گزارش تمرین دوم درس سیستم‌های نهفته بی‌درنگ

سینا کمالی (۸۱۰۱۹۷۵۶۹)
علیرضا آقایی (۸۱۰۱۹۷۶۷۹)
دانشور امراللهی (۸۱۰۱۹۷۶۸۵)
مهیار کریمی (۸۱۰۱۹۷۶۹۰)

مقدمه

در این پروژه ما وظیفه طراحی یک اپلیکیشن موبایل را داریم که پستی و بلندی‌های سطح را می‌سنجد. برای این امر از زبان جاوا و android studio استفاده خواهیم کرد و پس از طراحی منطق و UI برنامه و تست کردن آن، با استفاده از Perfetto به جمع کردن system trace گوشی در هنگام ران کردن اپلیکیشن می‌پردازیم که به ما insight هایی در رابطه با performance نرم‌افزار می‌دهد.

پیاده سازی

برای پیاده سازی این پروژه، نیاز به Android Studio و بقیه مواد مورد نیاز آن (SDK ها و ...) خواهیم داشت. پس از دریافت آنها و نصب تمامی پیش نیازها به سراغ پیاده سازی پروژه میرویم.

برای این پروژه ما نیازی به چندین button خواهیم داشت. تعریف button در اندروید کار نسبتا ساده ای است و به صورت زیر قابل انجام است:

```
Button start = findViewById(R.id.start);

Button finish = findViewById(R.id.finish);
```

و همینطور برای assign کردن یک تابع برای onclick آنها می توانیم به صورت زیر عمل کنیم:

```
start.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                  ...
            }
});
```

باید برای ادامه پیاده سازی پروژه با مبحث Sensor ها آشنا شویم.

در برنامه نویسی اندروید، شما می‌توانید سنسورهایی که به آنها احتیاج دارید را پس از init کردن، رجیستر کنید تا سیستم شروع به listen کردن بر روی آن سنسور کند.

```
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

accelerometer =
sensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION);
gyroscope = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);

sensorManager.registerListener(
                        MainActivity.this,
                        accelerometer,
                        SensorManager.SENSOR_DELAY_NORMAL
                );
sensorManager.registerListener(
                        MainActivity.this,
                        gyroscope,
                        SensorManager.SENSOR_DELAY_NORMAL
                );
```

پس از این امر می‌توانیم با override کردن تابع onSensorChanged در تغییر حالت sensor ها event ای دریافت کنیم. سپس می‌توانیم با بررسی آن event متوجه شویم که برای کدام سنسور بوده و بر اساس آن دیتا را آپدیت کنیم (که جلوتر به آن می‌پردازیم).

```
@Override
public void onSensorChanged(SensorEvent sensorEvent) {
       if (sensorEvent.sensor.getType() == Sensor.TYPE_LINEAR_ACCELERATION)
{
           handleAccelerometer(sensorEvent);
       }
       else if (sensorEvent.sensor.getType() == Sensor.TYPE_GYROSCOPE) {
           handleGyroscope(sensorEvent);
       }
       xyValueArray.add(new XYValue(position[0], position[2]));
}
```

فیزیک اپلیکیشن

برای محاسبه x و y (مختصات) گوشی از هر دو سنسور accelerometer و gyroscope استفاده شده است که با رخ دادن event برای هر سنسور، محاسبات متناسب با آن انجام می‌شود.

- سنسور accelerometer:

```
public void handleAccelerometer(SensorEvent sensorEvent) {
        Log.d(TAG, "onAccelerometerChanged:");
        float dt = (sensorEvent.timestamp - prevTimestampAcc) * NS2S;
        dt = Math.min(dt, 0.16f);
        for (int i = 0; i < 3; ++i) {
            if (sensorEvent.values[i] < A_THRESHOLD)
                continue;
            velocity[i] += ((sensorEvent.values[i] + prevAcceleration[i]) /
2.0f) * dt;
        }
        if (sensorEvent.values[0] >= A_THRESHOLD) {
            position[0] += velocity[0] * dt * Math.cos(theta[1]);
            position[2] += velocity[2] * dt * Math.sin(theta[1]);
        }
        System.arraycopy(sensorEvent.values, 0, prevAcceleration, 0, 3);
        prevTimestampAcc = sensorEvent.timestamp;
        Log.d(TAG, "--- r(X, Y, Z) = (" + position[0] + ", " +
                position[1] + ", " +
                position[2] + ")"
        );
    }
```

اولین نکته‌ای که بایستی به آن توجه شود کم کردن شتاب گرانش است که سنسور از نوع LINEAR_ACCELERATION این موضوع را مدیریت می‌کند.

با توجه به نویز داشتن سنسورها، مقدار شتاب اعمال شده حتی در حالت سکون نیز مقداری ناصفر بود. برای جلوگیری از افزایش/کاهش مختصات تا ابد، با در نظر گرفتن یک threshold از شتاب‌های بسیار کوچک صرف نظر شد.

