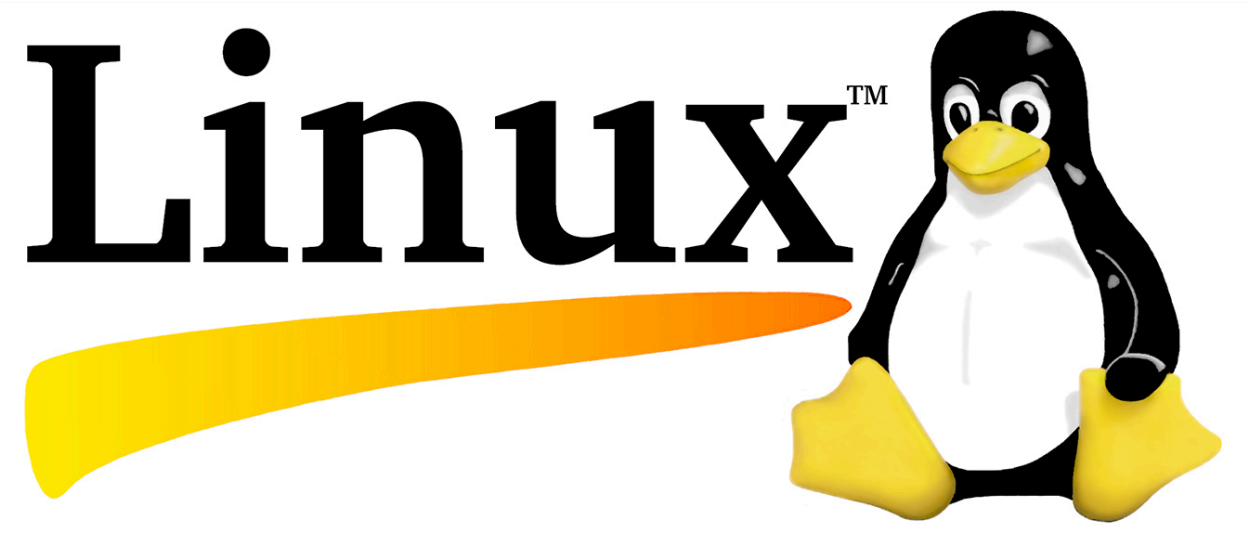
Linux Optimization



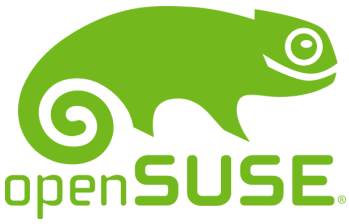


Table of content

optimize Linux services

Systemd is the init system used by many modern Linux distributions, and it comes with its own set of tools for managing services. Here are some tips for optimizing services using systemd:

View Service Status:

Use systemctl to view the status of services.

```
sudo systemctl status servicename
```

Enable/Disable Services:

Disable unnecessary services to improve boot time.

```
sudo systemctl disable servicename
```

Enable services when needed.

```
sudo systemctl enable servicename
```

Startup Priorities:

Adjust the startup priority of services using systemctl.

Check dependencies and order

```
sudo systemctl list-dependencies --after
```

Change startup priority

```
sudo systemctl edit --full servicename
```

In the service file, you can adjust the Before and After directives to control startup order.

Resource Limits:

Set resource limits for services using systemd resource control.

Example: Set CPU limits

```
sudo systemctl edit --full servicename
```

Add the following lines:

```
[Service]
CPUQuota=20%
This example limits the service to using 20% of the CPU.
```

Journal Control:

Configure the journal settings to control logging and disk usage.

```
# View journal status
sudo journalctl --disk-usage

# Limit the journal size
sudo systemctl edit --full systemd-journald
```

Add the following lines:

```
[Journal]
SystemMaxUse=50M
This example limits the journal size to 50 megabytes.
```

Restart Policies:

Define restart policies for services in the service file.

```
sudo systemctl edit --full servicename
```

Add the following lines:

```
[Service]
Restart=always
```

This example restarts the service always, regardless of the exit status.

Slice Configuration:

Use slices to group and manage services with similar resource requirements.

```
# Create a slice configuration
sudo systemctl edit --full slice-name
```

Add the following lines:

```
[Slice]
CPUQuota=20%
```

Assign services to the slice in their service files.

