



Robustness

1	Contents	
2	Introduction	2
3	Requirements	3
4	Approach	3
5	Application data	3
6	General guidelines	3
7	SQLite	4
8	Tracker	4
9	User settings	5
10	Media	5
11	Caches	5
12	Filesystems	5
13	Root filesystem	9
14	Other filesystems	9
15	Main storage	9
16	Removable devices	10
17	Mitigating the effects of lack of disk space	10
18	Resource management	11
19	CPU	12
20	I/O	13
21	Memory	13
22	Network queue	14
23	GPU	14
24	Accounting	15
25	USB undervoltage	15
26	Risks	15
27	Design notes	16
28	BTRFS Overview	16
29	BTRFS robustness supporting features	16
30	Cheap, fast, and atomic snapshots and rollback	16
31	Repair and recovery	17
32	Checksumming	17

33 Introduction

34 This design identifies circumstances that, though undesired because of the risk
35 of loss of functionality, cannot be completely avoided and provides suggestions
36 for dealing with them in such a way that as little functionality as possible is
37 lost.

38 Note that improving D-Bus's robustness is a topic that will be covered in a later
39 stage in its own design document. About securing D-Bus services, please see
40 the security design.

41 **Requirements**

42 Minimize loss of data and loss of functionality due to data corruption in these
43 abnormal circumstances:

- 44 • Unexpected power loss
- 45 • Unexpected removal of storage devices
- 46 • Unexpected lack of disk space
- 47 • Physical damage to the media and other hardware errors

48 Minimize loss of functionality due to processes hogging these shared resources:

- 49 • CPU
- 50 • GPU
- 51 • I/O
- 52 • memory
- 53 • network queue
- 54 • D-Bus daemon

55 **Approach**

56 This section explains how to address the requirements in several specific cases,
57 taking into account different data sets and circumstances.

58 **Application data**

59 This section contains recommendations about how to robustly deal with data
60 generated by applications.

61 **General guidelines**

62 No software should assume that opening files will always succeed. Failure condi-
63 tions should be dealt with and the process will either continue running with as
64 little loss of functionality as possible, or will log a message and exit. Programs
65 should do the same when writing data (the filesystem may be full, or any other
66 mode of error might occur).

67 For example, if the browser application finds out at start up that the cookies
68 file is corrupted, it should move the old file away (or just delete it) and run as

69 usual other than past persistent cookies will have been lost. Or if there was
70 an error when writing a new persistent cookie to disk, the browser would keep
71 running with that cookie being transient (in memory only).

72 In order to reduce the effects of data corruption, regardless of the causes, it
73 would make sense to store different data sets in separate files. So that if cor-
74 ruption happens in, for example, the browser cookie store, it would not affect
75 unrelated functionality such as playlists.

76 For big data sets, Apertis recommends SQLite with either Write-Ahead Logging
77 (WAL¹) or [roll-back journal](http://www.sqlite.org/draft/lockingv3.html#rollback)². For smaller data sets, a robust method is to write
78 to a temporary file and rename it on top of the old one once finished. This
79 method is called “atomic overwrite-by-rename” and is mostly used when editing
80 a file in-place.

81 POSIX requires the atomicity of [overwrite-by-rename](http://www.sqlite.org/draft/lockingv3.html#atomic)³. Btrfs, Ext3 and Ext4
82 give atomic overwrite-by-rename guarantees, as well as atomic truncate guaran-
83 tees. The FAT filesystem guarantees neither.

84 SQLite

85 For applications using SQLite for their storage, Apertis recommends using either
86 WAL or the rollback journal so that transactions are committed atomically. In
87 addition, filesystem-specific tuning would be done by configuring the SQLite
88 system library for optimal performance.

89 WAL will be the best option in most cases, except when transactions will be
90 very big (involving more than 100 MB) and when writes are very seldom, then
91 the rollback journal would be preferred.

