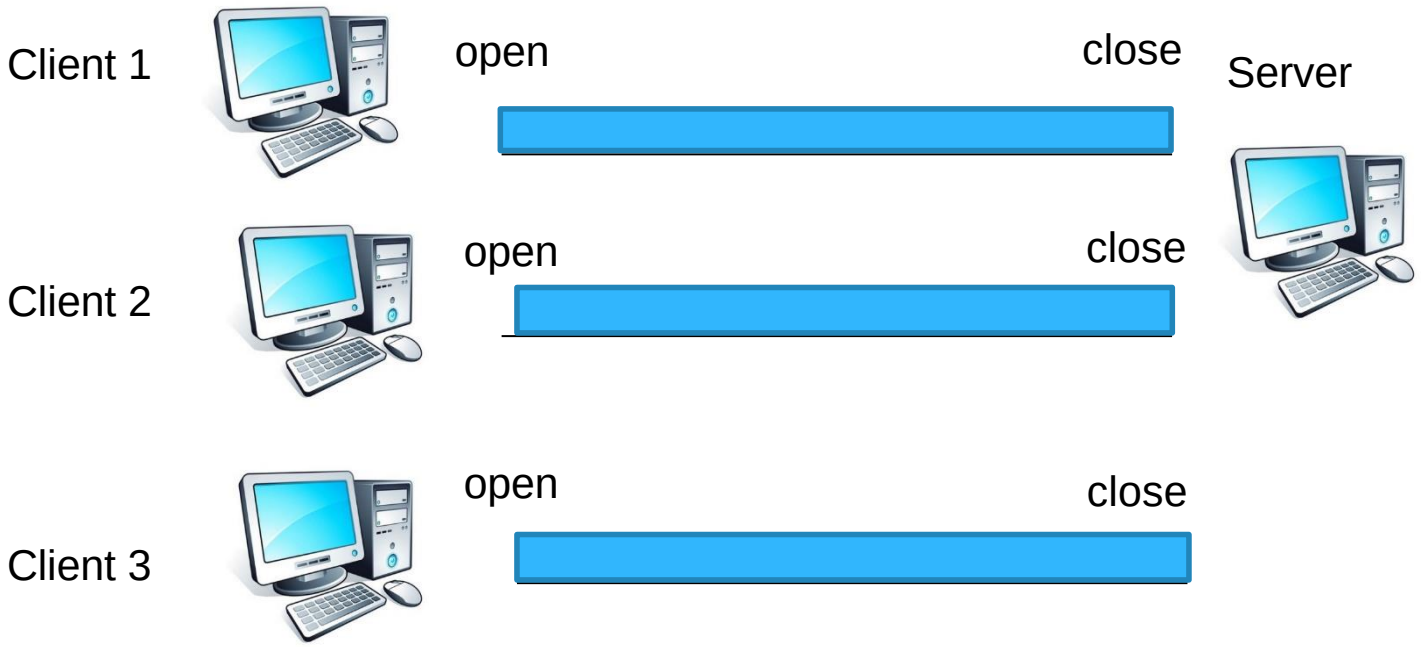
WORKSHOP PEMROGRAMAN JARINGAN
MODUL 8
(MULTIPLEXING SOCKET I/O: SELECT)

