El-Reqaba: A Linux Process Manager

Project Overview

Developed by:
Aly Elaswad
Ismail Sabry
Omar Ganna

Introduction

This Report provides a detailed manual for our Linux Task Manager “El-Reqaba”. “El-Reqaba” is a process manager that is built using Rust and Electron JS and available in two interfaces: Tabular User Interface (TUI) and Graphical User Interface (GUI). Our Process manager is designed to effectively monitor and manage running system processes. Linux users are divided into two sections, some desire an intuitive, accessible, easy to use interface and others are more command line enthusiasts who prefer lightweight tools that provide detailed insights without leaving the terminal. “El-Reqaba” is designed to bridge this gap by providing both a robust TUI version for terminal-based users and a user-friendly GUI for those looking for better visuals and system graphs. Both versions offer real-time process monitoring, resource tracking, and process management capabilities. “El-Reqaba” represents a good evolution in linux process management tools combining both the traditional approaches with some newly added features aiming to provide users with better control and performance optimization.

Structure

Our process manager is structured into three main components:

- Command Line Interface (CLI)
- Tabular User Interface (TUI)
- Graphical User Interface (GUI)

Features

1. **Command Line Interface**
   The command line interface is considered the root of our project, since it calls the other two components, these are all the features developed for the CLI as commands
   - **get_os** – Displays the current operating system information.
   - **ptable [file.csv]** – Shows the current process table; optionally exports it to a CSV file.
   - **change_nice <pid> <niceness>** – Changes the niceness (priority) of the specified process.
   - **kill <pid>** – Terminates the specified process.
   - **log <pid>** – Monitors and logs the status changes of a specific process in real-time.
   - **pause <pid>** – Suspends the execution of a process.
   - **resume <pid>** – Resumes a previously paused process.
   - **track_process <pid> <output.csv> <duration_secs>** – Tracks CPU and memory usage of a process for a set duration and writes the results to CSV.
   - **get_process_command <pid>** – Displays the full command that was used to launch the specified process.
   - **restart_if_failed <pid>** – Monitors a process and automatically restarts it if it crashes or exits.
   - **tui** – Launches the Terminal User Interface (TUI) for interactive, real-time process monitoring.
   - **gui** – Launches the Electron-based Graphical User Interface (GUI).

2. **Tabular User Interface**
   The Terminal User Interface (TUI) component of the process management system provides an interactive, real-time monitoring solution that balances the simplicity of command-line tools with the visual advantages of a graphical interface. The TUI is implemented using the Cursive library, which enables rich text formatting, keyboard shortcuts, and responsive layouts within the terminal environment.

   - **Real-time Process Monitoring:** The TUI continuously updates the process table to reflect the current state of all processes in the system. Users can see CPU usage, memory consumption, process status, and other vital metrics refreshed every second.

This update can be toggled on/off with the 'u' key to allow for focused analysis of specific data points

- **Interactive Process Table:**The central element of the TUI is a sortable, scrollable table that displays comprehensive process information. Users can:
    - ❖ Sort by any column (PID, CPU usage, memory, etc.) by clicking column headers
    - ❖ Select processes for detailed information or actions

- **System Information Dashboard:** A system information bar at the top of the interface displays critical system metrics:
    - ❖ CPU name and current frequency
    - ❖ Overall CPU usage percentage
    - ❖ Memory usage statistics (total, used, available)
    - ❖ System uptime and total process count
    - ❖ Core count (physical and logical)

- **Process Management Actions:**The TUI provides keyboard shortcuts for common process management tasks:
    - ❖ Kill a process ('k')
    - ❖ Pause a process ('p') using SIGSTOP
    - ❖ Resume a process ('r') using SIGCONT
    - ❖ Change process priority ('n') with nice value adjustment
- **Process Tree Visualization:** Users can toggle a hierarchical view of processes with the 't' key, showing parent-child relationships between processes. This tree view helps understand process dependencies and identify process groups
- **Process Filtering:** The filtering system ('f' key) allows users to narrow down the process list based on:
    - ❖ PID
    - ❖ PPID (parent process ID)
    - ❖ User
    - ❖ Process status
- **Priority Management:** The TUI includes the ability to change the process priorities using the nice value.

3. Graphical user interface

The Electron-based Graphical User Interface (GUI) includes rich visual elements and interactive controls, allowing users to efficiently track and manipulate processes without using the terminal. This interface is especially suitable for users who prefer graphical environments over command-line interactions.

- **Real-time Process Monitoring:** The interface continuously updates to reflect the current state of all processes, including CPU and memory usage. This can be paused though through a button.
- **Process Grouping:** Processes can be grouped by name or PPID to help users identify related tasks and manage them collectively.
- **Historical Process Tracking:** Users can view historical metrics and trends for individual processes, aiding in diagnostics and performance analysis. This is done in the graphs sections
- **CPU and Memory Usage Graphs:** Visual graphs display system-wide resource consumption over time, offering a quick overview of performance.
- **Tracking Certain Processes Graphs:** Same as the Graphs section however it can track certain processes only.
- **Process Actions:** Users can kill, pause, resume, or change the priority (niceness) of processes through the interface.
- **Dark/Light Theme Support:** The GUI provides customizable themes to enhance visual comfort in different environments.
- **Idle Process Detection:** The GUI integrates intelligent idle detection to identify processes that are no longer active but still consuming resources.
- **Group Actions:** Multiple processes can be selected and managed together, allowing batch operations like mass termination or priority adjustment. This group can also be used in the graph analytics.
- **Focus Mode:** A specialized view to prioritize a certain group of processes.

