

Internetprogrammierung

<http://www.informatik.uni-freiburg.de/proglang/teaching/ss2003/internet>

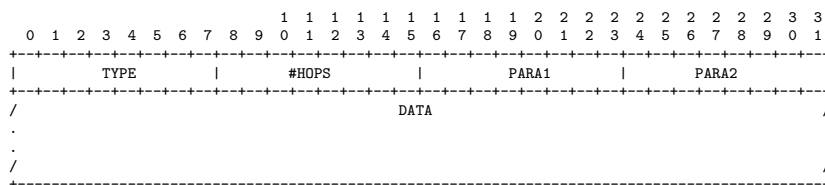
Übungsblatt 8

Abgabe: 7.7.2003

Dieses und das folgende Übungsblatt beschäftigen sich mit der Implementierung eines einfachen Peer-to-Peer-Netzwerkes zum Zwecke des Ressource-Sharings. Das Netzwerk soll den Austausch von Server-Listen, das Stellen von Suchanfragen und das Abfragen von gemeinsam zugänglichen Ressourcen unterstützen.

Zur Kommunikation zwischen den Servern dient ein einfaches Protokoll basierend auf UDP, das den Austausch von Paketen einer bestimmten Form regelt, um die oben genannten Funktionalitäten zu verwirklichen.

Generell besitzen Pakete unseres Peer-to-Peer-Netzwerkes das folgende Format:



Alle Pakete besitzen einen einheitlichen Header, beginnend mit einem Oktet, der den Pakettyp beschreibt, gefolgt von einem weiteren Oktet, das angibt, über wie viele andere Server das Paket weitergegeben wurde, und zwei weitere Oktets für zusätzliche Parameter.

Austausch von Severlisten

Jeder Server unseres Peer-to-Peer-Netzwerkes verwaltet eine Liste von IP-Adressen anderer ihm bekannter Server des Netzwerks.

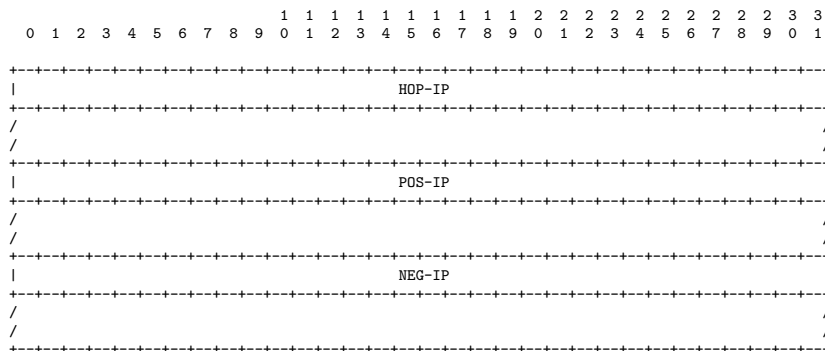
Zum Austausch von Server-Listen werden Pakete mit folgendem Header benutzt:

```

TYPE      0
#HOPS     Anzahl, wie oft das Paket weitergegeben wurde, beginnend bei 0, <= 255
PARA1     Anzahl der folgenden, positiven IP-Adressen (#POS), <= 255
PARA2     Anzahl der folgenden, negativen IP-Adressen (#NEG), <= 255

```

Das Datenfeld DATA sieht bei Server-Listen-Paketen wie folgt aus:



Es besteht aus aus einer Liste von #HOPS IP-Adressen, die angibt über welche anderen Server die Anfrage schon weitergeleitet wurde, gefolgt von einer Liste von #POS IP-Adressen, die für aktive Netzwerk-Server stehen, und einer Liste von #NEG IP-Adressen, die inaktive Netzwerk-Server repräsentieren. IP-Adressen werden dabei jeweils in der üblichen Netzwerk-Byte-Order angegeben.

Bekommt ein Server eine Anfrage mit einem Server-Listen-Paket, ist die Aufgabe des Servers, die eingehenden Informationen mit der bestehenden, aktuellen Liste abzugleichen, und ein Server-Listen-Paket mit aktuellen Informationen als Antwort zurückschicken.

Aufgabe 1 (Peer-to-Peer-Netzwerk/Server/Adressaustausch).

Implementieren Sie das Grundgerüst des Peer-to-Peer-Servers. Ihr Server sollte bisher nur Serverliste verwalten und den Austausch der Serverlisten über obiges Protokoll unterstützen.

