

Custom Linux Task Manager - Process Monitoring and Management

Project Presentation

Fabian Götschi, Jiri Käser, Manuel Buser, Valerio Job

Operating Systems Lecture Spring Semester 2025

June 13, 2025

Problem Statement

- Monitoring and managing system processes is crucial for system performance and resource allocation.

Problem Statement

- Monitoring and managing system processes is crucial for system performance and resource allocation.
- Existing tools (e.g., top, htop) provide insights but lack flexibility for customization.

Problem Statement

- Monitoring and managing system processes is crucial for system performance and resource allocation.
- Existing tools (e.g., top, htop) provide insights but lack flexibility for customization.
- Our goal was to develop a custom task manager that enables real-time process monitoring and management.

General Approach

- › collect the data with C

General Approach

- › collect the data with C
- › refine the data when needed in C

General Approach

- › collect the data with C
- › refine the data when needed in C
- › display the data in a web UI

General Approach

- › collect the data with C
- › refine the data when needed in C
- › display the data in a web UI
- › invoke resource management calls from the UI

General Approach

- › collect the data with C
- › refine the data when needed in C
- › display the data in a web UI
- › invoke resource management calls from the UI
- › handle the resource management calls correctly in C

Approach

- collect data from `/proc` folder

Approach

- collect data from /proc folder
- General Stats

Approach

- › collect data from /proc folder
- › General Stats
- › Processes

System Architecture

> C core module

System Architecture

- C core module
- http server called c-daemon

System Architecture

- C core module
- http server called c-daemon
- Java REST API

System Architecture

- C core module
- http server called c-daemon
- Java REST API
- web front end

Team organization

We split our group of four into two teams.

Note that Both teams had tasks that occasionally overlapped with the other team.

- Backend team responsible for the c-core and the maintenance of the c-daemon composed of [Jiri|Fabian]

Team organization

We split our group of four into two teams.

Note that Both teams had tasks that occasionally overlapped with the other team.

- Backend team responsible for the c-core and the maintenance of the c-daemon composed of [Jiri|Fabian]
- Frontend team responsible for the web frontend UI and the Java API composed of [Manuel|Valerio].

General Stats

> CPU

General Stats

- > CPU
- > Memory

General Stats

- CPU
- Memory
- Disk

General Stats

- › CPU
- › Memory
- › Disk
- › Network

General Stats

- CPU
- Memory
- Disk
- Network
- GPU(NVIDIA)

Processes Info

> State

Processes Info

- › State
- › CPU

Processes Info

- › State
- › CPU
- › RAM

Processes Info

- › State
- › CPU
- › RAM
- › Priority

Processes Info

- › State
- › CPU
- › RAM
- › Priority
- › Sleeper Detection

Background c-daemon

Http server running in the background handing frontend requests

Examples for such requests are:

- GET */api/processes returns list of processes and statistics on each of them

Background c-daemon

Http server running in the background handing frontend requests

Examples for such requests are:

- GET */api/processes returns list of processes and statistics on each of them
- GET */api/stats/all returns all general system data required for the system stats page

Background c-daemon

Http server running in the background handing frontend requests

Examples for such requests are:

- › GET `*/api/processes` returns list of processes and statistics on each of them
- › GET `*/api/stats/all` returns all general system data required for the system stats page
- › POST `*/api/processes/pid/signal` will kill the process with PID

Background c-daemon

- Uses microhttpd for the server

Background c-daemon

- Uses microhttpd for the server
- Most important files are `routes.c` and `[limit|process|stats]_routes.c`

Background c-daemon

- Uses microhttpd for the server
- Most important files are `routes.c` and `[limit|process|stats]_routes.c`
- `routes.c` checks the requests. If valid it executes the functionalities requested

Background c-daemon

- Uses microhttpd for the server
- Most important files are `routes.c` and `[limit|process|stats]_routes.c`
- `routes.c` checks the requests. If valid it executes the functionalities requested
- `[limit|process|stats]_routes.c` contain the handles for the functionalities that are called by `routes.c`

Background c-daemon

- › Uses microhttpd for the server
- › Most important files are `routes.c` and `[limit|process|stats]_routes.c`
- › `routes.c` checks the requests. If valid it executes the functionalities requested
- › `[limit|process|stats]_routes.c` contain the handles for the functionalities that are called by `routes.c`
- › These handles then rely on the c-core for final execution.