Timer Units:

Use timer units to schedule services.

```
# Create a timer configuration
sudo systemctl edit --full timer-name
```

Add the following lines:

```
[Timer]
OnCalendar=daily
This example runs the associated service daily.
```

Remember to reload the systemd manager after making changes:

```
sudo systemctl daemon-reload
```

Always test changes in a safe environment before applying them to a production system. Also, refer to the systemd documentation and man pages for detailed information on configuration options and best practices.

optimize Installed Linux with commands

Optimizing a Linux system involves a combination of fine-tuning configurations, managing services, and ensuring efficient resource utilization. Here are some commands and tips to help optimize an installed Linux system:

Update System Packages:

Keep the system up to date with the latest security patches and updates.

```
# Debian/Ubuntu
sudo apt update && sudo apt upgrade
```

Red Hat/CentOS

```
sudo yum update
```

Remove Unnecessary Packages:

Identify and remove unnecessary packages that are installed on the system.

```
# Red Hat/CentOS
sudo yum update
```

```
# Red Hat/CentOS
sudo yum autoremove
```

Check Disk Space:

Monitor disk space usage to prevent potential issues.

```
df -h
```

Optimize Disk I/O:

Use iotop to identify processes causing high disk I/O.

```
sudo iotop
```

Check Memory Usage:

Use free or htop to check memory usage.

```
free -m
```

Install and run htop

```
sudo apt install htop   # For Debian/Ubuntu
```

```
sudo yum install htop   # For Red Hat/CentOS
```

```
htop
```

CPU Performance:

Monitor CPU usage with top or htop.

```
top
```

Or install and run htop

Review and Disable Unnecessary Services:

Identify and disable unnecessary services.

List all services

```
systemctl list-unit-files --type=service
```

 # Disable a service

```
sudo systemctl disable servicename
```

Optimize Startup Applications:

Review and disable unnecessary startup applications.

Check startup applications

```
sudo systemctl list-unit-files --type=service | grep enabled
```

Check for Failed Services:

Identify and address any failed services.

```
systemctl --failed
```

Network Configuration:

Optimize network settings and use tools like iperf to measure network performance.

```
systemctl --failed
```

```
sudo apt install iperf   # For Debian/Ubuntu
```

```
sudo yum install iperf   # For Red Hat/CentOS
```

Measure network performance

```
iperf -c server_ip
```

Check for Open Ports:

Review open ports to ensure only necessary services are accessible.

```
ss -tuln
```

Kernel Tuning:

Adjust kernel parameters if needed.

View current kernel parameters

```
sysctl -a
```

Edit sysctl.conf to set parameters

```
sudo vi /etc/sysctl.conf
```

Example sysctl.conf entries:

```
# Disable IPv6
net.ipv6.conf.all.disable_ipv6 = 1

# Increase file descriptors limit
fs.file-max = 65535
```

Apply changes with

```
sudo sysctl -p
```

These commands provide a starting point for optimizing a Linux system. Depending on your specific use case and distribution, additional steps may be necessary. Always test changes in a controlled environment before applying them to a production system.

Tools 4 optimization

There are several tools available for optimizing and monitoring Linux systems. Here are some commonly used tools:

htop:

An interactive process viewer and system monitor.

```
sudo apt install htop   # For Debian/Ubuntu
```

```
sudo yum install htop   # For Red Hat/CentOS
```

```
htop
```

iotop:

A top-like utility for displaying I/O usage of processes.

```
sudo apt install iotop   # For Debian/Ubuntu
```

```
sudo yum install iotop   # For Red Hat/CentOS
```

```
iotop
```

iftop:

A real-time console-based network bandwidth monitoring tool.

```
sudo apt install iftop   # For Debian/Ubuntu
```

```
sudo yum install iftop   # For Red Hat/CentOS
```

```
iftop
```

NetHogs:

A small 'net top' tool that shows the bandwidth used by individual processes.

```
sudo apt install nethogs   # For Debian/Ubuntu
```

```
sudo yum install nethogs   # For Red Hat/CentOS
```

```
sudo nethogs
```

vmstat:

A system monitoring tool that provides information about processes, memory, paging, block I/O, traps, and CPU activity.

```
vmstat 1   # Updates every 1 second
```

sar:

Collects, reports, or saves system activity information.

```
sudo apt install sysstat   # For Debian/Ubuntu
```

```
sudo yum install sysstat   # For Red Hat/CentOS
```

```
sar
```

iostat:

Reports CPU statistics and input/output statistics for devices, partitions, and network filesystems.

```
iostat
```

Linux Kernel optimize Parameters for Desktop Systems

Optimizing the Linux kernel for a desktop system involves adjusting various parameters to enhance responsiveness, reduce latency, and improve overall performance. Here are some kernel parameters that you can consider tuning for a desktop system. As always, it's recommended to test changes in a controlled environment before applying them to a production system.

