Porting Replicant to Android 10

This page contains old build instructions for Replicant 10. The source code is kept to do regression testing. Active development has moved into Replicant 11.

Replicant 9 source code and build instruction have been kept to do regression tracking: Porting Replicant to Android 9.

Precautions

See RunningReplicant11 before installing Replicant 10 on your device to not break it.

Building Replicant 10

Source code

```
$ repo init -u https://git.replicant.us/replicant-next/manifest.git -b replicant-10-dev
$ repo sync
```

Alternatively a shallow copy of the source tree can be fetched in order to save on disk space:

```
$ repo init -u https://git.replicant.us/replicant-next/manifest.git -b replicant-10-dev --depth=1
$ repo sync -c
```

To unshallow a specific module:

```
$ cd path/to/module
$ git fetch --unshallow <remote>
```

Build dependencies

For Trisquel 8

```
sudo apt-get install bc bison build-essential bsdmainutils ccache curl flex g++-multilib gcc-multi
lib gettext git gnupg gperf imagemagick lib32ncurses5-dev lib32readline-dev lib32z1-dev liblz4-too
l libncurses5-dev libsdl1.2-dev libssl-dev libwxgtk3.0-dev libxml2 libxml2-utils lzop python-mako
pngcrush rsync schedtool squashfs-tools xsltproc zip zlib1g-dev
sudo apt-get install gcc-5-arm-linux-gnueabi
```

Fixing the build environment

Allow system binaries for building

By default, the Android 10 build system can only use the prebuilt binaries it ships.

While having binary toolchains is better for reproducible builds, and that the binaries are free software, this creates a number of issues:

- We need to be able to rebuild the binaries, and so far no one did it yet.
- Binaries are way harder to trust than source code and not everyone trust Google.

As GNU/Linux distribution's tools can be rebuilt and are easier to trust, we are using that for now.

Setting the following envrionment variable allows to use your distribution tools:

```
$ export TEMPORARY_DISABLE_PATH_RESTRICTIONS=true
```

Note that setting this variable does not automatically make the build system use only system binaries: if a prebuilt binary exist, it will use it, if not, it will use your system binary.

Lots of further effort must be put into transitioning to the system binaries and/or creating a scripts that would build all the required tools.

Mako (Python) for Mesa

To avoid the following error:

```
16:20:13 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for mo
re information.
[  3% 2585/70375] build out/target/product/i9305/gen/STATIC_LIBRARIES/libmesa_nir_intermediates/ni
r/nir_builder_opcodes.h
FAILED: out/target/product/i9305/gen/STATIC_LIBRARIES/libmesa_nir_intermediates/nir/nir_builder_op
codes.h
/bin/bash -c "python external/mesa3d/src/compiler/nir/nir_builder_opcodes_h.py external/mesa3d/src
/compiler/nir/nir_opcodes.py > out/target/product/i9305/gen/STATIC_LIBRARIES/libmesa_nir_intermedi
ates/nir/nir_builder_opcodes.h"
Traceback (most recent call last):
  File "external/mesa3d/src/compiler/nir/nir_builder_opcodes_h.py", line 106, in <module>
    from mako.template import Template
ImportError: No module named mako.template
16:20:20 ninja failed with: exit status 1
```

You need to run the following command:

```
$ cd prebuilts/build-tools/path/linux-x86/
$ rm python && ln -s /usr/bin/python python
```

Java heap space

The Java heap size is automatically set according to the available system memory. On machines with 8 GB or less RAM, it is set to a value which is too low, and will result in the following error during the build:

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
```

The heap size can be increased with an envirnoment variable:

```
$ export _JAVA_OPTIONS="-Xmx3g"
```

Reduce parallel jobs to avoid killed processes

Increasing the Java heap space is not enough to get a successful build on machines with 8 GB or less RAM. It is also necessary to reduce the number of parallel jobs, to avoid processes from being killed due to lack of memory. This typically happens during the build of frameworks/base components.

For greater speed, you may let your build run with the defaults, wait for it to fail due to killed processes, and then relunch the build with:

```
$ make -j1
```

By default, Ninja, the underlaying build system for Android, used when you run make bacon, computes the number of parallel jobs according to the number of CPUs on your machine (typically #CPUs + 2 parallel jobs).

Launching the build

