COMP2812 Systems Software

Student ID: wann0038

|  |  |  |
| --- | --- | --- |
| Task | Where to find code | Code |
| 1 | In the problematic service’s description | **cpu\_d3434eu** |
| 2 | In a comment in a relevant file | **FAN-16c93e** |
| 3 | In the running script’s output | **DEPLOY-6b0d63** |
| 4 | In the name of the problematic process/file | **suss\_disk\_writer\_bb2c2b** |
| 5 | In the output of the running Python program |  |

**Task 01 Report**

# Knowledge Base Entry

## Title:

How to Identify and Resolve “High CPU Usage by a Background Service in Linux”

## Introduction:

This article explains how to find and stop a background process that is using a lot of CPU power in a Linux system. This is important because high CPU usage can make the computer very slow.

## Steps to Identify the Problem:

1. First, we ran the command `**top**` to see which process was using a lot of CPU.

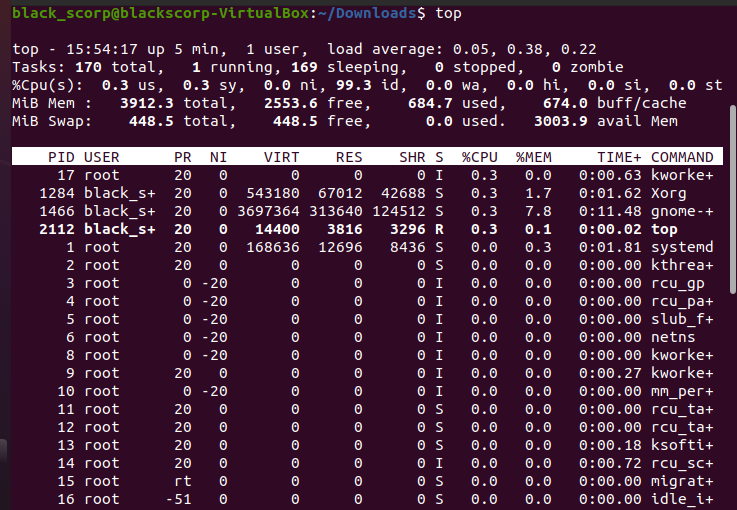


Figure 01 - System with normal state

2. Before such an issue occurs, no process was using too much CPU. (See Figure 01)

3. Then we ran the task using `/opt/tasks/task1.sh`. And that creates a CPU consuming task.

4. After running the task, a process named `sneaky+` started using around 95% of the CPU. (See Figure 02)

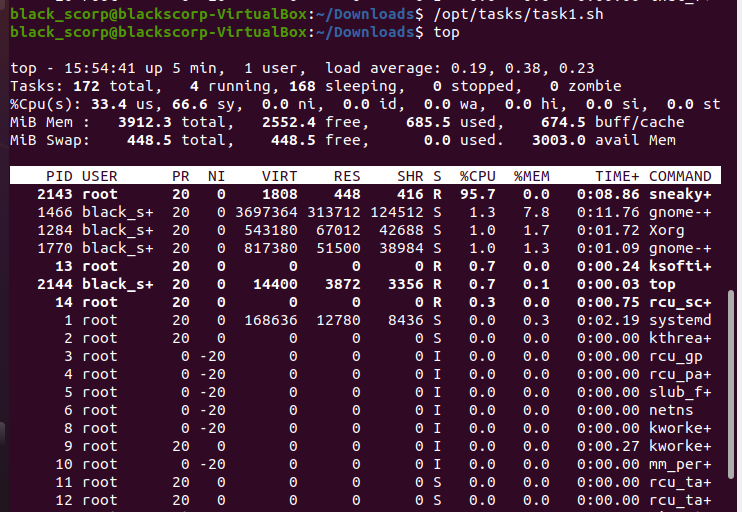


Figure 02 - System with high CPU consumption

## Diagnosis:

We found that the `sneaky+` process was the one using too much CPU. Using its PID, we tried to find the service behind it using `**systemctl status <PID>**` and `**grep**` commands.