Example for process data sent from c-daemon

```
▼ 0:
pid:      1
cmd:      "/sbin/init splash"
comm:     "systemd"
state:    "S"
cpuPercent: 0
ramPercent: 0.045231284681409731 JS: 0.04523128468140973
nice:     0
username: "root"
prio:     20
virt:     24272
res:      14740
shared:   9236
upTime:   12977.6
is_sleeper: 0
▼ 1:
pid:      2
cmd:      ""
comm:     "kthreadd"
state:    "S"
cpuPercent: 0
```

Web API and Front End structure

The basic structure contained:

- Java Spring Boot

Web API and Front End structure

The basic structure contained:

- Java Spring Boot
- Thymeleaf

Web API and Front End Main page

The screenshot displays the Linux Task Manager web interface. At the top, there's a header with the title "Linux Task Manager" and a button "View History & General Stats". Below the header, there are three input fields for PID, each with a dropdown menu, and three corresponding buttons: "New nice" (yellow), "Limit CPU" (green), and "Limit RAM" (green). Below these, there are checkboxes for "Hide Kernel Threads", "Show Sleepers Only", and "Freeze Table", along with two buttons for sorting: "Sort by CPU" and "Sort by RAM".

PID	User	Prio	Nice	VIRT	RES	SHR	Command	Up Time	Name	State	% CPU	% RAM	Actions
1825	manu	20	0	6138380	67004	173096	/usr/bin/gnome-shell	334.9	gnome-shell	S	7.8	5.6	[Kill]
4111	manu	20	0	3201940	448048	187376	/usr/bin/firefox-k259/usr/lib/firefox/firefox	129.1	Firefox	S	3.3	3.9	[Kill]
4601	manu	20	0	10968116	173060	101588	/usr/bin/firefox-k259/usr/lib/firefox/firefox -conten...	120.9	Isolated Web Co	S	1.6	1.4	[Kill]
2354	manu	20	0	1217894132	107204	74368	/usr/share/code/code --type=utility --utility-sub...	290.7	code	S	0.4	0.9	[Kill]
3888	manu	20	0	6381308	270300	24632	/usr/bin/java -classpath /usr/share/maven/boot/pla...	143.7	java	S	0.1	2.3	[Kill]
3966	manu	20	0	7513412	195392	24600	/usr/lib/jvm/java-21-openjdk-amd64/bin/java -XX:TL...	140.0	java	S	0.1	1.6	[Kill]
2587	manu	20	0	1217879112	240948	81152	/usr/share/code/code --type=utility --utility-sub...	290.5	code	S	0.1	2.0	[Kill]
3556	manu	20	0	884836	8456	3712	/http_server	192.4	http_server	S	0.1	0.1	[Kill]
1175	root	20	0	356494	2432	2304	/usr/bin/VBoxDRMClient	351.0	VBoxDRMClient	S	0.1	0.0	[Kill]
1	root	20	0	166732	11316	7968	/bin/init splash	362.0	systemd	S	0.0	0.1	[Kill]
291	root	19	-1	65289	29312	28160	/lib/systemd/systemd-journald	359.3	systemd-journal	S	0.0	0.2	[Kill]
344	root	20	0	27076	6912	4608	/lib/systemd/systemd-sdlevd	358.7	systemd-sdlevd	S	0.0	0.1	[Kill]
629	systemd-oom	20	0	14836	6784	6016	/lib/systemd/systemd-oomd	355.8	systemd-oomd	S	0.0	0.1	[Kill]
630	systemd-resolve	20	0	26464	14256	9344	/lib/systemd/systemd-resolved	355.8	systemd-resolve	S	0.0	0.1	[Kill]

Web API and Front End System statistic page

History & General Stats

[← Back to Dashboard](#)

Last updated: 19:22:59

Load Avg (1m,5m,15m)
4.28, 2.37, 1.40

CPU Util
19.8%

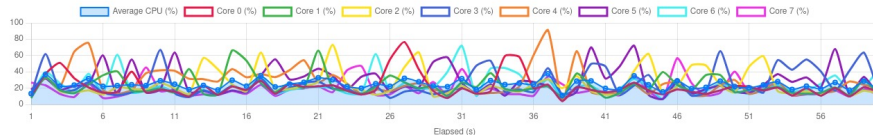
Tasks
total=1207, running=1

Memory (MB)
**total=11676 MB, free=4787 MB,
avail=6962 MB**

Buffers + Cached
buffers=61 MB, cached=2555 MB

Swap (MB)
total=2048 MB, free=2048 MB

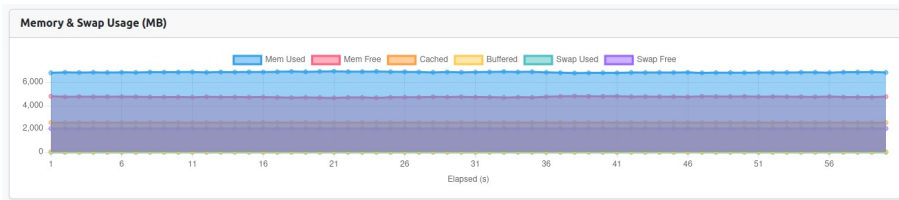
CPU Usage (%)



Web API and Front End System statistic page



Web API and Front End System statistic page



C Dependencies

- standard C
- libmicrohttpd
- libjson

Java / Spring Dependencies

- maven
- REST API
- Thymeleaf
- Jackson

Demo

The screenshot displays the Linux Task Manager web interface. At the top, there's a header with the title "Linux Task Manager" and a button "View History & General Stats". Below this, there are three input fields for PID, each with a dropdown menu and a button to perform an action: "New nice" (yellow button), "CPU secs" (green button), and "RAM MB" (green button). To the right of these buttons are labels "Renice", "Limit CPU", and "Limit RAM".

