

# Operating System Project Ice Cream Factory

# Group Members:

Muhammad Hamza ( 21k-3815 )

Huzaifa Awan ( 21k-3835 )

Zayn Khan( 21k-3828 )

# Table Of Contents

Contents

[Operating System Project Ice Cream Factory 1](#_Toc135395520)

[Group Members: 1](#_Toc135395521)

[Table Of Contents: 2](#_Toc135395522)

[Objective: 3](#_Toc135395523)

[Project Details: 3](#_Toc135395524)

[Results: 5](#_Toc135395525)

[Conclusion: 5](#_Toc135395526)

[Code (C file): 5](#_Toc135395527)

[Outputs: 14](#_Toc135395528)

[GIT-HUB Link: 15](#_Toc135395529)

# Objective:

Simulate an ice cream factory manufacturing process using a Linux kernel module.

Implement synchronization mechanisms such as semaphores to coordinate the manufacturing process.

Create a multi-threaded environment where customers can place orders and ice creams can be manufactured concurrently.

The project aims to provide a practical understanding of concurrent programming and resource synchronization in the context of an ice cream factory, while also highlighting the relevance and applicability of kernel modules in real-world scenarios.

# Project Details:

The project focuses on developing a simulation of an ice cream factory manufacturing process using a Linux kernel module. The implementation utilizes various synchronization mechanisms, such as semaphores, to coordinate the different stages of ice cream production and manage multiple orders concurrently.

The simulation begins by accepting a specified number of orders, each consisting of a varying number of ice creams. The maximum capacity of orders is defined by the MAX\_ORDERS constant. The project allows for customization of the number of orders by modifying the numOfOrders constant and specifying the details of each order in the orderDetails array.

To facilitate synchronization and resource management, the project employs several semaphores. The waitingLine semaphore ensures that the waiting area can only accommodate a limited number of orders at a time. The orderCounter semaphore guarantees that the counter can only handle one order at a time.

Additionally, semaphores are used for each stage of the ice cream manufacturing process. The boiler, sugar, flavor, cone, freezing, and wrapping semaphores control access to the respective stages, ensuring that only a limited number of ice creams can be processed simultaneously.

The project utilizes multiple threads to represent customers and the manufacturing process for each order. The customer function represents a customer thread, which waits in the waiting line, enters the counter, and initiates the manufacturing process by calling the startManufacture function.

The startManufacture function creates child threads, each representing the production of an individual ice cream in the order. These child threads are created using the manufacture function, which simulates the different stages of ice cream production, such as boiling, sugaring, flavoring, coning, freezing, and wrapping.

Throughout the simulation, the project ensures proper synchronization and resource management by utilizing semaphores to control access to shared resources and limit the number of concurrent processes. This enables the simulation to handle multiple orders concurrently and accurately represent the manufacturing process of an ice cream factory.

# Results:

The results of the project demonstrate the successful implementation of the ice cream factory simulation using a Linux kernel module. The simulation effectively coordinates the manufacturing process of multiple orders concurrently, ensuring proper synchronization and resource management.

During the execution of the simulation, the system accurately processes the specified number of orders, each consisting of a variable number of ice creams. The customers are placed in a waiting line, and as soon as the counter becomes available, they enter the counter and initiate the manufacturing process for their respective orders.

The manufacturing process, represented by child threads, progresses through the different stages of ice cream production, including boiling, sugaring, flavoring, coning, freezing, and wrapping. Each stage is properly synchronized using semaphores, allowing a limited number of ice creams to be processed simultaneously and preventing resource conflicts.

# Conclusion:

In conclusion, the ice cream factory simulation project has successfully achieved its objectives of creating a realistic representation of the manufacturing process using a Linux kernel module. The project effectively demonstrated the implementation of synchronization mechanisms, such as semaphores, to coordinate multiple orders concurrently and manage the different stages of ice cream production.

