ΑΡΧΙΚΟ ΕΡΓΑΣΤΗΡΙΟ

Ιστορικό

Οι sockets έχουν μακρά ιστορία. Η χρήση τους ξεκίνησε με το ARPANET το 1971 και αργότερα έγινε API στο λειτουργικό σύστημα Berkeley Software Distribution (BSD) που κυκλοφόρησε το 1983 και ονομάζεται Berkeley sockets .

Όταν το Διαδίκτυο απογειώθηκε τη δεκαετία του 1990 με τον Παγκόσμιο Ιστό, το ίδιο έκανε και ο προγραμματισμός δικτύου. Οι διακομιστές Ιστού και τα προγράμματα περιήγησης δεν ήταν οι μόνες εφαρμογές που εκμεταλλεύονταν τα πρόσφατα συνδεδεμένα δίκτυα και χρησιμοποιούσαν sockets. Οι εφαρμογές πελάτη-διακομιστή όλων των τύπων και μεγεθών ήρθαν σε ευρεία χρήση.

Σήμερα, αν και τα υποκείμενα πρωτόκολλα που χρησιμοποιούνται από το socket API έχουν εξελιχθεί με τα χρόνια και έχουν αναπτυχθεί νέα, το API χαμηλού επιπέδου παρέμεινε το ίδιο.

Ο πιο συνηθισμένος τύπος εφαρμογών υποδοχής είναι οι εφαρμογές πελάτη-διακομιστή, όπου η μία πλευρά λειτουργεί ως διακομιστής και περιμένει για συνδέσεις από πελάτες. Αυτός είναι ο τύπος εφαρμογής που θα δημιουργήσετε σε αυτό το σεμινάριο. Πιο συγκεκριμένα, θα εστιάσετε στο Socket API για υποδοχές Internet , που μερικές φορές ονομάζονται υποδοχές Berkeley ή BSD. Υπάρχουν επίσης υποδοχές τομέα Unix , οι οποίες μπορούν να χρησιμοποιηθούν μόνο για την επικοινωνία μεταξύ διεργασιών στον ίδιο κεντρικό υπολογιστή.

Socket API Επισκόπηση

Η μονάδα υποδοχής της Python παρέχει μια διεπαφή στο API των υποδοχών Berkeley . Αυτή είναι η ενότητα που θα χρησιμοποιήσετε σε αυτό το σεμινάριο.

Οι κύριες λειτουργίες και μέθοδοι API υποδοχής σε αυτήν την ενότητα είναι:

- socket()
- .bind()
- .listen()
- .accept()
- .connect()
- .connect_ex()
- .send()
- .recv()
- .close()

Η Python παρέχει ένα βολικό και συνεπές API που αντιστοιχίζεται απευθείας στις κλήσεις συστήματος, στα αντίστοιχα C. Στην επόμενη ενότητα, θα μάθετε πώς χρησιμοποιούνται μαζί.

Ως μέρος της τυπικής βιβλιοθήκης της, η Python διαθέτει επίσης κλάσεις που διευκολύνουν τη χρήση αυτών των λειτουργιών υποδοχής χαμηλού επιπέδου. Αν και δεν καλύπτεται σε αυτό το σεμινάριο, μπορείτε να ελέγξετε τη λειτουργική μονάδα socketserver , ένα πλαίσιο για διακομιστές δικτύου. Υπάρχουν επίσης πολλές διαθέσιμες λειτουργικές μονάδες που εφαρμόζουν πρωτόκολλα Διαδικτύου υψηλότερου επιπέδου όπως HTTP και SMTP.

Υποδοχές TCP

Θα δημιουργήσετε ένα αντικείμενο υποδοχής χρησιμοποιώντας socket.socket(), καθορίζοντας τον τύπο υποδοχής ως socket.SOCK_STREAM. Όταν το κάνετε αυτό, το προεπιλεγμένο πρωτόκολλο που χρησιμοποιείται είναι το Πρωτόκολλο Ελέγχου Μετάδοσης (TCP) . Αυτή είναι μια καλή προεπιλογή και πιθανώς αυτό που θέλετε.