Swappiness:

Adjust the swappiness parameter to control the tendency of the kernel to move processes out of physical memory and onto the swap space.

```
sudo sysctl vm.swappiness=10
```

Dirty Ratio and Background Ratio:

Adjust the dirty ratio and dirty background ratio to control how much memory the kernel can use for caching data before writing it to disk.

```
sudo sysctl vm.dirty_ratio=10
```

```
sudo sysctl vm.dirty_background_ratio=5
```

I/O Scheduler:

Choose an appropriate I/O scheduler for your disk. For desktop systems with traditional hard drives, the cfq scheduler is often recommended.

```
echo cfq | sudo tee /sys/block/sdX/queue/scheduler
```

Replace sdX with your actual disk identifier.

IRQ Balance:

Disable IRQ balance for desktop systems to prevent automatic distribution of interrupts.

```
sudo systemctl disable irqbalance
```

Preemption Model:

Choose a preemption model that suits desktop responsiveness. For low-latency desktop systems, you might prefer the desktop preemption model.

```
echo desktop | sudo tee
/sys/kernel/mm/transparent_hugepage/enabled
```

Timer Frequency:

Adjust the timer frequency to improve desktop responsiveness.

```
echo 1000 | sudo tee /sys/class/timer/timer0/hertz
```

Graphics Performance:

Depending on your graphics card and driver, you may need to adjust settings for optimal performance. For example, for NVIDIA graphics cards, consider using the proprietary NVIDIA driver and adjusting settings using the NVIDIA X Server Settings tool.

CPU Governor:

Set the CPU governor to a performance mode to ensure that the CPU runs at higher frequencies for better responsiveness.

```
echo performance | sudo tee
/sys/devices/system/cpu/cpu*/cpufreq/scaling_governor
```

Transparent Huge Pages:

Disable Transparent Huge Pages to reduce memory management overhead.

```
echo never | sudo tee
```

```
/sys/kernel/mm/transparent_hugepage/enabled
Swapiness:
```

Set a lower value for swappiness to reduce swap usage.

```
sudo sysctl vm.swappiness=10
```

NMI Watchdog:

Disable the NMI watchdog.

```
echo 0 | sudo tee /proc/sys/kernel/nmi_watchdog
```

Make sure to adapt these parameters based on your specific hardware and use case. It's crucial to test changes in a non-production environment to ensure they improve performance without causing instability. Additionally, the availability and applicability of some parameters may vary depending on your Linux distribution and kernel version.

Linux Kernel optimize Parameters for Embeded Devices

Optimizing the Linux kernel for embedded devices involves adjusting parameters to achieve efficient resource utilization and enhance performance within the constraints of embedded systems. Here are some considerations for optimizing the Linux kernel on embedded devices:

Kernel Configuration:

Configure the kernel to include only necessary modules and features for your embedded device. Use a minimal configuration to reduce the kernel size.
Use the make menuconfig or make xconfig commands to configure the kernel.
Kernel Compression:

Choose an appropriate kernel compression method to reduce the size of the kernel image. Options include gzip, bzip2, and LZMA.
Configure this during the kernel build process.

Kernel Command Line:

Optimize the kernel command line parameters based on the requirements of your embedded device. Remove unnecessary options and set parameters like console settings and memory allocation.

I/O Scheduler:

Choose the I/O scheduler that best suits the characteristics of your storage medium. For embedded systems, the noop scheduler is often suitable for flash-based storage.

```
echo noop > /sys/block/mmcblk0/queue/scheduler
```

CPU Governor:

Set the CPU governor to a power-efficient mode suitable for embedded systems. Options include powersave or ondemand for conserving power.

```
echo powersave >
/sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

Memory Settings:

Adjust memory-related parameters to optimize for the limited memory available on embedded devices.