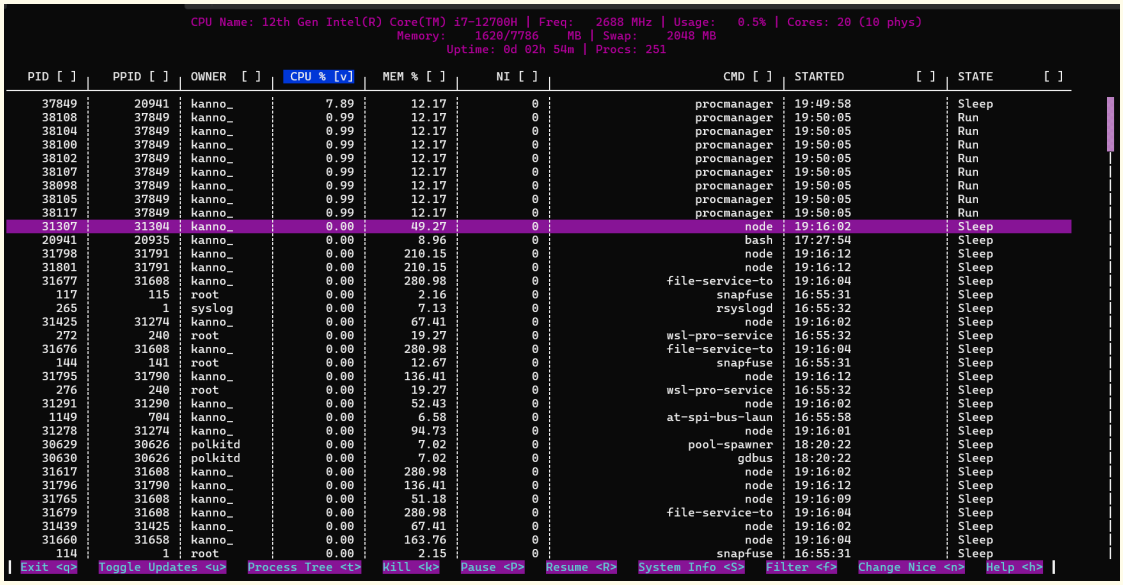
Initial Project Proposal vs. Final Outcome

In our initial report, we proposed a Linux Process Manager with many core features and requirements, including real-time monitoring, resource optimization, fault tolerance, and control over process states. We also outlined a set of functional requirements such as multiple interfaces (GUI, TUI), smart scheduling, and advanced process handling capabilities.

We are happy to report that all the promised components have been addressed and implemented to a significant extent. Each feature was integrated through one or more of our three interfaces—Command Line Interface (CLI), Terminal User Interface (TUI), and Graphical User Interface (GUI)—offering flexibility and depth in process management.

Below is a breakdown of each initially proposed feature, with implementation details to follow:

Real-Time Monitoring:
In the tui, When the TUI is launched, users are presented with a comprehensive process table that serves as the central element of the interface. This table displays real-time information about all running processes in the system, including: Process ID (PID), Process name, CPU usage percentage, Memory consumption, Process status (running, sleeping, stopped), User owner, Start time, Priority (nice value)

