به نام خدا

سیستم های نهفته بی درنگ

گزارش تمرین کامپیوتری دوم(surface scanner)

سینا سلیمیان - 810197528

حمید خدادادی - 810197499

نازنین یوسفیان - 810197610

ملیکا مرافق - 810197581

توضیح کد

برنامه با MainActivity آغاز می شود. در view آن یک دکمه با نام start وجود دارد که فشردن آن activity مربوط به خواندن سنسورها را فعال می کند. با کلیک بر این دکمه یک activity جدید (ReadSensorData) فعال می شود. در view این activity دو عدد نمایش داده می شود که عدد اول طول طی شده و عدد دوم ارتفاع است. دکمه stop در این view باعث فراخوانی activity بعدی با نام LineGraph می شود که نمودار مورد نظر را نمایش می دهد.

در ادامه کد نوشته شده مفصل تر توضیح داده خواهد شد:

دکمه start در صفحه اصلی یک attribute با نام onClick دارد که تابع sendMessage را فراخوانی می کند. در این تابع یک intent ساخته می شود و activity مربوط به خواندن سنسورها را فعال می کند.

```
public void sendMessage(View view) {
    Intent intent = new Intent(this, ReadSensorData.class);
    startActivity(intent);
}
```

حال به توضیح کلاس ReadSensorData می پردازیم.

فیلدهای این کلاس عبارتند از:

List<Float> xList,hList: لیستی است که عدد طول و ارتفاع را در هنگام تغییر هر کدام ذخیره می کند و برای رسم نمودار نهایی استفاده می شود.

SensorManager sensorManager: برای استفاده از امکانات سنسورهای گوشی استفاده می شود.

Sensor accelerometer ,gyroscope: دو سنسوری است که در انجام تمرین از آن ها استفاده شده است. از سنسور accelerometer برای پیدا کردن طول طی شده در جهت x و سرعت در جهت z استفاده می شود. بر اساس تغییر سرعت در جهت z رنج سمپل گیری از سنسور ژیروسکوپ تعیین می شود. از سنسور ژیروسکوپ برای تعیین سرعت زاویه ای و در نهایت به دست آوردن ارتفاع استفاده می شود.

float x, vx: سنسور شتاب سنج، شتاب در جهت x را به ما می دهد و لازم داریم که از روی آن سرعت و مکان در این جهت را بدست آوریم. برای این کار نیاز به انتگرال گیری داریم که به روش عددی برابر است با جمع مقادیر به دست آمده در بازه های زمانی بسیار کوچک. به همین دلیل نیاز داریم جمع سرعت و مکان تا به اینجا را در هر لحظه داشته باشیم و آن را آپدیت کنیم.

float lastValuex, lastValuez: سنسور شتاب سنج استفاده شده دارای مقداری خطا است و نویز دارد. هنگامی که گوشی به طور ثابت در یک مکانی قرار می گیرد، عدد بدست آمده از سنسور صفر نیست و بدین منظور لازم است که یک threshold تعریف شود تا اگر مقدار تغییر عدد سنسور از آن بیشتر بود، داده ی آن در نظر گرفته شود. این دو متغیر مقدار شتاب در جهت x و z را در مرحله قبل ذخیره می کنند و برای محاسبه threshold استفاده می شوند.

float accelerometerTimestamp, gyroscopeTimestamp: هنگامی که سنسور داده جدیدی دریافت می کند و عدد آن تغییر می کند، همراه با آن می توانیم زمان تغییر بر حسب نانو ثانیه را بدست آوریم. برای اینکه اختلاف زمانی در هر مرحله را بدانیم، نیاز داریم که timestamp مرحله قبل را داشته باشیم.

float h, theta: سنسور ژیروسکوپ به ما تغییر سرعت زاویه ای را می دهد. برای اینکه بتوانیم ارتفاع را محاسبه کنیم، لازم است که بر اساس زاویه ای که داریم فرمول زیر را برای محاسبه ارتفاع استفاده کنیم:

$$h \mathrel{+}= \sin\Theta * \text{mobile's width}$$

جون عدد به دست آمده از سنسور تغییرات سرعت زاویه ای است و نه زاویه در لحظه، لازم است یک متغیر کلی به این منظور داشته باشیم که زاویه را در هر لحظه با توجه به سرعت آپدیت کند و مجموع زاویه های طی شده نگهداری شود تا زاویه در لحظه را داشته باشیم.

