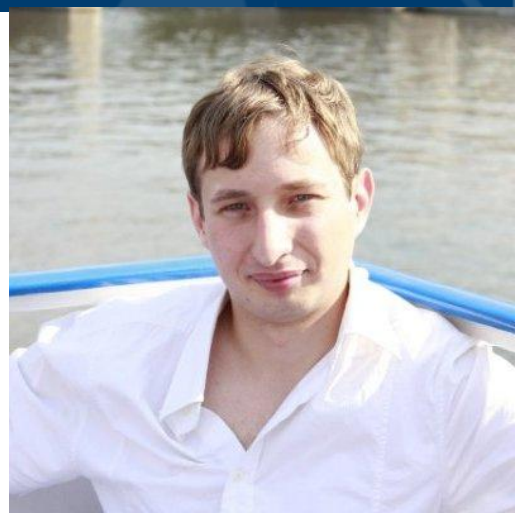
Тестирование native-приложений на Android

ручное тестирование и автоматизация тестирования

2013
CEE-SEC(R)
Software Engineering Conference in Russia

Elite Software R&D Services
Since 1990

Знакомство

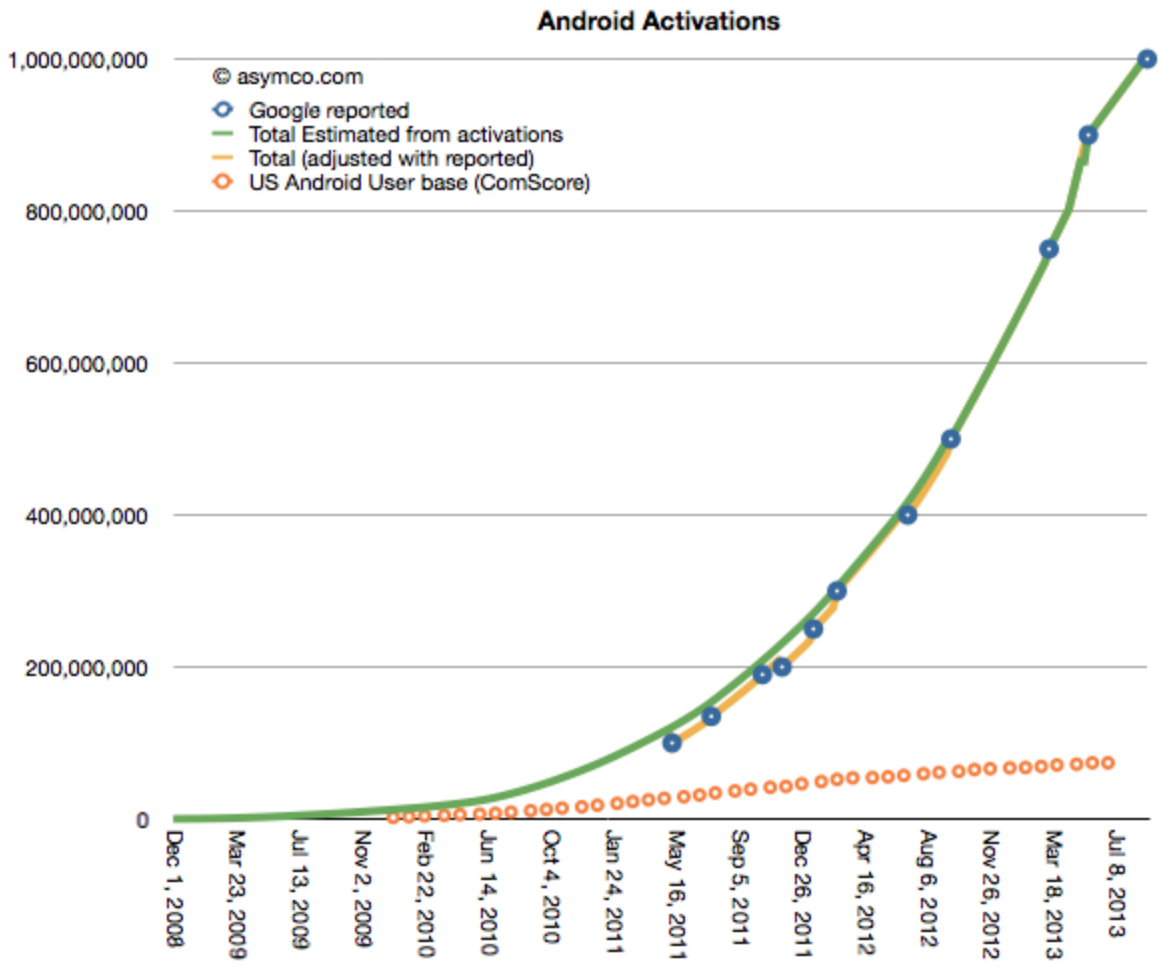


- Илья Верещагин
  - -Test Manager @ Auriga
  - -Test Team Leader @ Intel
  - Окончил факультет кибернетики МИРЭА, специальность - инженер-математик. С 2012 года занимается тестированием операционных систем Android и Tizen, разработкой и тестированием Android NDK x86.