92 Apertis will run the [TCL test harness](http://www.sqlite.org/testing.html#tcl)⁴ for SQLite in LAVA, to detect any issues
93 in the specific configuration and software in the target platform. These include
94 robustness tests that reproduce out-of-memory errors, input/output errors and
95 abnormal termination (crashes or power loss).

96 Tracker

97 Tracker stores data in SQLite files, so the robustness considerations that apply
98 to SQLite apply to Tracker as well. By default it uses WAL instead of the tra-
99 ditional rollback journal, which gives better performance for Tracker’s workload
100 with the same robustness guarantees.

¹<http://www.sqlite.org/draft/wal.html>

²<http://www.sqlite.org/draft/lockingv3.html#rollback>

³<http://pubs.opengroup.org/onlinepubs/009695399/functions/rename.html>

⁴<http://www.sqlite.org/testing.html#tcl>

101 **User settings**

102 For configuration settings in general, Apertis recommends using the [GSettings](http://developer.gnome.org/gio/stable/GSettings.html)⁵
103 API from GLib with the [dconf](https://wiki.gnome.org/Projects/dconf)⁶ backend. When updating the database, dconf
104 will write the whole new contents to a new file, then atomically renaming it on
105 top of the old one.

106 For bigger pieces of data (individual settings whose data component exceeds
107 1 KB), Apertis recommends using plain files via a known-robust file-handling
108 library (such as [GKeyFile](https://developer.gnome.org/glib/stable/glib-Key-value-file-parser.html)⁷ from Glib, which is already a dependency) or SQLite.

109 **Media**

110 For media, the meta-data is stored in Tracker, with the actual data files in the
111 /home filesystem and in attached removable devices.

112 If the Tracker database that contains the meta-data has been corrupted, it
113 should be moved to the side (or deleted) and recreated again by indexing all
114 available media files. To minimize the chances of corruption, refer to [Tracker](#).

115 Software that reads the actual media files should assume media files may contain
116 invalid data and ignore them without further loss of functionality. Corrupted
117 media files should not be displayed in the UI.

118 **Caches**

119 All software that uses a cache file should be ready to find that the cache is
120 unusable and cope with it without loss of functionality (temporary degradation
121 of performance is obviously expected in this case though the mechanism by
122 which the cache became corrupted will be treated by developers as a bug to
123 fix).

124 For example, if during start-up the Folks caches are found to be unreadable, lib-
125 folks would remove the corrupted cache files and recreate them, taking a longer
126 time to reply to queries. As the application using Folks would be executing
127 the queries asynchronously, the UI would keep being functional while the query
128 executes.

129 Examples of other components that use caches and that should cope with cache
130 corruption are the browser and the email client.

131 **Filesystems**

132 The reliability with which data is stored depends on both the storage medium
133 as well as the filesystem. In this section, we cover FAT32 and Btrfs. Ext4 is
134 mentioned, as it is a popular default filesystem on many Linux distributions –

⁵<http://developer.gnome.org/gio/stable/GSettings.html>

⁶<https://wiki.gnome.org/Projects/dconf>

⁷<https://developer.gnome.org/glib/stable/glib-Key-value-file-parser.html>

135 however it doesn't suit the needs of the rollback system –either for system roll-
136 backs (See the System Update and Rollback Design) or for application rollbacks
137 (See the Applications Design).

138 The FAT32 filesystem is not robust under abnormal circumstances since it was
139 not made for devices which could be disconnected at any moment. In general, an
140 approach where writes to the device are tightly controlled and restricted to small
141 time-windows would help minimize the chances of corruption. See the *Media*
142 *and Indexing* design for a detailed explanation of the issues and suggestions.

143 The Ext4 filesystem is quite robust under power failure by default. It can be
144 made even more robust by [mounting it under data=journal](#)⁸ mode, but at a
145 large cost to performance.

146 Btrfs has been created on very robust principles, building upon the experience
147 of Ext4. Some brief technical details are provided at the end of this document
148 in [BTRFS overview](#).

