Lucrarea nr. 2
Aplicaţii de tip client

Mihai IVANOVICI

6 martie 2006

Scopul acestei lucrări este de a vă familiariza cu modulul Python `socket` şi cu modul de implementare al unei aplicaţii de tip client.

1 `Socket`

`Socket`-ul este o extensie a sistemului I/O al sistemului de operare, care permite comunicaţia între procese ce rulează pe maşini diferite. Din punct de vedere istoric, conceptul de `socket` a apărut pe sistemul de operare BSD, un sistem de tip Unix.

Când suportul pentru reţea a fost adăugat la sistemele de operare, acesta a fost făcut menţinând uniformitatea cu operaţiile pe fişiere şi obiecte de tip fişier (e.g., obiectele de tip `pipe`). Pe sistemele de tip Unix, apeluri sistem cum ar fi `read()`, `write()`, `dup()`, `dup2()` şi `close()` vor funcţiona pentru socket-uri, la fel ca pentru orice descriptor de fişier.

Deşi socket-ul poate fi tratat la fel ca un fişier, există şi diferenţe: fişierele sunt create de apelul funcţiei `open()`, pe când socket-urile de apelul funcţiei `socket()`, şi alte apeluri sunt necesare pentru conectarea la sau activarea lor, cum ar fi `send()` şi `recv()`

2 Crearea unui socket

Pentru o aplicaţie de tip client, crearea unui socket presupune două faze: (i) crearea propriu-zisă a obiectului te tip socket, şi (ii) conectarea acestuia la un server.

La crearea obiectului trebuie specificate tipul comunicaţiei şi familia de protocoale. Tipul comunicaţiei poate fi: IPv4 (standardul Internet actual), IPv6 (standardul Internet viitor), IPX/SPX (NetWare) sau AFP (Apple file sharing). Pentru cele mai multe comunicaţii Internet, tipul este `AF_INET`, corespunzând standardului IPv4.

Familia de protocoale indică protocolul de transport folosit: `SOCK_STREAM` indică folosirea protocolului de transport TCP, iar `SOCK_DGRAM` protocolul

UDP.

Pentru crearea unui socket, în general, se foloseşte următorul apel al funcţiei `socket`:

```
s = scoket.socket( socket.AF_INET, socket.SOCK_STREAM )
```

Pentru a conecta un socket la un server, este nevoie de un dublet format din numele serverului sau adresa sa IP, şi numărul portului. Conectarea socketului se va face astfel:

```
s.connect( ("www.server.com", 80) )
```

Următorul exemplu crează un socket şi apoi va stabili o conexiune cu serverul www.google.com, pe portul 80.

```
#Basic Connection Example - connect.py

import socket

print "Creating socket..."
s = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
print "done."

print "Connecting to remote host..."
s.connect( ("www.google.com", 80) )
print "done."
```

2.1 Aflarea portului

Pentru aflarea numărului portului asociat unui protocol, se foloseşte funcţia `getsevbyname()` disponibilă în biblioteca `socket` din Python. Această funcţie acceptă două argumente: numele serviciului şi numele protocolului. Numele serviciului poate fi, de exemplu, 'http', iar numele protocolului este 'tcp' sau 'udp'.

```
#Basic Connection Example - connect2.py

import socket

print "Creating socket..."
s = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
print "done."

print "Looking up port number..."
port = socket.getservbyname( 'http', 'tcp' )
```

```
print "done."

print "Connecting to remote host on port %d..." % port
s.connect( ("www.google.com", port) )
print "done."
```

2.2 Aflarea de informaţii de la un socket

Rulaţi exemplul următor:

```
#Basic Connection Example - connect3.py

import socket

print "Creating socket..."
s = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
print "done."

print "Looking up port number..."
port = socket.getservbyname( 'http', 'tcp' )
print "done."

print "Connecting to remote host on port %d..." % port
s.connect( ("www.google.com", port) )
print "done."

print "Connected from", s.getsockname()
print "Connected to", s.getpeername()
```

Programul va afişa adresa IP a calculatorului pe care rulează şi numărul portului, adresa IP a serverului şi numărul portului la server. Pentru clienţi, numărul portului este alocat de sistemul de operare şi poate fi random. Prin urmare, puteşi observa diverse valori la rulări diferite ale aceluiaşi program.

3 Comunicaţia folosind socket

Odată ce conexiunea a fost stabilită, aceasta poate fi folosită pentru a trimite şi primit date. Pentru aceasta Python pune la dispoziţie două posibilităţi: prin socket-uri sau obiecte de tip fişier.

Socket-urile pun la dispoziţie interfeţe către apelurile sistem `send()`, `sendto()`, `recv()` şi `recvfrom()`. Obiectele de tip fişier oferă o interfaţă "tradiţională" prin apeluri de genul `read()`, `write()` sau `readline()`.

Obiectele socket sunt folositoare în cazul în care comunicaţia presupune un control mai bun asupra momentelor în care se trimit sau recepţionează

date, sau pentru protocoale în care datele sunt manipulate în cuante de o anumită lungime. Socket-urile sunt de asemenea preferate atunci când protocolul de transport folosit este UDP.

