

Debugging and Profiling

Lecture 10

Operating Systems Practical

11 January 2017

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

Logging

Debugging

Troubleshooting

Profiling

Bibliography

Keywords

Logging

Debugging

Troubleshooting

Profiling

Bibliography

Keywords

- ▶ Logd daemon
 - ▶ From Android 5.0
 - ▶ Centralized user-mode logger
 - ▶ Uses 4 sockets
- ▶ Liblog library
- ▶ `android.util.Log`
- ▶ `logcat`

- ▶ Logd sockets accessed only through liblog
- ▶ Write log messages:
 1. Log class
 2. Liblog library
 3. /dev/socket/logdw socket
- ▶ Read log messages:
 1. logcat
 2. Liblog library
 3. /dev/socket/logdr socket

- ▶ Priority - severity
 - ▶ Verbose, debug, info, warning, error, assert
- ▶ Tag identifies the component generating the message
 - ▶ Logcat can filter log messages based on the tag
- ▶ Message: actual log text

- ▶ `android.util.Log`
 - ▶ `Log.v(String tag, String msg)` -> verbose
 - ▶ `Log.d(String tag, String msg)` -> debug
 - ▶ `Log.i(String tag, String msg)` -> information
 - ▶ `Log.w(String tag, String msg)` -> warning
 - ▶ `Log.e(String tag, String msg)` -> error
 - ▶ `Log.wtf(String tag, String msg)` -> assert
 - ▶ `Log.println(int priority, String tag, String msg)`
- ▶ Example:
 - ▶ `Log.i("MyActivity", "Get item number " + pos);`
 - ▶ `I/MyActivity(1557): Get item number 1`

- ▶ Exposed through `android/log.h`
- ▶ `#include <android/log.h>`
- ▶ `Android.mk` dynamically link native code to log library
 - ▶ `LOCAL_LDLIBS += -llog`
 - ▶ Before include `$(BUILD_SHARED_LIBRARY)`

- ▶ `__android_log_write`
 - ▶ Generate a simple string message
 - ▶ Params: priority, tag, message

```
__android_log_write(ANDROID_LOG_WARN, "my_native_code",  
"Warning message!");
```

- ▶ `__android_log_print`
 - ▶ Generate formatted string (like printf)
 - ▶ Params: priority, tag, string format, other params

```
__android_log_print(ANDROID_LOG_ERROR, "my_native_code",  
"Errno =%d", errno);
```

- ▶ `__android_log_vprint`
 - ▶ Additional parameters as `va_list`

```
void log_verbose(const char* format, ...){
    va_list args;
    va_start(args, format);
    __android_log_vprint(ANDROID_LOG_VERBOSE, "my_-
native_code", format, args);
    va_end(args);
}
```

- ▶ `__android_log_assert`
 - ▶ Assertion failures
 - ▶ Priority is not specified, always fatal

```
__android_log_assert("0 != errno", "my_native_code", "Big
error!");
```

- ▶ SIGTRAP to process - debugger inspection

- ▶ Display log messages
- ▶ Command line through adb or Android Studio
- ▶ Set log level, search, apply filters
- ▶ Log format:

```
date time PID-TID/package priority/tag: message
```

- ▶ Example:

```
12-10 13:02:50.071 1901-4229/com.google.android.gms V/AuthZen:  
Handling delegate intent.
```

- ▶ Cannot suppress log messages based on priority
- ▶ Preprocessor based solution

```
#define MY_LOG_NOOP (void) 0

#define MY_LOG_PRINT(level,fmt,...) \
    __android_log_print(level, MY_LOG_TAG, "(%s:%u) □%s:□" fmt \
        __FILE__, __LINE__, __PRETTY_FUNCTION__, ##__VA_ARGS__)

#if MY_LOG_LEVEL_WARNING >= MY_LOG_LEVEL
#    define MY_LOG_WARNING(fmt,...) \
        MY_LOG_PRINT(ANDROID_LOG_WARN, fmt, ##__VA_ARGS__)
#else
#    define MY_LOG_WARNING(...) MY_LOG_NOOP
#endif
```

