

Daemon para Monitoreo de Temperatura del CPU

Cristhofer Daniel Azofeifa Ureña
Ingeniería en Computadores
Instituto Tecnológico de Costa Rica
 Cartago, Costa Rica
 azofeifacris4@estudiantec.cr

María Nicole Valverde Jiménez
Ingeniería en Computadores
Instituto Tecnológico de Costa Rica
 Cartago, Costa Rica
 marvalverde@estudiantec.cr

Abstract—Monitoring the temperature of a computer's CPU is essential to ensure system stability and prevent hardware damage caused by overheating. This project presents the design and implementation of a Linux daemon developed in C++ that continuously reads the CPU temperature and triggers user notifications when predefined temperature thresholds are exceeded. The daemon operates as a systemd service, facilitating its management and integration within the operating system. Installation and configuration are streamlined through CMake scripts, allowing easy deployment and control of the service lifecycle. The solution aims to provide a lightweight and effective tool for real-time thermal monitoring, suitable for various Linux environments where hardware temperature awareness is critical.

Palabras clave—CPU, daemon, Linux, systemd, C++, CMake.

I. INTRODUCCIÓN

La temperatura del procesador es uno de los parámetros más importantes para el correcto funcionamiento y la vida útil de un sistema computacional. Un aumento excesivo en la temperatura puede provocar fallos o daños en el hardware, por lo que es necesario monitorearla constantemente. En este proyecto se desarrolló un daemon [1] en Linux que permite supervisar la temperatura del CPU y enviar alertas cuando esta supera un umbral definido, ayudando a prevenir posibles problemas.

El programa está implementado en C++ y utiliza recursos propios del sistema operativo, como la lectura de archivos en `/sys/class/thermal/` para obtener la temperatura actual. Además, el daemon se configura y gestiona a través de un servicio systemd, que se instala y controla mediante CMake para facilitar su uso e integración en el sistema. Para notificar al usuario, el proyecto utiliza notify-send, que muestra alertas gráficas cuando la temperatura está alta.

Este trabajo busca ofrecer una solución simple pero efectiva para el monitoreo térmico en sistemas Linux, enfocándose en la facilidad de instalación, configuración automática y funcionamiento continuo en segundo plano. A lo largo del documento se explica la arquitectura del sistema, las herramientas usadas y los pasos para su instalación y uso.

II. AMBIENTE DE DESARROLLO

Este proyecto se desarrolló en un entorno Linux, específicamente en la distribución Ubuntu. Para el desarrollo del mismo se utilizaron las siguientes herramientas:

- C++11: Lenguaje que se usó para la implementación de la lógica del daemon para monitoreo de temperatura y el código Dummy para estresar el CPU.

- CMake: Herramienta de automatización para la construcción del proyecto. Este se utilizó para facilitar las tareas de compilar, instalar y gestionar el funcionamiento del daemon.
- Systemd: Administrador de sistemas y servicios en Linux que permitió la gestión del ciclo de vida del daemon [2]
- Notify-send: Esta herramienta permite mostrar notificaciones gráficas y se utilizó para alertar al usuario cuando la temperatura del CPU supera el umbral establecido.
- Syslog: Es el API estandar de Linux para registrar eventos. El daemon utiliza syslog() para registrar tanto información como advertencias sobre la temperatura.

III. ATRIBUTOS

Durante el desarrollo del proyecto, fue necesario adquirir nuevos conocimientos relacionados principalmente con la creación y gestión de daemons [1] en sistemas basados en Linux utilizando systemd. Esto incluyó entender cómo estructurar correctamente un archivo `.service`, conocer los comandos necesarios para habilitar y controlar servicios con `systemctl`, así como aprender buenas prácticas para que el daemon funcione correctamente al inicio del sistema. Asimismo, se profundizó en cómo implementar un proceso en segundo plano en C++, controlar sus salidas y asegurar su ejecución estable como parte del sistema operativo.

Las tecnologías utilizadas para el desarrollo de este proyecto, como las bibliotecas, herramientas de compilación y servicios del sistema, se explican con mayor detalle en la sección correspondiente de Ambiente de Desarrollo.

En cuanto a la organización del trabajo, el equipo implementó un enfoque basado en reuniones periódicas donde se planificaban los avances, se distribuían las responsabilidades y se discutían posibles soluciones a los retos que iban surgiendo. Durante las primeras etapas del proyecto, se priorizó la investigación y para asegurar una comprensión adecuada del contexto. Una vez establecida esta base, se procedió a la fase de implementación, donde cada integrante desarrolló partes del código según lo acordado previamente.

