SYSTEM INFORMATION RETRIEVAL USING MODULES

Project Overview

This project involves retrieving system information from a remote machine, processing the data, and then saving it in a Word document. The data includes details about the platform, CPU, memory, and network. The information is stored in a MySQL database and then exported into a report format for easy access and review.

Key Features and Steps

1. **Remote System Information Retrieval**

   - The project uses **paramiko** to SSH into a remote machine.
   - The script uploads and executes a Python file (mytestnew.py) on the remote machine.
   - The remote machine provides system data such as platform info, CPU info, memory info, and network info.

2. **Database Interaction**

   - The script inserts the fetched system data into a MySQL database (systeminfo).

   - The data is saved in four different tables:

     - platforminfo: Information like version, system, processor, architecture.
     - cpuinfo: CPU core information, frequencies, and CPU usage.
     - memoryinfo: Memory details such as file type, total memory, available memory, usage percentage.
     - networkinfo: Network details like hostname, IP address, bytes sent/received.

3. **Data Export**

   - The data from the MySQL database is fetched using SQL queries.
   - This data is inserted into a **Word document** using the python-docx library.
   - The report is saved with a timestamp, ensuring each report is unique.

4. **Automation and Reporting**

   - The script fetches and exports the system information automatically, creating a structured report.
   - The report includes a table for each type of system information (platform, CPU, memory, and network), making it easy to read and analyze.

Technology Stack

- **Python**: Main language used for interacting with the remote machine, MySQL, and generating the Word report.
- **paramiko**: For SSH connection and executing remote commands.
- **mysql.connector**: To interact with the MySQL database.
- **python-docx**: To generate the Word document.
- **MySQL**: To store system information.

Flow of the Program

1. The script establishes an SSH connection with the remote machine.
2. A Python script is uploaded and executed on the remote machine, which collects system data.
3. The fetched data is inserted into a MySQL database.

4. The script retrieves the stored data from the database and inserts it into a Word document.
5. The Word document is saved on the local machine with a timestamp.

Use Case

- **System Administrators**: Can use this tool to collect and archive system performance data regularly.
- **IT Support**: Useful for generating reports that help in diagnosing system health over time.
- **Developers**: Can automate the collection of system data and store it for analysis or debugging.

This approach combines the power of Python with remote system management, database interaction, and report generation, providing a comprehensive and automated solution for system information tracking.

---

1. OS Module

- **Functionality**:

  The os module allows interaction with the operating system, enabling tasks like file manipulation and system information retrieval.

- **Common Functions**:

  - **utime(path, times)**: Set the access and modification times of a file specified by the path.
  - **getcwd()**: Returns the current working directory.
  - **cpu_count()**: Retrieves the number of CPUs in the system.

- **getpid()**: Returns the current process ID.
- **getppid()**: Returns the parent process ID.
- **environ()**: Returns the environment variables of the system.
- **listdir(path)**: Lists all files and directories in a specified directory.
- **rename(src, dst)**: Renames a file or directory from src to dst.
- **remove(path)**: Removes the file at the specified path.
- **makedirs(path)**: Creates a directory, including any intermediate directories.

- **Use Case**:

  The os module is frequently used for system-level operations like file handling, process management, and interacting with the underlying operating system.

---

2. Platform Module

- **Functionality**:

  The platform module provides detailed information about the system, such as OS details, hardware specifications, and Python version.
- **Common Functions**:

  - **version()**: Retrieves the version of the current platform (OS).
  - **system()**: Returns the name of the operating system-dependent module (e.g., Linux, Windows).
  - **release()**: Returns the system's release version.

- **uname()**: Returns a tuple containing system information.
- **node()**: Returns the machine name on which the Python script is running.
- **machine()**: Returns the machine type (e.g., x86_64).
- **processor()**: Retrieves information about the processor.
- **architecture()**: Returns a tuple with the architecture information of the system (e.g., 64-bit or 32-bit).
- **libc_version()**: Retrieves the version of the C library used by the platform.
- **python_version()**: Returns the Python version running on the system.

- **Use Case**:

  The platform module is useful for retrieving platform-specific information, especially when building cross-platform applications that require conditional behavior based on the underlying OS.

---

3. Psutil Module

- **Functionality**:

  The psutil module (Platform System and Process Utilities) is used for retrieving system and process-related information.
- **Common Functions**:

  - **physical_cores()**: Retrieves the number of physical cores in the system.

- **max_frequency()**: Gets the maximum frequency of the CPU.
- **min_frequency()**: Gets the minimum frequency of the CPU.
- **cpu_usage()**: Retrieves the current usage of the CPU in percentage.
- **cpu_total()**: Retrieves the total CPU time.
- **disk_partitions()**: Retrieves information about disk partitions (e.g., mounted file systems).
- **virtual_memory()**: Returns memory usage details, including total, available, used, and percentage.
- **disk_io_counters()**: Provides disk input/output statistics like read and write speed.
- **cpu_percent()**: Retrieves the current CPU utilization percentage.
- **disk_usage()**: Retrieves the disk usage statistics for a specific path or partition.

