

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Υποχρεωτικό Μάθημα 4ου εξαμήνου

Εαρινό Εξάμηνο 2023-2024

Προγραμματιστική Εργασία

Μέλη Ομάδας	
Ονοματεπώνυμο	Αριθμός Μητρώου
Παπαγεωργίου Μάριος	3190156
Κατσαμπούκα Βικτωρία	3190254

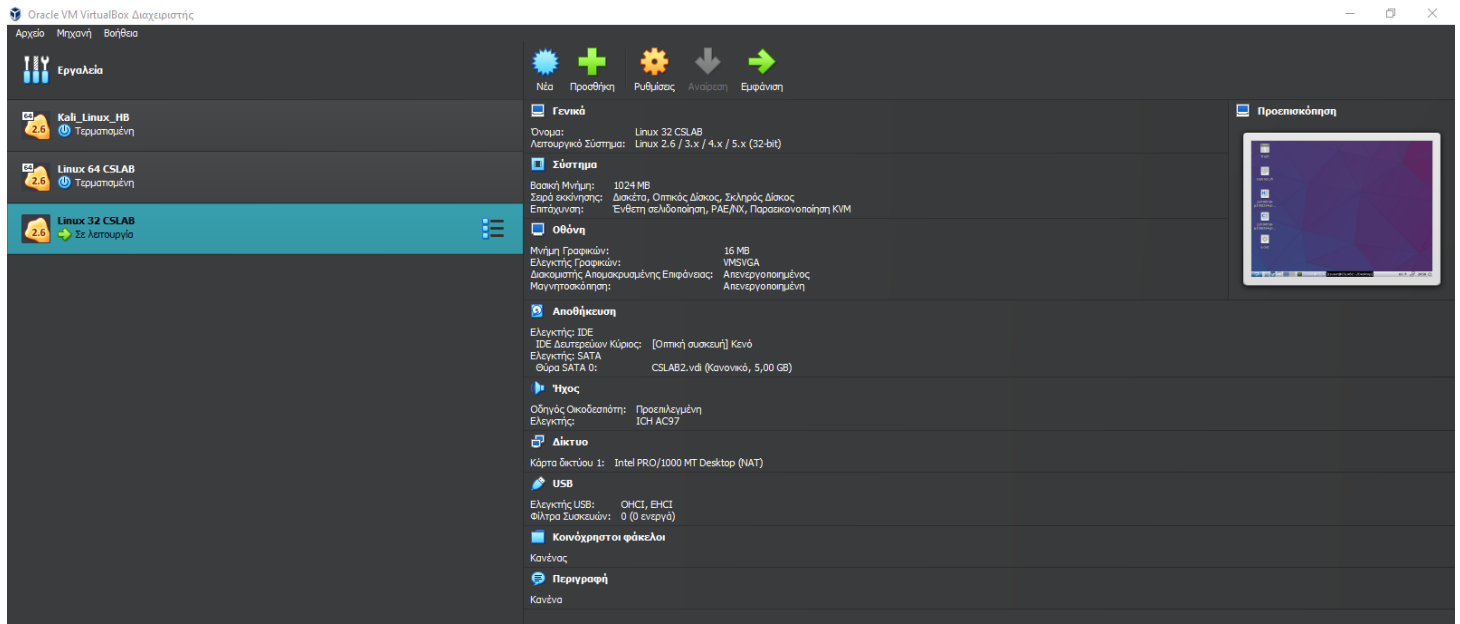
Παραδοτέα / Περιεχόμενο .zip αρχείου

- 1) **p3190156-p3190254-pizza.h**: Αρχείο δηλώσεων που περιλαμβάνει όλες τις απαραίτητες εισόδους και δεδομένα μας για την εκτέλεση του προγράμματος. Στην περίπτωση μας περιλαμβάνει κάποιες επιπρόσθετες μεταβλητές και μεθόδους που χρησιμοποιήθηκαν.
- 2) **p3190156-p3190254-pizza.c**: Εκτελέσιμος κώδικας C που χρησιμοποιεί ως αρχείο δηλώσεων το p3190156-p3190254-pizza.h. Μέσα σε αυτόν υπάρχει και η μέθοδος main που τρέχει τον κώδικα με παρεχόμενες από τον χρήστη εισόδους
- 3) **p3190156-p3190254-pizza.pdf**: Παρών έγγραφο. Αναλυτική παρουσίαση εργασίας
- 4) **test-res.sh**: Υπεύθυνο για τη μεταγλώττιση και εκτέλεση του προγράμματος μας

Ως ομάδα στόχος μας, πέρα του να φτιάξουμε ένα πρόγραμμα που να εκτελεί τα ζητούμενα της άσκησης, ήταν να γράψουμε έναν κώδικα που θα μπορούσε να είναι όσο πιο κατανοητός και εύκολος σε χρήση για τον οποιοδήποτε χρήστη.

Στην παρουσίαση αυτή θα κοιτάξουμε αναλυτικά όλα τα βήματα που ακολουθήσαμε για να πετύχουμε τα ζητούμενα της άσκησης.

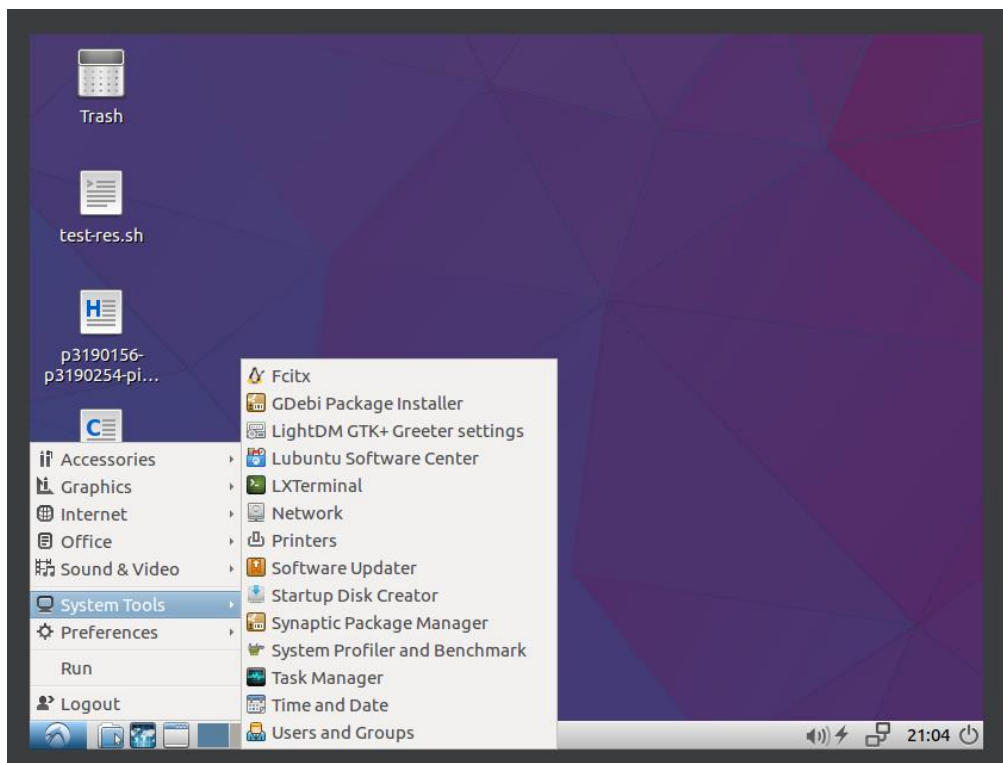
?) Πού εκτελέσαμε το πρόγραμμα μας

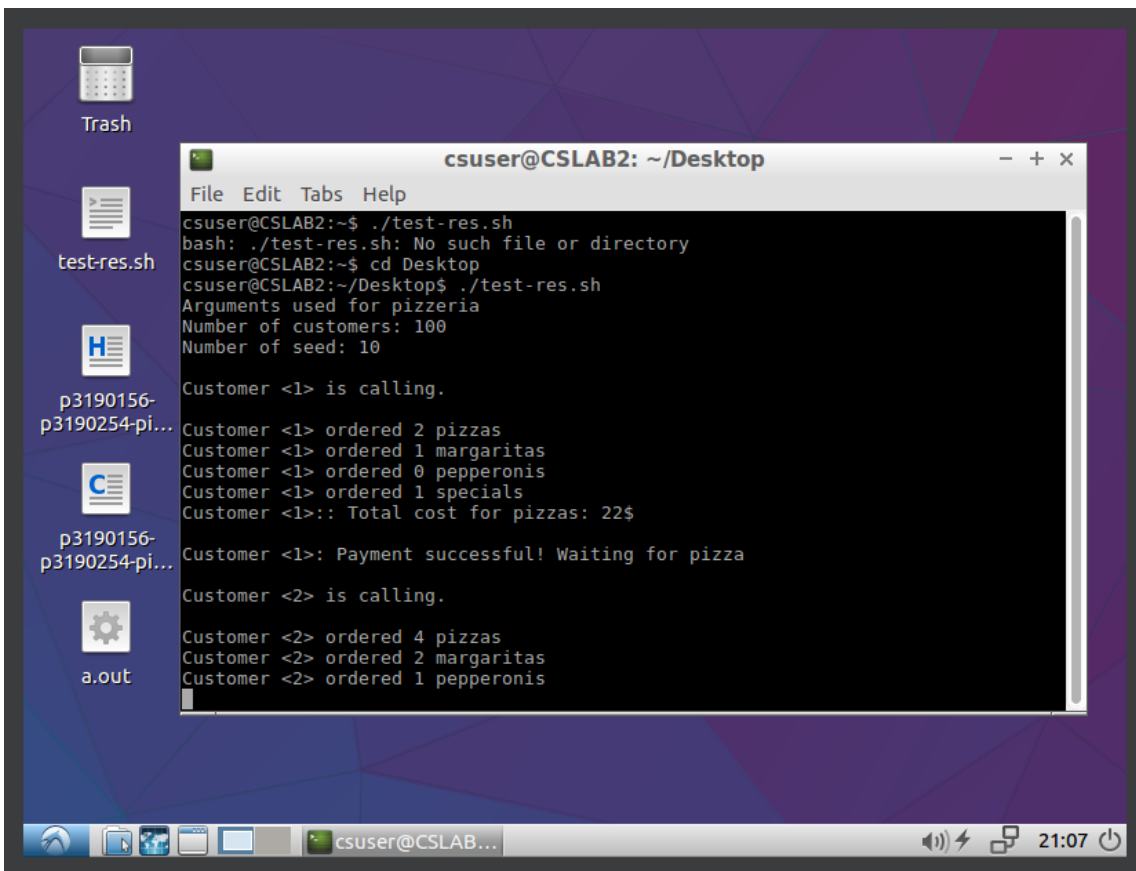


Το πρόγραμμα μας εκτελέστηκε σε εικονική μηχανή Ubuntu 16.04 LTS 32 bit την οποία εγκαταστήσαμε σύμφωνα με τις οδηγίες στο Ηλεκτρονικό βιβλίο του e-class.

