**Lab2**

*Last modified*: 10/17/2022

Try to implement the same problems as for lab 1 in the situation the server needs to be concurrent � i.e. be able to accept multiple clients at the same time.

Also try to solve the following problems in the same context. What approach would be the best in the following scenarios? (Threads, processes)

1.   The client takes a string from the command line and sends it to the server. The server interprets the string as a command with its parameters. It executes the command and returns the standard output and the exit code to the client.

2.   The client sends the complete path to a file. The server returns back the length of the file and its content in the case the file exists. When the file does not exist the server returns a length of -1 and no content. The client will store the content in a file with the same name as the input file with the suffix *�copy* appended (ex: for f.txt => f.txt-copy).

3.   The server chooses a random float number <SRF>. Run multiple clients. Each client chooses a random float number <CRF> and send it to the server. When the server does not receive any incoming connection for at least 10 seconds it chooses the client that has *guessed* the best approximation (is closest) for its own number and sends it back the message *�You have the best guess with an error of <SRV>-<CRF>�*. It also sends to each other client the string *�You lost !�.* The server closes all connections after this.

4.   The clients send an integer number N and an array of N float values. The server will merge sort the numbers received from all clients until it gets an empty array of floats (N=0). The server returns to each client the size of the merge-sorted array followed by the merge-sorted arrays of all floats from all clients.

5.   The client sends a domain name taken from the command line (Ex: [www.google.com](http://www.google.com/)) to the server. The server opens a TCP connection to the IP address corresponding to the received domain name on port 80 (called HTTP-Srv). It sends on the TCP connection the string: *�GET / HTTP/1.0\n\n�* and relays the answer back to the client. When *HTTP-Srv* closes connection to the server, the server closes the connection to the client at its turn.

6.   The server chooses a random integer number. Each client generates a random integer number and send it to the server. The server answers with the message �larger� if the client has sent a smaller number than the server�s choice, or with message �smaller� if the client has send a larger number than the server�s choice. Each client continues generating a different random number (larger or smaller than the previous) according to the server�s indication. When a client guesses the server choice � the server sends back to the winner the message �You win � within x tries�. It also sends back to all other clients the message �You lost � after y retries!� (x and y are the number of tries of each respective client). The server closes all connections upon a win and it chooses a different random integer for the next game (set of clients)

7.   The client reads a username and a password from the standard input. It sends the username to the server. The server uses the *getpwent* system call repeatedly to find the password information about the username. If the entry for the username is found, the password field from the *struct passwd* is returned to the client. The client recovers the salt of the password and checks the input password with the received encrypted version using the *crypt* system call. If there is no user *username*, the server returns back to the client the empty string and closes the connection.

8.   Change the Python Example Concurrent Number Guess Bellow to transmit with each answer to a client � the number of total clients that are competing. Also change the client to read the numbers from the standard input and launch a contest in the class during the lab

