Preservr: A web-based system to control system resource utilization.

Abdul Mannan
University of Central Florida, USA
amannan@knights.ucf.edu

***Abstract*— System resources such as CPU and memory can be monitored using tools such as Activity Monitor on Mac OS and Task Manager on Windows. Preservr extends this monitoring to a web-based application which allows setting limits on utilization per process.**

***Keywords*—CPU, memory, utilization, thresholds, Server-Sent Events.**

I. INTRODUCTION

Major operating systems, such as Mac OS and Windows, are equipped with native applications like Activity Monitor and Task Manager, which grant users the ability to monitor applications and processes concerning their resource utilization. Activity Monitor, for instance, enables users to observe processes and their consumption of resources such as CPU, memory, energy (when operating on a laptop), disk, and network. In Activity Monitor, this information is presented in the form of a table that updates at a predetermined frequency. This table provides sorting and searching capabilities on given features. Users have the option to view more details for individual processes or terminate them as needed. In this paper, I introduce a prototype application designed to expand upon the basic functionalities of these monitoring applications, which I will call 'Preservr' to reflect its ultimate objective.

Preservr is a prototype for a web-based solution that allows users to monitor system resources on their devices while also allowing them to set thresholds that prevent processes from excessively utilizing a particular resource. This innovative application aims to address the limitations of current resource monitoring applications along with additional features to improve user experience and control over processes.

A primary advantage of Preservr is its web-based architecture, which allows users to access the application from any device with an internet connection. This is a significant improvement over native applications, which may be limited by the specific operating system or device they were designed for. By offering a platform-agnostic solution, Preservr broadens the implications of resource monitoring and enables integration with external systems.

Additionally, Preservr's threshold-setting functionality provides users with expanded control over resource utilization, a major enhancement over native applications such as Task Manager and Activity Monitor. This feature allows users to define limits for specific resources, ensuring that processes do not exceed these constraints and negatively impact the overall performance of their system. The application will monitor processes continuously and automatically terminate or throttle them if they surpass the defined thresholds. As a result, this feature not only enhances system stability but also relieves the burden on the user to manually intervene when processes consume excessive resources.

In a full-fledged implementation, the Preservr system would be a N-tiered distributed system. Major components would be Agent, Backend System, and User-Interface (UI). The purpose of Agent would be to run on a target machine, gather necessary information, and terminate processes on demand. It would also transmit the gathered information with the backend system. The purpose of the backend would be mediate communication between UI and Agent along with providing user-management, analysis, security, and so on. The UI would allow a user to login to the Preservr system from any device with internet and use the platform.

Due to time constraints, monitoring and thresholds are implemented only for CPU usage by processes. The backend system is skipped since it would serve no major purpose in a proof-of-concept prototype. Similarly, the login functionality would just add programming overhead and stretch the effort beyond the semester. Therefore, this system will only monitor local CPU usage by processes and allow setting of thresholds and then terminate exceeders as specified by user.

This paper is organized as follows. Section II will present related works and discuss similar existing applications and the distinction of Preservr. Section III describes the system architecture of Preservr. It will dive into the different components, frameworks, and utilities used in the development. Section IV will present an evaluation of the Preservr prototype with a simple benchmarking method. Section V will present future work and implications and Section VI will conclude the paper.

II. RELATED WORKS

Resource monitoring is an important activity for enterprises and individual users since all stakeholders need to operate within allocated budgets. Therefore, it does not surprise me that there are several tenured monitoring applications in the market. Along with Task Manager and Activity Monitor, the following are the most relevant to Preservr:

- Process Explorer [1] is a Windows based powerful desktop application that is essentially Task Manager on steroids. It provides in-depth details for each process. It presents different views depending on the mode the application is running.
- htop (Linux) [2]: htop is an interactive process viewer for Linux systems that offers a user-friendly and feature-rich alternative to the standard 'top' command. It allows users to monitor system resources, including

CPU, memory, and swap usage, as well as manage running processes. htop is also available on Mac OS since it is also UNIX based.

- MenuMeters [3] is a set of CPU, memory, disk, and network monitoring tools for Mac OS that display real-time resource usage information in the menu bar. Users can customize the appearance and layout of the meters to suit their preferences.