Γιατί να χρησιμοποιήσετε το TCP; Το Πρωτόκολλο Ελέγχου Μετάδοσης (TCP):

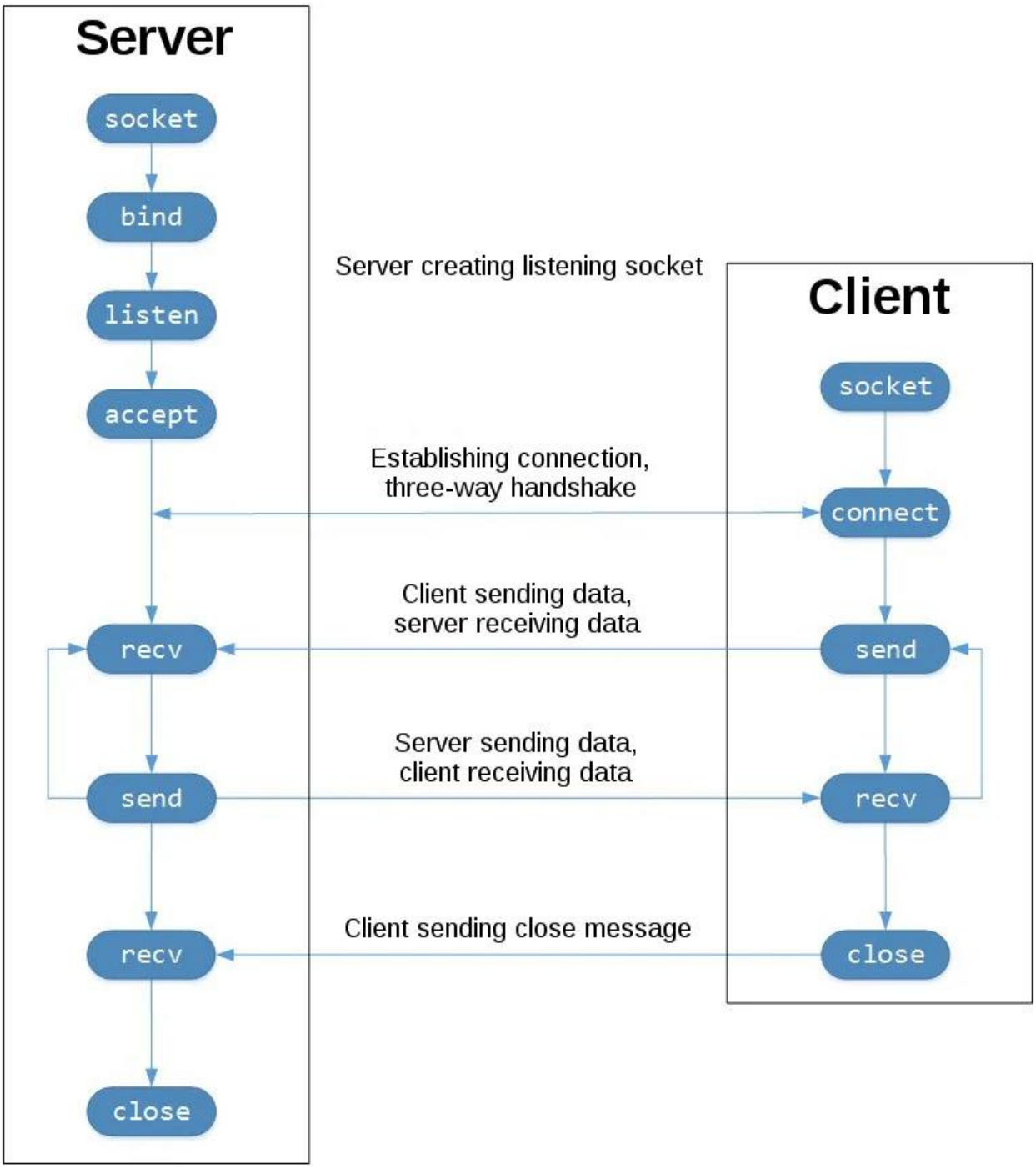
- **Είναι αξιόπιστο:** Τα πακέτα που πέφτουν στο δίκτυο εντοπίζονται και αναμεταδίδονται από τον αποστολέα.
- **Έχει παράδοση δεδομένων κατά παραγγελία:** Τα δεδομένα διαβάζονται από την εφαρμογή σας με τη σειρά που γράφτηκαν από τον αποστολέα.

