BÁO CÁO ĐỀ TÀI:

Đóng gói Linux service vào .deb package

Người thực hiện: Nguyễn Trọng Thiện

Email: trongthien.work@gmail.com

Mục Lục

I. Giới thiệu

1. Nội dung chính

- Ý tưởng chính của đề tài này là xây dựng một Linux service có nhiệm vụ giám sát tài nguyên hệ thống Linux bao gồm các thông tin về CPU, RAM và disk. Các thông tin này sẽ được ghi nhật ký vào một file log. Toàn bộ thư viện và service được xây dựng sẽ được đóng gói vào một .deb package để có thể dễ dàng cài đặt, triển khai cũng như gỡ bỏ.
- Mô tả sản phẩm: Linux service có tên là **system-logger** được đóng gói vào .deb package. Sau khi cài đặt, service này tự động chạy khi hệ thống khởi động, tự động ghi lại nhật ký thông tin về mức độ sử dụng tài nguyên của hệ thống vào địa chỉ **/var/log/system-usage.log**.

2. Mục tiêu

Mục tiêu của đề tài bao gồm:

- Nắm được cách Shared Library hoạt động: Hiểu rõ cách thức hoạt động của thư viện chia sẻ trong Linux, bao gồm cách tạo, sử dụng và quản lý các thư viện chia sẻ, giúp phần mềm có thể tái sử dụng mã nguồn và giảm dung lượng tổng thể.
- Hiểu được cách dùng Makefile: Sử dụng Makefile để tự động hóa quá trình biên dịch và liên kết mã nguồn thành các file thực thi và thư viện chia sẻ, giúp tăng hiệu suất làm việc và đảm bảo tính nhất quán của quá trình build.
- Nắm được cách đăng ký Linux service: Tìm hiểu và áp dụng cách tạo và quản lý các dịch vụ hệ thống trên Linux bằng cách sử dụng systemd, giúp phần mềm có thể chạy tự động khi hệ thống khởi động và quản lý các trạng thái của dịch vụ.
- Nắm được cách đóng gói .deb package: Giới thiệu quy trình và các công cụ cần thiết để tạo và quản lý các .deb package, giúp phần mềm có thể dễ dàng cài đặt, nâng cấp và gỡ bỏ trên các hệ thống Linux sử dụng định dạng .deb.

3. Tầm quan trọng

Việc giám sát tài nguyên hệ thống là một khía cạnh quan trọng trong quản lý hệ thống và vận hành dịch vụ. Nó giúp các quản trị viên hệ thống có thể:

- Theo dõi hiệu suất của các máy chủ và dịch vụ.
- Phát hiện sớm các vấn đề về hiệu suất và tài nguyên.
- Đảm bảo các dịch vụ hoạt động ổn định và hiệu quả.
- Tối ưu hóa việc sử dụng tài nguyên hệ thống.

Bằng cách hiểu và triển khai hệ thống giám sát tài nguyên này, chúng ta không chỉ nắm được các kỹ thuật lập trình cơ bản và nâng cao mà còn phát triển các kỹ năng quan trọng trong quản lý hệ thống và dịch vụ.

4. Tổng quan nội dung báo cáo

Báo cáo này được chia thành các phần chính như sau:

- **Giới thiệu**: Trình bày nội dung và mục tiêu của đề tài.
- **Xây dựng shared library**: Trình bày quá trình xây dựng thư viện chia sẻ.
- **Xây dựng Linux service**: Trình bày quá trình tạo và đăng ký Linux service.
- **Đóng gói .deb package**: Trình bày các bước để đóng gói một .deb package.
- **Kết luận**: Tóm tắt kết quả đạt được và hướng phát triển trong tương lai.

- **Phụ lục**: Mã nguồn, Danh sách tham chiếu

II. Xây dựng Shared Library

1. Ý tưởng và mục đích

- **Ý tưởng chính**: Xây dựng một thư viện chia sẻ (Shared Library) có tên là *linuxSystemMonitor* để thu thập thông tin về CPU, RAM và disk của hệ thống Linux.
- **Mục đích**: Thư viện chia sẻ sẽ cung cấp các hàm và phương thức để truy cập và thu thập các thông tin này một cách hiệu quả. Đây là một phần quan trọng để tái sử dụng mã nguồn, giảm thiểu sự phụ thuộc và tối ưu hóa hiệu năng của hệ thống.