There is a plethora of such resources and creating another application that runs on a local system to monitor resources would be analogous to reinventing the wheel. The reason Preservr is useful is that it extends the monitoring to a web-based application that could be accessed from another device. Most of the existing applications do not allow a user to set limits per process. If limits are in place, it is easier to estimate the load on a system. Accurate estimation of load on system resources has its own implications and potential applications. This extension from local to web platform also allows a single user to monitor multiple personal devices. However, the latter functionality was not implemented due to time constraints.

III. SYSTEM ARCHITECTURE

Preservr is a web-based application for monitoring of CPU utilization and act as a sentinel over user processes, so they don't exceed user-specified limit. A full-fledged version of Preservr would have the following architecture:

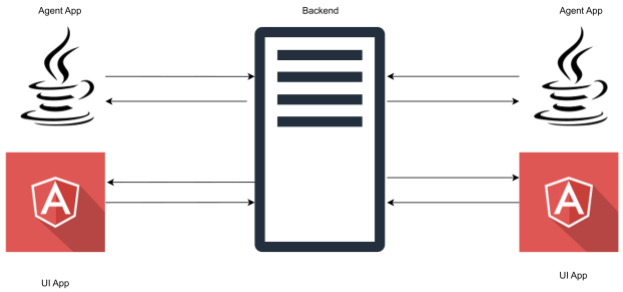

Fig 1. Preservr Target Implementation Diagram

Full implementation of Preservr would have three main components: Agent, UI, and backend system. The purpose of Agent application is to gather resource usage data for processes and terminate processes on request. The backend, as mentioned, would act as coordinator for the Agent instances and UIs for various users of the system. The UI web application would present visualizations and control panels to set thresholds.

The prototype that I implemented, shown in Fig 2, does not include the backend part of the system. The Agent application directly transmits data to the UI and when the user sets thresholds, the UI notifies the Agent if a process needs to be terminated. The Agent then terminates that process. The agent is written in Java 17 with spring boot framework. It uses a top command to gather CPU related data, parses them into a specific format, and periodically transmits it to the UI over network using server-sent events. The UI subscribes to these events and parses the data into a format that can be easily visualized. The UI is written in the latest version of Angular and Angular Material UI components are used.

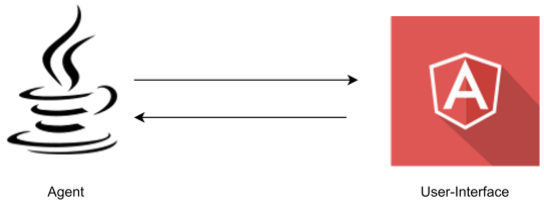

Fig 2. Preservr Prototype Implementation

A doughnut chart with concentric layers gives a visualization of the percentage CPU utilization per process. The doughnut assigns colours to the top ten processes, by percentage for improved distinction. The innermost layer of the doughnut represents the most recent snapshot of CPU utilization. The doughnut presents the last five snapshots with the outermost layer representing the fifth most recent utilization. For creating this real-time chart, I have used Chart.JS [4] library.

There is a section on the UI to allow the user to create a policy for maximum CPU utilization. A slider with a range of 0-100 lets the user to select maximum percentage on CPU utilization. It is not ideal to terminate based on a single snapshot, since there could be a random instantaneous spike for a given application. Therefore, there is a few radio buttons that allow the user to tell the system to monitor utilization over 5, 10, or 15 snapshots before terminating a process that exceeds the percentage specified using the slider. There is an additional group of radio buttons to allow the user to choose what percentage of monitored snapshots should contain the exceeding process. There is a slide-toggle button that allows the user to enable auto-termination. If this is checked, exceeding processes will be automatically terminated. Once the user creates a policy for termination, there is one more slide-toggle button to activate the policy. The system will start enforcing the specified policy once this is turned on.

The Agent application takes snapshots through `top` command and parses the CPU utilization by percentage utilization. Usually, 100s of processes are running on personal devices but only a few are CPU-hungry. The Agent application then transmits each snapshot as a Server-Sent Events (SSE). SSEs are more convenient for this kind of real-time communication. I did consider using a Http Get request but then client must call the API per snapshot every few seconds. Connection initialization is an expensive process, and SSEs are therefore better suited.