Desde una perspectiva crítica, las acciones implementadas fueron en general efectivas, permitiendo avanzar de forma ordenada en el cumplimiento de los objetivos del proyecto. El enfoque inicial de investigación facilitó una implementación más fluida en etapas posteriores. Sin embargo, se reconocen oportunidades de mejora en la planificación detallada y en la estimación de tiempos por tarea, lo cual podría haber optimizado aún más la gestión del tiempo y anticipado posibles

obstáculos técnicos. A pesar de ello, el aprendizaje autónomo y la colaboración constante entre los integrantes del equipo fueron fundamentales para enfrentar con éxito los desafíos que propuso el proyecto.

IV. DISEÑO DEL PROGRAMA

La figura 1 se puede observar el diagrama de arquitectura en el cual se tiene el sistema operativo linux. En él se tiene el programa systemd, este se encarga de ejecutar de manera automática el daemon al iniciar la computadora, además ayuda con el inicio, detención y reinicio de este programa. Cuando se cumple con una temperatura mayor a la establecida entonces se procede a ejecutar el notificador. Además, en el OS se tiene un programa Dummy el cual se esta ejecutando.

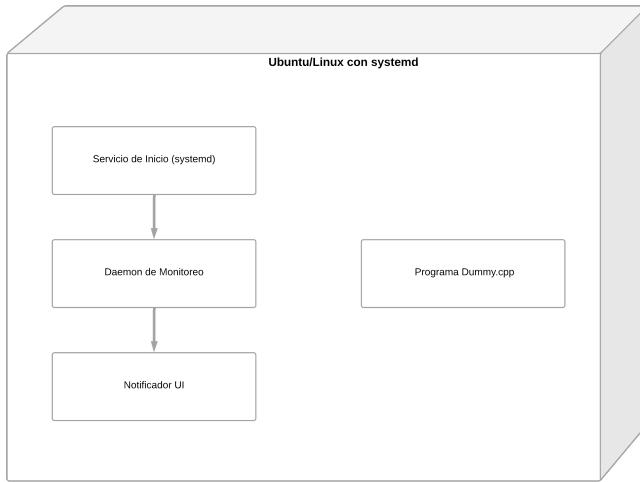


Figure 1. Diagrama de Arquitectura

En la figura 2 se puede observar el diagrama de flujo que se tiene en el proyecto se inicia teniendo al daemon temp_monitor limpiando el archivo de log, configurando el syslog como sistema de registro y obteniendo UID del usuario para establecer correctamente el canal de comunicación con D-Bus. Seguidamente, en el programa se lee y se escribe la temperatura del CPU. Si la temperatura supera el umbral de 65°C, el daemon genera una advertencia en el log del sistema y muestra una notificación visual al usuario. Finalmente, el proceso espera 5 segundos antes de repetir el ciclo, lo que permite un monitoreo continuo en segundo plano sin interferir con el uso del sistema.

En la figura 3 el sistema está compuesto por dos programas principales. El primero es el daemon TempMonitor, diseñado para ejecutarse en segundo plano, el cual monitorea continuamente la temperatura del CPU en Linux. Si la temperatura supera un umbral predefinido (por ejemplo, 65°C), el daemon registra la información en un archivo de log y en el syslog, y lanza una notificación al usuario. El segundo componente es el programa Dummy, una aplicación que simula una carga intensiva sobre el CPU mediante operaciones matemáticas costosas (uso de funciones trigonométricas en un bucle muy grande). Este programa también muestra la temperatura en

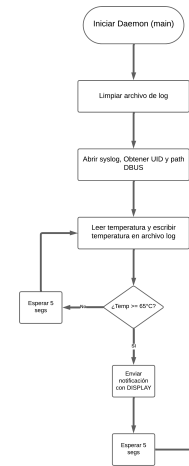


Figure 2. Diagrama de Flujo

tiempo real mientras ejecuta la carga, lo que permite probar el comportamiento del daemon bajo condiciones de estrés.

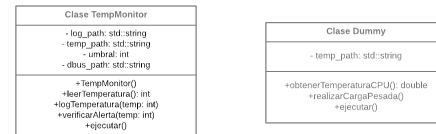


Figure 3. Diagrama de de Clases UML

V. INSTRUCCIONES DE CÓMO SE UTILIZA EL PROYECTO

Este proyecto implementa un daemon en C++ que monitorea la temperatura del CPU usando /sys/class/thermal/thermal_zone0/temp. Si se supera un umbral de temperatura (por defecto 65°C), el sistema registra la alerta y, si es posible, lanza una notificación gráfica mediante notify-send.