149 **Filesystem options** Filesystems usually have parameters that can be tuned
150 to suit specific workloads. Some of them affect performance as well as robust-
151 ness; either by trading off between the two, or by taking advantage of specific
152 hardware features available with the storage media.

153 • **FAT32** is a simple filesystem that does not have many filesystem options
154 related to performance or robustness. Since we will not be creating any
155 FAT32 partitions ourselves, only mount-time options are interesting for
156 us. The recommended options are listed below:

157 – sync, flush These filesystem options ensure that the kernel, as well as
158 the filesystem, flush data to the partition as soon as possible. This
159 greatly reduces the chances of data loss or filesystem corruption when
160 USB drives are yanked out by the user.

161 – ro (read only) It is recommended that FAT32 partitions be mounted
162 read-only to avoid filesystem corruption, and other related problems
163 as detailed in the “*Media and Indexing Design*” in the section “*Index-*
164 *ing database on removable device*”.

165 • **Btrfs** is relatively new, and so does not have many options relevant to
166 our needs of enhancing reliability on eMMC storage media. The available
167 options are listed below.

168 – *Mount-time options:*

169 * commit=number (default: 30) Set the interval of periodic com-
170 mit. This option is recent ([since kernel 3.12](#))⁹.

⁸<http://kernel.org/doc/Documentation/filesystems/ext4.txt>

⁹https://btrfs.wiki.kernel.org/index.php/Mount_options

- 171 * **ssd** This option enables SSD-specific optimizations and disables
- 172 some optimisations specifically for rotating media. This option
- 173 is enabled automatically on non-rotating storage.
- 174 * **Recovery** (default: off) This option can be used to attempt re-
- 175 covery of a corrupted filesystem (See [Repair and recovery](#)).

176 – *Filesystem creation options:*

- 177 * **-s sector-size** This is the size of the filesystem blocks used for
- 178 allocations. Ideally, this should be the same size as the block
- 179 size for the storage medium.

- 180 * **-M**

181 This sets BTRFS to use “mixed block groups”- a mode that stores

182 data and metadata chunks together on disk for more efficient

183 space utilization for small filesystems –but incurs a performance

184 penalty on large ones. This option is not mature and will be

185 evaluated in the future.

186 The System Updates and Rollback Design describes the partition layout for

187 Apertis. Not all the partitions have the same requirements, so both the FAT32

188 and BTRFS filesystems are used. The partitions are configured as:

- 189 • **Factory Recovery** –This partition is never mounted read-write and must
- 190 be readable by the boot loader. Currently the boot loader for Apertis –
- 191 U-boot –does not support BTRFS. While patches exist to add that func-
- 192 tionality, they have not yet seen widespread testing. FAT32 will likely be
- 193 the filesystem chosen for the factory recovery image.
- 194 • **Minimal Boot partitions** –These partitions must also be readable by
- 195 the boot loader, and are currently FAT32. They are not normally mounted
- 196 at run-time, instead they are created, mounted, and populated by the
- 197 system update software once –and only ever accessed by the boot loader
- 198 afterwards. They will be mounted with the “sync”and “flush”flags.
- 199 • **System** - Since BTRFS provides an excellent snapshot mechanism to
- 200 assist system rollbacks (See [Cheap, fast, and atomic snapshots and roll-](#)
- 201 [back](#)), this partition will be populated with a BTRFS filesystem created
- 202 with the appropriate sector size for the storage device. It may be created
- 203 with mixed block groups to save storage space if that option does not lead
- 204 to instability. It will be mounted with the **ssd** option as well as read-only.
- 205 During a system update a single subvolume of the system subvolume will
- 206 be mounted read-write. The repair mount option will never be attempted
- 207 on the system partition, instead rollbacks or factory recovery will be used
- 208 to avoid potentially putting the system into an unknown state.
- 209 • **General Storage** –This partition shares similar requirements to the sys-
- 210 tem partition. It will be BTRFS, created with an appropriate sector size
- 211 and possibly mixed block groups. It will be mounted with the **ssd** option.