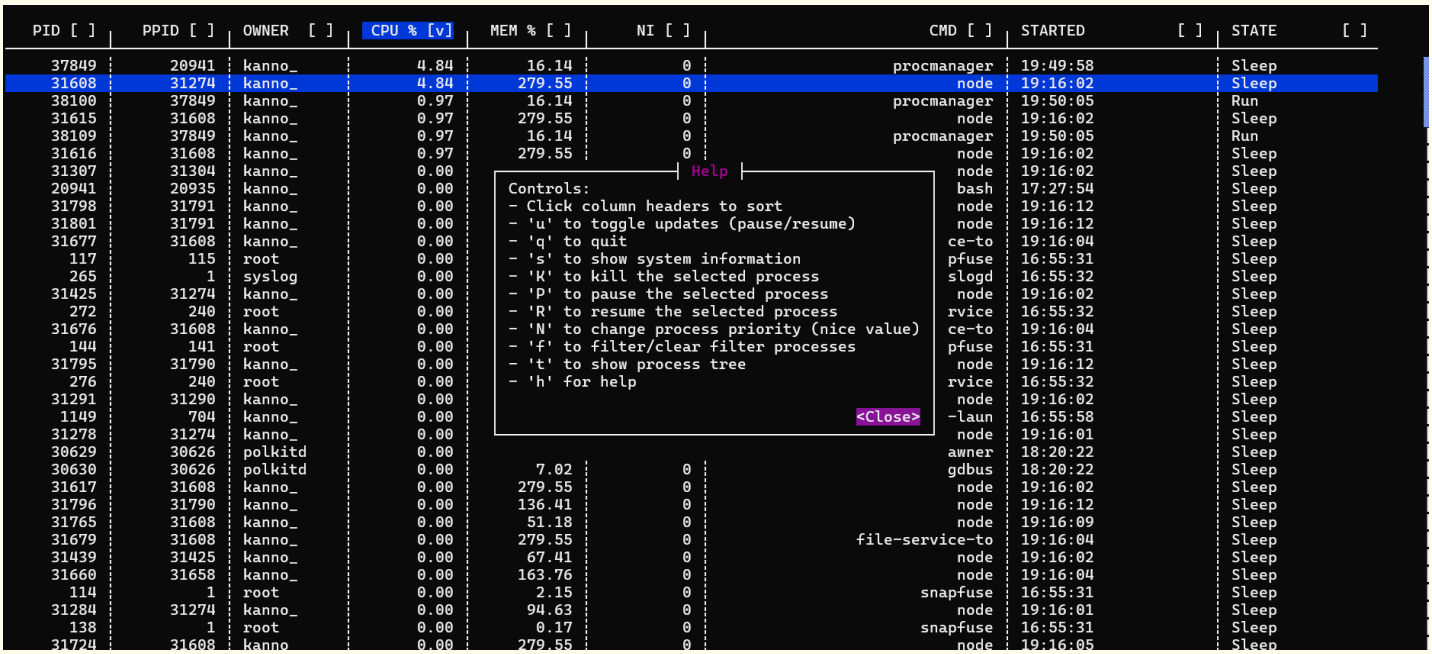


The processes table is automatically refreshed every 2 seconds to provide up-to-date information. Users still have the ability to toggle this auto refresh functionality by clicking a dedicated keyboard shortcut. Allowing them to pause updates.

Interactive Process Management

The TUI provides a rich set of keyboard-driven commands for process management:

- Selection System: Users can navigate through the process list using arrow keys or cursor,, with the currently selected process highlighted for visibility
- Process Actions: For any selected process, users can:
    - Kill the process (terminate immediately)
    - Pause the process (send SIGSTOP)
    - Resume a paused process (send SIGCONT)
    - Change the process priority (adjust nice value)

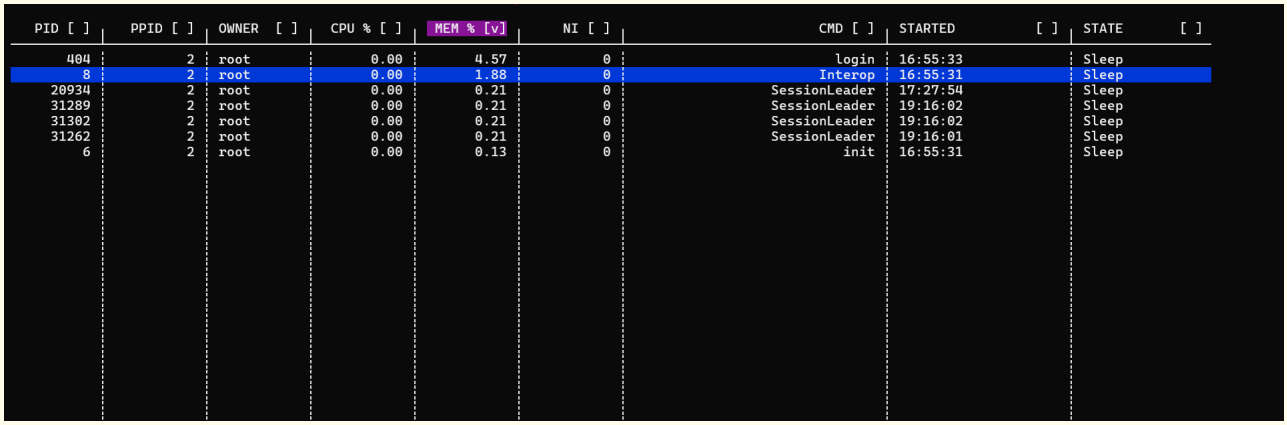


Each action is accessible through intuitive keyboard shortcuts that can be known using the help tab.

Process Filtering and Sorting

The TUI includes powerful filtering capabilities to help users find specific processes:

- Dynamic Filtering: Users can filter the process list by typing search terms, with results updating in real-time
- Multi-criteria Filtering: Filtering can be applied based on process name, PID, user, or status
- Column Sorting: Any column in the process table can be sorted in ascending or descending order by pressing the corresponding key



PID [ ]	PPID [ ]	OWNER [ ]	CPU % [ ]	MEM % [v]	NI [ ]	CMD [ ]	STARTED [ ]	STATE [ ]
404	2	root	0.00	4.57	0	login	16:55:33	Sleep
8	2	root	0.00	1.88	0	Interop	16:55:31	Sleep
20934	2	root	0.00	0.21	0	SessionLeader	17:27:54	Sleep
31289	2	root	0.00	0.21	0	SessionLeader	19:16:02	Sleep
31302	2	root	0.00	0.21	0	SessionLeader	19:16:02	Sleep
31262	2	root	0.00	0.21	0	SessionLeader	19:16:01	Sleep
6	2	root	0.00	0.13	0	init	16:55:31	Sleep

Process Tree Visualization

The TUI includes a process tree view that can be toggled with a keyboard shortcut. This hierarchical display shows the parent-child relationships between processes, helping users understand process dependencies and identify process groups.

