****

**National University of Computer & Emerging Sciences,**

**Karachi Campus**

**Course Title:**

CS220-Operating System

**Project Title:**

System Call for Ice Cream Factory Problem

**Group Members:**

19K-0264 Abdul Bari

19K-0252 Ariba

19K-1261 Hamza Ayub

**OBJECTIVE:**

Basically, this project is about the simulation of an ice cream factory where multiple customers are handled and scheduled to different counters accordingly.

**PROJECT DESCRIPTION:**

Firstly, the system call prints a string and customer ID according to the different activities performed by the customer such as departure or arrival of customer. Furthermore, the problem of occurrence of deadlock is solved by using threads and semaphores.

The flow of the program is as follows:

* First, Customer gets an ID
* Then, he moves towards ticket counter and gets his ticket checked
* After that, progress towards flavor counter and chooses a flavor
* Afterwards, he proceeds towards topping counter and chooses a topping
* Next, departs for the payment counter,
* At last, he exits.

**DESCRIPTION OF SOURCE CODE:**

Firstly, we initialize the prices of flavors and topping, number of tickets, flavor, and toppings, semaphores and the prototype of the function.

In main function, value of number of customer has to be inputted and each customer is allocated a unique ID. Then, each semaphore of counters, flavors and toppings are initialized. Next, thread array of size ‘number of customers’ is created, and for each customer, function ‘icecream’ is called.

In ‘icecream’ function, sem\_wait function is called so that the newly arrived customer waits on the ticket counter. After that, a condition is checked to verify if number of tickets are greater than 0. If(tickets < 0), then accept the ticket, decrement the value of ticket and depart from ticket counter using sem\_post function. Else, exit program. Next, we head towards flavor counter to choose flavor. This shall be handled through local thread variable. If no flavors are remaining, i.e. if(flavor[0] == 0 && flavor[1] == 0 && flavor[2] == 0) is true, pthread\_exit() shall be called. It is important to individually check the remaining flavors. If(flavor[0] <= 0) is true, the customer needs to leave. Else, the flavor is selected. This condition has to be checked for flavor[1] and flavor[2] as well. Now, the customer leads towards topping counter, choosing the topping and leaves the respective counter. If there is no topping left, he proceeds towards payment counter and the bill is generated. The ‘icecream’ function ends here.

After returning to the main function, all the threads are joined and we the number of customers, total revenue, and the number of tickets remaining are printed. At last, we destroy all the semaphores.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#include <semaphore.h>

#include <string.h>

#include <sys/syscall.h>

#include <linux/kernel.h>

#define priceFlav\_1 1.05

#define priceFlav\_2 2.00

#define priceFlav\_3 1.67

#define priceTopp\_1 0.8

#define priceTopp\_2 0.5

// resources

int ticket = 30, \_flavors[3] = {29, 34, 18}, \_toppings[2] = {20, 34};

double revenue = 0.0;

// semaphores declaration

sem\_t ticketC, flavorC, f1, f2, f3, toppingC, t1, t2, paymentC;

// function prototypes

void \*iceCream(void \*\_id);

// main

int main()

{

int noC;

// 101="Leaving Shop. [REASON]: Tickets Finished\n"

// 102="Got Ticket.\n"

// 103="Leaving Shop. [REASON]: Flavours Finished\n"

// 104="Got Flavour[0].\n"

// 105="Got Flavour[1].\n"

// 106="Got Flavour[2].\n"

// 107="Got Flavour(s). Leaving Flavor Counter\n"

// 108="Leaving Topping Counter.\n"

// 109="Leaving Ice-Cream Shop.\n\n"

printf("\n\nEnter Number Of Customers [1-%d]: ", ticket);

scanf("%d", &noC);

printf("\n\n");

if(noC > ticket || noC <= 0)

{

printf("\n\nInvalid Input!\n\n");

return 0;

}

int \_id[noC];

for(int i=0; i<noC; i++)

{

\_id[i] = i+100;

}

// semaphore initialization - START

sem\_init(&ticketC, 0, 1);

sem\_init(&flavorC, 0, 3);

sem\_init(&f1, 0, 1);

sem\_init(&f2, 0, 1);

sem\_init(&f3, 0, 1);

sem\_init(&toppingC, 0, 2);

sem\_init(&t1, 0, 1);

sem\_init(&t2, 0, 1);

sem\_init(&paymentC, 0, 1);

// semaphore initialization - END

// multithreading region - START

pthread\_t \_customer[noC];

for(int i=0; i<noC; i++)

{

pthread\_create(&\_customer[i], 0, &iceCream, (void\*) &\_id[i]);

}

for(int i=0; i<noC; i++)

{

pthread\_join(\_customer[i], NULL);

}

// multithreading region - END

printf("\n\nBusiness Journal - At Closing\n\n");

printf("Number Of Customers: %d", noC);

printf("\nRevenue Generated: $ %f", revenue);

printf("\nTickets Remaining: %d\n\n", ticket);

// semaphore destroying - START

sem\_destroy(&ticketC);

sem\_destroy(&flavorC);

sem\_destroy(&f1);

sem\_destroy(&f2);

sem\_destroy(&f3);

sem\_destroy(&toppingC);

sem\_destroy(&t1);

sem\_destroy(&t2);

sem\_destroy(&paymentC);

// semaphore destroying - END

return 0;

}

void \*iceCream(void \*\_id)

{

int \_ID = \*(int\*)\_id, checkRaceCond\_1 = 0;

double bill = 0.0;

// ticket counter - ENTER

sem\_wait(&ticketC);

if(ticket <= 0)

{

//printf("\nCustomer[%d]: Leaving Shop. [REASON]: Tickets Finished\n", \_ID);

syscall(332,101,\_ID);

sleep(1);

pthread\_exit(NULL);

}

ticket--;

//printf("Customer[%d] Got Ticket.\n", \_ID);

syscall(332,102,\_ID);

sem\_post(&ticketC);

// ticket counter - EXIT

sleep(2);

// flavours counter - ENTER

sem\_wait(&flavorC);

// Race Condition will occur here but it will be handled through thread Local Variable

// [REASON]: when this condition is checked by 4th thread

// and then \_flavours[i] is decremented by any of first 2 threads

// a thread will proceed even though flavors have been finished.

// WHY THIS CONDITION IS USED WHEN ITS WORK IS DONE BY LOCAL THREAD VARIABLE?

