What's going on with Snaps on Ubuntu Touch?

A technical deep dive

Alfred Neumayer, 2024

Agenda

Ubuntu Touch architecture

libhybris involvement

snapd changes

Integration changes

The All-Snap (TM) deal

Ubuntu Touch architecture

Individual pieces

Kernel

Early boot

Rootfs

Halium

Ubuntu Touch architecture

Ways of operation

- Mainline devices
  - Mainline kernel
  - Mesa
  - Most things work already
- Halium devices
  - Android drivers and HAL processes
  - Modified Android vendor kernels

Wait what?

Ding dong: time to resolve differences

- Android's generational fragmentation
- Kernelspace
    - Graphics and memory allocation
    - Out-of-memory killer in Android < 9.0
- Userspace
    - Interfaces to kernel APIs
    - "Binderization" of HALs vs loading Android libraries
- Initramfs vs Initramfs-less vs Multiple Concatenated Initramfs
- Introduction of A/B boot slots

Kernel

Requirements

- AppArmor
  - Downstream patches taken from a similar LTS patchset
  - Applied on top of Android vendor kernel
- Enabling namespaces & various kernel features
  - Namespacing for Halium LXC container
  - CONFIG_VT, CONFIG_SYSVIPC
  - Keeping kernel ABI stable

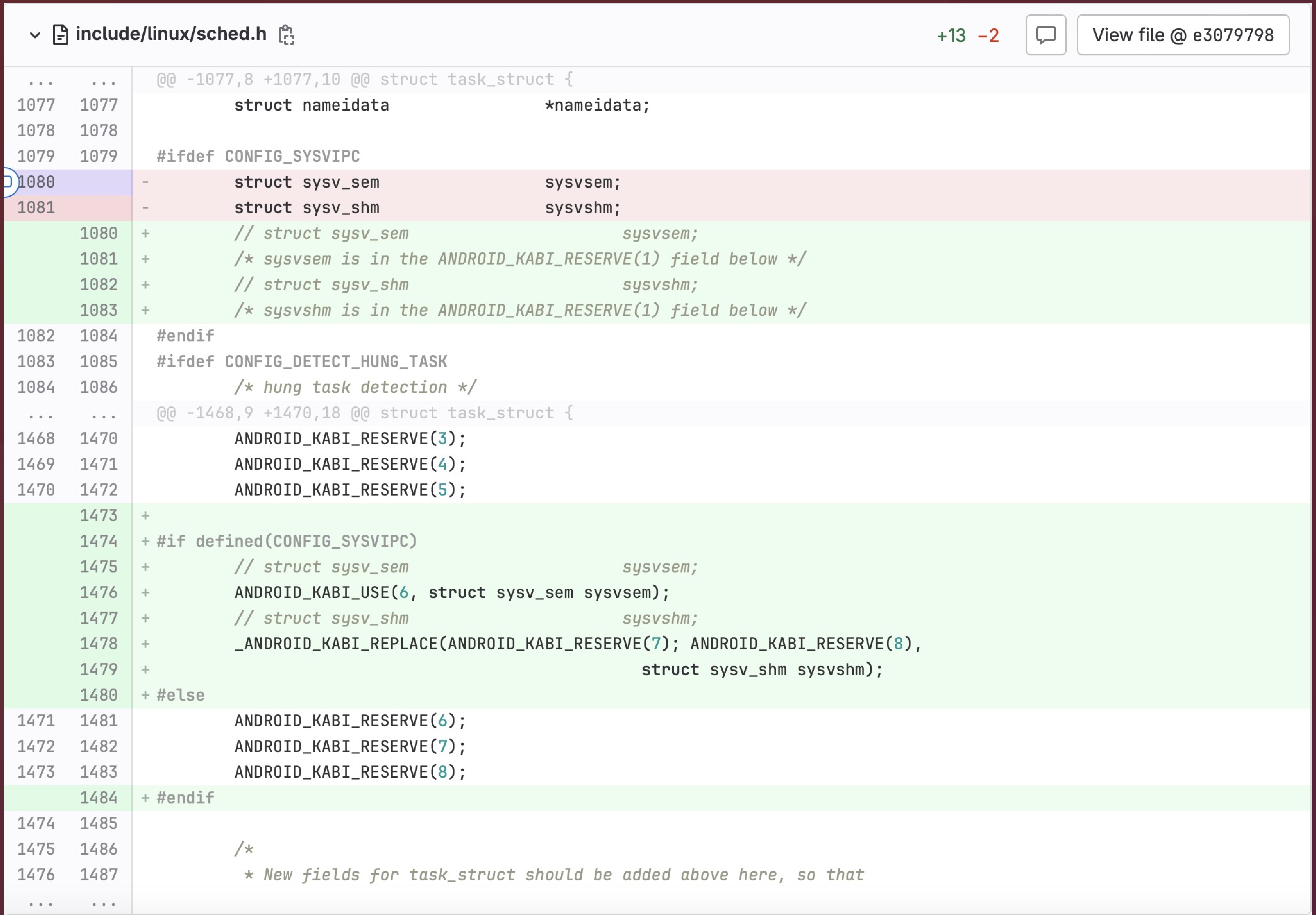
Kernel

Android standardization

- Generic Kernel Images
    - Stable kernel interface
    - Android vendors must follow
    - Kernel modifications to keep struct sizes compatible
    - Padding for struct members

