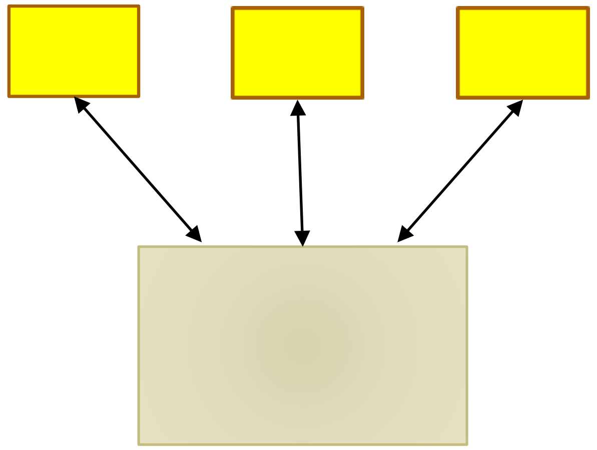
IMPLEMENTAREA CONCURENTEI IN LIMBAJE DE PROGRAMARE

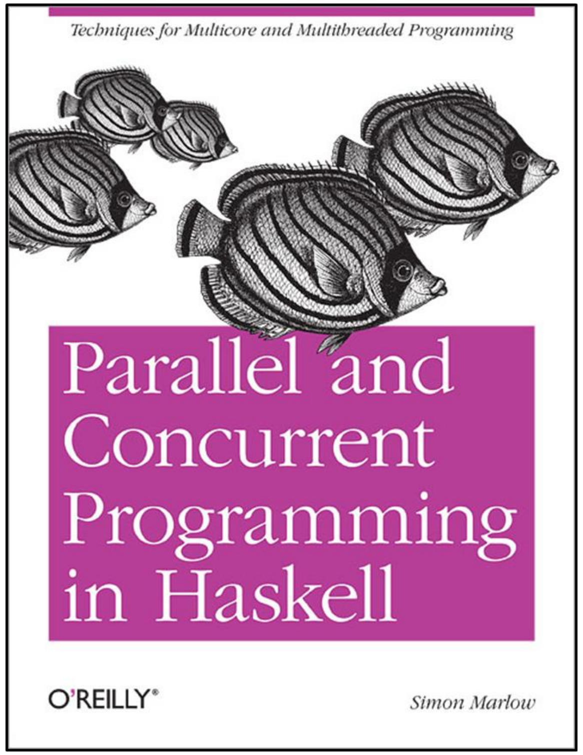
Concurenta

Implementarea unei aplicatii de tip SERVER

Ioana Leustean

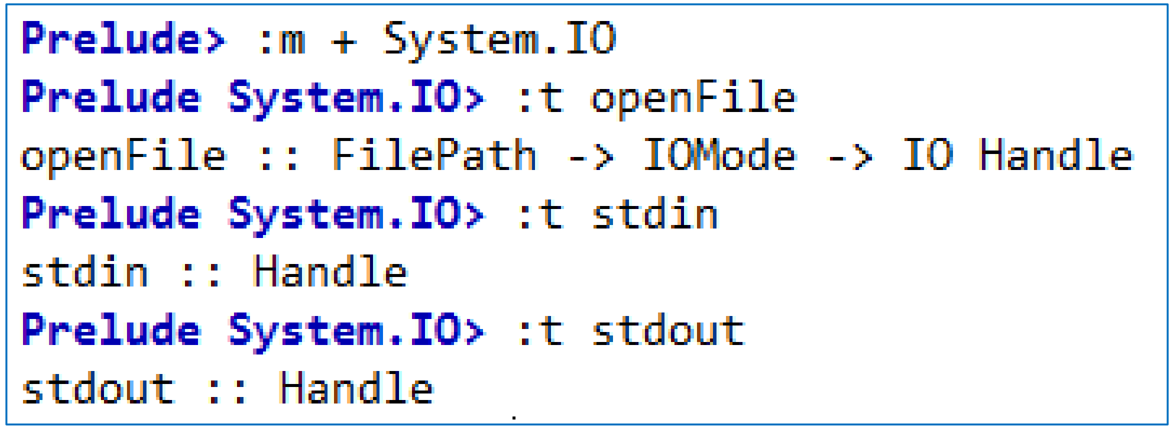


Haskell



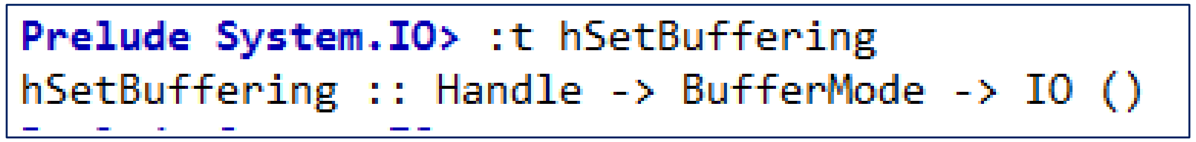
Part II. Concurrent Haskell
Cap. 12

➢ System.IO



```
Prelude> :m + System.IO
Prelude System.IO> :t openFile
openFile :: FilePath -> IOMode -> IO Handle
Prelude System.IO> :t stdin
stdin :: Handle
Prelude System.IO> :t stdout
stdout :: Handle
```

```
data Handle   – stdin, stdout
type FilePath =  String
data IOMode   =  ReadMode| WriteMode|
                 AppendMode| ReadWriteMode
```



```
Prelude System.IO> :t hSetBuffering
hSetBuffering :: Handle -> BufferMode -> IO ()
```

O data de tip Handle este o valoare extrasa dintr-o actiune IO si desemneaza fisierul curent

```
hdl <- openFile "fis.txt" ReadMode
hclose hdl
```

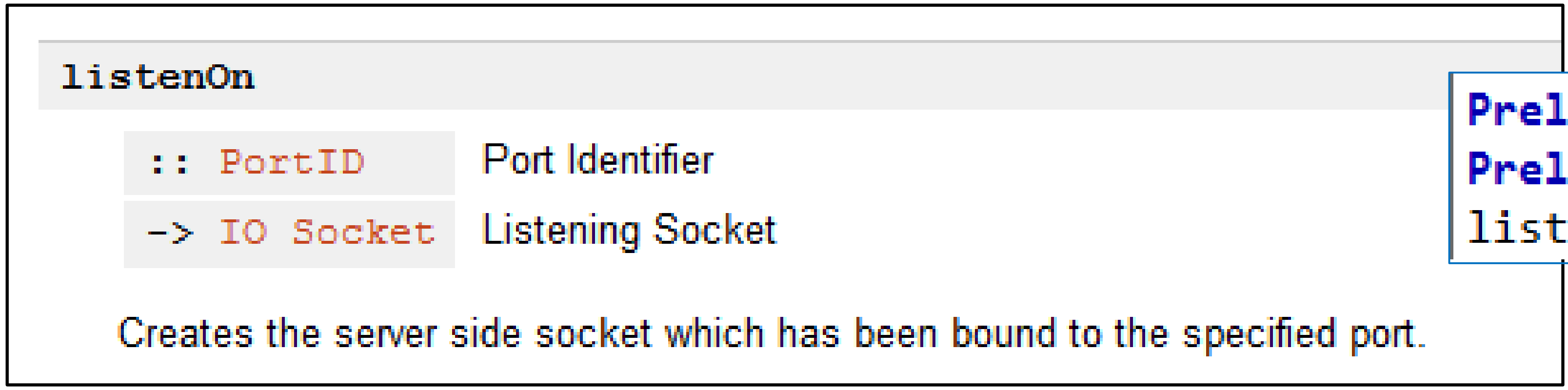
```
data BufferMode =  NoBuffering|
                   LineBuffering
                   BlockBuffering (Maybe Int)
```

- System.IO

```
import System.IO


exio1 = do
      hdl1 <- openFile "f1.txt" ReadMode
      hdl2 <- openFile "f2.txt" AppendMode
      s  <- hGetContents hdl1
      putStrLn s
      hPutStr hdl2 s
      hClose hdl1
      hClose hdl2
```

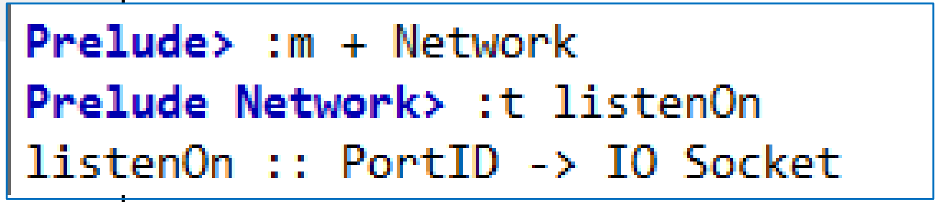
```
exio2 = do
      s <- readFile "f1.txt"
      putStrLn s
      writeFile "f2.txt" s
```

- **Network**
- **Server-side connections**

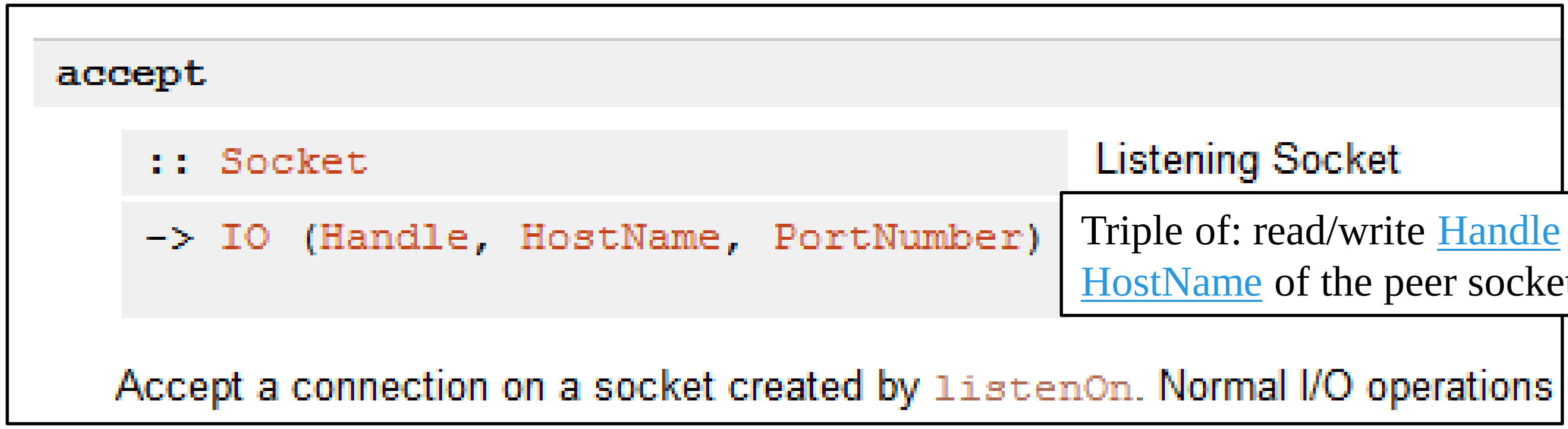


```
listenOn
   :: PortID        Port Identifier
   -> IO Socket     Listening Socket
```

Creates the server side socket which has been bound to the specified port.