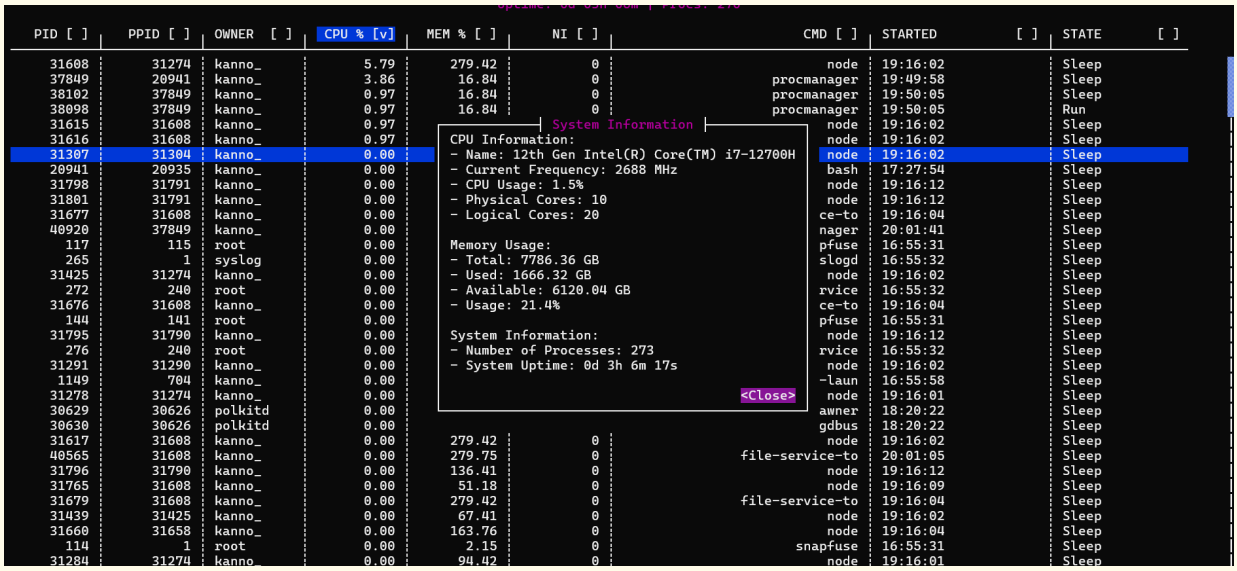
```
└── systemd (1)
    ├── init-systemd(Ub (2)
    │   ├── init (6)
    │   │   └── init (7)
    │   ├── Interop (8)
    │   ├── login (404)
    │   │   └── bash (716)
    │   ├── SessionLeader (20934)
    │   │   └── Relay(20941) (20935)
    │   │       └── bash (20941)
    │   │           └── procmanager (37849)
    │   │               ├── procmanager (38098)
    │   │               ├── procmanager (38099)
    │   │               ├── procmanager (38100)
    │   │               ├── procmanager (38101)
    │   │               ├── procmanager (38102)
    │   │               ├── procmanager (38103)
    │   │               ├── procmanager (38104)
    │   │               ├── procmanager (38105)
    │   │               ├── procmanager (38106)
    │   │               ├── procmanager (38107)
    │   │               ├── procmanager (38108)
    │   │               ├── procmanager (38109)
    │   │               ├── procmanager (38110)
    │   │               ├── procmanager (38111)
    │   │               ├── procmanager (38112)
    │   │               ├── procmanager (38113)
    │   │               ├── procmanager (38114)
    │   │               ├── procmanager (38115)
    │   │               ├── procmanager (38116)
    │   │               ├── procmanager (38117)
    │   │               ├── procmanager (38374)
    │   │               └── procmanager (39463)
    │   ├── SessionLeader (31262)
    │   │   └── Relay(31264) (31263)
    │   │       └── sh (31264)
    │   │           └── sh (31265)
    │   │               └── sh (31270)
    │   │                   └── node (31274)
    │   │                       ├── node (31275)
    │   │                       ├── node (31276)
```

System Resource Monitoring

At the top of the interface, or by clicking a dedicated keyboard shortcut, a system information panel displays critical system metrics:

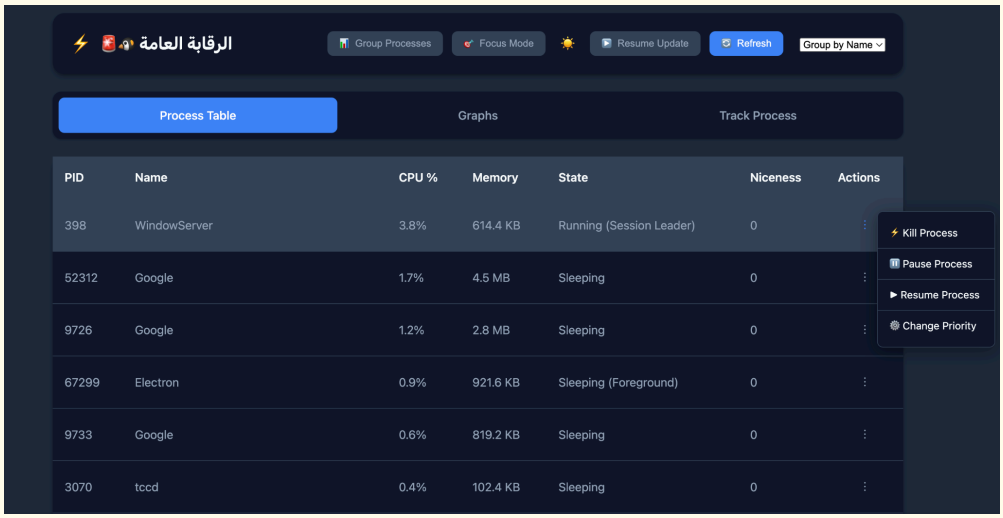
- Overall CPU usage percentage
- Total memory usage and availability
- System load averages
- Total number of processes

This information provides context for the process-specific data and helps users understand overall system health.



GUI Real-Time Monitoring:

In the gui, when the interface is launched, the user will see the table of processes as can be seen in the image below:



This table is refreshed every 2s. The user also has an option to pause this refreshment to be able to analyze the table comfortably.