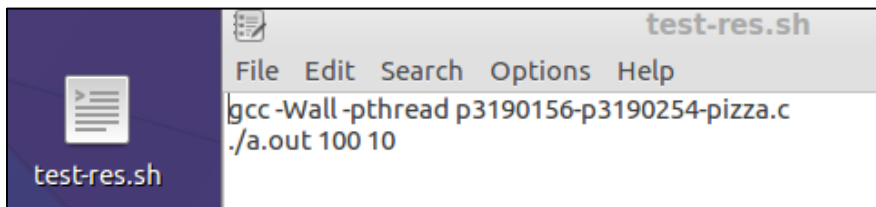
?) Πώς εκτελέσαμε το πρόγραμμα μας στην εικονική μηχανή

Στο παρακάτω screenshot, αφού τοποθετήσαμε όλα μας τα αρχεία στο Desktop ανοίγουμε το εργαλείο LXTerminal





Όπως βλέπουμε εδώ ο κώδικας εκτελείται με την εκτέλεση του .sh αρχείου. Ας ελέγξουμε γιατί γίνεται αυτό:

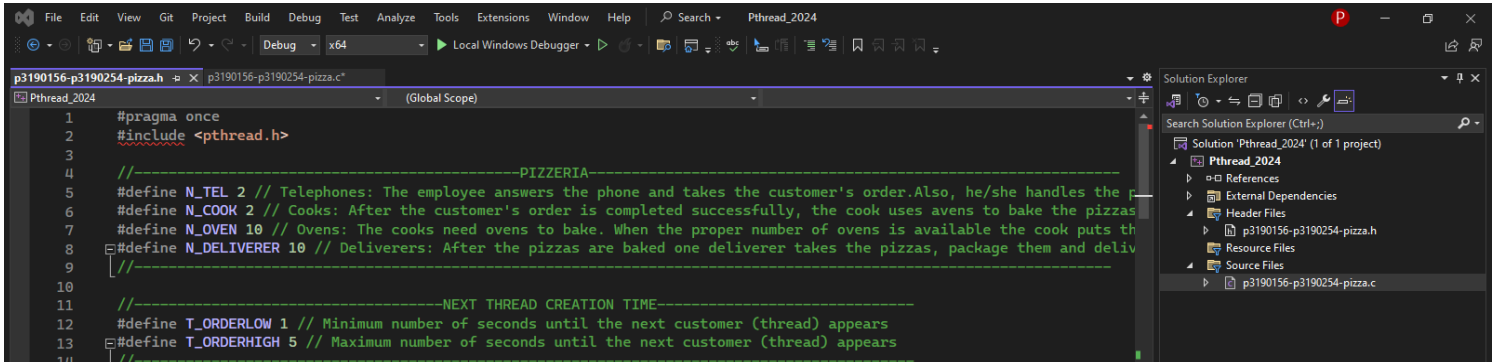


Το .sh αρχείο χρησιμοποιείται για να εκτελέσουμε τις εντολές που είναι γραμμένες σε αυτό χωρίς να χρειαστεί να τις εκτελούμε στο terminal. Οι εντολές αυτές είναι:

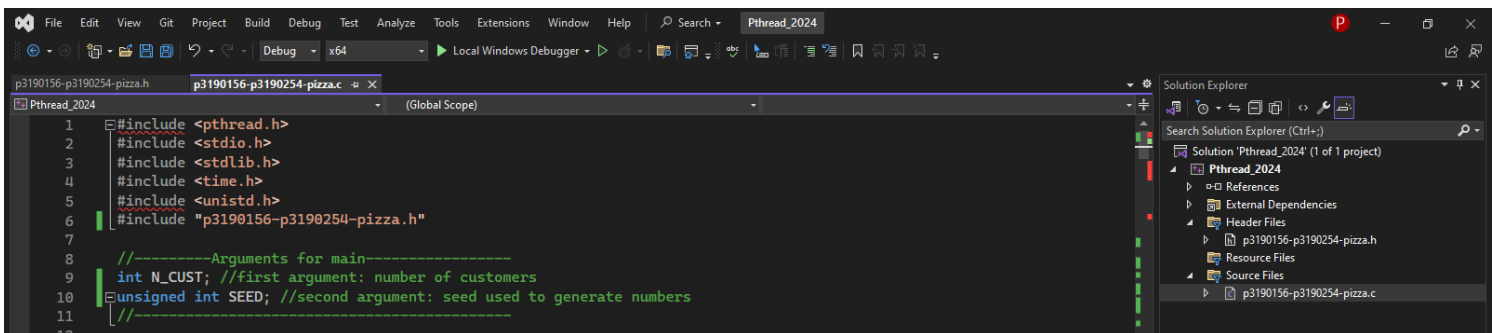
- **gcc -Wall -pthread p3190156-p3190254.c:** Η εντολή αυτή ελέγχει τυχόν errors ή warnings μέσα στον κώδικα. Στην περίπτωση μας, δεν μας εμφανίζεται τίποτα καθώς φροντίσαμε να μην υπάρχουν ούτε warnings στον κώδικα μας.
- **./a.out 100 10:** Με την μεταγλώττιση του προγράμματος το αρχείο .out παράγεται μόνο του. Το εκτελούμε δίνοντας τις παραμέτρους που μας δίνονται ("θα μεταγλωττίζει και θα εκτελεί το πρόγραμμά σας με παραμέτρους 100 πελάτες και αρχικό σπόρο 10")

Τώρα που ξεκαθαρίσαμε πως ελέγξαμε και τρέξαμε τον κώδικα μας, είναι η ώρα να εξετάσουμε προσεκτικά πως και τι ακριβώς γράψαμε.

?) Πως γράψαμε τον κώδικα μας



```
1 #pragma once
2 #include <pthread.h>
3
4 //-----PIZZERIA-----
5 #define N_TEL 2 // Telephones: The employee answers the phone and takes the customer's order. Also, he/she handles the pizzas.
6 #define N_COOK 2 // Cooks: After the customer's order is completed successfully, the cook uses ovens to bake the pizzas.
7 #define N_OVEN 10 // Ovens: The cooks need ovens to bake. When the proper number of ovens is available the cook puts the pizzas in the ovens.
8 #define N_DELIVERER 10 // Deliverers: After the pizzas are baked one deliverer takes the pizzas, package them and deliver them to the customer.
9
10 //-----NEXT THREAD CREATION TIME-----
11
12 #define T_ORDERLOW 1 // Minimum number of seconds until the next customer (thread) appears
13 #define T_ORDERHIGH 5 // Maximum number of seconds until the next customer (thread) appears
14
```



```
1 #include <pthread.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <time.h>
5 #include <unistd.h>
6 #include "p3190156-p3190254-pizza.h"
7
8 //-----Arguments for main-----
9 int N_CUST; //first argument: number of customers
10 unsigned int SEED; //second argument: seed used to generate numbers
11
12
```

Για τον κώδικα μας χρησιμοποιήσαμε το code editor Visual Studio Code 2022 version

Όπως φαίνεται και στην εικόνα, καθώς χρησιμοποιήσαμε το Visual Studio σε λειτουργικό σύστημα Windows, κάποιες από τις εντολές include προκαλούσαν error. Για αυτό κιόλας αξιοποιούμε την εικονική μηχανή.