▶ In native code

```
#include "my-log.h"

...

MY_LOG_WARNING("Message!");
```

▶ In Android.mk

```
MY_LOG_TAG := \"my_native_code\"

ifeq ($(APP_OPTIM),release)
    MY_LOG_LEVEL := MY_LOG_LEVEL_ERROR
else
    MY_LOG_LEVEL := MY_LOG_LEVEL_VERBOSE
endif

LOCAL_CFLAGS += -DMY_LOG_TAG=$(MY_LOG_TAG)
LOCAL_CFLAGS += -DMY_LOG_LEVEL=$(MY_LOG_LEVEL)
```

- ▶ STDOUT and STDERR not visible by default
- ▶ Redirect STDOUT and STDERR to logging system

```
adb shell stop
adb shell setprop log.redirect-stdio true
adb shell start
```

- ▶ Display with logcat - tags stdout and stderr
 - ▶ Temporary config -> erased when booting device
- ▶ Permanent config -> modify /data/local.prop on device

Logging

Debugging

Troubleshooting

Profiling

Bibliography

Keywords

- ▶ NDK supports debugging using GNU Debugger (GDB)
- ▶ `ndk-gdb` script
 - ▶ Handles error conditions
 - ▶ Outputs error messages
- ▶ Requirements
 - ▶ Use `ndk-build -> build` system generates files needed for debugging
 - ▶ `android:debuggable` in `AndroidManifest.xml`
 - ▶ Android version 2.2 or higher

- ▶ `ndk-gdb` script sets up the debug session
- ▶ Launches the app using Activity Manager through ADB
 - ▶ Activity Manager sends the request to Zygote
 - ▶ Zygote forks and creates new process
- ▶ `ndk-gdb` starts GDB server and attaches to the app
- ▶ Configures port forwarding to make GDB server accessible from the host machine (debug port)
- ▶ Copies binaries for Zygote and shared libraries to the host
- ▶ Starts GDB client
- ▶ Debug session is active -> You can start debugging app
 - ▶ Commands sent over the debug port

- ▶ Go to project directory
- ▶ `rm -rf bin obj libs`
- ▶ Compile native code using `ndk-build`
- ▶ We need `build.xml` -> `android update project -p`
- ▶ Compile and package the whole project in debug mode `ant debug`
- ▶ Deploy app on device `ant installd`
- ▶ `ndk-gdb --start` to start app and the debugging session
- ▶ When GDB prompt appears run commands

- ▶ `break`: Breakpoint in a location (function name, file name & line number)
- ▶ `clear`: deletes all breakpoints
- ▶ `enable/disable/delete`: operations on a certain breakpoint
- ▶ `next`: go to the next line in source code
- ▶ `continue`: continue execution
- ▶ `backtrace`: display call stack
- ▶ `backtrace full`: call stack with local variables on frames
- ▶ `print`: display variable, expression, memory address, register
- ▶ `display`: continue printing value after each step
- ▶ `info threads`: list running threads
- ▶ `thread`: select a certain thread

- ▶ Integrated in Android Studio
- ▶ Launch Android Debug Monitor -> DDMS button
- ▶ Works with real devices and emulator
- ▶ Debugging Android applications
- ▶ Port-forwarding, screen capture, thread info, heap info, process state, radio state, incoming call, SMS spoofing, location spoofing, etc.

- ▶ When started, DDMS connects to adb
- ▶ VM monitoring service is created between adb and DDMS
- ▶ The service notifies DDMS when a VM is started or terminated
- ▶ Obtains the pid, opens a connection to the VM's debugger through abdb
- ▶ Talks to the VM using a custom wire protocol

Android Device Monitor

Threads

ID	Tid	Status	uTime	sTime	Name
*1	1846	Runnable	53	32	main
*2	1851	Wait	0	0	Signal Catcher
*3	1852	Runnable	0	53	JWPW
*4	1855	Wait	0	0	FinalizerWatchdogDaemon
*5	1853	Wait	0	0	ReferenceQueueDaemon
6	1858	Runnable	1	1	Binder_1
*7	1856	Monitor	2	5	HeapTaskDaemon
*8	1854	Wait	0	5	FinalizerDaemon

Refresh Fri Apr 15 14:44:38 PDT 2016

```

at java.lang.Object.wait(Native Method)
at java.lang.Daemons$FinalizerWatchdogDaemon.waitForObject(Daemons.java:255)
at java.lang.Daemons$FinalizerWatchdogDaemon.run(Daemons.java:227)
at java.lang.Thread.run(Thread.java:818)