در آرایه‌ای به طول ۳ به اسم velocity، سرعت لحظه‌ای گوشی در هر بعد نگهداری می‌شود. با رخ دادن هر event جدید، شتاب لحظه‌ای میانگین شتاب قبلی و شتاب لحظه فعلی در نظر گرفته می‌شود. بازه زمانی نیز dt است. طبق رابطه

$$v = v_0 + \frac{a_{\text{prev}} + a_{\text{cur}}}{2} \times dt$$

مقدار آن به شکل

```
velocity[i] += ((sensorEvent.values[i] + prevAcceleration[i]) / 2.0f) * dt;
```

بروزرسانی می‌شود. نهایتا با داشتن سرعت در هر بعد و زاویه با محورها که توسط gyroscopeHandler محاسبه شده است، مختصات گوشی (x و y) به شکل زیر محاسبه می‌شود:

```
if (sensorEvent.values[0] >= A_THRESHOLD) {
    position[0] += velocity[0] * dt * Math.cos(theta[1]);
    position[2] += velocity[2] * dt * Math.sin(theta[1]);
}
```

که در عبارات بالا

theta[1]: Angle with y axis

است.

- سنسور gyroscope:

با رخ دادن هر event روی این سنسور، زاویه گوشی با محورها بروزرسانی می‌شود که برای محاسبه مختصات گوشی مورد استفاده قرار خواهد گرفت.

```
public void handleGyroscope(SensorEvent sensorEvent) {
    Log.d(TAG, "onGyroscopeChanged:");
    float omegaX = sensorEvent.values[0];
    float omegaY = sensorEvent.values[1];
    float omegaZ = sensorEvent.values[2];
    Log.d(TAG, "--- omega(X, Y, Z) = (" + omegaX + " , " +
            omegaY + ", " +
            omegaZ + ")"
    );
    float dt = (sensorEvent.timestamp - prevTimeStampGyro) * NS2S;
    dt = Math.min(dt, 0.16f);
    float omega = (float)Math.sqrt(omegaX*omegaX + omegaY*omegaY +
omegaZ*omegaZ);
    for (int i = 0; i < 3; i++) {
        if (Math.abs(omega) < AV_THRESHOLD)
            continue;
        theta[i] += ((sensorEvent.values[i] + prevOmega[i]) / 2f) * dt;
    }
    System.arraycopy(sensorEvent.values, 0, prevOmega, 0, 3);
    prevTimeStampGyro = sensorEvent.timestamp;
    Log.d(TAG, "--- theta(X, Y, Z) = (" + theta[0] + ", " +
            theta[1] + ", " +
            theta[2] + ")"
    );
}
```

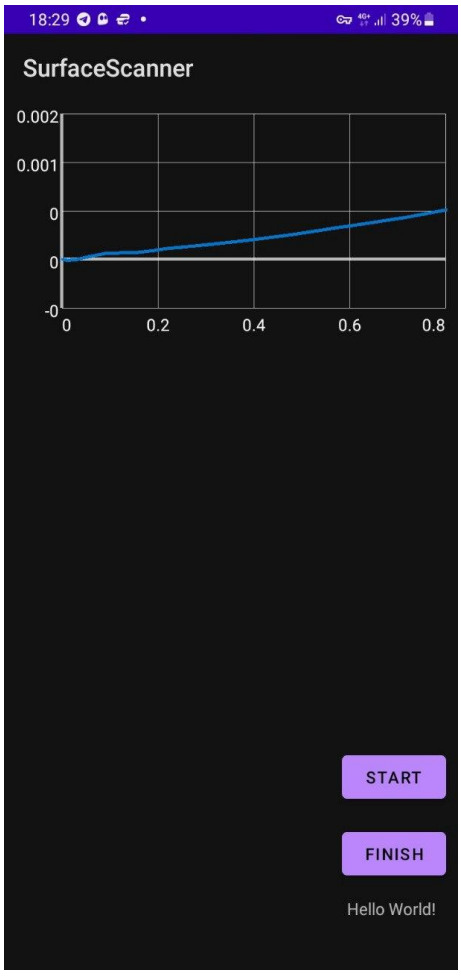
رسم نمودار

با رخ دادن هر event روی هر کدام از سنسورها، یک نمونه از مختصات گوشی در لیست xyValueArray اضافه می‌شود (انتهای تابع onSensorChanged) که بعدا برای رسم نمودار استفاده خواهد شد.

