
# Viettel Digital Talent 2024

Software Engineering - Systems Development

## Project: Đóng gói Linux service vào .deb package

Người thực hiện: Nguyễn Quang Trường

# I. Tổng quát

Dự án bao gồm ba nội dung chính:

- Xây dựng một shared library có chức năng lấy thông tin về CPU và RAM của hệ thống;
- Xây dựng một Linux service sử dụng shared library trên;
- Đóng gói service trên thành một .deb package để cài đặt trên các máy Ubuntu.

Dự án được xây dựng trên C++17, sử dụng trình biên dịch GNU GCC 12.3.0. Hệ thống được sử dụng để xây dựng dự án là Ubuntu 22.04, Linux kernel phiên bản 5.15.0, GLIBC phiên bản 2.35, CPU kiến trúc x86_64, systemd phiên bản 249.11.

# II. Nội dung chính

## 1. Shared library lấy thông tin về CPU và RAM

Định nghĩa trong manpage của **`proc(5)`**: **`proc`** filesystem là một pseudo-filesystem cung cấp giao diện tới các cấu trúc dữ liệu của kernel, thường được đặt tại **`/proc`**. Một vài pseudo-file ở trong **`/proc`** bao gồm **`/proc/stat`** (chứa các thông tin chung về kernel), **`/proc/cpuinfo`** (chứa các thông tin về CPU), **`/proc/meminfo`** (chứa các thông tin về memory), v.v.

### a. Thông tin về CPU

Để lấy được thông tin về CPU usage, các công cụ như top/htop/btop đọc và xử lý thông tin từ **`/proc/stat`** [1][2][3]. Các dòng đầu tiên của pseudo-file này bao gồm thông tin của toàn bộ CPU và thông tin của các core riêng biệt. Dưới đây là ví dụ khi thực hiện lệnh **`head -n9 /proc/stat`** để in ra 9 dòng đầu tiên của pseudo-file trên máy tính sử dụng CPU 8 core:

```
cpu  13000764 9507 2244913 74892485 972586 0 185645 0 705982 0
cpu0 1627644 413 282787 9286310 120868 0 23233 0 92085 0
cpu1 1680479 957 278270 9338707 118519 0 21156 0 89044 0
cpu2 1549828 845 284752 9410810 120622 0 48504 0 85688 0
cpu3 1667116 1351 296711 9332029 118666 0 3804 0 89494 0
cpu4 1669593 4997 268486 9349192 123495 0 4044 0 90480 0
cpu5 1553699 98 276142 9444654 124298 0 10193 0 84118 0
cpu6 1679272 692 274911 9356225 122835 0 5107 0 89354 0
cpu7 1573129 150 282852 9374554 123279 0 69601 0 85717 0
```

Mỗi con số là thời gian mà CPU hoặc các core chạy trong các trạng thái (đơn vị thường là 1/100 giây trên hầu hết các hệ thống). Số lượng trạng thái tùy thuộc vào phiên bản Linux kernel. Với các kernel hiện đại, mỗi dòng sẽ có 10 con số, tương đương với 10 trạng thái (từ trái sang phải):

- **user**: thời gian trong user mode;
- **nice**: thời gian trong user mode với độ ưu tiên thấp (low priority);
- **system**: thời gian trong system mode;
- **idle**: thời gian thực hiện idle task;
- **iowait**: thời gian chờ thực hiện các tác vụ I/O;
- **irq**: thời gian thực hiện hardware interrupt;
- **softirq**: thời gian thực hiện software interrupt;
- **steal**: thời gian trong các OS khác trong môi trường ảo hóa;
- **guest**: thời gian chạy CPU ảo cho các guest OS dưới sự kiểm soát của Linux kernel;
- **guest_nice**: thời gian chạy **guest** với độ ưu tiên thấp (low priority).

Trong các trạng thái trên, **idle** và **iowait** là các trạng thái nghỉ (idle). Ngoài ra, **guest** và **guest_nice** cũng đã được tính vào user và nice bởi Linux kernel. Các trạng thái còn lại là trạng thái hoạt động (work).