```
$ source build/envsetup.sh
$ lunch lineage_i9305-eng
$ make
```

Install the images

From scratch

```
$ cd out/target/product/i9305
$ sudo heimdall flash --BOOT boot.img --USERDATA userdata.img --SYSTEM system.img
```

Update previous installation

```
adb remount
```

```
adb sync
```

Get adb

As the device IDs are the ones given by the Linux kernel, they are not in the adb udev rules, so for now it requires to run adb as root:

```
$ sudo adb shell
* daemon not running; starting now at tcp:5037
* daemon started successfully
i9305:/ #
$ sudo adb kill-server
$ adb shell
* daemon not running; starting now at tcp:5037
* daemon started successfully
error: no devices/emulators found
```

So make sure to kill the adb-server and run it as root:

```
$ adb kill-server
$ sudo adb shell
* daemon not running; starting now at tcp:5037
* daemon started successfully
i9305:/ #
```

Boot progress

You can also follow the boot progress with adb:

```
adb logcat
adb logcat -b main
```

Note that the device can go into suspend at any time, so adb might be interrupted. That looks like that:
First you get a shell

```
$ sudo adb shell
i9305:/ #
```

Then the connection is interrupted:

```
$ adb shell
i9305:/ # [randomdev@fullyfreelaptop ]$
```

The effect with adb logcat is similar.

Getting the latest changes

- The repositories are being constantly modified with git push --force as we are trying things out, and don't want to make the commits history look too dirty, so be sure to backup your local changes.
- Sometimes the manifest repository is also modified with git push --force. In that case the following commands will loose all the work you did locally but will make the repository consistent with upstream repositories again:

```
$ rm -rf .repo/manifests .repo/manifests.git .repo/manifest.xml
$ repo init -u https://git.replicant.us/replicant-next/manifest.git -b replicant-10-dev
$ repo sync --force-sync
```

- The following command might also be necessary to make the state consistent with upstream repositories again, when the manifest history wasn't rewritten, but it will also loose all the work you did locally:

```
$ repo sync --force-sync
```

Build VM

If you use Parabola, you may be interested in running Trisquel 8 in LXC.

To do that first debootstrap a Trisquel 8 rootfs.

Parabola's debootstrap does support Trisquel 8 and its manual has an example on how to do that:

```
$ man debootstrap
[...]
# debootstrap flidas flidas-root http://archive.trisquel.info/trisquel
```

Then you can use virt-manager to setup the LXC instance.

The advantages of this solution are that:

- The LXC guest and host shares their resources (CPU, RAM) with almost no penalty
- Trisquel 8 is not a rolling release distribution

The disadvantage of this solution are that:

- you need to configure the Trisquel 8 LXC instance (vimrc, sshd_config, etc)
- It's more complicated to setup
- The Android build system outputs a warning message about not being able to use namespaces which may become mandatory in newer Android versions

Cleanups to be done

- Make adb work as user by using the right USB IDs, and make userspace do the USB setup.
- ~~Make the kernel not use hardcoded CMDLINE_FORCE~~ Not possible unless the bootloader is changed or Linux is very heavily patched.
- Make the kernel not use hardcoded partitions if possible (though we use system as root)
- Make a clean Gatekeeper HAL module implementation instead of using the same hack than goldfish
- Look at init.rc documentation to see if init.rc can be overriden clearly with the override statement to see if it's possible to keep the serial console patch for -eng

Upstreaming status

- The stock bootloader is incompatible with Linux, see BootloadersIncompatibleWithLinux for more information. So we maintain patches to enable the Galaxy SIII, and Galaxy Note II to boot with the stock bootloader. In the long run we need to look into using u-boot in the kernel partition as using u-boot instead of the stock bootloader currently require nonfree and non-redistributable software (BL1).
- For the patches that are not merged yet, see the issues of the redmine upstreaming sub-project

Graphics status

Progress of the graphics related tasks is tracked at GraphicsReplicant10.

Modem status

libsamsung-ipc:

- Fully tested under GNU/Linux only
- Can initialize completely the modem and receive messages (see the #1954 bug report for the logs)
- Needs more cleanup but there is now a better abstraction

libsamsung-ril:

- Ported to Replicant 9 using a wrapper for the API >=12 in libsamsung-ril source code that needs to be removed
- Tested under Replicant 9 without up to date libsamsung-ipc (no modem init)
- Tested and validated under Replicant 6 (doesn't break telephony)

Modem status TODO

- Implement the missing part to shut down the modem, close the interfaces and such, in order to need to reboot after each test.
- Continue to clean up the libsamsung-ipc and libsamsung-ril patches, test the patches in Replicant 6 when applicable, and merge them in the upstream repositories.
- Convert the firmware loading driver to the upstream API and then adapt libsamsung-ipc for that. This should also benefit other devices like the Galaxy SIII 4G, and the Galaxy Note II 4G which probably don't need much more to get their modem supported by upstream Linux.
- Cleanup and convert the rest of the drivers to look like the ones for the Nokia N900 and adapt the userspace in libsamsung-ipc, and merge libsamsung-ipc support for that once the kernel API is stable.
- Test the code under Replicant 10 when the graphics status will enable to have decent enough speed to do some testing through human interaction.

TODO

First month of full time equivalent work:

Time estimation	Task	Comments
DONE	~~boot a device under AOSP9~~	Only boots with graphics, not much more
~~7h~~ DONE	~~build it under a~~ FSDG compliant distribution ~~like Trisquel8~~	~~WIP for AOSP, It's difficult to do precise time estimations as it could work out of the box or require one full time month of work depending on how much issues are encountered~~ Builds under Trisquel8
~~21h~~ DONE	* ~~port the changes from AOSP9 to LineageOS 16~~ * ~~cleanup the code~~ * ~~build the kernel from the Android build system~~ * ~~make sure it builds with an FSDG compliant distribution~~ * ~~document the build procedure~~	Status: * Boots with adb. * Has ultra slow graphics
14h	find, remove and document proprietary software in LineageOS 16	
21h	find, remove and document privacy issues in LineageOS 16	
7h	~~Add support for the touch keys driver in the~~ ~~galaxy-s3 dts~~	applied
7h	~~upstream the AAT1290 flash led Linux dts for the galaxy-s3 boards~~	Now in 5.3
7h	rebrand LineageOS as Replicant	
70h	~~port and~~ cleanup the the Galaxy SIII (i9300) modem Linux driver from 4.16 to 5.0	See the modem status for more details
Total: 147h (~1 month)		

Second month of full time equivalent work:

* port libsamsung-ril and libsamsung-ipc to Android 9 * Make the modem driver and libsamsung-ipc work together	157h	See the modem status for more details
Total	157h	~1 month

Third month of full time equivalent work:

Task	Time estimation	Comments
~~port the sensors libraries and other device specific libraries~~ Look which sensor libraries can be used	70h	Already done by the unofficial LineageOS port of the Galaxy SIII (i9300), needs testing

	add support for Audio with the upstream kernel driver	70h	Might be way faster, depending on what Android 9 uses See also this bugreport
	~~add partial (no modem) support for the Galaxy SIII 4G (i9305) and~~ factorize the code with i9300	14h	* The source code on which the work was based changed from AOSP to an unofficial LineageOS port to a port of i9305 support for AOSP by Joonas to the official LineageOS so it's now supported by default * The work to factorize the code between the i9300 and i9305 still need to be done