Below the input fields, there are three checkboxes: "Hide Kernel Threads" (checked), "Show Sleepers Only", and "Freeze Table". To the right of these checkboxes are two buttons: "Sort by CPU" and "Sort by RAM".

The main part of the interface is a table listing running processes. The table has columns for PID, User, Prio, Nice, VIRT, RES, SHR, Command, Up Time, Name, State, % CPU, % RAM, and Action. The table contains 15 rows of process data.

PID	User	Prio	Nice	VIRT	RES	SHR	Command	Up Time	Name	State	% CPU	% RAM	Action
1825	manu	20	0	6138580	674344	173096	/usr/bin/gnome-shell	334.9	gnome-shell	S	7.8	5.6	Kill
4111	manu	20	0	3201040	468048	187376	/usr/bin/firefox /usr/lib/firefox/firefox	129.1	firefox	S	3.3	3.9	Kill
4601	manu	20	0	10968116	173060	101588	/usr/bin/firefox /usr/lib/firefox/firefox -content...	126.9	Isolard Web Co	S	1.6	1.4	Kill
2554	manu	20	0	1217894132	107204	74368	/usr/share/code/code --type=utility --utility-sub...	290.7	code	S	0.4	0.9	Kill
3888	manu	20	0	6381308	270300	24632	/usr/bin/java -classpath /usr/share/nasv/boot/spl...	143.7	java	S	0.1	2.3	Kill
3966	manu	20	0	7513412	195392	24600	/usr/lib/jvm/java-21-openjdk-amd64/bin/java XXXSL...	140.0	java	S	0.1	1.6	Kill
2587	manu	20	0	1217879112	240948	81152	/usr/share/code/code --type=utility --utility-sub...	290.5	code	S	0.1	2.0	Kill
3556	manu	20	0	86836	8656	3712	/http_server	192.4	http_server	S	0.1	0.1	Kill
1175	root	20	0	356404	2432	2304	/usr/bin/VBoxDMMClient	351.0	VBoxDMMClient	S	0.1	0.0	Kill
1	root	20	0	166732	11316	7988	/bin/init splash	342.0	systemd	S	0.0	0.1	Kill
291	root	19	-1	65280	29312	28160	/lib/systemd/systemd-journald	359.3	systemd-journal	S	0.0	0.2	Kill
344	root	20	0	27076	6912	4608	/lib/systemd/systemd-udev	358.7	systemd-udev	S	0.0	0.1	Kill

What We Achieved

- › task manager with C
- › sleeper detection
- › commands for process management
- › web UI instead of terminal based UI
- › stable communication between frontend and backend

Improvements

What could be added extra:

- more info per Process

Improvements

What could be added extra:

- more info per Process
- more Commands

Improvements

What could be added extra:

- more info per Process
- more Commands
- show entire history instead of 60 seconds

Improvements

What could be added extra:

- › more info per Process
- › more Commands
- › show entire history instead of 60 seconds
- › remote monitoring

Improvements

What could be added extra:

- › more info per Process
- › more Commands
- › show entire history instead of 60 seconds
- › remote monitoring
- › clean up

Improvements

What could be added extra:

- more info per Process
- more Commands
- show entire history instead of 60 seconds
- remote monitoring
- clean up
- full Disk support

Improvements

What could be added extra:

- › more info per Process
- › more Commands
- › show entire history instead of 60 seconds
- › remote monitoring
- › clean up
- › full Disk support
- › Windows support

Improvements

What could be added extra:

- › more info per Process
- › more Commands
- › show entire history instead of 60 seconds
- › remote monitoring
- › clean up
- › full Disk support
- › Windows support
- › full GPU support

Lessons learned

- Understanding of `/proc`

Lessons learned

- Understanding of `/proc`
- what is required to efficiently parse larger numbers of files

Lessons learned

- Understanding of `/proc`
- what is required to efficiently parse larger numbers of files
- how to use system calls

Lessons learned

- › Understanding of `/proc`
- › what is required to efficiently parse larger numbers of files
- › how to use system calls
- › how to build an API in C and Java

Lessons learned

- › Understanding of /proc
- › what is required to efficiently parse larger numbers of files
- › how to use system calls
- › how to build an API in C and Java
- › how to debug C

Lessons learned

- › Understanding of /proc
- › what is required to efficiently parse larger numbers of files
- › how to use system calls
- › how to build an API in C and Java
- › how to debug C
- › how to run a Java Spring Boot web application