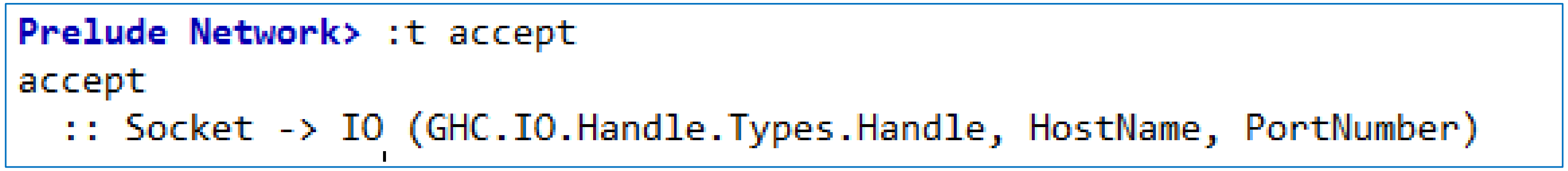
```
Prelude> :m + Network
Prelude Network> :t listenOn
listenOn :: PortID -> IO Socket
```



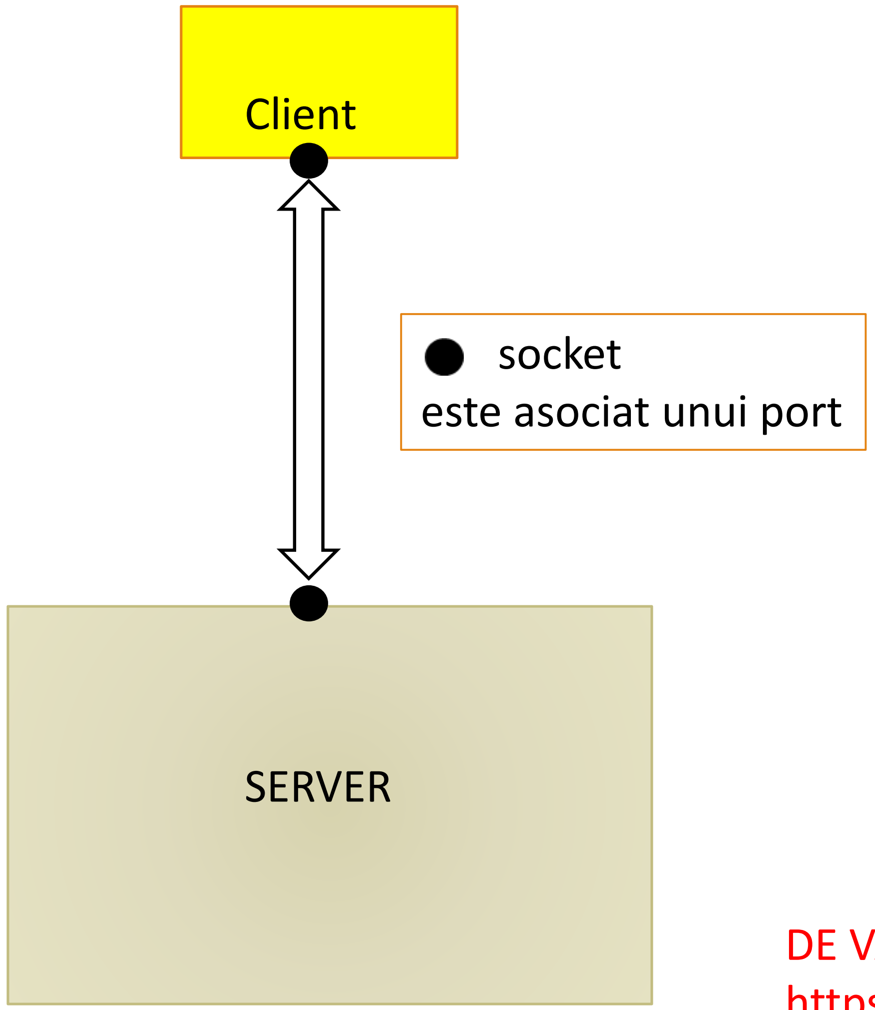
```
accept
   :: Socket                                   Listening Socket
   -> IO (Handle, HostName, PortNumber)
```

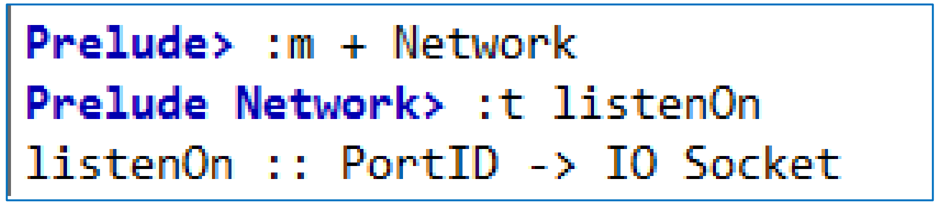
Triple of: read/write Handle for communicating with the client, the HostName of the peer socket, and the PortNumber of the remote connection.

Accept a connection on a socket created by `listenOn`. Normal I/O operations



```
Prelude Network> :t accept
accept
  :: Socket -> IO (GHC.IO.Handle.Types.Handle, HostName, PortNumber)
```



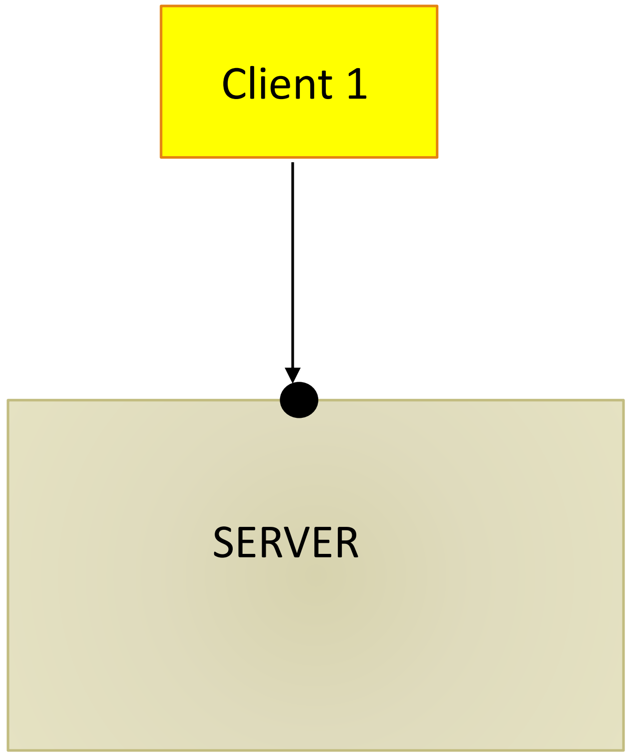


```
Prelude> :m + Network
Prelude Network> :t listenOn
listenOn :: PortID -> IO Socket
```

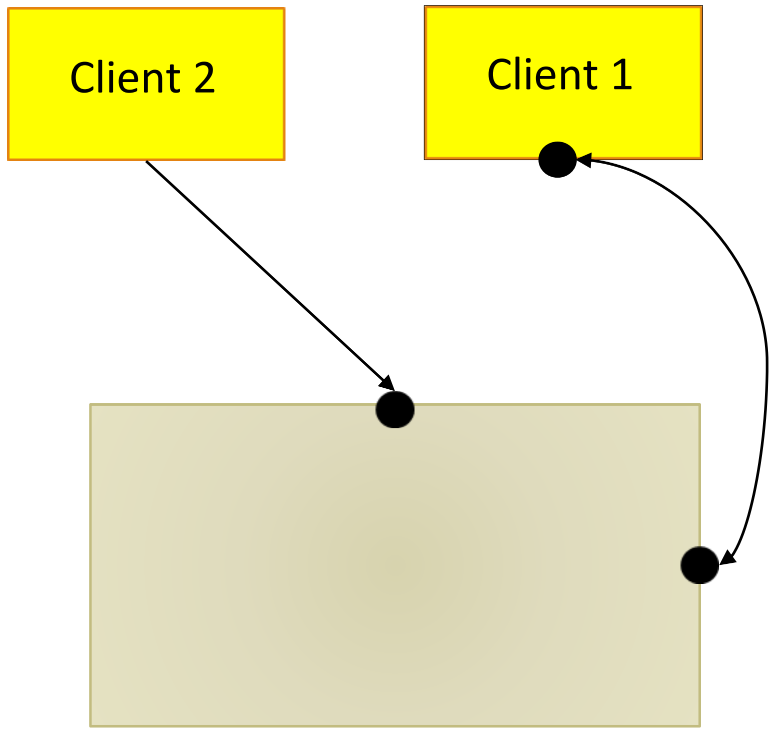
```
port :: Int
port = 44444
socket <- listenOn (PortNumber (fromIntegral  port))
```

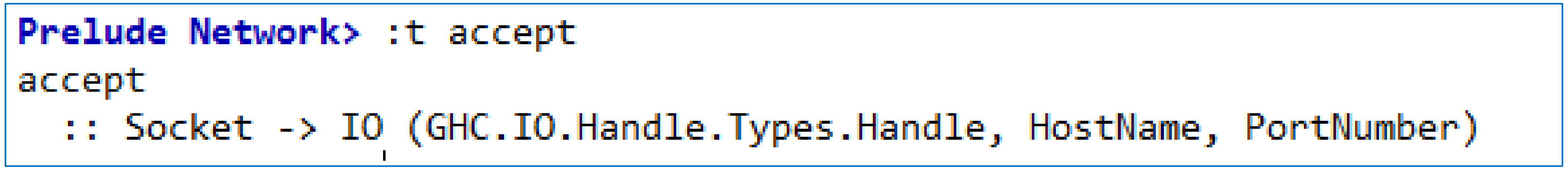
DE VAZUT
https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

```
port :: Int
port = 44444
socket <- listenOn (PortNumber (fromIntegral  port))
```



```
(handle, host, port) <- accept socket
```





