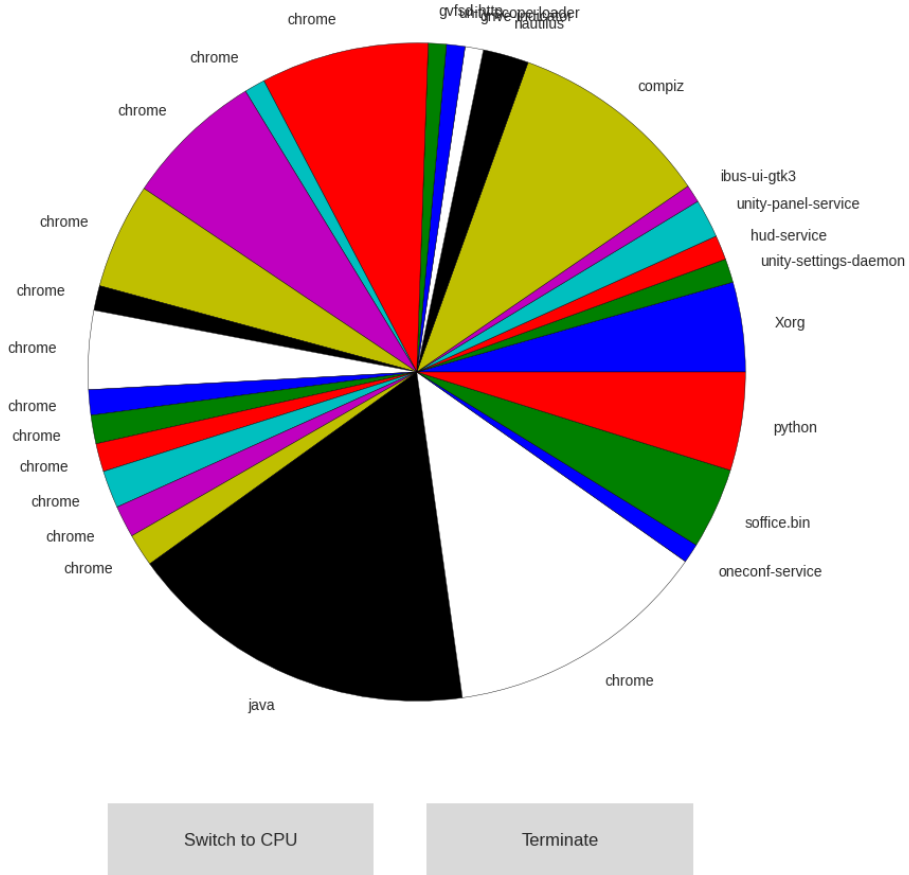
David Abrahams & Tom Heale

Project Overview:

We created a simple visual task manager for Linux in python. The user can view processes' CPU and RAM usage, and terminate selected processes. The task manager updates in real time.

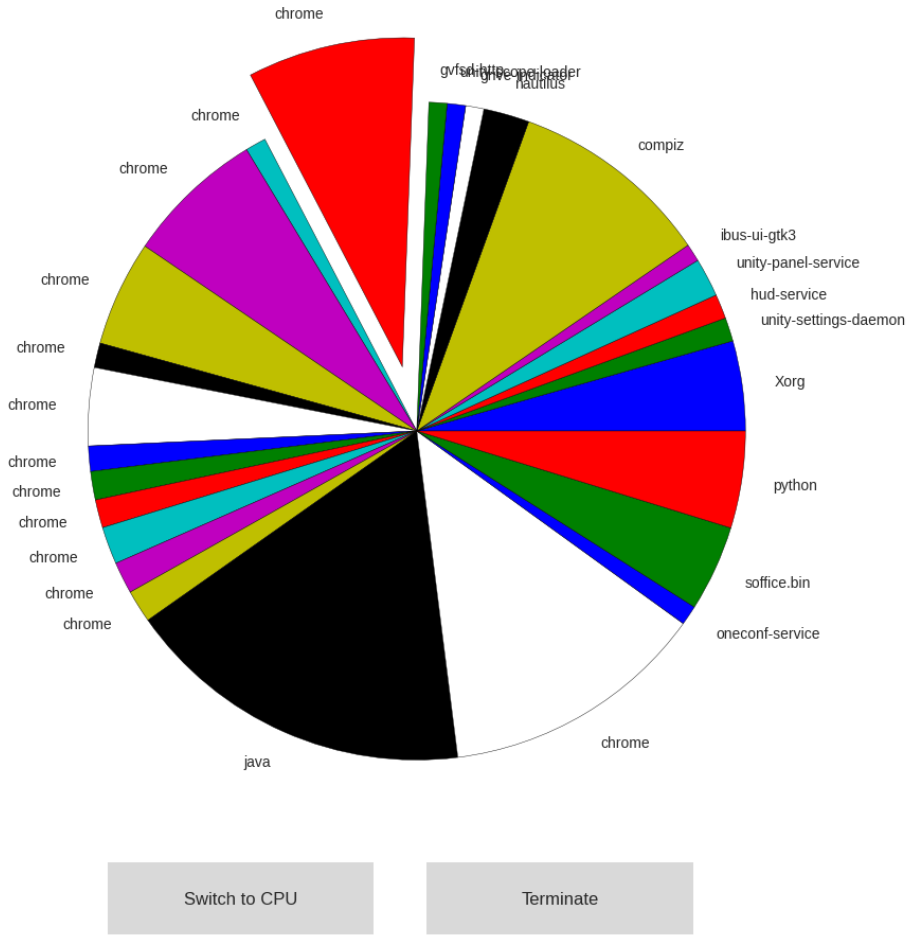
Results:

Our program provides similar functionality to Windows Task Manager. A pie chart allows the user to view which processes are the most “heavy” on the computer. Below are all the processes my computer is currently running:

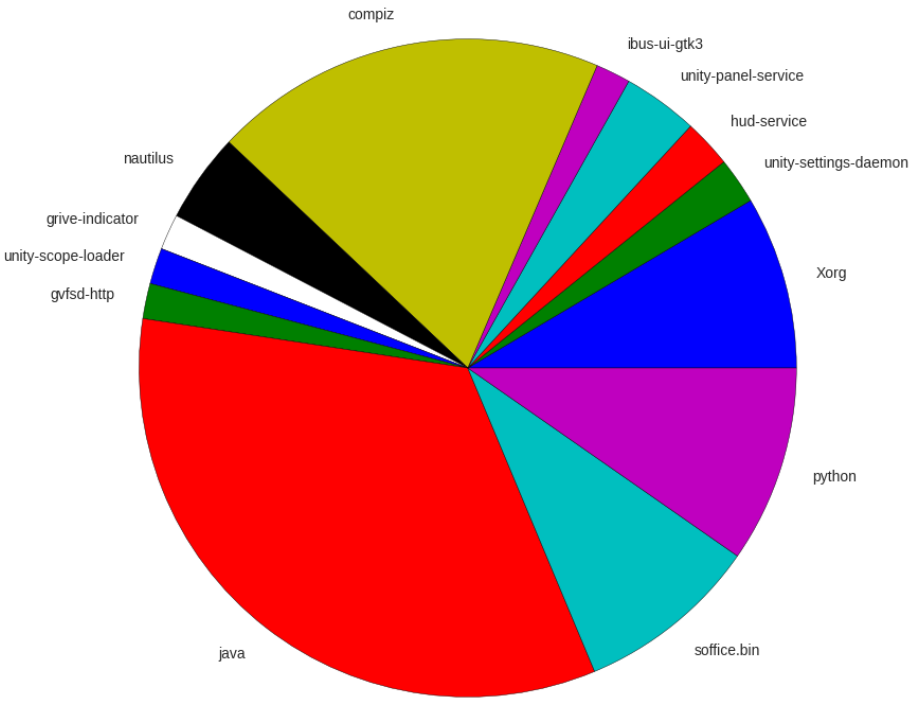


If I wanted to close one of my chrome processes, I would simply click on it in the pie chart, and that process would be selected:

Then, to close the processes, I would click terminate, and the process would end. In this case, closing the selected chrome process caused chrome to crash and end all chrome processes:

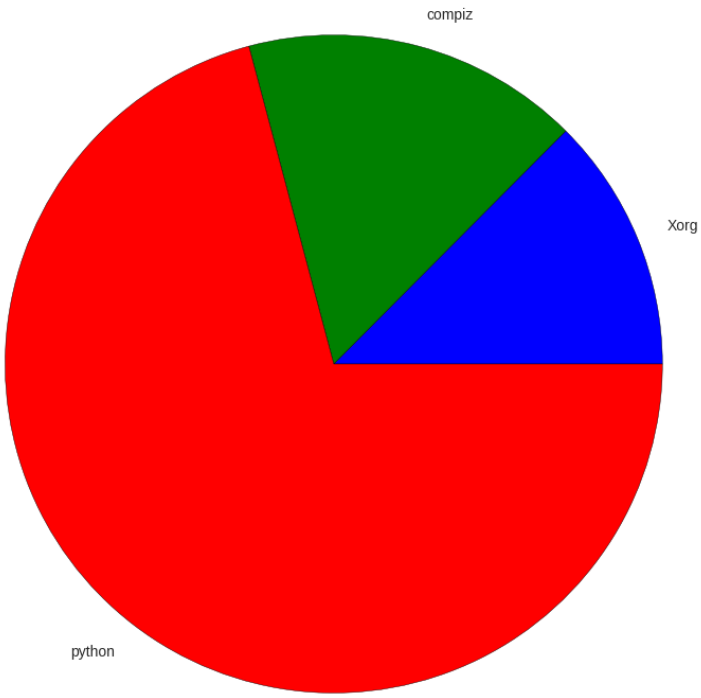


The pie chart updates real time, so if you open or close processes while viewing the window, the wedges in the pie chart will change size according to RAM/CPU usage. Finally, the user can click “Switch to CPU” if they would like to view CPU usage. However, this is less useful because the ProcessManager itself is very processor intensive, as it is constantly updating and checking which processes are running. Therefore, generally the ProcessManager will be using most of the CPU:



Switch to CPU

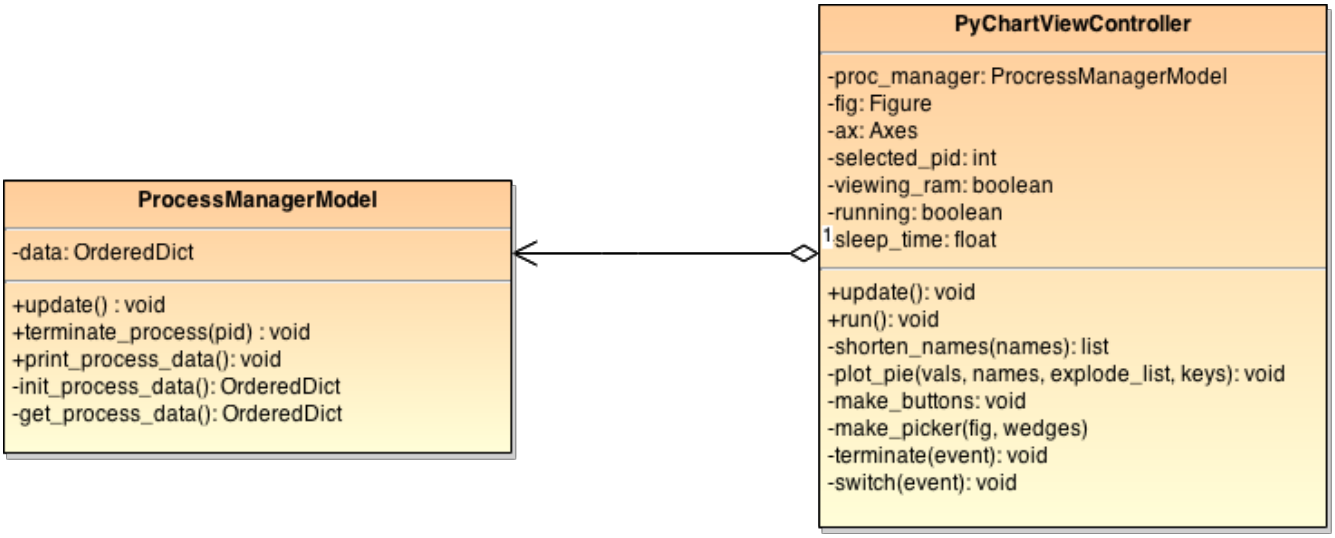
Terminate



Switch to RAM

Terminate

Implementation:



Our program is broken up into two separate classes, ProcessManagerModel. and PyChartViewController.

ProcessManagerModel is a class that can both access information about the computer's process and terminate them. ProcessManagerModel makes heavy use of the psutil library to access and control current processes. The only functions outside classes must use are update() and terminate_process(). Outside classes can then access data, an OrderedDict that maps from {pid:(process_name, cpu_percent, ram_usage_mb)}. update() works by first removing the processes from data that are no longer running. It then adds all new processes on to the end of data. terminate_process(pid) uses psutil to delete the process of a given pid.

PyChartViewController is a class that uses matplotlib.pyplot to display current information and allow the user to use the "Controller." PyChartViewController contains a ProcessManagerModel object that is uses to access and control data. PyChartViewController runs a function update() in a loop, which calls update() on the ProcessManagerModel, and then replots the information on the screen. When the user clicks on parts of the pie chart, the PyChartViewController uses a dictionary it creates to map from the Wedge object to a process pid. When the user clicks terminate, PyChartViewController calls terminate_process(pid) on its ProcessManagerModel.

An interesting design decision we made was making ProcessManagerModel contain an OrderedDict rather than a dict. This is because we knew we would be plotting the processes in a pie chart, and we wanted the order to stay the same on each loop iteration. Because dicts are unordered, the processes could jump around the pie chart with each loop iteration if we used a regular dict.

Reflection:

Our project went very well. Our biggest challenge was trying to implement bokeh to make our GUI. We circumvented that issue with matplotlib, which worked very well after some experimentation with redrawing figures. We scoped the project well because we found the psutil package before committing to the project, thus we knew we could get our data and just had to finds ways to encapsulate and display it. We enjoyed doing this project and feel like we had all the tools we needed to bring it to fruition. I, Tom, personally learned a lot about OOP that I will try to carry over to other projects. David knows Java, so he is better acquainted with objects and their uses.

We didn't make a clear plan for how we would work at the beginning. When we were first defining our classes and outlining our project, there was a lot of pair-programming and discussion. As the project developed and we started receiving output from the program, we did less side-by-side work and utilized git version control more. Generally, we would meet to decide which feature was most important to add next, then pull, work, and push at our leisure. Conversations would often follow the format: "David, I made the plot an actual figure, and now we can interact with it without blowing it up!" "That's great! Did you push?" "Yup." "Cool, I've been thinking about ways to make the slices pop out when you click them. I'll pull and do that later." "Cool, have fun." We learned so much about git and version control! We worked well as teammates overall.