2. Thực hiện

a) Xây dựng File Header

- Đầu tiên, ta có một **File Header**, đây là nơi định nghĩa các Hàm và Cấu trúc dữ liệu cho Shared Library: **"linuxSystemMonitor.hpp"**
- Các Struct được định nghĩa:
    + **CpuStats**: Lưu trữ thông tin về CPU như **user**, **nice**, **system**, **idle**, **ioWait**, **irq**, **softIrq**.
    + **MemoryStats**: Lưu trữ thông tin về bộ nhớ và swap như **totalMemory**, **availableMemory**, **totalSwap**, **freeSwap**.
    + **DiskStats**: Lưu trữ thông tin về ổ đĩa như **totalSpace**, **freeSpace**.

- Các hàm được định nghĩa:
    + **readCpuData()**: Đọc và phân tích dữ liệu từ file **/proc/stat** để lấy thông tin về CPU.
    + **readMemoryData()**: Đọc và phân tích dữ liệu từ file **/proc/meminfo** để lấy thông tin về RAM.
    + **readDiskData()**: Lấy thông tin về disk .
    + **getCpuUsage(const CpuStats& first, const CpuStats& second)**: Tính toán tỷ lệ sử dụng CPU dựa trên hai lần đo.
    + **getMemoryUsage()**: Tính toán tỷ lệ sử dụng RAM.
    + **getSwapUsage()**: Tính toán tỷ lệ sử dụng swap.

b) Xây dựng File Implementation

- Tiếp theo, ta viết **file implementation**, nơi chứa phần thực thi cho các hàm vừa được định nghĩa: **"linuxSystemMonitor.cpp"**
- Tính toán tỷ lệ sử dụng CPU:
    + Dựa trên dữ liệu từ file **/proc/stat:**

```
 1 cpu  560940 636 118488 23098722 15876 0 43529 0 0 0
 2 cpu0 44236 28 9850 1429624 1203 0 38140 0 0 0
 3 cpu1 19854 70 2550 1467659 567 0 2285 0 0 0
 4 cpu2 44572 135 9093 1429760 1205 0 869 0 0 0
 5 cpu3 32245 202 9428 1443827 539 0 165 0 0 0
 6 cpu4 45430 27 11257 1426550 1383 0 643 0 0 0
 7 cpu5 22445 53 2863 1464750 499 0 130 0 0 0
 8 cpu6 44104 7 10859 1429035 1409 0 323 0 0 0
 9 cpu7 21844 6 2748 1465930 591 0 22 0 0 0
10 cpu8 45232 0 9697 1429904 1437 0 33 0 0 0
11 cpu9 29125 0 6082 1447212 921 0 12 0 0 0
12 cpu10 46256 4 10310 1427635 1500 0 42 0 0 0
13 cpu11 22689 0 3232 1463804 728 0 228 0 0 0
14 cpu12 45596 4 11400 1426870 1385 0 607 0 0 0
15 cpu13 21313 15 3254 1465749 594 0 8 0 0 0
16 cpu14 44624 12 10218 1430362 1305 0 16 0 0 0
```

+ Ta sẽ lấy thông tin của CPU từ dòng đầu tiên.
+ 10 thông số theo sau có ý nghĩa:
    - **user**: Thời gian tiến trình người dùng chiếm dụng CPU.
    - **nice**: Thời gian tiến trình nice chiếm dụng CPU.
    - **system**: Thời gian tiến trình kernel chiếm dụng CPU.
    - **idle**: Thời gian CPU không làm việc.
    - **iowait**: Thời gian chờ vào/ra (I/O) chiếm dụng CPU.
    - **irq**: Thời gian xử lý Interrupts (IRQ) chiếm dụng CPU.
    - **softirq**: Thời gian xử lý Soft Interrupts (SoftIRQ) chiếm dụng CPU.
    - **steal**: Thời gian CPU bị chiếm dụng bởi các máy ảo (virtual machines).
    - **guest**: Thời gian CPU dành cho các tiến trình máy ảo khách.
    - **guest_nice**: Thời gian CPU dành cho các tiến trình nice của máy ảo khách.
+ Tính toán được **idleTime** là tổng thời gian CPU không hoạt động: **idleTime = idle + iowait.**
+ Tính toán được **activeTime** là tổng thời gian CPU hoạt động : **activeTime = user + nice + system + irq + softirq + steal + guest + guest_nice**
+ Lưu ý, đây là thống kê về hoạt động của CPU kể từ khi hệ thống khởi động. Vì vậy, để biết mức sử dụng CPU hiện tại, ta phải đo các thông số này 2 lần gần nhau (thường là 1 giây) sau đó tính hiệu của 2 lần đo.