Tại mỗi thời điểm đọc **`/proc/stat`**, có thể tính được tổng thời gian trong toàn bộ trạng thái (total) và tổng thời gian trong trạng thái hoạt động (work) kể từ lúc khởi động hệ thống.

```
struct Jiffies {
    unsigned long long total;
    unsigned long long work;
};

std::optional<Jiffies> parse_from_system() {
    std::ifstream in("/proc/stat");

    if (!in) {
        return std::nullopt;
    }

    std::string label;
    unsigned long long user, nice, system, idle, iowait, irq,
softirq, steal;
    in >> label >> user >> nice >> system >> idle >> iowait >>
irq >> softirq >> steal;
```

```
    Jiffies result;
    result.total = user + nice + system + idle + iowait + irq +
softirq + steal;
    result.work  = user + nice + system + irq + softirq + steal;
    return result;
}
```

CPU usage được định nghĩa bằng tỉ lệ thời gian CPU ở trạng thái work trong một khoảng thời gian (interval). Cách thực hiện tính toán sẽ như sau:

- Lấy thông tin về CPU trước và sau một khoảng thời gian (interval) tùy ý;
- Kết quả là chênh lệch thời gian trong trạng thái work chia cho chênh lệch thời gian tổng.

```
const auto previous_jiffies = parse_from_system();
std::this_thread::sleep_for(std::chrono::milliseconds(interval));
const auto current_jiffies = parse_from_system();

const auto total_diff =
    current_jiffies->total - previous_jiffies->total;
const auto work_diff =
    current_jiffies->work - previous_jiffies->work;

result = 1.0f * work_diff / total_diff;
```

*(đoạn mã đã được lược giản so với mã nguồn)*

Interface của thư viện có dạng:

```
extern "C" float jaltop_cpu_usage(int interval_ms);
```

Trong đó:

- `interval_ms == 0`: CPU usage sẽ được tính từ thời điểm cuối cùng **`/proc/stat`** được parse bởi thư viện đến thời điểm function được gọi;
- `interval_ms != 0`: thư viện sẽ parse lại **`/proc/stat`** hai lần (cách nhau `interval_ms` mili giây) để tính CPU usage.

## b. Thông tin về RAM

Thông tin về RAM cũng có thể được truy xuất thông qua **`/proc`**. Tuy nhiên, Linux kernel có cung cấp sẵn system call **`sysinfo()`**. Từ manpage của **`sysinfo(2)`**, `struct sysinfo` được định nghĩa như sau:

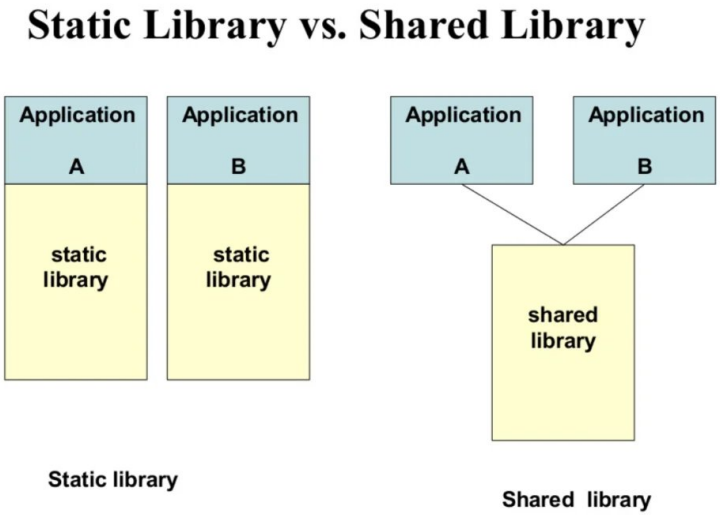
```
struct sysinfo {
    long uptime;             /* Seconds since boot */
    unsigned long loads[3];  /* 1, 5, and 15 minute load averages
*/
    unsigned long totalram;  /* Total usable main memory size */
    unsigned long freeram;   /* Available memory size */
    unsigned long sharedram; /* Amount of shared memory */
    unsigned long bufferram; /* Memory used by buffers */
    unsigned long totalswap; /* Total swap space size */
    unsigned long freeswap;  /* Swap space still available */
    unsigned short procs;    /* Number of current processes */
    unsigned long totalhigh; /* Total high memory size */
    unsigned long freehigh;  /* Available high memory size */
    unsigned int mem_unit;   /* Memory unit size in bytes */
    char _f[20-2*sizeof(long)-sizeof(int)];
                            /* Padding to 64 bytes */
};
```