Kernel

SYSVIPC ABI compatibility

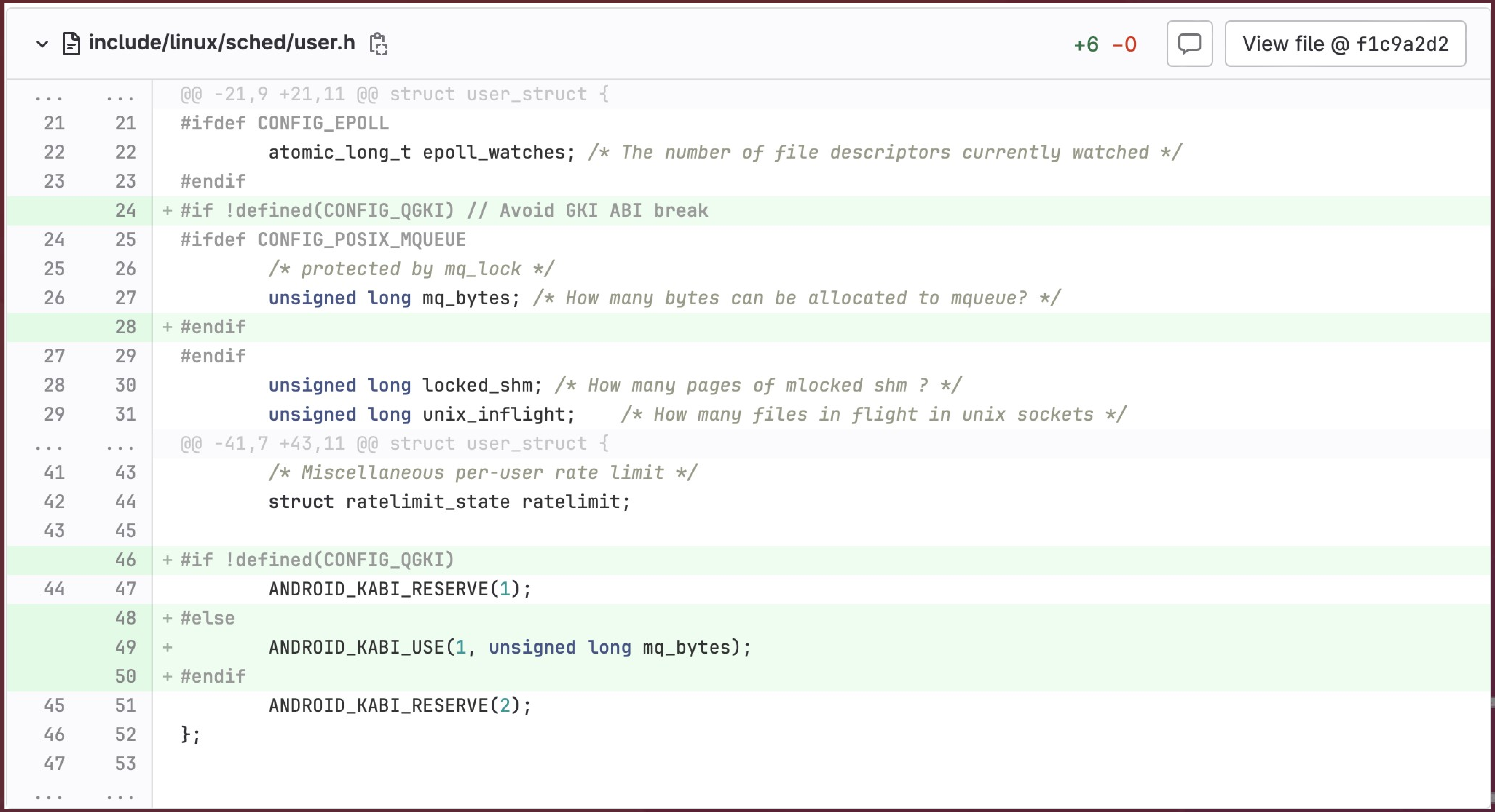


include/linux/sched.h +13 −2 View file @ e3079798

```diff
@@ -1077,8 +1077,10 @@ struct task_struct {
 	struct nameidata		*nameidata;
 
 #ifdef CONFIG_SYSVIPC
-	struct sysv_sem			sysvsem;
-	struct sysv_shm			sysvshm;
+	// struct sysv_sem				sysvsem;
+	/* sysvsem is in the ANDROID_KABI_RESERVE(1) field below */
+	// struct sysv_shm				sysvshm;
+	/* sysvshm is in the ANDROID_KABI_RESERVE(1) field below */
 #endif
 #ifdef CONFIG_DETECT_HUNG_TASK
 	/* hung task detection */
@@ -1468,9 +1470,18 @@ struct task_struct {
 	ANDROID_KABI_RESERVE(3);
 	ANDROID_KABI_RESERVE(4);
 	ANDROID_KABI_RESERVE(5);
+
+#if defined(CONFIG_SYSVIPC)
+	// struct sysv_sem				sysvsem;
+	ANDROID_KABI_USE(6, struct sysv_sem sysvsem);
+	// struct sysv_shm				sysvshm;
+	_ANDROID_KABI_REPLACE(ANDROID_KABI_RESERVE(7); ANDROID_KABI_RESERVE(8),
+						  struct sysv_shm sysvshm);
+#else
 	ANDROID_KABI_RESERVE(6);
 	ANDROID_KABI_RESERVE(7);
 	ANDROID_KABI_RESERVE(8);
+#endif
 
 	/*
 	 * New fields for task_struct should be added above here, so that
```