212 This is the only built-in non-volatile storage that will always be mounted
213 read-write. In the case of a damaged filesystem, repair may be attempted
214 on this partition.

215 Additionally, there are 2 partitions for raw status flag data that do not use
216 filesystems at all. See the System Updates and Rollback Design for more details.

217 **Checksumming** Checksumming is used for detecting filesystem corruption
218 due to any reason. Different filesystems have different mechanisms for check-
219 summing which give us coverage for various different causes of filesystem cor-
220 ruption. Each mechanism consumes I/O and CPU resources, and that must be
221 weighed against the advantages that it gives us.

222 It is important to note that checksumming does not protect us against corrup-
223 tion or help us in fixing the root cause of the corruption; it only allows us
224 to detect filesystem corruption when it happens. Hence, it is only useful as a
225 warning sign and recovering from data corruption is beyond the scope of this
226 feature.

- 227 • **FAT32** is a very old and simplistic filesystem, and it has no inbuilt facili-
228 ties for checksumming.
- 229 • **Btrfs** maintains a *checksum tree* for all the blocks that it allocates and
230 writes to. Hence, all file data and metadata is checksummed. This is the
231 default behaviour and the current checksum algorithm uses few resources.
232 This method of checksumming can detect all the ways in which corruption
233 can occur to data on the filesystem. See [Checksumming](#) for more detail.
- 234 • **Ext4** maintains checksums for journal data only, no checksumming of file
235 data takes place.

236 **Alignment** The first piece of tuning that a filesystem on flash storage needs,
237 is a proper mapping of the filesystem blocks to the page size of the erase blocks
238 on the flash. This consists of two parts:

- 239 1. Ensuring that the filesystem and storage erase block sizes match using
240 filesystem creation options.
- 241 2. Aligning the block allocations in the filesystem with the storage blocks
242 by using the appropriate offsets while partitioning, or while creating the
243 filesystem.

244 If either of these is not satisfied, each filesystem block write will trigger two or
245 more flash block writes, and reduce the performance as well as reliability of the
246 MMC card.

247 The storage erase block size [can be read](#)¹⁰ from `/proc/mtd` or from U-Boot but

¹⁰<https://bootlin.com/blog/managing-flash-storage-with-linux/>

248 the flash storage can report something different than the real numbers. Linaro-
249 image-tools is now able to generate images with a correct alignment.

250 **Testing** Apertis will add tests to LAVA for testing how FAT32 and Btrfs
251 behave on the i.MX6 under stress, as well as for tuning the above mentioned
252 parameters for reliability and performance.

253 **Root filesystem**

254 The approach will be to mount as many parts of the root filesystem read-only
255 as possible such that the only writes to it would be during updates. This would
256 reduce the chances of catastrophic filesystem corruption in the event of power
257 failure and invalid system file modification by bugs in system or application
258 software. The only partition that is to be mounted writable is the user partition
259 that will be mounted in /home. All the other writable parts of the / filesystem
260 will be backed by tmpfs, located in RAM. We will avoid the lack of space
261 problem by only storing small files in tmpfs or files which don't take space (lock
262 files, socket files). Bigger files such as programs, libraries, configuration files will
263 remain on disk and available read-only.

264 See the *System Updates and Rollback* design for detailed information about the
265 robustness of the update process.

266 **Other filesystems**

267 The system should be able to function even if mounting one or more of the
268 non-essential file systems fails. Even if the system is able to keep running, it
269 would do so with reduced functionality, so some recovery action would need
270 to be taken in order to regain the lost functionality. The system should try
271 to recover automatically as far as possible. In the case of unrecoverable system
272 failure, the user can be instructed at system boot to request technical assistance
273 at a service shop.