El proyecto utiliza CMake para la compilación y configuración automática del servicio systemd, que se instala en /usr/bin y se habilita como daemon del sistema.

A. Requisitos

Para ejecutar el programa se necesitan los siguientes elementos:

- CMake ≥ 3.10
- g++ (compilador C++)
- systemd
- notify-send
- Permisos de superusuario (sudo)

B. Pasos de instalación y ejecución

1) Clonar repositorio:

```
git clone https://github.com/CristhoferAU/T2_Daemons
```

2) Configurar archivo temp_notice.service:

USER = (cambiar nombre de usuario)

3) Compilar proyecto:

```
mkdir build
cd build
cmake ..
sudo make install
sudo make enable_service
```

4) Comandos disponibles con CMake

```
sudo make install # Compila y copia el
                    binario del daemon a la ruta
                    /usr/bin, adems de instalar el
                    archivo temp_monitor.service en
                    /etc/systemd/system/.
```

```
sudo make enable_service # Habilita e
                          inicia el servicio para que se
                          ejecute automaticamente al iniciar
                          el sistema.
```

```
sudo make stop_service # Detiene el
                        servicio actualmente en ejecucin.
```

```
sudo make restart_service # Reinicia
                           el servicio y limpia el archivo de
                           logs.
```

```
sudo make disable_service #
                           Deshabilita el servicio para que no
                           se inicie automaticamente con el
                           sistema.
```

```
sudo make uninstall_service # Detiene,
                              deshabilita y elimina completamente
                              el servicio, el binario y el
                              archivo de logs.
```

5) Comando disponibles para observar estados del daemon:

```
systemctl list-units --type=service #
Muestra los daemons activados
systemctl list-units --type=all #
Muestra todos los daemons
```

VI. RESULTADOS

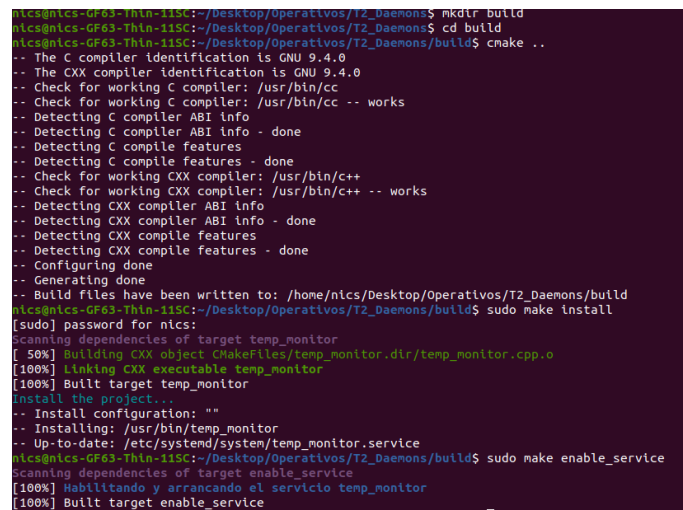
En la figura 4 se puede observar que se ingresaron los comandos para que se realizará la compilación e instalación del CMake. Este permite que se crea el binario y el archivo .service en el sistema. Además, esto permite que se pueda ejecutar automáticamente el servicio en segundo plano. [3]

En la 5 se puede ver como el archivo .service esta creado y se está ejecutando en segundo plano.

Por otro lado, en la figura 6 se puede ver un ejemplo de cómo se detiene el daemon usando el comando,

```
sudo make stop_service
```

y ya en la figura 7 se puede ver que este ya sale inactivado en el sistema al usar el comando



```
nics@nics-GF63-Thin-11SC:~/Desktop/Operativos/T2_Daemons$ mkdir build
nics@nics-GF63-Thin-11SC:~/Desktop/Operativos/T2_Daemons$ cd build
nics@nics-GF63-Thin-11SC:~/Desktop/Operativos/T2_Daemons/build$ cmake ..
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
-- Check for working C compiler: /usr/bin/cc
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/nics/Desktop/Operativos/T2_Daemons/build
nics@nics-GF63-Thin-11SC:~/Desktop/Operativos/T2_Daemons/build$ sudo make install
[sudo] password for nics:
Scanning dependencies of target temp_monitor
[ 50%] Building CXX object CMakeFiles/temp_monitor.dir/temp_monitor.cpp.o
[100%] Linking CXX executable temp_monitor
[100%] Built target temp_monitor
Install the project...
-- Install configuration: ""
-- Installing: /usr/bin/temp_monitor
-- Up-to-date: /etc/systemd/system/temp_monitor.service
nics@nics-GF63-Thin-11SC:~/Desktop/Operativos/T2_Daemons/build$ sudo make enable_service
Scanning dependencies of target enable_service
[100%] Habilitando y arrancando el servicio temp_monitor
[100%] Built target enable_service
```