GraphView graphView : برای رسم نمودار آنلاین از آن استفاده میکنیم.

در متد onCreate این کلاس، دو سنسور مورد نظر را تعریف می کنیم که از جنس Sensor.TYPE_LINEAR_ACCELERATION و Sensor.TYPE_GYROSCOPE هستند و مقدار اولیه متغیرها را مشخص می کنیم. با setGraphAtrr() مشخصات نمودار آنلاین را تعیین میکنیم.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_read_sensor_data);
    sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION);
    gyroscope = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
    setGraphAttr();
    accelerometerTimestamp = 0;
```

```
    gyroscopeTimestamp = 0;
    x = 0;
    lastValuex = 0;
    vx = 0;
    h = 0;
    theta = 0;
    hList = new ArrayList<>();
    xList = new ArrayList<>();


}
```

متد onSensorChanged هنگامی صدا زده می شود که تغییری در داده سنسورها ایجاد شود. پس ابتدا نیاز داریم که بدانیم کدام سنسور تغییر کرده است و با sensor.getType تعیین می کنیم سنسور ژیروسکوپ تغییر داشته یا شتاب سنج و سپس تابع مربوط به هرکدام را صدا می زنیم تا عددهای مورد نیاز را به دست آوریم. پس از آن نیز عددهای مربوطه را به آرایه ها اضافه می کنیم تا در ادامه بتوانیم بر اساس آن نمودار را رسم کنیم. هم چنین لازم داریم که rate سمپل گیری از سنسور ژیروسکوپ را با توجه به تغییر سرعت که از شتاب سنج به دست می آید تعیین کنیم.

```
@Override
public final void onSensorChanged(SensorEvent event) {
    if(event.sensor.getType() == Sensor.TYPE_LINEAR_ACCELERATION) {
        accelerationChange(event);
```

```
        sensorManager.registerListener(this, gyroscope, (int) (100 / Math.abs(vx)));
    }
    else
        gyroscopeChange(event);
    xList.add(x);
    hList.add(h);
    lineGraph(xList,hList);
}
```

همانطور که گفته شد، ژیروسکوپ تغییر سرعت زاویه ای را مشخص می کند و بر اساس سینوس زاویه می توان طبق فرمولی که در بالا گفته شد، ارتفاع را مشخص کرد. برای پیاده سازی آن لازم داریم که تغییر زمانی از مرحله قبل را محاسبه کرده و در عددی که سنسور در راستای y به ما می دهد ضرب کنیم تا بتوانیم زاویه را به دست آوریم. سپس این زاویه را با زاویه های طی شده تا اینجا که در متغیر theta ذخیره شده جمع می کنیم. برای به دست آوردن ارتفاع نیز نیاز است که سینوس زاویه را با ارتفاعی که از مرحله قبل داشتیم، جمع کنیم. هم چنین لازم است یک threshold برای شتاب در راستای z در نظر بگیریم تا در صورتی که موبایل ساکن بود، ارتفاع افزایش پیدا نکند. عدد داده شده به علت نویز سنسور است که در حالت سکون، شتاب را صفر نشان نمی دهد. هم جنین بر اساس تست های انجام شده لازم بود تا ضریبی به اعداد اضافه شود و این ضریب هنگام افزایش و کاهش متفاوت است تا تقریبا اعداد قرینه تولید شود.

```
private void gyroscopeChange(SensorEvent event) {
    Float y = event.values[1];
    Float alpha = 1.f;
    Float dT = (event.timestamp - gyroscopeTimestamp) / 1000000000.0f;
    if (lastValuez > 0.02 || lastValuez < -0.02) {
        theta += y * dT;
        if (Math.sin(theta) > 0)
            alpha = 2.5f;
        else
            alpha = 5f;
        h -= (float) Math.sin(theta) * alpha;
    }
    gyroscopeTimestamp = event.timestamp;
}
```

برای تغییرات سنسور شتاب سنج، تابع accelerationChange تعریف شده است که تغییرات در راستای x و z را به طور جداگانه محاسبه می کند.