274 **Main storage**

275 In case of power loss, the flash media can become corrupted due to how writes
276 are performed. Apertis will be notified via a GPIO signal 100 milliseconds before
277 power is completely lost, in order to give the flash controller time to commit to
278 non-volatile media what is in its cache.

279 Given the short time available and the general slowness of flash devices when
280 writing, we recommend that the signal is handled in the kernel, because
281 userspace will not have enough time to react (depending on the load and the
282 scheduler, it could take from 10 ms to 100 ms for the signal to start being
283 processed by a userspace process). A device driver should be written that,
284 when the GPIO signal is received:

- 285 1. stops flushing dirty pages to the drive,

- 286 2. tells the flash controller to flush its caches to permanent storage, and
- 287 3. starts the shutdown sequence.

288 The device driver will start handling the signal 10-100 μ s after the GPIO is
289 activated. In spite of this, if the device has big caches and is slow to write,
290 corruption of arbitrary data blocks can still happen.

291 In general, drive health data should be monitored so that the user can be notified
292 about disk failures which require a garage visit for hardware replacement.

293 As no more dirty pages will be flushed to the storage device when the GPIO
294 signal is received, the data in the page cache will be lost. To reduce the amount
295 of data that could be lost, eMMC reliable writes can be used, and the page cache
296 configuration can be tuned. But it has to be noted that use of reliable writes
297 and reducing the amount of in-flight data is a trade-off against performance
298 that can be quantified only on the final hardware configuration through direct
299 experimentation.

300 Removable devices

301 External devices that can be removed at any moment are not reliable for writ-
302 ing of critical data. In addition to the problem of corruption of files being
303 written, wear leveling by the controller might corrupt unrelated blocks which
304 might even contain the directory table or the file allocation table, rendering the
305 whole partition unusable.

306 The quality of external storage devices such as flash drives varies greatly, in some
307 cases the device will unexpectedly stop responding to commands, or data will
308 be lost. Applications that write to removable drives must be robust enough to
309 be able to continue in the face of such errors with minimal loss of functionality.

310 As mentioned in **Filesystems**, the safest way to use removable drives is by re-
311 stricting the processes that can write to the drive, and minimizing the time-
312 window for the writes. For that to be practical, there should be a system ser-
313 vice that is the only one allowed to write to removable devices and that would
314 accept requests from applications, remount the device read-write, write the new
315 contents, then remount read-only again.

316 Since, for interoperability reasons, the filesystem used in removable devices is
317 FAT32, in addition to the issues mentioned in this section, the robustness con-
318 siderations that were explained earlier in **Filesystems** also apply.

319 Mitigating the effects of lack of disk space

320 In order to reduce the chances that the system will find itself in a situation
321 where lack of disk space is problematic, it is recommended that available disk
322 space is monitored and applications notified so they can react and modify their
323 behavior accordingly. Applications may chose to delete unused files, delete or
324 reduce cache files or purge old data from their databases.

325 The recommended mechanism for monitoring available disk space is for a dae-
326 mon running in the user session to call *statvfs* (2) periodically on each mount
327 point and notify applications with a D-Bus signal. [Example code](#)¹¹ can be
328 found in the GNOME project, which uses a similar approach (polling every 60
329 seconds).

330 Additionally, so error messages can be stored also in low-space conditions, it
331 is recommended that *journal*d is configured to leave an amount of free space
332 smaller than the reserved blocks of the filesystem that backs the log files. This
333 way, applications will still be able to log messages after applications have con-
334 sumed all the space available to them.

335 In case applications cannot be trusted to properly delete non-essential files, a
336 possibility would be for them to state in their manifest where such files will be
337 stored, so the system can delete them when needed.

338 In order to make sure that malfunctioning applications cannot cause disruption
339 by filling filesystems, it would be required that each application writes to a
340 separate filesystem.