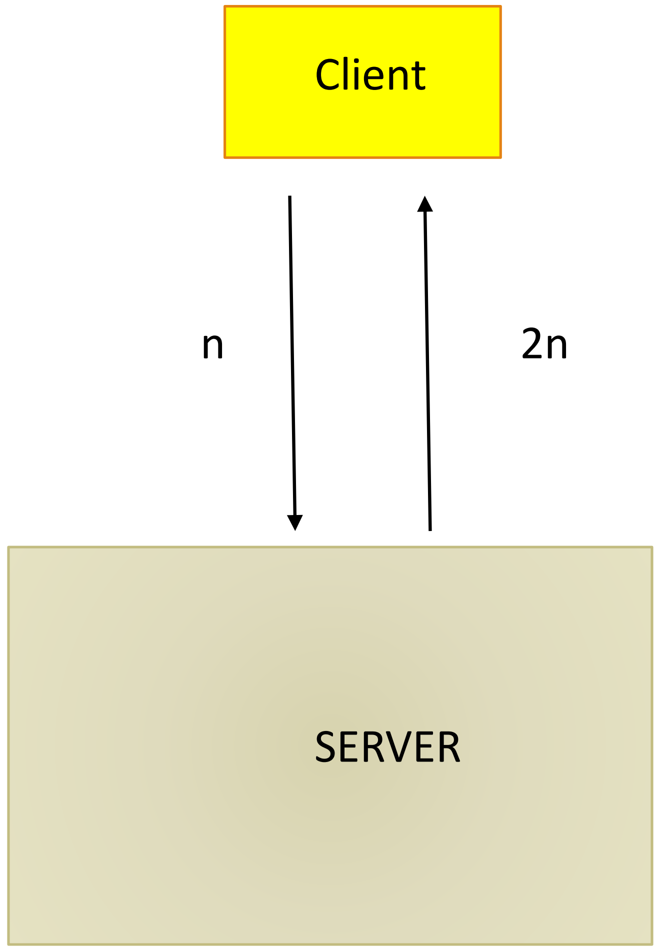
```
Prelude Network> :t accept
accept
  :: Socket -> IO (GHC.IO.Handle.Types.Handle, HostName, PortNumber)
```

serverer.hs

```haskell
port :: Int
port = 44444


main = withSocketsDo $ do
  sock <- listenOn (PortNumber (fromIntegral port))
  printf "Listening on port %d\n" port
  forever $ do
        (handle, host, port) <- accept sock
        printf "Accepted connection from %s: %s\n" host (show port)
        forkIO $ (talk handle)
```

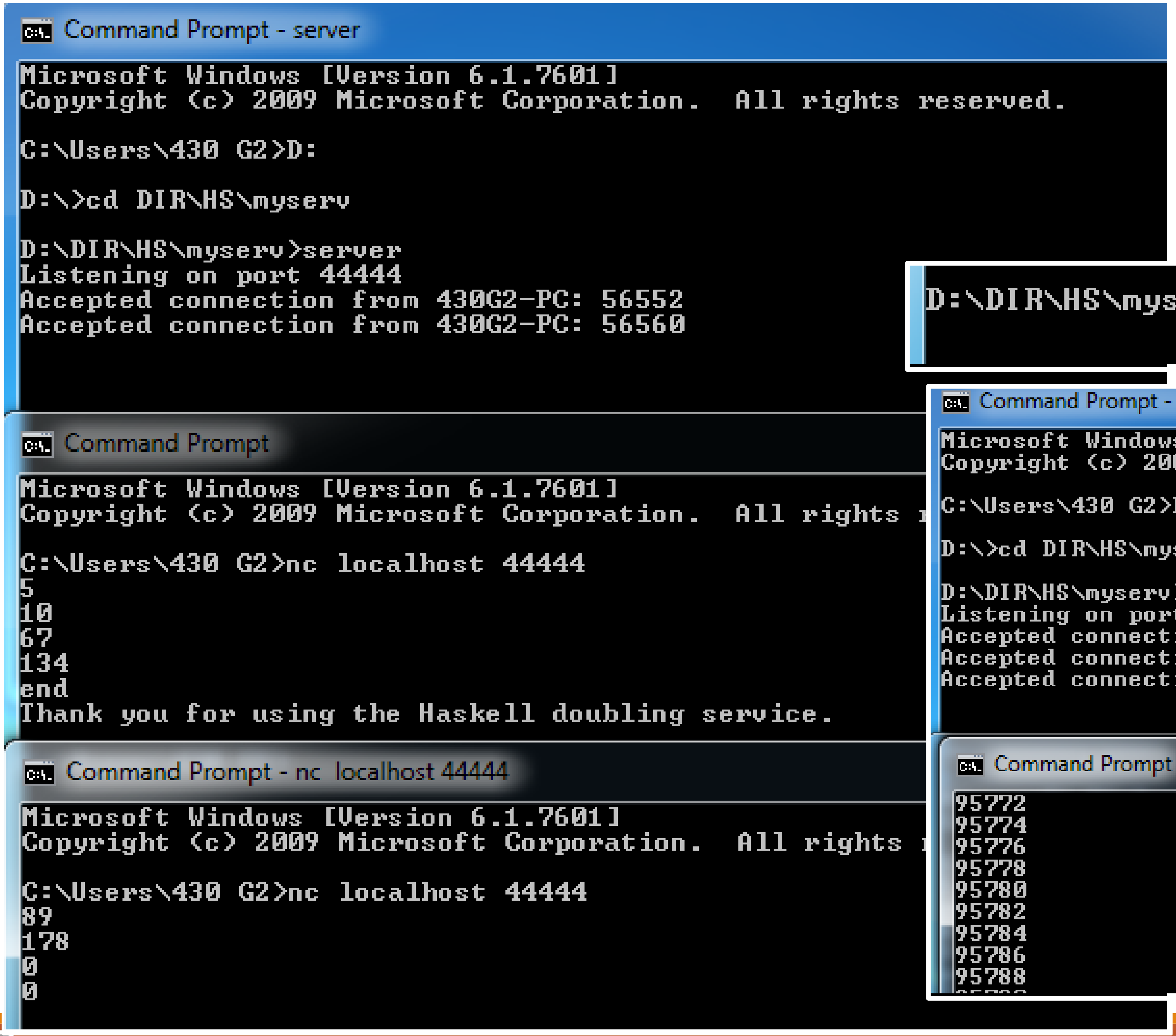
Se creaza un thread pentru fiecare "canal de comunicare" intre server si client.

valoarea h de tip Handle este asociata unui client si este furnizata de un socket



server.hs

```
talk :: Handle -> IO ()
talk h = do
    hSetBuffering h  LineBuffering
    loop
  where
    loop = do
         line <- hGetLine h
         if line == "end"
             then hPutStrLn h ("Thank you for using the "
                                ++ "Haskell doubling service.")
             else do hPutStrLn h (show (2 * (read line :: Integer)))
                     loop
```



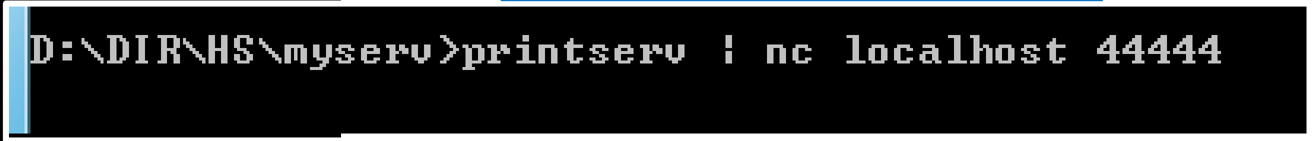
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\430 G2>D:

D:\>cd DIR\HS\myserv

D:\DIR\HS\myserv>server
Listening on port 44444
Accepted connection from 430G2-PC: 56552
Accepted connection from 430G2-PC: 56560
```

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights r

C:\Users\430 G2>nc localhost 44444
5
10
67
134
end
Thank you for using the Haskell doubling service.
```

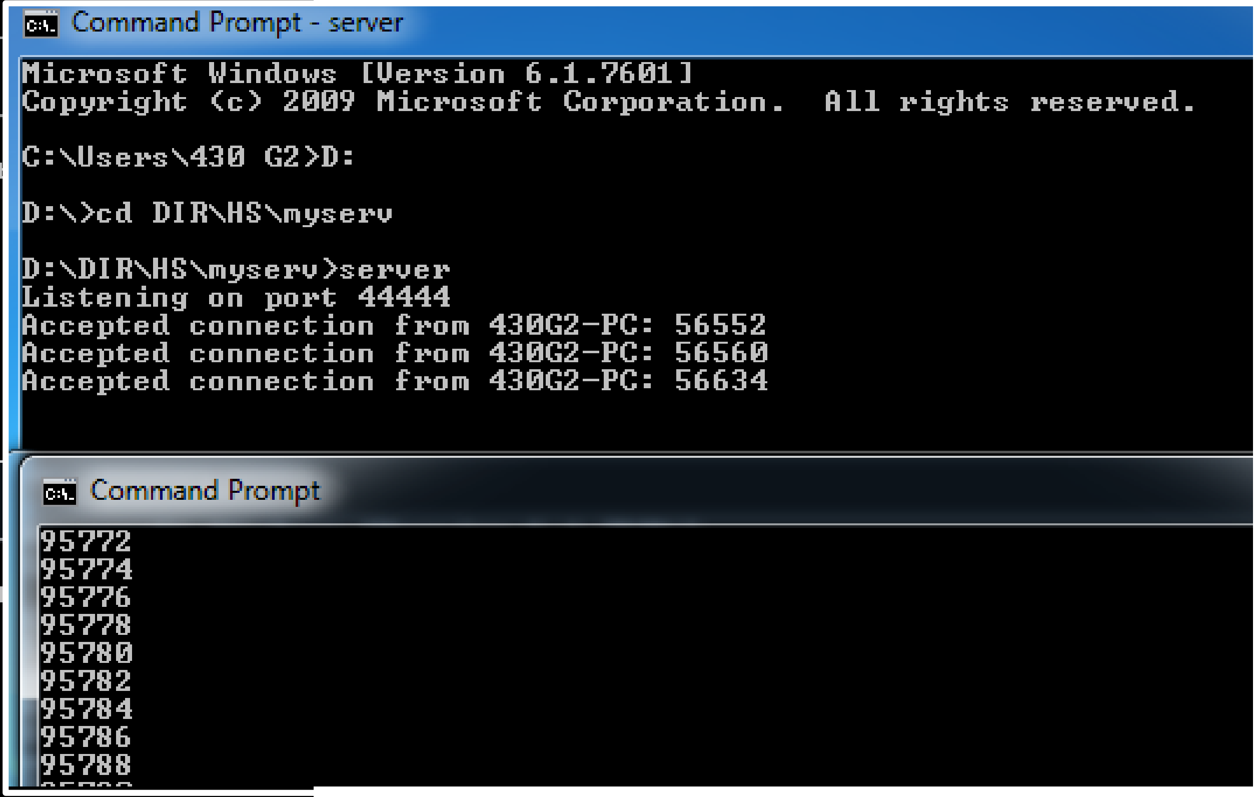
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights r