نتایج

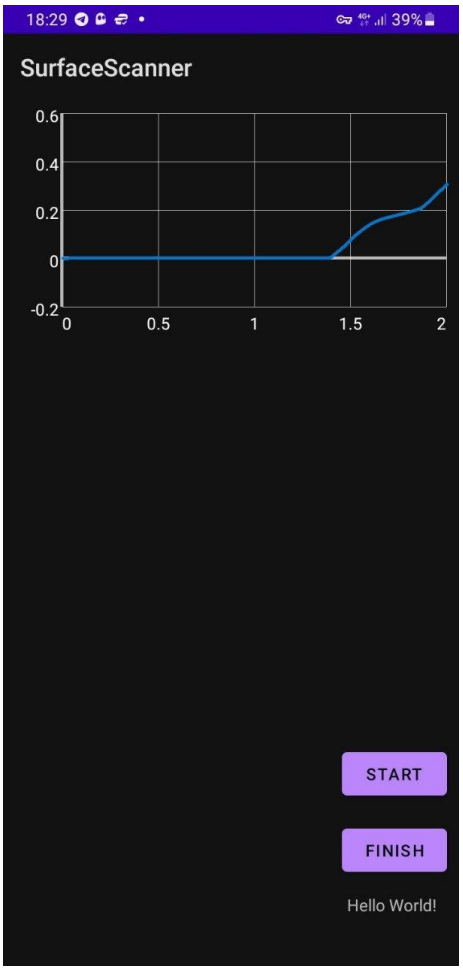
پس از پیاده سازی اپلیکیشن، آن را بر روی یک گوشی میریزیم تا تست کنیم. برای نمایش کارایی برنامه سه تست را در گزارش میاوریم: سطح صاف، سطح شیبدار و حرکت موجی!

سطح صاف: در یک سطح صاف گوشی را حرکت می‌دهیم.



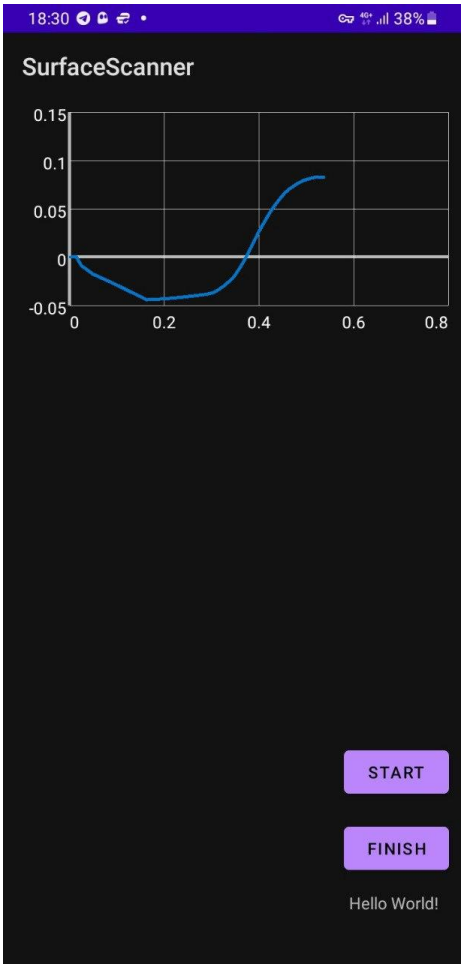
این نتیجه ممکن است که در نگاه اول اشتباه بنظر برسد، اما دقت کنید که از ۰ شروع می‌کنیم و به ۰ هم ختم می‌شویم. این تغییر ارتفاع کم ناشی از ارور های ناچیزی است که در سنسور وجود دارد. همینطور ممکن است سطح امتحان شده مقدار کمی شیب داشته باشد!

سطح شیبدار: در یک سطح شیبدار گوشی را حرکت می‌دهیم.



سطح شیبدار استفاده شده یک کتاب بوده که با دست نگه داری شده است، به دلیل نادقیق بودن دست انسان کتاب در طول تکان دادن کمی جابه جا میشود که در تصویر کاملا مشهود است. در این عکس مشاهده می‌کنید که سنسور ما به دقیق ترین حالت ممکن در حال کار کردن است!

حرکت موجی: در این حالت گوشی را به صورت موجی تکان می‌دهیم.



همانطور که در تصویر بالا می‌بینید به وضوح حرکت موجی ما بر روی نمودار نمایش پیدا می‌کند.

سوالات

۱. با استفاده از ابزار Perfetto که توسط تی‌ای درس در گروه اسکایپ پیشنهاد داده شد، مطابق دستورالعمل‌های https://perfetto.dev/docs/quickstart/android-tracing عمل شد:

```
$ ./record_android_trace -o trace_file.perfetto-trace -t 30s -b 32mb \
sched freq idle am wm gfx view binder_driver hal dalvik camera input res
memory
```

خود ابزار record_android_trace از طریق لینک زیر قابل دسترسی است:

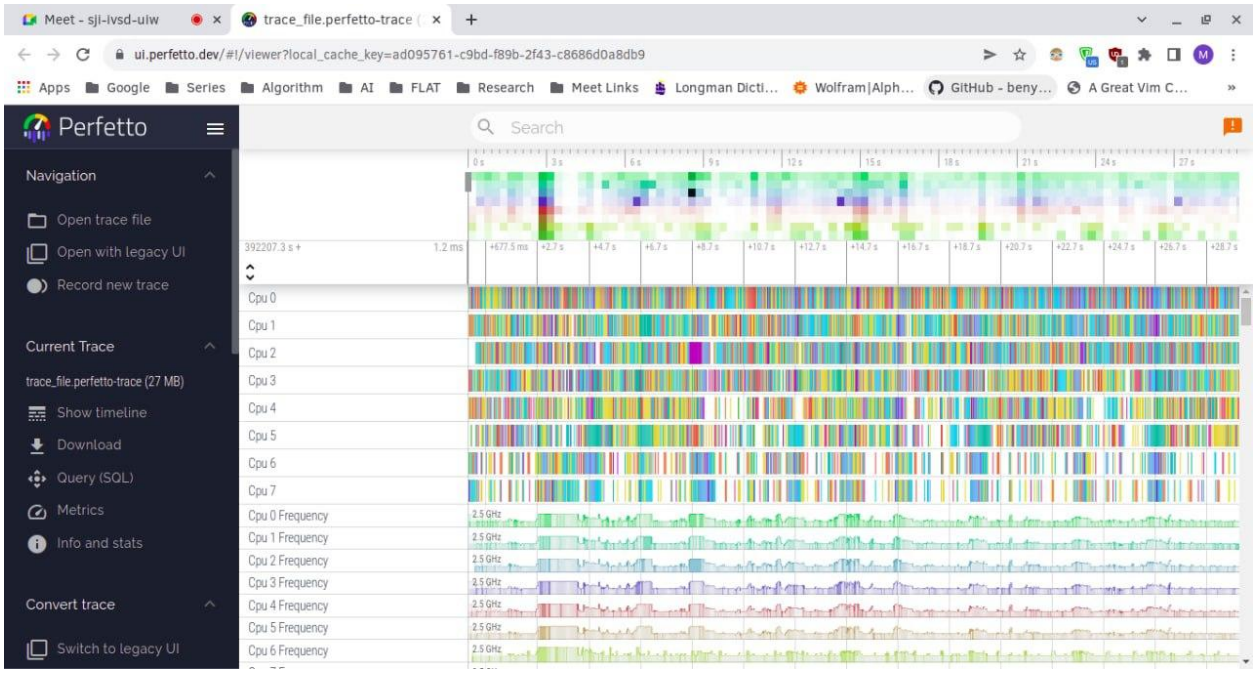
https://raw.githubusercontent.com/google/perfetto/master/tools/record_android_trace

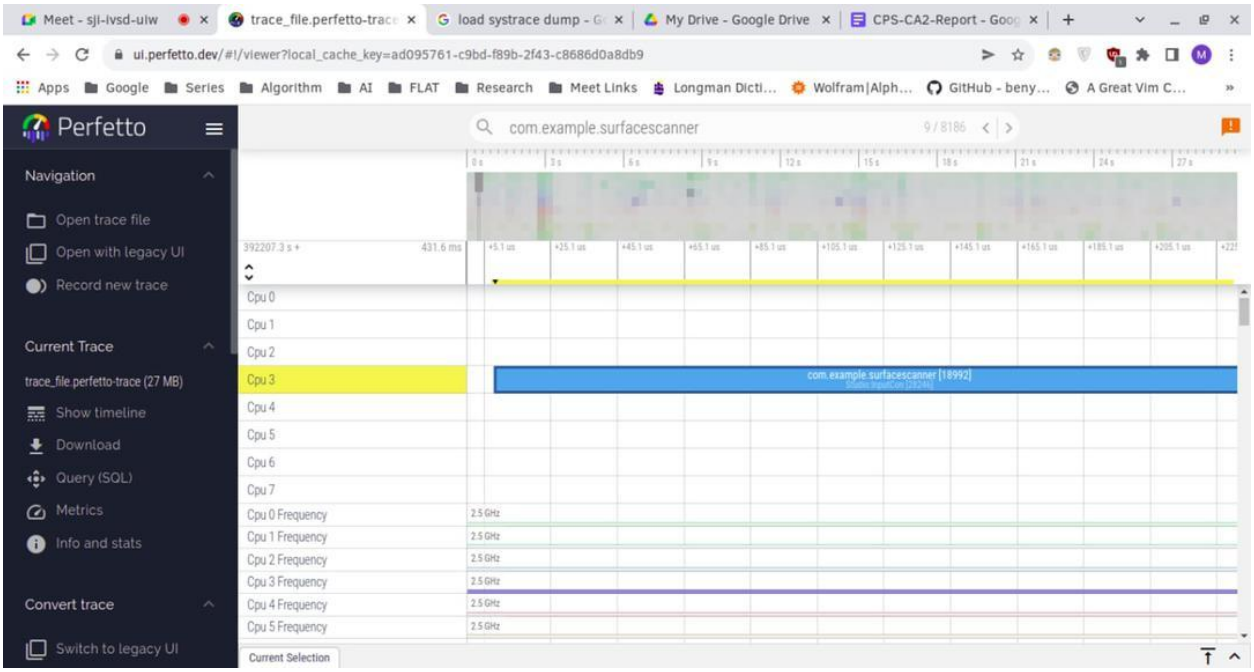
خروجی متنی این ابزار در خط فرمان به صورت زیر می‌باشد:

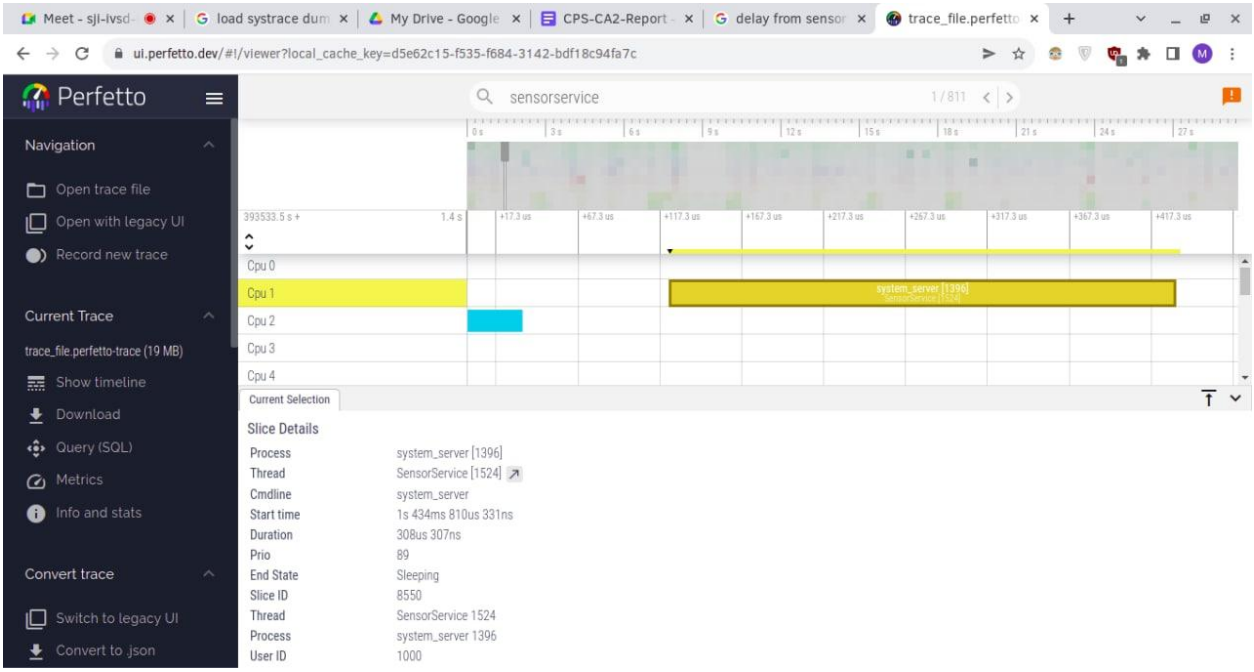
```
Running perfetto --background --txt -o
/data/misc/perfetto-traces/2022-05-16_22-32-2656ca.pftrace -t 30s -b 32mb
sched freq idle am wm gfx view binder_driver hal dalvik camera input res
memory
Trace started. Press CTRL+C to stop
--------- beginning of main
I/perfetto(  890): probes_producer.cc:230  Ftrace setup (target_buf=3)
I/perfetto(  890): ftrace_procfs.cc:176    enabled ftrace
I/perfetto(  890): probes_producer.cc:329  Producer stop (id=5)
I/perfetto(  890): ftrace_procfs.cc:183    disabled ftrace
I/perfetto(  890): probes_producer.cc:329  Producer stop (id=6)
I/perfetto(31518): perfetto_cmd.cc:810     Wrote 26205890 bytes into
/data/misc/perfetto-traces/2022-05-16_22-32-2656ca.pftrace
I/perfetto(  891): ng_service_impl.cc:1948 Tracing session 3 ended, total
sessions:0


Pulling into ./trace_file.perfetto-trace
/data/misc/perfetto-traces/2022-05-16_22-32-2656ca.pftrace: 1 file pulled.
23.0 MB/s (26205890 bytes in 1.085s)


Opening the trace (./trace_file.perfetto-trace) in the browser
127.0.0.1 - - [16/May/2022 22:33:00] code 404, message File not found
127.0.0.1 - - [16/May/2022 22:33:00] "POST /status HTTP/1.1" 404 -
```

```
127.0.0.1 - - [16/May/2022 22:33:00] "GET /trace_file.perfetto-trace
HTTP/1.1" 200 -
```

پس از اتمام فرآیند بالا نتایج با جزئیات کامل در وبسایت https://ui.perfetto.dev نشان داده می‌شوند که تصاویر آن در ادامه آمده است.