341 It may be worth noting that temporary directories should be emptied on reboot.

342 Resource management

343 The robustness goal of resource management is to prevent one or more applica-
344 tions from disrupting basic functionality due to excessive resource consumption.
345 The basic mechanism for this is to allocate resources in such a way that applica-
346 tions cannot starve services in the base system. This is to be achieved firstly by
347 changing the resource allocation policy to give higher priority to services, and
348 secondly by limiting the maximum amount of resources that an application can
349 consume at a time.

350 Resource limits are capable of helping ensure a process does not render the
351 whole system unresponsive. However, some design decisions also play an im-
352 portant role here. If the user has no way to kill the process that became too
353 slow or unresponsive, the user experience will suffer. The same goes for the
354 case in which an application gets stuck into a failing scenario, such as a web
355 browser automatically loading pages that were open when the browser closed
356 unexpectedly. For these reasons care must be exercised while designing the user
357 interactions for both the system chrome and applications to be sure such cases
358 are addressed.

359 If, despite throttling, some processes still impact the overall user experience
360 negatively because of excessive resource usage, there is the option of identifying
361 those processes and terminating them. Apertis recommends against this because
362 it is very difficult to automatically distinguish between processes that use large

¹¹<http://git.gnome.org/browse/gnome-settings-daemon/tree/plugins/housekeeping/gsd-disk-space.c#n693>

363 amounts of resources due to malfunction or maliciousness and processes that
364 use excessive resources for legitimate purposes. Killing the wrong process may
365 free up resources but is likely to be perceived by the user as a severe defect in
366 the overall user experience.

367 As a general recommendation, for optimal responsiveness, applications should
368 not block the UI thread when calling anything that is not assured to return
369 almost immediately, which includes all local or remote I/O operations. When
370 the potential duration of an operation is a considerable portion of the commonly-
371 considered maximum acceptable response time (100 ms), it should be done
372 asynchronously. GLib contains [asynchronous APIs](#)¹² for I/O in its [file](#)¹³ and
373 [streaming](#)¹⁴ classes.

374 CPU

375 To make sure that important processes have available CPU cycles even when mal-
376 functioning or malicious applications monopolise the CPU, it is recommended to
377 set task scheduler priorities according to the importance of processes. Systemd
378 can do this for services by setting the [CPUSchedulingPriority](#)¹⁵ property in the
379 service unit file of the process. When the process described by the service unit
380 file starts new processes, they stay in the same cgroups and they keep the same
381 CPUSchedulingPriority.

382 At present (Q1 2014), systemd manages the user session on target images but
383 not on the SDK. With the user session managed by systemd, the priorities of ap-
384 plications are no longer set by the application launcher using [sched_setscheduler](#)
385 [\(2\)](#)¹⁶.

386 If there are processes that need real-time capabilities, or that should have very
387 low CPU access, the CPUSchedulingPolicy property can be used to change to
388 the rr (real-time) or idle scheduling policies. Real-time access for a process
389 should be carefully considered and tested because it can have a negative impact
390 on the process and even the entire system.

391 For identifying processes that use an excessive amount of CPU, the [cpuacct](#)¹⁷
392 cgroups controller can be used.

393 Though it is not recommended to automatically terminate local applications
394 with excessive CPU usage, it makes sense for web pages. Web pages are not
395 screened before they execute on the system, hence it is important to ensure that
396 their ability to disrupt system functionality is minimised. For this, WebKit can
397 detect when a block of JavaScript code has been executing for too long, pause

¹²<http://developer.gnome.org/gio/stable/async.html>

¹³http://developer.gnome.org/gio/stable/file_ops.html

¹⁴<http://developer.gnome.org/gio/stable/streaming.html>

¹⁵<http://0pointer.de/public/systemd-man/systemd.exec.html>

¹⁶http://www.kernel.org/doc/man-pages/online/pages/man2/sched_setscheduler.2.html

¹⁷<https://www.kernel.org/doc/Documentation/cgroup-v1/cpuacct.txt>

398 it, and give the embedding application the possibility of canceling the execution
399 of this block of code.

