Select() and Poll()

Select:

- Python's `select()` function is a direct interface to the underlying operating system implementation. It monitors sockets, open files, and pipes (anything with a `fileno()` method that returns a valid file descriptor) until they become readable or writable, or a communication error occurs.
- `select()` makes it easier to monitor multiple connections at the same time, and is more efficient than writing a polling loop in Python using socket timeouts, because the monitoring happens in the operating system network layer, instead of the interpreter.
- The arguments to `select()` are three lists containing communication channels to monitor. The first is a list of the objects to be checked for incoming data to be read, the second contains objects that will receive outgoing data when there is room in their buffer, and the third those that may have an error (usually a combination of the input and output channel objects). The next step in the server is to set up the lists containing input sources and output destinations to be passed to `select()`.
- `select()` returns three new lists, containing subsets of the contents of the lists passed in. All of the sockets in the `readable` list have incoming data buffered and available to be read. All of the sockets in the `writable` list have free space in their buffer and can be written to. The sockets returned in `exceptional` have had an error (the actual definition of "exceptional condition" depends on the platform).
- The "readable" sockets represent three possible cases. If the socket is the main "server" socket, the one being used to listen for connections, then the "readable" condition means it is ready to accept another incoming connection. In addition to adding the new connection to the list of inputs to monitor, this section sets the client socket to not block.
- There are fewer cases for the writable connections. If there is data in the queue for a connection, the next message is sent. Otherwise, the connection is removed from the list of output connections so that the next time through the loop select() does not indicate that the socket is ready to send data.
- Finally, if there is an error with a socket, it is closed
-

Select() and Poll()

Program :

server.py

```
import select
import socket
import sys
import queue
# Create a TCP/IP socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#non blocking socket
server.setblocking(0)
# Bind the socket to the port
server_address = ('localhost', 10001)
server.bind(server_address)
# Listen for incoming connections
server.listen(5)
# Sockets from which we expect to read
inputs = [ server ]
# Sockets to which we expect to write
outputs = [ ]
# Outgoing message queues (socket:Queue)
message_queues = {}
while inputs:
    # Wait for at least one of the sockets to be ready for processing
    print(  '\nwaiting for the next event')
    readable, writable, exceptional = select.select(inputs, outputs, inputs)
    # Handle inputs
    for s in readable:
        if s is server:
            # A "readable" server socket is ready to accept a connection
            connection, client_address = s.accept()
            print(  'new connection from', client_address)
            connection.setblocking(0)
            inputs.append(connection)
        else:
            data = s.recv(1024)
            if data.decode():
                # A readable client socket has data
                print(  'received "%s" from %s' % (data.decode(), s.getpeername()))
                message_queues[s]=data.decode()
                # Add output channel for response
                if s not in outputs:
                    outputs.append(s)
            else:
                # Interpret empty result as closed connection
                print(  'closing', client_address, 'after reading no data')
                # Stop listening for input on the connection
                if s in outputs:
                    outputs.remove(s)
                inputs.remove(s)
                s.close()
                # Remove message queue
                del message_queues[s]
# Handle outputs
    for s in writable:
        try:
            next_msg = message_queues[s]
        except queue.Empty:
            # No messages waiting so stop checking for writability.
```

Select() and Poll()

```
            print(  'output queue for', s.getpeername(), 'is empty')
            outputs.remove(s)
        else:
            print(  'sending "%s" to %s' % (next_msg, s.getpeername()))
            s.send(next_msg.encode())
# Handle "exceptional conditions"
    for s in exceptional:
        print(  'handling exceptional condition for', s.getpeername())
        # Stop listening for input on the connection
        inputs.remove(s)
        if s in outputs:
            outputs.remove(s)
        s.close()
        # Remove message queue
        del message_queues[s]
```

client.py

```
import sys
import socket

messages = [ 'This is the message. ',
               'It will be sent ',
               'in parts.',
               ]
server_address = ('localhost', 10001)
# Create a TCP/IP socket
socks = [ socket.socket(socket.AF_INET, socket.SOCK_STREAM),
socket.socket(socket.AF_INET, socket.SOCK_STREAM),
socket.socket(socket.AF_INET, socket.SOCK_STREAM),
socket.socket(socket.AF_INET, socket.SOCK_STREAM)
           ]
# Connect the socket to the port where the server is listening
print(  'connecting to %s port %s' % server_address)
for s in socks:
    s.connect(server_address)
for message in messages:
    # Send messages on both sockets
    for s in socks:
        print(  '%s: sending "%s"' % (s.getsockname(), message))
        s.send(message.encode())
    # Read responses on both sockets
    for s in socks:
        data = s.recv(1024)
        print(  '%s: received "%s"' % (s.getsockname(), data.decode()))
        if not data.decode():
            print(  'closing socket', s.getsockname())
            s.close()
```