Total	154h	~1 month

Task	Time estimation	Comments
create a recovery	21h	
add internal WiFi support and validate the functionality	6h	
add external WiFi dongles support	20h	External dongles support might be tricky
~~create new~~ update the install and upgrade instructions	~~35h~~	~~Our current install instructions don't scale as we have one copy for each device.~~ ~~We also created generic instructions but they tend to be harder to follow[1] than the device specific ones.~~ ~~This will be made in a modular format (for instance in LaTeX) that enables to generate per device install instructions without requiring to have multiples copy of the same text.~~ ~~The instructions will need to be able to be modified and compiled on an FSDG compliant distribution.~~ Mostly done: * The installation instructions are now generic enough. * Some long standing TODO were also done along the way like adding backup instructions for the EFS. * The current instructions are still for Replicant 6.0 and will need to be updated for Replicant 9.0

Task	Time estimation	Comments
Estimate the amount of work to Reduce the attack surface	?	
Estimate the amount of work to add in-system upgrades	?	

[1] The generic instructions were tested at Install parties in Paris

Devices support:

Easy, because it's similar enough to the Galaxy SIII (I9300)

Galaxy Note II (N7100)		
Task	Time estimation	Comments
port the EA8061 LCD Linux driver	35h	

port the S6EVR02 LCD Linux driver	35h	
port the MAX77693 flash led Linux driver	7h	
android: add support for the Note II (N7100) and factorize the code with Galaxy SIII (i9300) and Galaxy SIII 4G (i9305)	14h	Should be similar to the Galaxy SIII
port the sensors libraries and other device specific libraries	70h	It's difficult to evaluate how much time it could take
add support for Audio with the upstream kernel driver	14h	Should be similar to the Galaxy SIII

Galaxy Note 8.0 (N5100) and 8.0 WiFi (N5110)		
Task	Time estimation	Comments
Evaluate the time required to do the port	14h	TODO

Needs more work and unknown upstream Linux status

Device	Time estimation	Comments
Galaxy S II (i9100)		Linux: devboard dts upstream? unknown status
Galaxy Note (N7000)		unknown Linux upstream status
Galaxy Nexus (I9250)		OMAP4, no dts upstream
Galaxy Tab 2 7.0 (P3100), 7.0 WiFi (P3110), 10.1 (P5100), 10.1 WiFi (P5110)		
GTA04 >= A4		TODO: a RIL needs to be written, userspace GPS support is missing, audio scenarios, etc

Documentation

Replicant 6.0 changes

See Replican6Changes.

Other rebases

See the Samsung-ipc page.

Other attempts

- It might be interesting to contact the people doing ports once we have something working well enough.
- It is also interesting to look at other attempts to understand if a given device is powerful enough to run Android 9 and what configuration was used to achieve it.

Device(s)	Repository	status	Comments
i9300	CustomROMs	* February 8 2020 Pie release	
i9300	Team InFusion	* August 20 2019 Pie release	Issues: * Uses a Samsung kernel * Uses too many nonfree libraries => Probably nothing we could reuse from its code
n7100	ComicoTeam	* January 4 2020 Pie release	
i9100	rINanDO	* March 20 2020 Pie release * July 19 2020 Android 10 release	

Links for other attempts

- https://www.xda-developers.com/android-pie-android-9-port-custom-roms/

CustomROMs i9300 components

Repository	Tree path	Dependencies	Function	Comments
https://github.com/CustomROMs/android_hardware_samsung lineage-16.0 branch	hardware/samsung/macloader		Loads the MAC Address of the WiFi network interface	Might be useful
	hardware/samsung/wifiloader		Loads the wifi kernel module (like modprobe) and setup firmware filesystems permissions	May be useful
	hardware/samsung/audio		seems to contains ril related stuff as well	Look if the ril stuff is required, go for standard audio
	hardware/samsung/lineagehw/hidl/livedisplay		livedisplay is a feature similar to what redshift does on GNU/Linux	Not sure if it works with mainline
	hardware/samsung/exynos/multimedia/utils/	seem meant for audio/video decoding offload	assembly obtimized color conversion and resize code	check assembly code license, not sure if useful
	all other directories in hardware/samsung/exynos/	nonfree firmwares, nonfree software?, smdk kernel?	audio/video decoding offload	Avoid using that
	hardware/samsung/exynos3	nonfree firmwares?, nonfree software?, smdk kernel?	some light libraries, display stuff (gralloc, etc), 2D acceleration (FIMG), camera (FIMC), 3D acceleration, etc	Avoid using that for now
	hardware/samsung/exynos4			

Known error messages that are safe to ignore

- TestHarnessModeService: Failed to start Test Harness Mode; no implementation of PersistentDataBlockManagerInternal was bound
- JniUtils: Could not load native library jni_latinimegoogle

Links

- Android build requirements - hardware and software
- Why LineageOS Developers are building Android Go-optimized custom ROMs
- Android Go recommended default values for propreties for optimization
- Team InFusion i9300 optimized system.prop
- Hack to fix high CPU usage caused by logd
- Use low-end video codecs
- Optimize ActivityManager cached apps
- Use 1Gb Dalvik config