|  |  |
| --- | --- |
| **Distributed System Programming – Assignment 2** | **SDH3-A**  **Laiba Asif (R00201303)**  **10/12/2023** |

Table of Contents

[Part 1: Basic Server and Client Connection 1](#_Toc152858629)

[Part 2: Handle Multiple Simultaneous 6](#_Toc152858635)

[Part 3: RabbitMQ 11](#_Toc152858641)

[Part 4: Docker Container 16](#_Toc152858664)

# 

# Part 1: Basic Server and Client Connection

Client Code

A screen shot of a computer program

Description automatically generated

A computer screen with white and green text

Description automatically generated

Server Code A screen shot of a computer

Description automatically generated A screenshot of a computer program

Description automatically generated

### Explanation:

The server script configures a server socket, manages incoming connections, and interprets client commands using an employee data dictionary that has been predefined.

The client script connects to the server, lets users request information about an employee's yearly leave or income, and keeps interacting with the server until the user chooses to stop.

Together, these scripts form a basic client-server architecture that facilitates the querying of HR-related data. The client communicates with the server to retrieve information depending on user input, and the server maintains employee data.

Here's the code explanation:

***Server file:***

**Importing module “socket”:**

* In order to facilitate network communication, the script first imports the socket module, which offers capability for handling sockets.

**Defining Dictionary “employees”:**

* To hold data about employees, such as their names, present pay, and leave entitlements, a dictionary called employees is created.

**Defining Function “handle\_command”:**

* To handle commands from clients, the handle\_command function is defined. It handles particular instructions, like getting the most recent salary information, and verifies command structures and employee IDs.

**Defining Function “main”:**

* The server-side logic is handled by the main function.
* It creates a socket, assigns it a port and address, and begins to wait for connections.
* Until a client provides the 'EXIT' command to cease the connection, the server constantly accepts client connections, receives commands, and acts accordingly.

**Verifying the Direct Execution of the Script:**

* The script invokes the main function to carry out the server logic after determining whether it is being run directly rather than being imported as a module.

***Client file***

**Importing module “socket”:**

* To handle sockets, the socket module is imported, just like the server script.

**Defining Function “send\_command”:**

* Define the function send\_command: To send commands to the server and get responses, use the send\_command function. The command is encoded, sent to the server, and the return is decoded.

**Defining Function “main”:**

* The client-side logic is handled by the main function.
* It connects to the server, asks the user for the employee ID and the nature of the inquiry (annual leave or salary), and then sends the order to the server.
* The user is stuck in the client loop until they choose to end it by typing 'X.'

**Verifying the Direct Execution of the Script:**

* Like the server script, the script determines whether it is being run directly by calling the main function, which then runs the client logic.

Code Run

A screenshot of a computer program

Description automatically generated

### Explanation:

Conclusion: This code demonstrates how to create shallow and deep copies of an object using the `copy` module in Python.

In the above screenshot, the provided output of the code execution shows examples of querying salary information for a valid employee ID (E00123) and handling an invalid employee ID (R00201303). The code terminates when the user chooses to exit.

# Part 2: Handle Multiple Simultaneous

Server Code

A screenshot of a computer program

Description automatically generated

Client Code A computer screen shot of a program code

Description automatically generatedA computer screen shot of a program

Description automatically generated

### Explanation:

program in place that acts as a server using sockets. This program likely listens for incoming connections from clients and responds to their requests. now in this part i added some modifications to enhance the server's capability to handle multiple client connections concurrently. For example introduce of concepts such as multi-threading or asynchronous programming to enable the server to juggle multiple client connections concurrently. This way, the server can accept connections from several clients and respond to their requests independently without waiting for one client to finish before handling the next one.

Here's the code explanation:

***Server file:***

**Functionality:**

* The server manages information about employees (hardcoded for simplicity).
* It provides functions to verify employee IDs and handle various commands related to employee information.
* It uses threading to handle multiple client connections simultaneously**.**

**Modifications for Handling Multiple Clients:**

* The ‘handle\_client’ function is responsible for handling individual client connections. It continuously receives commands from a client until the client sends an 'EXIT' command.
* The server is designed to accept multiple client connections concurrently. Each client connection is handled in a separate thread using ‘threading.Thread’.