```
private void accelerationChange(SensorEvent event) {
    xChange(event);
    zChange(event);
    accelerometerTimestamp = event.timestamp;

}
```

برای محاسبه مکان فعلی لازم است که دوبار از شتاب که توسط سنسور به دست می آید انتگرال بگیریم که به صورت عددی برابر است با اینکه شتاب ها در بازه های زمانی کوچک را با یکدیگر جمع کرده به سرعت برسیم و به همین روش از سرعت به مکان برسیم. ضریب های گذاشته شده برای اسکیل است تا به اعداد واقعی تری برسیم. عدد بدست آمده نهایی بر حسب سانتی متر است.

```
private void xChange(SensorEvent event) {
   Float ax = abs(event.values[0]);
   Float difference = ax - lastValuex;
   int alpha = 1;
   if(difference > 0.2 || difference < -0.2) {
       float dT = (event.timestamp - accelerometerTimestamp) / 1000000000.0f;
       vx += ax  * dT;
       if (x < 10)
           alpha = 50;
       else if (x < 50)
           alpha = 10;
       else if(x < 100)
           alpha = 7;
       else if (x < 500)
           alpha = 4;
       else
           alpha = 2;
       x += vx * dT * alpha;

   }
   lastValuex = ax;
}
```

برای شتاب در راستای z نیز تنها عدد را از سنسور دریافت می کنیم و در lastValuez ذخیره می کنیم.

```
public void stop(View view){

   Intent intent = new Intent(this, LineGraph.class);

   intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);


   float[] xArray = toFloatArray(xList);

   float[] hArray = toFloatArray(hList);
```

```
    intent.putExtra(X_ARRAY, xArray);
    intent.putExtra(H_ARRAY, hArray);
    startActivity(intent);
    finish();
}
```

در صفحه ی ReadSensorData یک دکمه stop داریم که attribute ی با نام onClick دارد که متد stop را فراخوانی می کند.

در متد stop یک intent ساختیم که activity مربوط به رسم نمودار با نام LineGraph را فعال میکند. آرایه ای از طول و ارتفاع را به intent اضافه میکنیم. `FLAG_ACTIVITY_CLEAR_TOP` را فعال می کنیم تا استک activiy پاک شود.

با فراخوانی startActivity یک Instance از LineGarph ایجاد میشود که در آن نمودار رسم میشود. در نهایت با فراخوانی finish به کار این activity خاتمه می دهیم.

نمودار

برای رسم نمودار از GraphView استفاده کردیم.

نمودار آنلاین

در () setGraphAttr ابتدا graphView را مقداردهی کرده و سپس ویژگی های نمودار اعم از عنوان نمودار، حداکثر مقدار x,y را مشخص میکنیم.

برای رسم نمودار آنلاین لازم است که در هر تغییر در سنسور ها نمودار رسم شود. برای این کار در onSensorChanged متد lineGraph را صدا می زنیم که لیستی از طول و ارتفاع را دریافت می کند و آرایه ای از dataPoint ها را با استفاده از آنها مقدار دهی میکند. سپس LineGraphSeries را با استفاده از این آرایه پر میکنیم. در اینجا لازم است که series قبلی را که در واقع در تغییر پیشین سنسور ساخته شده است را حذف کنیم و series جدید را اضافه کنیم.

```
 private void lineGraph(List<Float> xList, List<Float> hList){
   DataPoint[] dataPoints = new DataPoint[xList.size()];
   for(int i=0 ; i< xList.size() ; i++)
         dataPoints[i] = new DataPoint(xList.get(i),hList.get(i));

   LineGraphSeries<DataPoint> series = new LineGraphSeries<>(dataPoints);
   graphView.removeAllSeries();
   graphView.addSeries(series);