Содержание

- Вступление
- Обзор Android NDK
- Сборка и запуск приложений
- Отладка приложений
- Тестирование производительности
- Автоматизация тестирования

Android и тенденции



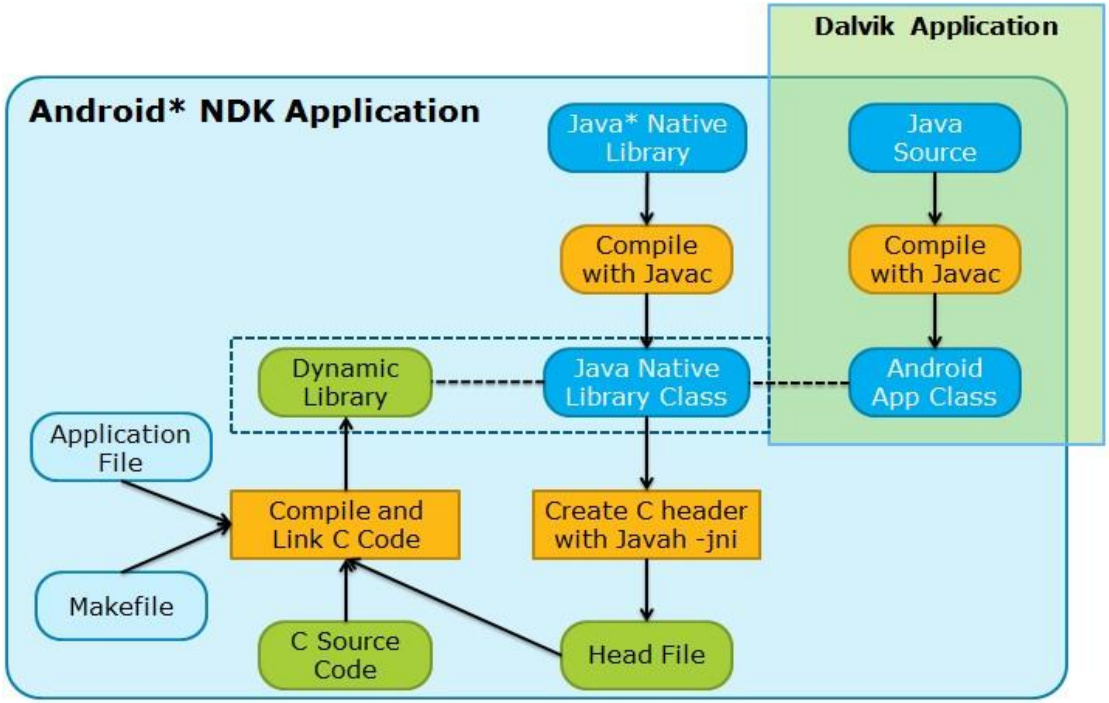
Устройства под управлением Android

- Количество и спектр устройств под управлением Android продолжает расти

Поддерживаемые архитектуры

Оптимизация приложений\образов

- Одним из самых эффективных решений для оптимизации и ускорения работы приложений, системы является оптимизация на родном, native-уровне.



Android приложения с native-кодом

- Использование для приложений оптимизированных, часто аппаратно-зависимых библиотек и кода, а в глобальном смысле – целого ряда библиотек, поставляющихся для каждой из целевых платформ
- Несмотря на то, что одним из постулатов Android является независимость от железа, и при поставке приложений в реальности используется fat binary (apk, включающий в себя native-код для всех возможных конфигураций оборудования под которым будет работать приложение).

Плюсы и минусы native-кода

- **Преимущества**:
  - Высокая производительность
  - Прямое использование CPU/HW особенностей
  - Возможность переиспользования существующего Linux кода

- **Недостатки**:
  - Индивидуальная настройка под CPU/HW
  - Недостаточная поддержка системных библиотек

Платформо-зависимый код

- Несмотря на то, что в идеале NDK компилятор должен создавать функционально развнозначный код для различных конфигураций, это, к сожалению, не всегда так и требует проверки, особенно когда дело касается оптимизации (Neon и SSE например) и использования сторонних библиотек.

ARM v5	ARM v7a	Atom
32-bit	32-bit	32-bit
little-endian	little-endian	little-endian
Soft FP	Hardware FP	Hardware FP
64-bit vars aligned	64-bit vars aligned	64-bit vars packed
None	NEON	SSE