Αντίθετα, οι υποδοχές User Datagram Protocol (UDP) που δημιουργούνται με socket.SOCK_DGRAM δεν είναι αξιόπιστες και τα δεδομένα που διαβάζονται από τον δέκτη μπορεί να είναι εκτός σειράς από τις εγγραφές του αποστολέα.

Γιατί είναι σημαντικό? Τα δίκτυα είναι ένα σύστημα παράδοσης με την καλύτερη δυνατή προσπάθεια. Δεν υπάρχει καμία εγγύηση ότι τα δεδομένα σας θα φτάσουν στον προορισμό τους ή ότι θα λάβετε ό,τι σας έχει σταλεί.

Οι συσκευές δικτύου, όπως οι δρομολογητές και οι μεταγωγείς, έχουν περιορισμένο εύρος ζώνης διαθέσιμο και έχουν τους δικούς τους εγγενείς περιορισμούς συστήματος. Έχουν CPU, μνήμη, διαύλους και buffer πακέτων διασύνδεσης, όπως και οι πελάτες και οι διακομιστές σας. Το TCP σάς απαλλάσσει από το να χρειάζεται να ανησυχείτε για απώλεια πακέτων , αφίξεις δεδομένων εκτός σειράς και άλλες παγίδες που συμβαίνουν πάντα όταν επικοινωνείτε σε ένα δίκτυο.

Για να το κατανοήσετε καλύτερα αυτό, ελέγξτε τη σειρά των κλήσεων API υποδοχής και τη ροή δεδομένων για το TCP:



Η αριστερή στήλη αντιπροσωπεύει τον διακομιστή. Στη δεξιά πλευρά είναι ο πελάτης.

Ξεκινώντας από την επάνω αριστερή στήλη, σημειώστε τις κλήσεις API που πραγματοποιεί ο διακομιστής για να δημιουργήσει μια υποδοχή "ακρόασης":

- socket()
- .bind()
- .listen()
- .accept()

Μια υποδοχή ακρόασης κάνει ακριβώς αυτό που υποδηλώνει το όνομά της. Ακούει για συνδέσεις από πελάτες. Όταν ένας πελάτης συνδέεται, ο διακομιστής καλεί .accept()για να αποδεχτεί ή να ολοκληρώσει τη σύνδεση.

Ο πελάτης καλεί .connect()για να δημιουργήσει μια σύνδεση με τον διακομιστή και να ξεκινήσει την τριπλή χειραψία. Το βήμα χειραψίας είναι σημαντικό γιατί διασφαλίζει ότι κάθε πλευρά της σύνδεσης είναι προσβάσιμη στο δίκτυο, με άλλα λόγια ότι ο πελάτης μπορεί να φτάσει στον διακομιστή και αντίστροφα. Μπορεί μόνο ένας κεντρικός υπολογιστής, πελάτης ή διακομιστής να μπορεί να φτάσει στον άλλο.

Στη μέση βρίσκεται η ενότητα μετ' επιστροφής, όπου ανταλλάσσονται δεδομένα μεταξύ του πελάτη και του διακομιστή χρησιμοποιώντας κλήσεις προς .send()και .recv().

Στο κάτω μέρος, ο πελάτης και ο διακομιστής κλείνουν τις αντίστοιχες υποδοχές τους.