C:\Users\430 G2>nc localhost 44444
89
178
0
0
```

printserv.hs

```
import Control.Monad
main = mapM_ print [1..]
```



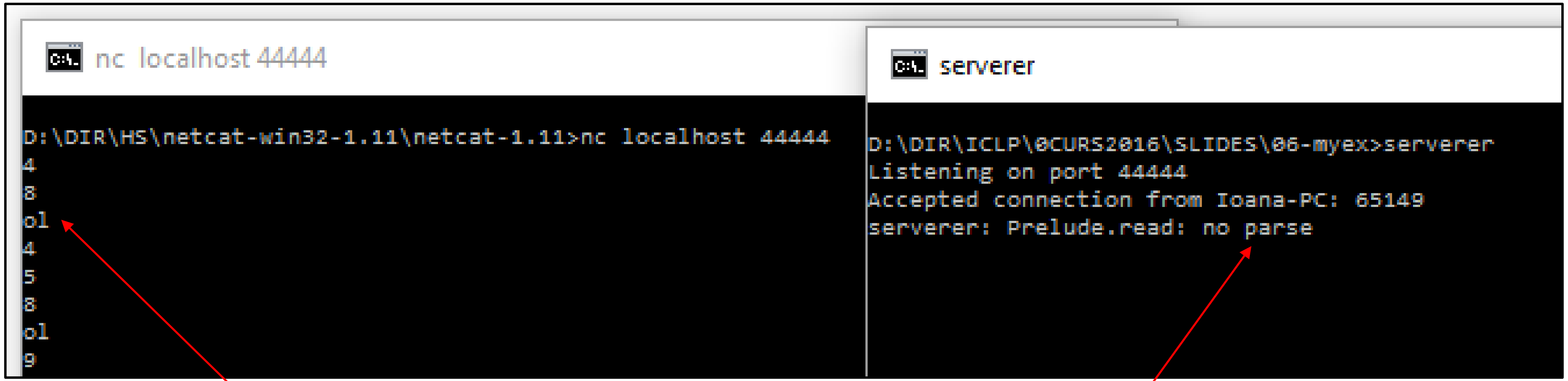
```
D:\DIR\HS\myserv>printserv | nc localhost 44444
```



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\430 G2>D:

D:\>cd DIR\HS\myserv

D:\DIR\HS\myserv>server
Listening on port 44444
Accepted connection from 430G2-PC: 56552
Accepted connection from 430G2-PC: 56560
Accepted connection from 430G2-PC: 56634
```

```
95772
95774
95776
95778
95780
95782
95784
95786
95788
```

https://www.haskell.org/hoogle/



```
D:\DIR\HS\netcat-win32-1.11\netcat-1.11>nc localhost 44444
4
8
ol
4
5
8
ol
9
```

```
D:\DIR\ICLP\0CURS2016\SLIDES\06-myex>serverer
Listening on port 44444
Accepted connection from Ioana-PC: 65149
serverer: Prelude.read: no parse
```

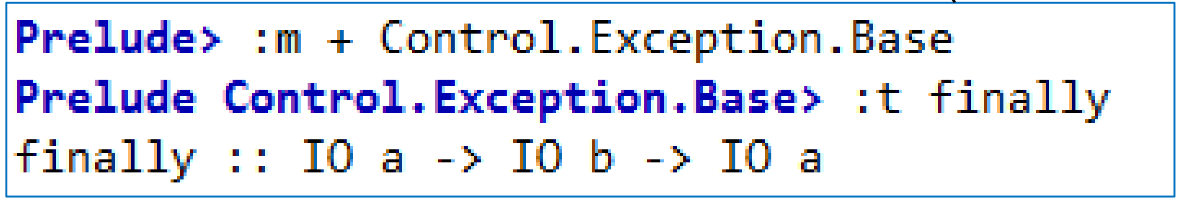
valoare nenumerica

mesaj de eroare

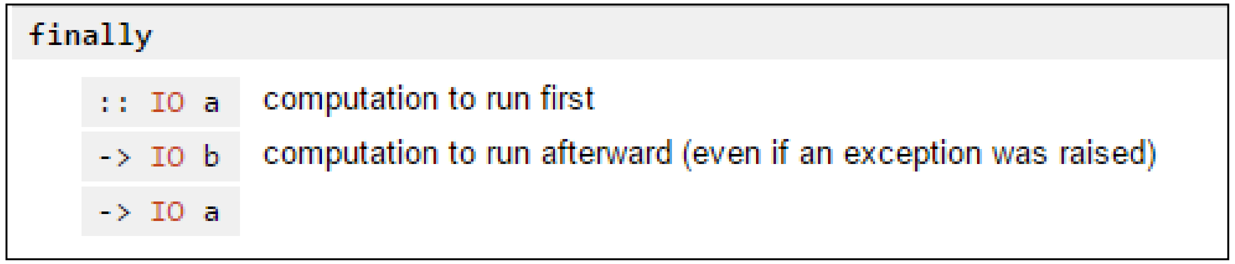
server.hs

```haskell
port :: Int
port = 44444

main = withSocketsDo $ do
  sock <- listenOn (PortNumber (fromIntegral port))
  printf "Listening on port %d\n" port
  forever $ do
        (handle, host, port) <- accept sock
        printf "Accepted connection from %s: %s\n" host (show port)
        forkIO $ (talk handle) `finally` (hClose handle)
```

tratarea erorilor

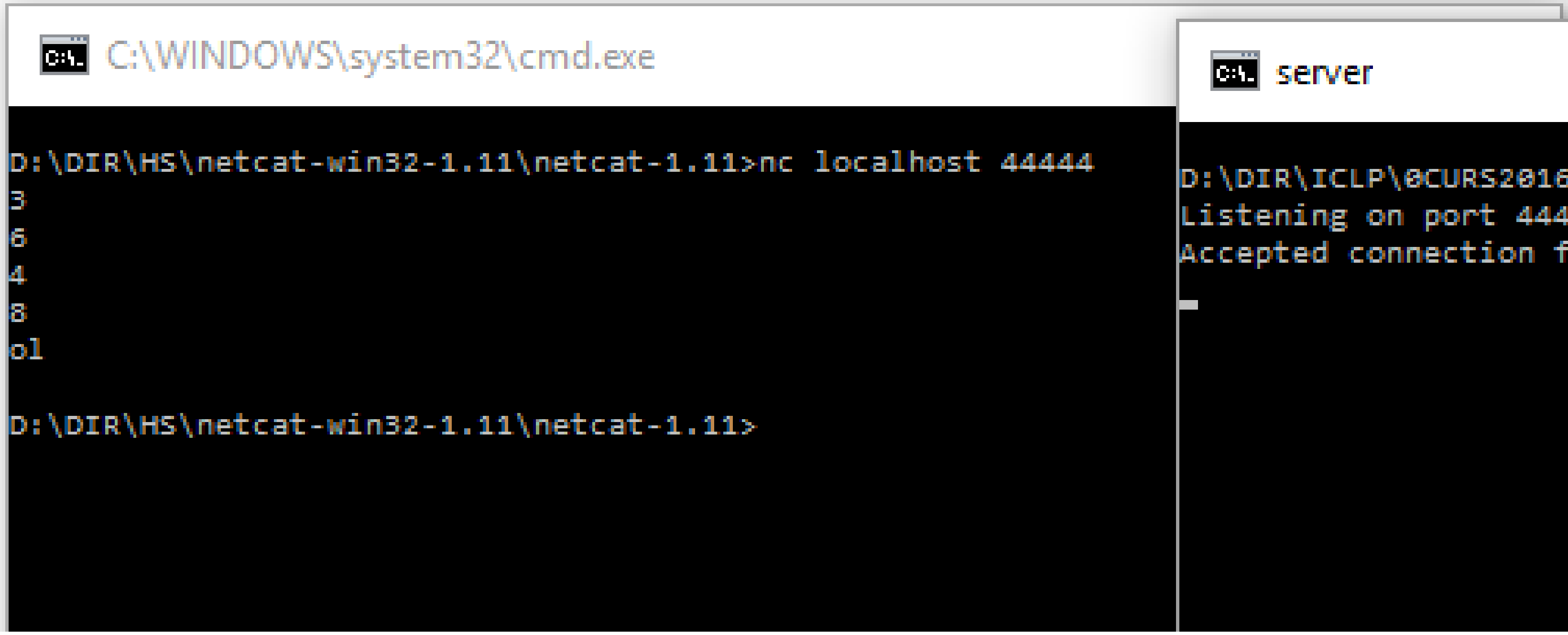


```
Prelude> :m + Control.Exception.Base
Prelude Control.Exception.Base> :t finally
finally :: IO a -> IO b -> IO a
```