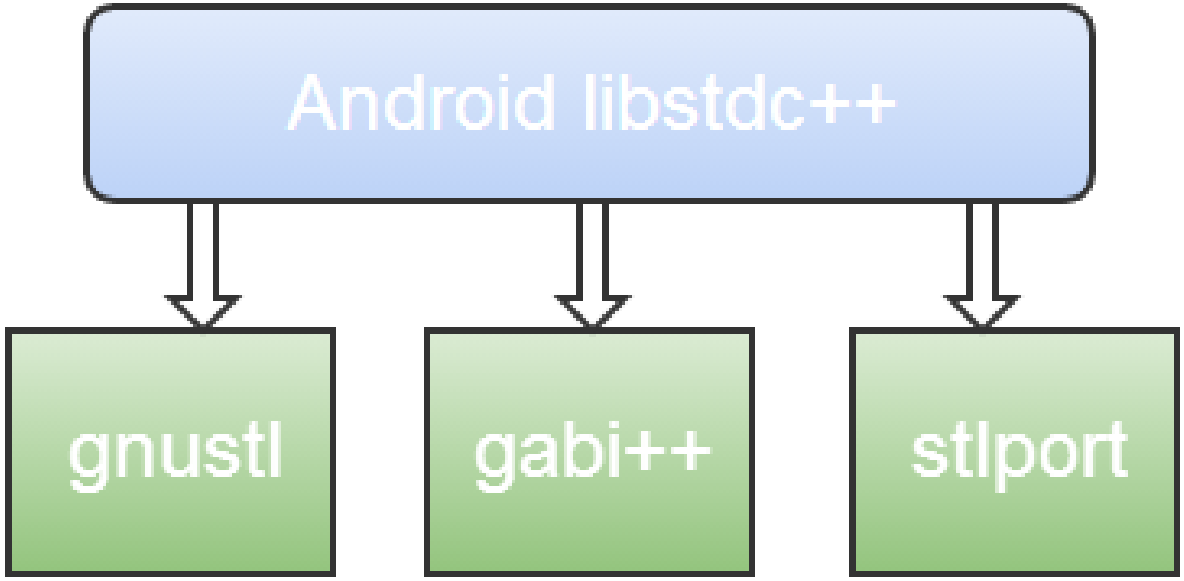
Различие native-сборок

Native-сборка Android может различаться в зависимости от платформы: software, hardware, задач (оптимизации под задачи)

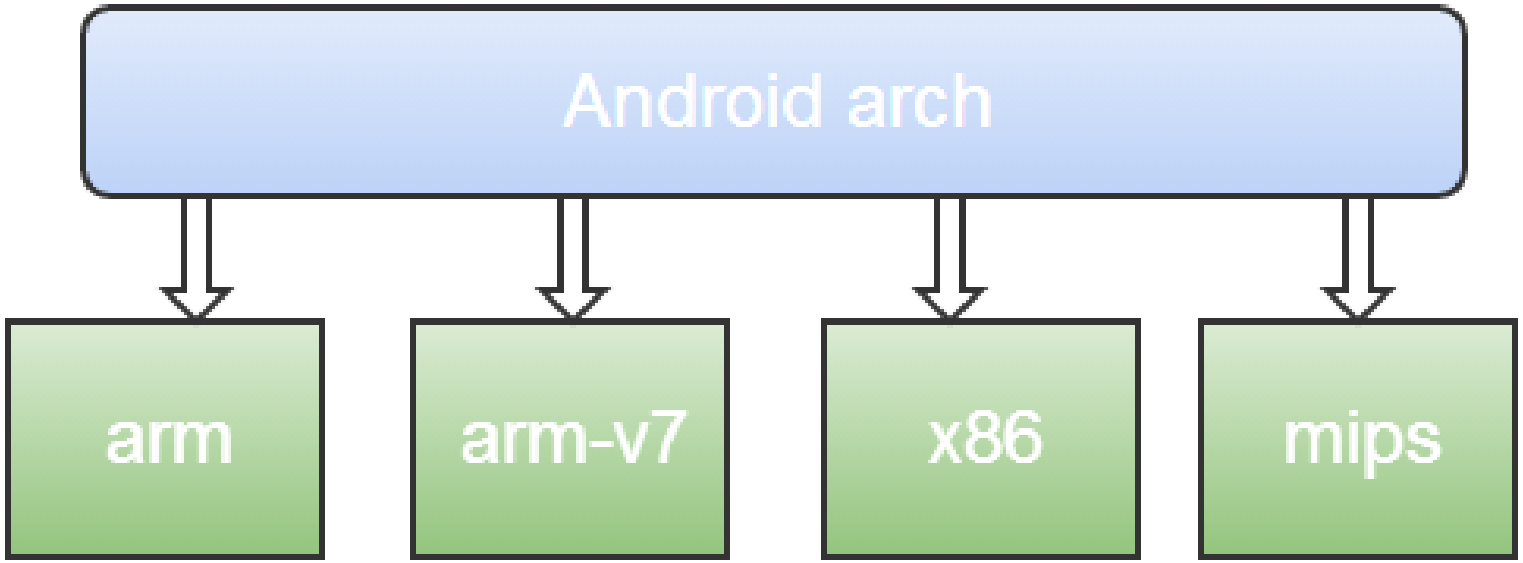
Различие по версии используемого API

Platform Version	API Level	VERSION_CODE	Notes
Android 4.3	18	JELLY_BEAN_MR2	Platform Highlights
Android 4.2, 4.2.2	17	JELLY_BEAN_MR1	Platform Highlights
Android 4.1, 4.1.1	16	JELLY_BEAN	Platform Highlights
Android 4.0.3, 4.0.4	15	ICE_CREAM_SANDWICH_MR1	Platform Highlights
Android 4.0, 4.0.1, 4.0.2	14	ICE_CREAM_SANDWICH	
Android 3.2	13	HONEYCOMB_MR2	
Android 3.1.x	12	HONEYCOMB_MR1	Platform Highlights
Android 3.0.x	11	HONEYCOMB	Platform Highlights
Android 2.3.4 Android 2.3.3	10	GINGERBREAD_MR1	Platform Highlights
Android 2.3.2 Android 2.3.1 Android 2.3	9	GINGERBREAD	
Android 2.2.x	8	FROYO	Platform Highlights
Android 2.1.x	7	ECLAIR_MR1	Platform Highlights
Android 2.0.1	6	ECLAIR_0_1	
Android 2.0	5	ECLAIR	
Android 1.6	4	DONUT	Platform Highlights
Android 1.5	3	CUPCAKE	Platform Highlights
Android 1.1	2	BASE_1_1	
Android 1.0	1	BASE	