Kernel

Enabling POSIX_MQUEUE



include/linux/sched/user.h

+6 −0

View file @ f1c9a2d2

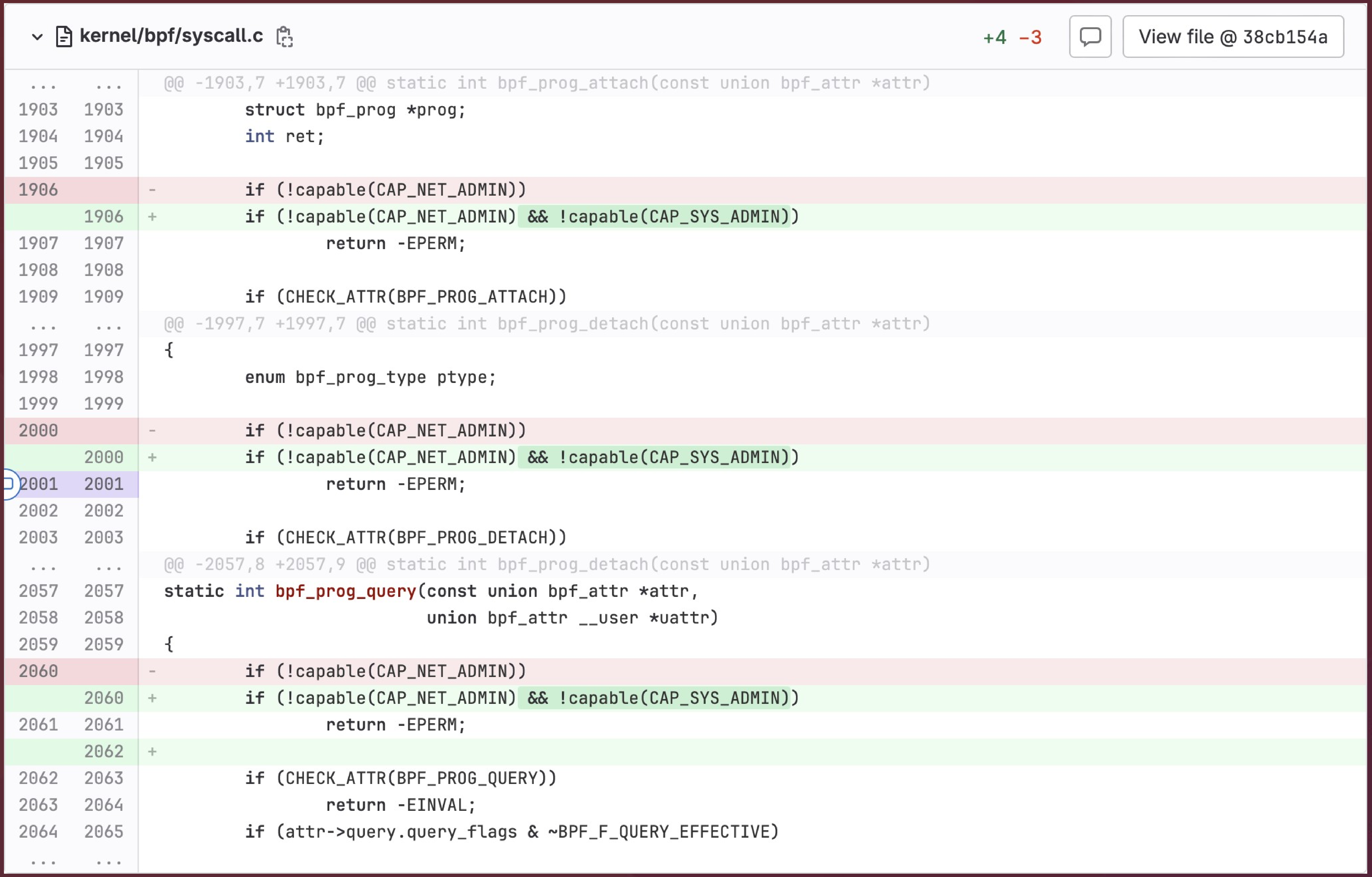
```diff
  @@ -21,9 +21,11 @@ struct user_struct {
  #ifdef CONFIG_EPOLL
          atomic_long_t epoll_watches; /* The number of file descriptors currently watched */
  #endif
+ #if !defined(CONFIG_QGKI) // Avoid GKI ABI break
  #ifdef CONFIG_POSIX_MQUEUE
          /* protected by mq_lock */
          unsigned long mq_bytes; /* How many bytes can be allocated to mqueue? */
+ #endif
  #endif
          unsigned long locked_shm; /* How many pages of mlocked shm ? */
          unsigned long unix_inflight;    /* How many files in flight in unix sockets */
  @@ -41,7 +43,11 @@ struct user_struct {
          /* Miscellaneous per-user rate limit */
          struct ratelimit_state ratelimit;

+ #if !defined(CONFIG_QGKI)
          ANDROID_KABI_RESERVE(1);
+ #else
+         ANDROID_KABI_USE(1, unsigned long mq_bytes);
+ #endif
          ANDROID_KABI_RESERVE(2);
  };