Ihr Server sollte zur Kommunikation Port 23456 verwenden und beim Aufruf eine initiale Serverliste als Kommandozeilenargument akzeptieren.

Lösung.

Zuerst definieren wir ein Modul zur Repräsentierung von P2P-Paketten inklusive Ausgabe- und Einlesefunktionen.

```
module P2PPackage where

import Network.Socket (HostAddress, inet_ntoa, inet_addr)
import List (intersperse)
import Char (chr, ord)
import GHC.IOBase (unsafePerformIO)

data P2PPackage =
  P2PServerListPackage [HostAddress] [HostAddress] [HostAddress]

-- printer for P2P packages

instance Show P2PPackage where
  showsPrec _ (P2PServerListPackage hopIPs posIPs negIPs) =
    showString ['\NUL', chr (length hopIPs), chr (length posIPs), chr (length negIPs)]
      . showIPs hopIPs . showIPs posIPs . showIPs negIPs

showIPs [] = id
showIPs (ip:ips) =
  showString (hostAddressToNetworkByteOrderedString ip) . showIPs ips

-- parser for P2P packages

parsePackage :: String -> Maybe P2PPackage
parsePackage (typ : hops : para1 : para2 : dataStuff)
  | typ == '\NUL' =
    parseServerListPackage (ord hops) (ord para1) (ord para2) dataStuff
parsePackage _ = -- wrong package format
```

```

Nothing

parseServerListPackage :: Int -> Int -> Int -> String -> Maybe P2PPackage
parseServerListPackage nrHops nrPos nrNeg dataStuff = do
  let (hopIPStrs, dataStuff') = splitAt (nrHops * 4) dataStuff
      (posIPStrs, dataStuff'') = splitAt (nrPos * 4) dataStuff'
      (negIPStrs, rest)       = splitAt (nrNeg * 4) dataStuff''
  hopIPs <- parseIPs hopIPStrs
  posIPs <- parseIPs posIPStrs
  negIPs <- parseIPs negIPStrs
  return $ P2PServerListPackage hopIPs posIPs negIPs

parseIPs :: String -> Maybe [HostAddress]
parseIPs [] = return []
parseIPs (c1:c2:c3:c4:rest) = do
  ips <- parseIPs rest
  let ip = networkByteOrderedStringToHostAddress (c1, c2, c3, c4)
  return (ip:ips)
parseIPs _ = Nothing

hostAddressToNetworkByteOrderedString :: HostAddress -> String
hostAddressToNetworkByteOrderedString n =
  let str = unsafePerformIO $ inet_ntoa n
      is = tokenize (== '.') str
  in map (chr . read) is

networkByteOrderedStringToHostAddress :: (Char, Char, Char, Char) -> HostAddress
networkByteOrderedStringToHostAddress (c1, c2, c3, c4) =
  let il = [show (ord c1), show (ord c2), show (ord c3), show (ord c4)]
      str = concat $ intersperse "." il
  in unsafePerformIO (inet_addr str)

--

tokenize :: (a -> Bool) -> [a] -> [[a]]
tokenize p str =
  case break p str of
    ([], _) -> []
    (f , (_:rest)) -> (f : tokenize p rest)
    (f , _) -> [f]

```

Und hier ist der eigentliche P2P-Server:

```

module Main where

import P2PPackage
import Network.Socket

```

```

import List (nub, union, (\\))
import Char (ord)

portNo = 23456

type ServerState = [HostAddress]

main = do
  s_req <- socket AF_INET Datagram 0
  bindSocket s_req (SockAddrInet portNo INADDR_ANY)
  s_res <- socket AF_INET Datagram 0
  serverLoop [] s_req s_res

serverLoop :: ServerState -> Socket -> Socket -> IO ()
serverLoop state s_req s_res = do
  (str, n, sa) <- recvFrom s_req 4096
  -- debugOutputRequest str n sa
  let mPackage = parsePackage str
      state'' <- case mPackage of
        Nothing ->
          return state
        (Just package) -> do
          (state', package') <- handleRequest state package
          sendTo s_res (show package') sa
          return state'
  serverLoop state'' s_req s_res

handleRequest :: ServerState -> P2PPackage -> IO (ServerState, P2PPackage)
handleRequest state (P2PServerListPackage hopIPs posIPs negIPs) = do
  let state' = (state `union` (nub posIPs)) \\ (nub negIPs)
      resp = P2PServerListPackage [] state' []
  -- debugOutputState state
  -- debugOutputState state'
  return (state', resp)

-- debugging output

debugOutputRequest str n sa = do
  putStrLn ">>>"
  case sa of
    SockAddrInet cp ch -> do
      host <- inet_ntoa ch
      putStrLn ("Request from " ++ host ++ " port " ++ show cp)
    - ->
      putStrLn "Request from a local Unix-Socket"
  putStrLn $ "Length:" ++ show n
  putStrLn $ "Content:" ++ str