?) Τι γράψαμε στο .h αρχείο

Εκτός από τα ζητούμενα της εργασίας προσθήσαμε τα εξής:

- **MARGARITA:** Αναπαριστά την επιλογή μαργαρίτας από τον πελάτη
- **PEPPERONI:** Αναπαριστά την επιλογή πεπερόνι από τον πελάτη
- **SPECIAL:** Αναπαριστά την επιλογή σπέσιαλ από τον πελάτη
-
- **INITIALIZATION:** Κατασκευή του mutex / condition variable
- **DESTRUCTION:** Καταστροφή του mutex / condition variable
- **LOCK:** Κλείδωμα του mutex / condition variable
- **UNLOCK:** Ξεκλείδωμα του mutex / condition variable
-
- **SUCCESS:** Αναπαριστά ότι η παραγγελία ήταν επιτυχής
- **FAIL:** Αναπαριστά ότι η παραγγελία ήταν αποτυχημένη

```

//-----REPRESENT PIZZA'S KIND-----
#define MARGARITA 111 // Variable to represent margarita
#define PEPPERONI 222 // Variable to represent pepperoni
#define SPECIAL 333 // Variable to represent special
//-----

//-----MUTEX OPERATIONS-----
#define INITIALIZATION 0
#define DESTRUCTION 1
#define LOCK 2
#define UNLOCK 3
//-----

//-----WAS THE ORDER SUCCESSFUL OR FAILED-----
#define SUCCESS 0 //variable to represent success
#define FAIL 1 //variable to represent failure
//-----

//-----CUSTOMER ORDER-----
typedef struct customer_order
{
    int order_number; //Number of order (id)
    int total_pizzas_ordered; // Number of pizzas ordered (include all kinds)
    int margaritas_ordered; // How many of them were margarita
    int pepperonis_ordered; // How many of them were pepperoni
    int special_ordered; // How many of them were special
    int cost; // Total price the customer pays
    int state; // State to represent the case the customer's order is accepted or rejected due to unsuccessful payment
}CUSTOMER_ORDER;
//-----

```

customer_order: Αναπαριστά την παραγγελία του πελάτη ως μία κλάση. Κάθε παραγγελία του πελάτη έχει:

- Αριθμό παραγγελίας
- Πλήθος όλων των πιτσών που παρήγγειλε
- Πλήθος των μαργαριτών που παρήγγειλε
- Πλήθος των πεπερόνι που παρήγγειλε
- Πλήθος των σπέσιαλ που παρήγγειλε
- Κόστος που πρέπει να πληρώσει
- Επιτυχία ή Αποτυχία παραγγελίας

Εκτός αυτών προσθέσαμε κάποιες μεθόδους για να διευκολύνουμε τη συγγραφή του κώδικα μας.

```

/*
 * mutex_operation: We try to organize our mutex operations so that we avoid
 * to repeat the same lines of code for the mutexes and make the .c file's understanding harder
 */
void mutex_operation(pthread_mutex_t* mutex, int operation);

/*
 * condition_operation: We try to organize our condition variables operations so that we avoid
 * to repeat the same lines of code for the mutexes and make the .c file's understanding harder
 */
void condition_operation(pthread_cond_t* cond, int operation);

/*
 * generate_probabilty: Used to calculate the payment's percentage using the customer's seed
 */
int generate_probabilty(float percentage, unsigned int seed); //generator for random probabilities

/*
 * telephone: There we handle the pizzeria's telephones. When the employee answers the phone the customer gets
 * his/her order id. Inside the code we handle both mutexes and condition variables for phones
 */
void telephone(CUSTOMER_ORDER* order, int operation);

/*
 * cook: There we handle the pizzeria's cooks. When one order is prepared, the cook must take the pizzas and
 * bake them in the ovens. Of course, the cook must wait for the proper number of ovens to be available.
 * Inside the code we handle both mutexes and condition variables for cooks
 */

```

```
void cook(int operation);
```

```

/*
 * bake: There we handle the pizzeria's ovens. Like we mentioned before, the cook must wait for the available
 * number of ovens to be available so that he/she can put the order's pizzas at the same time. Compared to
 * the previous methods we check the crowd of pizzas asked by the customer.
 * Inside the code we handle both mutexes and condition variables for ovens
 */
void bake(CUSTOMER_ORDER *customer_order, int operation);

/*
 * delivery: There we handle the pizzeria's delivery guys. When one order is ready to go, one deliverer must
 * package all pizzas and ride them to the customer. When they reach the customer the order is complete.
 * But we must take notice of the deliverer to come back to the pizzeria
 */
void delivery(int operation); //method to call mutex operations for all deliverers

/*
 * ask_how_many_pizzas: A simple number generator from N_ORDERLOW to N_ORDERHIGH (1 to 5 pizzas)
 */
int ask_how_many_pizzas(unsigned int seed);

/*
 * ask_what_kind_of_pizzas: After we get how many pizzas a customer wants, for each pizza
 * we use a number generator to determine each pizza's kind
 */
int ask_what_kind_of_pizzas(unsigned int seed);

```

```
void* order(void* customer_id); //method for the total transaction(used by all customer threads)
```

```

/*
 * When we run our program we use a .sh file which sets the arguments
 * Of course we always need to check what arguments are we given
 */
void arguments_check(int argc, char* argv[]); //method to check all arguments for the a.out and the .sh file

```

Μέθοδος	Λειτουργία
mutex_operation	Διαχειρίζεται όλες τις λειτουργίες ενός mutex (κατασκευή, διαγραφή, κλείδωμα, ξεκλείδωμα)
condition_operation	Διαχειρίζεται όλες τις λειτουργίες ενός condition variable (κατασκευή, διαγραφή, κλείδωμα, ξεκλείδωμα)
generate_probability	Επιστρέφει επιτυχία ή αποτυχία αναλόγως την πιθανότητα που δίνεται ως είσοδο και τον τυχαίο αριθμό που παράγεται από τον σπόρο
telephone	Διαχειρίζεται τα τηλέφωνα
cook	Διαχειρίζεται τους μάγειρες
bake	Διαχειρίζεται τους φούρνους
delivery	Διαχειρίζεται την παράδοση
ask_how_many_pizzas	Επιστρέφει έναν τυχαίο αριθμό για το πόσες πίτσες θέλει ο πελάτης
ask_what_kind_of_pizzas	Επιστρέφει το είδος μίας πίτσας από μία παραγγελία
order	Μέθοδος που χρησιμοποιείται από όλα τα νήματα. Εκτελεί όλα τα ζητούμενα της εκφώνησης για κάθε παραγγελία του πελάτη
arguments_check	Μέθοδος που ελέγχει το πλήθος και τις τιμές των εισόδων του προγράμματος

?) Τι κάνει η main μέθοδος

Ας τα δούμε με τη σειρά:

Πρώτα θα ελέγχουμε τις παραμέτρους. Χρειαζόμαστε αριθμό πελατών και τυχαίο σπόρο. Σε αντίθετη περίπτωση, θα εμφανιστεί μήνυμα σφάλματος (που θέσαμε εμείς)

```
int main(int argc, char* argv[])
{
    arguments_check(argc, argv); //first we check if all arguments given are correct
    printf("Arguments used for pizzeria\n");
    printf("Number of customers: %d\n", N_CUST);
    printf("Number of seed: %d\n", SEED);
}
```


Σε επόμενη φάση, θα κατασκευάσουμε όλα τα απαραίτητα mutexes και condition variables που αποτελούν σημαντικούς πόρους για την πιτσαρία.

```
//-----INITIALIZE ALL MUTEXES-----  
mutex_operation(&mutex_telephones, INITIALIZATION);  
mutex_operation(&mutex_cooks, INITIALIZATION);  
mutex_operation(&mutex_ovens, INITIALIZATION);  
mutex_operation(&mutex_deliverers, INITIALIZATION);  
mutex_operation(&mutex_payment, INITIALIZATION);  
mutex_operation(&mutex_common_variables, INITIALIZATION);  
mutex_operation(&mutex_print_screen, INITIALIZATION);  
//-----  
  
//-----INITIALIZE ALL CONDITION VARIABLES-----  
condition_operation(&condition_telephones, INITIALIZATION);  
condition_operation(&condition_cooks, INITIALIZATION);  
condition_operation(&condition_ovens, INITIALIZATION);  
condition_operation(&condition_deliverers, INITIALIZATION);  
//-----
```

Ας δούμε ξεχωριστά καθένα από αυτά:

Mutex	Ρόλος
mutex_telephones	Mutex τηλεφώνων
mutex_cooks	Mutex ψηστών
mutex_ovens	Mutex φούρνων
mutex_deliverers	Mutex παραδοτών
mutex_payment	Mutex πληρωμής
mutex_common_variables	Mutex διαχείρισης πράξεων που αφορούν κοινές μεταβλητές
mutex_print_screen	Mutex διαχείρισης εκτύπωσης μηνυμάτων

Condition Variable	Ρόλος
condition_telephones	Condition variable πλήθους τηλεφώνων
condition_cooks	Condition variable πλήθους ψηστών
condition_ovens	Condition variable πλήθους φούρνων
condition_deliverers	Condition variable πλήθους παραδοτών


```

int customer_id[N_CUST]; //array of all customers' ids

for (int id = 0; id < N_CUST; id++)
{
    customer_id[id] = id + 1; //we want all customers have proper ids from 1 to number_of_customers
}

int rc;

pthread_t running_threads[N_CUST];

pthread_t* customers_threads;

customers_threads = malloc(N_CUST * sizeof(pthread_t)); //we have to be careful for the memory we use

if (customers_threads == NULL)
{
    printf("\nNo memory..\n");
    return -1;
}

```

Για τα νήματα των πελατών πρέπει να δεσμεύσουμε την απαραίτητη μνήμη (μέσω της malloc). Ύστερα, θα κατασκευάσουμε το πρώτο νήμα πελάτη (με το αντίστοιχο id), και μετά από τυχαίο χρόνο θα παραχθούν όλα τα υπόλοιπα.

```

for (int i = 0; i < N_CUST; i++)
{
    if (i == 0) //we want to create the first customer with id number 1
    {
        rc = pthread_create(&running_threads[i], NULL, order, &customer_id[i]);


        if (rc != 0)
        {
            printf("\nFirst thread creation failed!\n");
            exit(-1);
        }
    }
    else //the rest will follow in order with random showing-up time
    {
        int next_customer_generation_time = rand_r(&SEED) % (T_ORDERHIGH + 1 - T_ORDERLOW) + T_ORDERLOW;

        sleep(next_customer_generation_time);

        rc = pthread_create(&running_threads[i], NULL,

        if (rc != 0)
        {
            printf("\nThread creation failed!\n");
            exit(-1);
        }
    }
}

```

 unsigned int SEED
second argument: seed used to generate numbers
[Search Online](#)