Mochammad Zen Samsono Hadi, ST. MSc. Ph.D

TOPIK PEMBAHASAN

- Penggunaan select.select
- Penggunaan select.epoll

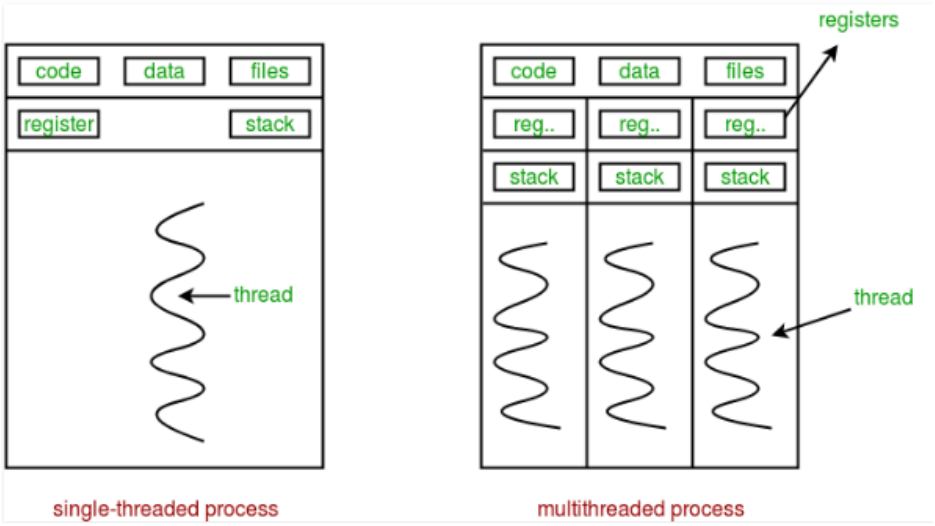
Multi Threaded Process TCP

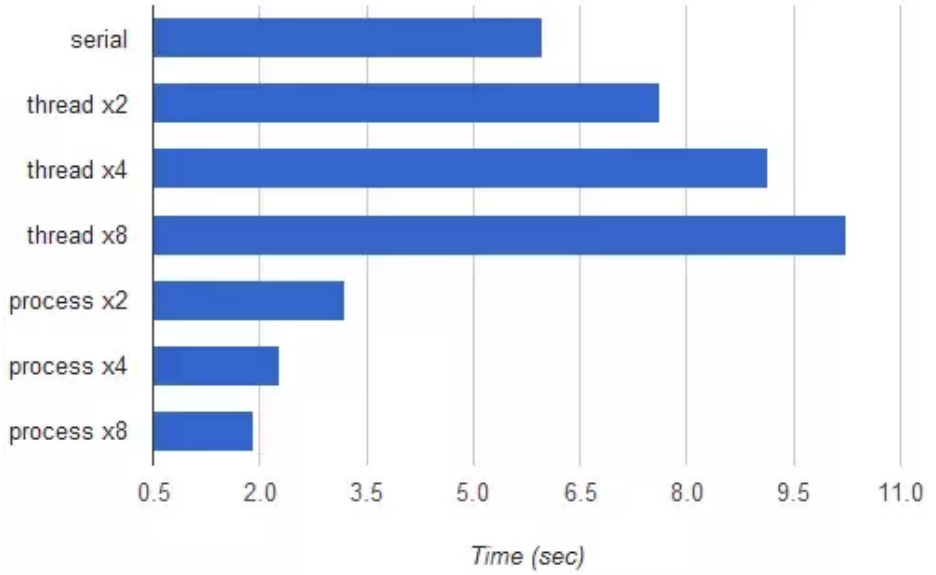


- Sistem multi threaded dengan membagi banyak jalur ke sisi client

Kelemahan Multi Threaded Process

- Penggunaan CPU dan sistem multi threaded





Penggunaan select.select

- Aplikasi server untuk menangani thread (process per client) yang berjumlah ratusan atau ribuan akan memboroskan memori dan CPU.
- Untuk mengatasi hal tersebut: **select** module
  - Memiliki fungsi monitoring I/O (non-blocking sockets)
  - Mengizinkan sebuah aplikasi utk memonitor banyak thread
  - Mengizinkan sebuah aplikasi server dan banyak client untuk menangani banyak tugas tanpa melakukan blocking proses send dan receive.