// [REASON]: If this is not used then a thread will check each condition below

// which will be time consuming.

if(\_flavors[0] <= 0 && \_flavors[1] <= 0 && \_flavors[2] <= 0)

{

syscall(332,103, \_ID);

sleep(1);

pthread\_exit(NULL);

}

else

{

// flavor1

sem\_wait(&f1);

if(\_flavors[0] > 0)

{

\_flavors[0]--;

checkRaceCond\_1++;

bill = bill + priceFlav\_1;

//printf("\nCustomer[%d]: Got Flavour[0].\n", \_ID);

syscall(332,104,\_ID);

sleep(1);

}

sem\_post(&f1);

// flavor2

sem\_wait(&f2);

if(\_flavors[1] > 0)

{

\_flavors[1]--;

checkRaceCond\_1++;

bill = bill + priceFlav\_2;

//printf("Customer[%d]: Got Flavour[1].\n", \_ID);

syscall(332,105,\_ID);

sleep(1);

}

sem\_post(&f2);

// flavor3

sem\_wait(&f3);

if(\_flavors[2] > 0)

{

\_flavors[2]--;

checkRaceCond\_1++;

bill = bill + priceFlav\_3;

//printf("Customer[%d]: Got Flavour[2].\n", \_ID);

syscall(332,106,\_ID);

sleep(1);

}

sem\_post(&f3);

// if any of the threads did not get any flavour,

// 'checkRaceCond\_1' will remain 0,

// threads will exit

if(checkRaceCond\_1 == 0)

{

//printf("\nCustomer[%d]: Leaving Shop. [REASON]: Flavours Finished\n", \_ID);

syscall(332,103,\_ID);

sleep(1);

pthread\_exit(NULL);

}

}

//printf("\nCustomer[%d]: Got Flavour(s). Leaving Flavor Counter\n", \_ID);

syscall(332,107,\_ID);

//sleep(2);

sem\_post(&flavorC);

// flavors counter - EXIT

// toppings counter - ENTER

sem\_wait(&toppingC);

// topping1

sem\_wait(&t1);

if(\_toppings[0] > 0)

{

\_toppings[0]--;

bill = bill + priceTopp\_1;

}

sem\_post(&t1);

// topping2

sem\_wait(&t2);

if(\_toppings[1] > 0)

{

\_toppings[1]--;

bill = bill + priceTopp\_2;

}

sem\_post(&t2);

//printf("\nCustomer[%d]: Leaving Topping Counter.\n", \_ID);

syscall(332,108,\_ID);

sem\_post(&toppingC);

// toppings counter - EXIT

sleep(2);

// payments counter - ENTER

sem\_wait(&paymentC);

revenue = revenue + bill;

printf("\nCustomer[%d]: Billed: $ %f.\n", \_ID, bill);

// char a[100];

// sprintf(a,"Billed: $ %f.\n",bill);

// syscall(332,109,\_ID);

sem\_post(&paymentC);

// payments counter - EXIT

sleep(2);

//printf("\nCustomer[%d]: Leaving Ice-Cream Shop.\n\n", \_ID);

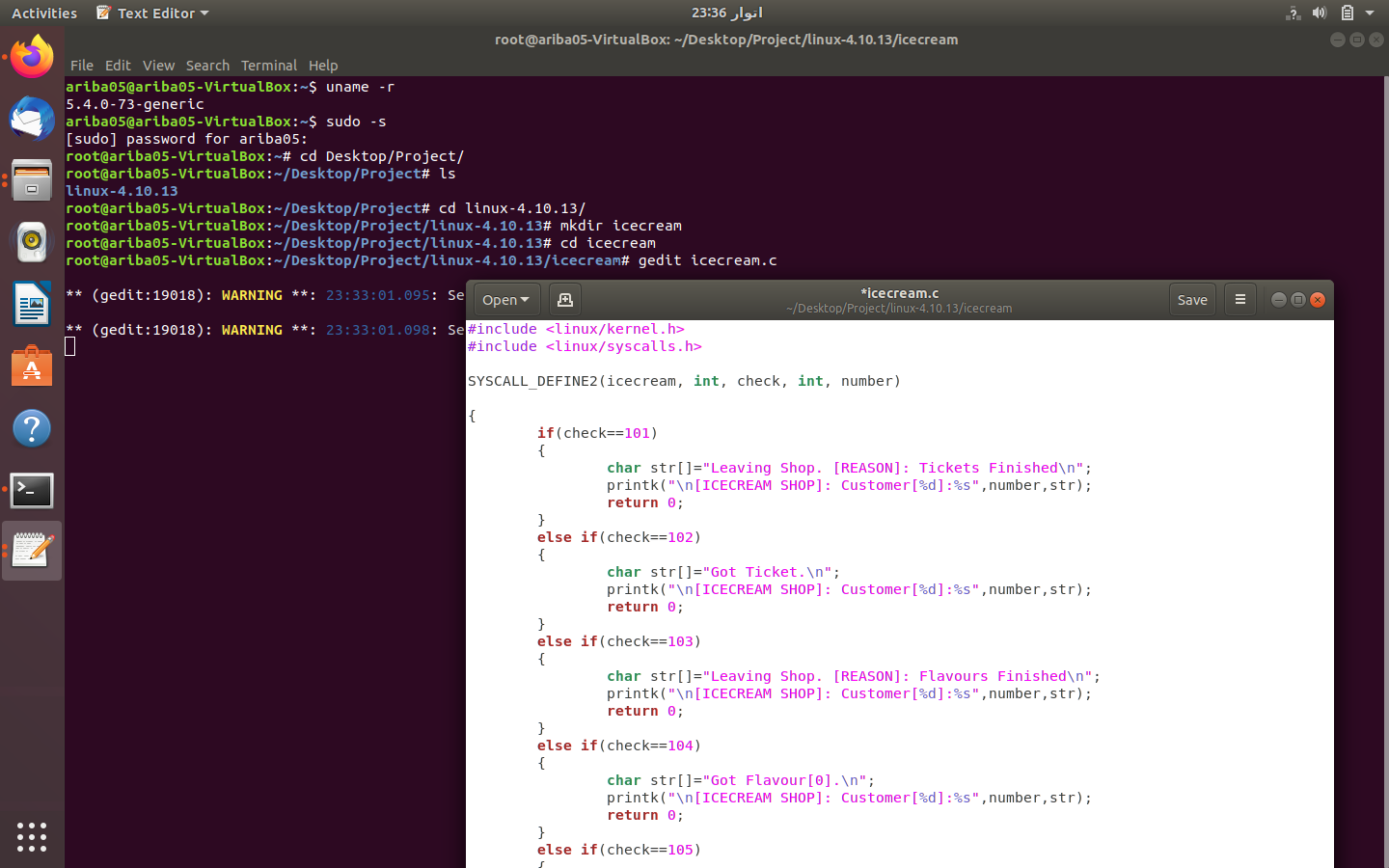
syscall(332,109,\_ID);

