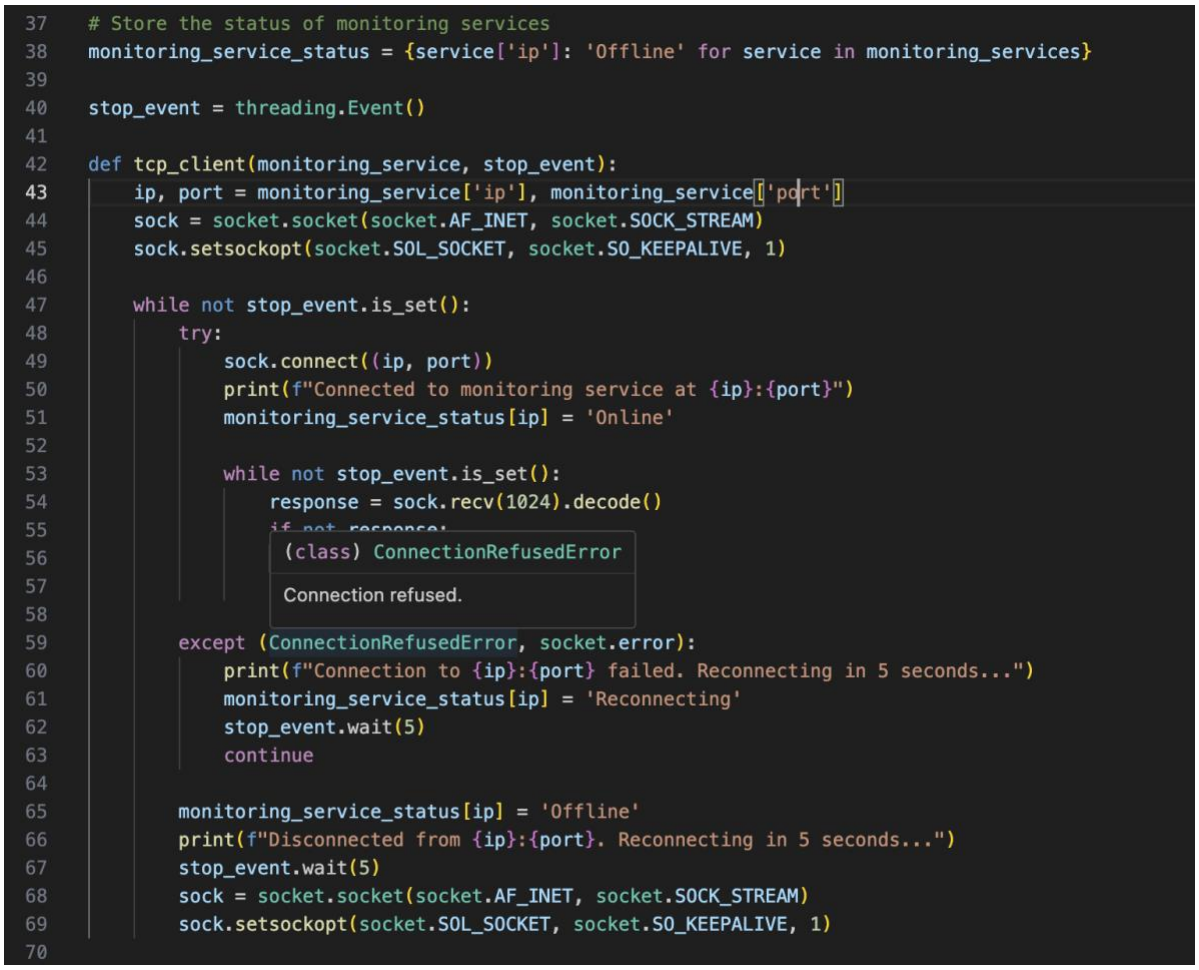
Socket Programming Project 2 - TCP/IP Programming

To run the application, first start the monitoring service by opening a terminal, navigating to the directory containing `monitoring_service.py`, and typing `**sudo python3 monitoring_service.py**`. Then, open another terminal, navigate to the directory containing management_service.py, and run the script with `**python3 management_service.py**`.