```

```

putStrLn $ "Content/Bytes:"
mapM_ (\ c -> putStr (show $ ord c) >> putStr "|") str
putStrLn ""
putStrLn "<<<<"

debugOutputState state = do
  putStrLn ">>>"
  putStrLn "State:"
  mapM_ (\ s -> inet_ntoa s >>= putStrLn) state
  putStrLn "<<<<"

```

Aufgabe 2 (Peer-to-Peer-Netzwerk/Adressaustausch-Client).

Implementieren Sie einen einfachen Client, mit dessen Hilfe die Serverliste eines Peer-to-Peer-Netzwerk-Servers manipuliert werden kann (Aufspielen bzw. Entfernen einer Serveradresse) und dessen aktueller Status am Bildschirm betrachtet werden kann.

Lösung.

```

module Main where

import P2PPackage
import Network.Socket
import System

portNo = 23456

main = do
  args <- getArgs
  action args

action :: [String] -> IO ()
action [hostname, "--show"] =
  actionServerListPackage hostname [] []
action [hostname, "--add", ip] =
  actionServerListPackage hostname [ip] []
action [hostname, "--del", ip] =
  actionServerListPackage hostname [] [ip]
action _ =
  showUsage

actionServerListPackage :: String -> [String] -> [String] -> IO ()
actionServerListPackage hostname posIPs negIPs = do
  socket <- openUDPSocket
  sendServerListPackage posIPs negIPs socket hostname
  mPackage <- receiveServerListPackage socket
  outputPackage mPackage

```

```

openUDPSocket :: IO Socket
openUDPSocket =
    socket AF_INET Datagram 0

sendServerListPackage :: [String] -> [String] -> Socket -> String -> IO ()
sendServerListPackage posIPs negIPs socket hostname = do
    -- construct server list package and ...
    posIPs' <- mapM inet_addr posIPs
    negIPs' <- mapM inet_addr negIPs
    let package = P2PServerListPackage [] posIPs' negIPs'
        -- ... send it to server
        server <- inet_addr hostname
        let serverSockAddr = SockAddrInet portNo server
            sendTo socket (show package) serverSockAddr
    return ()

receiveServerListPackage :: Socket -> IO (Maybe P2PPackage)
receiveServerListPackage socket = do
    (answer, _, _) <- recvFrom socket 4096
    return $ parsePackage answer

outputPackage :: Maybe P2PPackage -> IO ()
outputPackage Nothing =
    putStrLn "malformed package received"
outputPackage (Just (P2PServerListPackage hopIPs posIPs negIPs)) = do
    putStrLn "Server Response:"
    putStrLn "Hop IPs:"
    mapM_ outputIP hopIPs
    putStrLn "Pos IPs:"
    mapM_ outputIP posIPs
    putStrLn "Neg IPs:"
    mapM_ outputIP negIPs

outputIP ip = do
    str <- inet_ntoa ip
    putStrLn str

showUsage :: IO ()
showUsage =
    putStrLn "Usage: P2PClient hostname [--show | --add ip | --del ip]"

```

(optional) Aufgabe 3 (Peer-to-Peer-Netzwerk/rekursiver Adressaustausch).

Erweitern Sie Ihren Server dahingehend, dass dieser bei einer Anfrage nicht umgehend nur mit seiner aktuellen Liste antwortet, sondern zuerst andere Server um weitere Informationen über andere Server bittet, seine Server-Liste damit aktualisiert, und anschliessend mit

aktualisierten Informationen antwortet.

Benutzen Sie dazu das **#HOPS**-Feld im Paket-Header, um die Anfragetiefe mitzuverfolgen und zu begrenzen. Verwenden Sie die Adressen aus dem **HOP-IP-Log** zur Vermeidung von Schleifen.