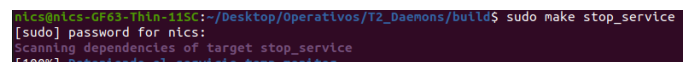
Figure 4. Compilaciones e instalación de CMake



```
systemd-user-sessions.service loaded active exited Permit User Sessions
temp_monitor.service loaded active running Daemon para monitorear temperatura CPU y notificar
thermald.service loaded active running Thermal Daemon Service
```

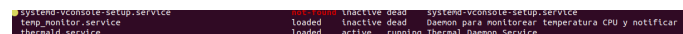
Figure 5. Daemon activado

```
systemctl list-units --type=all
```



```
nics@nics-GF63-Thin-11SC:~/Desktop/Operativos/T2_Daemons/build$ sudo make stop_service
[sudo] password for nics:
Scanning dependencies of target stop_service
[100%] Deteniendo el servicio temp_monitor
```

Figure 6. Deteniendo Daemon



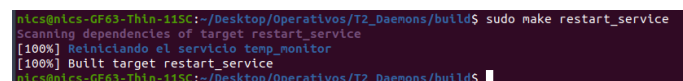
```
systemd-user-sessions.service loaded active exited Permit User Sessions
temp_monitor.service loaded inactive dead Daemon para monitorear temperatura CPU y notificar
thermald.service loaded active running Thermal Daemon Service
```

Figure 7. Daemon detenido

Sin embargo, se puede volver a reiniciar el daemon usando el siguiente comando

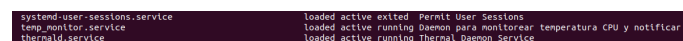
```
sudo make restart_service
```

como se ve en la figura 8. Ya en la figura 9 se puede ver cómo este ya está de nuevo activado.



```
nics@nics-GF63-Thin-11SC:~/Desktop/Operativos/T2_Daemons/build$ sudo make restart_service
Scanning dependencies of target restart_service
[100%] Reiniciando el servicio temp_monitor
[100%] Built target restart_service
nics@nics-GF63-Thin-11SC:~/Desktop/Operativos/T2_Daemons/build$
```

Figure 8. Reiniciando Daemon



```
systemd-user-sessions.service loaded active exited Permit User Sessions
temp_monitor.service loaded active running Daemon para monitorear temperatura CPU y notificar
thermald.service loaded active running Thermal Daemon Service
```

Figure 9. Daemon reiniciado

Finalmente, en la figura 10 se puede ver un ejemplo de cómo el sistema envía el mensaje cuando la temperatura del CPU llega al umbral.

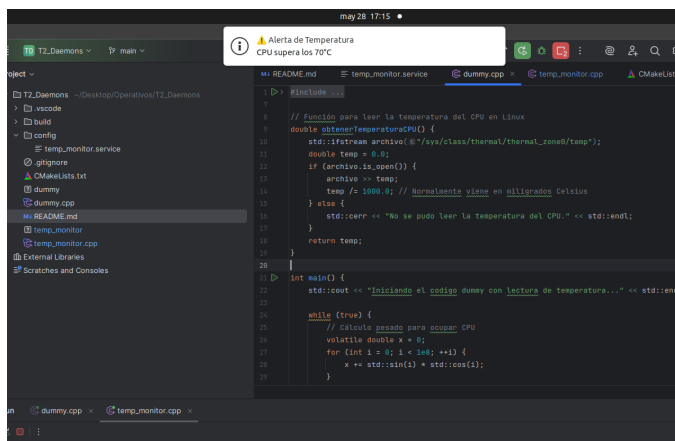


Figure 10. Ejemplo de advertencia dada por el sistema dado que se llegó a la temperatura umbral

VII. LINK DEL VIDEO

<https://youtu.be/xA4QAmhGomU>

REFERENCES

- [1] D. Watson, *Linux Daemon Writing HOWTO*, 2004. [Online]. Available: https://homes.cs.aau.dk/~adavid/teaching/MTP-05/exercises/10/Linux-Daemon_Writing.pdf
- [2] systemd Project, *systemd - System and Service Manager*, 2025. [Online]. Available: <https://systemd.io/>
- [3] A. Baderouaich, *daemonpp: Modern C++ Daemon Library*, 2021. [Online]. Available: <https://github.com/baderouaich/daemonpp>