Κάθε νήμα με το δικό του id θα εκτελέσει τη μέθοδο order και θα μας εμφανιστούν όλα τα ζητούμενα για κάθε πελάτη.

```

void* stat; //stat used for joining method (usually null)

for (int i = 0; i < N_CUST; i++)
{
    rc = pthread_join(running_threads[i], &stat);

    if (rc != 0)
    {
        printf("\nThread join failed!\n");
        exit(-1);
    }
}

```

Αφού ολοκληρωθεί η order για όλους τους πελάτες, δεν πρέπει να ξεχάσουμε να τερματίσουμε τα νήματα που φτιάξαμε. Στην περίπτωση αυτή, εκτελούμε την join.

```

if(successful_orders > 0)
{
    mutex_operation(&mutex_common_variables, LOCK);
    average_service_time_of_successful_orders = sum_service_time / successful_orders;
    average_cooling_time_of_successful_orders = sum_cooling_time / successful_orders;
    mutex_operation(&mutex_common_variables, UNLOCK);
}

// Print all statistics and calculated values

mutex_operation(&mutex_print_screen, LOCK);
printf("\n\nTotal orders: %d", number_of_orders);
mutex_operation(&mutex_print_screen, UNLOCK);

mutex_operation(&mutex_print_screen, LOCK);
printf("\n\nTotal successful orders: %d", successful_orders);
mutex_operation(&mutex_print_screen, UNLOCK);

mutex_operation(&mutex_print_screen, LOCK);
printf("\n\nTotal failed orders: %d", failed_due_to_payment);
mutex_operation(&mutex_print_screen, UNLOCK);

mutex_operation(&mutex_print_screen, LOCK);
printf("\n\nTotal revenue: %d$", total_revenue);
mutex_operation(&mutex_print_screen, UNLOCK);

mutex_operation(&mutex_print_screen, LOCK);
printf("\n\nTotal pizzas ordered: %d", total_pizzas_ordered);
mutex_operation(&mutex_print_screen, UNLOCK);

mutex_operation(&mutex_print_screen, LOCK);
printf("\n\nTotal margaritas ordered: %d", total_margaritas_ordered);
mutex_operation(&mutex_print_screen, UNLOCK);

```

Αφού τερματιστούν όλα τα νήματα, εμφανίζονται όλα τα στατιστικά που ζητούνται στην εργασία.

```

free(customers_threads); // free memory used for the threads (to avoid garbage staying)

//-----DESTROY ALL MUTEXES-----
mutex_operation(&mutex_telephones, DESTRUCTION);
mutex_operation(&mutex_cooks, DESTRUCTION);
mutex_operation(&mutex_ovens, DESTRUCTION);
mutex_operation(&mutex_deliverers, DESTRUCTION);
mutex_operation(&mutex_payment, DESTRUCTION);
mutex_operation(&mutex_common_variables, DESTRUCTION);
mutex_operation(&mutex_print_screen, DESTRUCTION);
//-----

//-----DESTROY ALL CONDITION VARIABLES-----
condition_operation(&condition_telephones, DESTRUCTION);
condition_operation(&condition_cooks, DESTRUCTION);
condition_operation(&condition_ovens, DESTRUCTION);
condition_operation(&condition_deliverers, DESTRUCTION);
//-----

return 0;

```

Στο τέλος του προγράμματος, δεν πρέπει να αφήσουμε κανένα ίχνος περιττής μνήμης. Αρχικά, θα αποδεσμεύσουμε τη μνήμη που χρησιμοποιήσαμε για τα νήματα μας. Εκ των υστέρων, θα διαγράψουμε όλα τα mutexes και condition variables που δεν έχουν πλέον κάποια χρησιμότητα.

?) Τι είναι η order που χρησιμοποιείται στα νήματα

Με λίγα λόγια, αναπαριστά ολόκληρη την παραγγελία για κάθε πελάτη. Προφανώς δεν θα εξηγήσουμε τι ακριβώς κάνει η πιτσαρία μιας και μας το αναφέρει ήδη η εκφώνηση. Ωστόσο, οφείλουμε να τονίσουμε κάποια σημαντικά σημεία της μεθόδου:

```

void* order(void* customer_id)
{
    int* cid = (int*)customer_id; // we get the customer's id
    CUSTOMER_ORDER c_order; // each customer has their own struct representing their order

    // We put default values in each customer's order (to avoid random default values for example -1253467)
    c_order.total_pizzas_ordered = 0;
    c_order.margaritas_ordered = 0;
    c_order.pepperonis_ordered = 0;
    c_order.special_ordered = 0;
    c_order.cost = 0;

    unsigned int seed = SEED + *cid; // the seed for each customer must be unique

    struct timespec time_order_started; // Time the order started
    struct timespec time_order_baked; // Time the order finished baking
    struct timespec time_order_packed; // Time the order finished packing
    struct timespec time_order_delivered; // Time the order was delivered

```

Κάθε πελάτης έχει αρχικοποιημένες τιμές στην παραγγελία του και για τον καθένα ο τυχαίος σπόρος είναι διαφορετικός (αλλάζει αναλόγως του id του)

Οι τιμές timespec είναι χρονικές στιγμές που χρησιμοποιούνται για τον υπολογισμό ζητούμενων χρόνων.

```
// The customer shows up and waits someone to pick up the phone
clock_gettime(CLOCK_REALTIME, &time_order_started);

telephone(&c_order, LOCK); // one of the two employees answers the phone
```

Οι τιμές κάθε timespec λαμβάνονται μέσω της clock_gettime. Στο παραπάνω screenshot βλέπουμε την «εμφάνιση» του πελάτη, ο οποίος θέλει να παραγγείλει καλεί την πιτσαρία και περιμένει κάποιον υπάλληλο να απαντήσει στην κλήση του.

```
int customer_pizza_number = ask_how_many_pizzas(seed); // how many pizzas the customer asked
c_order.total_pizzas_ordered = customer_pizza_number; // save the number in struct
```

```
for(int pizza = 0; pizza < customer_pizza_number; pizza++) // for each pizza the customer asked
{
    // We must add something to the seed as well because if we put seed all by itself then
    // all pizzas will be the same

    int what_pizza = ask_what_kind_of_pizzas(seed + pizza); // MARGARITA/PEPPERONI/SPECIAL
    // Why + pizza?: Because every pizza the customer asks must be totally random.
    // If we put the same seed we might expect the same pizzas all the time
```

Παραγγελία πελάτη: Με τον σπόρο του, ζητάει έναν τυχαίο αριθμό πιτσών. Για κάθε πίτσα που ζήτησε παράγουμε ποιο είδος θέλει. Για να αποφύγουμε έναν πελάτη να παραγγέλνει το ίδιο είδος για όλες τις πίτσες (π.χ. 5 πίτσες => 5 σπέσιαλ), λαμβάνουμε υπόψη και τη μεταβλητή pizza στο for loop.

```

int payment_probability = generate_probabilty(P_CARDSUCCESS, seed);

if (payment_probability == SUCCESS)
{
    c_order.state = SUCCESS;

    mutex_operation(&mutex_print_screen, LOCK);
    printf("\nCustomer <%d>: Payment successful! Waiting for pizza\n", *cid);
    mutex_operation(&mutex_print_screen, UNLOCK);

    mutex_operation(&mutex_common_variables, LOCK);
    successful_orders++;
    mutex_operation(&mutex_common_variables, UNLOCK);

    mutex_operation(&mutex_payment, LOCK);
    total_revenue += c_order.cost;
    mutex_operation(&mutex_payment, UNLOCK);

    telephone(&c_order, UNLOCK); // The phone call ends. Time to make the pizzas
}
else
{
    c_order.state = FAIL;

    mutex_operation(&mutex_print_screen, LOCK);
    printf("\nCustomer <%d>: Payment failed. Order is canceled\n", *cid);
    mutex_operation(&mutex_print_screen, UNLOCK);

    mutex_operation(&mutex_common_variables, LOCK);
    failed_due_to_payment++;
    mutex_operation(&mutex_common_variables, UNLOCK);

    telephone(&c_order, UNLOCK); // The phone call ends. Next customer...
}