**Problem Definition**

**Write a Client Server implementation where the client sends a string to the server**

**and the server counts the number of spaces inside the string. The problems should**

**handle all error and abnormal cases**

|  |  |
| --- | --- |
| **Client – Character-Count** | **Server – Character-Count** |
| #include <sys/socket.h>  #include <netinet/in.h>  #include <stdio.h>  #include <string.h>  #include <stdint.h>    #define max 100    int main**() {**    int c**,** cod**;**    int32\_t r**;**  *// Observatie: Deoarece dimensiunea tipului int difera de la platforma la platforma,*  *// (spre exemplu, in Borland C in DOS e reprezentat pe 2 octeti, iar in C sub Linux pe*  *// 4 octeti) este necesara utilizarea unor tipuri intregi standard. A se vedea*  *// stdint.h.*    struct sockaddr\_in server**;**    char s**[**max**];**      c **=** socket**(**PF\_INET**,** SOCK\_STREAM**,** 0**);**    if **(**c **<** 0**) {**      fprintf**(**stderr**,** "Eroare la creare socket client.\n"**);**      return 1**;**  **}**      memset**(&**server**,** 0**,** sizeof**(**struct sockaddr\_in**));**    server**.**sin\_family **=** AF\_INET**;**    server**.**sin\_port **=** htons**(**4321**);**    server**.**sin\_addr**.**s\_addr **=** inet\_addr**(**"127.0.0.1"**);**      cod **=** connect**(**c**, (**struct sockaddr **\*) &**server**,** sizeof**(**struct sockaddr\_in**));**    if **(**cod **<** 0**) {**      fprintf**(**stderr**,** "Eroare la conectarea la server.\n"**);**      return 1**;**  **}**      printf**(**"Dati o fraza pentru trimis la server: "**);**    fgets**(**s**,** max**,** stdin**);**    *// !!! important - trimit lungimea sirului + 1 pentru a trimite pe socket si caracterul NULL (0) care marcheaza sfarsitului sirului.*  *// paragraful 1 din protocol*    cod **=** send**(**c**,** s**,** strlen**(**s**) +** 1**,** 0**);**    if **(**cod **!=** strlen**(**s**) +** 1**) {**      fprintf**(**stderr**,** "Eroare la trimiterea datelor la server.\n"**);**      return 1**;**  **}**    *// paragraful 5 din protocol*    cod **=** recv**(**c**, &**r**,** sizeof**(**int32\_t**),** MSG\_WAITALL**);**    r **=** ntohl**(**r**);**    if **(**cod **!=** sizeof**(**int**)) {**      fprintf**(**stderr**,** "Eroare la primirea datelor de la client.\n"**);**      return 1**;**  **}**      printf**(**"Serverul a returnat %d caractere spatiu in sirul trimis.\n"**,** r**);**      close**(**c**);**  **}** | #include <sys/socket.h>  #include <netinet/in.h>  #include <stdio.h>  #include <string.h>  #include <stdint.h>  #include <signal.h>  #include <unistd.h>  #include <stdlib.h>    int c**;**    *// Mecanismul de time-out. Paragraful 3.*  void time\_out**(**int semnal**) {**    int32\_t r **= -**1**;**    r **=** htonl**(**r**);**    printf**(**"Time out.\n"**);**    send**(**c**, &**r**,** sizeof**(**int32\_t**),** 0**);**    close**(**c**);** *// desi nu am primit nimic de la client in 10 secunde, inchidem civilizat conexiunea cu acesta*    exit**(**1**);**  **}**    void tratare**() {**    int cod**;**    int32\_t r**;**    uint8\_t b**;**  *// Observatie: Deoarece dimensiunea tipului int difera de la platforma la platforma,*  *// (spre exemplu, in Borland C in DOS e reprezentat pe 2 octeti, iar in C sub Linux pe*  *// 4 octeti) este necesara utilizarea unor tipuri intregi standard. A se vedea*  *// stdint.h.*    struct sockaddr\_in server**;**      if **(**c **<** 0**) {**      fprintf**(**stderr**,** "Eroare la stabilirea conexiunii cu clientul.\n"**);**      exit**(**1**);**  **}**    else      printf**(**"Un nou client s-a conectat cu succes.\n"**);**      signal**(**SIGALRM**,** time\_out**);**    alarm**(**10**);**      r **=** 0**;** *// rezultatul, numarul de caractere spatii primite de la client*    do **{**      cod **=** recv**(**c**, &**b**,** 1**,** 0**);**      printf**(**"Am primit %d caractere.\n"**,** cod**);**        if **(**cod **==** 1**)** *// citire cu succes a unui caracter*        alarm**(**10**);***// resetam timerul si asteptam din nou 10 secunde urmatorul caracter*        if **(**cod **!=** 1**) {**        r **= -**1**;**        break**;**  **}**        if **(**b **==** ' '**) {**  *// Paragraful 5 din protocolul*        if **(**r **==** INT32\_MAX**) {** *// intregul maxim pozitiv pe 4 octetii cu semn sau 0x7FFFFFFF (a se vedea stdint.h)*          r **= -**2**;**          break**;**  **}**        r**++;**  **}**  **}**    while **(**b **!=** 0**);** *// sirul de caractere de la client se considera terminat la intalnirea caracterului 0 (NULL, '\0')*  *// Paragraful 2 - terminam citirea sirului de la client la primirea caracterului NULL*      alarm**(**0**);** *// oprim timerul*      r **=** htonl**(**r**);**    send**(**c**, &**r**,** sizeof**(**int32\_t**),** 0**);**    r **=** ntohl**(**r**);**      close**(**c**);**      if **(**r **>=** 0**)**      printf**(**"Am inchis cu succes conexiunea cu un client. I-am trimis %d spatii.\n"**,** r**);**    else **{**      printf**(**"Am inchis cu eroare conexiunea cu un client. Cod de eroare %d.\n"**,** r**);**      exit**(**1**);**  **}**      exit**(**0**);**  *// Terminam procesul fiu - foarte important!!! altfel numarul de procese creste exponential.*  *// Fiul se termina dupa ce deserveste clientul.*  **}**      int main**() {**    int s**,** l**,** cod**;**    struct sockaddr\_in client**,** server**;**      s **=** socket**(**PF\_INET**,** SOCK\_STREAM**,** 0**);**    if **(**s **<** 0**) {**      fprintf**(**stderr**,** "Eroare la creare socket server.\n"**);**      return 1**;**  **}**      memset**(&**server**,** 0**,** sizeof**(**struct sockaddr\_in**));**    server**.**sin\_family **=** AF\_INET**;**    server**.**sin\_port **=** htons**(**4321**);**   server**.**sin\_addr**.**s\_addr **=** INADDR\_ANY**;**      cod **=** bind**(**s**, (**struct sockaddr **\*) &**server**,** sizeof**(**struct sockaddr\_in**));**    if **(**cod **<** 0**) {**      fprintf**(**stderr**,** "Eroare la bind. Portul este deja folosit.\n"**);**      return 1**;**  **}**      listen**(**s**,** 5**);**      while **(**1**) {** *// deserveste oricati clienti*        memset**(&**client**,** 0**,** sizeof**(**client**));**      l **=** sizeof**(**client**);**        printf**(**"Astept sa se conecteze un client.\n"**);**      c **=** accept**(**s**, (**struct sockaddr **\*) &**client**, &**l**);**      printf**(**"S-a conectat clientul cu adresa %s si portul %d.\n"**,** inet\_ntoa**(**client**.**sin\_addr**),** ntohs**(**client**.**sin\_port**));**        if **(**fork**() ==** 0**) {** *// server concurent, conexiunea va fi tratata de catre un proces fiu separat*        tratare**();**  **}**  *// parintele continua bucla while asteptand un nou client*  **}**    **}** |