- Tính toán tỷ lệ sử dụng RAM:
    + Dựa trên dữ liệu từ file **/proc/meminfo**:

```
 1 MemTotal:       15751052 kB
 2 MemFree:         6057040 kB
 3 MemAvailable:    8900752 kB
 4 Buffers:          258556 kB
 5 Cached:          2960360 kB
 6 SwapCached:            0 kB
 7 Active:          1152320 kB
 8 Inactive:        6206144 kB
 9 Active(anon):       3104 kB
10 Inactive(anon):  4355520 kB
11 Active(file):    1149216 kB
12 Inactive(file):  1850624 kB
13 Unevictable:          48 kB
14 Mlocked:              48 kB
15 SwapTotal:       2097148 kB
16 SwapFree:        2097148 kB
17 Dirty:                96 kB
18 Writeback:             0 kB
19 AnonPages:       4139132 kB
```

    + Tính toán tỷ lệ sử dụng Memory dựa vào **MemTotal** và **MemFree**
    + Tính toán tỷ lệ sử dụng Swap dựa vào **SwapTotal** và **SwapFree**
- Tính toán tỷ lệ sử dụng Disk:
    + Thông tin này được lấy từ struct **statvfs** trong thư viện **sys/statvfs.h**.
    + Tính toán được freeSpace và totalSpace, ta sẽ tính được tỷ lệ sử dụng Disk.

c) Xây dựng Shared Library

- Để xây dựng thư viện chia sẻ **"liblinuxSystemMonitor.so"**, ta chạy lệnh sau:

  g++ -Wall -Werror -fPIC -linclude -shared -o lib/liblinuxsystemmonitor.so src/linuxSystemMonitor.cpp

- Trong đó:
    + **g++**: Đây là trình biên dịch cho ngôn ngữ C++.
    + **-Wall**: Tùy chọn này bật tất cả các cảnh báo cảnh báo tiêu chuẩn.
    + **-Werror**: Tùy chọn này biến tất cả các cảnh báo thành lỗi.
    + **-fPIC**: Tùy chọn này là viết tắt của "Position Independent Code". Khiến trình biên dịch tạo ra mã máy có thể chạy ở bất kỳ vị trí nào trong bộ nhớ (không phụ thuộc vào vị trí tuyệt đối). Điều này quan trọng khi xây dựng các thư viện chia sẻ (shared libraries), vì mã máy có thể được tải và thực thi từ bất kỳ vị trí nào trong bộ nhớ.
    + **-linclude**: Tùy chọn này chỉ ra thư mục chứa các file header (file .hpp hoặc .h) để trình biên dịch có thể tìm thấy và sử dụng khi biên dịch mã nguồn. Trong trường hợp này, thư mục `include` chứa file `linuxSystemMonitor.hpp` mà mã nguồn cần để sử dụng.
    + **-shared**: Tùy chọn này chỉ ra rằng mã nguồn được biên dịch thành một thư viện chia sẻ (shared library). Shared library là một file thực thi có thể được nạp vào bộ nhớ và dùng chung bởi nhiều chương trình. Đây là định dạng thông dụng để chia sẻ code giữa các ứng dụng.

    + **-o lib/liblinuxsystemmonitor.so**: Tùy chọn này chỉ ra tên và đường dẫn của file đầu ra của chương trình biên dịch. Trong trường hợp này, mã nguồn sẽ được biên dịch thành thư viện chia sẻ có tên là `liblinuxsystemmonitor.so` và được lưu vào thư mục `lib`.
    + **src/linuxSystemMonitor.cpp**: Đây là file mã nguồn đầu vào để biên dịch thành thư viện chia sẻ. Trình biên dịch sẽ sử dụng file này để biên dịch thành mã máy tương ứng.
- Sau khi xây dựng, ta phải đặt file này vào vị trí mà hệ thống có thể tìm được, thường là **usr/lib/**.

d) Sử dụng Makefile

- Makefile giúp ta không cần gõ dòng lệnh bên trên nhiều lần, mỗi lần muốn build lại Shared Library, ta chỉ cần gõ 1 lệnh **“make”**
- Source Code của Makefile sẽ dở phần **Phụ lục.**
- Lợi ích của Makefile:
    + Tự động hóa quy trình xây dựng: Makefile giúp tự động hóa các bước biên dịch và liên kết, tiết kiệm thời gian và giảm thiểu lỗi phát sinh do sự can thiệp của con người.
    + Dễ dàng quản lý và triển khai: Bằng cách sử dụng Makefile, các tập tin mã nguồn và thư viện có thể được tổ chức một cách rõ ràng và dễ dàng quản lý trong dự án.
    + Đảm bảo tính nhất quán: Makefile giúp đảm bảo rằng mọi thành phần của dự án được biên dịch và liên kết đúng cách mỗi khi cần thiết.