Như vậy, những thông tin cần có như **totalram** và **freeram** đều có thể truy xuất được qua system call.

```
#include <sys/sysinfo.h>

#include <limits>

constexpr auto NaN = std::numeric_limits<float>::quiet_NaN();

extern "C" float jaltop_ram_usage() {
    struct sysinfo sysinfo;
    const auto retval = ::sysinfo(&sysinfo);

    if (retval != 0) {
        return NaN;
    }

    return 1.0f * (sysinfo.totalram - sysinfo.freeram) /
sysinfo.totalram;
}
```

Ưu điểm của việc sử dụng system call là các thông tin đều được trả về trực tiếp bởi kernel, thay vì phải parse thủ công. Tuy nhiên, chính vì phụ thuộc vào kernel nên phiên bản của kernel của người dùng phải phù hợp với phiên bản phát triển.

c. Shared library




Shared library là một object file được chia sẻ giữa các chương trình khác nhau tại lúc thực thi, thay vì được chèn vào mã máy của chương trình tại lúc biên dịch. Shared library có một số ưu điểm như cải thiện memory, storage, thời gian load và cả hiệu năng trong một số trường hợp [4].

Giả sử có một chương trình mẫu như sau:

```
#include "libjaltop.hpp"

#include <iostream>

int main() {
    std::cout << jaltop_cpu_usage(0) << ' ' << jaltop_ram_usage();
}
```

Có được một đoạn Assembly ở đầu của **`main()`**:

```
0000000000001100 <main>:
    1100:  f3 0f 1e fa           endbr64
    1104:  53                    push   rbx
    1105:  31 ff                 xor    edi,edi
    1107:  e8 e4 ff ff ff        call   10f0 <jaltop_cpu_usage@plt>
```

Kiểm tra **`jaltop_cpu_usage@plt`**:

```
00000000000010f0 <jaltop_cpu_usage@plt>:
```

```
    10f0:  f3 0f 1e fa           endbr64
    10f4:  f2 ff 25 cd 2e 00 00  bnd jmp QWORD PTR [rip+0x2ecd]
# 3fc8 <jaltop_cpu_usage@Base>
    10fb:  0f 1f 44 00 00        nop    DWORD PTR [rax+rax*1+0x0]
```

Thay vì gọi trực tiếp đến **jaltop_cpu_usage()**, chương trình lại gián tiếp đi qua một Procedure Linkage Table (PLT). Hiểu nôm na, tại thời điểm biên dịch, chương trình không biết được địa chỉ của hàm. Vậy nên, tại thời điểm thực thi, dynamic linker mới điền địa chỉ của hàm vào PLT, và hàm được truy cập gián tiếp qua PLT.

Sau khi khởi động GDB lên chương trình và kiểm tra lại **jaltop_cpu_usage@plt**, lúc này **jaltop_cpu_usage@Base** trở thành **jaltop_cpu_usage@got.plt**. Tuy nhiên, vì chưa thực sự khởi động chương trình nên **jaltop_cpu_usage@got.plt** chưa được nạp vào.

```
(gdb) disas 'jaltop_cpu_usage@plt'
Dump of assembler code for function jaltop_cpu_usage@plt:
  0x00000000000010f0 <+0>:     endbr64
  0x00000000000010f4 <+4>:     bnd jmp QWORD PTR [rip+0x2ecd]
# 0x3fc8 <jaltop_cpu_usage@got.plt>
  0x00000000000010fb <+11>:    nop    DWORD PTR [rax+rax*1+0x0]
End of assembler dump.
(gdb) disas 'jaltop_cpu_usage@got.plt'
No function contains specified address.
```