Preservr UI continuously receives snapshots from the Agent and then parses into a format that can fit the visualization component, that is a doughnut chart from Chart.js library. The snapshots are saved in a FIFO queue data-structure of a fixed size the corresponds to the number of layers in the doughnut chart. The doughnut component in the UI is refreshed with every event sent by the Agent.

When the auto-termination is turned on, the system will automatically terminate processes on the system where the Agent is running, given they exceed the specified policy which is active. When the policy is not active, the system does not enforce anything. However, the doughnut will continue to show the real-time usage. In this case, the terminated process will be displayed in the corresponding table. If the auto-termination is not enabled, the system marks the processes for manual termination by the system user. When they exceed the active policy, it pushes the process into the 'marked' table. There is an option for the user to terminate process by clicking a button. Once the user terminates a process in 'marked' table, it then moves into the terminated table.

The system will continue to monitor and enforce terminations with respect to the active policy until the activation toggle is turned off. Once it is turned off, the system will no longer enforce the policy. If the user decides to change the policy, they can deactivate the currently active policy and update the components accordingly. They can set a new policy and activate that to let the system enforce the updated policy. When a process is terminated, the system pops up a 'snack bar' notification that notifies the user that a process has been terminated. It is important to mention that this system only terminates the processes that were initiated by the user of the operating system. It does not terminate any system processes or any processes that were not started by the given user.

I wrote the Agent application in Java 17 with Spring Boot framework. Spring Boot is a widely used and popular framework that aims to simplify the development, configuration, and deployment of Spring applications. It uses the Maven build tool to ensure different components can integrate and allow the whole application to function seamlessly. We can define some dependencies in the `pom.xml` file. Main dependencies, apart from native Java 17 libraries, for Agent application were `spring-boot-starter-web` and `spring-boot-starter-websocket`. When the Agent application starts up, it injects some necessary dependencies for the web application. In the Spring Framework, the Application Context is a central component that manages the configuration, instantiation, and wiring of beans (objects) within a Spring application. Once this context is ready, it emits an event that triggers the logic of Agent application. It starts taking periodic snapshots of process usage using `top` command. There is an endpoint within this application that awaits a connection request from the UI application.

The UI application was written with Angular framework's latest version, which is based on Typescript. The learning curve for Angular is a bit steep compared to other popular frameworks such as React and Vue.js. However, it is a full-fledged framework with a wide range of tools, and it is possible to build performant and large applications without needing third-party libraries, since it has native libraries for most of functionalities. Components in Angular represent a logical component of application that corresponds to a specific functionality. It is possible to re-render a specific component as needed without having to refresh the whole webpage. As soon as the UI application starts up, it registers an EventSource [5] with the Agent. This creates a persistent connection between the applications, and the Agent starts transmitting events with the payload of CPU usage.

IV. Evaluation

I will run the Agent and UI applications on my MacBook Pro. It has a 2.3 GHz 8-Core Intel i9 processor, Intel UHD Graphics 630 1536 MB graphics card, and 16 GB 2667 MHz DDR4 memory. The operating system is MacOS Ventura 13.2.1. I will first run the Agent application before executing the UI. Once the Agent application is ready, it shows the following the logs:

```
Tomcat initialized with port(s): 8080 (http)
Starting service [Tomcat]
Starting Servlet engine: [Apache Tomcat/10.1.5]
Initializing Spring embedded WebApplicationContext
Root WebApplicationContext: initialization completed in 1291 ms
LiveReload server is running on port 35729
Tomcat started on port(s): 8080 (http) with context path ''
Started AgentApplication in 2.05 seconds (process running for 2.514)
```

Fig 3. Agent ready state logs

It is important to note that Spring Boot applications have an embedded Tomcat webserver, and it is possible to just run them in the same way as we would run a standalone Java application. The default port 8080 is assigned to the application. The Agent uses a SimpleAsyncTaskExecutor that gathers the periodic data. The task assigned to this thread is to run the top command with specified frequency and pass each output of the command into a service.