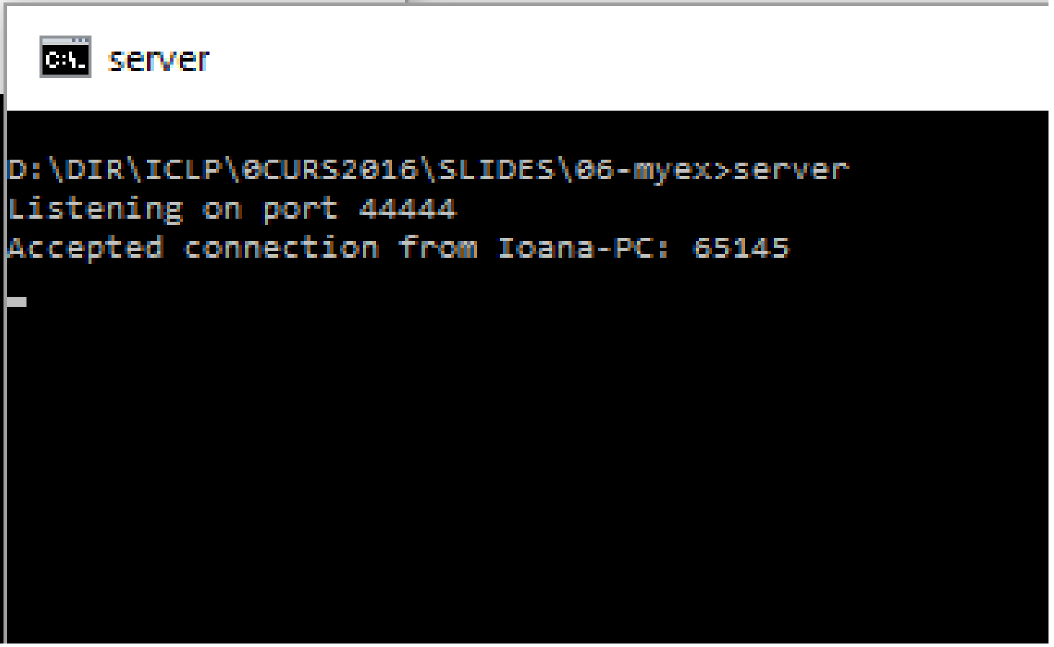


finally	
:: IO a	computation to run first
-> IO b	computation to run afterward (even if an exception was raised)
-> IO a	

Functia finally executa prima actiune si apoi o executa pe a doua, chiar daca prima s-a terminat cu eroare.


C:\WINDOWS\system32\cmd.exe

```
D:\DIR\HS\netcat-win32-1.11\netcat-1.11>nc localhost 44444
3
6
4
8
ol

D:\DIR\HS\netcat-win32-1.11\netcat-1.11>
```


server

```
D:\DIR\ICLP\0CURS2016\SLIDES\06-myex>server
Listening on port 44444
Accepted connection from Ioana-PC: 65145
```

➢ Server cu stare partajata

"The new behavior is as follows: instead of multiplying each number by two, the server will multiply each number by the current factor. Any connected client can change the current factor by sending the command *N, where N is an integer. When a client changes the factor, the server sends a message to all the other connected clients informing them of the change.

While this seems like a small change in behavior, it introduces some interesting new challenges in designing the server.

- There is a shared state—the current factor—so we must decide how to store it and how it is accessed and modified.
- When one server thread changes the state in response to its client issuing the *N command, we must arrange to send a message to all the connected clients."

➢ Server cu stare partajata

Detalii de implementare:

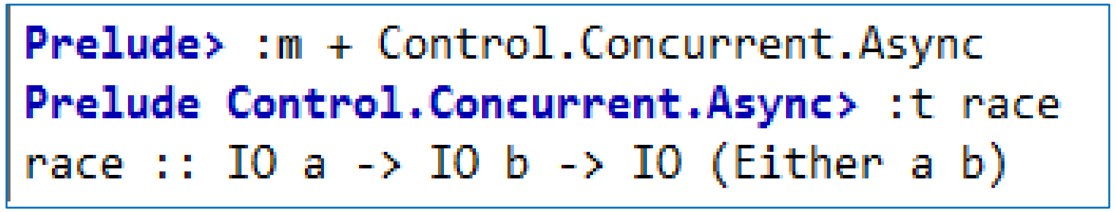
Pentru fiecare conexiune (client) se creaza un thread nou in care se executa functia talk.

Functia talk creaza un canal de comunicare si executa in paralel o functiile server si receive.

Functia receive citeste comenzile clientului si le introduce in canalul de comunicare, de unde sunt citite si prelucrate de functia server.

Functia server implementeaza actiunile serverului: citeste factorul initial, citeste si executa comenzile clientului; comanda *N a clientului poate modifica valoarea factorului .

Pentru executarea in paralel a functiilor server si receive se foloseste functia race.
Functia race executa doua actiuni in parallel si o intoarce pe prima care se termina



```
Prelude> :m + Control.Concurrent.Async
Prelude Control.Concurrent.Async> :t race
race :: IO a -> IO b -> IO (Either a b)
```

Memoria partajata este implementata in STM TVar, TChan

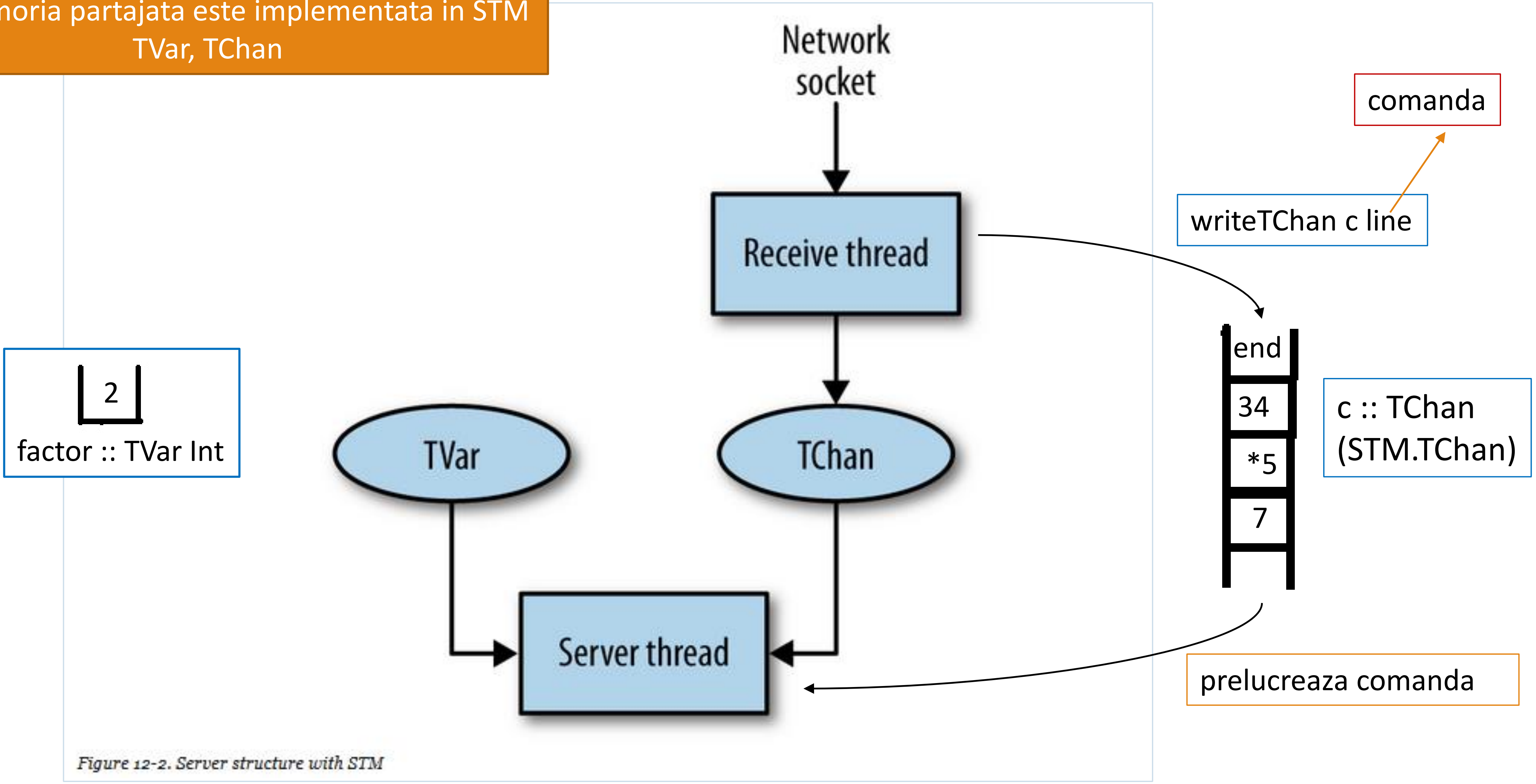


Figure 12-2. Server structure with STM

server2.hs

```
port :: Int
port = 44444

main = withSocketsDo $ do
  sock <- listenOn (PortNumber (fromIntegral port))
  printf "Listening on port %d\n" port
  factor <- atomically $ newTVar 2
  forever $ do
    (handle, host, port) <- accept sock
    printf "Accepted connection from %s: %s\n" host (show port)
     forkIO $ (talk handle factor) `finally` (hClose handle)
```