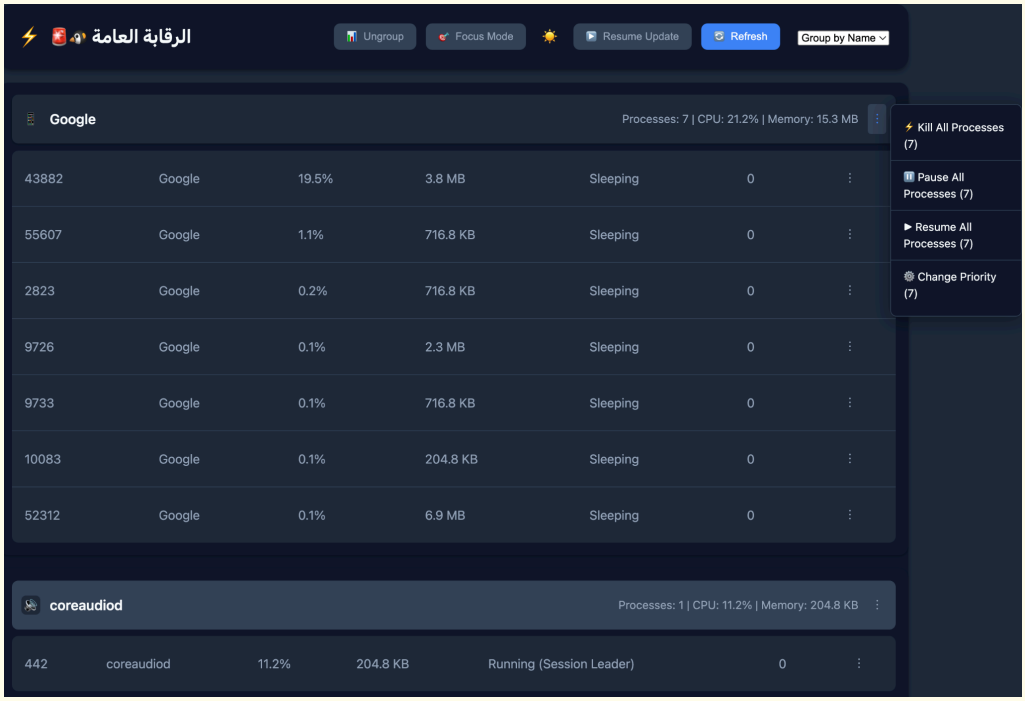
- The actions tab on the right hand side enables the user to either `(kill/ pause/ resume/ or change priority)` to the selected process.
- The user also has the option to monitor the processes over time periods in the “Graphs” and “Track Process” tabs that will be discussed later

Grouping Processes:

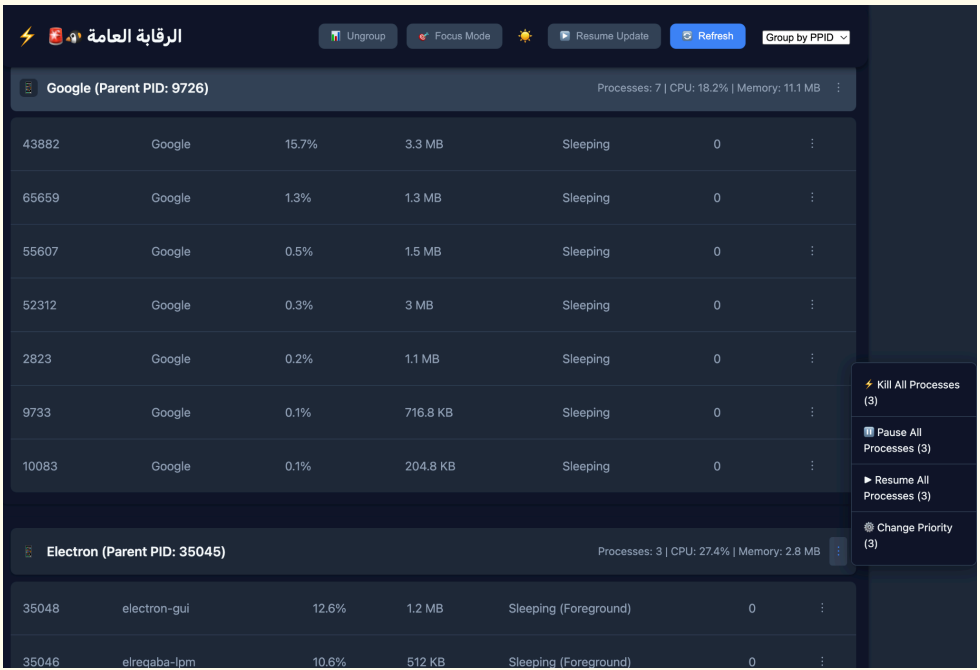
This feature is solely implemented in the GUI, where there is a button to group processes. This grouping can be done in two ways: either by Parent Process ID (PPID) or by the name, since multiple processes happen to have the same name.

Below is a snapshot of both grouping methods as they appear in the GUI application:

- **Grouped by name:**



- **Grouped by PPID:**



The main objective of grouping in the first place was to be able to do collective actions as a whole that are somehow related. As is demonstrated in both screenshots, the user is able to `(kill/ pause/ resume/ change priority)` of an entire group, which was the main goal of grouping. These grouped processes are also useful and are propagated to the graphs and tracking sections, which will be demonstrated later.

Idle Process Detection:

Some heavyweight processes like games or computationally heavy programs, consume, on average, a lot of resources, when these processes are no longer in use, they are still active and consuming CPU resources.

That was the reason we proposed idle process detection in our initial survey, we found such a feature to be very useful and not implemented by other process managers. As for the details, of the implementation, we check on average how much a certain process is consuming resources (ex: CPU, and memory) and when this process starts consuming less and less resources, the idle process unit detects that and sends a message to the user prompting him to terminate the process.