Sau khi chạy chương trình và kiểm tra lại **jaltop_cpu_usage@got.plt**:

```
(gdb) b main
Breakpoint 1 at 0x1100: file examples/main.cpp, line 5.
(gdb) r
Starting program: /home/jalsol/Documents/libjaltop/build/main
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-
gnu/libthread_db.so.1".

Breakpoint 1, main () at examples/main.cpp:5
5       int main() {
(gdb) disas 'jaltop_cpu_usage@got.plt'
Dump of assembler code for function jaltop_cpu_usage(int):
  0x00007ffff7fb7790 <+0>:     endbr64
  0x00007ffff7fb7794 <+4>:     push   rbx
```

```
  0x00007ffff7fb7795 <+5>:     movsxd rbx,edi
  0x00007ffff7fb7798 <+8>:     sub    rsp,0x60
  # ...
```

*(output đã được lược bớt)*

Như vậy, **`jaltop_cpu_usage@got.plt`** đã được nạp vào chương trình và trỏ đến **`jaltop_cpu_usage()`** sau khi khởi động.

Makefile để build shared library:

```
CC=g++
CFLAGS=-std=c++17 -Wall -Wextra -O2 -Iinclude

all:
	$(CC) $(CFLAGS) -o build/libjaltop.so -fPIC -shared
src/cpu.cpp src/ram.cpp

clean:
	rm build/*.so
```

Cờ **`-shared`** được sử dụng để xây dựng shared library. Cờ **`-fPIC`** đảm bảo trình biên dịch sẽ xuất ra Position Independent Code (PIC) - code mà không phụ thuộc vào vị trí (ví dụ: các lệnh **`jmp`** sẽ sử dụng vị trí tương đối với register **`rip`** thay vì tuyệt đối).

## 2. Xây dựng Linux service sử dụng shared library

Một ví dụ của việc sử dụng Linux service để lấy thông số CPU và RAM là việc kiểm soát những thông số trên của các máy tính trong cùng một mạng. Để minh họa đơn giản, một HTTP server có khả năng gửi các thông số về CPU và RAM usage được xây dựng. Chương trình này được kết nối vào port 6969.

```
#include "libjaltop.hpp"
#include "httplib.h"

#include <string>

int main() {
    using namespace httplib;
    Server server;
```

```
    server.Get("/ram/total", [&](const Request&, Response& res) {
        res.set_content(std::to_string(jaltop_ram_total()),
"text/plain");
    });

    server.Get("/ram/free", [&](const Request&, Response& res) {
        res.set_content(std::to_string(jaltop_ram_free()),
"text/plain");
    });

    server.Get("/ram/usage", [&](const Request&, Response& res) {
        res.set_content(std::to_string(jaltop_ram_usage()),
"text/plain");
    });

    server.Get("/cpu/usage", [&](const Request&, Response& res) {
        res.set_content(std::to_string(jaltop_cpu_usage(0)),
"text/plain");
    });

    server.Get("/cpu/usage/:interval", [&](const Request& req,
Response& res) {
        const auto interval =
std::stoi(req.path_params.at("interval"));

res.set_content(std::to_string(jaltop_cpu_usage(interval)),
"text/plain");
    });

    server.listen("0.0.0.0", 6969);
}
```

Tạo unit file **`/etc/systemd/system/jaltop.service`** với nội dung [5]:

```
[Unit]
Description=just a little TOP (service)
StartLimitIntervalSec=0

[Service]
User=root
```

```
ExecStart=/usr/bin/jaltop
Restart=always
RestartSec=1

[Install]
WantedBy=multi-user.target
```

**[Unit]**: thông tin về bản thân service;
- **Description**: mô tả về service;
- **StartLimitIntervalSec**: thời gian chờ tối đa trước khi khởi động lại service nếu service khởi động lỗi (giá trị 0 nghĩa là hệ thống sẽ cố gắng liên tục khởi động lại service mà không chờ).

**[Service]**: thông tin về cách service hoạt động;
- **User**: service sẽ chạy dưới quyền của người dùng nào (**root** nghĩa là thực thi quyền cao nhất);
- **ExecStart**: lệnh của chương trình mà service muốn thực thi;
- **Restart**: chế độ khởi động lại service khi bị dừng đột ngột (**always** nghĩa là luôn khởi động lại);
- **RestartSec**: số giây chờ trước khi khởi động lại service.

**[Install]**: hành vi của service khi **enable**/**disable** bởi **systemctl**;
- **WantedBy**: chỉ định các unit hoặc target mà service phụ thuộc vào (**multi-user.target** có thể hiểu là service sẽ được khởi động tại multi-user boot stage).