Echo Client and Server

Τώρα που έχετε αποκτήσει μια επισκόπηση του API υποδοχής και του τρόπου επικοινωνίας του πελάτη και του διακομιστή, είστε έτοιμοι να δημιουργήσετε τον πρώτο πελάτη και διακομιστή σας. Θα ξεκινήσετε με μια απλή υλοποίηση. Ο διακομιστής απλώς θα επαναλάβει οτιδήποτε λαμβάνει πίσω στον πελάτη.

Διακομιστής Echo

Εδώ είναι ο διακομιστής:

Πύθων

```
# echo-server.py

import socket

HOST = "127.0.0.1"  # Standard loopback interface address (localhost)
PORT = 65432  # Port to listen on (non-privileged ports are > 1023)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print(f"Connected by {addr}")
        while True:
            data = conn.recv(1024)
            if not data:
                break
            conn.sendall(data)
```

Σημείωση: Μην ανησυχείτε για την κατανόηση όλων των παραπάνω τώρα. Συμβαίνουν πολλά σε αυτές τις λίγες γραμμές κώδικα. Αυτό είναι απλώς ένα σημείο εκκίνησης, ώστε να μπορείτε να δείτε έναν βασικό διακομιστή σε δράση.

Εντάξει, τι ακριβώς συμβαίνει στην κλήση API;

socket.socket()δημιουργεί ένα αντικείμενο υποδοχής που υποστηρίζει τον τύπο διαχείρισης περιβάλλοντος , ώστε να μπορείτε να το χρησιμοποιήσετε σε μια with δήλωση . Δεν χρειάζεται να καλέσετε s.close():

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    pass  # Use the socket object without calling s.close().
```

Τα ορίσματα που μεταβιβάζονται socket()είναι σταθερές που χρησιμοποιούνται για τον καθορισμό της οικογένειας διευθύνσεων και του τύπου υποδοχής. AF_INETείναι η οικογένεια διευθύνσεων Διαδικτύου για το IPv4 . SOCK_STREAMείναι ο τύπος υποδοχής για το TCP , το πρωτόκολλο που θα χρησιμοποιηθεί για τη μεταφορά μηνυμάτων στο δίκτυο.

Η .bind()μέθοδος χρησιμοποιείται για τη συσχέτιση της υποδοχής με μια συγκεκριμένη διεπαφή δικτύου και έναν αριθμό θύρας:

```
# echo-server.py

# ...

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    # ...
```

Οι τιμές που μεταβιβάζονται .bind()εξαρτώνται από την οικογένεια διευθύνσεων της υποδοχής. Σε αυτό το παράδειγμα, χρησιμοποιείτε socket.AF_INET(IPv4). Άρα αναμένει μια διπλή: (host, port).

hostμπορεί να είναι όνομα κεντρικού υπολογιστή, διεύθυνση IP ή κενή συμβολοσειρά. Εάν χρησιμοποιείται μια διεύθυνση IP, hostθα πρέπει να είναι μια συμβολοσειρά διεύθυνσης με μορφή IPv4. Η διεύθυνση IP 127.0.0.1είναι η τυπική διεύθυνση IPv4 για τη διεπαφή επαναφοράς , επομένως μόνο οι διεργασίες στον κεντρικό υπολογιστή θα μπορούν να συνδεθούν στον διακομιστή. Εάν περάσετε μια κενή συμβολοσειρά, ο διακομιστής θα αποδεχτεί συνδέσεις σε όλες τις διαθέσιμες διεπαφές IPv4.

Port αντιπροσωπεύει τον αριθμό θύρας TCP για την αποδοχή συνδέσεων από πελάτες. Θα πρέπει να είναι ακέραιος από 1έως 65535, όπως 0έχει δεσμευτεί. Ορισμένα συστήματα ενδέχεται να απαιτούν δικαιώματα υπερχρήστη εάν ο αριθμός θύρας είναι μικρότερος από 1024.

Ακολουθεί μια σημείωση σχετικά με τη χρήση ονομάτων κεντρικών υπολογιστών με .bind():

> "Εάν χρησιμοποιείτε όνομα κεντρικού υπολογιστή στο τμήμα κεντρικού υπολογιστή της διεύθυνσης υποδοχής IPv4/v6, το πρόγραμμα μπορεί να εμφανίσει μια μη ντετερμινιστική συμπεριφορά, καθώς η Python χρησιμοποιεί την πρώτη διεύθυνση που επιστρέφεται από την ανάλυση DNS. Η διεύθυνση υποδοχής θα επιλυθεί διαφορετικά σε μια πραγματική διεύθυνση IPv4/v6, ανάλογα με τα αποτελέσματα από την ανάλυση DNS ή/και τη διαμόρφωση του

> κεντρικού υπολογιστή. Για ντετερμινιστική συμπεριφορά χρησιμοποιήστε μια αριθμητική διεύθυνση στο τμήμα κεντρικού υπολογιστή."

Κατανοήστε ότι όταν χρησιμοποιείτε ένα όνομα κεντρικού υπολογιστή, θα μπορούσατε να δείτε διαφορετικά αποτελέσματα ανάλογα με το τι επιστρέφεται από τη διαδικασία επίλυσης ονόματος. Αυτά τα αποτελέσματα μπορεί να είναι οτιδήποτε. Την πρώτη φορά που θα εκτελέσετε την εφαρμογή σας, ενδέχεται να λάβετε τη διεύθυνση 10.1.2.3. Την επόμενη φορά, θα λάβετε διαφορετική διεύθυνση, 192.168.0.1. Την τρίτη φορά, θα μπορούσατε να πάρετε 172.16.7.8, και ούτω καθεξής.

Στο παράδειγμα διακομιστή, .listen()επιτρέπει σε έναν διακομιστή να δέχεται συνδέσεις. Κάνει τον διακομιστή μια υποδοχή "ακρόασης":

Πύθων

```
# echo-server.py

# ...

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    # ...
```

Η .listen()μέθοδος έχει μια backlog παράμετρο. Καθορίζει τον αριθμό των μη αποδεκτών συνδέσεων που θα επιτρέψει το σύστημα πριν αρνηθεί νέες συνδέσεις.

Εάν ο διακομιστής σας λαμβάνει πολλά αιτήματα σύνδεσης ταυτόχρονα, η αύξηση της backlog τιμής μπορεί να βοηθήσει ορίζοντας το μέγιστο μήκος της ουράς για τις εκκρεμείς συνδέσεις. Η μέγιστη τιμή εξαρτάται από το σύστημα.

Η .accept() μέθοδος αποκλείει την εκτέλεση και περιμένει για μια εισερχόμενη σύνδεση. Όταν ένας πελάτης συνδέεται, επιστρέφει ένα νέο αντικείμενο υποδοχής που αντιπροσωπεύει τη σύνδεση και μια πλειάδα που κρατά τη διεύθυνση του πελάτη. Η πλειάδα θα περιέχει (host, port)για συνδέσεις IPv4 ή (host, port, flowinfo, scopeid)για IPv6.

Ένα πράγμα που είναι επιτακτική ανάγκη να κατανοήσετε είναι ότι έχετε τώρα ένα νέο αντικείμενο υποδοχής από .accept(). Αυτό είναι σημαντικό γιατί είναι η υποδοχή που θα χρησιμοποιήσετε για να επικοινωνήσετε με τον πελάτη. Διαφέρει από την υποδοχή ακρόασης που χρησιμοποιεί ο διακομιστής για να δέχεται νέες συνδέσεις:

Πύθων

```
# echo-server.py

# ...

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print(f"Connected by {addr}")
        while True:
```