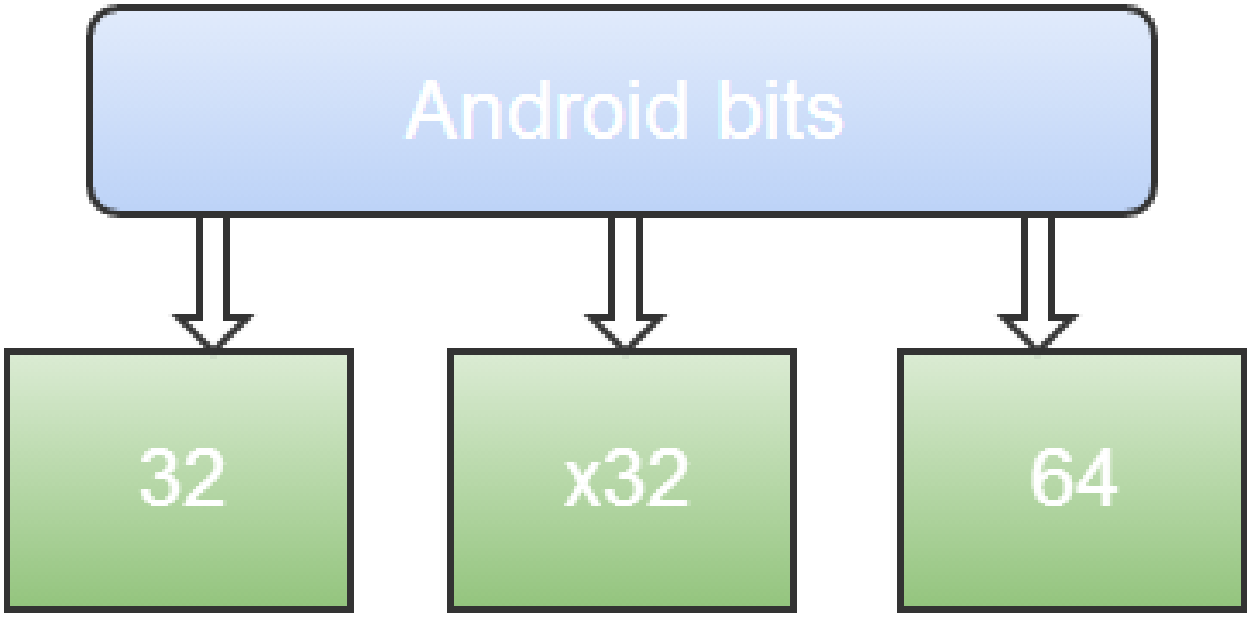
Различие по порту libstdc++:



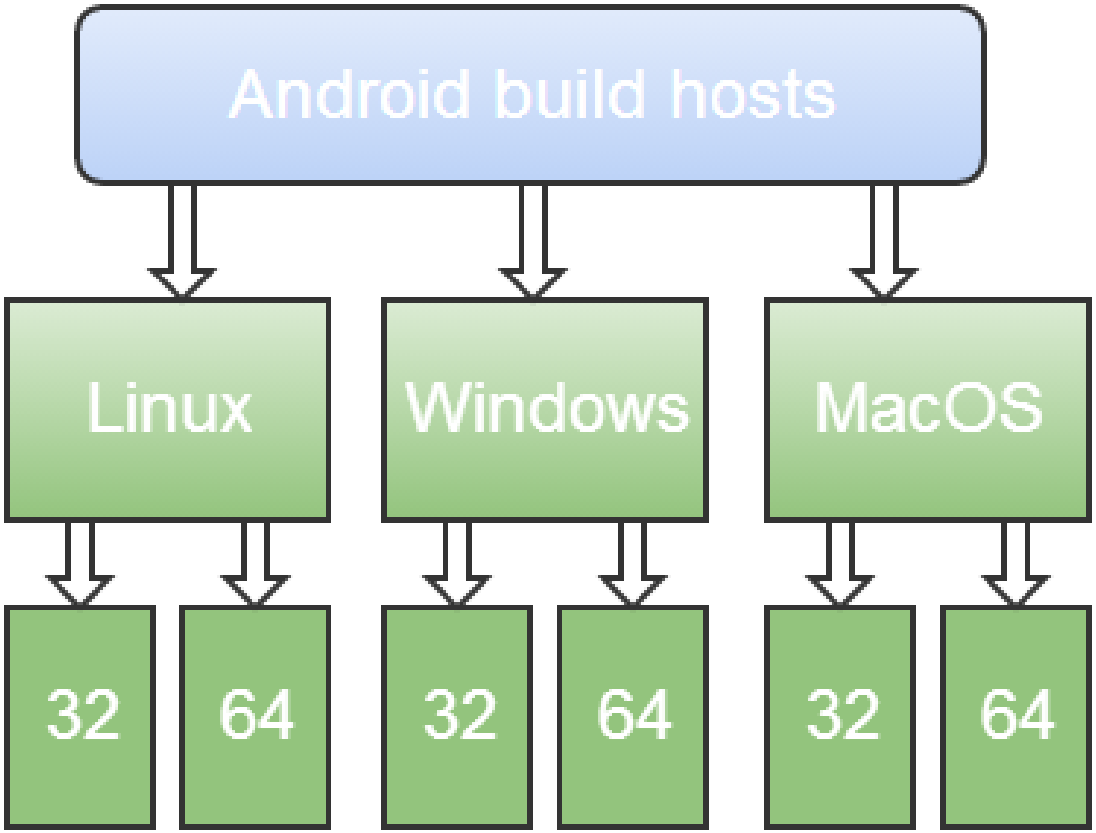
Различие по по архитектуре устройства (target: –march)