```
sudo sysctl -w vm.swappiness=1
```

Transparent Huge Pages:

Disable Transparent Huge Pages to save memory.

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

Network Settings:

Configure network settings based on the specific requirements of your embedded device. Disable unnecessary network protocols and features.
Adjust TCP parameters to optimize network performance as needed.
Timer Frequency:

Adjust the timer frequency to balance power consumption and responsiveness.

```
echo 100 > /sys/class/rtc/rtc0/max_user_freq
```

Console Output:

Optimize console output for embedded devices. Reduce verbosity and use a minimal console configuration.

```
quiet console=ttyS0,115200
```

Device Drivers:

Only include necessary device drivers for your embedded hardware. Disable unnecessary drivers to reduce kernel size.
Optimize driver configurations to match the specific requirements of your embedded system.

Power Management:

Enable power management features suitable for your embedded device, such as CPU frequency scaling, suspend/resume support, and other power-saving options.

File System:

Choose a file system optimized for flash storage, such as UBIFS or JFFS2 for NAND-based storage, or F2FS for NAND and NOR flash.

Security Features:

Disable security features that may not be necessary for your embedded use case to reduce overhead.

When optimizing the Linux kernel for embedded devices, it's essential to carefully consider the specific requirements and constraints of the target hardware. Additionally, thorough testing in a controlled environment is crucial to ensure stability and functionality. The exact parameters and options may vary depending on the specific Linux distribution and kernel version used in your embedded system.

Optimize Linux with tuned

The tuned utility in Linux is designed to automatically adapt the system's settings based on the chosen profile. It can help optimize performance, power consumption, or a balance between the two. Here's a basic guide on how to use tuned to optimize a Linux system:

Install tuned:

```
sudo apt-get install tuned   # For Debian/Ubuntu
```

```
sudo yum install tuned   # For Red Hat/CentOS
```

Start and Enable tuned:

```
sudo systemctl start tuned
```

```
sudo systemctl enable tuned
```

List Available Profiles:

```
tuned-adm list
```

Choose a Profile:

Select an appropriate profile based on your system's use case. For example:

```
sudo tuned-adm profile throughput-performance   # For
performance
```

```
sudo tuned-adm profile powersave             # For power saving
```

```
sudo tuned-adm profile latency-performance   # For low-latency
```

View Active Profile:

```
tuned-adm active
```

This command will show you the currently active profile.

```
Customize a Profile (Optional):
```

You can create or modify a profile based on your specific needs. Profiles are stored in the **/etc/tuned/** directory.

```
sudo vi /etc/tuned/my_custom_profile
```

Edit the file according to your requirements. For example, you can adjust kernel parameters, disk I/O settings, CPU frequency scaling, etc.

Apply Custom Profile:

```
sudo tuned-adm profile my_custom_profile
```

! Replace my_custom_profile with the name of your custom profile.

Monitor tuned:

Check the status of tuned and monitor its activity.

```
sudo systemctl status tuned
```

```
sudo journalctl -u tuned
```

Ensure there are no errors or warnings related to tuned.

```
Disable tuned (Optional):
```

If you want to return to the default settings without tuned, you can disable it.

```
sudo systemctl stop tuned
```

```
sudo systemctl disable tuned
```

Keep in mind that tuned is most effective when the system has variable workloads or power requirements. For static or specialized environments, manual tuning might be more appropriate.

Always test any changes, especially custom profiles, in a controlled environment before applying them to production systems. Additionally, make sure to consult the documentation of your specific Linux distribution, as tuned behavior may vary.

Systemd optimization for Embeded Linux

Optimizing systemd for embedded Linux systems involves configuring its various components to minimize resource usage, enhance boot time, and tailor the system to the specific needs of embedded devices. Here are some tips for optimizing systemd on embedded Linux:

Minimize Services:

Disable unnecessary services that are not required for the embedded system's functionality. Use the systemctl command to list and disable services.

List all services

```
sudo systemctl list-unit-files --type=service
```

Disable a service

```
systemctl disable servicename
```

Reduce Startup Time:

Analyze and optimize the startup time using systemd-analyze. Identify critical paths and consider adjusting dependencies.

Analyze startup time

```
sudo systemd-analyze blame
```

Parallelize Service Startup:

Use the DefaultDependencies=no option in service unit files to parallelize service startup, speeding up the boot process.

