Lab 3

Lab 3: Đo Timer Latency & Phân phối Jitter trên Raspberry Pi 4B/5 (PREEMPT_RT)

Mục tiêu

- Sử dụng timer định kỳ chính xác (absolute) bằng `clock_nanosleep()` để đo **latency** = (thời điểm thực tế − thời điểm kỳ vọng).
- Ghi log latency ra **CSV** để phân tích: min / max / mean / std và **histogram**.
- So sánh **SCHED_OTHER** (mặc định) với **SCHED_FIFO** (real-time) trong 2 trường hợp: *nhàn rỗi* và *có tải CPU*.

Kiến thức nền tảng

- `clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, ...)` cho hẹn giờ **theo mốc tuyệt đối** → giảm drift.
- Scheduler:
- `SCHED_OTHER` (mặc định) – time-sharing, không đảm bảo deadline.
- `SCHED_FIFO` (ưu tiên 1..99) – real-time, ưu tiên tuyệt đối.
- **Latency** ở đây là **jitter** của timer so với lịch mong đợi (ví dụ 1ms/2ms/10ms).

Chuẩn bị

- Raspberry Pi 4B/5, Pi OS 64-bit với **kernel PREEMPT_RT** (đã cài ở bài Lab 1).
- G++ và Python3:

```
sudo apt update

sudo apt install -y g++ python3 python3-matplotlib
```

- (Tuỳ chọn) `htop`, `taskset`, `chrt` để quan sát & pin CPU:

```
sudo apt install -y htop util-linux
```

Cấu trúc file

```
lab3/
├── timer_other.cpp # đo latency với SCHED_OTHER
├── timer_fifo.cpp # đo latency với SCHED_FIFO (priority đặt qua tham số)
├── hop.cpp # tiến trình tạo nhiễu CPU
└── plot_latency.py # vẽ histogram + time-series, in min/max/mean/std
```

Code

1) `timer_other.cpp` — SCHED_OTHER

```cpp
#include <iostream>
#include <fstream>
#include <chrono>
#include <thread>
#include <cstdlib>
#include <string>
#include <ctime>

static inline timespec add_ns(const timespec& t, long long ns) {
timespec r = t;
r.tv_nsec += ns % 1000000000LL;
r.tv_sec += ns / 1000000000LL;
```

```
if (r.tv_nsec >= 1000000000L) { r.tv_nsec -= 1000000000L; r.tv_sec++; }
return r;
}

int main(int argc, char* argv[]) {
int interval_us = 1000; // default 1ms
int samples = 60000; // ~60s
std::string out = "lat_other.csv";

for (int i=1; i<argc-1; ++i) {
std::string a = argv[i];
if (a == "--interval-us") interval_us = std::atoi(argv[++i]);
else if (a == "--samples") samples = std::atoi(argv[++i]);
else if (a == "--out") out = argv[++i];
}

std::ofstream csv(out);
csv << "index,latency_us\n";

timespec ts; clock_gettime(CLOCK_MONOTONIC, &ts);
long long period_ns = static_cast<long long>(interval_us) * 1000LL;


for (int i=0; i<samples; ++i) {
ts = add_ns(ts, period_ns);
clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &ts, nullptr);


timespec tnow; clock_gettime(CLOCK_MONOTONIC, &tnow);
```

```
long long actual_ns = (tnow.tv_sec - ts.tv_sec)*1000000000LL +
(tnow.tv_nsec - ts.tv_nsec);

double latency_us = static_cast<double>(actual_ns) / 1000.0;

csv << i << "," << latency_us << "\n";

}

csv.close();

std::cout << "Done. Data -> " << out << "\n";

return 0;

}
```