Различие по по разрядности (32, 64 или x32 target)



Различие по по билд-хосту и его разрядности



Прочие оптимизации

- Оптимизация по инструкциям:
  - -sse, sse3, ssse3, neon и пр.
  - -neon, core-avx2, atom, slm и пр.
  - -Различие в bionic
  - -Оптимизация по размеру или производительности (-Os, -O0, -flto и пр.,)

Задача тестирования

- Все это декартово произведение вариантов полученного кода (исходного и бинарного) является исходными данными для тестирования – системы и приложений.
- Типичная задача тестировщика – анализ стабильности и производительности для каждой из конфигураций

Создание и запуск приложения (gcc, g++)

igveresx@msticlxl101:/users/igveresx

```
bash-4.2$
```

Сбор данных, профилей и покрытия

igveresx@msticlxl101:/users/igveresx

```
bash-4.2$ echo $?
0
bash-4.2$ #работает, теперь мы можем запустить и протестировать приложение и на хосте
bash-4.2$
bash-4.2$ #теперь давайте сделаем тоже самое для g++
bash-4.2$ cat ./hello_world.C
#include <iostream>

int main(void)
{
    std::cout << "Hello, World!\n";
    return 0;
}
bash-4.2$ #если мы будем использовать STL, то
bash-4.2$ #нам дополнительно понадобится указать пути к библиотекам и заголовочным файлам
bash-4.2$ #а так же сообщить компоновщику, какую библиотеку надо добавить
bash-4.2$ #обратите внимание, что инструкция -l%STL_LIB% должна быть в конце коммандной строки!
bash-4.2$ #динамический вариант имеет суффикс _shared, а статический _static соотвественно
bash-4.2$ #теперь добавим к вызову g++ -L, -I и одну из библиотек - gnustl/stlport/gabi
bash-4.2$ /home/NDK_current/toolchains/x86-4.7/prebuilt/linux-x86/bin/i686-linux-android-g++ --sysroot=/home/NDK_current/platforms/android-9/arch-x86 -I/home/NDK_curren
t/sources/cxx-stl/gnu-libstdc++/4.7/include -I/home/NDK_current/sources/cxx-stl/gnu-libstdc++/4.7/libs/x86/include -L/home/NDK_current/sources/cxx-stl/gnu-libstdc++/4.7
/libs/x86 ./hello_world.C -o ./hello_world.exe  -lgnustl_shared
bash-4.2$ echo $?
0
bash-4.2$ #собралось, но не всегда это возможно из-за громоздкости связанных библиотек и/или наличия нужных статичных/динамических версий
bash-4.2$ adb push ./hello_world.exe /data/local/
143 KB/s (6424 bytes in 0.043s)
bash-4.2$ #что-то отправилось, теперь запустим
bash-4.2$ adb shell "/data/local/hello_world.exe && echo $?"
soinfo_link_image(linker.cpp:1635): could not load library "libgnustl_shared.so" needed by "/data/local/hello_world.exe"; caused by load_library(linker.cpp:745): librar
y "libgnustl_shared.so" not foundCANNOT LINK EXECUTABLE
bash-4.2$ #не работает, не хватает библиотеки
bash-4.2$ #libgnustl_shared.so не входит в поставку Android, поэтому её надо скопировать на устройство
bash-4.2$ adb push /home/NDK_current/sources/cxx-stl/gnu-libstdc++/4.7/libs/x86/libgnustl_shared.so /data/local/libgnustl_shared.so
4590 KB/s (1325880 bytes in 0.282s)
bash-4.2$ #переписали, запустим ещё раз, явным образом указав LD_LIBRARY_PATH
bash-4.2$ adb shell "export LD_LIBRARY_PATH=/data/local/:$LD_LIBRARY_PATH && /data/local/hello_world.exe && echo $\?"
Hello, World!
0
bash-4.2$ #либо перенесём библиотеку в путь по умолчанию - /system/lib
bash-4.2$ adb shell "mount -o remount,rw -t ext4 /dev/block/mmcblk0p8 /system && cp /data/local/libgnustl_shared.so /system/lib/libgnustl_shared.so && rm /data/local/li
bgnustl_shared.so && mount -o remount,ro -t ext4 /dev/block/mmcblk0p8 /system"
bash-4.2$ #запустим
```