```

Πληρωμή πελάτη: Αφού υπολογίσουμε το κόστος της παραγγελίας του πελάτη πρέπει να δούμε εάν η πληρωμή με τα στοιχεία της κάρτας του γίνει με επιτυχία. Όπως φαίνεται, η επιτυχία θα αναπαριστάται από την state της παραγγελίας. Εάν είναι επιτυχής η πληρωμή, προσέχουμε να αυξήσουμε τα έσοδα (κρίσιμο σημείο του κώδικα) και το πλήθος πετυχημένων παραγγελιών. Σε αντίθετη περίπτωση, απλώς αυξάνουμε το πλήθος αποτυχημένων παραγγελιών.

```

clock_gettime(CLOCK_REALTIME, &time_order_packed); // get the moment the order is packaged
preperation_time = time_order_packed.tv_sec - time_order_started.tv_sec;

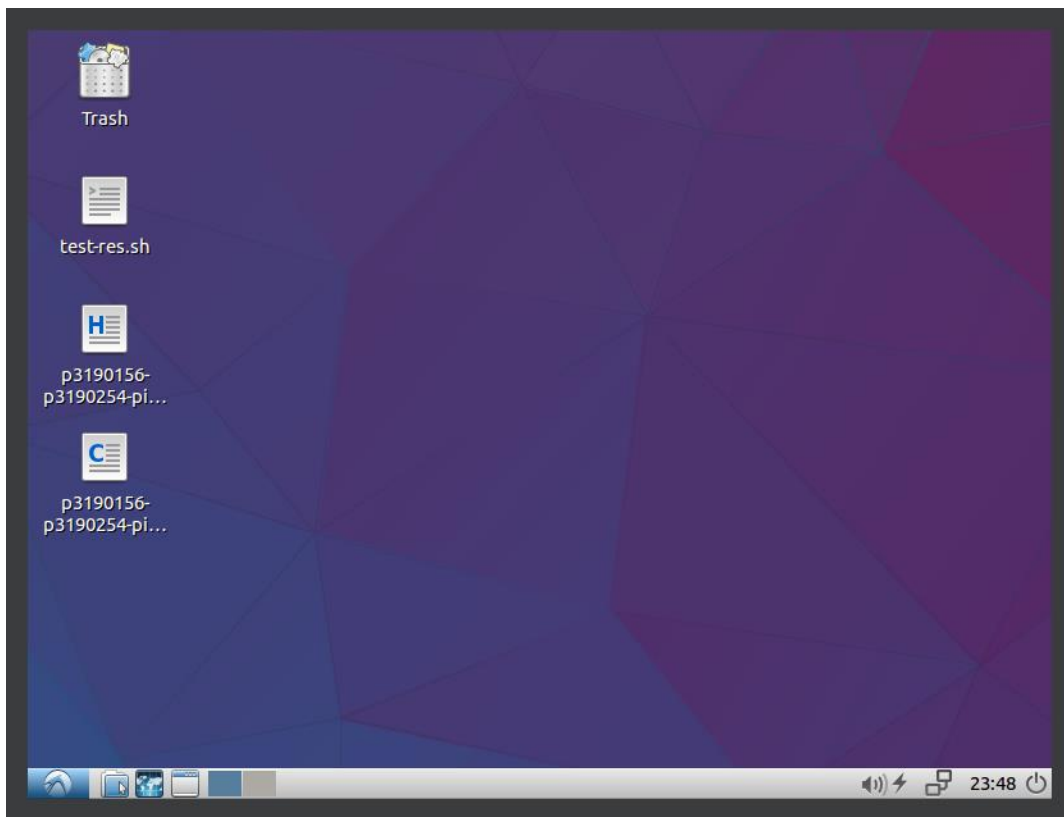
mutex_operation(&mutex_print_screen, LOCK);
printf("\nCustomer <%d>: Order prepared in %d minutes\n", *cid, preperation_time);
mutex_operation(&mutex_print_screen, UNLOCK);

```

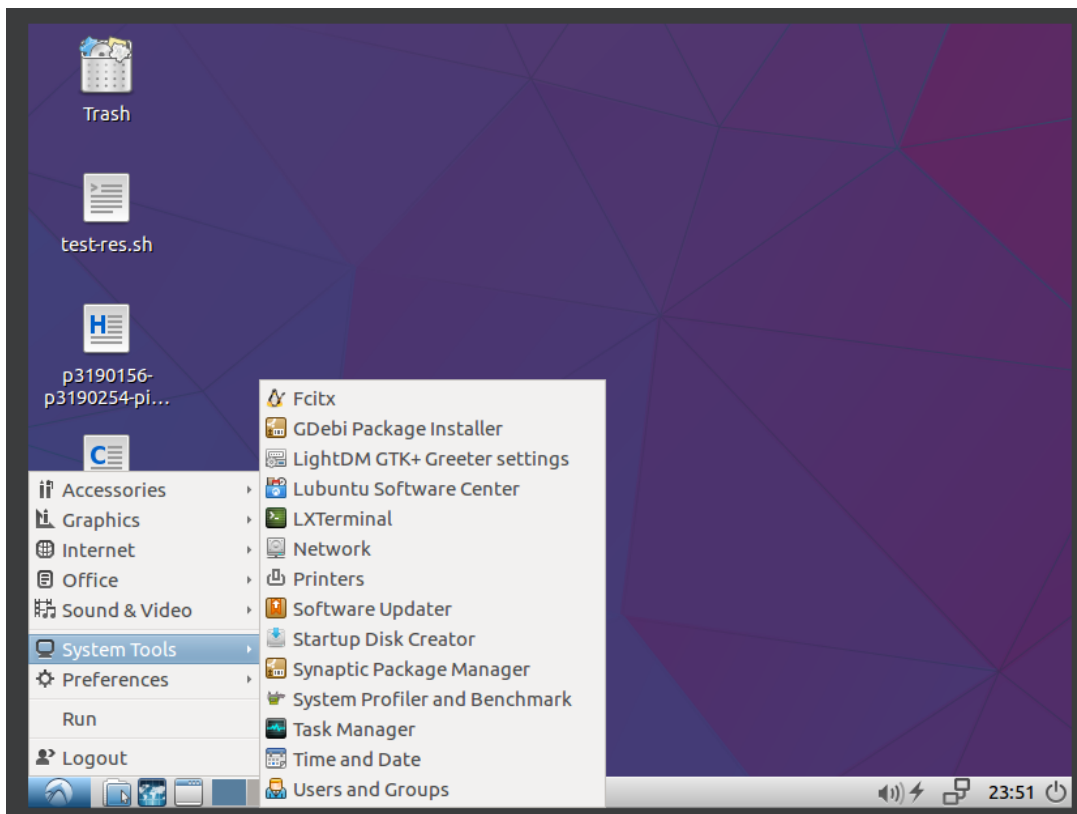
Για τον υπολογισμό των ζητούμενων χρόνων χρειαζόμαστε πάντα 2 timespec, ένα για το ξεκίνημα και ένα για τον τερματισμό μίας πράξης.

Στις επόμενες σελίδες θα παρουσιάσουμε κάποια screenshots του προγράμματος στην εικονική μηχανή και θα σχολιάσουμε τις εξόδους. Μαζί με τα ζητούμενα μηνύματα αποφασίσαμε να εμφανίζουμε και κάποια επιπρόσθετα (π.χ. τι είδη πίτσας και πόσα από αυτά παρήγγειλε ο πελάτης).

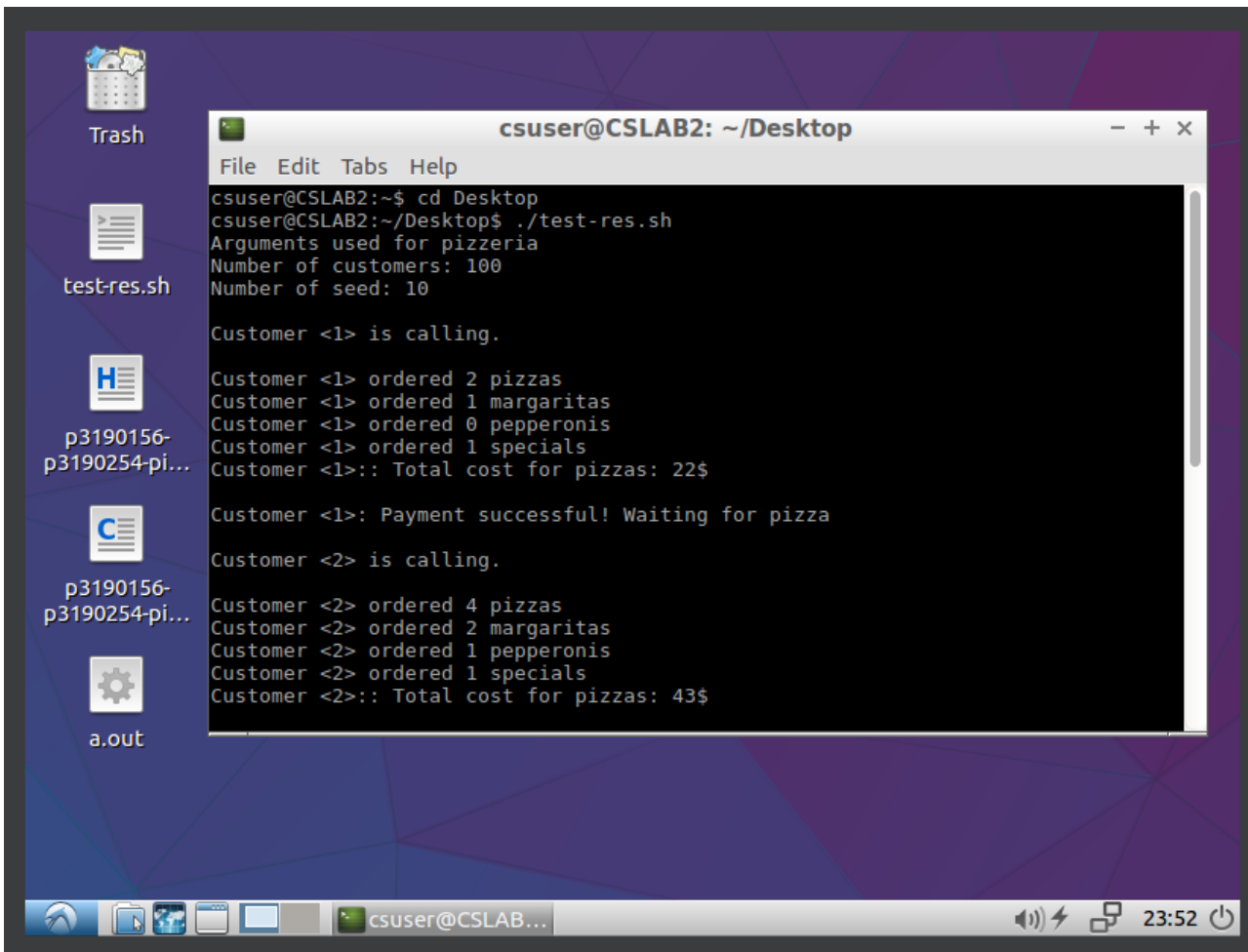
1) Τοποθέτηση ζητούμενων αρχείων στο Desktop