```
[Unit]
Description=Your Service
After=network.target
Wants=network.target
DefaultDependencies=no
```

Use tmpfs for Temporary Data:

Move temporary data to tmpfs (RAM-based filesystem) to reduce disk I/O and improve performance.

```
[Service]
ExecStartPre=/bin/mount -t tmpfs tmpfs /your/tmpfs/directory
ExecStart=/your/executable
ExecStopPost=/bin/umount /your/tmpfs/directory
```

Disable Journaling or Optimize Journal Configuration:

Disable journaling or optimize the journal configuration to reduce disk I/O.

```
[Journal]
```

```
Storage=volatile
RuntimeMaxUse=30M
```

```
Optimize Timer Accuracy:
```

Improve timer accuracy by configuring timer_slack_nsec in the [Manager] section of system.conf.

```
[Manager]
TimerSlackNSec=50000
```

Minimize Logging:

Reduce the verbosity of logs to minimize disk I/O and storage usage. Adjust the LogLevel in journald.conf.

```
[Journal]
LogLevel=info
```

Use systemd-readahead:

systemd-readahead can be used to optimize the system's disk I/O during boot. This tool reads files sequentially into memory to reduce seek time.

```
sudo systemctl enable systemd-readahead-collect.service
```

```
sudo systemctl enable systemd-readahead-replay.service
```

Optimize Swap Usage:

Adjust swap settings in systemd-swap to optimize memory usage.

```
[Swap]
SwapFC=1
```

Enable and start systemd-swap service:

```
sudo systemctl enable systemd-swap
```

```
sudo systemctl start systemd-swap
```

Kernel Parameters:

Set kernel parameters using the sysctl command to optimize resource usage. These settings can be placed in /etc/sysctl.conf.

```
sudo sysctl -w kernel.param=value
```

Optimize Network Services:

Disable unnecessary network services or daemons that are not required for the embedded system.

Use systemd-nspawn for Containerization (Optional):

systemd-nspawn can be used to run services in containers, isolating them from the host system and potentially improving security and resource utilization. These are general guidelines, and the specific optimizations may vary depending on your embedded Linux distribution and hardware. Always test optimizations in a controlled environment before deploying them to production systems. Refer to the official systemd documentation and the documentation for your specific Linux distribution for more detailed information.

Optimize opensuse in Desktop

Optimizing openSUSE for desktop use involves tweaking various settings to improve performance, responsiveness, and user experience. Here are some general optimization tips for openSUSE desktop:

Update the System:

Ensure your system is up-to-date to benefit from the latest software updates, bug fixes, and security patches.

```
sudo zypper refresh
```

```
sudo zypper update
```

Graphics Drivers:

Install proper graphics drivers, especially if you have dedicated graphics cards. For NVIDIA, use the official repository:

```
sudo zypper addrepo --refresh
https://download.nvidia.com/opensuse/leap/<version> NVIDIA
sudo zypper install-new-recommends
```

Desktop Environment Tweaks:

KDE Plasma:

Animations: Adjust animation settings in System Settings > Desktop Behavior > Desktop Effects.
Desktop Effects: Disable effects that may impact performance if needed.

GNOME:

Extensions: Manage GNOME Shell extensions carefully. Disable unnecessary ones to improve performance.

Optimize Boot Time:

Identify and disable unnecessary services to speed up boot time.

```
sudo systemctl list-unit-files --state=enabled
```

```
sudo systemctl disable <service_name>
```

```
Reduce Swappiness:
```

Modify swappiness to control how often the system swaps to disk.

```
# Edit /etc/sysctl.conf and add or modify the following line
vm.swappiness = 10
# Apply the changes
sudo sysctl -p
```


```
# Edit /etc/sysctl.conf and add or modify the following line
vm.swappiness = 10
# Apply the changes
sudo sysctl -p
```


Optimize Disk I/O:

If using an SSD, enable TRIM for better performance and longevity.

```
sudo systemctl enable fstrim.timer
```

```
sudo systemctl start fstrim.timer
```

Manage Startup Applications:

Review and disable unnecessary startup applications in your desktop environment settings.

Enable Firewalld:

Ensure the firewall is active for improved security.

