Project Description

Abstract:

The aim of this project is to develop a process monitoring tool that can provide real-time information about the memory and CPU usage of a given process. The tool is developed using C++ programming language and Windows API functions.

The project involves the creation of two classes, namely the Process and ResourceUsage classes. The Process class is responsible for locating and opening a handle to the target process, while the ResourceUsage class provides methods to retrieve and display information about the process's resource usage, such as memory and CPU usage.

The tool works by asking the user to input the name of the process to be monitored, and then locating and opening a handle to that process using the Process class. Once the handle is obtained, the ResourceUsage class is used to retrieve and display information about the process's memory and CPU usage. The tool continuously updates this information every second until it is manually terminated.

The key learning from this project includes gaining an understanding of the Windows API functions, developing an object-oriented programming approach, and gaining experience in working with system-level resources.

One limitation of this tool is that it only works on Windows operating systems. Future scope includes extending the tool's capabilities to work on other operating systems and providing more detailed information about the process's resource usage, such as network usage and disk I/O usage. Additionally, the tool could be integrated with a graphical user interface to provide a more user-friendly experience.

SRS:

The software requirements specification (SRS) for this project is as follows:

1. The program shall allow the user to input the name of the process to be monitored.
2. The program shall locate the process by its name and obtain its process ID and handle.
3. The program shall use the process handle to monitor the process's memory usage and CPU usage.
4. The program shall display the process's memory usage and CPU usage in a human-readable format.
5. The program shall continue to monitor the process until the user terminates it.

HLD/LLD:

The high-level design (HLD) and low-level design (LLD) for this project are as follows:

HLD:
- The main function prompts the user for the name of the process to monitor.

- The `Process` class is responsible for locating the process by its name and obtaining its process ID and handle.
- The `ResourceUsage` class is responsible for monitoring the process's memory usage and CPU usage.
- The main function creates an instance of the `Process` class and obtains the process handle.
- The main function creates an instance of the `ResourceUsage` class and passes it the process handle.
- The main function enters a loop that calls the `print_memory_usage` and `print_cpu_usage` functions of the `ResourceUsage` class once per second.

LLD:
- The `Process` class has a private member `pid` to store the process ID and a private member `handle` to store the process handle.
- The `Process` class has a public constructor that takes the name of the process to be monitored as an argument.
- The `Process` class has a public destructor that closes the process handle.
- The `Process` class has a public member function `open_by_name` that takes the name of the process to be monitored as an argument and returns the process handle.
- The `ResourceUsage` class has a private member `process` to store the process handle, a private member `last_time` to store the last system time, and a private member `last_total_time` to store the last total system time.
- The `ResourceUsage` class has a public constructor that takes the process handle as an argument and initializes the last system time and last total system time to 0.
- The `ResourceUsage` class has a public destructor that closes the process handle.
- The `ResourceUsage` class has a public member function `print_memory_usage` that retrieves and prints the process's memory usage in kilobytes.
- The `ResourceUsage` class has a public member function `print_cpu_usage` that retrieves and prints the process's CPU usage as a percentage.

Object/Class diagram:

```
      +--------+
      | Process|
      +--------+
      | pid    |
      | handle |
      +--------+
         |
         |
         v
+-----------------+
| ResourceUsage   |
+-----------------+
| process         |
| last_time       |
```

| last_total_time |
+------------------+
| print_memory_usage() |
| print_cpu_usage() |
+------------------+

Key learnings:

- How to use Windows API functions to obtain information about a process, such as its process ID, handle, memory usage, and CPU usage.
- How to create a C++ class to encapsulate the process and resource usage monitoring functionality.

Limitations and Future Scope

One of the main limitations of the code is that it only works on Windows operating systems. The use of Windows-specific APIs and functions such as `CreateToolhelp32Snapshot`, `GetProcessMemoryInfo`, and `GetSystemTimes` makes it incompatible with other operating systems.

Another limitation is that the program only monitors one process at a time. If a user wants to monitor multiple processes simultaneously, they would need to run separate instances of the program for each process.

Additionally, the program does not provide any options for configuring the frequency of data collection or the duration of the monitoring process. The program collects resource usage data every second and continues running until it is manually terminated by the user.

Finally, the program does not provide any advanced visualization or analysis tools for the collected data. The output is printed to the console in a simple text format, which can be difficult to interpret and analyze for long-running monitoring sessions or large datasets.

These limitations could be addressed in future iterations of the program, potentially through the development of a more flexible and extensible monitoring framework or the addition of advanced data visualization and analysis tools.