**Concurrency:**

* The server creates a new thread (‘client\_thread’) for each incoming client connection using ‘threading.Thread’.
* This allows the server to handle multiple clients simultaneously.
* The main function of the server create a socket and bind it to a specific address and port.
* It then enters into a loop where it continuously accepts incoming connections using ‘server\_socket.accept()’.

***Client file:***

**Functionality:**

* The client interacts with the server by sending commands and receiving responses.
* It has a simple text-based interface for querying employee information.

**Modifications for Handling Multiple Clients:**

* The client is designed to interact with the server in a loop until the user decides to exit.
* Each client operates independently and does not interfere with others.
* Inside the loop, when a new client connection is accepted, the ‘handle\_client’ function is called in a new thread.
* This is achieved by creating a new ‘threading.Thread’ object and passing the ‘handle\_client’ function as the target, along with the client socket and address as arguments.

**Independence of Threads:**

* Each client connection is handled in its own thread. Meaning that multiple clients can connect to the server simultaneously, and each client's communication is processed independently in

## Code Run

A screen shot of a computer

Description automatically generated

### Explanation:

Conclusion: This code demonstrates how the server is structured to handle multiple client connections concurrently through threading. And the client is designed to interact with the server and can be run independently.

1. The server will output "Server started!" and then enter a loop where it waits for incoming connections.
2. Each client will start by connecting to the server and print "HR System 1.0."
3. The client will then prompt the user for an employee ID and a command
4. The server will print the "Connected to:" message each time a new client connects.
5. As clients are handled in separate threads, there may be an interleaved output from different clients, demonstrating the concurrent nature of the server's handling of multiple connections.

# Part 3: RabbitMQ

Server Code A screenshot of a computer

Description automatically generated A screen shot of a computer

Description automatically generated A screenshot of a computer program

Description automatically generated

Log Code

A screen shot of a computer program

Description automatically generatedExplanation:

use of RabbitMQ as a message broker to send messages from the server to a queue. The server communicates with RabbitMQ via the 'pika' library and logs activity information, such as employee ID, command, and client IP address, to the RabbitMQ queue. I built a basic Python file named "log\_consumer.py" in order to build a simple script that prints out the information of the activity log from the RabbitMQ queue. After this is executed, it will keep checking the RabbitMQ queue for new messages and publish the log information whenever one is received. Prior to executing the log consumer script for activity\_log, I will run the server and client scripts. In this manner, the log consumer script will print out the log messages that the server publishes to the RabbitMQ queue.

Here's the code explanation:

***Modified Server file***

**Importing Modules:**

* The script starts by importing the necessary modules, including socket, threading, and pika for RabbitMQ integration.

**Defining Employee Data and RabbitMQ Configuration:**

* A dictionary named employees is defined to store employee information.
* RabbitMQ configuration details such as host, port, and queue name are set.

**Defining Function” verify\_employee”:**

* The ‘verify\_employee’ function checks if an employee is present in the employees dictionary.

**Defining Function”** **send\_to\_rabbitmq”:**

* The ‘send\_to\_rabbitmq’ function sends information about a command, its options, client address, and employee ID to RabbitMQ.
* It accepts connections in a loop and spawns a new thread (‘handle\_client’) for each connected client.
* The ‘handle\_client’ function in the server manages the client's requests.
* It verifies the employee ID and processes the command accordingly.
* For each valid command, it calls the rabbitmq function to send information to the RabbitMQ queue.
* The log\_consumer.py script is started, creating a connection to RabbitMQ and waiting for messages.
* It prints received messages from the RabbitMQ queue.

**Defining Function”** **handle\_command”:**

* The ‘handle\_command’ function processes client commands received through sockets.
* It verifies employee IDs, handles various salary and leave queries, and sends the information to RabbitMQ using rabbitmq.

**Defining Function”** **handle\_client”:**

* The ‘handle\_client’ function manages the communication with a connected client.
* It continuously receives commands from the client, handles them using ’handle\_command’, and exits when the client sends 'EXIT.'