2) Άνοιγμα LXTerminal



3) Έναρξη προγράμματος



The screenshot shows a Linux desktop with a purple and blue geometric background. On the left sidebar, there are icons for 'Trash', 'test-res.sh', and 'a.out'. The terminal window, titled 'csuser@CSLAB2: ~/Desktop', displays the output of a script named 'test-res.sh'. The script simulates a pizza ordering system with two customers. Customer 1 orders 2 pizzas (1 margarita, 0 pepperoni, 1 special) for a total of 22\$. Customer 2 orders 4 pizzas (2 margaritas, 1 pepperoni, 1 special) for a total of 43\$. The desktop also shows files named 'p3190156-p3190254-pi...' and a taskbar at the bottom with the time 23:52.

```
csuser@CSLAB2: ~/Desktop
File Edit Tabs Help
csuser@CSLAB2:~$ cd Desktop
csuser@CSLAB2:~/Desktop$ ./test-res.sh
Arguments used for pizzeria
Number of customers: 100
Number of seed: 10

Customer <1> is calling.

Customer <1> ordered 2 pizzas
Customer <1> ordered 1 margaritas
Customer <1> ordered 0 pepperonis
Customer <1> ordered 1 specials
Customer <1>:: Total cost for pizzas: 22$

Customer <1>: Payment successful! Waiting for pizza

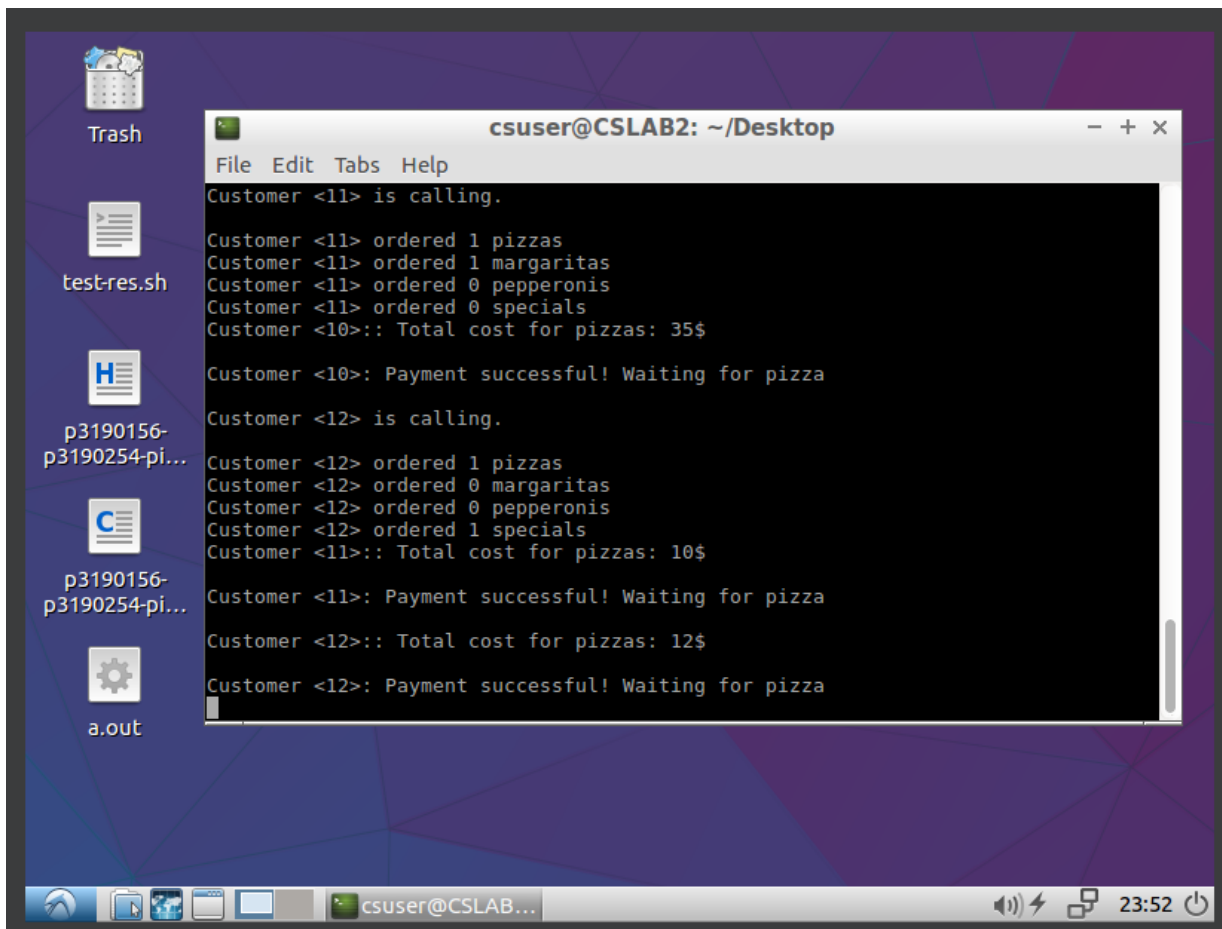
Customer <2> is calling.

Customer <2> ordered 4 pizzas
Customer <2> ordered 2 margaritas
Customer <2> ordered 1 pepperonis
Customer <2> ordered 1 specials
Customer <2>:: Total cost for pizzas: 43$
```

Εδώ παρατηρούμε τα εξής:

- Από δικό μας μήνυμα φαίνονται ποιες παράμετροι έχουν ενταχθεί στο a.out (πελάτες και σπόρος)
- Για κάθε πελάτη εκτός από το πόσες πίτσες θέλει, για να εξασφαλίσουμε πως το κόστος και τα είδη υπολογίζονται σωστά, αποφασίσαμε να εμφανίζουμε αναλυτικά την παραγγελία του πελάτη

4) Πληρωμή παραγγελίας



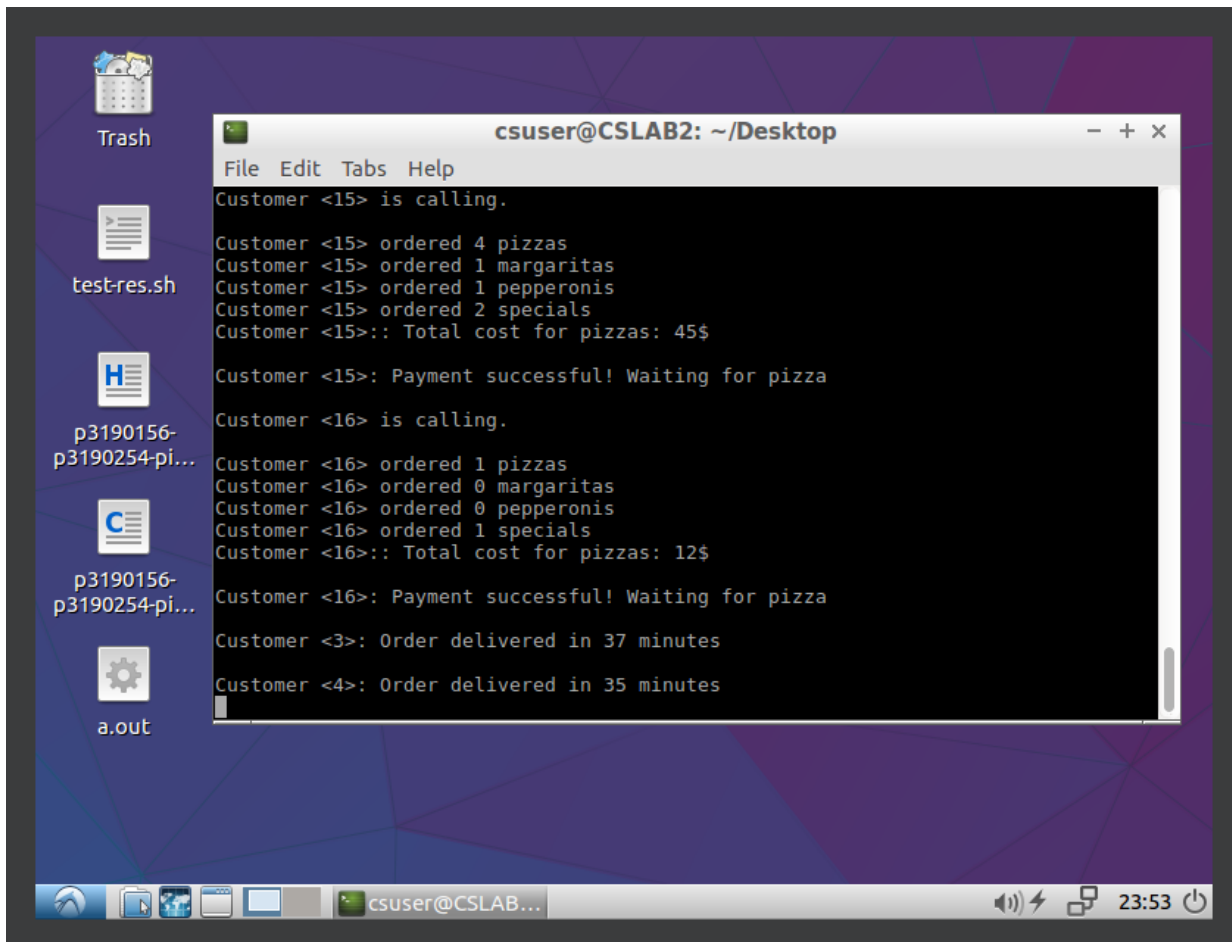
The screenshot shows a Linux desktop with a purple geometric background. On the left sidebar, there are icons for 'Trash', 'test-res.sh', 'p3190156-p3190254-pi...', 'p3190156-p3190254-pi...', and 'a.out'. A terminal window titled 'csuser@CSLAB2: ~/Desktop' is open, displaying the following text:

```
File Edit Tabs Help
Customer <11> is calling.
Customer <11> ordered 1 pizzas
Customer <11> ordered 1 margaritas
Customer <11> ordered 0 pepperonis
Customer <11> ordered 0 specials
Customer <10>:: Total cost for pizzas: 35$
Customer <10>: Payment successful! Waiting for pizza
Customer <12> is calling.
Customer <12> ordered 1 pizzas
Customer <12> ordered 0 margaritas
Customer <12> ordered 0 pepperonis
Customer <12> ordered 1 specials
Customer <11>:: Total cost for pizzas: 10$
Customer <11>: Payment successful! Waiting for pizza
Customer <12>:: Total cost for pizzas: 12$
Customer <12>: Payment successful! Waiting for pizza
```

The terminal window has a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The desktop environment includes a taskbar at the bottom with icons for a web browser, file manager, and terminal, along with system status icons (volume, network, battery) and the time '23:52'.