Testing this functionality was a bit challenging, but here is how we did it:

We wrote a C++ program that drains the CPU for 15 seconds:

```
void burn_cpu(int seconds) {
    clock_t start = clock();
    while ((clock() - start) / CLOCKS_PER_SEC < seconds) {
        for (volatile int i = 0; i < 1000000; i++);
    }
}
```

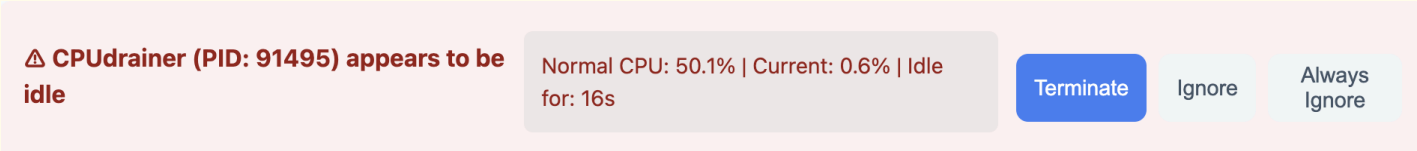
After this burning phase, the program goes into a light work phase, executing an arithmetic operation every 1s.

```
int main() {
    printf("Simulating heavy CPU usage for 15 seconds...\n");
    burn_cpu(15);

    printf("Switching to light CPU load...\n");
    light_work();

    return 0;
}
```

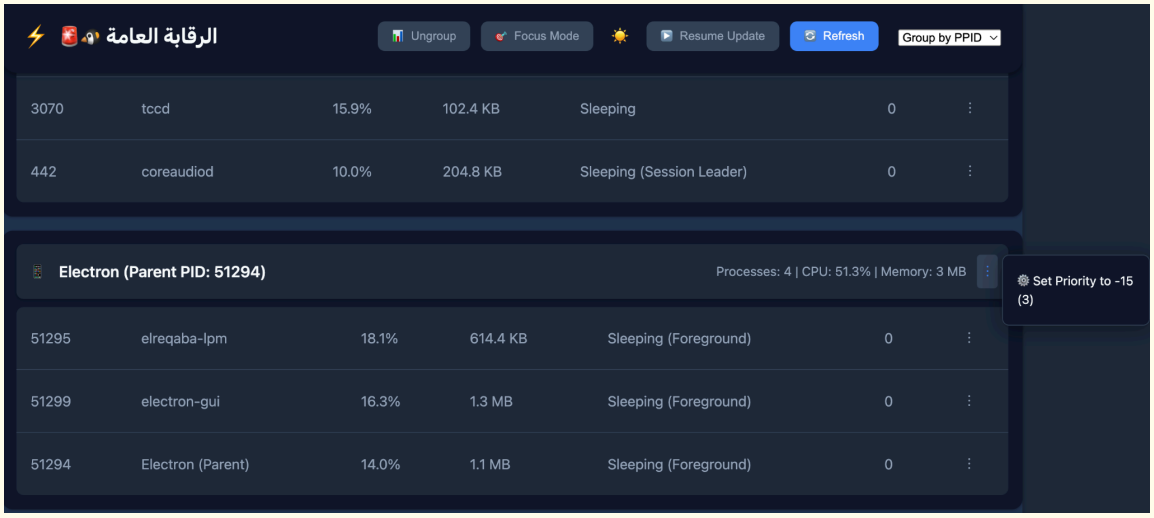
This was detected by our process manager, and a prompt to the user was shown indicating that the process has gone idle as can be seen below:



This verifies that the idle phase has been detected, and the user is then given the option to terminate, ignore, or always ignore, which prevents this process from being suggested by the idle detection unit anymore.

Fault Tolerance:
This is done through the CLI only to keep the process alive. Once the process manager is booted up, it saves the initial state, and when the function is called through the Process ID, it checks if it failed or not and then returns to the initial state to get the command of the Process. Then it runs this command again if found failed and is found in the initial state of the process table. It is called through restart_if_failed <PID>.

Focus Mode:
The purpose of the Focus mode is to give the user the ability to enter a focus mode to give higher priority to certain grouped processes and to hide non-essential system processes. This happens through clicking the focus mode button in the header and then choosing the grouped processes to give them a priority of -15 automatically. Once exiting the focus mode, the priority goes back to what it essentially was.



