Miscellaneous Topics

memcached

memcached

- Database lookups and reading files can be expensive
- Caching is used to speed up this process, but caches are usually machine-specific, leading to much duplication across server farms
- memcached creates a big memory cache across multiple servers
- Data is added/removed from this cache using the LRU policy
- Result is a huge Python dictionary cache in RAM

Message Queues

Message Queues

- Requests are queued for subsequent processing
- e.g., account signup confirmation emails are examples of message queues
- Can use message queues to create a group of workers that process incoming requests
  - Requests come in via the front end and are added to the message queue
  - On the back end, there are multiple clients each taking requests from the message queue, processing them, and (commonly) returning the result to the front end
  - Easy to add, replace, or remove back end clients

Message Queues

- To install, run the following in a pipenv shell:
  pipenv install pyzmq

Message Queue Server

- Derived from *https://zeromq.org/languages/python/*

```
#!/usr/bin/env python3

import zmq

context = zmq.Context()
socket = context.socket(zmq.REP)
socket.bind("tcp://*:5555")

while True:
    #  Wait for next request from client
    message = socket.recv()
    print("Received request: %s" % message)

    #  Send reply back to client (can use this to delegate work)
    socket.send(b"Send email to horiem@camosun.bc.ca")
```

Message Queue Server Notes

- *socket = context.socket(zmq.REP)* sets up a queue which expects a server to receive a request and then send a reply
- Can expand this by having the client go to a URL (e.g., server.camosun.ca) which resolves to multiple IPs, each running an instance of the message queue server

Message Queue Client

- Derived from *https://zeromq.org/languages/python/*

```
#!/usr/bin/python3.11

from zmq import Context, REQ

context = Context()

#  Socket to talk to server
print("Connecting to the server…")
socket = context.socket(REQ)
socket.connect("tcp://localhost:5555")

#  Do 10 requests, waiting each time for a response
for request in range(10):
    print("Sending request %s …" % request)
    socket.send(b"Ready")

    #  Get the reply.
    message = socket.recv()
    print("Received reply %s [ %s ]" % (request, message))
```

Message Queue Client Notes

- *socket = context.socket(zmq.REQ)* sets up a queue which expects a client to send a request and then receive a reply

Long-Term Connections

Long-Term Connections

- Most TCP connections are fairly short-lived
  - e.g., connect to web server, GET resource(s), close the connection
- Sometimes, connections must last hours, days, or longer
  - e.g., remote sensor monitoring
- Problem: The OS usually shuts down connections that have been idle for some time
- Solution: Send “keep alive” packets or segments on a regular basis

Server

```
#!/usr/bin/python3.11
from socket import socket, AF_INET, SOCK_STREAM, IPPROTO_TCP, SOL_SOCKET,
SO_KEEPALIVE, TCP_KEEPCNT, TCP_KEEPIDLE, TCP_KEEPINTVL
from traceback import format_exc

PORT = 12345

def echo(port):
    sock = socket(AF_INET, SOCK_STREAM)
    if hasattr(socket, 'SO_KEEPALIVE'):
        print('Enabling Keep Alive')
        sock.setsockopt(SOL_SOCKET, SO_KEEPALIVE, 1)
    if hasattr(socket, 'TCP_KEEPIDLE'):
        n = 5
        print('Starting idle check after', n, 'seconds')
        sock.setsockopt(IPPROTO_TCP, TCP_KEEPIDLE, n)
    if hasattr(socket, 'TCP_KEEPINTVL'):
        n = 10
        print('Continuing to ping every', n, 'seconds')
        sock.setsockopt(IPPROTO_TCP, TCP_KEEPINTVL, n)
    if hasattr(socket, 'TCP_KEEPCNT'):
        n = 3
        print('Deeming connection to be disconnected after', n, 'fails')
        sock.setsockopt(IPPROTO_TCP, TCP_KEEPCNT, n)
```

Enables Keep Alive functionality

Starts to check liveness after *n* seconds

Repeats liveness checks every *n* seconds

Deems disconnection after *n* fails

Server

```
    try:
        sock.bind(('localhost', port))
        sock.listen(1)
        sc, name = sock.accept()

        while True:
            data = sc.recv(1)
            if data == b'':
                break
            print(data)

    except Exception as e:
        print(e)
        print(format_exc())

    sock.close()


echo(PORT)
```

Client

```
#!/usr/bin/python3.11

from socket import socket, AF_INET, SOCK_STREAM
from traceback import format_exc

HOST = '127.0.0.1'
PORT = 12345


def echo(host, port):
    sock = socket(AF_INET, SOCK_STREAM)
    try:
        sock.connect((host, port))

        while True:
            data = input()
            sock.sendall(data.encode('utf-8'))

    except Exception as e:
        print(e)
        print(format_exc())

    sock.close()

echo(HOST, PORT)
```

Exercises

- Download the client and server code
- Run Wireshark
- Start the client and server and observe the network traffic flow

Key Skills

- Explain what memcached does
- Explain what message queues do
- Use keep alive packets to maintain a long-term network connection