- **Use Case**:

  This module is widely used for system monitoring, particularly in performance optimization tools, server monitoring, and system health checkers.

---

4. Socket Module

- **Functionality**:

  The socket module provides the ability to create network connections and perform operations like sending and receiving data over TCP/IP, UDP, and other protocols.

- **Common Functions**:
    - **socket.gethostbyname(hostname)**: Converts a hostname to an IP address.
    - **socket.gethostbyaddr(ip)**: Converts an IP address to a hostname.
    - **socket.gethostname()**: Returns the local machine's hostname.
    - **socket.connect(address)**: Establishes a connection to the specified address (IP, port).
    - **socket.send(data)**: Sends data over a socket.
    - **socket.receive()**: Receives data over a socket.
    - **socket.socket()**: Creates a new socket object.
    - **socket.bind(address)**: Binds a socket to an address (IP and port).
    - **socket.listen(backlog)**: Prepares the socket to accept connections, with a specified backlog.
    - **socket.accept()**: Accepts a connection from a client.
- **Use Case**:

  The socket module is essential for network programming, enabling the creation of client-server applications, handling data transfers, and implementing various communication protocols.

---

5. Paramiko Library

- **Functionality**:

  The paramiko library is a Python implementation used to handle SSH

(Secure Shell) and SFTP (Secure File Transfer Protocol) connections. It facilitates secure connections to remote machines, enabling file transfers and remote command execution.

- **Common Functions**:
  - **paramiko.SSHClient()**: Creates an SSH client to interact with a remote system over a secure connection.
  - **client.set_missing_host_key_policy(paramiko.AutoAddPolicy())**: Sets the policy to automatically accept unknown host keys, bypassing the need for manual verification.
  - **client.connect()**: Establishes an SSH connection to the specified remote host using credentials.
  - **client.exec_command()**: Executes a command on the remote host via SSH.
  - **client.open_sftp()**: Opens an SFTP session for secure file transfers.
  - **dest_sftp.put()**: Uploads a local file to the remote machine.
  - **dest_sftp.get()**: Downloads a file from the remote machine.
  - **client.close()**: Closes the SSH client connection.
  - **dest_sftp.close()**: Closes the SFTP session.
  - **client.get_transport()**: Returns the transport layer for the SSH session.

- **Use Case**:

  paramiko is used to securely connect to a remote machine, execute

commands, and transfer files, often in applications where remote administration or file management is required.

---

6. ast (Abstract Syntax Trees) Module

- **Functionality**:

  The ast module allows you to parse, process, and evaluate Python expressions safely. The ast.literal_eval() function evaluates a string containing a Python literal expression (e.g., dictionaries, lists, tuples) and safely converts it into the corresponding Python object.

- **Common Functions**:

  - **ast.literal_eval()**: Safely evaluates a string containing a literal expression and returns the corresponding Python object.
  - **ast.parse()**: Parses a string into an AST.
  - **ast.dump()**: Returns a string representation of an AST.
  - **ast.walk()**: Iterates over all nodes in the AST.
  - **ast.NodeVisitor()**: A base class to visit all nodes in an AST.
  - **ast.NodeTransformer()**: A base class to transform an AST by modifying its nodes.
  - **ast.increment_lineno()**: Increments the line number of nodes.
  - **ast.get_docstring()**: Retrieves the docstring from a node.

- **Use Case**:

  ast is used when working with dynamic Python code execution, ensuring

safe evaluation of Python expressions and facilitating code analysis or transformation.

---

7. mysql.connector Library

- **Functionality**:

  The mysql.connector library is used to interact with MySQL databases. It provides a set of functions to establish connections, execute SQL queries, and manage transactions.

- **Common Functions**:

  - **mysql.connector.connect()**: Establishes a connection to a MySQL database using specified host, username, password, and database.
  - **connection.cursor()**: Creates a cursor object that allows SQL queries to be executed.
  - **cursor.execute()**: Executes an SQL query on the database.
  - **connection.commit()**: Commits the transaction, making changes permanent in the database.
  - **connection.close()**: Closes the connection to the database.
  - **cursor.close()**: Closes the cursor, releasing associated resources.
  - **cursor.fetchall()**: Retrieves all rows from the result set of a query.
  - **cursor.fetchone()**: Retrieves a single row from the result set.
  - **connection.is_connected()**: Checks if the database connection is active.

- **connection.reconnect()**: Reconnects to the database if the connection was lost.

- **Use Case**:

  mysql.connector is used to interact with MySQL databases for tasks like executing queries, retrieving data, and managing transactions within database-driven applications.

---

8. Error (from mysql.connector)

- **Functionality**:

  The mysql.connector.Error class is used to handle exceptions related to database operations. It helps in managing and debugging MySQL-specific issues like connection errors and query issues.

- **Common Functions**:

  - **Error.code**: Provides the MySQL error code that occurred.
  - **Error.msg**: Retrieves the error message.
  - **Error.__str__()**: Returns a string representation of the error message.

- **Use Case**:

  This class is essential for handling errors in MySQL operations, providing more detailed insights into issues encountered while interacting with the database.