III. Xây dựng Linux Service

1. Ý tưởng và Mục đích

- **Ý tưởng chính**: Xây dựng một Linux service có tên là *system-logger* có nhiệm vụ ghi lại nhật ký về mức độ sử dụng các tài nguyên của hệ thống vào một file log. Service sẽ chạy tự động khi hệ thống khởi động và hoạt động định kỳ mà không cần sự can thiệp của người dùng.
- **Mục đích**: Bằng việc tự động hóa quá trình giám sát, có thể nhanh chóng phát hiện và xử lý các vấn đề về tài nguyên như quá tải CPU, RAM quá tải, hoặc sự cạn kiệt không gian đĩa.

2. Thực hiện

a) Tạo file thực thi

- Trước hết, ta cần viết chương trình **linuxSystemLogger.cpp** sử dụng các hàm trong thư viện **linuxSystemMonitor** để mỗi 5 phút, tính toán mức độ sử dụng tài nguyên của hệ thống và ghi lại vào file **/var/log/system-usage.log**.
- Tiếp theo, thực hiện tạo file thực thi (executable file) bằng lệnh:

  g++ -Wall -Werror -fPIC -Iinclude -o bin/linuxSystemLogger src/linuxSystemLogger.cpp -Llib -llinuxsystemmonitor

- Trong đó:
    + **g++**: Đây là trình biên dịch cho ngôn ngữ C++.

    + **-Wall**: Tùy chọn này bật tất cả các cảnh báo cảnh báo tiêu chuẩn.
    + **-Werror**: Tùy chọn này biến tất cả các cảnh báo thành lỗi.
    + **-fPIC**: Tùy chọn này là viết tắt của "Position Independent Code". Khiến trình biên dịch tạo ra mã máy có thể chạy ở bất kỳ vị trí nào trong bộ nhớ (không phụ thuộc vào vị trí tuyệt đối). Điều này quan trọng khi xây dựng các thư viện chia sẻ (shared libraries), vì mã máy có thể được tải và thực thi từ bất kỳ vị trí nào trong bộ nhớ.
    + **-Iinclude:** Tùy chọn này chỉ ra thư mục chứa các file header (file .hpp hoặc .h) để trình biên dịch có thể tìm thấy và sử dụng khi biên dịch mã nguồn. Trong trường hợp này, thư mục include chứa file **linuxSystemMonitor.hpp** mà mã nguồn cần để sử dụng.
    + **-o bin/linuxSystemLogger**: Tùy chọn -o dùng để chỉ định tên của file đầu ra sau khi biên dịch. Trong trường hợp này, tên file đầu ra là **bin/linuxSystemLogger.**
    + **src/linuxSystemLogger.cpp**: Đây là tập tin nguồn C++ mà bạn muốn biên dịch.
    + **-Llib**: Tùy chọn này chỉ định thư mục chứa các thư viện (library) để liên kết. Trong trường hợp này, -Llib cho biết rằng trình liên kết (linker) nên tìm kiếm các thư viện trong thư mục lib.
    + **-llinuxsystemmonitor**: Tùy chọn -l được sử dụng để chỉ định tên của thư viện mà bạn muốn liên kết. Trong trường hợp này, -llinuxsystemmonitor yêu cầu trình liên kết tìm kiếm và liên kết với thư viện **liblinuxsystemmonitor.so**.
- Với câu lệnh này, ta cũng sẽ thực hiện bằng Makefile như cách ta tạo Shared Library.

b) Viết file cấu hình Linux service

- Phần source code được để ở phần **Phụ lục.**
- File cấu hình cho service thường được đặt trong thư mục **`/etc/systemd/system/`** với phần mở rộng **`.service`.**

```
[Unit]
Description=System Logger Service
After=network.target

[Service]
Type=simple
ExecStart=/usr/local/bin/linuxSystemLogger
Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target

```

- **[Unit]**: Phần này mô tả service và xác định các điều kiện cần thiết để service hoạt động.
    + **`Description`**: Mô tả ngắn gọn về service.
    + **`After`**: Xác định các đơn vị khác mà service này cần khởi động sau. Ở đây, service sẽ khởi động sau khi mạng đã sẵn sàng (`network.target`).
- **[Service]**: Phần này chứa các thông tin về cách khởi động, dừng và quản lý service.

