Milestone 1 - Problem Statement and Methodology

Problem Statement

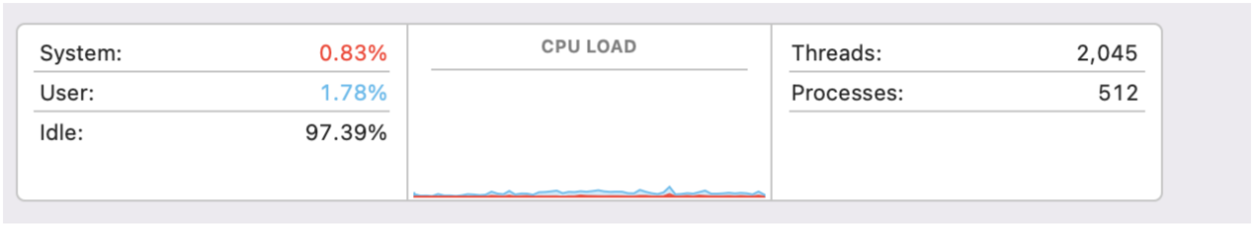
Resource monitoring applications are part of most modern Operating Systems. On Mac OS, it is known as Activity Monitor and on Windows platforms it is called Task Manager. These existing applications allow users to monitor the consumption of CPU, Memory, Disk, and Network by each process. These applications also allow actions such as forced termination of a given process. However, these applications are not very interactive.

I propose a web-based solution which I will call Preservr. The basic features of Preservr will be the following:

- Preservr will allow a user to view all processes
- Preservr will allow a user to terminate a given process
- Preservr will allow a user to set maximum thresholds on resource utilization
- Preservr will allow a user to set *automated* actions in case thresholds are exceeded

There are a few implications of moving the resource monitoring from local system to a web-based environment. Having access to a system's resources *remotely* allows monitoring from another physical location. Also, a user can monitor *multiple systems* from a single web application.
There is an additional implication of local to web migration of resource monitoring. The following is a snapshot of Activity Monitor from my laptop:

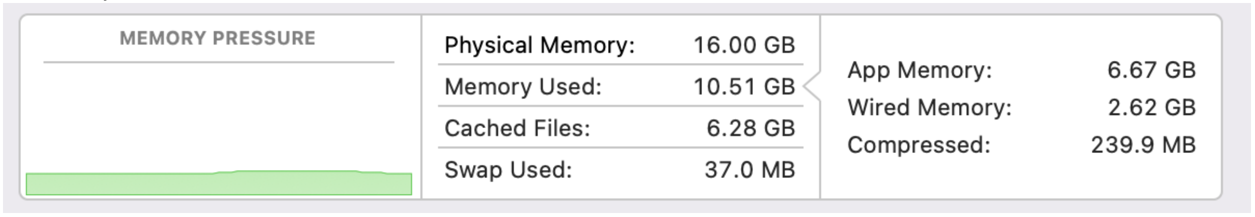
CPU:



System:	0.83%
User:	1.78%
Idle:	97.39%

Threads:	2,045
Processes:	512

Memory:



Physical Memory:	16.00 GB
Memory Used:	10.51 GB
Cached Files:	6.28 GB
Swap Used:	37.0 MB

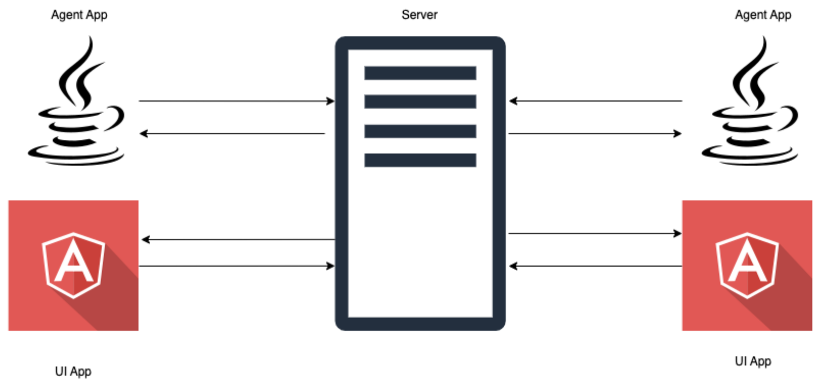
App Memory:	6.67 GB
Wired Memory:	2.62 GB
Compressed:	239.9 MB

At the time of these snapshots, I had many applications running concurrently. It shows that the total available resources are commonly underutilized. There is a possibility to extend basic Preservr system where users will be able to 'rent out' resources such as CPU or collaboratively use a cluster of systems with friends or team. However, I do not intend on implementing this whole distributed system during the semester, and the functionality of Preservr application will be limited to the features listed above.

Methodology

Preservr application will have three components: Agent, UI and Server.

1) Agent – This application will obtain the necessary information from a target system, pass the information to the server over network, and execute actions on the target system when required.
2) API server – This will essentially act as a coordinator between the agent and UI application.
3) UI – This will be a single-page web-application that will allow the user to access the processes running on their target system



Basic Architecture Diagram of Preservr

Building the Preservr application modules will require no direct external hardware, except the development laptop. The Agent module and the Server module will be written with Java technologies including Maven and Spring Boot. I plan on using Java 17 since it is the latest version with long-term support. This will allow me to upgrade my knowledge from Java8 to a more recent version.

The UI application will be entirely built with TypeScript/Angular, with bare minimum focus on styles and UI/UX. Some of the APIs I have tinkered with at this point are ProcessAPI and Runtime, both are in *java.lang.** package. I *may* have to run some shell commands from the Agent app, but ideally will try to avoid that.

I am not sure about any explicit use of any algorithms at this point in the process. However, if a *subset* of processes from a list needs to be terminated due to set thresholds, there must be a policy to determine which ones will have to be terminated first. I will present more details in later milestones.

The per-process data will be gathered and analyzed on different modules of the Preservr application. Some of the information will be relevant to the operating system itself and the computer's physical and virtual resources.

I will be using a Kanban board for tracking the features for this project and will implement them in a logical sequence.