Select() and Poll()

output :

server

```
(venv) praveen@praveen:/media/praveen/praveen/programs/python/select_and_poll$ python3 echo_server.py

waiting for the next event
new connection from ('127.0.0.1', 56950)

waiting for the next event
new connection from ('127.0.0.1', 56952)
received "This is the message. " from ('127.0.0.1', 56950)

waiting for the next event
new connection from ('127.0.0.1', 56954)
received "This is the message. " from ('127.0.0.1', 56952)
sending "This is the message. " to ('127.0.0.1', 56950)

waiting for the next event
new connection from ('127.0.0.1', 56956)
received "This is the message. " from ('127.0.0.1', 56954)
sending "This is the message. " to ('127.0.0.1', 56950)
sending "This is the message. " to ('127.0.0.1', 56952)

waiting for the next event
received "This is the message. " from ('127.0.0.1', 56956)
sending "This is the message. " to ('127.0.0.1', 56950)
sending "This is the message. " to ('127.0.0.1', 56952)
sending "This is the message. " to ('127.0.0.1', 56954)

waiting for the next event
sending "This is the message. " to ('127.0.0.1', 56950)
sending "This is the message. " to ('127.0.0.1', 56952)
sending "This is the message. " to ('127.0.0.1', 56954)
sending "This is the message. " to ('127.0.0.1', 56956)

waiting for the next event
sending "This is the message. " to ('127.0.0.1', 56950)
```

client

```
(venv) praveen@praveen:/media/praveen/praveen/programs/python/select_and_poll$ python3 echoclient.py
connecting to localhost port 10001
('127.0.0.1', 56950): sending "This is the message. "
('127.0.0.1', 56952): sending "This is the message. "
('127.0.0.1', 56954): sending "This is the message. "
('127.0.0.1', 56956): sending "This is the message. "
('127.0.0.1', 56950): received "This is the message. "
('127.0.0.1', 56952): received "This is the message. "
('127.0.0.1', 56954): received "This is the message. "
('127.0.0.1', 56956): received "This is the message. "
('127.0.0.1', 56950): sending "It will be sent "
('127.0.0.1', 56952): sending "It will be sent "
('127.0.0.1', 56954): sending "It will be sent "
('127.0.0.1', 56956): sending "It will be sent "
('127.0.0.1', 56950): received "This is the message. This is the message. This is the message. This is the message. "
('127.0.0.1', 56952): received "This is the message. This is the message. This is the message. "
('127.0.0.1', 56954): received "This is the message. This is the message. "
('127.0.0.1', 56956): received "This is the message. "
('127.0.0.1', 56950): sending "in parts."
('127.0.0.1', 56952): sending "in parts."
('127.0.0.1', 56954): sending "in parts."
('127.0.0.1', 56956): sending "in parts."
('127.0.0.1', 56950): received "It will be sent It will be sent It will be sent It will be sent It will be sent It will be sent It will be sent It will be sent It will be sent It w
ill be sent It will be sent It will be sent It will be sent It will be sent It will be sent It will be sent It will be sent "
('127.0.0.1', 56952): received "It will be sent in parts."
('127.0.0.1', 56954): received "It will be sent in parts."
('127.0.0.1', 56956): received "It will be sent It will be sent It will be sent It will be sent It will be sent It will be sent It will be sent It will be sent It will be sent It w
ill be sent It will be sent It will be sent It will be sent It will be sent It will be sent It will be sent It will be sent "
(venv) praveen@praveen:/media/praveen/praveen/programs/python/select_and_poll$
```

Select() and Poll()

Poll:

- The `poll()` function provides similar features to `select()`, but the underlying implementation is more efficient. The trade-off is that `poll()` is not supported under Windows, so programs using `poll()` are less portable.
- `poll()` scales better because the system call only requires listing the file descriptors of interest, while `select()` builds a bitmap, turns on bits for the fds of interest, and then afterward the whole bitmap has to be linearly scanned again. `select()` is O(highest file descriptor), while `poll()` is O(number of file descriptors).
- Python implements `poll()` with a class that manages the registered data channels being monitored. Channels are added by calling `register()` with flags indicating which events are interesting for that channel. The full set of flags is:

Event	Description
POLLIN	Input ready
POLLPRI	Priority input ready
POLLOUT	Able to receive output
POLLERR	Error
POLLHUP	Channel closed
POLLNVAL	Channel not open

Program :

server.py

```
import select
import socket
import sys
import queue
# Create a TCP/IP socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setblocking(0)
# Bind the socket to the port
server_address = ('localhost', 10001)
print('starting up on %s port %s' % server_address)
server.bind(server_address)
# Listen for incoming connections
```

Select() and Poll()

```
server.listen(5)
# Keep up with the queues of outgoing messages
message_queues = {}
# Do not block forever (milliseconds)
TIMEOUT = 1000
# Commonly used flag setes
READ_ONLY = select.POLLIN | select.POLLPRI | select.POLLHUP | select.POLLERR
READ_WRITE = READ_ONLY | select.POLLOUT
# Set up the poller
poller = select.poll()
poller.register(server, READ_ONLY)
# Map file descriptors to socket objects
fd_to_socket = { server.fileno(): server,
                 }
while True:
    # Wait for at least one of the sockets to be ready for processing
    print('\nwaiting for the next event')
    events = poller.poll(TIMEOUT)
    for fd, flag in events:
        # Retrieve the actual socket from its file descriptor
        s = fd_to_socket[fd]
        # Handle inputs
        if flag & (select.POLLIN | select.POLLPRI):
            if s is server:
                # A "readable" server socket is ready to accept a connection
                connection, client_address = s.accept()
                print('new connection from', client_address)
                connection.setblocking(0)
                fd_to_socket[connection.fileno()] = connection
                poller.register(connection, READ_ONLY)
                # Give the connection a queue for data we want to send
                message_queues[connection] = queue.Queue()
            #s is client
            else:
                data = s.recv(1024)
                if data.decode():
                    # A readable client socket has data
                    print('received "%s" from %s' % (data.decode(), s.getpeername()))
                    message_queues[s]=data.decode()
                    # Add output channel for response
                    poller.modify(s, READ_WRITE)
                else:
                    # Interpret empty result as closed connection
                    print('closing', client_address, 'after reading no data')
                    # Stop listening for input on the connection
                    poller.unregister(s)
                    s.close()
                    # Remove message queue
                    del message_queues[s]
        elif flag & select.POLLOUT:
            # Socket is ready to send data, if there is any to send.
            try:
                next_msg = message_queues[s]
            except queue.Empty:
                # No messages waiting so stop checking for writability.
                print('output queue for', s.getpeername(), 'is empty')
                poller.modify(s, READ_ONLY)
            else:
                print('sending "%s" to %s' % (next_msg, s.getpeername()))
                s.send(next_msg.encode())
```

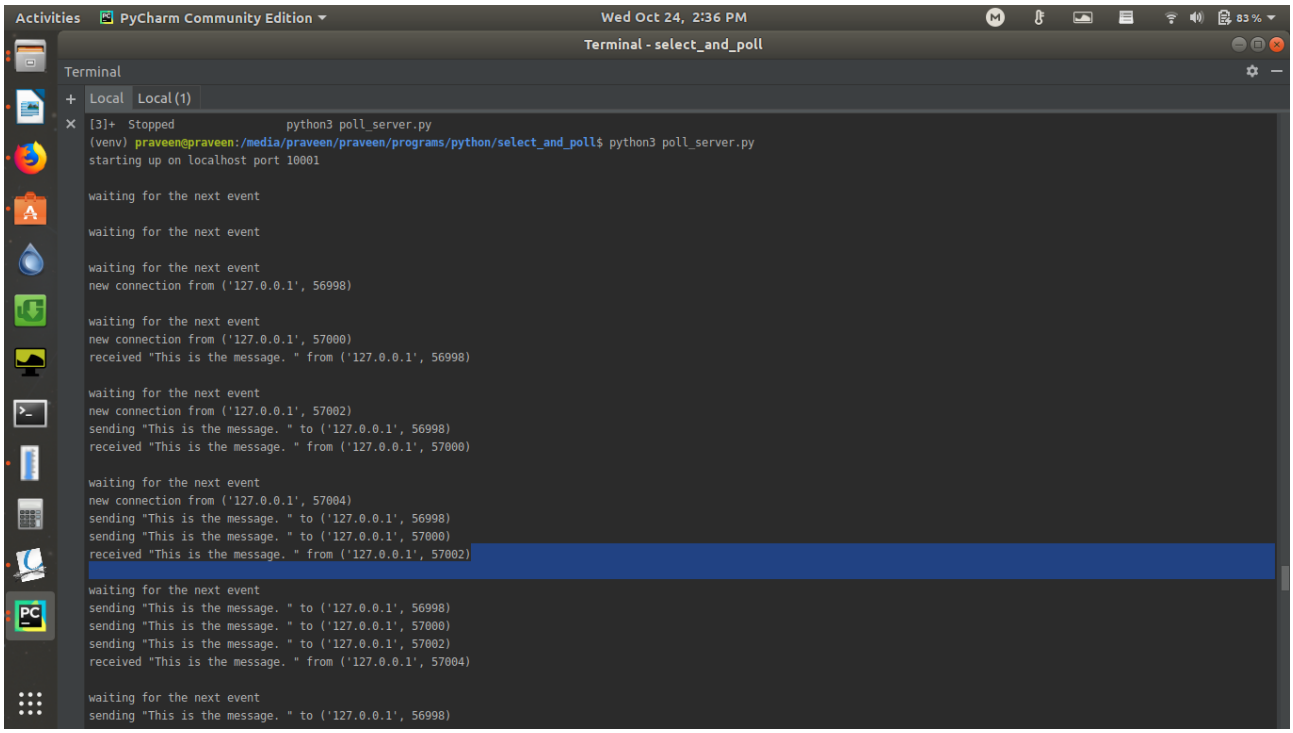
Select() and Poll()