    + **`Type=simple`**: Loại service đơn giản, service sẽ được coi là đang chạy khi lệnh `ExecStart` được chạy.
    + **`ExecStart`**: Chỉ định lệnh để khởi động service, trong trường hợp này là `/usr/local/bin/linuxSystemLogger`.
    + **`Restart`**: Định nghĩa chính sách khởi động lại service, ở đây là **`always`** để luôn khởi động lại service nếu nó bị dừng.
    + **`RestartSec`**: Khoảng thời gian chờ trước khi khởi động lại service, ở đây là 5 giây.
- **[Install]**: Phần này xác định khi nào và cách nào service sẽ được khởi động.
    + **`WantedBy`**: Xác định các target mà service sẽ được kích hoạt, ở đây là **`multi-user.target`**, để service hoạt động ở chế độ multi-user.

c) Cấu hình Logrotate

- Bài toán đặt ra là: Ta đang liên tục ghi vào file **system-usage.log**, nếu cứ liên tục ghi với tần suất lớn, sẽ có thể gây ra thừa thãi, lãng phí tài nguyên. Để giải quyết bài toán này, ta dùng Logrotate.
- Logrotate là một công cụ trong hệ điều hành Linux dùng để quản lý log files của các dịch vụ và ứng dụng. Nó tự động xoay, nén, và xóa log files cũ theo các quy tắc được định sẵn. Điều này giúp ngăn ngừa tình trạng log files chiếm quá nhiều không gian đĩa và giữ cho hệ thống của bạn sạch sẽ và có tổ chức.
- Để cấu hình Logrotate cho **system-usage.log**, ta thêm 1 file có tên **system-usage** vào địa chỉ **/etc/logrotate.d/** có nội dung như sau:

```
/var/log/system-usage.log {
    daily
    rotate 7
    missingok
    notifempty
    compress
    create 0644 root root
}
```

- Trong đó:
    + **/var/log/system-usage.log**: Đường dẫn đến log file cần quản lý.
    + **daily**: Xoay log files hàng ngày.
    + **rotate 7**: Giữ lại 7 bản log cũ.
    + **missingok**: Bỏ qua nếu log file không tồn tại.
    + **notifempty**: Không xoay log file nếu nó trống.
    + **compress**: Nén log files cũ.
    + **create 0644 root root**: Tạo log file mới với quyền sở hữu và quyền truy cập xác định.
- Tổng kết: Logrotate là một công cụ mạnh mẽ giúp quản lý log files hiệu quả, đảm bảo rằng hệ thống luôn hoạt động ổn định và log files không chiếm dụng quá nhiều không gian đĩa.

IV. Đóng gói .deb package

1. Chuẩn bị thư mục để đóng gói

- Tạo folder tên là **system-logger.**
- Bên trong, tạo folder **DEBIAN**.
- Đặt những file muốn đóng gói vào bên trong folder với vị trí giống với với vị trí muốn đặt trong hệ thống. Ví dụ nếu muốn đặt file **system-logger.service** vào địa chỉ **/etc/systemd/system/** của hệ thống thì ta sẽ copy file này vào **system-logger/etc/systemd/system/**.
- Những file ta cần copy là:
    + **system-logger/etc/logrotate.d/system-usage**: file cấu hình logrotate.
    + **system-logger/etc/systemd/system/system-logger.service**: file cấu hình service
    + **system-logger/usr/lib/liblinuxsystemmonitor.so**: file Shared Library.
    + **system-logger/usr/local/bin/linuxSystemLogger**: file thực thi.
    + **system-logger/var/log/system-usage.log**: file log.

2. Cấu hình các thông tin của package (Information)

- Trong thư mục **DEBIAN**, viết file **control** cấu hình cho package:

system-logger > DEBIAN > control

```
Package: system-logger
Version: 1.0
Architecture: amd64
Maintainer: Nguyen Trong Thien <trongthien.work@gmail.com>
Description: System Logger Service
 This package provides a system logging service that logs CPU, memory, swap, and disk usage ev
Section: utils
Depends: libc6 (>= 2.28)
License: GPL-3.0
```

- Trong đó:
    + **Package:** Đây là tên của gói phần mềm. Nó định danh duy nhất cho gói phần mềm được mô tả.
    + **Version:** Xác định số phiên bản của gói phần mềm. Nó giúp theo dõi các bản cập nhật và tính tương thích.
    + **Architecture:** Cho biết kiến trúc CPU mà gói phần mềm này được biên dịch. amd64 thường chỉ kiến trúc x86 64-bit.
    + **Maintainer:** Chỉ định người chịu trách nhiệm bảo trì gói phần mềm, bao gồm tên và địa chỉ email của người bảo trì.
    + **Description:** Mô tả ngắn gọn về chức năng của gói phần mềm. Ở đây, đây là một dịch vụ ghi nhật ký hệ thống.
    + **Section:** Xác định phần mà gói phần mềm này thuộc về. Trong trường hợp này, utils cho biết đó là các tiện ích hệ thống.
    + **Depends:** Liệt kê các phần mềm phụ thuộc cần được cài đặt trên hệ thống để gói phần mềm này hoạt động. Ở đây, libc6 (>= 2.28) yêu cầu phiên bản libc6 từ 2.28 trở lên.
    + **License:** Chỉ ra giấy phép mà gói phần mềm này được phát hành. Trong trường hợp này là GPL-3.0.