# Code (C file):

#include<linux/module.h> #include<linux/kernel.h> #include<linux/kthread.h> #include<linux/sched.h> #include<linux/time.h> #include<linux/timer.h> #include<linux/semaphore.h> #include<linux/delay.h>

#define MAX\_ORDERS 15

static struct semaphore waitingLine; static struct semaphore orderCounter;

static struct semaphore boiler, sugar, flavor, cone, freezing, wrapping; static struct semaphore custWait;

int allDone = 0;

int currentOrderRunning;

int Numbers[MAX\_ORDERS];

#define numOfOrders 4

int const orderDetails[numOfOrders] = {3,1,2,4};

static struct task\_struct \*childThreads[MAX\_ORDERS]; static struct task\_struct \*tid[MAX\_ORDERS];

int customer(void \*id);

int startManufacture(void \*); int manufacture(void \*id);

asmlinkage long sys\_icecream(void)

{

int i;

if (numOfOrders > MAX\_ORDERS)

{

Exiting...\n");

}

printk(KERN\_INFO "The number of orders in greater than the MAX capacity. return 0;

printk(KERN\_INFO "\t---DATA\n");

printk(KERN\_INFO "numOfOrders: %d\n", numOfOrders);

for (i = 0; i < numOfOrders; i++) printk(KERN\_INFO "%d \n", orderDetails[i]); printk(KERN\_INFO " \n");

printk(KERN\_INFO "\t\t---Solution for Ice cream factory \n");

for (i = 0; i < MAX\_ORDERS; ++i)

{

Numbers[i] = i + 1;

}

sema\_init(&waitingLine, numOfOrders); sema\_init(&orderCounter, 1);

sema\_init(&boiler, 3);

sema\_init(&sugar, 2);

sema\_init(&flavor, 2);

sema\_init(&cone, 2);

sema\_init(&freezing, 3);

sema\_init(&wrapping, 2);

sema\_init(&custWait, 0);

char our\_thread[8]="thread1"; for (i = 0; i < numOfOrders; ++i)

{

tid[i] = kthread\_create(customer,(void\*)&Numbers[i],our\_thread); if((tid[i]))

{

wake\_up\_process(tid[i]);

}

}

msleep(3000);

allDone = 1;

printk(KERN\_INFO "Program ended\n"); return 0;

}

int customer(void \*id)

{

int num = \*(int \*)id;

printk(KERN\_INFO "The order number %d has been received.\n", num); msleep(10);

down(&waitingLine);

printk(KERN\_INFO "The order number %d is in waiting area now.\n", num);

down(&orderCounter); up(&waitingLine);

printk(KERN\_INFO "The order number %d has entered the counter\n", num); currentOrderRunning = num;

startManufacture((void \*)0);

printk(KERN\_INFO "The order #%d has been processed. Order leaving.\n", num); up(&orderCounter);

do\_exit(0);

return 0;

}

int startManufacture(void \*nothing)

{

int tempNumbers[orderDetails[currentOrderRunning - 1]]; int i;

for (i = 0; i < orderDetails[currentOrderRunning - 1]; ++i)

{

tempNumbers[i] = i+1;

}

for (i = 0; i < orderDetails[currentOrderRunning-1]; ++i)

{

childThreads[i] = kthread\_create(manufacture,(void\*)&tempNumbers[i],"thread");

if((childThreads[i]))

{

wake\_up\_process(childThreads[i]);

}

}

msleep(500);

return 0;

}

int manufacture(void \*id)

{

int num = \*(int \*)id;

down(&boiler);

printk(KERN\_INFO "The order number # %d's ice cream # %d is in boiling stage.\n", currentOrderRunning, num);

msleep(15); up(&boiler);

down(&sugar);

printk(KERN\_INFO "The order number # %d's ice cream # %d is in sugar stage.\n", currentOrderRunning, num);

msleep(10); up(&sugar);

down(&flavor);

printk(KERN\_INFO "The order number # %d's ice cream # %d is in flavor stage.\n", currentOrderRunning, num);

msleep(10); up(&flavor);

down(&cone);

printk(KERN\_INFO "The order number # %d's ice cream # %d is in coning stage.\n", currentOrderRunning, num);

msleep(10); up(&cone);

down(&freezing);

printk(KERN\_INFO "The order number # %d's ice cream # %d is in freezing stage.\n", currentOrderRunning, num);

msleep(15); up(&freezing);

down(&wrapping);

printk(KERN\_INFO "The order number # %d's ice cream # %d is in wrapping stage.\n", currentOrderRunning, num);