EN 17:05 13.10.2013

Отладка

igveresx@msticlxl101:/users/igveresx

```
y "libgnustl_shared.so" not foundCANNOT LINK EXECUTABLE
bash-4.2$ #не работает, не хватает библиотеки
bash-4.2$ #libgnustl_shared.so не входит в поставку Android, поэтому её надо скопировать на устройство
bash-4.2$ adb push /home/NDK_current/sources/cxx-stl/gnu-libstdc++/4.7/libs/x86/libgnustl_shared.so /data/local/libgnustl_shared.so
4590 KB/s (1325880 bytes in 0.282s)
bash-4.2$ #переписали, запустим ещё раз, явным образом указав LD_LIBRARY_PATH
bash-4.2$ adb shell "export LD_LIBRARY_PATH=/data/local/:$LD_LIBRARY_PATH && /data/local/hello_world.exe && echo $\?"
Hello, World!
0
bash-4.2$ #либо перенесём библиотеку в путь по умолчанию - /system/lib
bash-4.2$ adb shell "mount -o remount,rw -t ext4 /dev/block/mmcblk0p8 /system && cp /data/local/libgnustl_shared.so /system/lib/libgnustl_shared.so && rm /data/local/li
bgnustl_shared.so && mount -o remount,ro -t ext4 /dev/block/mmcblk0p8 /system"
bash-4.2$ #запустим
bash-4.2$ adb shell "/data/local/hello_world.exe && echo $\?"
Hello, World!
0
bash-4.2$
bash-4.2$ #некоторые файлы, вполне возможно, что-то порождают, и это мы так же можем вытащить с устройства
bash-4.2$ #скомпилируем, например, файл порождающий coverage информацию
bash-4.2$ #обратите внимание, что компилируете вы на хосте, и надо явным образом будет указать директорию профилирования
bash-4.2$ #это можно сделать во время компиляции, добавив флаг -fprofile-dir=%android_exec_dir%
bash-4.2$ #либо перед исполнением, на устройстве, выставив GCOV_PREFIX и GCOV_PREFIX_STRIP
bash-4.2$ #так же обратите внимание, что манипуляции с env действительны лишь на время сессии adb (открытой shell или команды adb shell %command%
bash-4.2$ /home/NDK_current/toolchains/x86-4.7/prebuilt/linux-x86/bin/i686-linux-android-gcc --sysroot=/home/NDK_current/platforms/android-18/arch-x86 ./gcov-1.c   -fpr
ofile-arcs -fprofile-dir=. -ftest-coverage   -lm   -o ./gcov-1.exe
bash-4.2$ adb push ./gcov-1.exe /data/local/gcov-1.exe
910 KB/s (38464 bytes in 0.041s)
bash-4.2$ #adb shell "export GCOV_PREFIX=/data/local && export GCOV_PREFIX_STRIP=13 && /data/local/gcov-1.exe && echo $\?"
bash-4.2$ unset GCOV_PREFIX && unset GCOV_PREFIX_STRIP && cd /data/local && ./gcov-1.exe && echo $\?"
0
bash-4.2$ adb shell ls /data/local/gcov-1.gcda
/data/local/gcov-1.gcda
bash-4.2$ #вытащим профиль
bash-4.2$ adb pull /data/local/gcov-1.gcda .
1 KB/s (148 bytes in 0.080s)
bash-4.2$ ls ./gcov-1.gcda
./gcov-1.gcda
bash-4.2$ #в состав NDK входит gcov, который достаточно вызвать, как обычно
bash-4.2$ /home/NDK_current/toolchains/x86-4.7/prebuilt/linux-x86/bin/i686-linux-android-gcov gcov-1.gcda
File 'gcov-1.c'
Lines executed:100.00% of 6
Creating 'gcov-1.c.gcov'
```

EN 17:06 13.10.2013

Автоматизация тестирования

DejaGnu

- Фреймворк Dejagnu
- Борд androideabi
- Использование раздельной компиляции и запуска (через debug bridge)

Запуск testsuite

- export ADB_SERIAL=$ANDROID_SERIAL
- runtest --target_board=androideabi

Local config file (site.exp)

```
set rootme "."
set tmpdir "."
set srcdir "/path/to/gcc_%version%_release/gcc/testsuite"
set CFLAGS ""
set CXXFLAGS ""
set GDB "/path/to/GDB_UNDER_TEST"
set GCOV_UNDER_TEST "/path/to/GCOV_UNDER_TEST"
set GCC_UNDER_TEST "/path/to/GCC_UNDER_TEST"
set GXX_UNDER_TEST "/path/to/GXX_UNDER_TEST"
set GFORTRAN_UNDER_TEST "no"
set OBJC_UNDER_TEST "no"

set libiconv ""
set HOSTCC "gcc"
set HOSTCFLAGS ""
set TESTING_IN_BUILD_TREE 1
set GMPINC ""
set ENABLE_LTO 1
set HAVE_LIBSTDCXX_V3 1
set host_triplet i686-pc-linux-gnu
set build_triplet i686-pc-linux-gnu
set target_triplet i686-pc-linux-android-gnu
set target_alias i686-pc-linux-android
set android_tmp_dir "/temporary/folder/on/device/with/executable/permissions"
set bridge_tmp_dir "/temporary/folder/on/device/with/executable/permissions"
append boards_dir "/path/to/share/dejagnu/baseboards"
```

Запуск make check

- **export ADB_SERIAL=$ANDROID_SERIAL**
- **make -j $parallel check DEJAGNU=/path/to/site.exp RUNTESTFLAGS="--target_board=androideabi"**

Не забудьте!