```

Kernel

Device access with cgroups v2 & eBPF



kernel/bpf/syscall.c +4 −3 View file @ 38cb154a

```diff
@@ -1903,7 +1903,7 @@ static int bpf_prog_attach(const union bpf_attr *attr)
         struct bpf_prog *prog;
         int ret;

-        if (!capable(CAP_NET_ADMIN))
+        if (!capable(CAP_NET_ADMIN) && !capable(CAP_SYS_ADMIN))
                 return -EPERM;

         if (CHECK_ATTR(BPF_PROG_ATTACH))
@@ -1997,7 +1997,7 @@ static int bpf_prog_detach(const union bpf_attr *attr)
 {
         enum bpf_prog_type ptype;

-        if (!capable(CAP_NET_ADMIN))
+        if (!capable(CAP_NET_ADMIN) && !capable(CAP_SYS_ADMIN))
                 return -EPERM;

         if (CHECK_ATTR(BPF_PROG_DETACH))
@@ -2057,8 +2057,9 @@ static int bpf_prog_detach(const union bpf_attr *attr)
 static int bpf_prog_query(const union bpf_attr *attr,
                           union bpf_attr __user *uattr)
 {
-        if (!capable(CAP_NET_ADMIN))
+        if (!capable(CAP_NET_ADMIN) && !capable(CAP_SYS_ADMIN))
                 return -EPERM;
+
         if (CHECK_ATTR(BPF_PROG_QUERY))
                 return -EINVAL;
         if (attr->query.query_flags & ~BPF_F_QUERY_EFFECTIVE)
```

Early boot

initramfs

- initramfs-tools-halium
  - Fork of Ubuntu Touch’s previous initramfs
  - Builds on top of Debian initramfs environment
  - Optionally sets up Android super partition
  - Mounts system & writable userdata partitions
  - Mounts early writable bind-mounts from /etc
  - Very similar to Ubuntu Core’s previous initramfs
- Reason: It looks like a regular GNU/Linux bootup environment

Early boot

initramfs-less

- Jumpercable
  - Soft-fork of initramfs-tools-halium
  - Simple Bash script setting up basic environment
  - Sits in /init in the rootfs/system partition
  - Mounts writable userdata
  - Mounts early writable /etc bind-mounts
  - Chainloads systemd afterwards
- Reason: Google made the recovery partition optional in Android 9

Early boot

Concatenated initramfs

- Introduced in Android 12
  - Android ships a generic initramfs
  - Vendors add their own in a separate partition
    - Setup scripts
    - Kernel modules
  - Concatenated by the bootloader
    - Gets Halium initramfs concatenated with it
  - Passed to the kernel
- Reason: Google reintroduced recovery partitions + A BUNCH OF OTHERS

Early boot

Android’s "super" partition

- Introduced in Android 10
- LVM-style volume container
  - Custom header format
  - Convertible to something usable using `parse-android-dynparts`
- Contains vendor blobs in various volumes
- With A/B variants per volume

Rootfs

The actual system

- Ubuntu as a base
  - arm64, amd64, armhf
  - systemd
  - Typical Ubuntu userspace libraries and services
- lxc-android-config
  - Mounts typical Android partitions
  - Sets up remaining writable paths from read-only partition to writable partition
  - Initializes Halium LXC container
  - Sets up optional device-specific hacks

Rootfs

Providing usability

- Additional services
  - Mir for handling displays and input
  - Device-specific services (telephony, sensor frameworks...)
    - ofono
    - sensorfw
    - hfd-service
  - Functionality & UX services (online accounts, download manager...)

Halium

Hardware enablement

Stripped down Android environment (mostly C/C++ components)

Running in a LXC container

Android /init starts HAL services

Just enough to have hardware enablement services running

Typically used IPC mechanisms:

- Sockets
- Binder

Halium

Generic system images

Android „Generic system image“

- One single Android /system for all devices
- Started with Project Treble
- Eases burden on device port maintainers
- Binderized HALs shine here

Binder

Android's preferred IPC mechanism

- In-kernel IPC
- Multiple contexts
  - One for the Android framework (or our stub services)
  - One for hardware services
  - One for vendor services
- Previously: debate about D-Bus-over-Binder
  - Discussions stopped
  - Main reason: differences in threading model