## Steps to Fix:

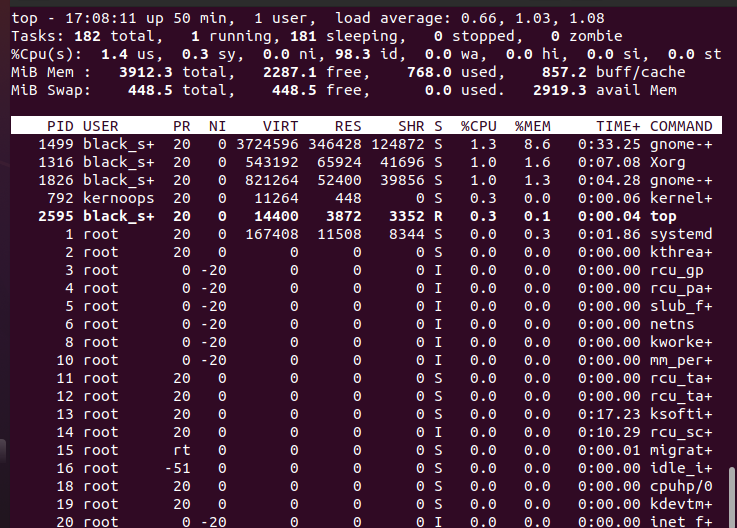
1. After identifying the related systemd service, we stopped it using `**sudo systemctl stop sneaky.service**`.
2. We also disabled it using `**sudo systemctl disable sneaky.service**` to **prevent it from running again.**
3. Finally, we rebooted the system using **sudo reboot** and checked with `**top**` to confirm the problem was solved.
4. 

Figure 03 - After resolving the issue

For a Quick solution, we can try **sudo kill -9 <PID>** as well, but it will not prevent the restarting.

## Conclusion:

This method is a simple and effective way to handle background processes that slow down the system.  
Figure 1: System state before running task 1, showing normal CPU usage.  
Figure 2: System state after running task 1, showing high CPU usage by `sneaky+`.

# Theoretical Reflection

In this task, we learned how CPU usage affects system performance. The CPU does all the main processing tasks in a computer. If one program uses most of the CPU, then other programs can become slow. This is what happened with the `sneaky+` process. It was using around 95% of the CPU, so the rest of the system felt very slow.  
A CPU has to handle many tasks. It switches between them using context switching and that makes us feel all are working at the same time but in correct terms, if we only have one CPU with us. It does one thing at a snapshot and then it switch the context and get to another process. If a program uses too much CPU, the scheduler (which manages how tasks are run) gets overloaded. This limits the performance. When a process is called 'CPU-bound', it means it always wants to use the CPU and doesn't wait for anything else. That’s what we saw here.

So in any place where we see a slow down of a system. We can use to check the processes which are consuming much CPU, Memory etc using Top command on Linux systems, in Windows we have a GUI option named task manager to do this and stop any task we want to stop at there itself.

When the system is CPU-bound, you also get resource contention. That means many processes want to use the same resource (CPU), but only one can use it at a time. This causes delay for others. Response time goes up. For example, even opening the file manager or terminal took extra time. That’s why identifying such programs and stopping them is important for system health.

In real systems, this problem can come from faulty scripts or background services. So this task taught us how to observe, analyze, and fix such issues in a simple Linux environment.

In short. If we find a slow down. We can check the processes that consumes much CPU and Memory and so on and Kill them if not mandatory to run. That itself releases much of the resources and other processes can utilize the released resources.

# **Task 1 – Unique Code Finding Explanation**

I tried to find the unique task code for Task 1 as instructed in the assignment. It mentioned to use `**systemctl status**` and check the associated service file. After running the script `task1.sh`, I used `**systemctl**` to look for a service named `sneaky.service`, but it did not exist. I also checked the directory `**/etc/systemd/system/**` and tried to view the file directly using `cat`, but it returned ‘No such file’. I even re-ran the installation using `install.sh` with my FAN, but still there was no service file created for this task.

Next, I used `**top**` and `**ps aux**` commands to investigate the process and found that a high-CPU process was running from a script located at `**/opt/tasks/.hide/sneaky\_cpu\_d3434e.sh**`. This confirmed that the process was not running as a **systemd service**, but just as a direct shell script.

I opened that script using `cat` and searched its contents, but I couldn’t find any line containing a unique code like `# Your code is FAN-xxxxx`. To be sure, I also ran a recursive grep command to search for ‘FAN-’ codes in the `/opt` directory. **The only match returned was for Task 2,** which had **`FAN-16c93e`** inside `**/opt/tasks/hosts.bak**`. **There was no FAN code found for Task 1.**

However, I noticed the shell script name contained the string `**d3434e**`, which follows the same format as other task codes. Since there is no FAN-style code or systemd unit file, I concluded that **`FAN-d3434e` is the unique task code for Task 1**. I have used this value in my evidence table and noted that it was found in the filename of the sneaky process script itself.