Sau khi tạo unit file, sử dụng lệnh **systemctl start jaltop** để bắt đầu service. Kiểm tra trạng thái của service bằng lệnh **systemctl status jaltop**. Lưu ý đường dẫn của **ExecStart** trước khi thực thi.

```
$ systemctl start jaltop
$ systemctl status jaltop
● jaltop.service - just a little TOP (service)
    Loaded: loaded (/etc/systemd/system/jaltop.service; disabled;
vendor preset: enabled)
    Active: active (running) since Thu 2024-06-13 05:46:53 +07;
1min 43s ago
  Main PID: 316483 (jaltop)
     Tasks: 9 (limit: 13904)
    Memory: 600.0K
       CPU: 4ms
```

```
    CGroup: /system.slice/jaltop.service
            └─316483 /usr/bin/jaltop

Jun 13 05:46:53 jalsolPC systemd[1]: Started just a little TOP
(service).
```

Sau khi khởi động service:

```
$ curl -X GET "127.0.0.1:6969/ram/usage"
0.773185
$ curl -X GET "127.0.0.1:6969/ram/free"
2792280064
$ curl -X GET "127.0.0.1:6969/ram/total"
12269572096
$ curl -X GET "127.0.0.1:6969/cpu/usage"
0.025000
$ curl -X GET "127.0.0.1:6969/cpu/usage/1000"
0.020177
```

## 3. Đóng gói thành .deb package

Giả sử package có tên là **jaltop_0.2-1** (tên là **jaltop**, phiên bản **0.2**, build number **1**).

Tạo các đường dẫn có cấu trúc như sau:

```
jaltop_0.2-1
├── DEBIAN
│   └── control
├── etc
│   └── systemd
│       └── system
│           └── jaltop.service
└── usr
    ├── bin
    │   └── jaltop
    └── lib
        └── libjaltop.so
```

Ngoại trừ thư mục đặc biệt **DEBIAN**, các thư mục khác bên trong **jaltop_0.2-1** đều có cấu trúc sao cho khi cài đặt package, các tệp tin ở các thư mục con sẽ được cài đặt ở thư mục **/** (thư mục cao nhất của Linux) tương ứng.

Ví dụ, sau khi cài đặt package, file **jaltop_0.2-1/usr/lib/libjaltop.so** sẽ được cài đặt tại **/usr/lib/libjaltop.so**.

Nội dung của **DEBIAN/control** có dạng như sau [6]:

```
Package: jaltop
Version: 0.2-1
Section: testing
Priority: optional
Architecture: amd64
Maintainer: Quang-Truong Nguyen <jalsol@protonmail.com>
Description: just a little TOP
```

- **Package**: tên của package;
- **Version**: số phiên bản;
- **Section**: chỉ định section trên Debian archive mà package được phân loại
- **Priority**: mức độ ưu tiên (**optional** nghĩa là người dùng có thể lựa chọn tải xuống hoặc không);
- **Architecture**: kiến trúc của máy tính có thể cài đặt được package;
- **Maintainer**: tên và địa chỉ email của người phát triển;
- **Description**: mô tả của package.

Cuối cùng, sử dụng lệnh **dpkg-deb --build jaltop_0.2-1** để đóng gói.

Sau khi đóng gói, package có thể được cài đặt bằng lệnh **dpkg -i jaltop_0.2-1.deb** để cài đặt. Để gỡ package, sử dụng lệnh **apt remove jaltop**.

# III. Mã nguồn

Shared library: https://github.com/jalsol/libjaltop
Linux service: https://github.com/jalsol/jaltop

# IV. Tham khảo


[1]: https://gitlab.com/procps-ng/procps/-/blob/master/library/stat.c#L729
[2]: https://github.com/htop-dev/htop/blob/main/linux/LinuxMachine.c#L407
[3]: https://github.com/aristocratos/btop/blob/main/src/linux/btop_collect.cpp#L875
[4]: https://www.ibm.com/docs/en/aix/7.3?topic=concepts-shared-libraries-shared-memory

[5]: https://www.freedesktop.org/software/systemd/man/latest/systemd.unit.html
[6]: https://www.debian.org/doc/debian-policy/ch-controlfields