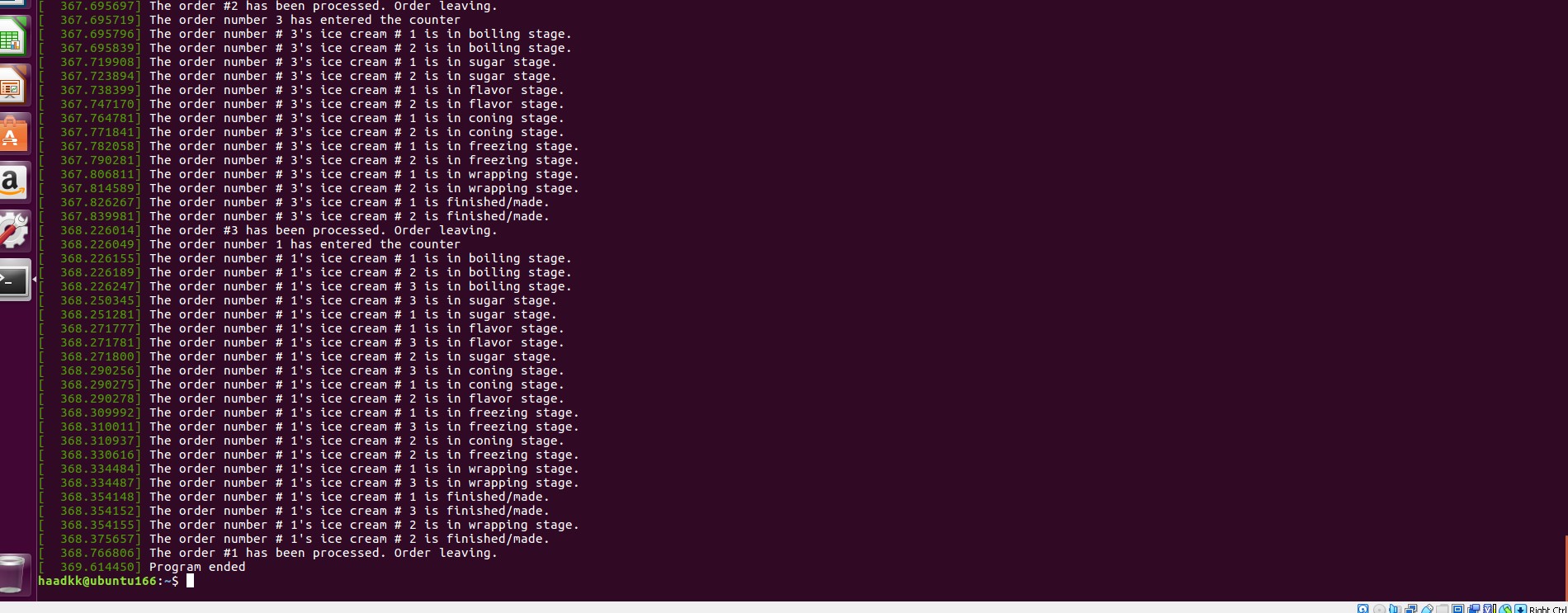
msleep(10); up(&wrapping);

printk(KERN\_INFO "The order number # %d's ice cream # %d is finished/made.\n", currentOrderRunning, num);

do\_exit(0); return 0;

}

# Outputs:



# GIT-HUB Link:

<https://github.com/hamzaowaisog/ICE-CREam-FACTORY>