3. Cấu hình các hành vi của package (Behaviors)

a) preinst

- Được thực thi trước khi gói phần mềm được cài đặt lên hệ thống.
- Kiểm tra xem service có đang chạy không, nếu chạy thì dừng lại.

b) postinst

- Được thực thi sau khi gói phần mềm được cài đặt lên hệ thống.
- Tải lại các cấu hình của systemd.
- Kích hoạt chế độ tự động khởi động cùng với hệ thống của service.
- Start Service ngay lập tức.

c) prerm

- Được thực thi trước khi gói phần mềm bị gỡ bỏ khỏi hệ thống.
- Kiểm tra xem service có đang chạy không, nếu chạy thì dừng lại.

d) postrm

- Tải lại các cấu hình của systemd.

4. Tạo .deb package

- Chạy lệnh build:

  dpkg --build system-logger

- Đây là bước cuối của quá trình đóng gói .deb package, file .deb sẽ xuất hiện tại vị trí chạy dòng lệnh.
- Để thực hiện cài đặt, ta chạy lệnh install:

  sudo dpkg -i system-logger.deb

5. Kiểm tra

- Cài đặt thành công .deb package trên các máy khác nhau, các phiên bản Ubuntu khác nhau.
- Kiểm tra thấy file log có được ghi lại:

```
[2024-06-13 10:09:51] CPU Usage: 0.942803%, Memory Usage: 34.4381%, Swap Usage: 0%, Disk Usage: 26.8018%
[2024-06-13 10:14:52] CPU Usage: 1.88798%, Memory Usage: 34.7865%, Swap Usage: 0%, Disk Usage: 26.7923%
[2024-06-13 10:27:41] CPU Usage: 1.94357%, Memory Usage: 35.0962%, Swap Usage: 0%, Disk Usage: 26.9007%
[2024-06-13 10:32:42] CPU Usage: 0.692695%, Memory Usage: 35.2015%, Swap Usage: 0%, Disk Usage: 26.9007%
[2024-06-13 10:37:43] CPU Usage: 1.45294%, Memory Usage: 35.6722%, Swap Usage: 0%, Disk Usage: 26.896%
[2024-06-13 10:42:44] CPU Usage: 1.83081%, Memory Usage: 36.1523%, Swap Usage: 0%, Disk Usage: 26.9055%
```

- Kiểm tra Logrotate được chạy đúng theo cấu hình.

V. Kết luận

1. Kết quả

- Hoàn thành xây dựng và đóng gói Linux service.
- Đảm bảo tính khả dụng và hiệu quả của service.

2. Hướng phát triển

- Cải thiện hiệu năng và tính năng bảo mật.
- Mở rộng chức năng của service.
- Xây dựng các ứng dụng sử dụng service.

3. Thu hoạch

- Học về khái niệm và cách hoạt động của Static Library và Shared Library.
- Học được cách build Shared Library.
- Học về khái niệm và cách hoạt động của Linux Service.
- Học được cách đăng ký Linux Service.
- Học được cách đóng gói .deb package.
- Học được khái niệm một file .log và cấu hình logrortate cho nó.

VI. Phụ lục

1. Mã nguồn (Source code)

a) “linuxSystemMonitor.hpp”

```
#ifndef _LINUX_SYSTEM_MONITOR_HPP_
#define _LINUX_SYSTEM_MONITOR_HPP_
#include <string>

namespace linuxSystemMonitor {

// CPU Information
struct CpuStats {
   int user;
   int nice;
   int system;
   int idle;
   int iowait;
   int irq;
   int softirq;
   int steal;
   int guest;
   int guest_nice;

   int getTotalIdle() const;
   int getTotalActive() const;
};

// Memory Information
struct MemoryStats {
   int totalMemory;
```

```
    int availableMemory;
    int totalSwap;
    int freeSwap;

    float getMemoryUsage() const;
    float getSwapUsage() const;
};

struct DiskStats {
    // Disk space in MB
    int totalSpace;
    int freeSpace;
};

// Read the CPU data from /proc/stat
CpuStats readCpuData();

// Read the memory data from /proc/meminfo
MemoryStats readMemoryData();

// Read the disk data fro
DiskStats readDiskData();
float getCpuUsage(const CpuStats &first, const CpuStats &second);
float getDiskUsage(const DiskStats &disk);
}  // namespace linuxSystemMonitor

#endif  // _LINUX_SYSTEM_MONITOR_HPP_
```