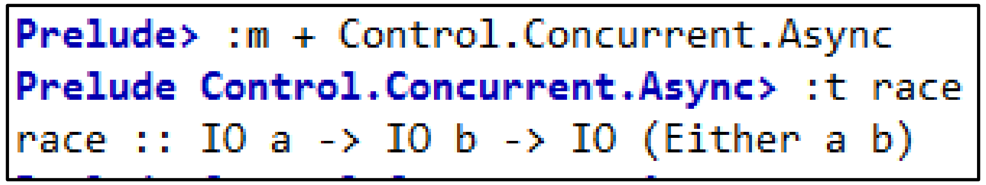
```
talk :: Handle -> TVar Integer -> IO ()
talk h factor = do
  hSetBuffering h LineBuffering
  c <- atomically newTChan
  race (server h factor c) (receive h c)
  return ()
```

server2.hs

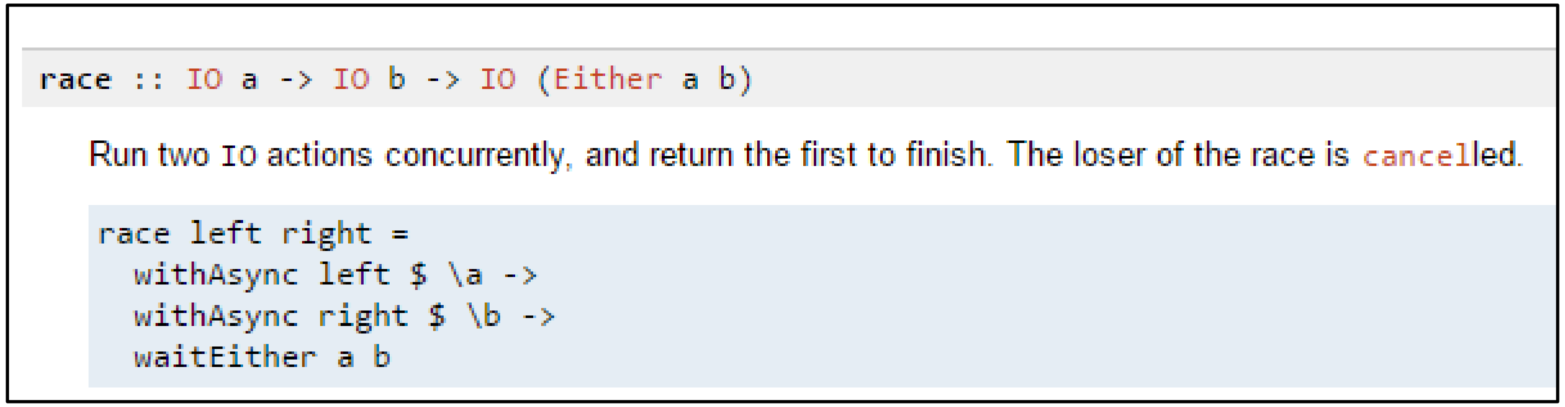
```haskell
talk :: Handle -> TVar Integer -> IO ()
talk h factor = do
  hSetBuffering h LineBuffering
  c <- atomically newTChan
  race (server h factor c) (receive h c)
  return ()
```

se termina odata cu primul dintre server si receive

in aceasta implementare, server se termina cand primeste comanda end, iar receive este o actiune definita cu forever



```
Prelude> :m + Control.Concurrent.Async
Prelude Control.Concurrent.Async> :t race
race :: IO a -> IO b -> IO (Either a b)
```



```
race :: IO a -> IO b -> IO (Either a b)
```

Run two `IO` actions concurrently, and return the first to finish. The loser of the race is `cancel`led.

```haskell
race left right =
  withAsync left $ \a ->
  withAsync right $ \b ->
  waitEither a b
```

```
server :: Handle -> TVar Integer -> TChan String -> IO ()
server h factor c = do
  f <- atomically $ readTVar factor
  hPrintf h "Current factor: %d\n" f
  loop f
 where
   loop f =  join $ atomically $ do
                                  f' <- readTVar factor
                                  if (f /= f')
                                  then return (newfactor f')
                                  else do
                                        cline <- readTChan c
                                        return (command f cline)

  newfactor f =    .....
  command f cline =   ......
```

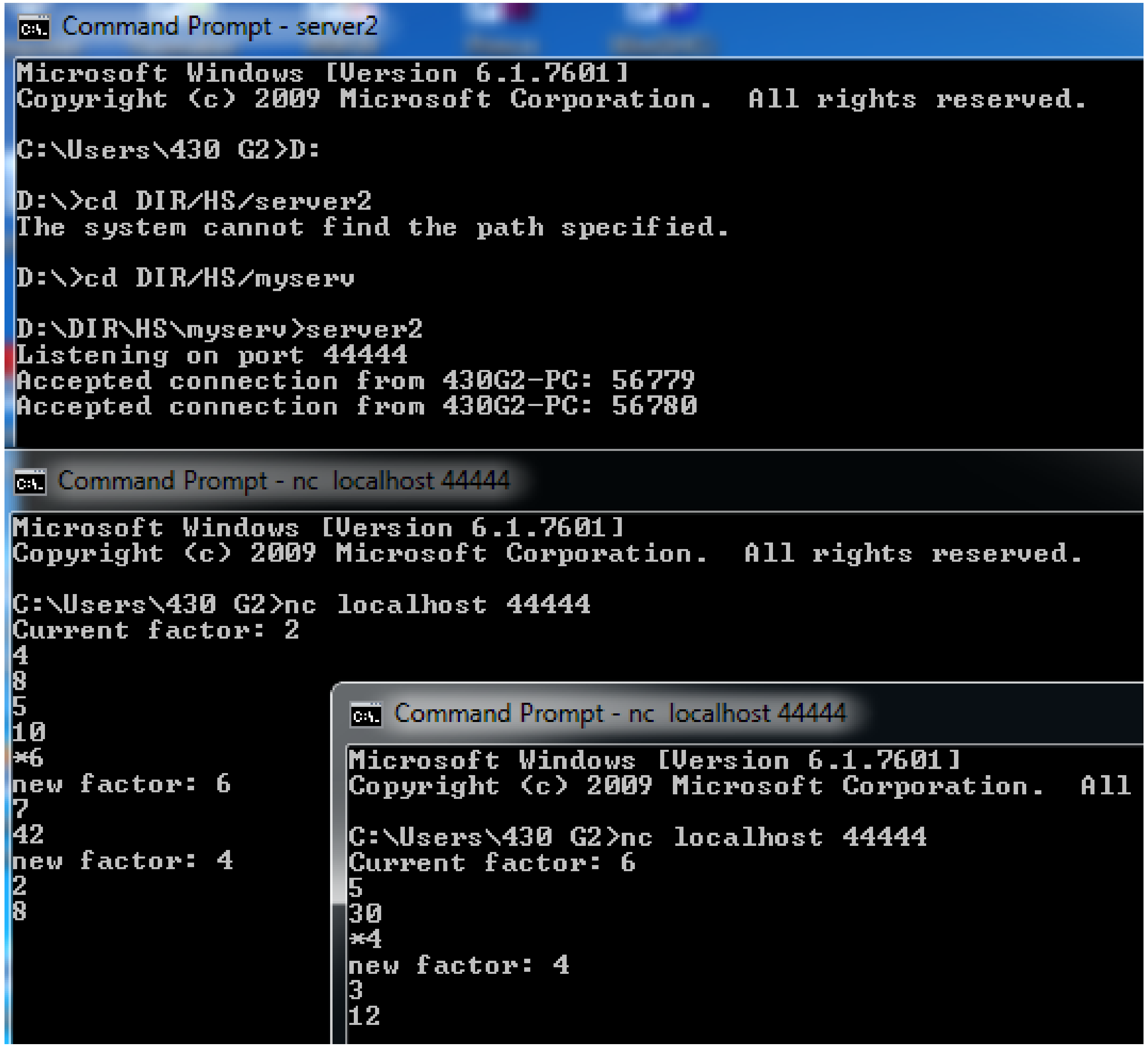
```
receive :: Handle -> TChan String -> IO ()
receive h c = forever $ do
       line <- hGetLine h
       atomically $ writeTChan c line
```

newfactor -anunta modificarea factorului tururor clientilor
command executa comanda
ambele apeleaza recursive loop f

server2.hs

```
server :: Handle -> TVar Integer -> TChan String -> IO ()
server h factor c = do
  f <- atomically $ readTVar factor
  hPrintf h "Current factor: %d\n" f
  loop f
 where
    loop f = join $ atomically $ do
                f' <- readTVar factor
                if (f /= f')
                then return (newfactor f')
                else do
                     cline <- readTChan c
                     return (command f cline)

newfactor f = do
    hPrintf  h "new factor: %d\n" f
    loop f
command f cline = .....
```

```
command f cline  = case cline of
      "end" ->
                hPutStrLn h ("Thank you for using the " ++
                  "Haskell doubling service.")
       '*':s  -> do
                atomically $ writeTVar factor (read s :: Integer)
                loop f
      line  ->  do
                hPutStrLn h (show (f * (read line :: Integer)))
                loop f
```



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\430 G2>D:

D:\>cd DIR/HS/server2
The system cannot find the path specified.

D:\>cd DIR/HS/myserv

D:\DIR\HS\myserv>server2
Listening on port 44444
Accepted connection from 430G2-PC: 56779
Accepted connection from 430G2-PC: 56780
```

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\430 G2>nc localhost 44444
Current factor: 2
4
8
5
10
*6
new factor: 6
7
42
new factor: 4
2
8
```

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All

C:\Users\430 G2>nc localhost 44444
Current factor: 6
5
30
*4
new factor: 4
3
12
```

➢ Server cu stare partajata si tip de data pentru clienti

Detalii de implementare:

client={nume, handle, canal}

Pentru fiecare conexiune se creaza un thread nou in care se executa functia talk.

Functia talk creaza un client nou, reprezentat printr-o structura {nume, handle, canal} si apeleaza functia runclient.

Functia runclient citeste factorul initial si executa in parallel functiile server si receive (folosind race).

Functia receive citeste comenzile clientului si le introduce in canalul de comunicare, de unde sunt citite si prelucrate de functia server.

Functia server implementeaza actiunile serverului: citeste factorul curent, citeste si executa comenzile clientului; comanda *N a clientului poate modifica valoarea factorului .

Pentru executarea in paralel a functiilor server si receive se foloseste functia race.
Functia race executa doua actiuni in parallel si o intoarce pe prima care se termina

➢ Server cu stare partajata si tip de date client

myserver3.hs

```
type ClientName = String

data Client = Client {clientName :: ClientName
                     ,clientHandle :: Handle
                     ,clientSendChan :: TChan String  }
```

```
newClient :: ClientName -> Handle -> STM Client
newClient name handle = do
  c <- newTChan
  return Client { clientName     = name
                , clientHandle = handle
                , clientSendChan = c
                }
```

```haskell
main = withSocketsDo $ do
  sock <- listenOn (PortNumber (fromIntegral port))
  printf "Listening on port %d\n" port
  factor <- atomically $ newTVar 2
  forever $ do
     (handle, host, port) <- accept sock
     printf "Accepted connection from %s: %s\n" host (show port)
     forkIO $ (talk handle factor)  `finally` (hClose handle)

port :: Int
port = 44444
```

```haskell
talk :: Handle -> TVar Integer -> IO ()
talk h factor = do
    hSetBuffering h LineBuffering
    hPutStrLn h "Name"
    name <- hGetLine h
    client <- atomically $ newClient name h
    hPutStrLn h ("Hello " ++ name)
    runClient factor client
runClient ::  TVar Integer -> Client -> IO()
runClient factor client@(Client clientName clientHandle clientSendChan) = do
                f <- atomically $ readTVar factor
                hPrintf clientHandle  "Current factor: %d\n" f
                race (server f factor client) (receive client)
                return ()