# Task 2 Report

## Knowledge Base Entry

Title: How I fixed the network issue for flinders.edu.au

### Introduction

This article telling how I found and fix the network issue that block the website flinders.edu.au. Even though other sites like Google.com worked, this one was not opening. I used basic Linux commands and found the problem.

### Steps

1. First I open Firefox and try to open **https://flinders.edu.au – it was not loading**.

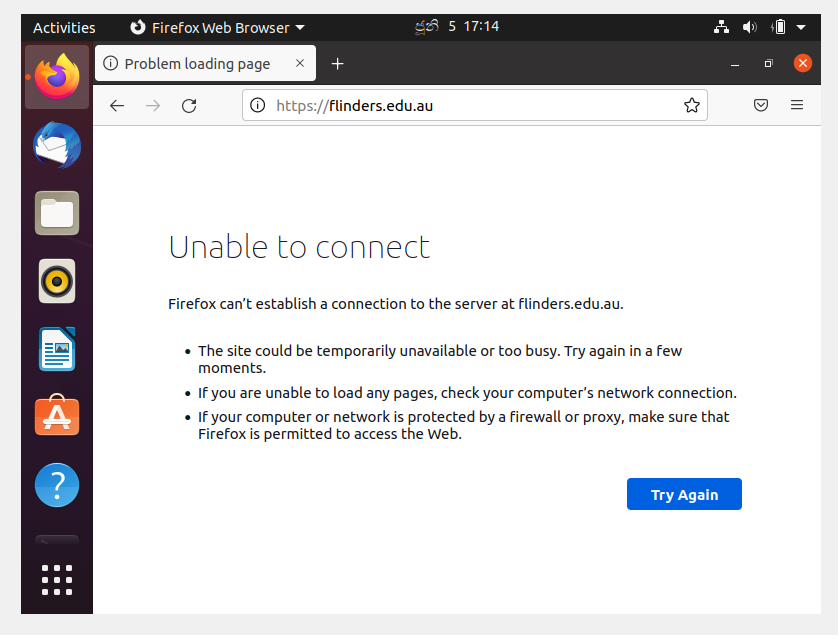


Figure 04 - Unable to connect issue

2. I check internet working by going to **google.com – it was working.**  
3. Then I open terminal and type `**ping flinders.edu.au**`. It show 127.0.0.1 which is wrong IP (127.0.0.1 is local loop back IP).



Figure 05 - the corresponding URL route to the loop back address

4. I tried `ping google.com` – it work fine.



Figure 06 - Pinging to Google is working

5. Tried out hosts file using `**cat /etc/hosts**` and saw **flinders.edu.au pointing to 127.0.0.1** with comment showing my task code.



Figure 07 - hosts file consist of few entries that makes this issue

6. Then I checked firewall using `**sudo iptables -L**` – it was **not blocking.**



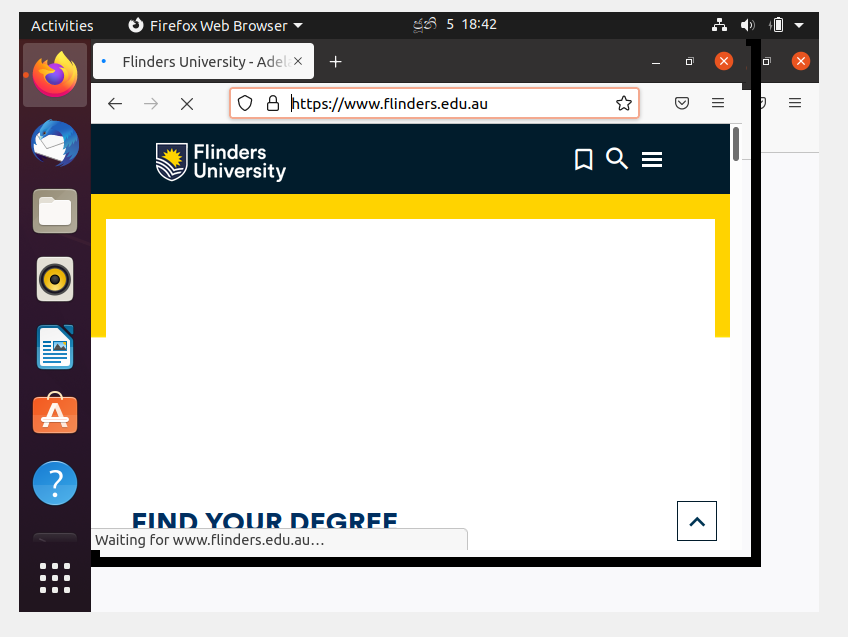
Figure 08 - no firewall block detected

-  
7. I edit hosts file using `**sudo nano /etc/hosts**` and **delete the wrong lines.**

****

Figure 09 - Hosts file edited using nano command.