2) `timer_fifo.cpp` — SCHED_FIFO

```
#include <iostream>

#include <fstream>

#include <string>

#include <cstdlib>

#include <ctime>

#include <sched.h>

#include <unistd.h>

static inline timespec add_ns(const timespec& t, long long ns) {

timespec r = t;

r.tv_nsec += ns % 1000000000LL;

r.tv_sec += ns / 1000000000LL;

if (r.tv_nsec >= 1000000000L) { r.tv_nsec -= 1000000000L; r.tv_sec++; }

return r;

}
```

```
int main(int argc, char* argv[]) {

int interval_us = 1000; // default 1ms

int samples = 60000; // ~60s

int priority = 80;

std::string out = "lat_fifo.csv";


for (int i=1; i<argc-1; ++i) {

std::string a = argv[i];

if (a == "--interval-us") interval_us = std::atoi(argv[++i]);

else if (a == "--samples") samples = std::atoi(argv[++i]);

else if (a == "--priority") priority = std::atoi(argv[++i]);

else if (a == "--out") out = argv[++i];

}


sched_param sch; sch.sched_priority = priority;

if (sched_setscheduler(0, SCHED_FIFO, &sch) == -1) {

perror("sched_setscheduler"); return 1;

}


std::ofstream csv(out);

csv << "index,latency_us\n";


timespec ts; clock_gettime(CLOCK_MONOTONIC, &ts);

long long period_ns = static_cast<long long>(interval_us) * 1000LL;


for (int i=0; i<samples; ++i) {

ts = add_ns(ts, period_ns);
```

```
clock_nanosleep(CLOCK_MONOTONIC, TIMER_ABSTIME, &ts, nullptr);



timespec tnow; clock_gettime(CLOCK_MONOTONIC, &tnow);

long long actual_ns = (tnow.tv_sec - ts.tv_sec)*1000000000LL +
(tnow.tv_nsec - ts.tv_nsec);

double latency_us = static_cast<double>(actual_ns) / 1000.0;

csv << i << "," << latency_us << "\n";

}

csv.close();

std::cout << "Done. Data -> " << out << "\n";

return 0;

}
```

3) `hop.cpp`

```
int main() {
    while (true) {
// No Operation - khong lam gi het, chi ton 1 CPU cycle
// chiem dung CPU - 1 CPU cycle - chay lien tuc
    asm volatile ("nop");
    }
}
```

4) `plot_latency.py`

```
import sys, csv

import matplotlib.pyplot as plt

# ==== Student Info ====

hoten = "Nguyen Van A"

mssv = "20220989383"
```

```
# ======================

if len(sys.argv) < 2:

print("Usage: python3 plot_latency.py <latency_csv>")

sys.exit(1)

latencies = []

with open(sys.argv[1], 'r') as f:

r = csv.DictReader(f)

for row in r:

latencies.append(float(row['latency_us']))

# Separate normal and outliers

threshold = 1000.0 # 1000 us

normal_lat = [x for x in latencies if x <= threshold]

outlier_lat = [x for x in latencies if x > threshold]

# Stats

n = len(latencies)

mean_val = sum(latencies)/n

min_val, max_val = min(latencies), max(latencies)

std_val = (sum((x-mean_val)**2 for x in latencies)/n) ** 0.5


print(f"Samples={n} mean={mean_val:.3f} us std={std_val:.3f} us min=
{min_val:.3f} us max={max_val:.3f} us")

print(f"Outliers (> {threshold} us): {len(outlier_lat)} samples")

# ----- Time-series -----

plt.figure()

plt.title(f"Latency Time-series (us)\n{hoten} - {mssv}")

plt.plot(latencies, label="Latency (us)")
```

```
plt.axhline(y=threshold, color='red', linestyle='--', label=f"Threshold
{threshold} us")

plt.xlabel("Sample"); plt.ylabel("Latency (us)")

plt.legend()

plt.grid(True)

# ----- Histogram -----

plt.figure()

plt.title(f"Latency Histogram (us)\n{hoten} - {mssv}")

plt.hist(normal_lat, bins=100, color='blue', label=f"<= {threshold} us")

if outlier_lat:

plt.hist(outlier_lat, bins=20, color='red', label=f"> {threshold} us")

plt.xlabel("Latency (us)"); plt.ylabel("Count")

plt.legend()

plt.grid(True)

plt.show()
```

Biên dịch

```
g++ timer_other.cpp -o timer_other

g++ timer_fifo.cpp -o timer_fifo

g++ hop.cpp -o hop
```

Chạy thí nghiệm

1. Không nhiễu (Idle):

```
./timer_other --interval-us 1000 --samples 600000 --out lat_other_idle.csv

sudo ./timer_fifo --interval-us 1000 --samples 600000 --priority 80 --out
lat_fifo_idle.csv
```

2. Có nhiễu CPU:

```
./hop & ./hop & ./hop & ./hop & ./hop &

./timer_other --interval-us 1000 --samples 600000 --out lat_other_load.csv

sudo ./timer_fifo --interval-us 1000 --samples 600000 --priority 80 --out
lat_fifo_load.csv

killall hop
```

Phân tích

```
python3 plot_latency.py lat_other_idle.csv

python3 plot_latency.py lat_fifo_idle.csv

python3 plot_latency.py lat_other_load.csv

python3 plot_latency.py lat_fifo_load.csv
```

Báo cáo

1. Kết quả min/max/mean/std
2. Biểu đồ time-series & histogram
3. Nhận xét & kết luận