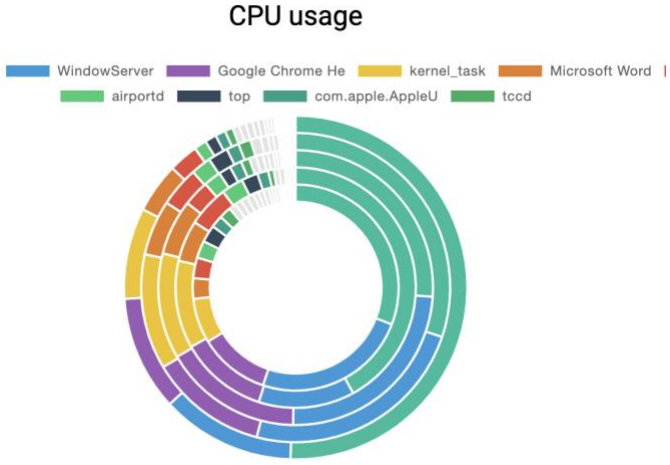

Fig 4. CPU usage doughnut chart

Once the UI application is ready, it will display the above chart. The innermost layer is the most recent snapshot. The doughnut chart is periodically re-rendered when the data-source is updated. When the maximum limit is set to a low number, as shown in the following diagram, the application starts marking regular most of the applications. We may not

want to set this number too low, because many useful processes must continue to run.

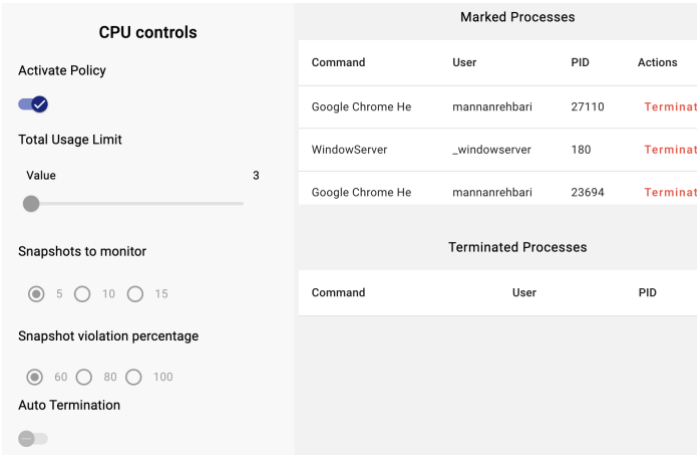

Fig 5. Low Percentage threshold – marks most processes.

On Mac OS, there is utility called `yes` that we can run from terminal. It continues to print 'yes' or a specified text but its basic purpose is to overload the CPU. We can hit ctrl+c to stop the command within terminal. Once the command is run, it can be observed from the doughnut chart that terminal hogs most of the CPU. In the following figure, the application was monitoring the CPU utilization as expected. However, no policy was active.

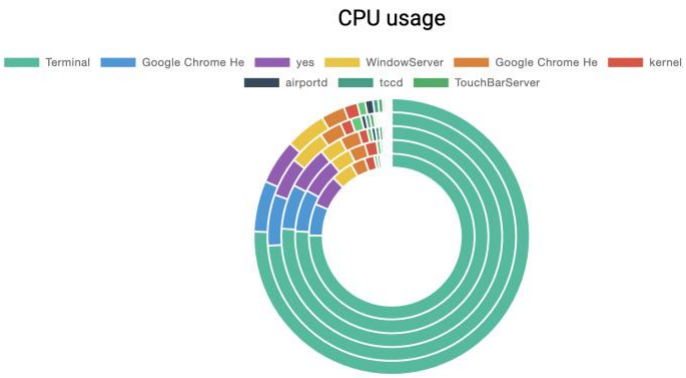

Fig 6. `yes` ran from terminal leads to high CPU utilization

I will re-run the application, this time with an active policy with 60% as maximum limit for CPU usage, window of 5 snapshots to monitor, and snapshot violation percentage as 100%. This means that any process which continuously shows in 5 consecutive snapshots with over 60% CPU utilization needs to be terminated. I will turn on the auto-termination policy. The system detected the high usage of CPU by the `yes` command that ran from terminal and terminates the terminal. A result of this is shown in the following figure.