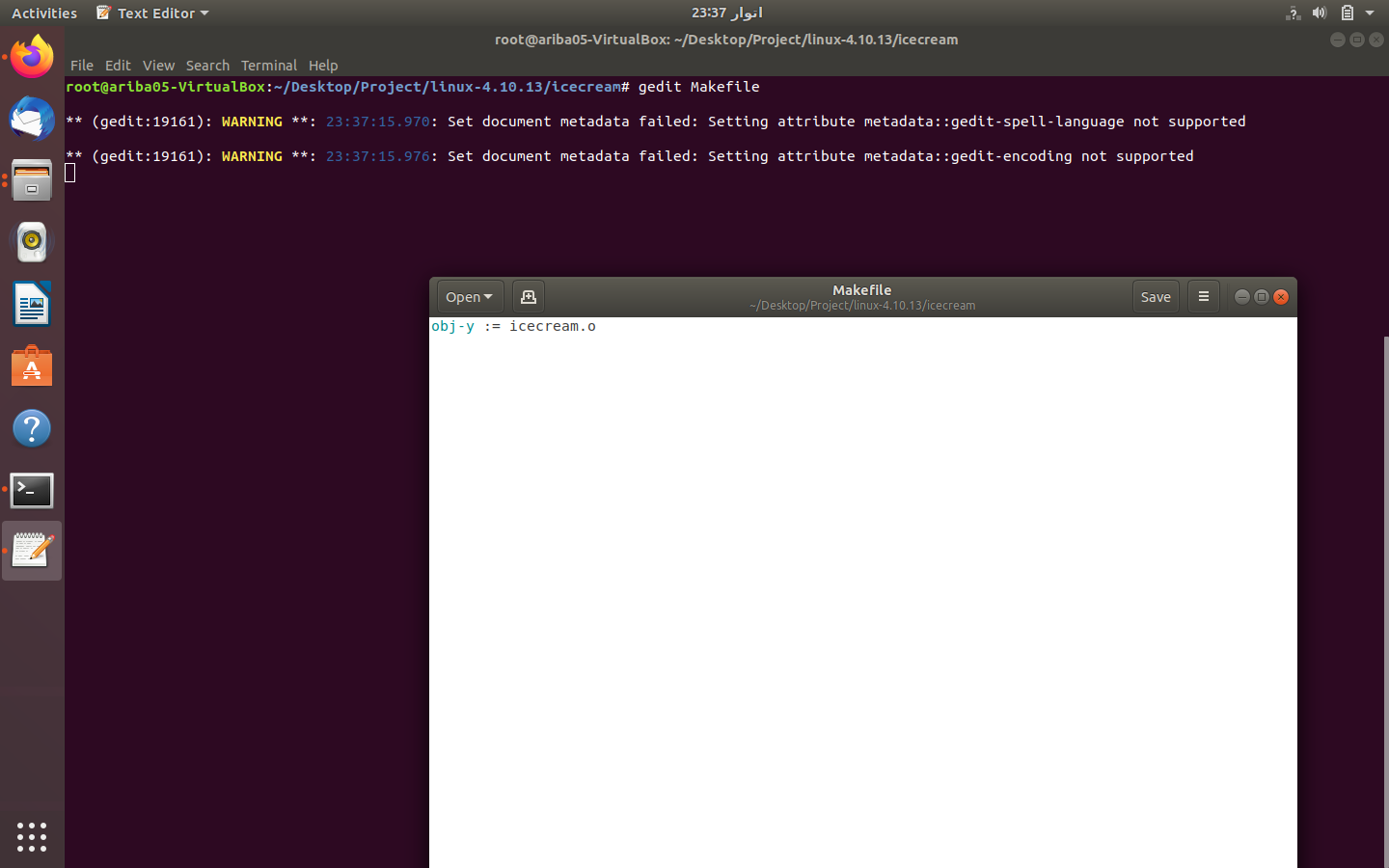
return NULL;