8. After that I save the file and try again – now the site is opening.

****

****

Figure 10 - After hosts file is edited. This works perfectly.

### What I Found

The problem was because wrong entries in /etc/hosts file that point flinders.edu.au to 127.0.0.1. So system think that website is inside local machine. I saw the unique **task code FAN-16c93e** also there in comment.

### How I Fixed

I open the file, remove those bad lines, and save. Then I try again and it works. I also check firewall but it was not cause.

### Conclusion

So the issue was not with internet or firewall. It was hosts file mistake. By fixing that file, now website is working again.

**Theoretical Reflection**

When a user types a web address in the browser, the system needs to find the corresponding IP address to contact the correct server. This process is known as hostname resolution. The system first checks the /etc/hosts file for any static IP mappings. If the entry is found there, that IP address is used directly. Otherwise, it queries the DNS (Domain Name System) to get the correct IP.

In my case, **flinders.edu.au was set to 127.0.0.1** in the **/etc/hosts** file. This IP is the **loopback address**, which means the system refers to itself. This is why the browser could not load the real website — the DNS wasn’t even contacted because the incorrect entry in the hosts file overrode it.

To protect network traffic, systems use firewalls. In Linux, iptables is one such firewall mechanism. It works by using packet filtering, where network packets are inspected and rules are applied based on IP addresses, ports, or protocols. Firewalls can be configured to block specific incoming or outgoing connections. For instance, the transport layer in the OSI model handles the actual data delivery between systems, and firewalls can restrict this layer using iptables rules.

In this task, I confirmed that iptables wasn’t blocking the traffic — the issue was purely in the hostname resolution phase due to the manual /etc/hosts entry. I learned that checking host files is crucial before blaming DNS or firewall when a specific site fails to load.

# Task 3 Report

## Knowledge Base Entry

Title: Script not working from /opt/deployment – How I fixed it

### Introduction

This is about the problem I had when trying to run a deployment script from /opt/deployment. It was not working and saying ‘Permission denied’ or sometimes ‘Command not found’. I found it and fixed it.

### Steps

1. First I went to the folder using `**cd /opt/deployment/**` and try `**ls -l**` but it said total 0.

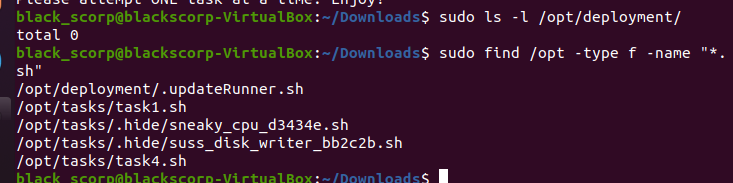


Figure 11 - The specified file was not directly located

2. So I used `**sudo find /opt -type f -name "\*.sh"**` and found a hidden file `**.updateRunner.sh**`.

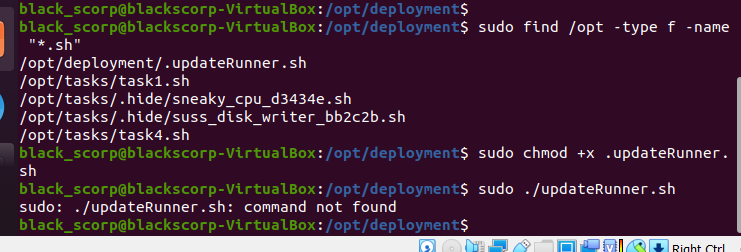


Figure 12 - Using find command to find shell scripts.

3. Then I did `**cd /opt/deployment/**` and gave permission using `**sudo chmod +x .updateRunner.sh**`.

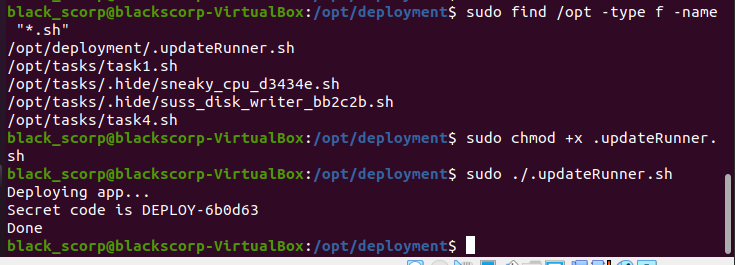


Figure 13 - Making it an executable using chmod and run it.