b) “linuxSystemMonitor.cpp”

```
#include "linuxSystemMonitor.hpp"

#include <sys/statvfs.h>

#include <fstream>
#include <sstream>
#include <string>

namespace linuxSystemMonitor {

// Global paths for the files
const std::string PROC_STAT_PATH = "/proc/stat";
const std::string PROC_MEMINFO_PATH = "/proc/meminfo";
const std::string DEFAULT_DISK_PATH = "/";
```

```
// **CPU Information**
// Read the CPU data from /proc/stat
CpuStats readCpuData() {
   CpuStats result;
   std::ifstream procStat(PROC_STAT_PATH);

   if (procStat.good()) {
       std::string line;
       getline(procStat, line);

       unsigned int* statsPointer = (unsigned int*)&result;
       std::stringstream iss(line);
       std::string cpu;
       iss >> cpu;
       while (iss >> *statsPointer) {
           statsPointer++;
       };
   }

   procStat.close();
   return result;
}

// Get the total idle and active time
int CpuStats::getTotalIdle() const {
   return idle + iowait;
}
int CpuStats::getTotalActive() const {
   return user + nice + system + irq + softirq + steal + guest +
guest_nice;
}

// Calculate the CPU usage
float getCpuUsage(const CpuStats& first, const CpuStats& second) {
   int activeTime = second.getTotalActive() - first.getTotalActive();
   int idleTime = second.getTotalIdle() - first.getTotalIdle();
   int totalTime = activeTime + idleTime;
   return static_cast<float>(activeTime) / totalTime;
}

// **Memory Information**
// Read the memory data from /proc/meminfo
```

```
MemoryStats readMemoryData() {
   MemoryStats result;
   std::ifstream procMeminfo(PROC_MEMINFO_PATH);

   if (procMeminfo.good()) {
       std::string line;
       for (int i = 0; i < 17; ++i) {
           getline(procMeminfo, line);
           std::stringstream iss(line);
           std::string key;
           int value;
           iss >> key >> value;
           if (key == "MemTotal:") {
               result.totalMemory = value;
           } else if (key == "MemAvailable:") {
               result.availableMemory = value;
           } else if (key == "SwapTotal:") {
               result.totalSwap = value;
           } else if (key == "SwapFree:") {
               result.freeSwap = value;
           }
       }
   }
   procMeminfo.close();
   return result;
}

// Calculate the memory usage
float MemoryStats::getMemoryUsage() const {
   return static_cast<float>(totalMemory - availableMemory) /
totalMemory;
}

// Calculate the swap usage
float MemoryStats::getSwapUsage() const {
   return static_cast<float>(totalSwap - freeSwap) / totalSwap;
}

// **Disk Information**
// Read the disk data from the specified path
DiskStats readDiskData() {
   DiskStats result;
   struct statvfs diskData;
```

```
    if (statvfs(DEFAULT_DISK_PATH.c_str(), &diskData) == 0) {
        // calculate total and free space in MB
        unsigned long int totalSpace = diskData.f_blocks *
diskData.f_frsize / 1024 / 1024;
        unsigned long int freeSpace = diskData.f_bfree *
diskData.f_frsize / 1024 / 1024;
        result.totalSpace = totalSpace;
        result.freeSpace = freeSpace;
    }
    return result;
}

// Calculate the disk usage
float getDiskUsage(const DiskStats& disk) {
    return static_cast<float>(disk.totalSpace - disk.freeSpace) /
disk.totalSpace;
}

}  // namespace linuxSystemMonitor
```