400 I/O

401 Similar to CPU usage, Apertis recommends giving priority to important pro-
402 cesses when there is contention for I/O bandwidth. Apertis recommends that
403 important services have a value for the property `IOSchedulingPriority` lower
404 than 4 (the default). If, for any reason, some applications need priorities other
405 than the default, the application launcher can use the `ioprio_set`¹⁸ (2) syscall
406 to change their priority. When the process described by the service unit file
407 starts new processes, they stay in the same cgroups and they keep the same
408 `IOSchedulingPriority`.

409 Memory

410 Apertis recommends putting a single limit on the amount of memory that the
411 whole application set can allocate so a fair reserve is left for the base software.
412 This limit should be just big enough so that the Apertis instance never reaches
413 the “out of memory”(OOM¹⁹) condition at the system level. For example, if the
414 total of memory available for processes is 1GB, there is no swap, and we know
415 that the services in the base system should need a maximum of 300MB, then
416 all applications should belong to a cgroup that is limited to 700MB of memory.

417 In specific cases, it may make sense to put a different limit on a specific appli-
418 cation, but it can easily be counterproductive and cause a waste of memory.

419 Something else worth doing is to make sure that the OOM killer²⁰ selects ap-
420 plications for killing instead of system services. For this, the `systemd` prop-
421 erty `OOMScoreAdjust` can be used to reduce the chances that a service will be
422 killed. For applications, it is recommended that the application launcher sets
423 its `/proc/<pid>/oom_score_adj` (see [here](#)²¹) to be higher than 0. The ideal
424 value may vary depending upon the importance of each application.

425 With the example setup mentioned before, the OOM killer will terminate the
426 bulkiest application when one of these conditions are met:

- 427 • The total memory taken by applications all together is going to increase
428 over 700MB.
- 429 • The total memory taken by all processes (services plus applications) is
430 going to increase over 1GB.

431 To make better use of the available memory, it's recommended that applications
432 listen to the cgroup notification `memory.usage_in_bytes`²² and when it gets

¹⁸http://www.kernel.org/doc/man-pages/online/pages/man2/ioprio_set.2.html

¹⁹http://en.wikipedia.org/wiki/Out_of_memory

²⁰<http://lwn.net/Articles/317814/>

²¹<http://www.kernel.org/doc/Documentation/filesystems/proc.txt>

²²<http://www.kernel.org/doc/Documentation/cgroup-v1/memory.txt>

close to the limit for applications, start reducing the size of any caches they hold in main memory. It may be good to do this inside the SDK and provide applications with a glib signal that they can listen for.

Network queue

Processes would be classified into cgroup classes such as:

- Interactive (VoIP, internet radio)
- Semi-interactive (web pages, maps)
- Asynchronous (mail, app notifications, etc)
- Bulk (downloads, system updates)

Cgroup controllers are only used for classification of outgoing packets. `NET_PRIO_CGROUP`²³ and `NET_CLS_CGROUP`²⁴ would be used for setting the priority, and for classifying processes into cgroups. By thus tagging packets with the cgroup of applications and services, `tc`²⁵ can be used to set limits to the rate at which processes send packets (<http://lartc.org/howto/>).

Bandwidth rate-limiting would be required to ensure interactive streams do not get starved by lower priority streams.

There is little we can do about latency for applications like VoIP, since even when the bandwidth is sufficient, the bottlenecks are the hardware buffers, queues, and scheduling on various devices outside the control of our system. This is an open problem in networking, and a large part of it is related to `Bufferbloat`²⁶.

Note that there's no robustness issue that can be prevented by limiting the rate at which processes receive incoming packets.

GPU

As explained in the WebGL design, the `GL_EXT_robustness`²⁷ extension provides a mechanism by which the watchdog in the GL implementation can reset the GPU, invalidating all GL contexts and thus stopping all GPU activity.

Unfortunately, this only prevents denial of service (DoS) conditions caused by WebGL, because processes must opt-in to use this extension. Thus, applications may intentionally or unintentionally ignore the extension and continue monopolising the GPU. Within the web browser, scripts that use WebGL and take over the GPU will be interrupted and terminated by the browser.