```

File Explorer

Name	Size	Date	Time	Permissions	Info
acct		2016-04-15	14:31	drwxr-xr-x	
cache		2016-04-13	20:08	drwxrwx---	
charger		1969-12-31	16:00	lrwxrwxrwx	-> /sbin/...
config		2016-04-15	14:31	dr-xr-xr-x	
d		2016-04-15	14:31	lrwxrwxrwx	-> /sys/k...
data		2016-02-23	17:32	drwxrwxr-x	

LogCat

Saved Filters: All messages (no filters) (3249)

Level	Time	PID	TID	Application	Tag	Text
I	04-15 14:34:15.842	1879	2452	com.google.android.gms.persistent	ClearcutLogg	disconnect managed GoogleApiClient
I	04-15 14:34:16.439	1879	3843	com.google.android.gms.persistent	PhenotypeFla	Experiment Configs successfully retrieved for com.s.playlog.uploader
I	04-15 14:34:16.439	1879	3843	com.google.android.gms.persistent	GoogleURLCon	Using platform SSLCertificateSocketFactory
I	04-15 14:34:16.469	1551	1940	system_process	AccountManag	getTypesVisibleToCaller: isPermitted? true
I	04-15 14:34:16.713	1551	2033	system_process	AccountManag	getTypesVisibleToCaller: isPermitted? true
I	04-15 14:34:16.716	1551	2172	system_process	AccountManag	getTypesVisibleToCaller: isPermitted? true
W	04-15 14:34:16.716	1879	3843	com.google.android.gms.persistent	Uploader	no longer exists, so no auth token
I	04-15 14:34:16.823	1551	2172	system_process	AccountManag	getTypesVisibleToCaller: isPermitted? true
T	04-15 14:34:16.967	1551	1563	system_process		

Source: <http://developer.android.com>

- ▶ View how much heap the process is using
 - ▶ Select process in *Devices* tab
 - ▶ *Update Heap* to obtain heap info
 - ▶ *Cause GC* to invoke Garbage Collection (refresh data)
 - ▶ Select object type to view number of allocated objects

Heap updates will happen after event

ID	Heap Size	Allocated	Free	% Used	# Objects
1	23.273 MB	21.779 MB	1.494 MB	93.58%	49,040

Display:

Type	Count	Total Size	Smallest	Largest	Med
free	1,317	1.483 MB	16 B	1.373 MB	2-
data object	30,545	1.030 MB	16 B	1.047 KB	3-
class object	3,165	903.914 KB	168 B	40.992 KB	16-
1-byte array (byte[], boolean[])	525	18.878 MB	24 B	5.695 MB	67-

Source: <http://developer.android.com>

- ▶ Track memory allocation
 - ▶ *Start Tracking* in the *Allocation Tracker* tab
 - ▶ *Get Allocations* to obtain list of allocated objects
 - ▶ Finally *Stop Tracking*
 - ▶ Detailed info about the method and line that allocated a certain object
- ▶ Examine thread info
 - ▶ *Update Threads* to obtain thread info for the selected process

Android Debug Monitor

DDMS Hierarchy Vi...

Thre... Heap Alloc... Netw... File... Emu... Syst...

Stop Tracking Get Allocations Filter: Inc. trace

Alloc Order	Allocation Size	Allocated Class	Thread Id	Allocated In
149	5740832	byte[]	1	android.graphics.BitmapFactory
152	1435220	byte[]	1	android.graphics.BitmapFactory
83	704	com.android.int...	1	java.lang.reflect.Constructor
194	456	android.widget.L...	1	java.lang.reflect.Constructor
159	184	int[]	1	java.lang.Throwable
122	180	char[]	1	java.lang.AbstractStringBuilder
...

```

at android.graphics.BitmapFactory.nativeDecodeAsset(Native Method)
at android.graphics.BitmapFactory.decodeStream(BitmapFactory.java:503)
at android.graphics.BitmapFactory.decodeResourceStream(BitmapFactory.java:356)
at android.graphics.drawable.Drawable.createFromResourceStream(Drawable.java:800)
at android.content.res.Resources.loadDrawable(Resources.java:2105)
at android.content.res.Resources.getDrawable(Resources.java:695)
at android.widget.ImageView.resolveUri(ImageView.java:636)
at android.widget.ImageView.setImageResource(ImageView.java:365)
at com.example.android.navigationdrawerexample.MainActivity$PlanetFragment.onCreate...