c) “linuxSystemLogger.cpp”

```
#include <chrono>
#include <ctime>
#include <fstream>
#include <iostream>
#include <thread>

#include "linuxSystemMonitor.hpp"

const std::chrono::minutes LOG_INTERVAL(5);
const std::string LOG_FILE_PATH = "/var/log/system-usage.log";

// Function to get the current time as a string
std::string getCurrentTime() {
    std::time_t now = std::time(nullptr);
    char timeStr[100];
    std::strftime(timeStr, sizeof(timeStr), "%Y-%m-%d %H:%M:%S",
std::localtime(&now));
    return std::string(timeStr);
}

// Function to log system usage information
void logSystemUsage() {
```

```
    // Get initial CPU stats
    auto cpu1 = linuxSystemMonitor::readCpuData();
    std::this_thread::sleep_for(std::chrono::seconds(1));
    // Get CPU stats again after 1 second
    auto cpu2 = linuxSystemMonitor::readCpuData();
    // Calculate CPU usage
    float cpuUsage = linuxSystemMonitor::getCpuUsage(cpu1, cpu2) *
100.0f;

    // Get memory statistics
    auto memStats = linuxSystemMonitor::readMemoryData();
    float memoryUsage = memStats.getMemoryUsage() * 100.0f;
    float swapUsage = memStats.getSwapUsage() * 100.0f;

    // Get disk statistics
    auto diskStats = linuxSystemMonitor::readDiskData();
    float diskUsage = linuxSystemMonitor::getDiskUsage(diskStats) *
100.0f;

    // Open the log file and append the system usage information
    std::ofstream logFile(LOG_FILE_PATH, std::ios_base::app);
    if (logFile.is_open()) {
        logFile << "[" << getCurrentTime() << "] "
                << "CPU Usage: " << cpuUsage << "%, "
                << "Memory Usage: " << memoryUsage << "%, "
                << "Swap Usage: " << swapUsage << "%, "
                << "Disk Usage: " << diskUsage << "%" << std::endl;
        logFile.close();  // Đóng file sau khi ghi
    } else {
        std::cerr << "Error opening log file: " << LOG_FILE_PATH <<
std::endl;
    }
}

int main() {
    // Continuously log system usage every LOG_INTERVAL
    while (true) {
        logSystemUsage();
        std::this_thread::sleep_for(LOG_INTERVAL);
    }
    return 0;
}
```

d) “Makefile”

```
# Compiler
CC = g++

# Compiler flags
CFLAGS = -Wall -Werror -fPIC -I$(INC_DIR)

# Target library name
TARGET_LIB = liblinuxsystemmonitor.so
# Target executable name
TARGET_EXE = linuxSystemLogger

# Directories
SRC_DIR = src
LIB_DIR = lib
BIN_DIR = bin
INC_DIR = include

# Phony targets
.PHONY: all clean

# Default target
all: $(LIB_DIR)/$(TARGET_LIB) $(BIN_DIR)/$(TARGET_EXE)

# Build library
$(LIB_DIR)/$(TARGET_LIB): $(SRC_DIR)/linuxSystemMonitor.cpp
$(INC_DIR)/linuxSystemMonitor.hpp $(LIB_DIR)
    $(CC) $(CFLAGS) -shared -o $(LIB_DIR)/$(TARGET_LIB)
$(SRC_DIR)/linuxSystemMonitor.cpp

# Build executable
$(BIN_DIR)/$(TARGET_EXE): $(SRC_DIR)/linuxSystemLogger.cpp
$(INC_DIR)/linuxSystemMonitor.hpp $(LIB_DIR)/$(TARGET_LIB) $(BIN_DIR)
    $(CC) $(CFLAGS) -o $(BIN_DIR)/$(TARGET_EXE)
$(SRC_DIR)/linuxSystemLogger.cpp -L$(LIB_DIR) -llinuxsystemmonitor

# Create library directory
$(LIB_DIR):
    mkdir -p $(LIB_DIR)

# Create binary directory
$(BIN_DIR):
    mkdir -p $(BIN_DIR)
```

```
# Clean
clean:
	rm -rf $(LIB_DIR) $(BIN_DIR)
```

e) “system-logger.service”

system-logger.service

```
[Unit]
Description=System Logger Service
After=network.target

[Service]
Type=simple
ExecStart=/usr/local/bin/linuxSystemLogger
Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target

```

f) “system-usage”

system-usage

```
/var/log/system-usage.log {
    daily
    rotate 7
    missingok
    notifempty
    compress
    create 0644 root root
}

```

2. Tham chiếu (References)

https://www.iodigital.com/en/history/intracto/creating-debianubuntu-deb-packages

https://blog.knoldus.com/create-a-debian-package-using-dpkg-deb-tool/

https://www.linuxhowtos.org/System/procstat.htm

https://www.baeldung.com/linux/proc-meminfo

https://man7.org/linux/man-pages/man3/statvfs.3.html

https://www.shubhamdipt.com/blog/how-to-create-a-systemd-service-in-linux/

https://www.cprogramming.com/tutorial/shared-libraries-linux-gcc.html