Για τον πελάτη <11> βλέπουμε πως αφού υπολογιστεί το κόστος παραγγελίας του, πραγματοποιείται με επιτυχία η πληρωμή του. Κατόπιν πληρωμής, μένει η προετοιμασία και η παράδοση της παραγγελίας του.

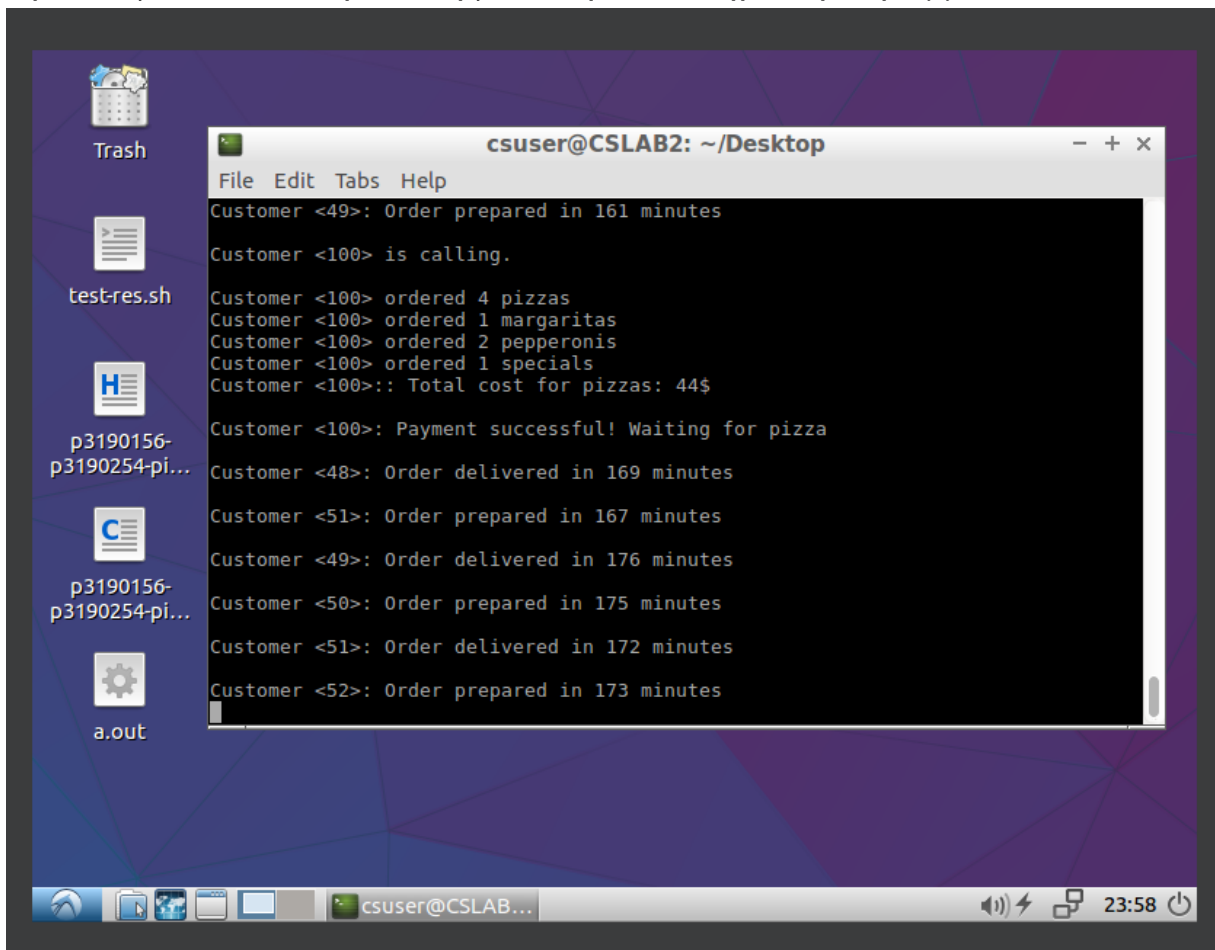
5) Παράδοση παραγγελίας



The screenshot shows a Linux desktop with a purple geometric background. On the left sidebar, there are icons for 'Trash', 'test-res.sh', two files named 'p3190156-p3190254-pi...', and 'a.out'. A terminal window titled 'csuser@CSLAB2: ~/Desktop' is open, displaying a pizza ordering simulation. The simulation shows two customers, 15 and 16, placing orders with various pizzas and toppings, receiving payment confirmation, and finally getting their orders delivered within 37 and 35 minutes respectively. The desktop's taskbar at the bottom shows the user 'csuser@CSLAB...', system icons for network, volume, and battery, and the time '23:53'.

```
csuser@CSLAB2: ~/Desktop
File Edit Tabs Help
Customer <15> is calling.
Customer <15> ordered 4 pizzas
Customer <15> ordered 1 margaritas
Customer <15> ordered 1 pepperonis
Customer <15> ordered 2 specials
Customer <15>:: Total cost for pizzas: 45$
Customer <15>: Payment successful! Waiting for pizza
Customer <16> is calling.
Customer <16> ordered 1 pizzas
Customer <16> ordered 0 margaritas
Customer <16> ordered 0 pepperonis
Customer <16> ordered 1 specials
Customer <16>:: Total cost for pizzas: 12$
Customer <16>: Payment successful! Waiting for pizza
Customer <3>: Order delivered in 37 minutes
Customer <4>: Order delivered in 35 minutes
```

6) Προετοιμασία και παράδοση μετά την ολοκλήρωση παραγγελιών



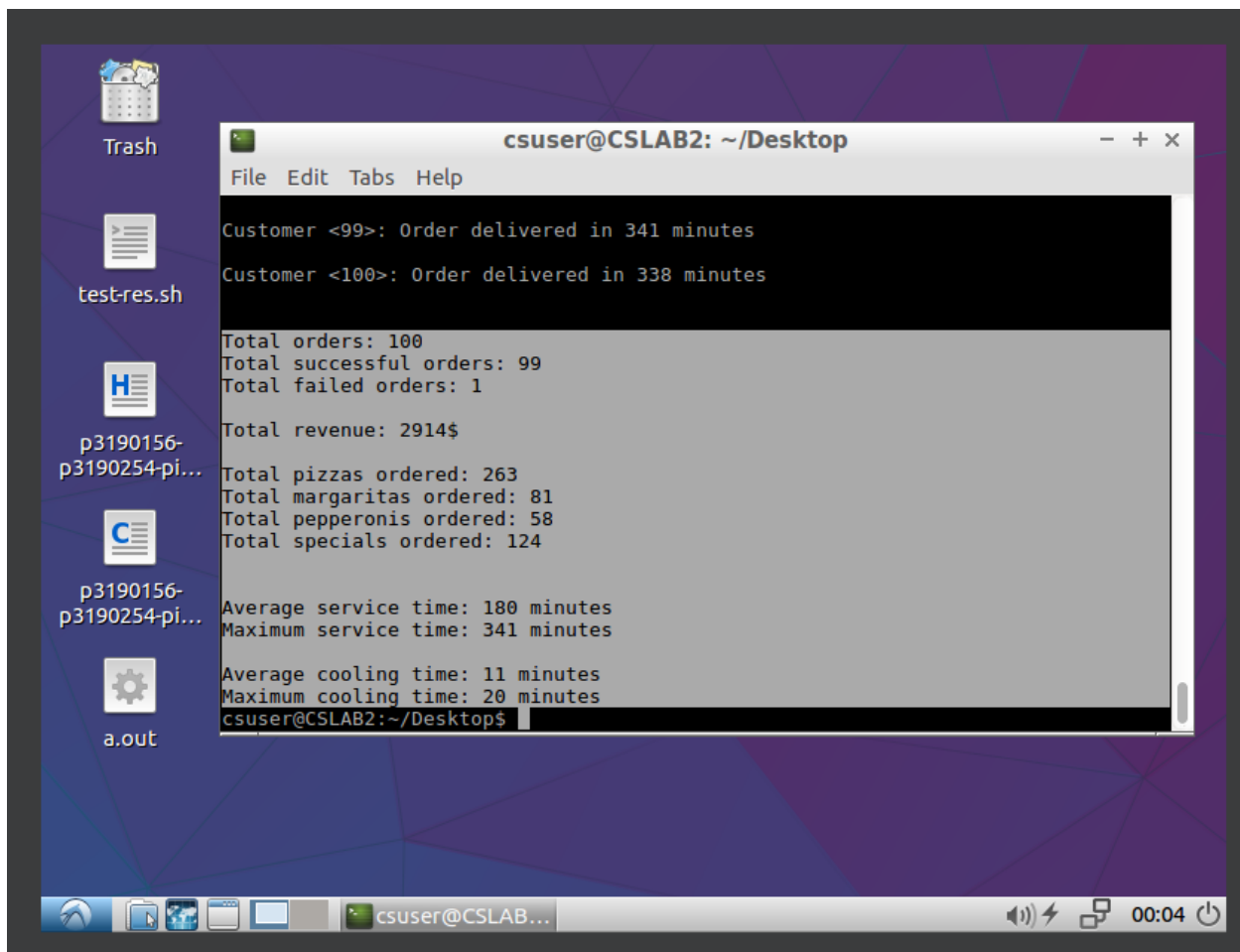
The screenshot shows a Linux desktop with a purple and blue geometric background. On the left sidebar, there are icons for 'Trash', 'test-res.sh', 'p3190156-p3190254-pi...', 'p3190156-p3190254-pi...', and 'a.out'. A terminal window titled 'csuser@CSLAB2: ~/Desktop' is open, displaying the following text:

```
File Edit Tabs Help
Customer <49>: Order prepared in 161 minutes
Customer <100> is calling.
Customer <100> ordered 4 pizzas
Customer <100> ordered 1 margaritas
Customer <100> ordered 2 pepperonis
Customer <100> ordered 1 specials
Customer <100>: Total cost for pizzas: 44$
Customer <100>: Payment successful! Waiting for pizza
Customer <48>: Order delivered in 169 minutes
Customer <51>: Order prepared in 167 minutes
Customer <49>: Order delivered in 176 minutes
Customer <50>: Order prepared in 175 minutes
Customer <51>: Order delivered in 172 minutes
Customer <52>: Order prepared in 173 minutes
```