```

LogCat Console

Source: <http://developer.android.com>

Logging

Debugging

Troubleshooting

Profiling

Bibliography

Keywords

- ▶ Use troubleshooting tools and techniques to identify the cause of a problem
- ▶ Observe the stack trace when an app crashes with logcat
 - ▶ Lines starting with # represent stack calls
 - ▶ Line #00 is the crash point
 - ▶ After #00 the address is specified (pc)

- ▶ `ndk-stack`
 - ▶ To add file names and line numbers to the stack trace
 - ▶ `adb logcat | ndk-stack -sym obj/local/armeabi`
 - ▶ Run command in the project directory
 - ▶ Obtain exact file name and line number where it crashed

- ▶ Extended series of checks before calling JNI functions
- ▶ Enable CheckJNI on a device

- ▶ Rooted device

```
adb shell stop
adb shell setprop dalvik.vm.checkjni true
adb shell start
```

- ▶ Logcat: D AndroidRuntime: CheckJNI is ON

- ▶ Regular device

```
adb shell setprop debug.checkjni 1
```

- ▶ Logcat: D Late-enabling CheckJNI

- ▶ Error detected by CheckJNI

```
W JNI WARNING: method declared to return
'Ljava/lang/String;' returned '[B'
W failed in LJniTest;.exampleJniBug
```

- ▶ Troubleshoot memory issues
- ▶ Enable libc debug mode

```
adb shell setprop libc.debug.malloc 1
adb shell stop
adb shell start
```

- ▶ Libc debug mode values
 - ▶ 1 - detects memory leaks
 - ▶ 5 - detects overruns by filling allocated memory
 - ▶ 10 - detects overruns by filling memory and adding sentinel

```
... testapp using MALLOC_DEBUG = 10 (sentinels, fill)
... *** FREE CHECK buffer 0xa5218, size=1024, corrupted 1
bytes after allocation
```

- ▶ Advanced memory analysis
- ▶ Open-source tool for memory debugging, memory leaks detection and profiling
- ▶ Support for Android
- ▶ Build from sources
 - ▶ Binaries and components in Inst directory
 - ▶ `adb push Inst /data/local/`
 - ▶ Give execution permissions

- ▶ Helper script

```
#!/system/bin/sh
export TMPDIR=/sdcard
exec /data/local/Inst/bin/valgrind --error-limit=no $*
```

- ▶ Push in `/data/local/Inst/bin` and set execution permissions

- ▶ To run app under Valgrind, inject the script into the startup sequence

```
adb shell setprop wrap.com.example.testapp "logwrapper  
/data/local/Inst/bin/valgrind_wrapper.sh"
```

- ▶ Property wrap.packageName
- ▶ Execute app
- ▶ Logcat displays Valgrind output

- ▶ Intercepts system calls and signals
- ▶ System call name, arguments and return value
- ▶ Useful for analyzing closed-source applications
- ▶ Included in Android emulator
- ▶ Run the application and obtain pid

```
adb shell ps | grep com.example.testapp
```

- ▶ Attach strace to running app

```
adb shell strace -v -p <PID>
```

- ▶ Tombstone - generated when a process crashes
- ▶ `/data/tombstones/tombstone_*`
- ▶ A file containing information about the crashed process
 - ▶ Build fingerprint
 - ▶ Crashed process, PID, TIDs
 - ▶ Signal and fault address
 - ▶ CPU registers
 - ▶ Call stack
 - ▶ Stack content of each call
- ▶ Use with `ndk-stack` and `addr2line` to obtain the file and line where the process has crashed

```
Build fingerprint: 'Android/aosp_hammerhead/hammerhead:4.4.2/
KOT49H/eng.upb.20140407.130154:userdebug/test-keys'
Revision: '11'
pid: 27836, tid: 27836, name: test.nativeapp4  >>>
com.test.nativeapp4 <<<
signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 00000000
    r0 00000000   r1 00000000   r2 6f8b70e4   r3 6f8b8328
[...]
backtrace:
    #00  pc 00008bbc   /system/lib/libandroid.so (AAsset_close+3)
    #01  pc 00000d47   /data/app-lib/com.test.nativeapp4-2/
libNativeApp4.so (displayAsset(ANativeActivity*)+18)
    #02  pc 00000db1   /data/app-lib/com.test.nativeapp4-2/
libNativeApp4.so (ANativeActivity_onCreate+96)
[...]
stack:
    bea91430   00000000
    bea91434  401a7315   /system/lib/libutils.so
(android::SharedBuffer::release(unsigned int) const+28)
    bea91438   bea91450   [stack]
    bea9143c   00000000
    bea91440   00000000
    bea91444  402ad59b   /system/lib/libandroidfw.so
    bea91448  6f8b70e0   [anon:libc_malloc]
```