Management Service Software Requirements Specification (SRS) (Abbreviated):

1. The Management Service is performing the existing tasks related to configuring and managing monitoring tasks. It manages connections to monitoring services and distributes tasks for checking the status of various servers (HTTP, HTTPS, ICMP, DNS, NTP, TCP, UDP). It maintains real-time status updates of these monitoring services and provides a command interface for users to control the application.




```
37  # Store the status of monitoring services
38  monitoring_service_status = {service['ip']: 'Offline' for service in monitoring_services}
39
40  stop_event = threading.Event()
41
42  def tcp_client(monitoring_service, stop_event):
43      ip, port = monitoring_service['ip'], monitoring_service['port']
44      sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
45      sock.setsockopt(socket.SOL_SOCKET, socket.SO_KEEPALIVE, 1)
46
47      while not stop_event.is_set():
48          try:
49              sock.connect((ip, port))
50              print(f"Connected to monitoring service at {ip}:{port}")
51              monitoring_service_status[ip] = 'Online'
52
53              while not stop_event.is_set():
54                  response = sock.recv(1024).decode()
55                  if not response:
56
57
58
59          except (ConnectionRefusedError, socket.error):
60              print(f"Connection to {ip}:{port} failed. Reconnecting in 5 seconds...")
61              monitoring_service_status[ip] = 'Reconnecting'
62              stop_event.wait(5)
63              continue
64
65          monitoring_service_status[ip] = 'Offline'
66          print(f"Disconnected from {ip}:{port}. Reconnecting in 5 seconds...")
67          stop_event.wait(5)
68          sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
69          sock.setsockopt(socket.SOL_SOCKET, socket.SO_KEEPALIVE, 1)
70
```

2. It is also establishing and maintaining persistent TCP connections to multiple monitoring services, each identified by IP and port numbers. It has a saved configuration that contains the list of Monitoring Services.

```
10    # Configuration of monitoring services
11    monitoring_services = [
12        {'ip': '127.0.0.1', 'port': 54322},
13    ]
```

3. It also holds all configuration information for monitoring tasks, including service types to monitor, parameters for checks, and check frequencies.

```
15    # Configuration of tasks
16    config = {
17        'google.com': {
18            'HTTP': {'url': 'http://www.google.com', 'frequency': 2},
19            'HTTPS': {'url': 'https://www.google.com', 'frequency': 2},
20            'ICMP': {'server_address': '8.8.8.8', 'frequency': 2},
21            'DNS': {'server_address': '8.8.8.8', 'frequency': 1},
22            'NTP': {'server_address': 'pool.ntp.org', 'frequency': 2},
23            'TCP': {'port': 80, 'frequency': 2},
24            'UDP': {'server_address': '8.8.8.8', 'port': 53, 'frequency': 2}
25        },
26        'oregonstate.edu': {
27            'HTTP': {'url': 'http://oregonstate.edu/', 'frequency': 2},
28            'HTTPS': {'url': 'https://oregonstate.edu/', 'frequency': 2},
29            'ICMP': {'server_address': '128.193.0.10', 'frequency': 2},
30            'DNS': {'server_address': '128.193.0.10', 'frequency': 1},
31            'NTP': {'server_address': 'pool.ntp.org', 'frequency': 2},
32            'TCP': {'port': 80, 'frequency': 2},
33            'UDP': {'server_address': '128.193.0.10', 'port': 53, 'frequency': 2}
34        }
35    }
```

It also assigns and manages unique IDs for each monitoring service, linking them to specific monitoring tasks.

```
126        task_id = str(uuid.uuid4())  # Generate a unique task ID
```

4. Status Display: It provides real-time status updates of all connected monitoring services.

```
96    def display_status(stop_event):
97        while not stop_event.is_set():
98            print("\nReal-time status of monitoring services:")
99            for ip, status in monitoring_service_status.items():
100               print(f"{ip}: {status}")
101           stop_event.wait(10)
102
```

```
project-2 — Python management_service.py — 90×43
Real-time status of monitoring services:
127.0.0.1: Online

------Distributing tasks for server: google.com-----

[2024-05-15 11:26:32] Task ID: da2ada48-6a82-4c7e-9dae-0df6a849f6b2, HTTP URL: http://www.
google.com, HTTP server status: True, Status Code: 200
[2024-05-15 11:26:33] Task ID: ecfabcc9-8562-4cd9-90c1-6fef681b1577, HTTPS URL: https://ww
```

```python
def tcp_client(monitoring_service, stop_event):
    ip, port = monitoring_service['ip'], monitoring_service['port']
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_KEEPALIVE, 1)

    while not stop_event.is_set():
        try:
            sock.connect((ip, port))
            print(f"Connected to monitoring service at {ip}:{port}")
            monitoring_service_status[ip] = 'Online'

            while not stop_event.is_set():
                response = sock.recv(1024).decode()
                if not response:
                    break
                print(f"Received: {response}")

        except (ConnectionRefusedError, socket.error):
            print(f"Connection to {ip}:{port} failed. Reconnecting in 5 seconds...")
            monitoring_service_status[ip] = 'Reconnecting'
            stop_event.wait(5)
            continue

        monitoring_service_status[ip] = 'Offline'
        print(f"Disconnected from {ip}:{port}. Reconnecting in 5 seconds...")
        stop_event.wait(5)
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_KEEPALIVE, 1)
```

