

Unix Domain Sockets

From the programmer's perspective there are two essential differences between using a Unix domain socket and an TCP/IP socket. First, the address of the socket is a path on the file system, rather than a tuple containing the server name and port. Second, the node created in the file system to represent the socket persists after the socket is closed, and needs to be removed each time the server starts up. The echo server example from earlier can be updated to use UDS by making a few changes in the setup section.

The socket needs to be created with address family `AF_UNIX`. Binding the socket and managing the incoming connections works the same as with TCP/IP sockets.

```
# socket_echo_server_uds.py

import socket
import sys
import os

server_address = './uds_socket'

# Make sure the socket does not already exist
try:
    os.unlink(server_address)
except OSError:
    if os.path.exists(server_address):
        raise

# Create a UDS socket
sock = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)

# Bind the socket to the address
print('starting up on {}'.format(server_address))
sock.bind(server_address)

# Listen for incoming connections
sock.listen(1)

while True:
    # Wait for a connection
    print('waiting for a connection')
    connection, client_address = sock.accept()
    try:
        print('connection from', client_address)

        # Receive the data in small chunks and retransmit it
        while True:
            data = connection.recv(16)
            print('received {!r}'.format(data))
            if data:
                print('sending data back to the client')
                connection.sendall(data)
            else:
                print('no data from', client_address)
                break

    finally:
        # Clean up the connection
        connection.close()
```

The client setup also needs to be modified to work with UDS. It should assume the file system node for the socket exists, since the server creates it by binding to the address. Sending and receiving data works the same way in the UDS client as the TCP/IP client from before.

```
# socket_echo_client_uds.py
```

```

# socket_echo_client_uds.py

import socket
import sys

# Create a UDS socket
sock = socket.socket(socket.AF_UNIX, socket.SOCK_STREAM)

# Connect the socket to the port where the server is listening
server_address = './uds_socket'
print('connecting to {}'.format(server_address))
try:
    sock.connect(server_address)
except socket.error as msg:
    print(msg)
    sys.exit(1)

try:

    # Send data
    message = b'This is the message. It will be repeated.'
    print('sending {!r}'.format(message))
    sock.sendall(message)

    amount_received = 0
    amount_expected = len(message)

    while amount_received < amount_expected:
        data = sock.recv(16)
        amount_received += len(data)
        print('received {!r}'.format(data))

finally:
    print('closing socket')
    sock.close()

```

The program output is mostly the same, with appropriate updates for the address information. The server shows the messages received and sent back to the client.

```

$ python3 socket_echo_server_uds.py
starting up on ./uds_socket
waiting for a connection
connection from
received b'This is the mess'
sending data back to the client
received b'age. It will be'
sending data back to the client
received b' repeated.'
sending data back to the client
received b''
no data from
waiting for a connection

```

The client sends the message all at once, and receives parts of it back incrementally.

```

$ python3 socket_echo_client_uds.py
connecting to ./uds_socket
sending b'This is the message. It will be repeated.'
received b'This is the mess'
received b'age. It will be'
received b' repeated.'
closing socket

```

Permissions

Since the UDS socket is represented by a node on the file system, standard file system permissions can be used to control access to the server.

```

$ ls -l ./uds_socket

```

```
srwxr-xr-x  1 dhellmann  dhellmann   0 Aug 21 11:19 uds_socket

$ sudo chown root ./uds_socket

$ ls -l ./uds_socket

srwxr-xr-x  1 root  dhellmann   0 Aug 21 11:19 uds_socket
```

Running the client as a user other than root now results in an error because the process does not have permission to open the socket.

```
$ python3 socket_echo_client_uds.py

connecting to ./uds_socket
[Errno 13] Permission denied
```

Communication Between Parent and Child Processes

The `socketpair()` function is useful for setting up UDS sockets for inter-process communication under Unix. It creates a pair of connected sockets that can be used to communicate between a parent process and a child process after the child is forked.

```
# socket_socketpair.py

import socket
import os

parent, child = socket.socketpair()

pid = os.fork()

if pid:
    print('in parent, sending message')
    child.close()
    parent.sendall(b'ping')
    response = parent.recv(1024)
    print('response from child:', response)
    parent.close()

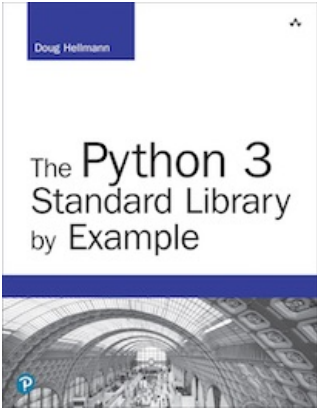
else:
    print('in child, waiting for message')
    parent.close()
    message = child.recv(1024)
    print('message from parent:', message)
    child.sendall(b'pong')
    child.close()
```

By default, a UDS socket is created, but the caller can also pass address family, socket type, and even protocol options to control how the sockets are created.

```
$ python3 -u socket_socketpair.py

in parent, sending message
in child, waiting for message
message from parent: b'ping'
response from child: b'pong'
```

This page was last updated 2016-12-18.





[Get the book](#)

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

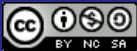
Looking for [examples for Python 2?](#)

This Site

-  Module Index
-  Index



© Copyright 2019, Doug Hellmann



Other Writing

-  Blog
-  The Python Standard Library By Example