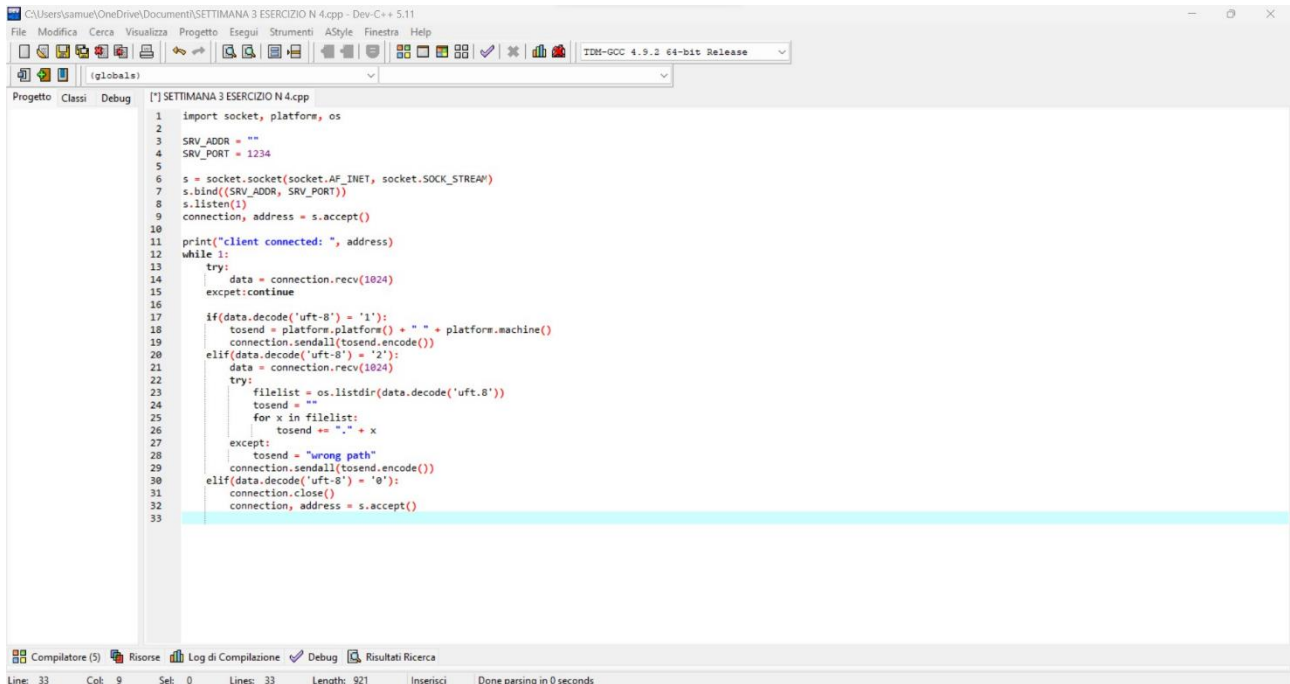


## SETTIMANA 3 COMPITO N° 4



```
1 import socket, platform, os
2
3 SRV_ADDR = ""
4 SRV_PORT = 1234
5
6 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
7 s.bind((SRV_ADDR, SRV_PORT))
8 s.listen(1)
9 connection, address = s.accept()
10
11 print("client connected: ", address)
12 while 1:
13     try:
14         data = connection.recv(1024)
15         except: continue
16
17         if(data.decode('uft-8') == '1'):
18             tosend = platform.platform() + " " + platform.machine()
19             connection.sendall(tosend.encode())
20         elif(data.decode('uft-8') == '2'):
21             data = connection.recv(1024)
22             try:
23                 filelist = os.listdir(data.decode('uft-8'))
24                 tosend = ""
25                 for x in filelist:
26                     tosend += " " + x
27             except:
28                 tosend = "wrong path"
29             connection.sendall(tosend.encode())
30         elif(data.decode('uft-8') == '0'):
31             connection.close()
32             connection, address = s.accept()
33
```

Spiegazione del programma:

Come libreria importata abbiamo usato << import socket, platform, os >> (socket fornisce le funzionalità di rete, platform è un modulo che serve per recuperare più informazioni possibili sul computer su cui si sta eseguendo il programma, invece OS serve per le funzionalità del sistema operativo, tipo creare cartelle dal programma, aprire e chiudere file e cartelle, gestione dei processi).

Poi abbiamo usato << SRV\_ADDR e SRV\_PORT >> che rispettivamente il primo contiene l'indirizzo IP del server, mentre il secondo contiene il numero della porta su cui il server ascolterà le connessioni d'ingresso.

Poi abbiamo usato << s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) >> Il primo parametro (socket.AF\_INET) specifica che si tratta di un socket di tipo IPv4, e il secondo parametro (socket.SOCK\_STREAM) specifica che si tratta di un socket TCP.

Subito dopo abbiamo usato << s.bind((SRV\_SADDR, SRV\_PORT)) >> associa il socket all'indirizzo IP e alla porta specificati con bind(). In questo modo, il server è in ascolto su quell'indirizzo IP e porta per le connessioni in ingresso.

In seguito abbiamo usato il comando << s.listen(1) >> Mette il socket in modalità "ascolto" utilizzando listen(). Il parametro 1 indica che il server può accettare una sola connessione in entrata alla volta.

Successivamente usiamo << connection, address = s.accept() >> accetta una connessione in entrata utilizzando << accept() >>. Questo metodo blocca il programma finché non viene stabilita una connessione da un client. connection è il nuovo socket creato per comunicare con il client, e address contiene l'indirizzo IP e la porta del client.

Poi abbiamo usato << print(Client connected, address) >> che stampa un messaggio per indicare che un client è stato connesso e visualizza l'indirizzo del client.

Qui abbiamo usato il comando << while 1 >> e << data = connection.recv(1024) >> entra in un ciclo while infinito

per ricevere dati dal client. Utilizza `recv(1024)` per ricevere dati dal client, con una dimensione massima di 1024 byte alla volta. Con il comando `<< except:continue >>`

Anche se non dovessero più arrivare dati il ciclo `while` non si interrompe.

Invece tutti i codici inseriti dal rigo 17 al rigo 32

```
17     if(data.decode('uft-8') == '1'):
18         tosend = platform.platform() + " " + platform.machine()
19         connection.sendall(tosend.encode())
20     elif(data.decode('uft-8') == '2'):
21         data = connection.recv(1024)
22         try:
23             filelist = os.listdir(data.decode('uft.8'))
24             tosend = ""
25             for x in filelist:
26                 tosend += "." + x
27         except:
28             tosend = "wrong path"
29         connection.sendall(tosend.encode())
30     elif(data.decode('uft-8') == '0'):
31         connection.close()
32     connection, address = s.accept()
```

sono una serie di comandi che servono per dire all'utente che input da dare per far funzionare il programma es:

-Premendo 1 send (`connection.sendall(tosend.encode())`) in base alla piattaforma (`platform.platform()`) e alla macchina (`platform.machine`) che si sta usando ci invierà dei dati.

-Premendo 2 il programma cerca di fare una lista delle cartelle (`<< filelist >>`) e poi dei file (`<< os.listdir >>`) elencandoli tramite una virgola (`<< tosend += "," + x >>`) se tutto riesce avrai un elenco dei file, mentre se qualcosa va

male uscirà scritto wrong path (ovvero percorso sbagliato) grazie al comando << tosend = "wrong\_path" >>.

-Premendo 0 il programma verrà chiuso.