Operating Systems Lab (CS 470):

Lab 4: Write a simulation for a burger joint and its customers. The burger joint should be implemented as a server application, while the clients should be implemented as client applications connecting to the server application via TCP/IP. The burger joint will have several chefs producing the burgers. Each burger will take some time to be produced and the clients (see customers) will command and eat the burgers if they are available. However, they also will need some time to consume the burgers. The burger joint can produce a certain number of burgers and each client is capable to consume a certain amount of burgers. The simulation stops either when there are no more burgers available or each of the clients consumed the maximum number of burgers they were supposed to eat. If there are still left some burgers (or some of the burgers were not yet prepared by the chefs), the burger joint remains open waiting for possible new customers.

Overview

Communication between two software programs can be done using sockets among others. A socket is identified by an IP address concatenated with a port number. The server waits for incoming clients requests by listening to a specified port. Once the request is received, the server accepts a connection from the client to complete the connection. Servers which implement specific services such as ftp, telnet, http listen on some dedicated ports (telnet (23), ftp (21), http (80), therefore use port numbers bigger than 1024 for that purpose. Ex. Use 54321 – probably nobody is using that.

Instructions

1) Both the server and the client program should be written in C/C++ under Linux.
2) The client program(s) should connect via socket to the running server program.
3) If the server is not available (not running) the client should timeout (try several attempts at some given interval) and exit.
4) The connection IP address (see server's location), the port number, and the maximum number of burgers (in this particular order!) what that particular client can eat will be provided as command line parameters for the client application.
5) Once a client is served with a burger, the application should select randomly between 1, 3 or 5 seconds necessary to consume that particular burger. During that period of time the client can not ask for a new burger. The client should be served till he/she is reaching the maximum number of burgers he/she can eat (see instruction 4) or there are no more burgers left.
6) The maximum number of burgers what the burger joint can produce and the number of chefs available in the burger joint (in that particular order) should be provided as command line arguments when we start the server application.

7) The chefs producing the burgers should run in separate threads. Producing each burger could take randomly 2 or 4 seconds. During that period of time the chef can not produce a new burger. The chef(s) will stop working if the maximum number of burgers provided as command line argument is reached.
8) For each client the server should listen in a separate thread as the clients should be served in parallel. [This specification can be omitted, but it would be very good if you would implemente it in separate threads!]

Notes

1) If no parameters are provided for the command line arguments, the following default values should be considered:
    a. for the server: **25** (total number of burgers to be produced) and **2** (number of chefs)
    b. for the client: **127.0.0.1** (ip address of the server), **54321** (port number) **10** (maximum number of burgers to be consumed by the client)
2) At each stage, we should see on the screen how many burgers are produced by which chef, how much time it took to produce each burger, how many burgers have left to be produced, how many burgers are consumed by each client, how many burgers can be still eaten by each client, and all other information necessary to closely follow the simulation.
3) Consider errors like no network connection, sudden stop of the server, sudden stop of a client, etc.

Rubric

Task	**Points**
Error handling/Information	1/1
1 Client/ 1 Server	4/4