client.py

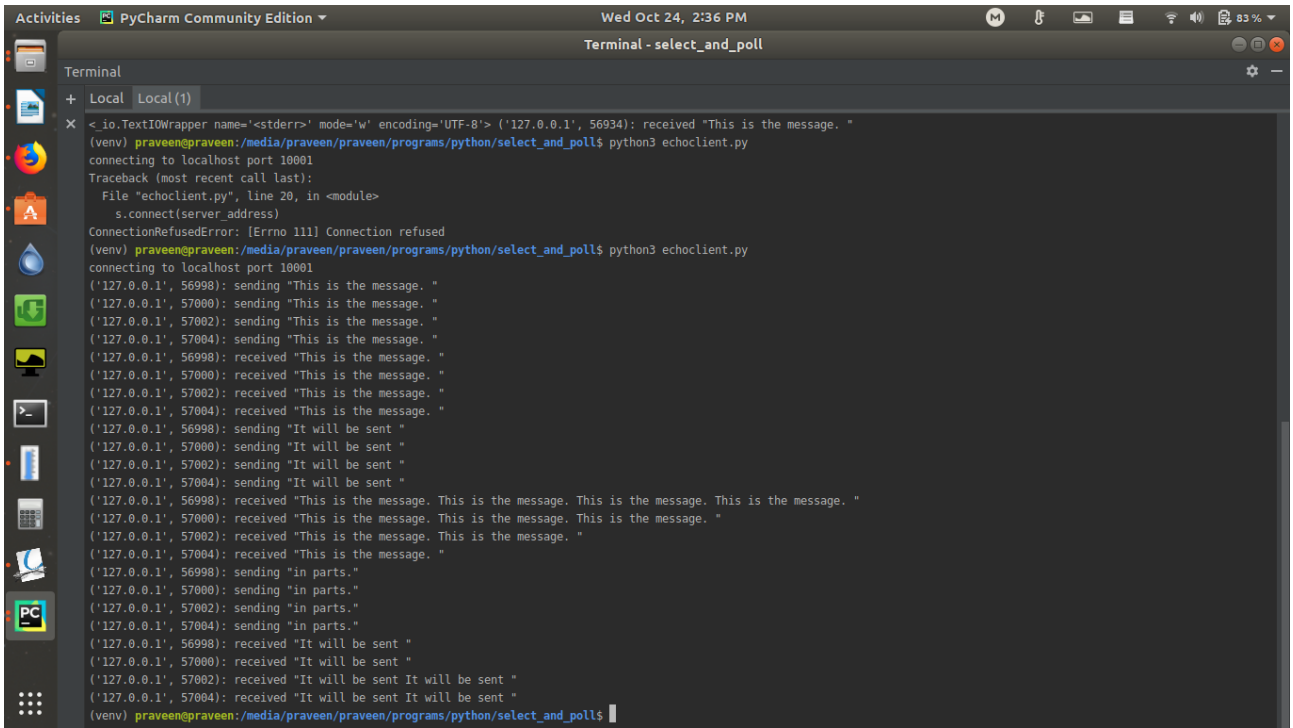
same client

Output :

poll_server:



poll_client :



Select() and Poll()

References:

1)https://pymotw.com/2/select/
2)https://docs.python.org/3/library/select.html#poll-objects