```
            data = conn.recv(1024)
            if not data:
                break
            conn.sendall(data)
```

Μετά .accept()την παροχή του αντικειμένου υποδοχής πελάτη , χρησιμοποιείται conn ένας άπειρος while βρόχος για την ολοκλήρωση του αποκλεισμού κλήσεων προς conn.recv(). Αυτό διαβάζει τα δεδομένα που στέλνει ο πελάτης και τα επαναλαμβάνει χρησιμοποιώντας conn.sendall().

Εάν conn.recv()επιστρέψει ένα κενό bytesαντικείμενο, b'', αυτό σηματοδοτεί ότι ο πελάτης έκλεισε τη σύνδεση και ο βρόχος τερματίστηκε. Η with δήλωση χρησιμοποιείται με conn για να κλείσει αυτόματα η υποδοχή στο τέλος του μπλοκ.

Echo Client

Ας δούμε τώρα τον πελάτη:

Πύθων

```
# echo-client.py

import socket

HOST = "127.0.0.1"  # The server's hostname or IP address
PORT = 65432  # The port used by the server

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b"Hello, world")
    data = s.recv(1024)

print(f"Received {data!r}")
```

Σε σύγκριση με τον διακομιστή, ο πελάτης είναι αρκετά απλός. Δημιουργεί ένα αντικείμενο υποδοχής, χρησιμοποιεί .connect()για να συνδεθεί με τον διακομιστή και καλεί s.sendall()για να στείλει το μήνυμά του. Τέλος, καλεί s.recv()να διαβάσει την απάντηση του διακομιστή και μετά την εκτυπώνει .

Εκτέλεση του προγράμματος-πελάτη και διακομιστή Echo

Σε αυτήν την ενότητα, θα εκτελέσετε τον πελάτη και τον διακομιστή για να δείτε πώς συμπεριφέρονται και να ελέγξετε τι συμβαίνει.

```
$ python echo-server.py
```

Το τερματικό σας θα φαίνεται να κολλάει. Αυτό συμβαίνει επειδή ο διακομιστής έχει αποκλειστεί ή έχει τεθεί σε αναστολή στις .accept():

Πύθων

```
# echo-server.py

# ...
```

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    conn, addr = s.accept()
    with conn:
        print(f"Connected by {addr}")
        while True:
            data = conn.recv(1024)
            if not data:
                break
            conn.sendall(data)
```

Περιμένει για σύνδεση πελάτη. Τώρα, ανοίξτε ένα άλλο παράθυρο τερματικού ή γραμμή εντολών και εκτελέστε τον πελάτη:

```
$ python echo-client.py
Received b'Hello, world'
```

Στο παράθυρο του διακομιστή, θα πρέπει να παρατηρήσετε κάτι σαν αυτό:

Κέλυφος

```
$ python_echo-server.py
Connected by ('127.0.0.1', 64623)
```

Στην παραπάνω έξοδο, ο διακομιστής εκτύπωσε την addrπλειάδα που επιστράφηκε από s.accept(). Αυτή είναι η διεύθυνση IP του πελάτη και ο αριθμός θύρας TCP. Ο αριθμός θύρας, 64623, πιθανότατα θα είναι διαφορετικός όταν την εκτελείτε στον υπολογιστή σας.