Logging

Debugging

Troubleshooting

Profiling

Bibliography

Keywords

- ▶ Unix-based profiling tool
- ▶ Compute absolute execution time spent in each function
 - ▶ Instrumentation with `gcc` when using `-pg` at compile time
 - ▶ Sampling data stored at run-time in `gmon.out`
 - ▶ `gprof` uses `gmon.out` to produce profiling reports
- ▶ Android NDK includes `gprof` tool
 - ▶ Android NDK toolchain lacks the implementation of `__gnu_mcount_nc` used for timing
- ▶ Open-source project Android NDK Profiler

- ▶ Install module
 - ▶ Download zip, extract in \$NDK_HOME/sources, rename directory to android-ndk-profiler
- ▶ Enable profiler
 - ▶ Update Android.mk to statically link profiling library
 - ▶ Include prof.h in the native code

```
#ifdef MY_ANDROID_NDK_PROFILER_ENABLED
#include <prof.h>
#endif
```

- ▶ Start collecting profiling data

```
#ifdef MY_ANDROID_NDK_PROFILER_ENABLED
    monstartup("libModule.so");
#endif
```

- ▶ Stop collecting data

```
#ifdef MY_ANDROID_NDK_PROFILER_ENABLED
    moncleanup();
#endif
```

- ▶ The collected data is stored in /sdcard/gmon.out
- ▶ App needs permission to write on the SD card

```
<uses-permission android:name="android.permission.WRITE_-  
EXTERNAL_STORAGE" />
```

- ▶ Pull gmon.out from the SD card
- ▶ Run gprof

```
$NDK_HOME/toolchains/arm-linux-androideabi-4.4.3/prebuilt/  
linux-x86/bin/arm-linux-androideabi-gprof  
obj/local/armeabi-v7a/libModule.so gmon.out
```

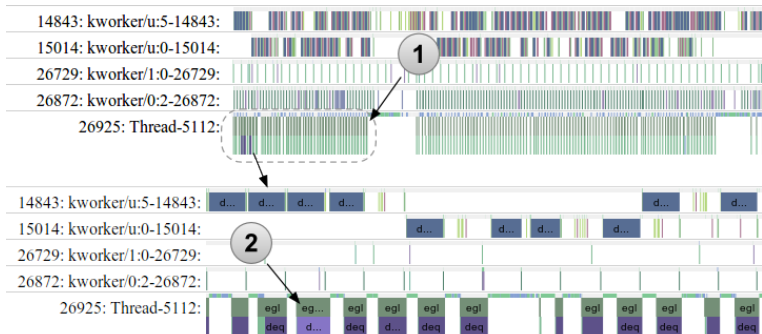
- ▶ Gprof analyses data and generates a report
- ▶ Two sections: flat profile and call graph
- ▶ Duration of each function

- ▶ Gather code execution data -> Identify execution problems and improve performance
- ▶ Show all processes on a common timeline
- ▶ Execution times, CPU frequency, CPU load, disk activity, threads
- ▶ Android 4.1 or higher, root access, developer debugging enabled
- ▶ GUI and CLI

```
$ cd android-sdk/platform-tools/systrace
$ python systrace.py --time=10 -o mynewtrace.html sched gfx
view wm
```

- ▶ Open trace in a web browser

- ▶ Thicker bars (tasks that take longer than others) can indicate a performance problem