}
```

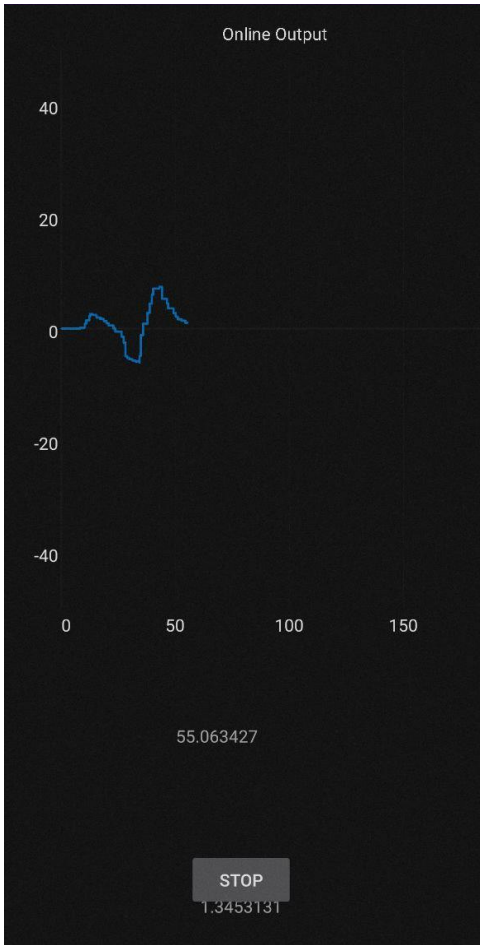
نمودار غیرآنلاین

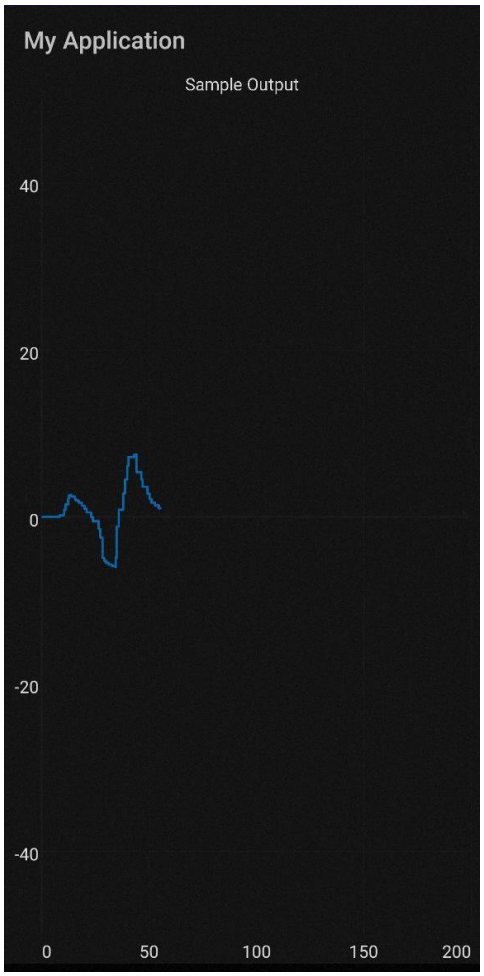
```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_line_graph);
   Intent intent = getIntent();
   float[] xArray =  intent.getFloatArrayExtra(ReadSensorData.X_ARRAY);
   float[] hArray =  intent.getFloatArrayExtra(ReadSensorData.H_ARRAY);