-- {-# LANGUAGE RecordWildCards #-}
--  runClient factor client@Client{..} = …
```

```
f (x:xs)  = x:x:xs
```

Folosind **as-pattern** se poate scrie

```
f s@x:xs = x:s
```

```
data C = C {a :: Int, b :: Int, c :: Int, d :: Int}
f (C {a = 1, b = b, c = c, d = d}) = b + c + d
```

Folosind **wild-card pattern** se scrie

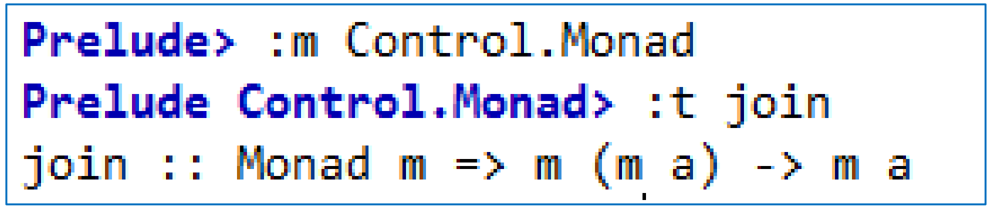
```
f (C{a=1, ..})=b+c+d
```

https://www.haskell.org/tutorial/patterns.html
https://downloads.haskell.org/~ghc/7.0.2/docs/html/users_guide/syntax-extns.html

```haskell
runClient ::  TVar Integer -> Client -> IO()
runClient factor client@Client{..} = do
            f <- atomically $ readTVar factor
            hPrintf clientHandle  "Current factor: %d\n" f
            race (server f factor client) (receive client)
            return ()
```

```haskell
server  :: Integer  -> TVar Integer -> Client -> IO()
server f  factor client@Client{..} =  join $ atomically $ do
            f' <- readTVar factor
            if (f /= f')
            then  return (newfactor f'  factor client)
            else do
                s <- readTChan clientSendChan
                return (command f factor client s)
```

```haskell
receive :: Client -> IO ()
receive client@Client{..}= forever $ do
   line <- hGetLine clientHandle
   atomically $ writeTChan clientSendChan line
```

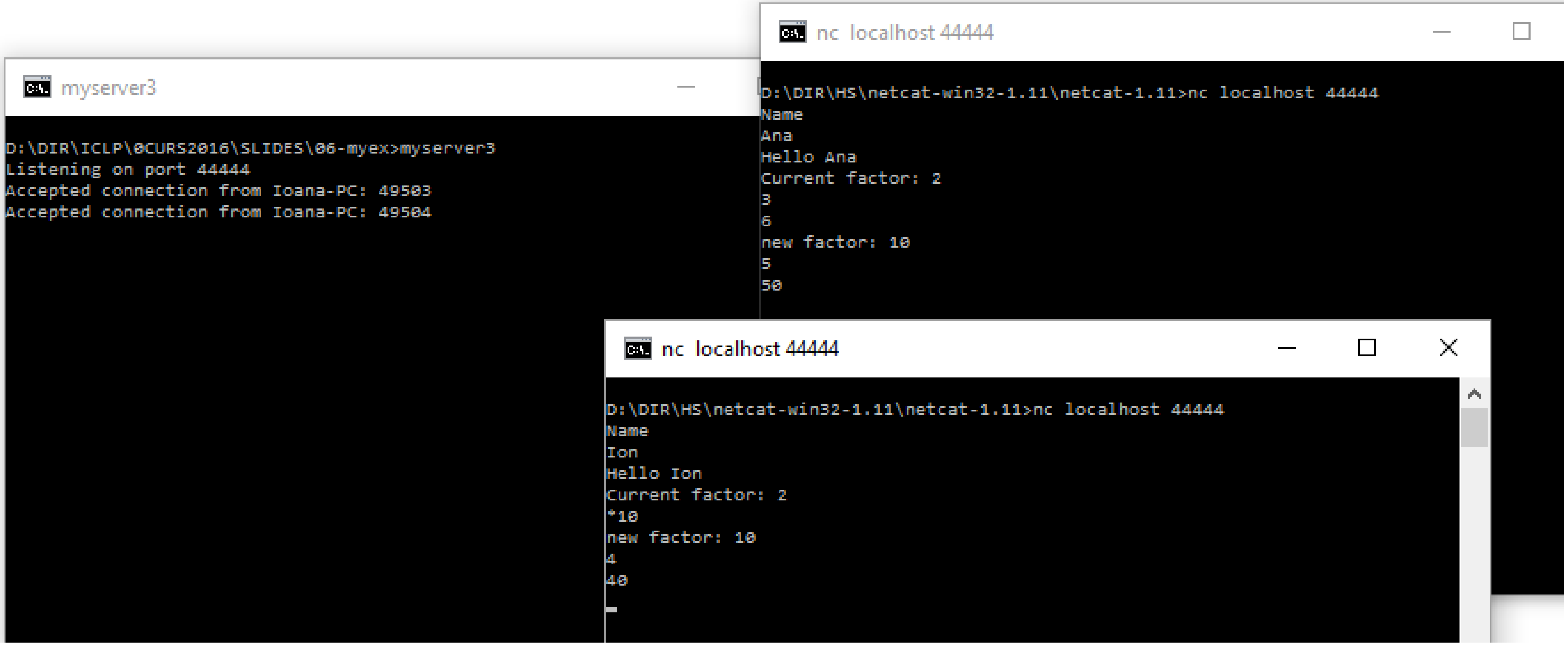


```
Prelude> :m Control.Monad
Prelude Control.Monad> :t join
join :: Monad m => m (m a) -> m a
```

```
server  :: Integer  -> TVar Integer -> Client -> IO()
server f  factor client@Client{..} =  join $ atomically $ do
            f' <- readTVar factor
            if (f /= f')
            then  return (newfactor f'  factor client)
            else do
                   cline <- readTChan clientSendChan
                   return (command f factor client cline)
```

```
newfactor :: Integer -> TVar Integer -> Client  -> IO()
newfactor f  factor client@Client{..} = do
            hPrintf clientHandle "new factor: %d\n" f
            server f  factor client
```

```
command  :: Integer ->  TVar Integer -> Client  -> String -> IO()
command f factor client@Client{..} cline = do
       case cline of
           "end" ->  hPutStrLn clientHandle ("Thank you for using the " ++ "Haskell doubling service.")
           '*':s -> do
                     atomically $ writeTVar factor (read s :: Integer)
                     server f factor client
           line  -> do
                     hPutStrLn clientHandle (show (f * (read line :: Integer)))
                     server f  factor client
```


myserver3

```
D:\DIR\ICLP\0CURS2016\SLIDES\06-myex>myserver3
Listening on port 44444
Accepted connection from Ioana-PC: 49503
Accepted connection from Ioana-PC: 49504
```

nc localhost 44444

```
D:\DIR\HS\netcat-win32-1.11\netcat-1.11>nc localhost 44444
Name
Ana
Hello Ana
Current factor: 2
3
6
new factor: 10
5
50
```

nc localhost 44444

```
D:\DIR\HS\netcat-win32-1.11\netcat-1.11>nc localhost 44444
Name
Ion
Hello Ion
Current factor: 2
*10
new factor: 10
4
40
```

- **Varianta: la crearea unui client nou sunt anuntati ceilalti clienti**
- **Tipul de date Server este definit prin lista clientilor**

myserver4.hs

```
data Server = Server { clients :: TVar (Map ClientName Client) }

newServer :: IO Server
newServer = do
        c <- newTVarIO Map.empty  -- newTVar in monada  IO
        return Server { clients = c }
```

Functia broadcast trimite mesajul tuturor clientilor, i.e.
mesajul este scris in canalul
de comunicare asociat fiecarui client

```
type Message = String

broadcast :: Server -> Message -> STM ()
broadcast Server{..} msg = do
      clientmap <- readTVar clients
      mapM_ (\client -> sendMessage client msg) (Map.elems clientmap)

sendMessage :: Client -> Message -> STM ()
sendMessage Client{..} msg =  writeTChan clientSendChan msg
```

La aparitia unui client nou se creaza comanda @nume care este transmisa tuturor celorlalti client prin functia broadcast

```
addClient :: Server -> ClientName -> Handle -> IO Client
addClient serv@Server{..} name handle = atomically $ do
      clientmap <- readTVar clients
      client <- newClient name handle
      writeTVar clients $ Map.insert name client clientmap
      broadcast serv   ("@"++ name) -- comanda din partea serverului
      return client
```