Source: <http://developer.android.com>

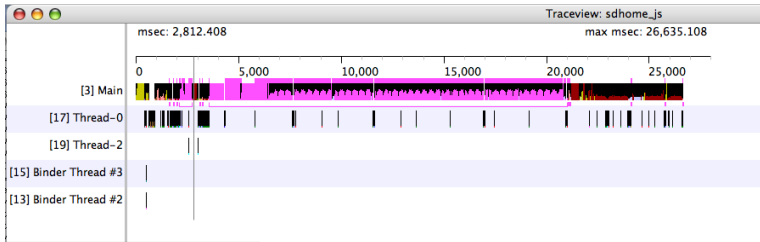
- ▶ From Android 4.3 use Trace class to add instrumentation to the application code
- ▶ Trace calls can be nested
- ▶ Traces must begin and end in the same thread

```
Trace.beginSection("Start_trace");  
try {  
    // executing tasks  
} finally {  
    Trace.endSection(); // end trace  
}
```

- ▶ Graphical viewer for execution logs (generated with Debug class)
- ▶ Timeline panel - displays each thread and method started/stopped
- ▶ Profile panel - summary of all time spent in a method
- ▶ Two methods to generate trace logs:
 - ▶ Use methods of the Debug class to start and stop tracing
 - ▶ Use method profiling feature of DDMS (no precise log timing)

```
Debug.startMethodTracing("data"); // start tracing to  
                                   // "/sdcard/data.trace"  
  
// execute tasks  
Debug.stopMethodTracing(); // stop tracing
```

- ▶ Displays the execution of each thread in a separate row
- ▶ Each method - a different color
- ▶ Thin line below - extent of all calls to the selected method



Source: <http://developer.android.com>

- ▶ All time spent in a method (inclusive and exclusive times)
- ▶ Exclusive time = time spent in a method
- ▶ Inclusive time = time spent in a method + time spent in any called functions
- ▶ Last column - number of calls to this method + number of recursive calls

Name	Incl %	Inclusive	Excl %	Exclusive	Calls+Rec
4 android/webkit/LoadListener.nativeFinished (JV)	66.6%	17734.382	53.2%	14161.950	14+0
3 android/webkit/LoadListener.tearDown (JV)	100.0%	17734.382			14/14
6 android/view/View.invalidate (IIII)V	19.8%	3516.410			2413/2853
57 android/webkit/BrowserFrame.startLoadingResource (LJjava	0.3%	44.636			3/15
53 java/util/HashMap.put (Ljava/lang/Object;Ljava/lang/Objec	0.0%	6.223			6/326
20 android/webkit/JWebCoreJavaBridge.setSharedTimer (JV)	0.0%	2.593			2/730
378 android/view/ViewGroup.requestLayout (V)	0.0%	1.139			2/54
315 java/util/HashMap.<init> (II)V	0.0%	0.879			3/41
629 android/webkit/BrowserFrame.loadCompleted (JV)	0.0%	0.285			1/1
598 android/webkit/WebView.didFirstLayout (V)	0.0%	0.231			1/2
703 android/webkit/BrowserFrame.windowObjectCleared (I)V	0.0%	0.036			1/2
5 android/webkit/JWebCoreJavaBridge\$TimerHandler.handleMessa	16.3%	4342.697	0.5%	132.018	730+0
6 android/view/View.invalidate (IIII)V	15.6%	4161.341	1.2%	319.164	2853+0
7 android/webkit/JWebCoreJavaBridge.access\$300 (Landroid/webk	15.1%	4025.658	0.1%	26.727	729+0

Source: <http://developer.android.com>

Logging

Debugging

Troubleshooting

Profiling

Bibliography

Keywords

- ▶ Onur Cinar, Pro Android C++ with the NDK, Chapter 5, 14
- ▶ Sylvain Ratabouil, Android NDK, Beginner's Guide, Chapter 11
- ▶ <https://code.google.com/p/android-ndk-profiler/wiki/Usage>
- ▶ <http://developer.android.com/tools/debugging/ddms.html>
- ▶ <http://bytesthink.com/blog/?p=133>
- ▶ <http://developer.android.com/tools/debugging/systrace.html>

Logging

Debugging

Troubleshooting

Profiling

Bibliography

Keywords

- ▶ Logger
- ▶ Logging API
- ▶ Log control
- ▶ GDB
- ▶ DDMS
- ▶ Stack trace
- ▶ Tombstones
- ▶ CheckJNI
- ▶ Libc Debug Mode
- ▶ Valgrind
- ▶ Strace
- ▶ Gprof
- ▶ Android NDK Profiler
- ▶ Systrace
- ▶ Traceview