   LineGraphSeries<DataPoint> series = new LineGraphSeries<DataPoint>();
   for(int i=0 ; i< xArray.length ; i++)
    series.appendData(new DataPoint(xArray[i],hArray[i]),true,xArray.length);
```

همانطور که پیش تر گفته شد با زدن دکمه stop یك instance از LineGraph ساخته میشود. هنگام ساخته شدن این instance، ابتدا Intent را دریافت کرده سپس آرایه های طول و ارتفاع را که در متد stop به آن اضافه کرده بودیم را دریافت میکنیم. به LineGraphSeries ساخته شده DataPoint ها را اضافه می کنیم و در نهایت همانند حالت آنلاین، series را به graphView اضافه میکنیم.

خروجی برنامه



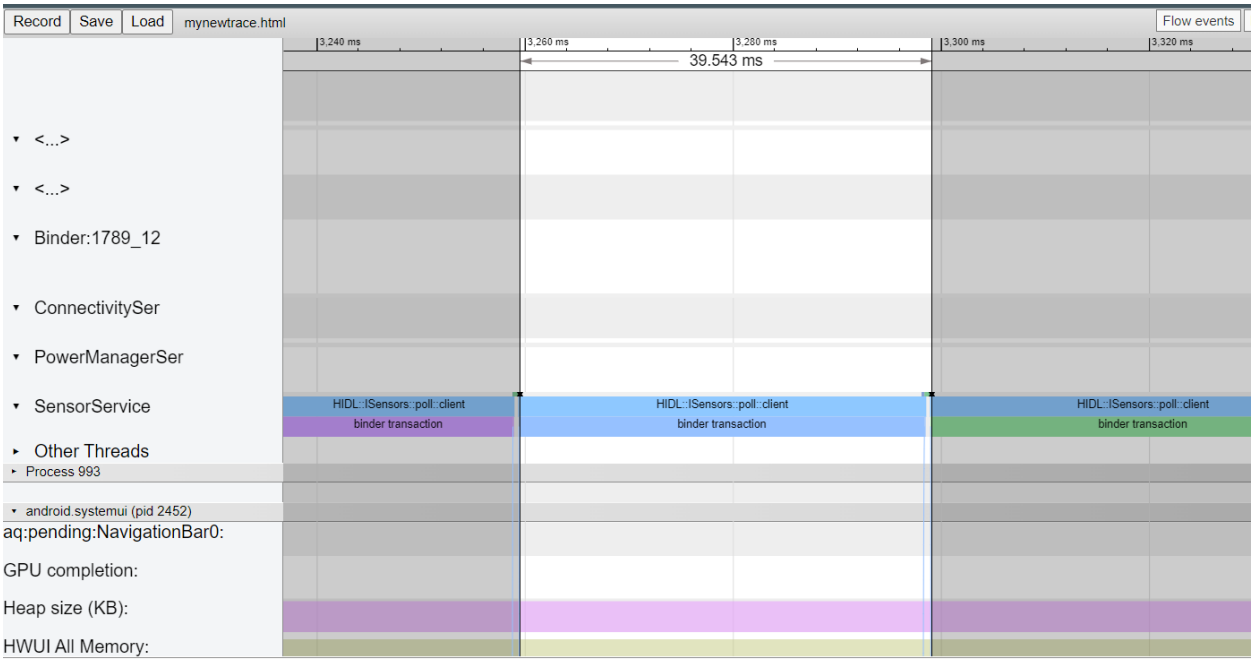


بخش امتیازی

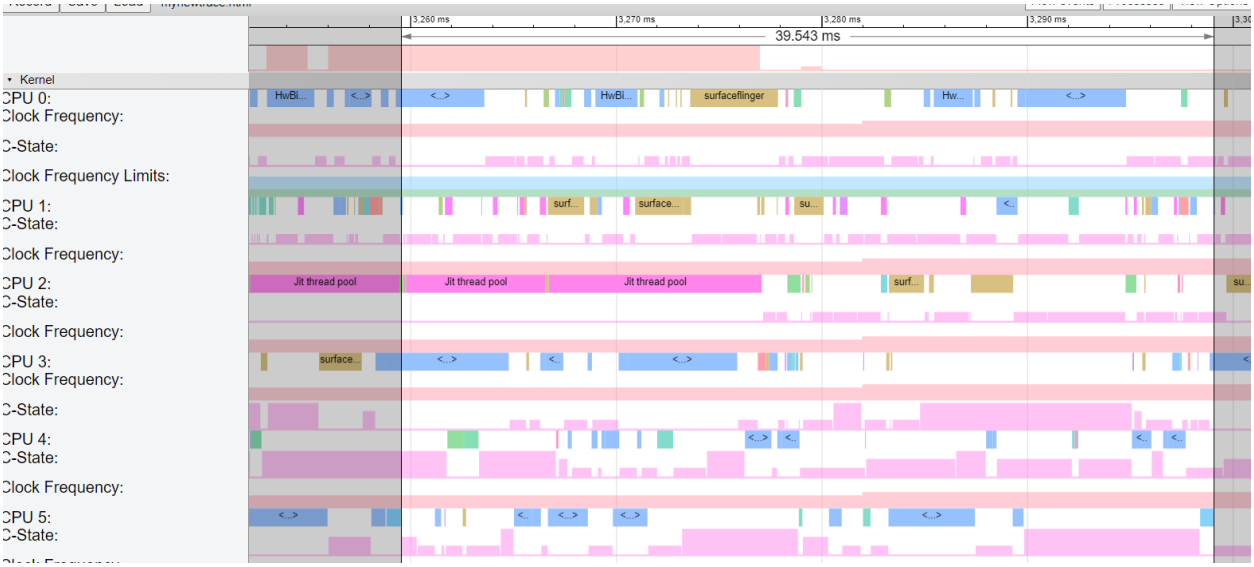
در حالت آنلاین دقت کمتری داریم زیرا با هر تغییر سنسور نمودار را رسم می کنیم و همچنین نیازمند انجام عملیات بیشتری هستیم. به عنوان مثال باید series قبل را پاک کنیم که در حالت غیر آنلاین نیازی به این کار نداشتیم.

سوالات

۱)



اطلاعات از سنسور حدود هر ۳۹ میلی ثانیه گرفته می شود.



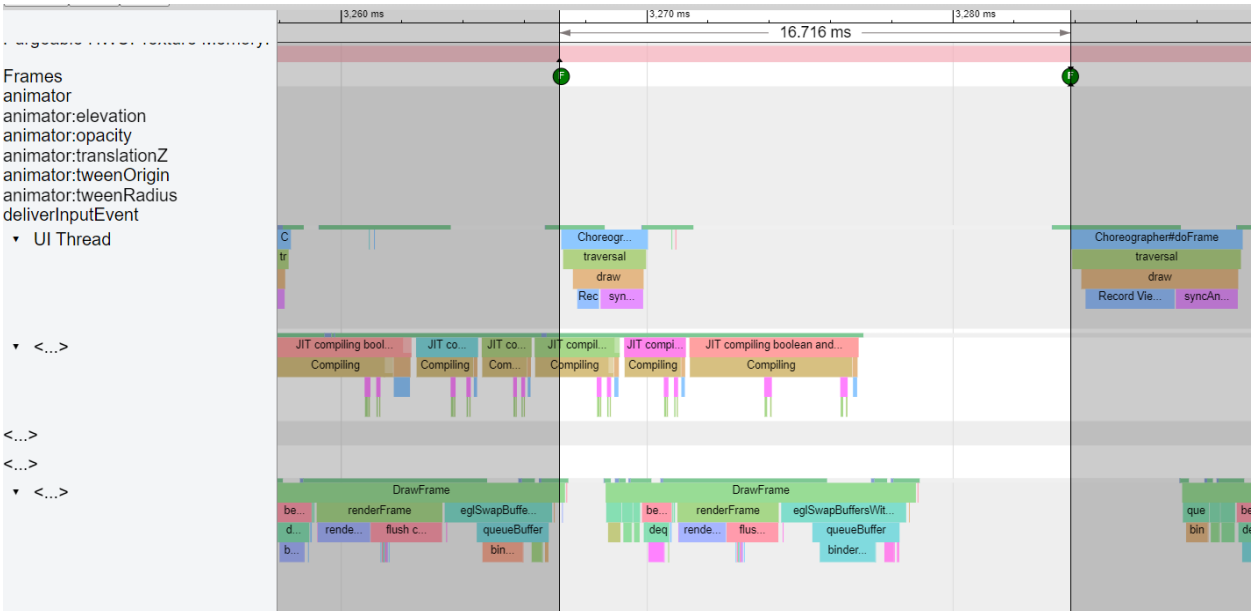
بیشتر اتفاقات مربوط به نمایش صفحه و ارتباط بین پردازه های مختلف است.

SurfaceFlinger مسئول دریافت تمامی سطوح برنامه ها و سیستم ها که قرار است نمایش داده شوند و ترکیب آنها در یک بافر است که در نهایت توسط display controller نمایش داده می شود.

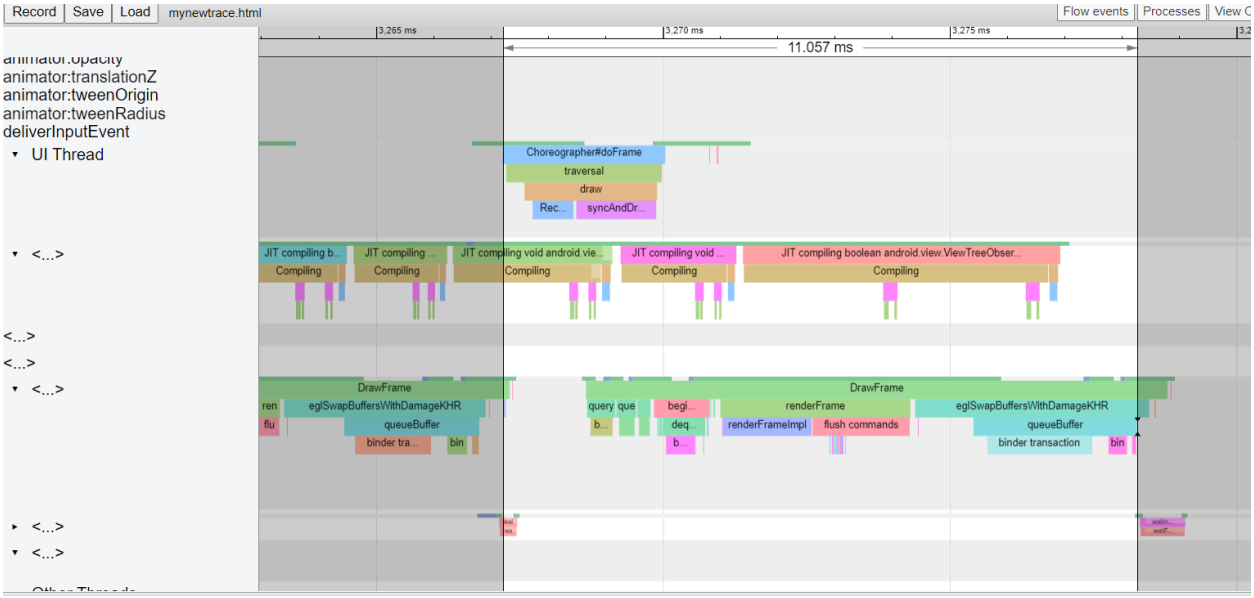
binder یک مکانیزم IPC است که فراخوانی remote متدها را در پراسس های دیگر امکان پذیر می کند.

HWBinder وظیفه انتقال اطلاعات سنسور ها از لایه فیزیکی به لایه اپلیکیشن را دارد.

(۲



فاصله زمانی بین دو فریم حدود ۱۶ میلی ثانیه است و در واقع صفحه با نرخ 60Hz به روز رسانی می شود.