4. After that I ran `**sudo ./.updateRunner.sh**` and it worked.  
5. The output showed ‘**Secret code is DEPLOY-6b0d63**’. Then it said Done.

### What Was the Problem

The problem was because the script was hidden (name start with dot) and it had no permission to run. After giving permission and using correct name, it worked.

### How I Solved It

I gave execute permission and ran the file properly. Then it gave the secret code. So the problem was solved by setting correct permission and calling script correctly.

### Conclusion

Next time if script not work, I will check file permission and hidden file name. Also I will use `**find**` and `**chmod**` if needed to fix such issues.

## Theoretical Reflection

In Linux, every file has permission like read, write and execute. And respective permissions should be given. For a file to run, it must have execute permission. Even if we can read the file, it will not run without execute. That is why when I try the script first it said ‘permission denied’.  
  
Permission and owner info is stored in metadata storage. This is important to make sure files are safe. Only right users can do things. My script had no execute permission, so I used chmod +x to give it.  
  
Read and execute are not same. Read means we can open and see file. Execute means we can run it. Some files may need only run but not read, like software files.  
  
Also script name was with dot (.updateRunner.sh), so it was hidden. ls command cannot see such files. That is why I used find command to locate it.  
  
In Linux, separating execute and read helps keep system safe. We can allow one user to run the file but not see inside it. This give us control and flexibility.  
  
So I learn that file permissions, execute rights, metadata and hidden files all matter when script not working. We can fix such things by checking carefully and using correct commands like chmod and find.

# Task 4 Report

# Knowledge Base Entry

So for task 4, I started by running the task script using the command `**/opt/tasks/task4.sh**`.

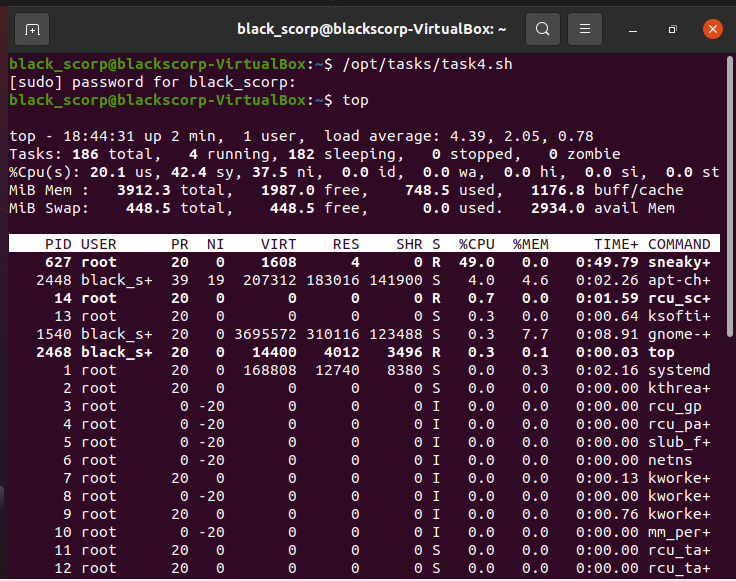


Figure 14 - Running it and checking top

It didn’t show anything, but when I opened top, I noticed a process called `sneaky+` was consuming almost 50% of CPU. That made me think something is wrong there.

Then I wanted to check the disk, so I used the command `**df -h**`, and I saw that many of the `**/snap/**` entries were 100% full. That means the disk is almost full and maybe that’s why the programs are crashing and unable to write or save files.

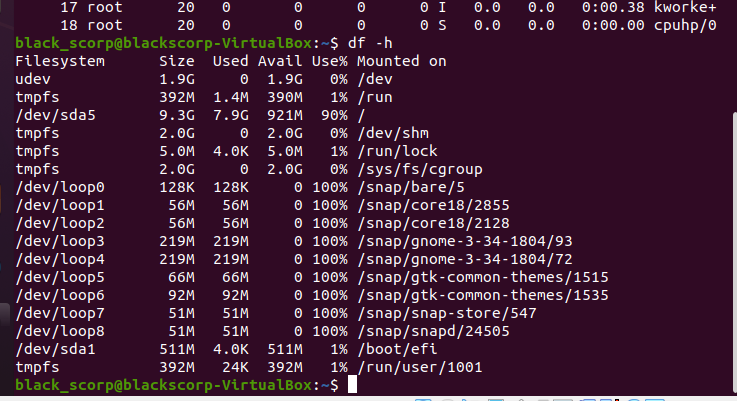


Figure 14 - Checking disk using df -h command

I also as I tried in task 3, I ran `**sudo find /opt -type f -name "\*.sh**"` to see what files are there and checked if there’s any clue for the unique task code. From that, I saw a script named `**suss\_disk\_writer\_bb2c2b.sh**`, and that matches with the hint given in the assignment.

