Load Balancer

המערכת שבנינו מתחברת לשלושת השרתים הקיימים ושומרת את הפרטים עליהם במילון:

- כתובת Ip
- socket fd
- זמן סיום בקשות צפוי
- סוג הבקשות שהשרת יודע לקבל

```
#servers - keep ip, sock pd, finish time for servers management and types can handle
servers = {
    '1': {'addr': '192.168.0.101', 'sock': None, 'finish_time': 0, 'can_handle': ['V', 'P']},
    '2': {'addr': '192.168.0.102', 'sock': None, 'finish_time': 0, 'can_handle': ['V', 'P']},
    '3': {'addr': '192.168.0.103', 'sock': None, 'finish_time': 0, 'can_handle': ['M']}
}
```

כדי לאפשר עבודה מקבילית ונכונה עשינו שימוש בחוטים (הגבלנו את מספרם ל-50 ע״י semaphore) ומנעול על נתוני המילון.

```
SERV_HOST = '10.0.0.1'
PORT_80 = 80
lock = threading.Lock() # To calculate finish time safely - each time only one req
thread_semaphore = threading.Semaphore(50) # Limit to 50 concurrent threads
```

המערכת שלנו מאזינה על פורט 80 וממתינה לבקשות (bind, listen). כל בקשה נשמרת בתור בגודל 10 עד לקבלה וטיפול על ידי אחד החוטים (accept).

```
server_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

#listening on '10.0.0.1' port 80 for clients; new request handled using new thread
try:
    server_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_sock.bind((SERV_HOST, 80))
    server_sock.listen(10) #queue size 10
    TimePrint("Listening on %s:80" % SERV_HOST)

    while True:
        client_sock, client_addr = server_sock.accept()
        thread_semaphore.acquire()
        thread = threading.Thread(target=handle_client, args=(client_sock, client_addr))
        thread.daemon = True
        thread.start()
finally:
    server_sock.close()
```

כאשר חוט מטפל בבקשה הוא בודק תחילה את תוכנה והאם תואמת את סוגי הבקשות שהשרתים שלנו יודעים לקבל.

```
def handle_client(client_sock, client_addr):
    try:
        req = client_sock.recv(2)
        if len(req) < 2:
            TimePrint("Invalid request from %s" % (client_addr,))
            client_sock.close()
            return

        req_type, req_time = parse(req)
        if req_type is None or req_type not in ['V', 'M', 'P']:
            TimePrint("Invalid request type '%s' from %s" % (req_type, client_addr))
            client_sock.close()
            return

        servID = getOptimalServer(req_type, req_time)
```

לאחר מכן, המערכת תשלח את הבקשה אל השרת שזמן הסיום הצפוי שלו הוא הקצר ביותר מבין כל השרתים שיכולים לטפל בבקשה מסוג זה.

```
with lock:
    for server_id, server_info in servers.items():
        # Check if server can handle this request type
        if req_type not in server_info['can_handle']:
            continue

        # Check if server socket is available
        if server_info['sock'] is None:
            continue

        # Calculate when this server will be free
        server_finish_time = max(current_time, server_info['finish_time'])

        if server_finish_time < earliest_finish:
            earliest_finish = server_finish_time
            best_server = server_id
```

לאחר בחירת השרת נעדכן את זמן הסיום הצפוי שלו (נוסיף את זמן ביצוע הבקשה הנוכחית).

דוגמה:

התקבלה בקשה מסוג 'וידאו', שרת 1 צפוי לסיים בעוד 10 שניות, שרת 2 בעוד 5 שניות ושרת 3 פנוי. שרת 2 יקבל את הבקשה לביצוע כיוון שמסיים לפני שרת 1. שרת 3 מטפל בקבצי מוזיקה בלבד ולכן לא יקבל בקשה זו ('וידאו').