حدود ۱۱ میلی ثانیه طول می کشد تا تغییرات اسکن شده از سطح بر روی صفحه نمایش نشان داده شود.

(۳

نرخ به روز رسانی صفحه نمایش به صورت استاندارد در اکثر گوشی ها 60Hz است و در برخی گوشی ها میتوان آن را تا 120Hz افزایش داد. با توجه به این موضوع در حالت استاندارد(60Hz) اگر بخواهیم داده ای را نمایش دهیم گذاشتن دوره تناوبی کمتر ازین مقدار بی فایده است. همچنین اگر دوره تناوب را خیلی کوچک بگیریم و نرخ به روز رسانی خیلی بالا باشد، سربار زیادی بر روی پردازنده ها خواهد بود و توان مصرفی افزایش می یابد.نرخ دریافت اطلاعات از شتاب سنج را برابر SENSOR_DELAY_UI قرار دادیم که حدود ۶۷ میلی ثانیه است.

(۴

NDK به ما اجازه می دهد تا کد نوشته شده به زبان C/C++ را در برنامه خود اجرا کنیم. چون برنامه به صورت مستقیم در پردازنده اجرا می شود، به جای آنکه توسط Dalvik Virtual Machine ترجمه شود، سرعتش افزایش می یابد.

همچنین کد نوشته شده به زبان C/C++ را می توان به راحتی در جاهای دیگری مانند ios و windows نیز استفاده کرد و برای مواقعی که برنامه ی multi platform می خواهیم بسازیم، مفید است.

NDK همچنین پیچیدگی برنامه را افزایش می دهد ولی کارایی برنامه را محدود می سازد. به همین دلیل، فقط در مواقع ضروری باید از آن استفاده کنیم.

- مناسب برای کار های سنگین برای پردازنده: بازی های موبایلی، پردازش سیگنال و ...
- استفاده از کد C/C++ ای که از قبل داریم در اندروید
- توسعه برنامه های multiplatform

SDK از طرف دیگر، از زبان جاوا استفاده می کند و شامل پروژه هایی برای نمونه، ابزار های توسعه و IDE می باشد. همچنین شامل API های رایج مورد نیاز برای برنامه های اندرویدی می باشد.