ΕΡΓΑΣΤΗΡΙΟ TRY-EXCEPT

Ας δημιουργήσουμε μερικά τμήματα κώδικα try-except και ας τοποθετήσουμε έναν πιθανό τύπο σφάλματος σε κάθε τμήμα. Για να λάβουμε μια είσοδο χρήστη, μπορούμε να χρησιμοποιήσουμε το πρόσθετο argparse. Αυτό το πρόσθετο είναι πιο ισχυρό από την απλή ανάλυση των ορισμάτων γραμμής εντολών χρησιμοποιώντας το sys.argv. Στα τμήματα try-except, τοποθετήστε τυπικές λειτουργίες socket, για παράδειγμα, δημιουργία ενός αντικειμένου socket, σύνδεση σε ένα διακομιστή, αποστολή δεδομένων και αναμονή για απάντηση. Το ακόλουθο παράδειγμα επιδεικνύει τα έννοια σε λίγες γραμμές κώδικα.

```
#!/usr/bin/env python
# Python Network Programming Cookbook -- Chapter – 1
# This program is optimized for Python 2.7. It may run on any
# other Python version with/without modifications.

import sys
import socket
import argparse

def main():
    # setup argument parsing
    parser = argparse.ArgumentParser(description='Socket Error Examples')
    parser.add_argument('--host', action="store", dest="host", required=False)
```

```
    parser.add_argument('--port', action="store", dest="port", type=int, required=False)
    parser.add_argument('--file', action="store", dest="file", required=False)
    given_args = parser.parse_args()
    host = given_args.host
    port = given_args.port
    filename = given_args.file

    # First try-except block -- create socket
    try:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    except socket.error, e:
        print "Error creating socket: %s" % e
        sys.exit(1)

    # Second try-except block -- connect to given host/port
    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)

    # Third try-except block -- sending data
    try:
        s.sendall("GET %s HTTP/1.0rr" % filename)
    except socket.error, e:
        print "Error sending data: %s" % e
        sys.exit(1)

    while 1:
        # Fourth tr-except block -- waiting to receive data from remote host
        try:
            buf = s.recv(2048)
        except socket.error, e:
            print "Error receiving data: %s" % e
            sys.exit(1)
        if not len(buf):
            break
        # write the received data
        sys.stdout.write(buf)

if __name__ == '__main__':
    main()
```

Πώς λειτουργεί

Στην Python, η παράδοση παραμέτρων γραμμής εντολών σε ένα σενάριο και η ανάλυσή τους μπορεί να γίνει χρησιμοποιώντας το πρόσθετο argparse. Αυτό είναι διαθέσιμο στην Python 2.7. Για προηγούμενες εκδόσεις της Python, αυτό το πρόσθετο είναι διαθέσιμο

ξεχωριστά στο Python Package Index (PyPI). Μπορείτε να το εγκαταστήσετε μέσω του easy_install ή του pip.

Σε αυτήν τη συνταγή, ορίζονται τρία ορίσματα: ένα όνομα υπολογιστή, ένας αριθμός θύρας και ένα όνομα αρχείου. Η χρήση αυτού του σεναρίου είναι ως εξής:

$ python 1_7_socket_errors.py --host= --port= --file=

Αν δοκιμάσετε με ένα μη υπάρχον όνομα υπολογιστή, αυτό το σενάριο θα εκτυπώσει ένα σφάλμα διεύθυνσης όπως παρακάτω:

$ python 1_7_socket_errors.py --host=www.pytgo.org --port=8080 --file=1_7_socket_errors.py

Σφάλμα σχετικά με τη διεύθυνση κατά τη σύνδεση στο διακομιστή: [Errno -5] Δεν υπάρχει διεύθυνση που συσχετίζεται με το όνομα υπολογιστή

Αν δεν υπάρχει υπηρεσία σε μια συγκεκριμένη θύρα και αν προσπαθήσετε να συνδεθείτε σε αυτήν τη θύρα, τότε αυτό θα πετάξει ένα σφάλμα χρονικού περιορισμού σύνδεσης ως εξής:

$ python 1_7_socket_errors.py --host=www.python.org --port=8080 --file=1_7_socket_errors.py

Αυτό θα επιστρέψει το ακόλουθο σφάλμα αφού ο υπολογιστής, www.python.org, δεν ακούει στη θύρα 8080:

Σφάλμα σύνδεσης: [Errno 110] Ο χρόνος σύνδεσης έληξε

Ωστόσο, αν στείλετε ένα αυθαίρετο αίτημα σε μια σωστή θύρα, το σφάλμα ενδέχεται να μην πιαστεί σε επίπεδο εφαρμογής. Για παράδειγμα, η εκτέλεση του ακόλουθου σεναρίου δεν επιστρέφει κανένα σφάλμα, αλλά η έξοδος HTML μας λέει τι δεν είναι σωστό με αυτό το σενάριο:

$ python 1_7_socket_errors.py --host=www.python.org --port=80 --file=1_7_socket_errors.py

HTTP/1.1 404 Δεν βρέθηκε

Διακομιστής: Varnish

Επανάληψη-Μετά: 0

Τύπος-περιεχομένου: text/html

Μήκος-περιεχομένου: 77

Αποδοχή-Εύρους: bytes

Ημερομηνία: Πέμ, 20 Φεβ 2014 12:14:01 GMT

Μέσω: 1.1 varnish

Ηλικία: 0

Σύνδεση: κλειστή

άγνωστος τομέας:

Σε αυτό το παράδειγμα, έχουν χρησιμοποιηθεί τέσσερα τμήματα try-except. Όλα τα τμήματα χρησιμοποιούν το socket.error εκτός από το δεύτερο, το οποίο χρησιμοποιεί το socket.gaierror. Αυτό χρησιμοποιείται για σφάλματα που σχετίζονται με τη διεύθυνση. Υπάρχουν και άλλα δύο είδη εξαιρέσεων: το socket.herror χρησιμοποιείται για το κληρονομημένο API C, και εάν χρησιμοποιήσετε τη μέθοδο settimeout() σε ένα socket, τότε το socket.timeout θα εκτοξευθεί όταν συμβεί ένας χρονικός περιορισμός σε αυτό το socket.

ΕΡΓΑΣΤΗΡΙΟ: ΚΩΔΙΚΑΣ ΠΟΛΛΑΠΛΩΝ ΣΥΝΔΕΣΕΩΝ

Τρέξτε τα παρακάτω προγράμματα και προσπαθήστε να τα εξηγήσετε.

Client:

```
import sys
import socket

def start_connections(host, port, num_conns):
    server_addr = (host, int(port))
    messages = [b"Message 1 from client.", b"Message 2 from client."]
    for i in range(int(num_conns)):
        connid = i + 1
        print(f"Starting connection {connid} to {server_addr}")
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            sock.connect(server_addr)
            for msg in messages:
                sock.sendall(msg)
                print(f"Sent message: {msg}")
        except Exception as e:
            print(f"Error connecting or sending message: {e}")
        finally:
            sock.close()

host = '127.0.0.1'
port = 65432
num_conns = 2

start_connections(host, port, num_conns)
```

Server:

```
import socket
import selectors
import types

sel = selectors.DefaultSelector()

def accept_wrapper(sock):
    conn, addr = sock.accept()
    print("Accepted connection from", addr)
    conn.setblocking(False)
    data = types.SimpleNamespace(addr=addr, inb=b"", outb=b"")
    events = selectors.EVENT_READ | selectors.EVENT_WRITE
    sel.register(conn, events, data=data)

def service_connection(key, mask):
    sock = key.fileobj
    data = key.data
    if mask & selectors.EVENT_READ:
        recv_data = sock.recv(1024)
        if recv_data:
            data.inb += recv_data
            print(f"Received message from {data.addr}: {recv_data.decode()}")
        else:
            print("Closing connection to", data.addr)
            sel.unregister(sock)
            sock.close()
    if mask & selectors.EVENT_WRITE:
        if data.outb:
            sent = sock.send(data.outb)
            data.outb = data.outb[sent:]

host = '127.0.0.1'
port = 65432
num_conns = 2

lsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
lsock.bind((host, port))
lsock.listen()
print("listening on", (host, port))
lsock.setblocking(False)
sel.register(lsock, selectors.EVENT_READ, data=None)

try:
    while True:
        events = sel.select(timeout=None)
        for key, mask in events:
            if key.data is None:
                accept_wrapper(key.fileobj)
            else:
```

```
                service_connection(key, mask)
except KeyboardInterrupt:
    print("caught keyboard interrupt, exiting")
finally:
    sel.close()
```