```
sudo systemctl enable firewalld
```

```
sudo systemctl start firewalld
```

Package Cleanup:

Remove unnecessary packages and old kernels.

```
sudo zypper packages --old
```

```
sudo zypper remove <package_name>
```

Optimize Fonts:

Adjust font settings and install additional fonts if needed.

```
sudo zypper install google-roboto-fonts
```

Desktop Search:

Adjust or disable desktop search services like Baloo (KDE) or Tracker (GNOME) if they impact performance.

Adjust Power Settings:

Configure power settings to balance performance and energy efficiency

KDE: System Settings > Power Management
GNOME: Settings > Power

Additional Repositories:

Consider adding additional repositories for access to a broader range of software.

openSUSE Leap

```
sudo zypper addrepo --refresh https://ftp.gwdg.de/pub/opensuse/repositories/<repository_path>/repo/oss repo_name
```

openSUSE Tumbleweed

```
sudo zypper addrepo --refresh https://download.opensuse.org/repositories/<repository_path>/repo/oss repo_name
```

Performance Monitoring:

Use tools like htop or gnome-system-monitor to monitor system resources and identify resource-hungry processes.

```
Desktop Theme:
```

Choose a lightweight desktop theme for better performance.

Btrfs Filesystem (If Used):

If using Btrfs, consider enabling compression for better disk space utilization.

```
sudo btrfs filesystem defragment -r -v /
```

Linux optimization involves various tools and techniques to enhance the performance, security, and efficiency of a Linux system. Here are some common tools and techniques for optimizing a Linux system:

1. Package Management:
   - Keep the system and software packages up-to-date using the package manager (`apt`, `yum`, `zypper`, etc.).
   - Remove unnecessary packages to reduce the system's footprint.

2. Performance Monitoring Tools:
   - Use tools like `top`, `htop`, and `iotop` to monitor system resource usage (CPU, memory, disk I/O).
   - `sar` and `vmstat` provide detailed system statistics over time.

3. Filesystem Optimization:
   - Use `df` and `du` to analyze disk space usage.
   - Consider using the `noatime` mount option in `/etc/fstab` to reduce disk writes for file access time.
   - Optimize disk performance with tools like `fstrim` for SSDs.

4. Kernel Tuning:
   - Adjust kernel parameters using `sysctl` to optimize network and system settings.
   - Review and modify the `/etc/security/limits.conf` file for process resource limits.

5. CPU and Memory Management:
   - Adjust CPU scaling governor and frequency using tools like `cpufrequtils`.
   - Optimize memory usage with tools like `sysstat` and `smem`.

6. Swap Optimization:
   - Tune the swap space usage and priority using `swappiness` in `/etc/sysctl.conf`.
   - Monitor swap usage with tools like `free` and `swapon`.

7. Network Optimization:
 - Optimize network performance using `ethtool`.
 - Adjust network parameters using `sysctl` and `/etc/security/limits.conf`.

8. Security Hardening:
 - Regularly update and patch the system to address security vulnerabilities.
 - Configure and use a firewall (e.g., `iptables`, `ufw`) to secure network traffic.

9. Service Optimization:
 - Review and disable unnecessary services and daemons.
 - Optimize and tune specific services like Apache, Nginx, MySQL, etc.

10. Log Management:
    - Configure log rotation (`logrotate`) to manage log files efficiently.
    - Analyze logs using tools like `journalctl`, `dmesg`, and `tail` for troubleshooting.

11. Automation and Scripting:
    - Automate routine tasks using scripts and tools like `cron`.
    - Optimize startup processes and services using `systemd`.

12. Hardware and Driver Optimization:
    - Use updated and optimized device drivers for hardware components.
    - Monitor hardware health using tools like `smartctl` for hard drives.

13. Application Optimization:
    - Optimize application configurations for performance.
    - Use caching mechanisms where applicable (e.g., Varnish, Memcached).

14. Containerization:
    - Consider containerization with tools like Docker or Podman for efficient resource utilization and isolation.

Remember that optimizations should be approached carefully, and changes should be tested in a controlled environment before applying them to a production system. Always keep backups and documentation of configurations. The specific tools and techniques may vary based on the Linux distribution and system requirements.