4 Tratarea erorilor

În principiu, orice funcţie poate genera o eroare: server-ul nu răspunde, conexiunea nu mai este valabilă etc. Comportamentul aplicaţiei depinde de modul în care aceste erori sunt tratate.

Citiţi şi înţelegeţi modul în care au fost tratate erorile în exemplul următor. Programul primeşte 3 argumente: numele unui server la care să se conecteze, un număr de port sau nume de serviciu şi numele fişierului care se doreşte a fi primit de la server.

```
#Error Handling Example - socketerrors.py

import socket, sys

host = sys.argv[ 1 ]
textport = sys.argv[ 2 ]
filename = sys.argv[ 3 ]

try:
   s = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
except socket.error, e:
   print "Strange error creating socket: %s" % e
   sys.exit( 1 )

#Try parsing is as a numeric port number
try:
   port = int( textport )
except ValueError:
   #That didn't work, so it's probably a protocol name
   #Look it up instead
   try:
      port = socket.getservbyname( textport, 'tcp' )
   except socket.error, e:
      print "Couldn't find your port: %s" % e
      sys.exit( 1 )

try:
   s.connect( (host, port) )
except socket.gaierror, e:
   print "Address-related error connecting to server: %s" % e
```

```
   sys.exit( 1 )
except socket.error, e:
   print "Connection error: %s" % e
   sys.exit( 1 )

try:
   s.sendall( "GET %s HTTP/1.0\r\n\r\n" % filename )
except socket.error, e:
   print "Error sending data: %s" % e
   sys.exit( 1 )

while 1:
   try:
      buf = s.recv( 2048 )
   except socket.error, e:
      print "Error receiving data: %s" % s
      sys.exit( 1 )

   if not len( buf ):
      break
   sys.stdout.write( buf )
```

În exemplul de mai sus, tratarea erorilor se face prin tipărirea unui mesaj şi terminarea programului. Modulul Python `socket` defineşte patru tipuri de erori:

- `socket.error` - pentru erori generale de I/O şi probleme privind comunicarea,
- `socket.gaierror` - pentru erori cu privire la adresa serverului,
- `socket.herror` - pentru alte erori de adresare, şi
- `socket.timeout` - pentru tratarea cazurilor în care a expirat un anumit timer setat cu ajutorul funcţiei `settimeout()`.

Rulaţi programul `socketerrors.py` şi observaţi comportamentul acestuia în următoarele situaţii:

```
$ python2.3 socketerrors.py www.google.com 80 index.html
$ python2.3 socketerrors.py www.nonexisting.server.com 80 index.html
$ python2.3 socketerrors.py www.google.com port80 index.html
$ python2.3 socketerrors.py www.google.com http index.html
$ python2.3 socketerrors.py www.yahoo.com http nonexisting.txt
```

Un caz care poate apărea şi care nu este tratat în exemplul anterior este atunci când conexiunea cu serverul se pierde după apelul `connect`, înainte de a se trimite sau primi date. Prin urmare, apelurile `recv()` nu vor returna date, iar programul se va termina în mod normal.

Este posibil ca un apel `sendall()` să nu fie recepţionat de server, iar clientul să nu sesizeze acest lucru. Pentru a preveni acest caz se foloseşte funcţia `shutdown()`, care forţează golirea bufferelor temporare şi va genera o excepţie în cazul în care a intervenit vreo eroare.

```
#Error Handling Example with Shutdown - shutdown.py

import socket, sys, time

host = sys.argv[ 1 ]
textport = sys.argv[ 2 ]
filename = sys.argv[ 3 ]

try:
   s = socket.socket( socket.AF_INET, socket.SOCK_STREAM )
except socket.error, e:
   print "Strange error creating socket: %s" % e
   sys.exit( 1 )

#Try parsing is as a numeric port number
try:
   port = int( textport )
except ValueError:
   #That didn't work, so it's probably a protocol name
   #Look it up instead
   try:
      port = socket.getservbyname( textport, 'tcp' )
   except socket.error, e:
      print "Couldn't find your port: %s" % e
      sys.exit( 1 )

try:
   s.connect( (host, port) )
except socket.gaierror, e:
   print "Address-related error connecting to server: %s" % e
   sys.exit( 1 )
except socket.error, e:
   print "Connection error: %s" % e
   sys.exit( 1 )
```

```
print "Sleeping..."
time.sleep( 10 )
print "Continuing"

try:
   s.sendall( "GET %s HTTP/1.0\r\n\r\n" % filename )
except socket.error, e:
   print "Error sending data: %s" % e
   sys.exit( 1 )

try:
   s.shutdown( 1 )
except socket.error, e:
   print "Error sending data (detected by shutdown): %s" % e
   sys.exit( 1 )

while 1:
   try:
      buf = s.recv( 2048 )
   except socket.error, e:
      print "Error receiving data: %s" % s
      sys.exit( 1 )

   if not len( buf ):
      break
   sys.stdout.write( buf )
```

Rulaţi într-un terminal serverul de la Lucrarea nr. 1 şi porniţi clientul `shutdown.py`. Opriţi serverul (Ctrl+C) atunci când clientul afişează "Sleeping...". Observaţi comportamentul clientului.

```
$ python2.3 shutdown.py localhost 55999 index.html
Sleeping...
Continuing
Error sending data (detected by shutdown): (107, 'Transport endpoint
      is not connected')
```