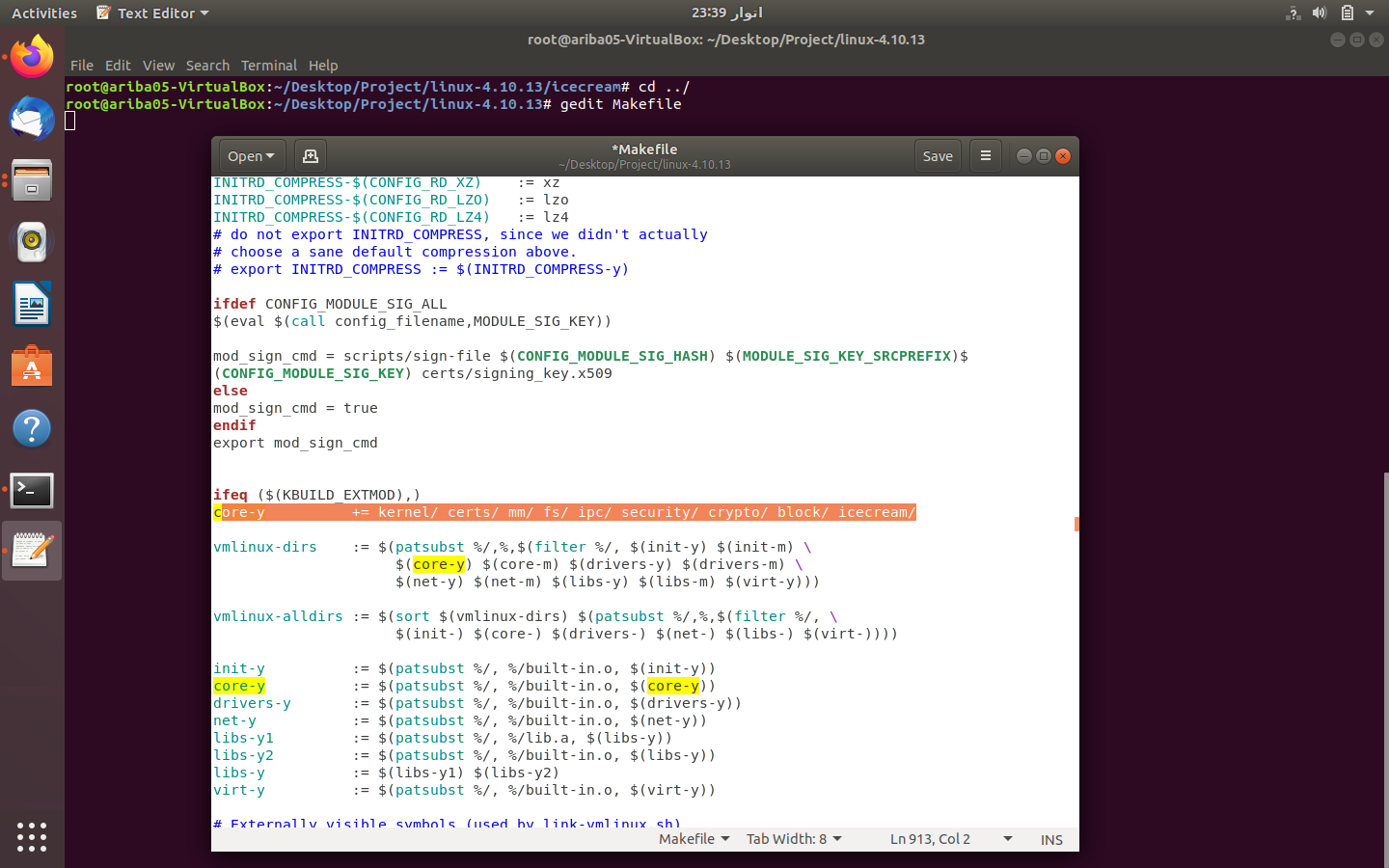
}

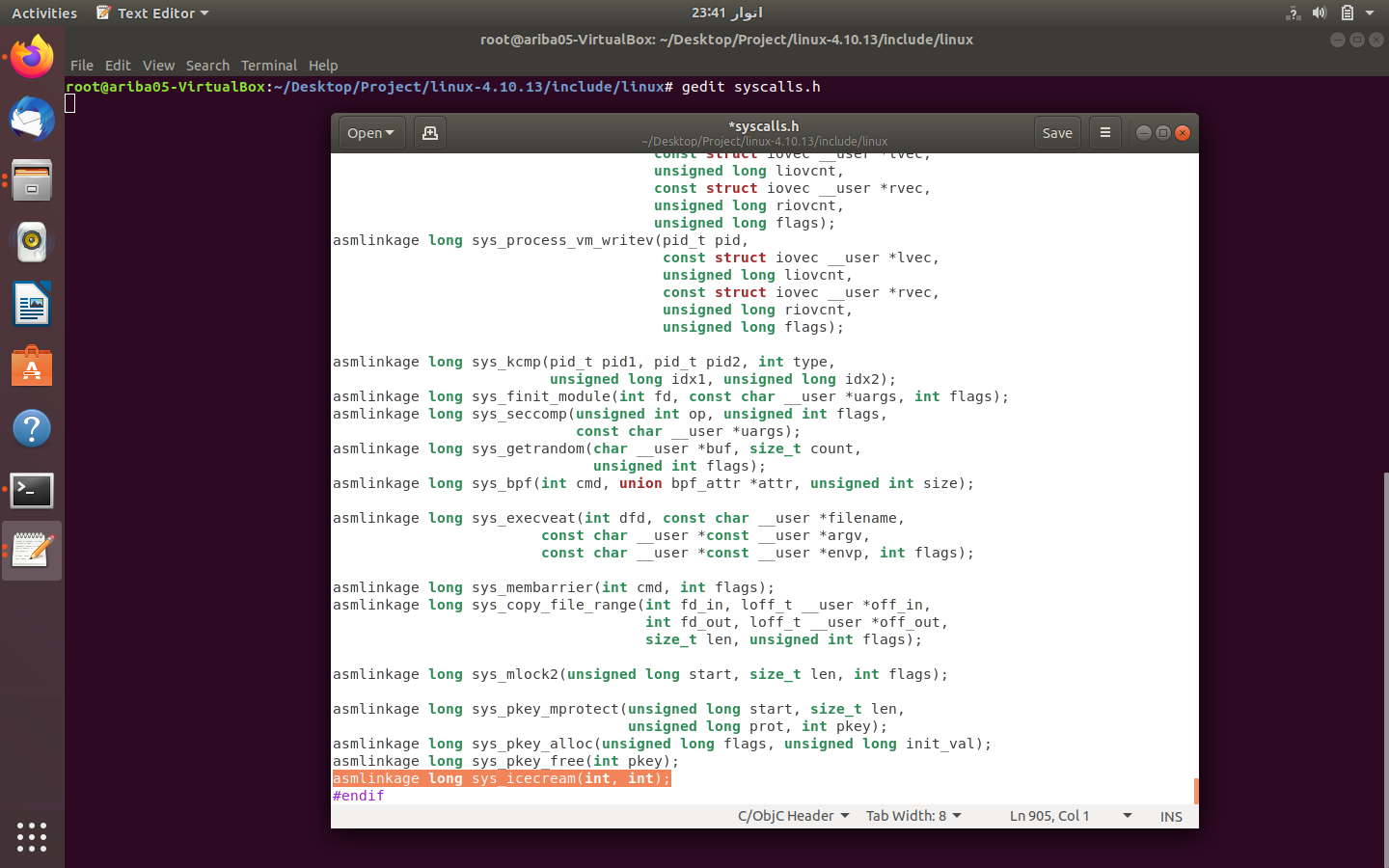
**SCREENSHOTS OF PROJECT:**

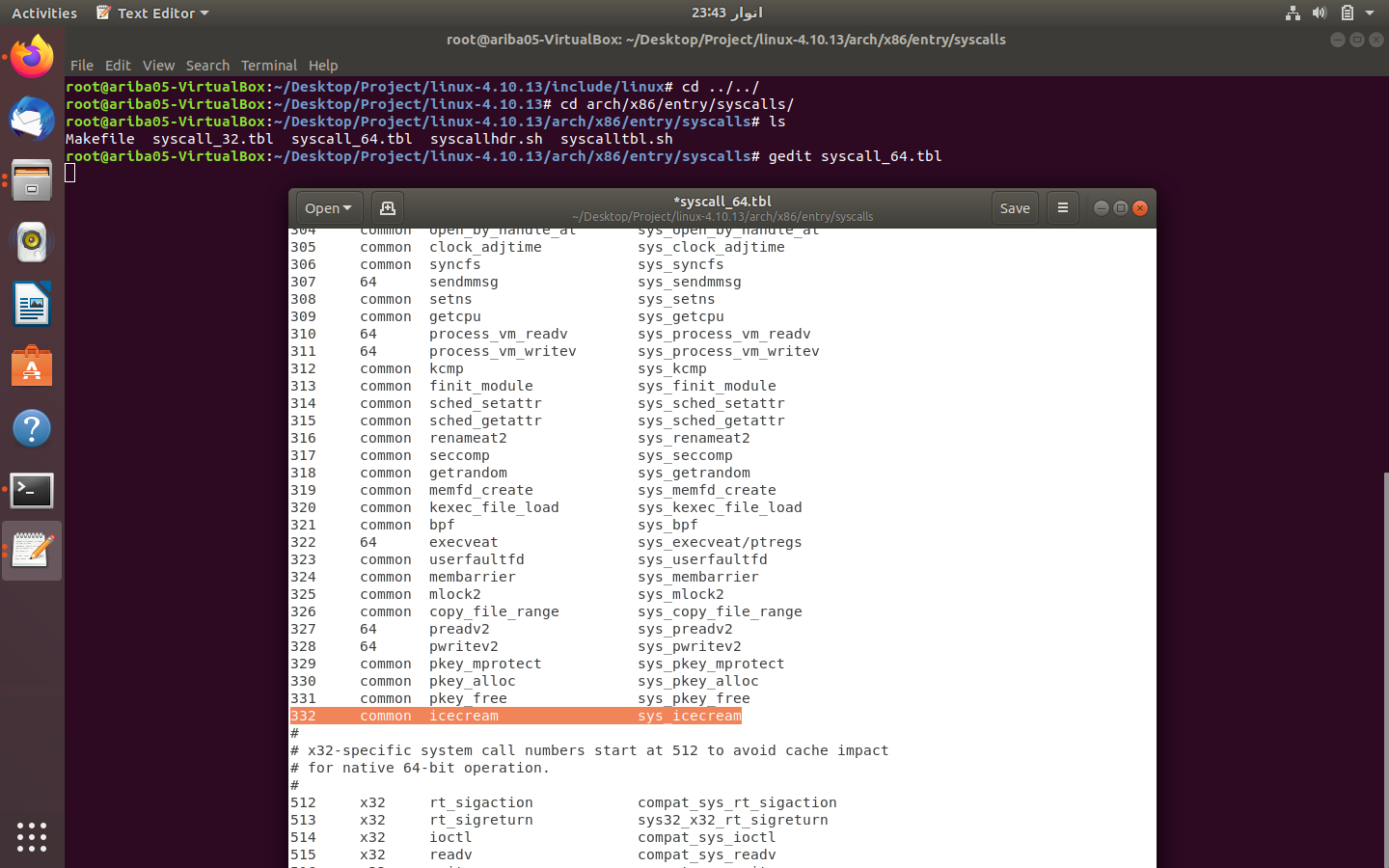
This section contains screenshots of building system call for ice cream factory problem and executing the program.