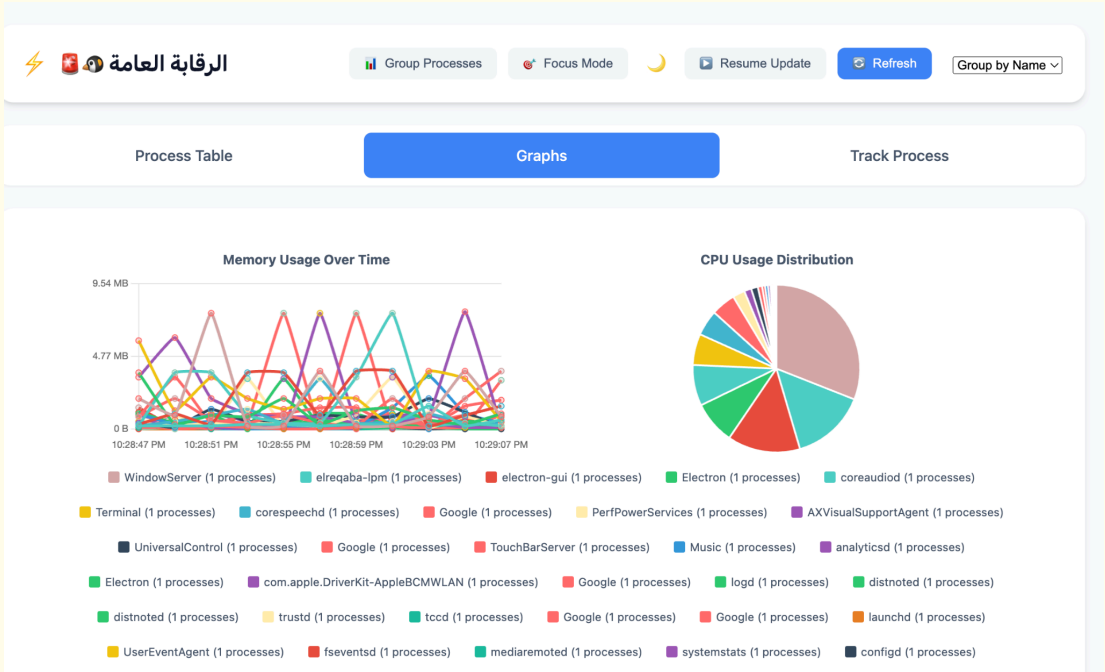
Graphs:

Seeing how effective graphs are in assessing how processes are operating, either by memory usage or CPU consumption percentage. We decided to add graphs accordingly, one tab for all the processes and the other to track processes individually.

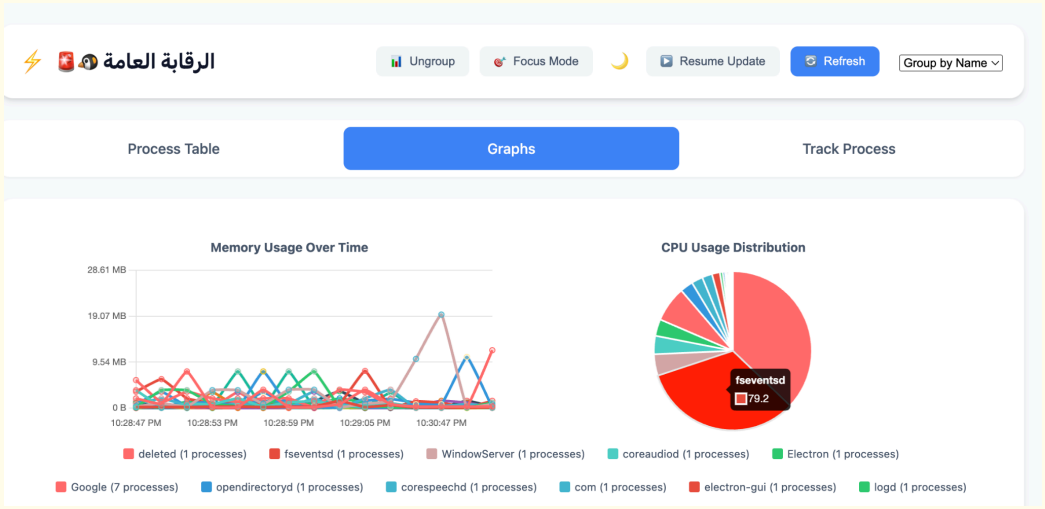
All processes:

As we can see in the photos below, there are two graphs for memory usage and CPU consumption. For both graphs, they are affected by the grouping as mentioned above, either by having the same name or having the same parent process ID. For better visibility, we decided to stick with line charts for memory usage over time, with different colours for the legends. This will help the user determine if any unexpected spikes are occurring. For the CPU usage distribution, given that it adds up to 100%, it was included through a pie chart to see how dominant the process is in terms of CPU usage compared to others.

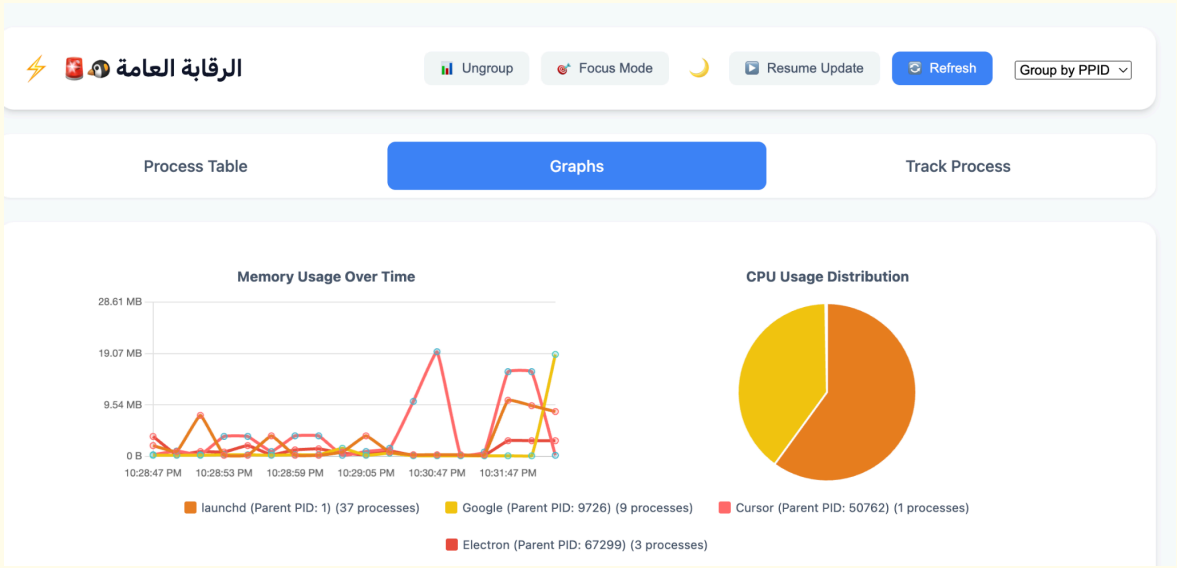
Ungrouped Processes:



Grouped Processes by Name:



Grouped Processes by PPID:



Track Procceses Individaully:

As we can see in the photos below, they target the same graphs as above. However, given that there are not many processes, we stuck with another line chart for better visualization. There is a box to enter the PID of the process to track, and if grouped, it groups the process and shows the memory usage and CPU usage. The user can add as many processes as they may like.

Ungrouped Tracked Processes:

Tracked Processes

PID: 88812 deleted Remove

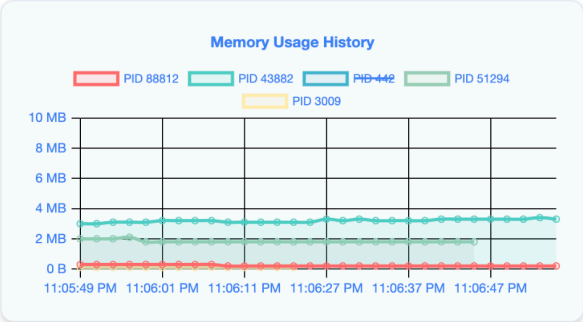
PID: 43882 Google Remove

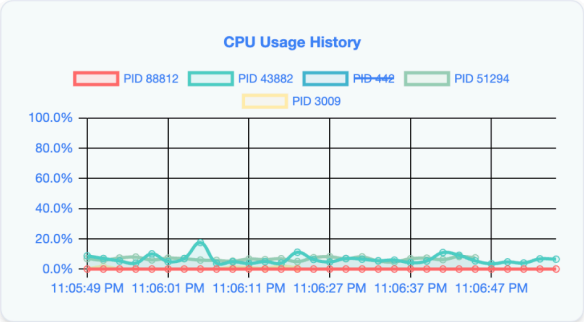
PID: 442 coreaudiod Remove

PID: 51294 Electron Remove

PID: 3009 AXVisualSupportAgent Remove

Grouped Processes





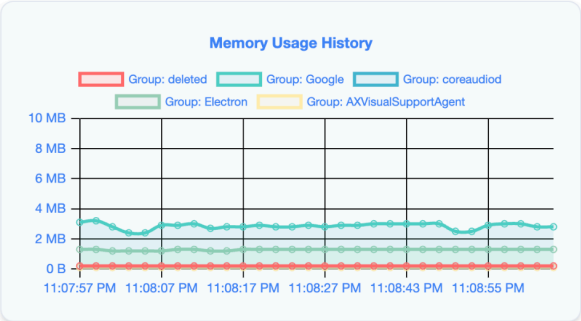
Grouped Tracked Processes by Name:

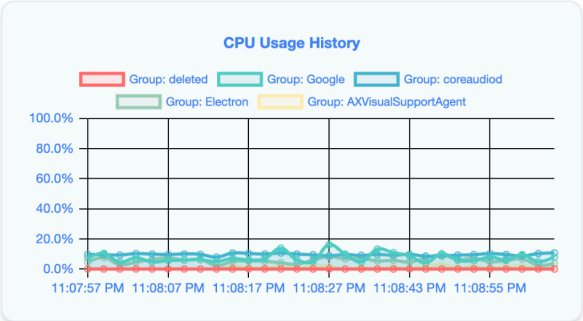
Electron

Process Count: 1
PIDs: 51294
Total Memory: 1.3 MB
Average CPU: 3.5%
Remove Group

AXVisualSupportAgent

Process Count: 1
PIDs: 3009
Total Memory: 102.4 KB
Average CPU: 1.6%
Remove Group





Grouped Tracked Processes by PPID:

No expectations or requirements were disregarded. All goals of the functionality were done and added upon as mentioned above.

Some details about design and implementation

- The CLI and TUI were implemented solely in **rust**, where as the gui was implemented using HTML, CSS, and Javascript
- Despite the different frameworks used, the system is centralized and both the TUI and the GUI can be run from the rust CLI.
- While the Rust interfaces utilize the `sysinfo` crate for fetching process data, some extended process details—like longer or more descriptive states—were extracted using the `ps` command on Linux systems. This hybrid approach allows the application to present richer information beyond what's exposed by the Rust crate alone.
- The process management is implemented through a comprehensive set of controls that allow users to monitor and control system processes. The interface uses the `sysinfo` crate to obtain detailed process information including PID, CPU/memory usage, process state, and priority levels. Users can perform critical operations like killing processes (`SIGKILL`), pausing processes (`SIGSTOP`), and resuming processes (`SIGCONT`) through keyboard shortcuts. The implementation makes use of several Rust crates: `cursive` for the text-based interface components, `cursive_table_view` for displaying sortable process tables, `num_cpus` for hardware information, and `users` for translating user IDs to usernames.
- For the idle detection unit, we first tried a constant threshold (e.g. CPU <0.1) but realized that this was not effective since multiple processes are crucial, yet they consume low CPU percentages, so we figured if we find the decline from the average CPU consumption of a certain process, that proved to be more effective. We also ignored some system-level processes to never appear to the user requesting a prompt.