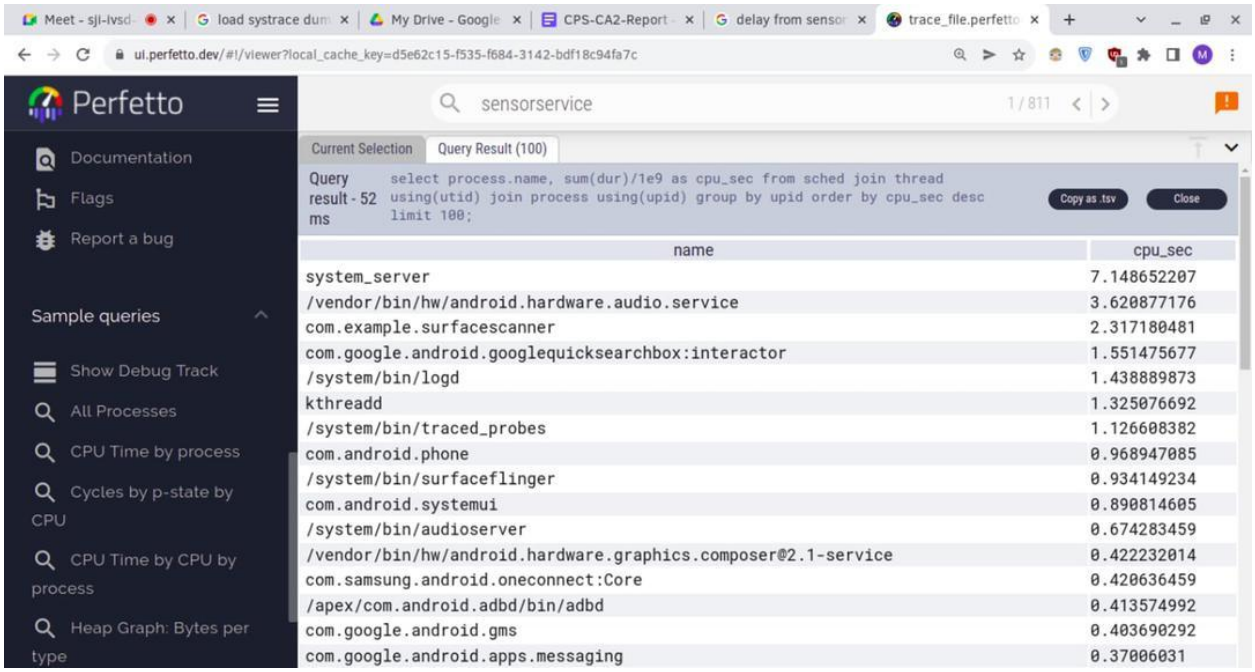


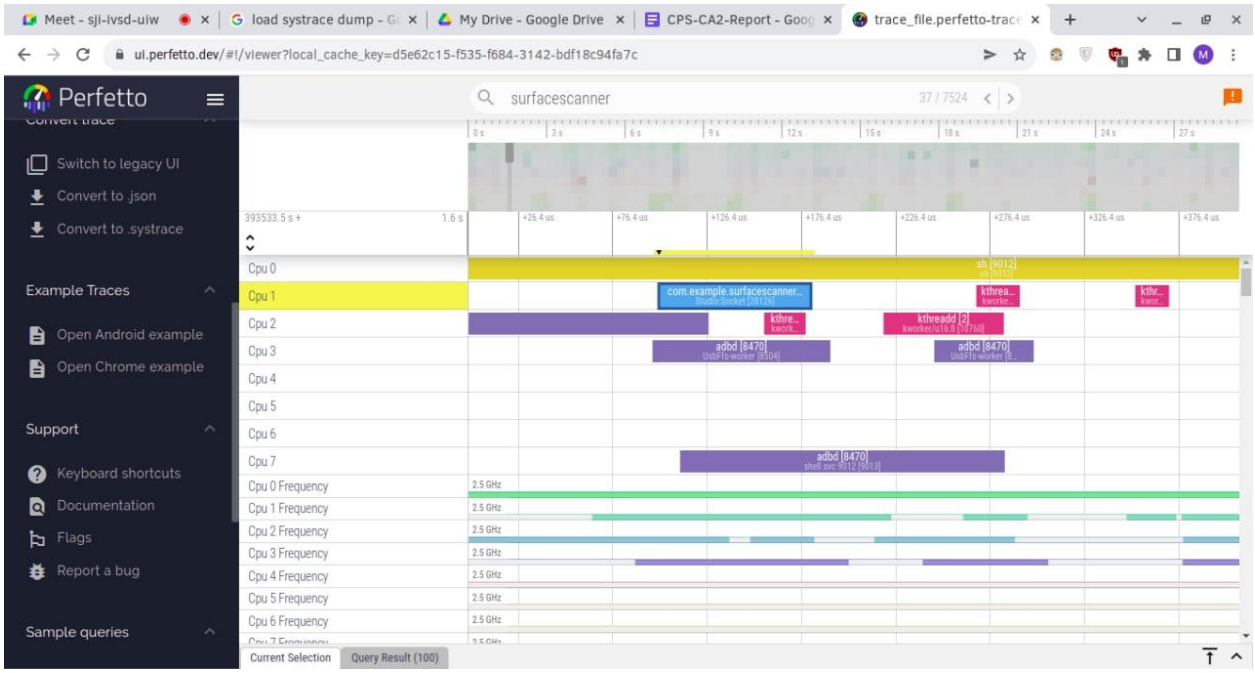




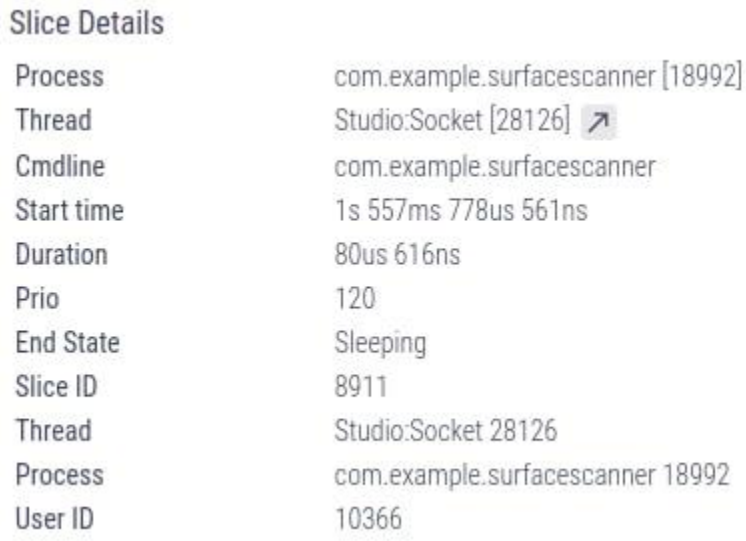
برای مثال در تصویر بالا duration time اندازه بازه زمانی پاسخ‌دهی سنسور را نشان می‌دهد.

ابزار Perfetto کارایی‌هایی بیش‌تر از صرف نمودار برای تحلیل این خروجی به ما می‌دهد؛ برای مثال، می‌توانیم کوئری‌هایی مانند کوئری زیر برای خروجی این نمودار تولید کنیم.


Meet - sji-ivsd
load systrace dum
My Drive - Google
CPS-CA2-Report
delay from senso
trace_file.perfetto
ui.perfetto.dev/#!/viewer?local_cache_key=d5e62c15-f535-f684-3142-bdf18c94fa7c
Perfetto
Documentation
Flags
Report a bug
Sample queries
Show Debug Track
All Processes
CPU Time by process
Cycles by p-state by CPU
CPU Time by CPU by process
Heap Graph: Bytes per type
sensorservice
1 / 811
Current Selection
Query Result (100)
Query result - 52 ms
select process.name, sum(dur)/1e9 as cpu_sec from sched join thread using(utid) join process using(upid) group by upid order by cpu_sec desc limit 100;
Copy as .tsv
Close
name
cpu_sec
system_server 7.148652207
/vendor/bin/hw/android.hardware.audio.service 3.620877176
com.example.surfacescanner 2.317180481
com.google.android.googlequicksearchbox:interactor 1.551475677
/system/bin/logd 1.438889873
kthreadd 1.325076692
/system/bin/traced_probes 1.126608382
com.android.phone 0.968947085
/system/bin/surfaceflinger 0.934149234