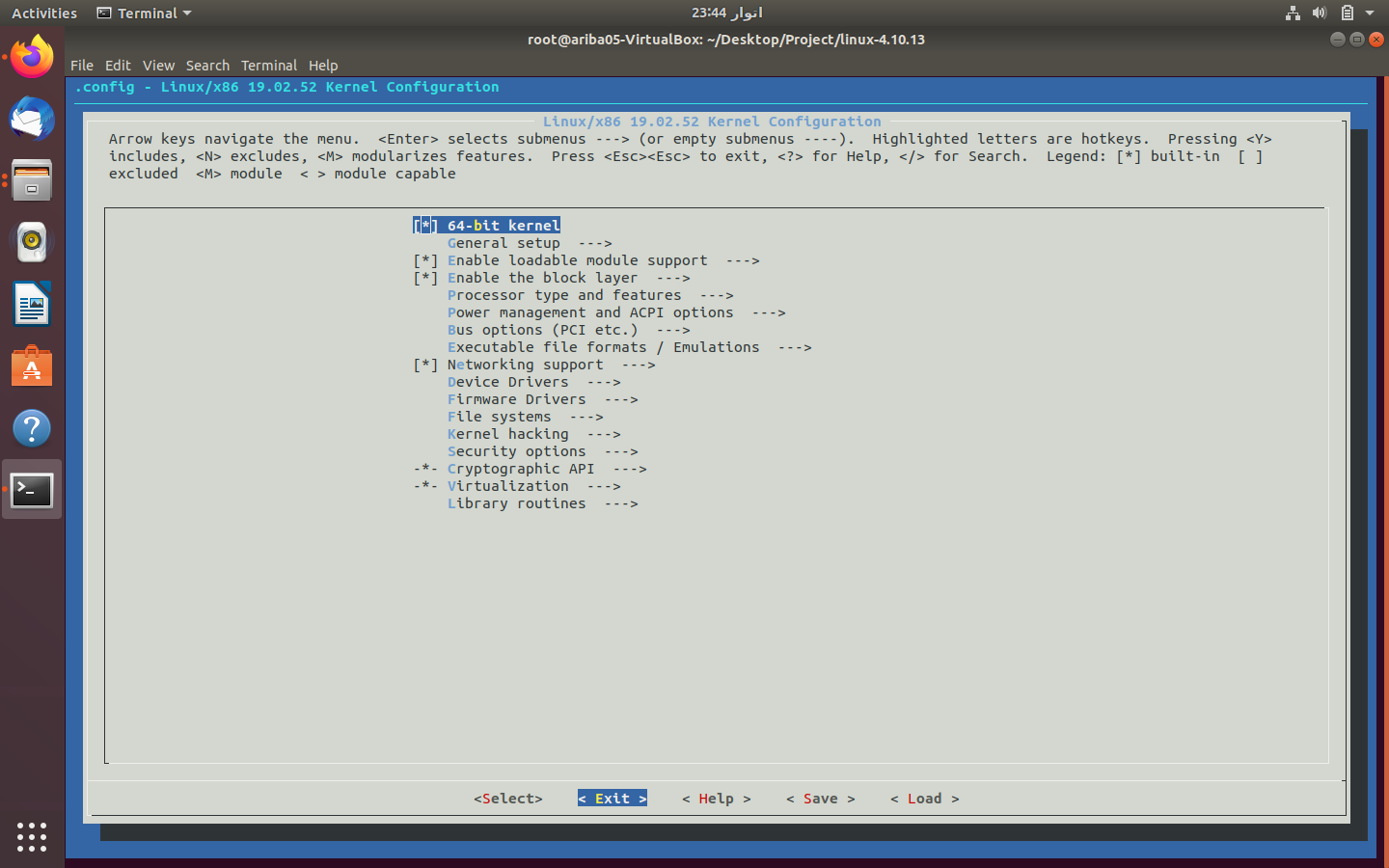
****

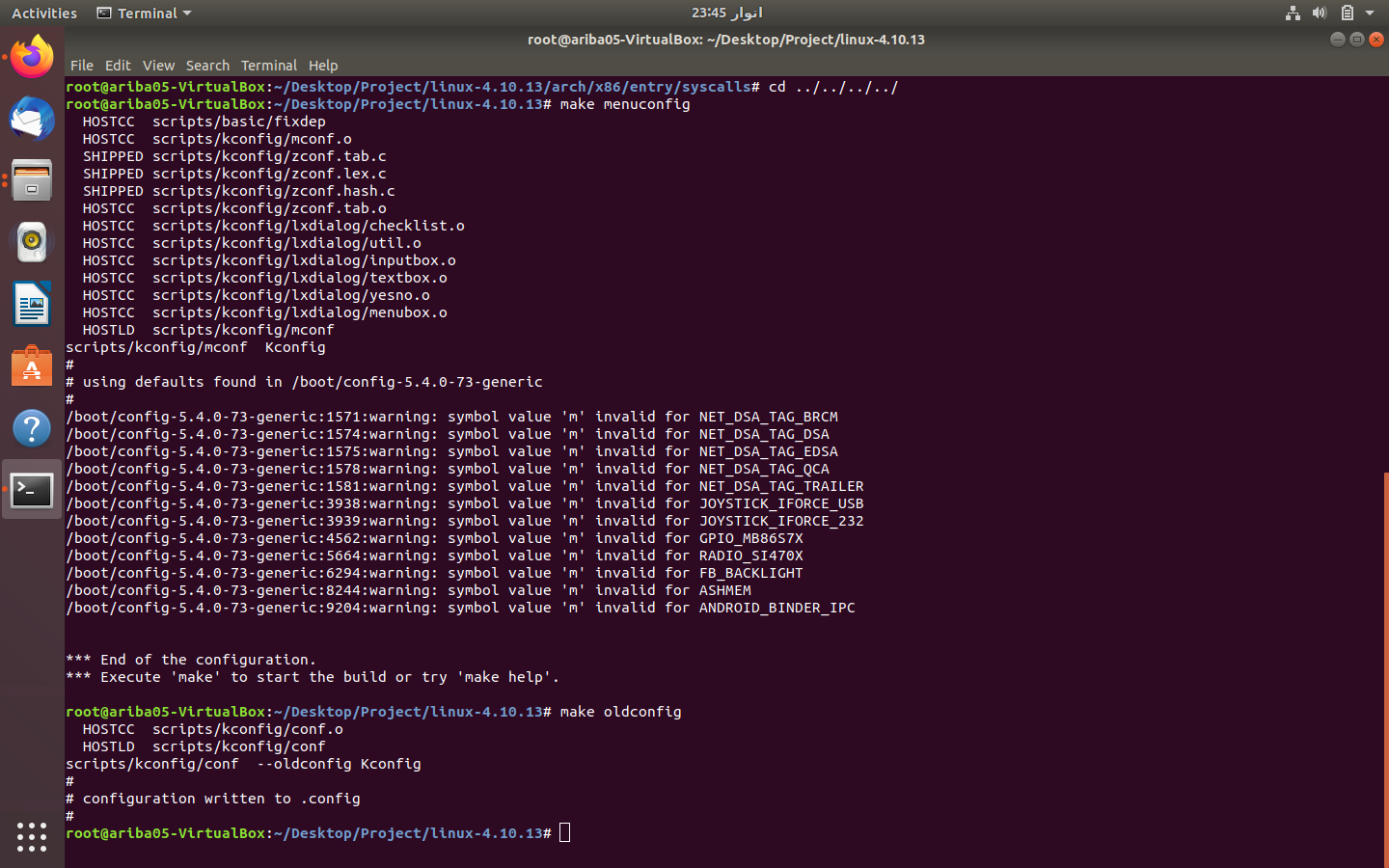