برخی از برنامه های اندرویدی از NDK استفاده می کنند تا به یک کارایی مشخص دست یابند. این به بیانی NDKو SDK را مکمل یکدیگر قرار می دهد.

- پورتابل بودن دستگاه را مستقل از معماری پردازنده تضمین می کند
- کتابخانه های مفید و جامع
- Automatic memory management

(۵

سنسور های hardware-based مولفه های فیزیکی ای هستند که داخل دستگاه ها (موبایل ها و تبلت ها) قرار گرفته اند. آن ها داده ها را با سنجش مستقیم ویژگی های به خصوصی از محیط به دست می آورند مانند: شتاب، قدرت جاذبه زمین، تغیر زاویه.

سنسور های software-based دستگاه های فیزیکی نیستند، اگرچه مانند سنسور های hardware-based رفتار می کنند. این سنسور ها داده های خود را از یک یا چند سنسور hardware-based دریافت می کنند و به آن ها virtual sensors یا synthetic sensors هم می گویند. سنسور شتاب سنج خطی و جاذبه از جمله این سنسور ها هستند.

TYPE_GYROSCOPE : hardware-based

TYPE_ACCELEROMETER: hardware-based

TYPE_LINEAR_ACCELERATION:software-based

(۶

:Non-wake-up sensors

سنسور هایی هستند که از رفتن SoC به حالت تعلیق جلوگیری نمی کنند و SoC را برای گزارش داده ها بیدار نمی کنند. به ویژه، درایور ها مجاز به نگه داشتن wake-lock ها نیستند. اگر برنامه‌ها می‌خواهند event ها را در حالی که صفحه نمایش خاموش است، از سنسور های non-wake-up دریافت کنند، وظیفه نگه داشتن یک partial wake lock برعهده خودشان است. در حالی که SoC در حالت تعلیق است، سنسور ها باید به کار خود ادامه دهند و event هایی را تولید کنند که در یک FIFO سخت افزاری قرار می گیرند. Eventهای موجود در FIFO زمانی که SoC بیدار می شود به برنامه ها تحویل داده می شوند. اگر FIFO برای ذخیره همه event ها خیلی کوچک باشد، event های قدیمی‌تر از بین می‌روند. قدیمی‌ترین داده‌ها حذف می‌شوند تا آخرین داده‌ها را در خود