A client server implementation in python for the number guess problem. The server chooses a random number. The clients connect and send numbers to server. The server returns to each client a status message:

·         ‘H’ – send a larger number

·         ‘S’ – send a lower number

·         ‘G’ – you guessed my number

·         ‘L’ – another client guessed the number. You are a looser !

The implementation bellow is a concurrent multithreaded - didactic approach to the problem – not checking for every possible error. Also pay attention to thread synchronization mechanisms and the way a simple solution becomes complicated when threads need to be synchronized

|  |  |
| --- | --- |
| **Client** | **Server** |
| \_\_author\_\_ = **'dadi' import**socket, struct, random,sys, time   **if**\_\_name\_\_ == **'\_\_main\_\_'**:     **try**:         s = socket.create\_connection( (**'localhost'**,1234))     **except**socket.error **as**msg:         print(**"Error: "**,msg.strerror)         exit(-1)      finished=**False**sr = 1; er=2\*\*17-1     random.seed()      data=s.recv(1024)     print(data.decode(**'ascii'**))     step\_count=0     **while not**finished:         my\_num=random.randint(sr,er)         **try**:             s.sendall( struct.pack(**'!I'**,my\_num) )             answer=s.recv(1)         **except**socket.error **as**msg:             print(**'Error: '**,msg.strerror)             s.close()             exit(-2)         step\_count+=1         print(**'Sent '**,my\_num,**' Answer '**,answer.decode(**'ascii'**))         **if**answer==**b'H'**:             sr = my\_num         **if**answer==**b'S'**:             er = my\_num         **if**answer==**b'G' or**answer==**b'L'**:             finished=**True**time.sleep(0.25)      s.close()     **if**answer==**b'G'**:         print(**"I am the winner with"**,my\_num,**"in"**, step\_count,**"steps"**)     **else**:         print(**"I lost !!!"**) *#    input("Press Enter")* | \_\_author\_\_ = **'dadi'  import**socket **import**threading **import**random **import**struct **import**sys **import**time  random.seed() start=1; stop=2\*\*17-1 my\_num= random.randint(start,stop) print(**'Server number: '**,my\_num) mylock = threading.Lock() client\_guessed=**False** winner\_thread=0 e = threading.Event() e.clear() threads = [] client\_count=0  **def**worker(cs):     **global**mylock, client\_guessed, my\_num, winner\_thread, client\_count,e      my\_idcount=client\_count     print(**'client #'**,client\_count,**'from: '**,cs.getpeername(), cs )     message=**'Hello client #'**+str(client\_count)+**' ! You are entering the number guess competion now !'**cs.sendall(bytes(message,**'ascii'**))      **while not**client\_guessed:         **try**:             cnumber=cs.recv(4)             cnumber=struct.unpack(**'!I'**,cnumber)[0]             **if**cnumber > my\_num:                 cs.sendall(**b'S'**)             **if**cnumber < my\_num:                 cs.sendall(**b'H'**)             **if**cnumber== my\_num:                 mylock.acquire()                 client\_guessed=**True**winner\_thread=threading.get\_ident()                 mylock.release()          **except**socket.error **as**msg:             print(**'Error:'**,msg.strerror)             **break      if**client\_guessed:         **if**threading.get\_ident() == winner\_thread:             cs.sendall(**b'G'**)             print(**'We have a winner'**, cs.getpeername())             print(**"Thread "**,my\_idcount,**" winner"**)             e.set()         **else**:             cs.sendall(**b'L'**)             print(**"Thread "**,my\_idcount,**" looser"**)     time.sleep(1)     cs.close()     print(**"Worker Thread "**,my\_idcount, **" end"**)   **def**resetSrv():     **global**mylock, client\_guessed, winner\_thread, my\_num, threads,e, client\_count     **while True**:         e.wait()         **for**t **in**threads:             t.join()         print(**"all threads are finished now"**)         e.clear()         mylock.acquire()         threads = []         client\_guessed = **False**winner\_thread=-1         client\_count = 0         my\_num = random.randint(start,stop)         print(**'Server number: '**,my\_num)         mylock.release()   **if**\_\_name\_\_==**'\_\_main\_\_'**:     **try**:         rs=socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)         rs.bind( (**'0.0.0.0'**,1234) )         rs.listen(5)     **except**socket.error **as**msg:         print(msg.strerror)         exit(-1)     t=threading.Thread(target=resetSrv, daemon=**True**)     t.start()     **while True**:         client\_socket,addrc = rs.accept()         t = threading.Thread(target=worker, args=(client\_socket,) )         threads.append(t)         client\_count+=1         t.start() |