****

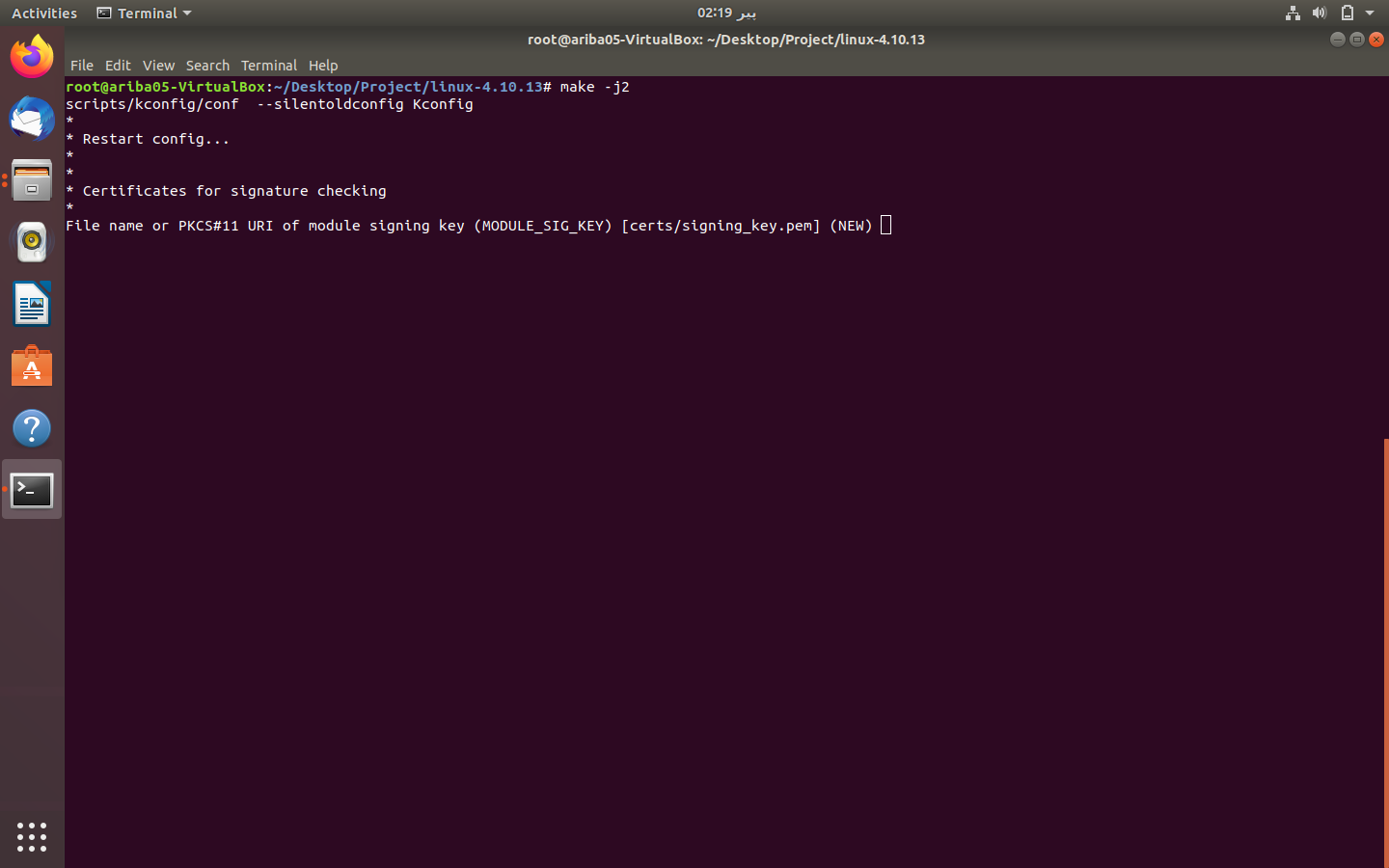
****

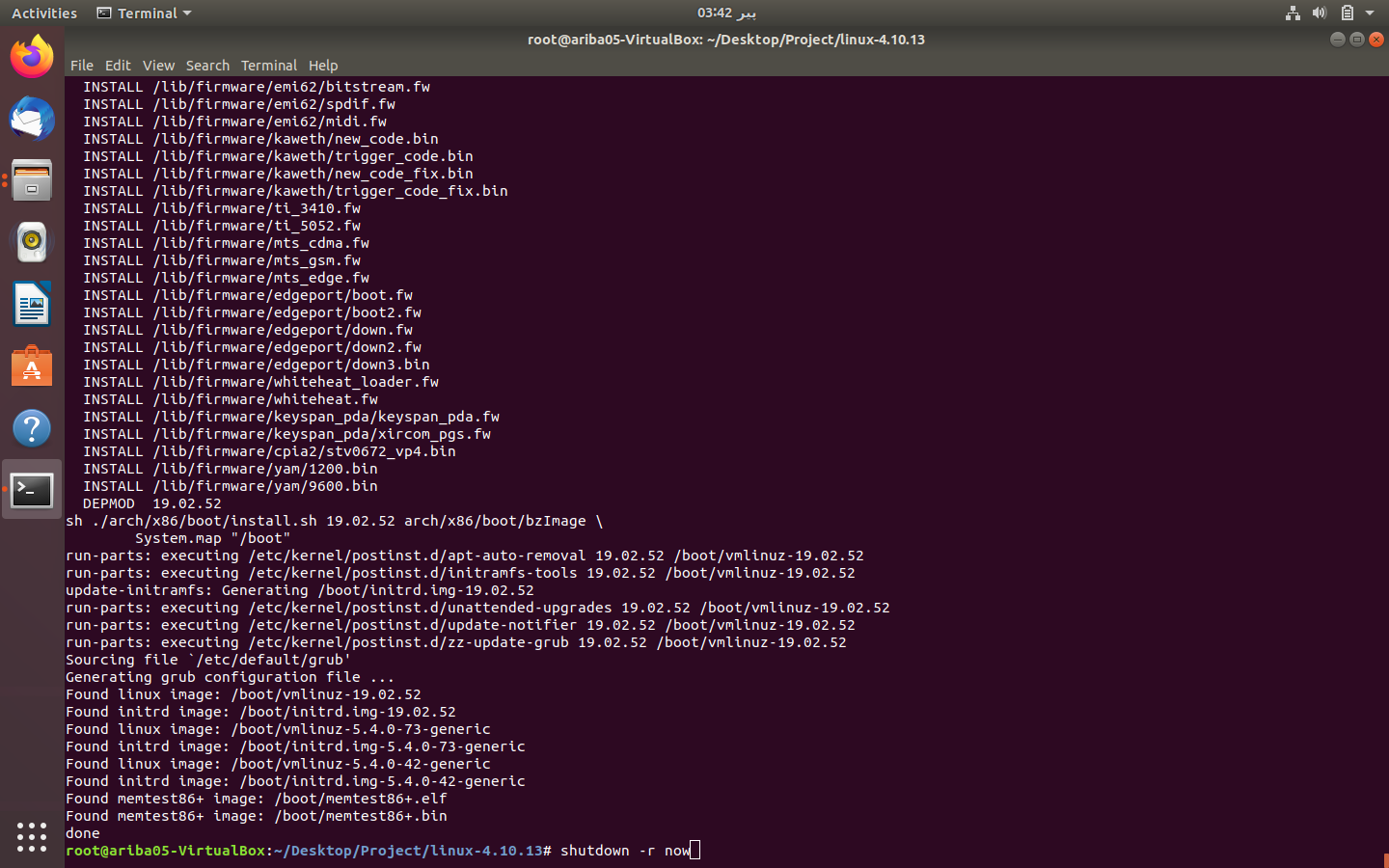
****

