# Assignment: Designing A Network

---

Team members: Maryam Bunama, Eric Russon, Roman Kapitoulski

*CPRG-217 | Date Due: Jul 30, 2023 | Instructor: Sanoop Sadique*

# Table of Contents

# Assignment Problem

In this assignment, we have been hired by a company to have important information be sent from their client machines to a centralized information storage system server so they can continuously process and monitor their machines. Having to do this for all machines manually would take much too long, so we have been given permission to create scripts to automatically create documents and have them be sent off to the storage system. These scripts will utilize Linux cron jobs that will run as soon as the machine is turned on. The client machines will need one script to create the document needed, and another script to connect and send the documents to the server or storage system. The server or storage systems will be required to run a script to receive the data and store it periodically.

# Solution

>> Please see the attached code!

Our solution to this problem was to create 3 scripts. The first script will create the documents and save them as a JSON file for easy transport over the network. The second script will act as the network communication device for the machine and have it automatically connect to the receiving machine, have it establish the framework for messages sent, and ensure that the documents are sent complete and clean of corruption. The final script will act as the receiver machine to receive communication messages from client machines, ensure a solid connection, ensure the documents received are complete, and continue to listen for other clients that wish to connect.

For the first script, it will be run as a cron job on a Linux machine. This will ensure that the important document will be made at a certain time or date as frequently or as sparsely as needed by the company. The document will be created and sent as a JSON file and will contain the name of the machine, the users and groups of that machine, as well as the services, and their activity status.

The second script will be used as the network connection between it and the server or storage system. For this to work, we need to have some groundwork set up before it's allowed to send any important documents. For the connection to work, it must first establish a connection to the

correct server or storage system using a handshake. After that, if the connection was successful, our client will send an initial message indicating the name of the file, the size of the file being sent, the hash of the document, the separator between the message, as well as the “end of file” message. Optionally, it can also say how many bytes will be sent as a buffer size. Once all this is established with the server, it can then proceed to send the document. Following the document will be the end of the data message telling the server that all the data has been sent and would like to confirm that it has been received and is ok to close the connection. If the reply is the confirmation message, the client will automatically close its connection to the server. If the reply shows that the data was incomplete or corrupted during transmission, then the client will be required to resend the document until the reply becomes a confirmation, or a warning message is sent to the IT to have the client machine checked over.

The final script is the one acting as the receiver. In this case, it will be the server receiving the data and storing it. Just like with the client, the server must first ensure that it is connected to a client. We can ensure this by showing the IP address of the client machine that is currently connected. After that, the same groundwork must be in place before the server can receive any documents. It will wait for the initial greeting message that has all the information about the document being sent and its hash, the size to expect, the separator indicator, the end of data message, and the optional buffer size that the client will send it as. Once this has been established, it will indicate to the client that it is ready to receive the document. Upon completion of sending all the data and checking for the end of data message, the server will take extra steps by also verifying the hash of the sent document with the initial hash sent by the client. If it sees that the two hashes are the same, it will reply with a confirmation for the client to close its connection, while the server will continue to listen for other clients. If the server detects that the hash is different from the initial hash, the server will then remove the newly downloaded document to prevent any viruses that may be present in the document, as well as send a reply to the client informing it of the possible corruption or incomplete document sent and ask that the client resend the document. The server will repeat this for each client that initiates communication with the server but can only communicate with one client machine at a time.

# Pseudocode

---

**SysAdminTask.py**

1. Check the computer info (SysAdminTask.py)
    a. Locate needed information
        i. Check for computer name/machine name
            1. if no machine name located:
                a. locate: host name
            2. write machine name to .json
        ii. Locate all users and associated groups (alphabetically)
            1. Check /etc/passwd for users and groups
                a. Record Users
                b. Lookup groups User is part of
                c. Record Groups
            2. write users and groups to .json
        iii. Retrieve processor info
            1. Read /proc/cpuinfo
                a. Retrieve vender_id
                b. Retrieve Model
                c. Retrieve Model name
                d. Retrieve Cache
            2. write cpu info to .json
        iv. Check status of currently running services on machine
            1. locate the current running services
            2. place services into lists
            3. copy status of server to list
            4. write services to .json
                a. optional: copy all services into a new temp text file.
                    i. read lines of temp text file
                    ii. copy information into lists
                    iii. write lists into JSON file
    b. Write information as a .json file
        i. Retrieve the current date and time in the format
            1. Append the time information to the json file name
            2. Name file as SystemResults_MM-DD-YYYY_HH-MM.json
        ii. Append Machine Name
        iii. Append CPU Info
        iv. Append Users

1. Append Groups

v. Append Services
    1. Append Service Status
vi. Format .json file
vii. Save file to /var/log/SysCheckLogs

**Client.py**

1. Retrieve JSON file from /var/log/SysCheckLogs
    a. Select the most recent file
        i. Iterate through files in a folder using some algorithm to obtain the most recent file
            1. Iterate through file name to obtain information regarding the date /time it was made.
    b. If no files were found, print error.
2. Establish a connection with the server
    a. If connection was made, print message
    b. Else, program does not continue
3. Perform handshake
    a. Send file information to the server
        i. If “OK” reply is received, send JSON data
        ii. If no reply was received or another reply was received, print an error
4. Send JSON data
    a. If all the data was sent, print confirmation
    b. Else, the program does not continue
5. Wait for confirmation
    a. If confirmation reply was “SUCCESS”, the operation was completed
    b. If confirmation reply was “CHECKSUM_FAILED”, an error occurred in the sent file’s data
    c. Else, an unknown error occurred.

**Server.py**

1. Wait for a connection from Client.py
    a. If connection was accepted, print a confirmation
2. Perform handshake
    a. If the sent information was received, send an “OK” reply
        i. Display the file information
3. Receive JSON file
    a. Verify that the file information is accurate.
4. Send confirmation
    a. If information is accurate, send a “SUCCESS” message.

b. If an error in the file data was detected, send a “CHECKSUM_ERROR” message.
c. Else, send an unknown error message.

# Each Member’s Role

---

We agreed to divide each task in the programming process so that everyone can focus on a specific aspect based on their abilities.

**Eric Russon**
For this task, Eric was placed in charge of completing the business document and importing the pseudocode. He was informed that he needed to ensure extra details were placed within the solution to aid in explaining the functions shown by the pseudocode, such as the handshake communication and how the server will handle errors.

**Maryam Bunama**
Maryam worked with Roman to finalize the demonstration code and add comments. She also ensured communication between the group members and created the Github repository.

**Roman Kapitoulski**
Roman worked with Maryam to finalize the demonstration code and add comments. His main contribution was the algorithm to sort through the JSON file list and select the newest file for the client to send.

# Brief Summary

---

This was a major collaboration between all group members. With each of us learning how to create a network connection via python, as well as negotiating the minor information such as the separator and end of data message. Thanks to our hard work, we are able to create 3 scripts that not only collaborate with each other, but can also be used on multiple machines.