com.android.systemui 0.890814605
/system/bin/audioserver 0.674283459
/vendor/bin/hw/android.hardware.graphics.composer@2.1-service 0.422232014
com.samsung.android.oneconnect:Core 0.420636459
/apex/com.android.adbd/bin/adbd 0.413574992
com.google.android.gms 0.403690292
com.google.android.apps.messaging 0.37006031


Meet - sji-ivsd-uiw
load systrace dump - G
My Drive - Google Drive
CPS-CA2-Report - Goog
trace_file.perfetto-trace
ui.perfetto.dev/#!/viewer?local_cache_key=d5e62c15-f535-f684-3142-bdf18c94fa7c
Perfetto
Switch to legacy UI
Convert to .json
Convert to .systrace
Example Traces
Open Android example
Open Chrome example
Support
Keyboard shortcuts
Documentation
Flags
Report a bug
Sample queries
surfacescanner
37 / 7524
Cpu 0
Cpu 1
Cpu 2
Cpu 3
Cpu 4
Cpu 5
Cpu 6
Cpu 7
Cpu 0 Frequency
Cpu 1 Frequency
Cpu 2 Frequency
Cpu 3 Frequency
Cpu 4 Frequency
Cpu 5 Frequency
Cpu 6 Frequency
com.example.surfacescanner
adbd [8470]
kthreadd [2]
Current Selection
Query Result (100)

۲. ابتدا به عکس زیر که مربوط به یکی از رویداد[1]های ضبط شده توسط ابزار Perfetto می‌باشد، دقت کنیم:



با توجه به عکس بالا، میزان جزئیاتی که توسط ابزار Perfetto خروجی داده می‌شود، در سطح پردازه و نخ اجرایی‌است و نمی‌توانیم جزئیاتی مربوط به فراخوانی توابع در سطح پردازه‌ها را دنبال کنیم. در نتیجه، بدست آوردن خواسته‌ی مورد نظر سوال به کمک خروجی Perfetto امکان‌پذیر نیست.