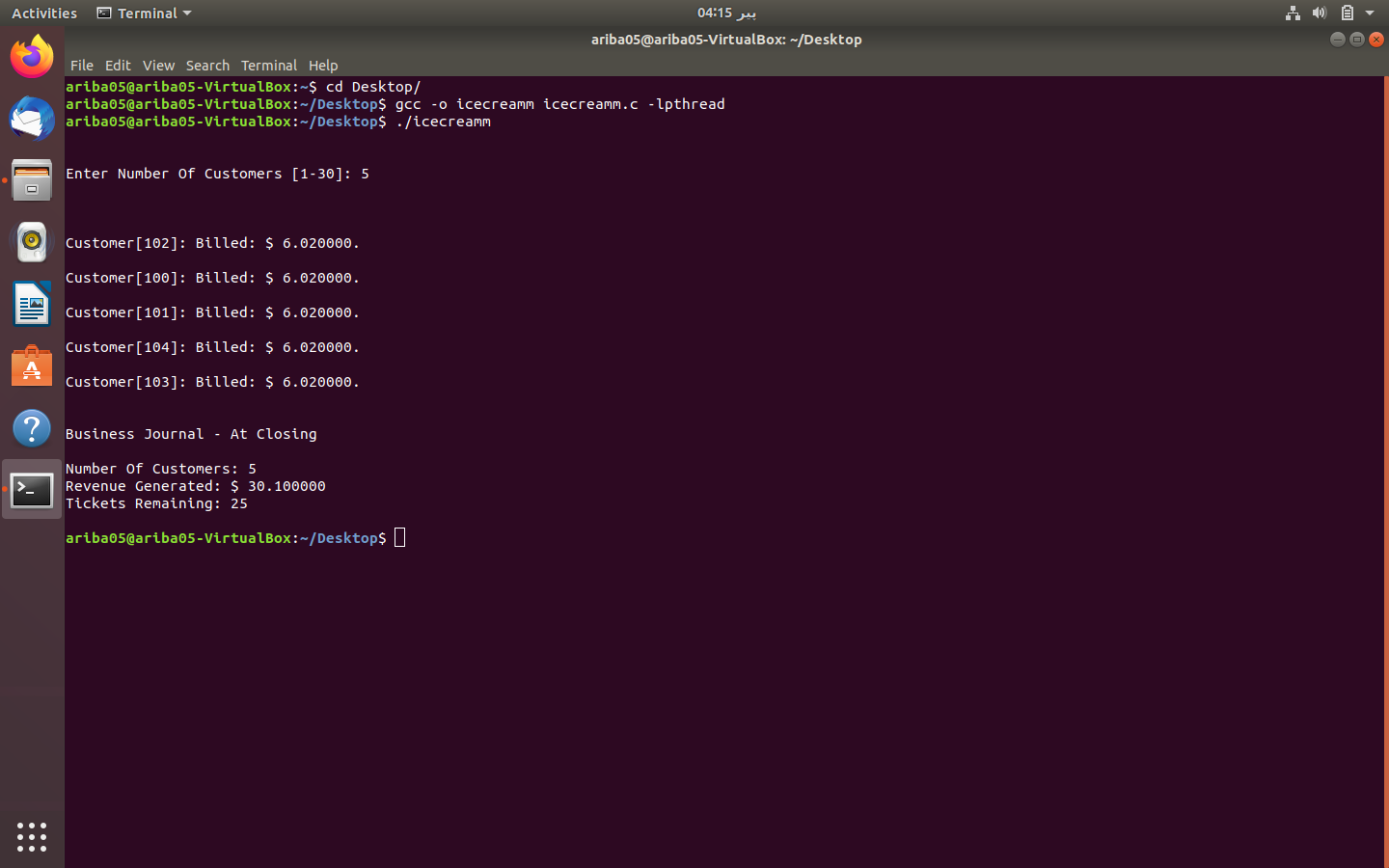
****

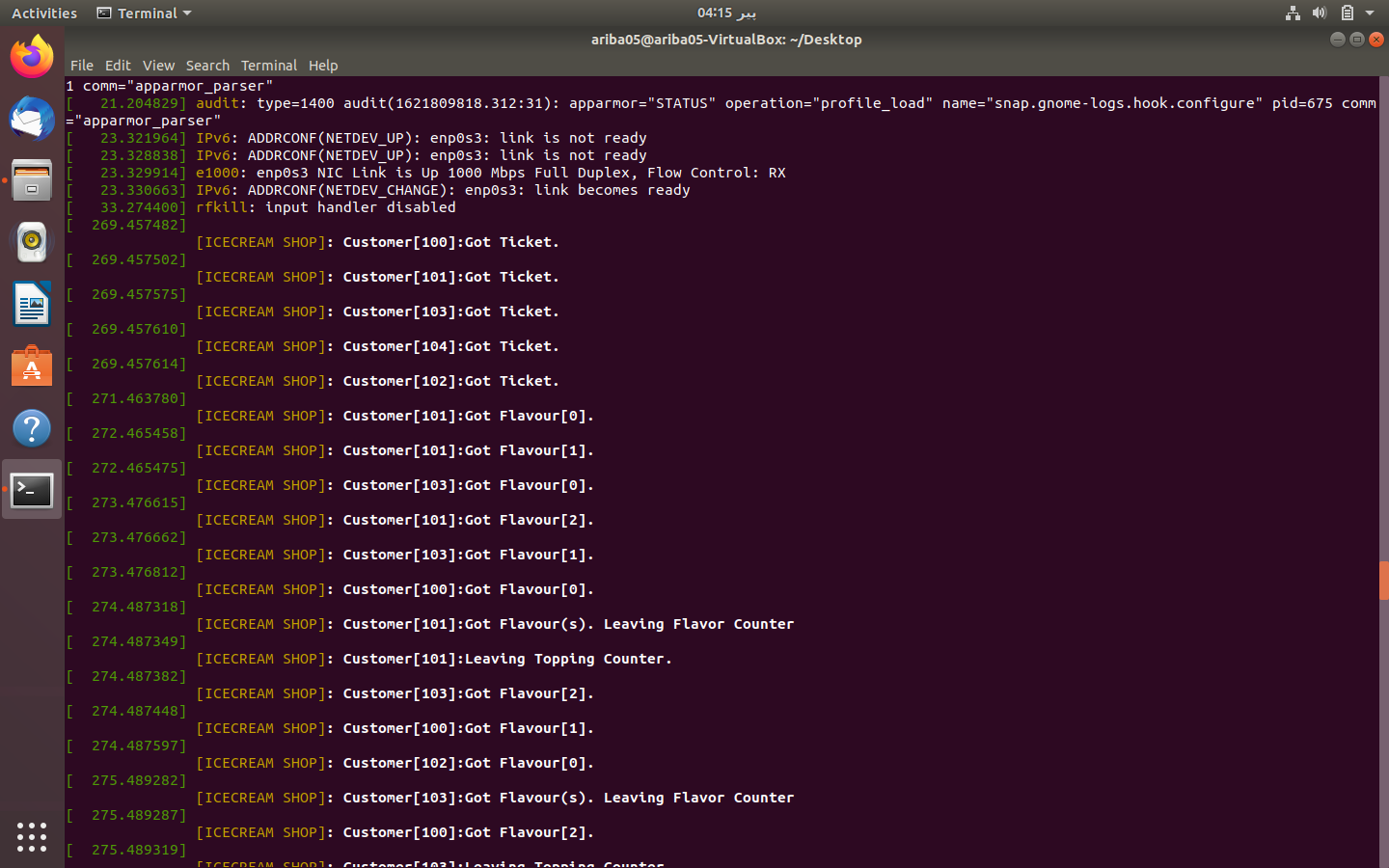
****

****

****

****

****

****