- Указать
  - -sysroot
  - -пути к библиотекам
  - -флаги и соответствующие подключаемые библиотеки
- Использовать обёртку

wrapper-gcc

```
#!/bin/bash

/path/to/NDK_folder/toolchains/$arch-
%compiler_version%/prebuilt/linux-x86/bin/%arch_prefix%-
linux-android-gcc --
sysroot=/path/to/NDK_folder/platforms/android-
${device_platform}/arch-$arch  "$@"
```

wrapper-g++

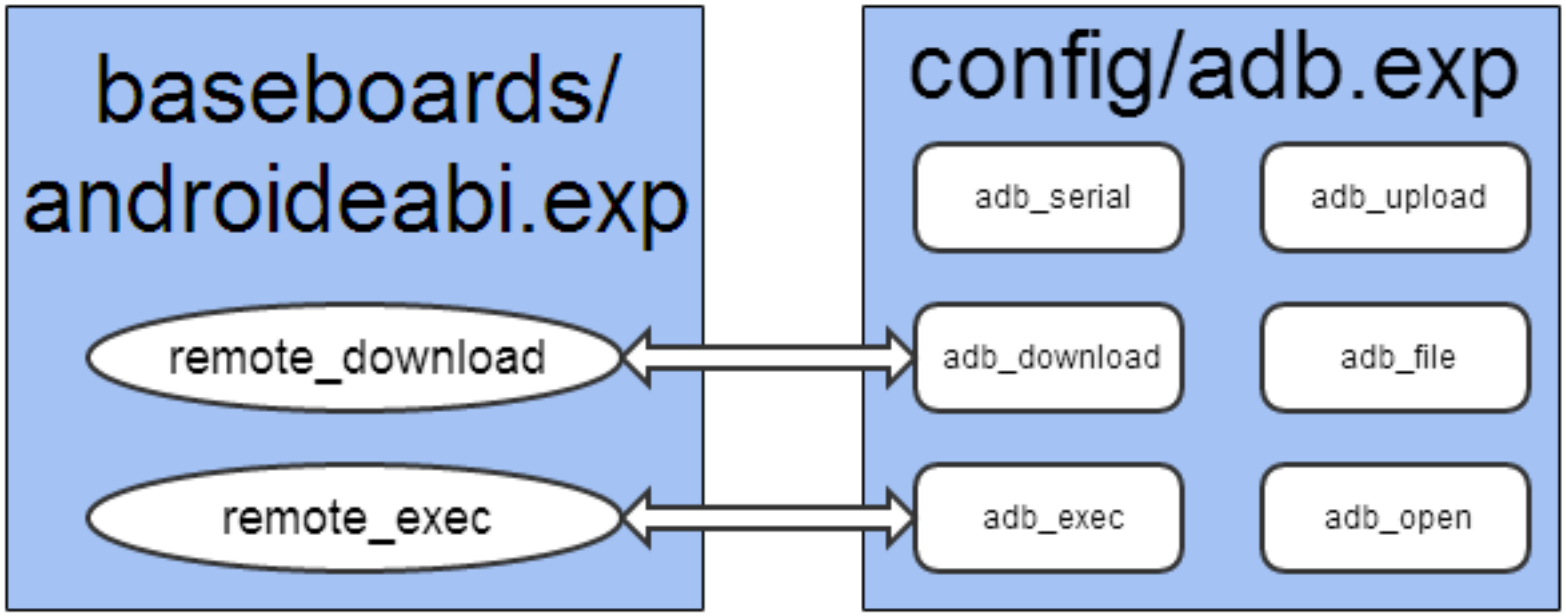
```
#!/bin/bash

echo $@ | grep " \-nostdlib" 1>/dev/null 2>/dev/null
if [ $? != 0 ]; then
    echo $@ | grep " \-static" 1>/dev/null 2>/dev/null
    if [ $? != 0 ]; then
        /path/to/NDK_folder/toolchains/$arch-%compiler_version%/prebuilt/linux-x86/bin/%arch_prefix%-linux-android-g++ --sysroot=/path/to/NDK_folder/platforms/android-${device_platform}/arch-$arch -I/path/to/NDK_folder/sources/cxx-stl/%stl lib folder%/%compiler version%/include -I/path/to/NDK_folder/sources/cxx-stl/%stl lib folder%/%compiler version%/libs/$arch/include -L/path/to/NDK_folder/sources/cxx-stl/%stl lib folder%/%compiler version%/libs/$arch  "$@" -lgnustl_shared
    else
        /path/to/NDK_folder/toolchains/$arch-%compiler_version%/prebuilt/linux-x86/bin/%arch_prefix%-linux-android-g++ --sysroot=/path/to/NDK_folder/platforms/android-${device_platform}/arch-$arch -I/path/to/NDK_folder/sources/cxx-stl/%stl lib folder%/%compiler version%/include -I/path/to/NDK_folder/sources/cxx-stl/%stl lib folder%/%compiler version%/libs/$arch/include -L/path/to/NDK_folder/sources/cxx-stl/%stl lib folder%/%compiler version%/libs/$arch  "$@" -lgnustl_static
    fi
else
    /path/to/NDK_folder/toolchains/$arch-%compiler_version%/prebuilt/linux-x86/bin/%arch_prefix%-linux-android-g++ --sysroot=/path/to/NDK_folder/platforms/android-${device_platform}/arch-$arch -I/path/to/NDK_folder/sources/cxx-stl/%stl lib folder%/%compiler version%/include -I/path/to/NDK_folder/sources/cxx-stl/%stl lib folder%/%compiler version%/libs/$arch/include -L/path/to/NDK_folder/sources/cxx-stl/%stl lib folder%/%compiler version%/libs/$arch  "$@"
fi
```

Получение данных из Android

- Функциональность со стороны testsuite
- Нужно использовать вызовы **remote_upload (adb_upload %target_board% %source% %dest%)**.

Структура dejagnu



Внимание!