۳. در دستگاه‌های اندروید، سنسورهای متفاوت دقت‌های متفاوتی برای نمونه‌برداری دارند؛ برای مثال، کوتاه‌ترین بازه‌ی نمونه‌برداری برای سنسور شتاب‌سنج و شدت نور متفاوت هستند؛ از این رو، برای نمونه‌برداری از سنسورهای متفاوت، از دقت‌های متفاوتی استفاده می‌کنیم.
در پیاده‌سازی ما، برای تاخیر نمونه‌برداری برای هر دو سنسور ژیروسکوپ و شتاب‌سنج از مقدار ثابت SENSOR_DELAY_NORMAL استفاده می‌کنیم. این مقدار بطور تقریبی برابر ۲۰۰ میلی‌ثانیه است و برای کارکرد برنامه‌ی ما این دقت کافی بود! اگرچه که برای بدست آوردن این مقدار تست های زیادی گرفته شده و در نهایت بهترین مقدار بدست آمده به جای این متغیر قرار داده شده است.

۴. عیب اصلی استفاده از NDK نداشتن cross-platform capability و دسترسی به برخی از لایبری های سطح بالاتر است. اما از مزایای NDK می‌توان به پرفورمنس بالاتر و در نتیجه سرعت بالاتر اپلیکیشن اشاره کرد. در برنامه اسکن سطح، نیازی به cross-platform capability خاصی نداریم. همچنین با توجه به real-time بودن این اپلیکیشن، پرفورمنس برنامه امتیاز بزرگی محسوب می‌شود.

[1] event

با توجه به پیچیده نبودن اپلیکیشن اسکن سطح و نیاز نداشتن به ابزارهای زیاد، نیازی ضروری به استفاده از SDK وجود ندارد. اما برای سهولت پیاده سازی و دسترسی داشتن به لایبری‌های مدیریت سنسورها، استفاده از آن کار را برای ما راحت تر کرد.

۵. سنسورهای hardware-based مؤلفه‌های فیزیکی هستند که داده‌های خود را به صورت مستقیم از اندازه گیری خواص محیطی مانند شتاب یا تغییرات زاویه‌ای به دست می‌آورند.
اما سنسورهای software-based دستگاه‌های فیزیکی نیستند و تنها روش مورد استفاده‌ی سنسورهای hardware-based را تقلید می‌کنند. این سنسورها داده‌های خود را از تعدادی سنسور hardware-based دریافت می‌کنند و با نام‌های virtual-sensor یا composite-sensor هم شناخته می‌شوند. برای نمونه سنسور محاسبه‌ی نزدیکی و step-counter از این نوع سنسور هستند. با توجه به تعاریف بالا، هر دو سنسور accelerometer و gyroscope از نوع hardware-based هستند.

۶. جواب این بخش را با استفاده از داک های رسمی android می‌دهیم. برای توضیح این سوال باید توضیح دهیم که یک SoC و یا همان چیپ دستگاه ما، در حالت کلی در سه استیت مختلف می‌تواند قرار داشته باشد. یا روشن است، یا idle است و یا suspend. در حالت idle چیپ ما روشن است اما کار خاصی را انجام نمی‌دهد و در حالت suspend به طور کامل خاموش است و برقی به آن تامین نمی‌شود. حال به سراغ جواب دادن سوال می‌رویم.

سنسورهای non-wake-up سنسورهایی هستند که اجازه جلوگیری از چیپ برای ورود به حالت suspend را ندارند و نمی‌توانند چیپ را برای داده رسانی بیدار کنند. این سنسورها در صورتی که بخواهند در حالت suspend گوشی به آن اطلاعات گزارش کنند باید این وظیفه را به اپلیکیشن پدر خود بسپارند که قفلی را نگه دارند تا به آنها این اجازه را بدهد. در حالتی که چیپ در suspend است، دیتاها وارد یک fifo می‌شوند و ذخیره می‌شوند و به محض اینکه سیستم از حالت suspend در بیاید این fifo خالی می‌شود و اطلاعات ذخیره شده گزارش می‌شود. برنامه هایی که از non-wake-up سنسورها استفاده می‌کنند یا باید قفل مربوط به suspend نشدن سیستم را بگیرند و یا در صورتی که این کار را نمیکنند توقع داشته باشند که در صورت suspend شدن دستگاه مقداری از اطلاعات را از دست بدهند.

از طرف دیگر، سنسور های wake-up در طرف دیگر قرار می‌گیرند. این چیپ ها تضمین میکنند که دیتای آنها فارغ از وضعیت چیپ همیشه دلیور شود. در صورتی که دیوایس suspend نباشند که این سنسور ها تفاوتی با سنسور های غیر wake-up ندارند. اما در صورتی که چیپ در حالت suspend باشد، آن را از حالت suspend درمی‌آورند، اطلاعات را دلیور می‌کنند و سپس دوباره آن را suspend می‌کنند.

حال با دانستن این اطلاعات، اگر سنسور ما در حالت wake-up باشد، می‌تواند بدون وقفه و بدون مشکل دیتا را به صورت مداوم به سنسور ما بفرستد. اما در صورت non-wake-up به دلیل تاخیر هایی که ممکن است در گرفتن اطلاعات رخ دهد، شکل نمودار ما نا دقیق تر می‌شود. اگر چه که استفاده از سنسور های non-wake-up مصرف انرژی و سی پی یو کمتری را در کل نتیجه می‌دهند.