جای دهند. در حالت شدید که FIFO وجود ندارد، تمام eventهای ایجاد شده در حالی که SoC در حالت تعلیق است از بین می روند. یک استثنا آخرین رویداد از هر حسگر در حال تغییر است: آخرین رویداد باید خارج از FIFO ذخیره شود تا از بین نرود.

به محض اینکه SoC از حالت تعلیق خارج شد، همه رویدادها از FIFO گزارش می شوند و عملیات به حالت عادی از سر گرفته می شود.

برنامه‌هایی که از non-wake-up sensors استفاده می‌کنند باید یک wake lock داشته باشند تا اطمینان حاصل شود که سیستم به حالت تعلیق نمی‌رود، زمانی که به حسگرها نیازی ندارند از آن ها unregister کنند، یا زمانی که SoC در حالت تعلیق است انتظار از دست دادن eventها را داشته باشند.

Wake-up sensors:

بر خلاف سنسور های non-wake-up، این سنسور ها اطمینان می دهند که داده های آن‌ها مستقل از وضعیت SoC تحویل داده می شود. زمانی که SoC بیدار است، سنسور های wake-up مانند سنسور های non-wake-up رفتار می کنند. هنگامی که SoC خواب است، سنسور های wake-up باید SoC را برای دلیور کردن event ها بیدار کنند. آنها همچنان باید به SoC اجازه دهند تا به حالت تعلیق برود، اما همچنین باید زمانی که event ای باید گزارش شود، آن را بیدار کنند. یعنی سنسور باید SoC را بیدار کند و eventها را قبل از سپری شدن maximum reporting latency یا پر شدن سخت افزار FIFO ارائه دهد.

برای اطمینان از اینکه برنامه‌ها قبل از اینکه SoC به خواب برود، زمان دریافت event را دارند، درایور باید هر بار که رویدادی گزارش می‌شود، یک «timeout wake lock» را به مدت 200 میلی‌ثانیه نگه دارد. یعنی SoC نباید در 200 میلی ثانیه پس از وقفه بیدار شدن دوباره به خواب برود.