- Убедитесь:
  - в значении GCC_EXEC_PREFIX, по умолчанию она unset
  - в директории запуска тестов
  - в правах на запись\исполнение файлов на устройстве
  - в проверках
    (особенно target <-> host)
- Имейте ввиду, что в NDK Android всегда имеет флаг –fpic
- Все ключи компилятора можно посмотреть -###

Профилирование

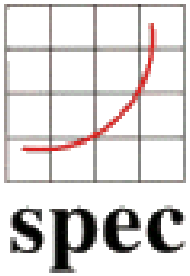
- perf
- oprofile
- sep

Профилирование

- adb push /path/to/bin/perf /data/local
- adb shell
- cd /data/local
- chmod 777 perf
- perf record ./coremark.exe 0x0 0x0 0x66 0 7 1 2000 # or any arguments required
- # data saved at ./perf.data
- export PAGER=cat # otherwise it will look for "less"
- perf report

Тестирование производительности

- На результат влияют:
  - режим работы процессора
  - приложения, запущенные в фоне
  - «чистота» запуска, и ошибки (которые можно и нужно отслеживать через logcat/dmesg)



Тестирование производительности

- Убедитесь, что:
  - Отключены все второстепенные службы и приложения
  - Устройство используется монопольно
  - На устройстве жёстко выставлен режим работы
  - Нет перегрева устройства (или используется динамический мониторинг)

Тестирование производительности

- Например для тестирования CPU типичная практика – это установка всех процессорных ядер и частот в равное значение и их фиксация. При необходимости отключение всех ядер, кроме того, на котором происходит тестирование.
  - adb shell
  - grep 2000000 /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
  - grep 2000000 /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
  - adb shell
  - echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
  - echo 2000000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
  - echo 2000000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
- Убедитесь, что
  - */sys/devices/system/cpu/cpu0/cpufreq/online* так же включено или выключено.
- Только убедившись, что всё работает в нужном режиме (cat /proc/cpuinfo) можно начинать тестирование и анализ.

Удобство использования

- Под Android существует набор утилит busybox, заметно упрощающий работу в native-режиме. Busybox не входит в образ Android по умолчанию.
- Установка возможна только от прав администратора (root).
- Перечень утилит:

  [, [[, ar, arp, awk, base64, basename, bbconfig, beep, blkid, blockdev, bootchartd, bunzip2, bzcat, bzip2, cal, cat, catv, chat, chattr, chgrp, chmod, chown, chpst, chroot, chrt, chvt, cksum, clear, cmp, comm, cp, cpio, crond, crontab, cttyhack, cut, dc, dd, deallocvt, depmod, devmem, diff, dirname, dmesg, dnsd, dos2unix, dpkg, dpkg-deb, du, dumpkmap, echo, ed, egrep, env, envdir, envuidgid, expand, expr, fakeidentd, false, fbset, fbsplash, fdflush, fdformat, fdisk, fgconsole, fgrep, find, findfs, flash_lock, flash_unlock, flashcp, flock, fold, free, freeramdisk, fsync, ftpd, ftpget, ftpput, fuser, getopt, grep, gunzip, gzip, halt, hd, hdparm, head, hexdump, httpd, hwclock, ifconfig, ifdown, ifup, init, inotifyd, insmod, install, iostat, ip, ipaddr, ipcalc, iplink, iproute, iprule, iptunnel, klogd, less, linuxrc, ln, loadkmap, losetup, lpd, lpq, lpr, ls, lsattr, lsmod, lsof, lspci, lsusb, lzcat, lzma, lzop, lzopcat, makedevs, makemime, man, md5sum, mdev, mesg, mkdir, mkfifo, mknod, mkswap, mktemp, modinfo, modprobe, more, mpstat, mv, nbd-client, nc, netstat, nice, nmeter, nohup, od, openvt, patch, pidof, ping, pipe_progress, pmap, popmaildir, poweroff, powertop, printenv, printf, ps, pscan, pstree, pwd, pwdx, raidautorun, rdev, readlink, readprofile, realpath, reboot, reformime, renice, reset, resize, rev, rm, rmdir, rmmod, route, rpm, rpm2cpio, rtcwake, run-parts, runsv, runsvdir, rx, script, scriptreplay, sed, sendmail, seq, setconsole, setkeycodes, setlogcons, setserial, setsid, setuidgid, sha1sum, sha256sum, sha3sum, sha512sum, showkey, sleep, smemcap, softlimit, sort, split, start-stop-daemon, strings, stty, sum, sv, svlogd, switch_root, sync, sysctl, tac, tail, tar, tcpsvd, tee, telnet, telnetd, test, tftp, tftpd, time, timeout, top, touch, tr, traceroute, true, ttysize, tunctl, tune2fs, udpsvd, uname, uncompress, unexpand, uniq, unix2dos, unlzma, unlzop, unxz, unzip, uptime, usleep, uudecode, uuencode, vconfig, vi, volname, watch, wc, wget, which, whoami, whois, xargs, xz, xzcat, yes, zcat

Вопросы?

- Спасибо за внимание!

Contacts

Thank You and

We Look Forward to Working with You

Auriga, USA
92 Potter Rd, Ste 1
Wilton, NH 03086, USA
Phone: +1 (866) 645-1119
Fax: +1 (603) 386-6097
info@auriga.com
www.auriga.com

Auriga, Russia
125 Varshavskoe Shosse, Unit 16A
Moscow, 117587
Tel:+7 (495) 713-9900
Fax:+7 (495) 939-0300
info@auriga.com
www.auriga.com