Fungsi send dan receive

```
# Some utilities
def send(channel, *args):
    buffer = pickle.dumps(args)
    value = socket.htonl(len(buffer))
    size = struct.pack("L",value)
    channel.send(size)
    channel.send(buffer)

def receive(channel):
    size = struct.calcsize("L")
    size = channel.recv(size)
    try:
        size = socket.ntohl(struct.unpack("L", size)[0])
    except struct.error as e:
        return ''
    buf = ""
    while len(buf) < size:
        buf = channel.recv(size - len(buf))
    return pickle.loads(buf)[0]
```

pickle: utk menyimpan dan membaca data ke/dari sebuah file

Stirng data: struct.pack => L (long)

Sisi server

```
class ChatServer(object):
    """ An example chat server using select """
    def __init__(self, port, backlog=5):
        self.clients = 0
        self.clientmap = {}
        self.outputs = [] # list output sockets
        self.server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.server.bind((SERVER_HOST, port))
        print ('Server listening to port: %s ...' %port)
        self.server.listen(backlog)
        # Catch keyboard interrupts
        signal.signal(signal.SIGINT, self.sighandler)
    def sighandler(self, signum, frame):
        """ Clean up client outputs"""
        # Close the server
        print ('Shutting down server...')
        # Close existing client sockets
        for output in self.outputs:
            output.close()
        self.server.close()
```

Menyimpan jumlah client dan memetakannya.

Banyak antrian

Hanya menangani thread yang mengirim / terima data

Sisi server

```
def run(self):
    inputs = [self.server, sys.stdin]
    self.outputs = []
    running = True
    while running:
        try:
            readable, writeable, exceptional = select.
            select(inputs, self.outputs, [])
        except select.error as e:
            break

        for sock in readable:
            if sock == self.server:
                # handle the server socket
                client, address = self.server.accept()
                print ("Chat server: got connection %d from %s" %
                        (client.fileno(), address))
```

readable: server siap menerima koneksi dari client
writeable: server broadcast informasi ke semua client.

Sisi Client

```
# Connect to server at port
try:
    self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.sock.connect((host, self.port))
    print ("Now connected to chat server@ port %d" % self.port)
    self.connected = True
    # Send my name...
    send(self.sock,'NAME: ' + self.name)
    data = receive(self.sock)
    # Contains client address, set it
    addr = data.split('CLIENT: ')[1]
    self.prompt = '[' + '@'.join((self.name, addr)) + ']> '
except socket.error as e:
    print ("Failed to connect to chat server
            @ port %d" % self.port)
    sys.exit(1)
```

Sisi Client

```
def run(self):
    """ Chat client main loop """
    while self.connected:
        try:
            sys.stdout.write(self.prompt)
            sys.stdout.flush()
            # Wait for input from stdin and socket
            readable, writeable,exceptional = select.select
                                              ([0, self.sock], [],[])
            for sock in readable:
                if sock == 0:
                    data = sys.stdin.readline().strip()
                    if data: send(self.sock, data)
                elif sock == self.sock:
                    data = receive(self.sock)
                    if not data:
                        print ('Client shutting down.')
                        self.connected = False
                        break
                    else:
                        sys.stdout.write(data + '\n')
                        sys.stdout.flush()
```

readable: server siap menerima koneksi dari client
writeable: server broadcast informasi ke semua client.

Hasil

Server

```
$ python 2_3_chat_server_with_select.py --name=server --port=8800
Server listening to port: 8800 ...
Chat server: got connection 4 from ('127.0.0.1', 59254)
Chat server: got connection 5 from ('127.0.0.1', 59256)
```

Client 1

Client 1

```
$ python 2_3_chat_server_with_select.py --name=client1 --port=8800
Now connected to chat server@ port 8800
[client1@127.0.0.1]>
(Connected: New client (2) from client2@127.0.0.1)
[client1@127.0.0.1]> Hello from client1
[client1@127.0.0.1]>
#[client2@127.0.0.1]>>hello from client2
[client1@127.0.0.1]>
```

```
$ python 2_3_chat_server_with_select.py --name=client2 --port=8800
Now connected to chat server@ port 8800
[client2@127.0.0.1]>
#[client1@127.0.0.1]>>Hello from client1
[client2@127.0.0.1]> hello from client2
[client2@127.0.0.1]>
```

Percobaan

- Cobalah listing 2.3 dengan jumlah client 4, amati dan catat hasilnya
- Cobalah listing 2.4, select yang diterapkan pada web server
  - Buka browser dengan 4 koneksi ke server
  - Amati dan catat hasilnya