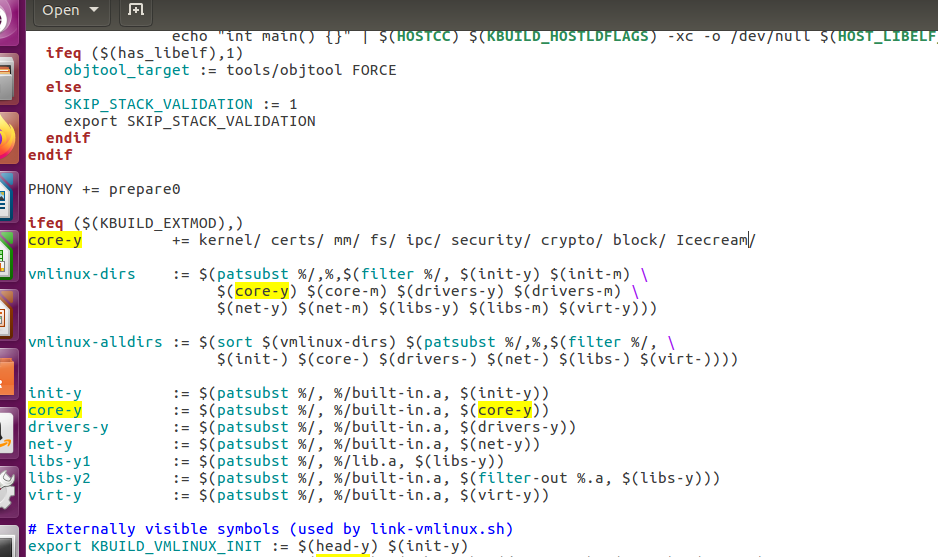
Then created a makefile. After it added the new code into the system table file:



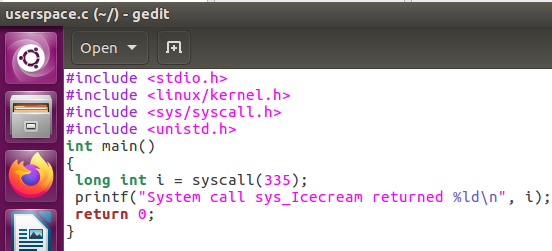
And then added the prototype of the new system call into the system calls header file:



Then, I changed version and added the Icecream folder in the kernel’s Makefile:



Then created a config file, cleaned, and compiled the kernel, installed the modules, and ran this code to check if everything works fines:

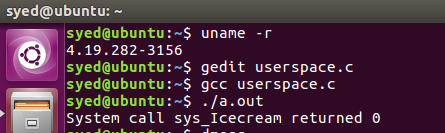


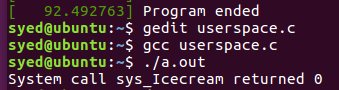
# **Results:**

Simulation results show that using semaphores can effectively prevent deadlocks in multithreaded systems. By the usage of semaphore, we ensured mutual exclusion and ensured that only one thread can access the resource at a time, and this prevented issues like race condition and data inconsistencies.