Hardware support

Make it work

- Different services drive our hardware stack
    - PulseAudio ↔ hybris-loads Android audio HAL .so
    - sensorfw ↔ Halium-side vendor sensor HAL
    - repowerd ↔ Halium-side vendor PowerHAL
    - ofono ↔ Halium-side vendor RIL daemon
    - NetworkManager
    - Location services ↔ Halium-side vendor GPS HAL
- Deprecating the unity8 platform-api
    - GPS is the only remaining consumer

Halium overlays

Hardware enablement

- Configurations differ between devices
- Often between devices of the same SoC manufacturer
- Halium overlays
  - Checks for files to overlay from multiple places
  - Released as device tarballs
  - Refreshed from UBports CI & system-image
- Hint: a true device integration ("port") needs this

libhybris

Making drivers pop

- Android library loader built as a glibc library
- Provides glibc wrappers for libEGL, libGLESv2 etc.
- Shoves in Wayland listeners to handle Android buffer passing
- Requires some ability to talk to typical Android services
  - "Which linker should I use?"
  - -> asks over a socket to the property service running in Halium container
  - Resolves symbols with the correct linker

libhybris

As used in the wild

- Typical procedure when used by an app
  - Load libEGL & libGLESv2
  - Resolve glibc symbols to bionic libc equivalents
  - Optionally set up hooks for redirection
  - Call bionic function in the back when calling glibc function
- "Compatibility layers"
  - Media encoding & decoding (OMX)
  - Camera (Android camera stack)
  - Graphics (Android’s GraphicBuffers)

libhybris

The old way

Using Mirclient:

libhybris driver userspace gets loaded

Linker setup

Mirclient Android client implementation gets loaded

Android EGL initializes in the back

libhybris

The new way

Using Wayland:

- libhybris driver userspace gets loaded
- Linker setup
- wayland-egl loads libhybris’ libEGL implementation
- Sets up Wayland buffer passing on active socket
- Android EGL initializes in the back

TLS padding hack

An ugly necessity

- Processes have two libc implementations loaded now
  - glibc process starts up
  - libhybris loads bionic libraries afterwards
- Both might have different thread-local storage requirements
  - Proprietary drivers statically linking bionic
- Solution: LD_PRELOAD a library with the only contents being:

```
thread_local void* tls_padding[16];
```

snapd changes

Fulfilling requirements

- Kernels need adaptations for Snap support
  - AppArmor is recommended
  - SQUASHFS with compression
  - Potentially requires device cgroup v2 enablement
  - Relies on device maintainers

snapd changes

Making it work

- Enablement patches for running on Ubuntu Touch
  - Read-only rootfs
  - Not classic, not Snap-only
  - Needs to choose ways appropriate for environment
    - libhybris environment setup
    - extrausers

Integration changes

Ubuntu Touch apps in the Snap Store?

- Services integration with Snaps
  - Content Hub
  - Media Hub
  - Download Manager
  - Online Accounts
- Adaptations due to Lomiri renaming
  - D-Bus interfaces & AppArmor policies
- Hooking things up
  - Content Hub would need metadata generated by snapd

The All-Snap (TM) deal

Ubuntu Core on Android hardware?

Vendor blobs are often enough not redistributable

We need /android + symlinks in many Snap runtime environments

Early-boot challenges from version to version

snapd & Gadget Snaps are too far away (right now) to work with Android bootloaders

Google dictates the partition layout, not you or me

Still old vendor kernels, but we’ve learned to live with it

The All-Snap (TM) deal

... maybe?

- Personal idea: Snapium
  - Halium initramfs partition setup + Ubuntu Core initrd
  - No idea how much Core initrd diverged from Debian initrd
  - Project outside of customer-focused Ubuntu Core
  - Initial mindset: "It might never truly be Ubuntu Core"
  - No proof-of-concept yet
- Hire me?
  - Let’s get shot done!

Resources

Learn more & how to join

https://ubuntu-touch.io

https://ubports.com

https://halium.org

https://lomiri.com

https://fredl.me

Thank you!