```
talk :: Handle -> TVar Integer -> Server -> IO ()
talk h factor serv  = do
   hSetBuffering h LineBuffering
   hPutStrLn h "Name"
   name <- hGetLine h
   client <- addClient serv name h
   runClient factor client
```

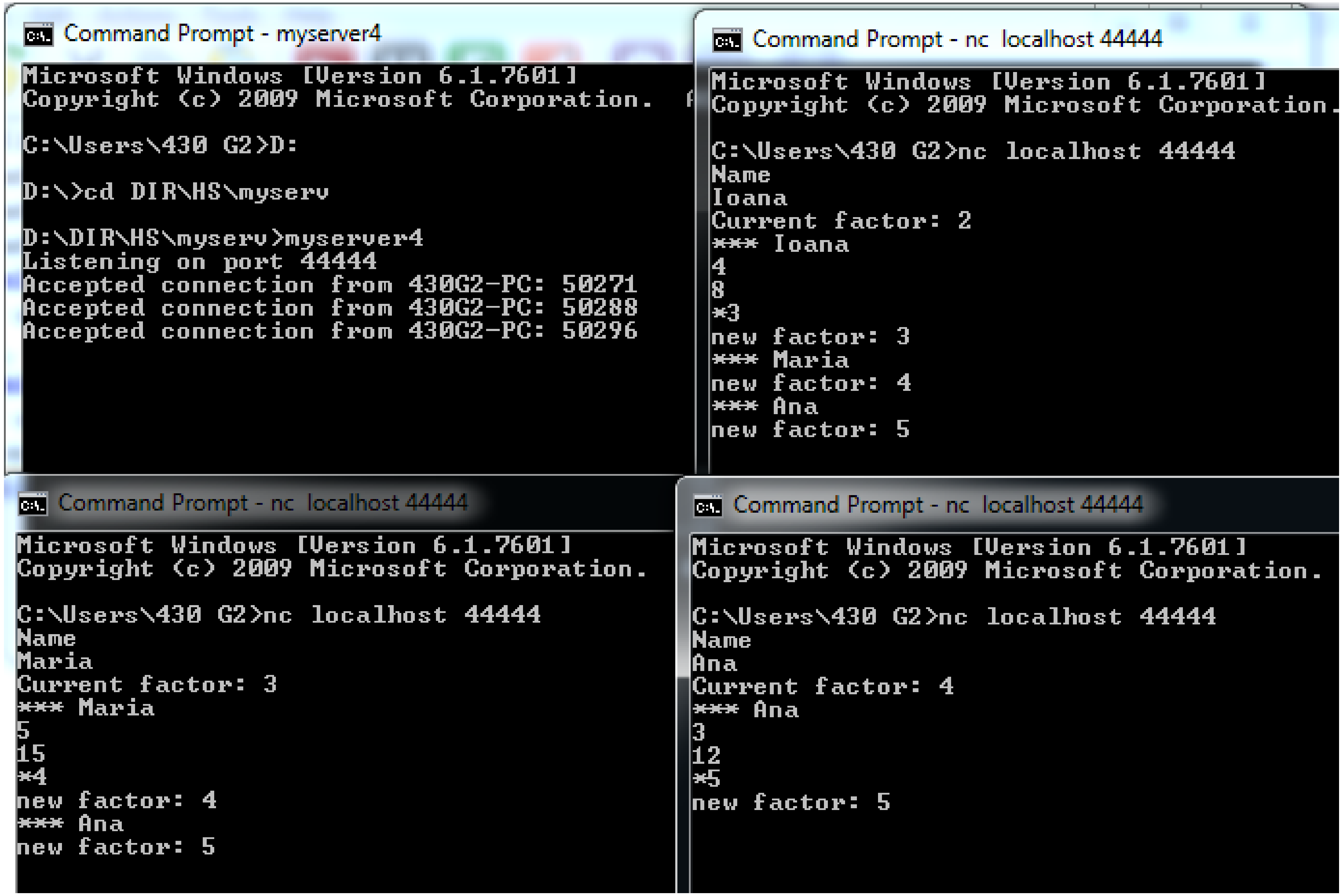
In aceasta variant nu se verifica daca exista deja un client cu acelasi nume; pentru o functionare corecta trebuie introduse nume diferite;
programul **chat.hs** face aceasta verificare

```
runClient ::  TVar Integer -> Client -> IO()
runClient factor client@Client{..} = do
               f <- atomically $ readTVar factor
               hPrintf clientHandle  "Current factor: %d\n" f
               race (server f factor client) (receive client)
               return ()
```

```
runClient ::  TVar Integer -> Client -> IO()
runClient factor client@(Client clientName clientHandle clientSendChan) = do
                                                     f <- atomically $ readTVar factor
                                                     hPrintf clientHandle  "Current factor: %d\n" f
                                                     race (server f factor client) (receive client)
                                                     return ()
```

```
command  :: Integer ->  TVar Integer -> Client  -> String -> IO()
command f factor client@Client{..} cline = do
    case cline of
     "end" ->  hPutStrLn clientHandle ("Thank you….")
     '*':s -> do
            atomically $ writeTVar factor (read s :: Integer)
             server f factor client
     '@':s -> do                                    -- comanda din partea serverului
              hPutStrLn clientHandle ("*** " ++ s)
              server f factor client
     line  -> do
              hPutStrLn clientHandle (show (f * (read line :: Integer)))
              server f  factor client
```

```
receive :: Client -> IO ()
server  :: Integer  -> TVar Integer -> Client -> IO()
newfactor :: Integer -> TVar Integer -> Client  -> IO()
```



Command Prompt - myserver4

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.

C:\Users\430 G2>D:

D:\>cd DIR\HS\myserv

D:\DIR\HS\myserv>myserver4
Listening on port 44444
Accepted connection from 430G2-PC: 50271
Accepted connection from 430G2-PC: 50288
Accepted connection from 430G2-PC: 50296
```

Command Prompt - nc localhost 44444

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.

C:\Users\430 G2>nc localhost 44444
Name
Ioana
Current factor: 2
*** Ioana
4
8
*3
new factor: 3
*** Maria
new factor: 4
*** Ana
new factor: 5
```

Command Prompt - nc localhost 44444

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.

C:\Users\430 G2>nc localhost 44444
Name
Maria
Current factor: 3
*** Maria
5
15
*4
new factor: 4
*** Ana
new factor: 5
```

Command Prompt - nc localhost 44444

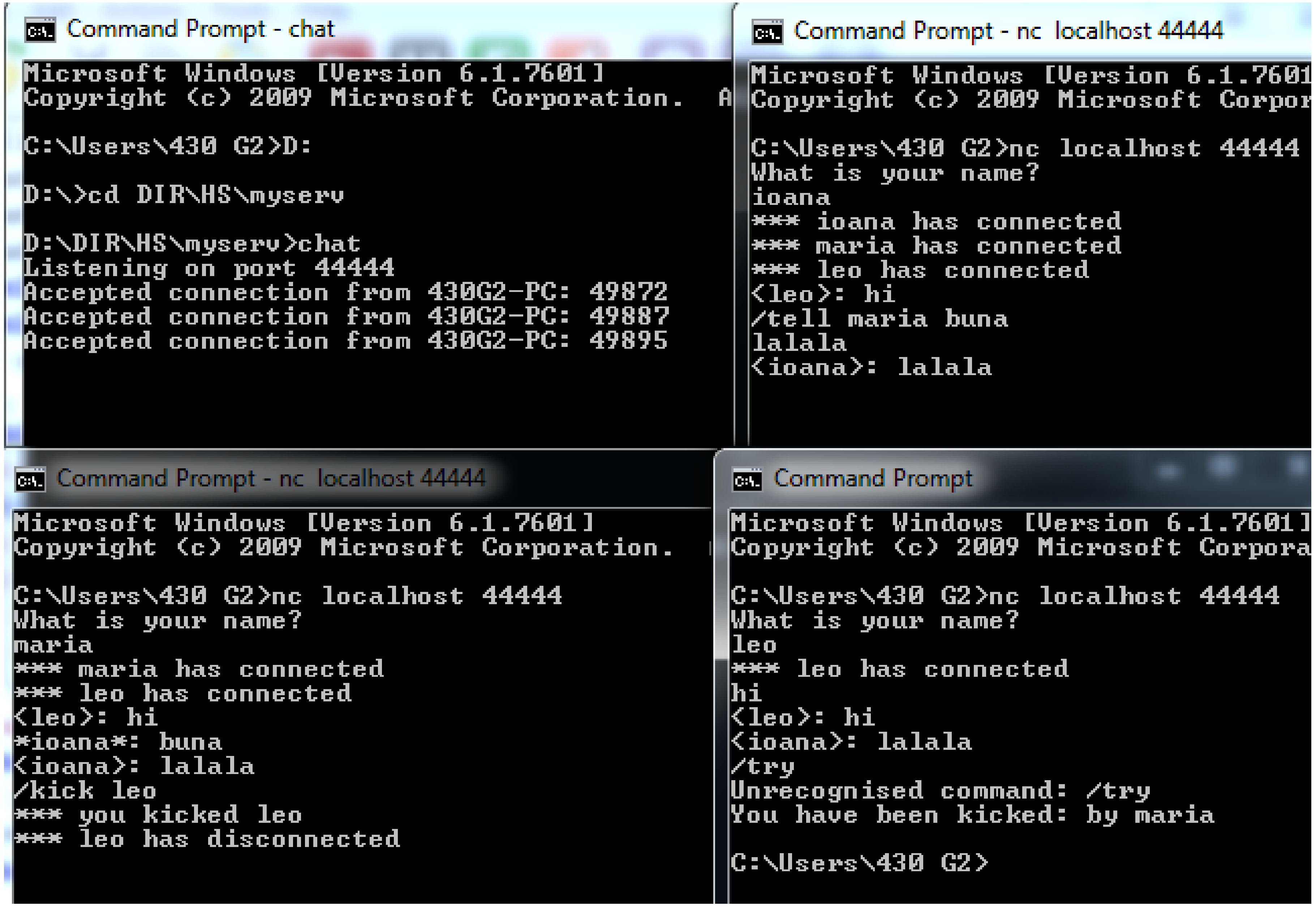
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.

C:\Users\430 G2>nc localhost 44444
Name
Ana
Current factor: 4
*** Ana
3
12
*5
new factor: 5
```

➢ Varianta mai complexa:

se adauga comenzi

/tell <name><mes>
/kick <name>
/quit

orice altceva este transmis tuturor



Command Prompt - chat

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  A

C:\Users\430 G2>D:

D:\>cd DIR\HS\myserv

D:\DIR\HS\myserv>chat
Listening on port 44444
Accepted connection from 430G2-PC: 49872
Accepted connection from 430G2-PC: 49887
Accepted connection from 430G2-PC: 49895
```

Command Prompt - nc localhost 44444

```
Microsoft Windows [Version 6.1.7601
Copyright (c) 2009 Microsoft Corpor

C:\Users\430 G2>nc localhost 44444
What is your name?
ioana
*** ioana has connected
*** maria has connected
*** leo has connected
<leo>: hi
/tell maria buna
lalala
<ioana>: lalala
```

Command Prompt - nc localhost 44444

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.

C:\Users\430 G2>nc localhost 44444
What is your name?
maria
*** maria has connected
*** leo has connected
<leo>: hi
*ioana*: buna
<ioana>: lalala
/kick leo
*** you kicked leo
*** leo has disconnected
```

Command Prompt

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corpora

C:\Users\430 G2>nc localhost 44444
What is your name?
leo
*** leo has connected
hi
<leo>: hi
<ioana>: lalala
/try
Unrecognised command: /try
You have been kicked: by maria

C:\Users\430 G2>
```