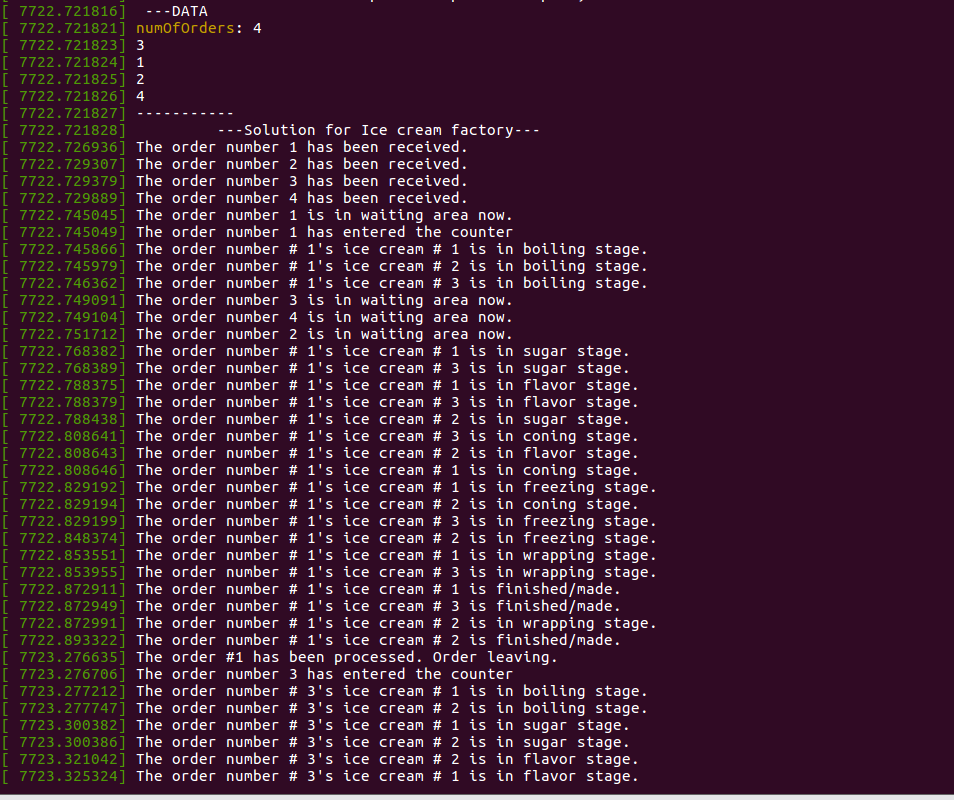
# **Output:**

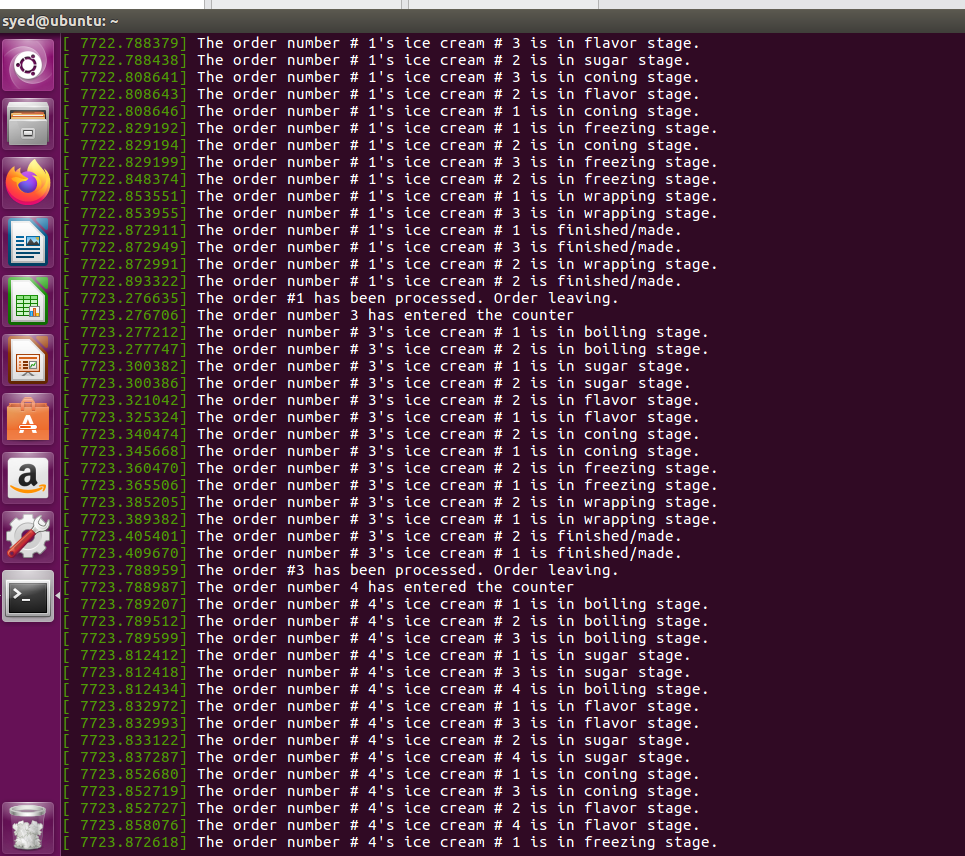
Printed my roll no. and compiled successfully since it returned 0

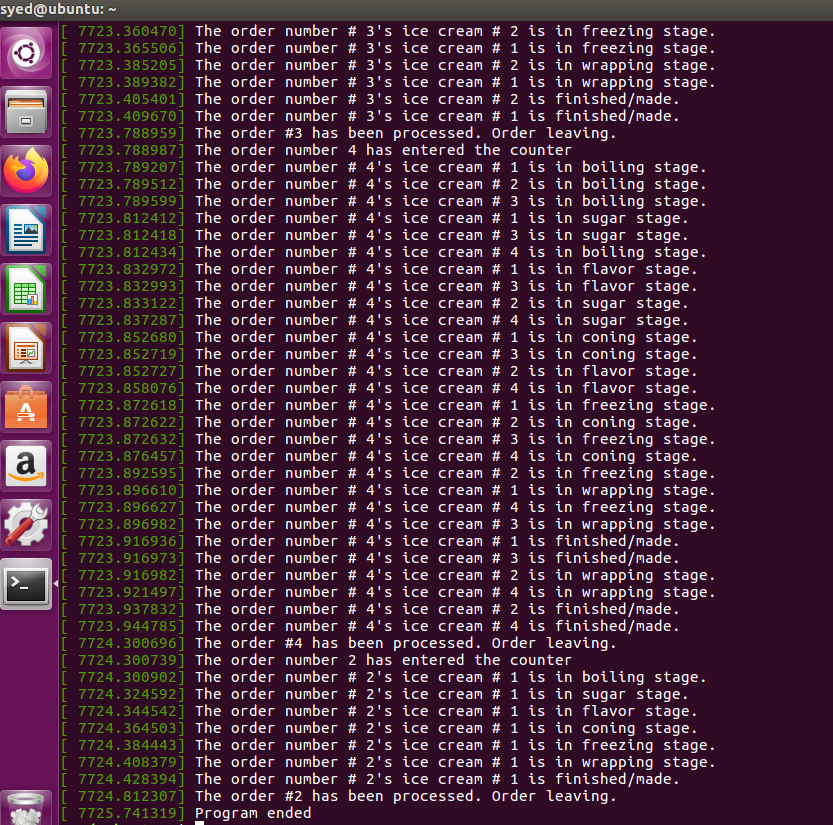




The output of code is:







**Conclusion:**  
Based on the implementation of process synchronization using semaphores in the provided code, we can conclude that semaphores can easily and effectively handle issues related to concurrent access to shared resources. This code shows how to properly use semaphores to ensure mutual exclusion, synchronization, and avoidance of race conditions between processes. Moreover, only one process can access critical section at a time. Lastly, project provided insight of kernel threads which also provided benefits. And we gain a sound learning experience of the topics discovered in this project.