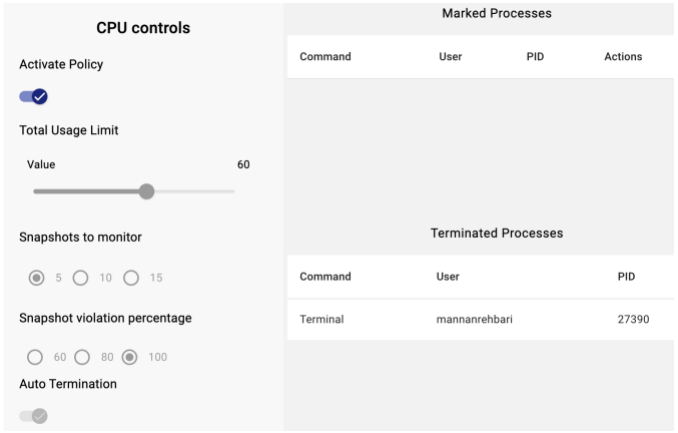

Fig 7. terminal terminated.

V. Future Work and Implications

The prototype implemented has a minimum feature of monitoring CPU utilization and capability to terminate policy violating user processes. Given the effort was to be continued after the end of the semester, the following potential features could be added to the system:

- User authentication – arguably the most important feature any web-based application. It would enable users to login and monitor their own resources.
- Multi-device monitoring – extended Preservr could let users register and monitor multiple devices from the same interface. However, each device would have to run the Agent application in the background.
- Memory, network, and other resources could also be monitored.

Monitoring of resources has additional implications. If we can monitor and predict the load on our system resources, it allows us to expand the capability of the system in many ways. One of the ways is to `rent-out` a specific threshold of our resources to run self-contained jobs on our system. The jobs, without violating target system security and integrity, could achieve a business purpose for a third party. A specific use-case could be workloads where it is possible to use a 'divide-and-conquer' strategy on a large dataset that can be divided into independent smaller sets. A job for Preservr could contain a single or a few smaller subsets of the data. Further research is needed to evaluate the existing software and compare with the functionalities promised by Preservr, the extended version.

I researched different ways to obtain process information on a system within a Java environment. Some relevant native Java APIs such as ProcessHandle and several others within `*java.lang.management*.*` package exist. However, these APIs do not have visibility to the *resource usage* by an individual process running on the system. The prototype, therefore, used `top` command to obtain resource consumption information from the OS. In a future

implementation, it would be ideal to find, adapt, or create a novel technique.

VI. Conclusion

In conclusion, Preservr is a web-based resource monitoring application that allows a user to set maximum usage limits. The prototype application consists of an Agent and a UI application. The former was written in Java 17, and the latter, in Angular 15. Angular Material components are used along with Chart.js library. The Agent and UI communicate over HTTP using Server-Sent Events. The UI presents a doughnut chart with concentric layers for CPU utilization for processes. Each layer corresponds to a snapshot of CPU utilization on the system.

A control panel allows the user to set a maximum limit for CPU for a given process. When the policy is activated, the system monitors the usage patterns and terminates the user processes that violate the specified policy. Preservr will, however, not terminate the processes that are system processes or ones that are not started by the user. If auto-termination is turned on, the system will kill the processes, otherwise, it will mark them for the user to terminate manually.

The primary goal of this application is to allow monitoring and setting limits on system resources. However, its nature allows potential expansion into a full-fledged distributed system for resource monitoring with the implication of `renting-out` resources.

References

[1] "Process Explorer," Microsoft, [Online]. Available: https://learn.microsoft.com/en-us/sysinternals/downloads/process-explorer. [Accessed: Apr. 19, 2023].

[2] [2] "htop," GitHub, [Online]. Available: https://github.com/htop-dev/htop. [Accessed: Apr. 19, 2023].

[3] Y. Tachikawa, "MenuMeters for El Capitan," [Online]. Available: https://member.ipmu.jp/yuji.tachikawa/MenuMetersElCapitan/. [Accessed: Apr. 20, 2023].

[4] "Chart.js," Chart.js, [Online]. Available: https://www.chartjs.org/docs/latest/. [Accessed: Apr. 20, 2023].

[5] "EventSource," Mozilla Developer Network, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/EventSource. [Accessed: Apr. 20, 2023].