The desktop environment includes a taskbar at the bottom with icons for a web browser, file manager, and terminal, along with system status icons (volume, network, battery) and the time '23:58'.

Αφού και το 100^{ος} πελάτης δώσει την παραγγελία του μένει μόνο το κομμάτι προετοιμασίας και παράδοσης όλων των παραγγελιών

7) Αποτελέσματα



The screenshot shows a Linux desktop with a purple geometric background. On the left sidebar, there are icons for Trash, test-res.sh, and two files named p3190156-p3190254-pi... with icons for a document and a code editor. At the bottom left, there is a gear icon labeled a.out. The terminal window, titled 'csuser@CSLAB2: ~/Desktop', displays the following output:

```
csuser@CSLAB2: ~/Desktop
File Edit Tabs Help

Customer <99>: Order delivered in 341 minutes
Customer <100>: Order delivered in 338 minutes

Total orders: 100
Total successful orders: 99
Total failed orders: 1

Total revenue: 2914$

Total pizzas ordered: 263
Total margaritas ordered: 81
Total pepperonis ordered: 58
Total specials ordered: 124

Average service time: 180 minutes
Maximum service time: 341 minutes

Average cooling time: 11 minutes
Maximum cooling time: 20 minutes
csuser@CSLAB2:~/Desktop$
```

Στο τέλος του προγράμματος μας εμφανίζονται κάποια στατιστικά. Ας τα σχολιάσουμε:

Total orders: 100 → Λογικό, αφού είχαμε 100 νήματα. Αυτό σημαίνει πως κανένα νήμα δεν είχε πρόβλημα δημιουργίας και εκτέλεσης

Total successful orders: 99 → Η πιθανότητα μία παραγγελία να είναι επιτυχής είναι 95%. Το αποτέλεσμα βρίσκεται σε λογικά πλαίσια

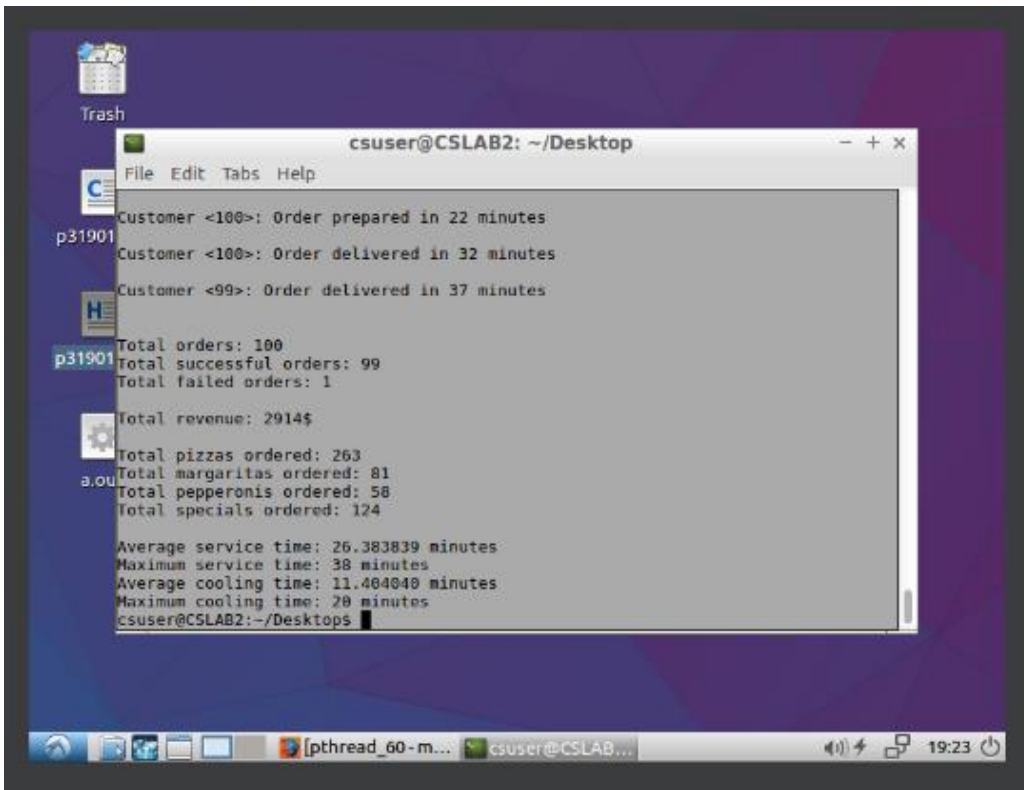
Total failed orders: 1 → Η πιθανότητα μία παραγγελία να αποτύχει είναι 5%. (1 στις 20). Το αποτέλεσμα βρίσκεται σε λογικά πλαίσια.

Total revenue 2914\$ → Με επαλήθευση αν λάβουμε υπόψιν:

$81 \cdot 10$ (μαργαρίτες) + $58 \cdot 11$ (πεπερόνι) + $124 \cdot 12$ (σπέσιαλς) - 22\$ (αποτυχημένη παραγγελία <14>) = $810 + 638 + 1488 - 22 = 2914\$$

Average service time: 180 minutes → 3 ώρες; Με δύο τηλέφωνα και δύο μάγειρες και με σχεδόν συνεχή εμφάνιση πελατών σίγουρα κάποιοι πελάτες θα χρειάζονταν να περιμένουν πολλή ώρα (η εμφάνιση του πελάτη είναι πριν καν σηκώσει το τηλέφωνο ο

υπάλληλος). Για σιγουριά τρέξαμε το πρόγραμμα με περισσότερους πόρους (+10 τηλέφωνα και μάγειρες) και η διαφορά των χρόνων ήταν μεγάλη.



The screenshot shows a Linux desktop with a purple background. A terminal window titled 'csuser@CSLAB2: ~/Desktop' is open, displaying the output of a simulation. The output includes order status updates for customers 100 and 99, and a summary of 100 total orders. The summary shows 99 successful orders and 1 failed order, with a total revenue of 2914\$. It also lists the number of pizzas ordered (263) and the breakdown by type: 81 margaritas, 58 pepperonis, and 124 specials. Service and cooling time statistics are provided at the bottom.

```
csuser@CSLAB2: ~/Desktop
File Edit Tabs Help
Customer <100>: Order prepared in 22 minutes
p31901 Customer <100>: Order delivered in 32 minutes
Customer <99>: Order delivered in 37 minutes
p31901
Total orders: 100
Total successful orders: 99
Total failed orders: 1
Total revenue: 2914$
Total pizzas ordered: 263
Total margaritas ordered: 81
Total pepperonis ordered: 58
Total specials ordered: 124
Average service time: 26.383839 minutes
Maximum service time: 38 minutes
Average cooling time: 11.404040 minutes
Maximum cooling time: 20 minutes
csuser@CSLAB2:~/Desktops
```

ΠΡΟΣΟΧΗ: Η ΕΙΚΩΝΑ ΑΥΤΗ ΕΊΝΑΙ ΜΕ ΔΙΑΦΟΡΕΤΙΚΌ ΑΡΙΘΜΌ ΤΗΛΕΦΏΝΩΝ ΚΑΙ ΜΆΓΕΙΡΩΝ

Maximum service time: 341 minutes → Παραγγελία <99>. Ήταν η μέγιστη δυνατή τιμή

Average cooling time: 11 minutes → Ο αριθμός βρίσκεται σε λογικά πλαίσια. Αν σκεφτούμε πως το πακετάρισμα κατά μέσο όρο είναι 2-3 λεπτά και η παράδοση είναι κατά μέσο όρο κοντά στα 7-10 λεπτά, το αποτέλεσμα είναι λογικό.

Maximum cooling time: 20 minutes → Χωρίς να δούμε όλες τις παραγγελίες επαληθεύεται με σιγουριά. Ο χρόνος κρυώματος κυμαίνει στο διάστημα [6,20] λεπτά.

Τα 20 λεπτά είναι η περίπτωση όπου ο πελάτης παρήγγειλε 5 πίτσες και το κρύωμα είναι: 5 λεπτά το πακετάρισμα όλων συν 15 λεπτά η παράδοση.

Εν ολίγοις, το πρόγραμμα εκτελείται με επιτυχία, τα αποτελέσματα επαληθεύονται με σιγουριά και όλοι οι πελάτες (εκτός του 14) μπορούν να απολαύσουν την παραγγελία τους.

Σας ευχαριστούμε.