5. Task Distribution and Validation:

   The unique ID is retrieved from the Monitoring Service and displayed on the terminal.

```
project-2 — Python management_service.py — 90×43
ort 53 - Open, Description: Port 53 on 8.8.8.8 is open or no response received.

------Distributing tasks for server: oregonstate.edu-----

[2024-05-15 11:26:36] Task ID: 9c352961-6d1d-4d3e-881e-03e8dc7d8020, HTTP URL: http://oreg
onstate.edu/, HTTP server status: True, Status Code: 200
```

Monitoring Service Software Requirements Specification (SRS) (Abbreviated):

1. The Monitoring Service is performing the existing tasks related to running service checks (HTTP, HTTPS, ICMP, DNS, NTP, TCP, UDP).

```
27  def create_icmp_packet(icmp_type=8, icmp_code=0, sequence_number=1, data_size=192):
28      thread_id = threading.get_ident()
29      process_id = os.getpid()
30      icmp_id = zlib.crc32(f"{thread_id}{process_id}".encode()) & 0xffff
31      header = struct.pack('bbHHh', icmp_type, icmp_code, 0, icmp_id, sequence_number)
32      random_char = random.choice(string.ascii_letters + string.digits)
33      data = (random_char * data_size).encode()
34      chksum = calculate_icmp_checksum(header + data)
35      header = struct.pack('bbHHh', icmp_type, icmp_code, socket.htons(chksum), icmp_id,
        sequence_number)
36      return header + data
37
38  def ping(host, ttl=64, timeout=1, sequence_number=1):
39      with socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_ICMP) as sock:
40          sock.setsockopt(socket.IPPROTO_IP, socket.IP_TTL, ttl)
41          sock.settimeout(timeout)
42          packet = create_icmp_packet(sequence_number=sequence_number)
43          sock.sendto(packet, (host, 1))
44          start = time.time()
45          try:
46              data, addr = sock.recvfrom(1024)
47              end = time.time()
48              total_ping_time = (end - start) * 1000
49              return addr, total_ping_time
50          except socket.timeout:
51              return None, None
52
53  def check_server_http(url):
54      try:
55          response = requests.get(url)
56          is_up = response.status_code < 400
57          return is_up, response.status_code
58      except requests.RequestException:
59          return False, None
60
61  def check_server_https(url, timeout=5):
62      try:
63          headers = {'User-Agent': 'Mozilla/5.0'}
64          response = requests.get(url, headers=headers, timeout=timeout)
65          is_up = response.status_code < 400
```

2. It is using a unique identifier for interacting with the Management Service.

```
126     task_id = str(uuid.uuid4())  # Generate a unique task ID
127
128     if service == 'HTTP':
129         if 'url' in details:
130             url = details['url']
131             is_up, status_code = check_server_http(url)
132             response = f"[{timestamp}] Task ID: {task_id}, HTTP URL: {url}, HTTP server status:
                {'True' if is_up else 'False'}, Status Code: {status_code if status_code is not None
                else 'N/A'}"
133         else:
134             response = f"[{timestamp}] Task ID: {task_id}, HTTP: No URL configured for
                {server_name}"
```

3. It performs assigned monitoring tasks according to the received configurations and frequencies.

```
118 def handle_client(client_sock):
119     message = client_sock.recv(1024).decode()
120     data = json.loads(message)
121     server_name = data['server_name']
122     service = data['service']
123     details = data['details']
124     timestamp = data['timestamp']
125
126     task_id = str(uuid.uuid4())  # Generate a unique task ID
127
128     if service == 'HTTP':
129         if 'url' in details:
130             url = details['url']
131             is_up, status_code = check_server_http(url)
132             response = f"[{timestamp}] Task ID: {task_id}, HTTP URL: {url}, HTTP server status:
                {'True' if is_up else 'False'}, Status Code: {status_code if status_code is not None
                else 'N/A'}"
133         else:
134             response = f"[{timestamp}] Task ID: {task_id}, HTTP: No URL configured for
                {server_name}"
135     elif service == 'HTTPS':
136         if 'url' in details:
137             url = details['url']
138             is_up, status_code, description = check_server_https(url)
139             response = f"[{timestamp}] Task ID: {task_id}, HTTPS URL: {url}, HTTPS server status:
                {'True' if is_up else 'False'}, Status Code: {status_code if status_code is not None
                else 'N/A'}, Description: {description}"
140         else:
141             response = f"[{timestamp}] Task ID: {task_id}, HTTPS: No URL configured for
                {server_name}"
```

4. Result Reporting:

5. The Monitoring Service must be able to allow for the Management Service to reconnect without needing to restart the Monitoring Service.

```
        monitoring_service_status[ip] = 'Offline'
        print(f"Disconnected from {ip}:{port}. Reconnecting in 5 seconds...")
        stop_event.wait(5)
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.setsockopt(socket.SOL_SOCKET, socket.SO_KEEPALIVE, 1)
```