**Defining Function “main”:**

* The main function sets up a server socket, binds it to a specified address and port, and listens for incoming connections.
* It accepts connections in a loop and spawns a new thread (‘handle\_client’) for each connected client.

***Activity Log - RabbitMQ:***

**Importing “pika”:**

* The script imports the pika module for handling RabbitMQ communication.

**Configuring RabbitMQ Connection:**

* RabbitMQ host, port, and queue details are configured.

**Connect to RabbitMQ and Set Up Callback:**

* The script connects to RabbitMQ, declares a queue, and sets up a callback function (callback) to handle incoming log messages.

**Defining Function” Callback”:**

* The ‘callback’ function extracts and prints received log messages, including employee ID, command, options, client address, and IP details.

**Starting Message Consumption:**

* The script starts consuming messages from the RabbitMQ queue and prints activity log details.

Code RunA computer screen with white text

Description automatically generated

Explanation:

* The server.py script is started, creating a server socket, and listening for incoming connections.

# Part 4: Docker Container

Requirement 

Docker file A screenshot of a computer

Description automatically generated

Client code A screenshot of a computer

Description automatically generated A screenshot of a computer program

Description automatically generated

Server code A computer screen shot of a black screen

Description automatically generated A screen shot of a computer code

Description automatically generated A screen shot of a computer

Description automatically generated A screen shot of a computer program

Description automatically generated

Explanation:

Python scripts and Docker containers are used to construct a rudimentary Human Resources (HR) system. An in-memory database of personnel data is maintained by the server script (`s.py`), which also handles client connections and commands. The user can communicate with the server by submitting commands and receiving responses through the client script `c.py`. Furthermore, by consuming messages from a queue, a RabbitMQ log consumer script ({log\_consumer.py}) logs activity. The server script and dependencies are packaged as a Docker image using Dockerization, making deployment simple. The system operates on a client-server architecture, allowing users to view server-side logs and RabbitMQ messages while interacting with the HR System via the client script. For testing and scalability purposes, this configuration mimics a distributed system with containerized components.

Here's the code explanation:

***Server file:***

* In-Memory Employee Data: The script defines a dictionary (‘employees’) containing information about employees.
* RabbitMQ Configuration: Configures RabbitMQ connection details such as host, port, and queue name.
* verify\_employee Function: Checks if a given employee ID is in the employee data.
* rabbitmq Function: Sends information to RabbitMQ for logging purposes.
* handle\_command Function: Processes received commands and sends appropriate responses. Also logs information using RabbitMQ.
* handle\_client Function: Handles communication with a client, processing commands until the client sends an 'EXIT' command.
* main Function: Sets up a server socket, listens for incoming client connections, and spawns a new thread to handle each connected client.

***Client file:***

* snd\_cmd Function: Sends a command to the server and receives the response.
* main Function: Connects to the server, queries employee information, sends commands, and displays server responses. It allows the user to continue or exit the interaction.

***RabbitMQ Log Consumer file:***

* Connects to RabbitMQ and consumes log messages from the 'activity\_logs' queue.
* Defines a callback function (‘callback’) to process received log messages.
* Starts consuming messages and prints them to the console.

***Requirements.txt:***

* Specifies the Python package (‘pika’) and its version required for the project.

***Dockerfile:***

* Uses a Python 3.8 slim base image.
* Sets the working directory inside the container.
* Copies local files to the container, including the ‘requirements.txt’.
* Installs dependencies specified in ‘requirements.txt’.
* Exposes port 8888.
* Launches the ‘s.py’ script when the container starts.

***Dockerization Process:***

* The Dockerfile sets up the Python environment and includes necessary dependencies.
* The server script (‘s.py’) is launched when the container starts, exposing port 8888.
* The RabbitMQ log consumer script (‘log\_consumer.py’) is not directly part of the Dockerfile but can be executed separately.

Code Run A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

Explanation:

1. Build the Docker image using: **docker build -t hr\_system\_server .**
2. Run the Docker container using: **docker run -p 8888:8888 hr\_system\_server**
3. On new terminal run the client file: **c.py**
4. On new terminal build the activity log: **Python log\_consumer.py**
5. Interact will the c.py file.
6. Observe the activity log terminal for log message received.