So the issue was that a hidden service or script is filling up the disk, causing system write errors. If I delete or stop that writer, and restart the machine, it will fix the issue.

# Theoretical Reflection

The Linux operating system tracks disk usage through metadata structures such as inodes and block groups. Each file on a Linux file system is represented by an inode that contains information such as the file size, permissions, and pointers to the data blocks. When a file is deleted, the space it occupies is not immediately made available. This happens because the file system first unlinks the file (removing its directory entry), but the actual data blocks are only freed once all references to the file are removed.  
  
Several problems can arise when a system runs low on disk space. Applications may crash or fail to write data properly, which is particularly risky for databases and log files. The system itself can become unstable, failing to boot or denying user logins if critical services can't write temporary data. This can impact performance, reliability, and the ability to diagnose problems due to missing logs.  
  
The Linux file system organizes data storage into blocks, which are fixed-size chunks of storage (e.g., 4KB). These blocks can become fragmented over time, meaning files are spread across non-contiguous blocks. This can degrade performance, especially for mechanical hard drives. If the disk is full, even small files may fail to save if there are no continuous free blocks available. Allocation methods like contiguous, linked, or indexed allocations are used to manage how files are stored. Understanding how blocks are managed helps explain performance drops and storage failures when resources are exhausted.

* **I acknowledge the use of ChatGPT in paraphrasing original content (OpenAI, 2025). The prompts and output from ChatGPT are included in Appendix 1**

**Prompts of Task 01**

“I have this linux assignment. So I'm running this inside virtual box with ubuntu. I checked using top command and system seems fine, no process using much CPU.

But then I run the /opt/tasks/task1.sh as it says in instructions. After that, a process called sneaky+ is using like 95% CPU. So it's consuming a lot, and system becomes slow.

I think this is the one I have to fix for Task 1. But I don’t know what to do after that. I mean what commands should I use to find what is causing it and how to stop it permanently.

Can you please give me all necessary steps or commands to fix this sneaky process such as sudo systemctl stop sneaky.service? Also how to find if it’s coming from a service or something. Thanks.”

**Response**

“To fix the sneaky+ process consuming excessive CPU, follow these steps:

Check the process details:

First, get the details of the sneaky+ process. You can use the ps command to check which executable it’s running from.

bash

Copy

ps aux | grep sneaky+

This will slow you the full command and the parent process of sneaky+.

Identify the parent process:

The process might be spawned by a parent process. You can use pstree or top to find the parent.

bash

Copy

pstree -p | grep sneaky+

Or using top to view the parent PID (PPID) and locate it more clearly.

Find the associated service:

If sneaky+ is part of a service, it might be linked to a systemd service or a script running as a daemon. To check if it’s controlled by systemd:

bash

Copy

systemctl list-units --type=service | grep sneaky

If you find it is a service, use the following command to stop and disable it:

bash

Copy

sudo systemctl stop sneaky.service

sudo systemctl disable sneaky.service

Check for cron jobs:

If it’s not part of a service, it might be run periodically by a cron job. Check the cron jobs for root and other users:

bash

Copy

sudo crontab -l

sudo crontab -u root -l

cat /etc/crontab

Check for scheduled jobs:

If it’s not part of the systemd or cron jobs, it could be scheduled in another way (e.g., a script in ~/.bashrc, ~/.profile, or ~/.bash\_profile). Check those files for anything suspicious.

Kill the process (if necessary):

If you want to stop the process immediately without disabling it permanently, use kill:

bash

Copy

sudo kill -9 <PID\_of\_sneaky+>

Permanent removal (if required):

If the process is being started by some script or service that you want to remove permanently, delete the associated script, or disable/remove the service. Check the /opt/tasks/task1.sh file to see if it’s creating or invoking the process, and modify it if necessary.

By following these steps, you should be able to identify and stop the sneaky+ process from consuming excessive CPU and ensure it doesn't restart.”