²³<http://lwn.net/Articles/474695/>

²⁴http://docs.fedoraproject.org/en-US/Fedora/16/html/Resource_Management_Guide/sec-net_cls.html

²⁵<http://lartc.org/manpages/tc.txt>

²⁶<http://en.wikipedia.org/wiki/Bufferbloat>

²⁷http://www.khronos.org/registry/gles/extensions/EXT/EXT_robustness.txt

464 If it runs its own GL implementation, then it could monitor GPU resource
465 usage and reset those contexts that seem to be disrupting the rest of the sys-
466 tem. It could notify processes via the GL_EXT_robustness extension and even
467 terminate them if they ignore the context reset notifications.

468 Accounting

469 Besides setting limits on resources, cgroups also allows to retrieve resource us-
470 age metrics. As examples, for CPU usage the *cpuacct* cgroup controller con-
471 tains the *usage*, *stat* and *usage_percpu* reports; the *memory* controller provides
472 usage data in its *stat* report; the *blkio* controller has *throttle.io_serviced* and
473 *throttle.io_service_bytes*.

474 USB undervoltage

475 In the case that the system momentarily isn't able to power connected USB de-
476 vices such as MP3 players or smartphones due to voltage drops, the system will
477 power off and on again these devices, so that the connection gets reestablished
478 and the user experience gets affected as little as possible.

479 Risks

- 480 • FAT32 is fundamentally unreliable, specially on removable devices.
- 481 • Robustness of flash media varies greatly and the user may not be able
482 to distinguish failures caused by the hardware from failures due to the
483 software.
- 484 • Excessively-low resource limits for applications can lead to resource waste;
485 excessively-high may be less effective in avoiding DoS. There may not exist
486 a good middle point.
- 487 • Heuristics used to determine when to kill a process with excessive resource
488 usage are not perfect and can cause major failure from the user point of
489 view.
- 490 • If Vivante does not implement GL_EXT_robustness properly, web pages
491 could DoS the whole system.
- 492 • Bugs in the OpenGL implementation can lead to instability, data loss and
493 privacy breaches that can be triggered from web pages.
- 494 • If the flash media loses power while a block is open for writing, it is possible
495 that several random blocks elsewhere in the same drive will be corrupted.
496 This can affect other filesystems, even if they are mounted read-only.

497 Design notes

498 The following items have been identified for future investigation and design work
499 later in the project and are thus not addressed in this design:

- 500 • Vulnerability to DoS attacks in D-Bus and proposed solutions.
- 501 • Optimization of the SQLite configuration parameters for the specific
502 filesystems in use in Apertis.

503 No updates as of March 2014.

504 BTRFS Overview

505 The most powerful feature of [Btrfs](#)²⁸ is the fact that all information (data +
506 metadata) is stored in the same basic data structures, and all modification of
507 these data structures is performed in a copy-on-write (CoW) fashion.

508 Since all information on disk is stored using the same type of data structure, this
509 allows metadata and data to share features such as checksumming and striping.

510 This combined with the fact that Btrfs uses CoW while modifying all information,
511 means that in theory, the filesystem is always consistent if the storage
512 device supports “[Force Unit Access](#)”²⁹ correctly. However, in practice, filesystem
513 bugs, a lack of maturity in the code, and other (unforeseen) problems may
514 prevent this.

515 BTRFS robustness supporting features

516 Cheap, fast, and atomic snapshots and rollback

517 All snapshots in Btrfs are CoW copies of the subvolume being snapshotted with
518 an incremented reference count for the blocks. As a result, creating snapshots
519 is very fast, and they take up a negligible amount of space. Just like every
520 other operation, the snapshot is created atomically by the use of transactions
521 and sequenced flushes. Further, all snapshots are actually just subvolumes, and
522 hence can be mounted on their own.

523 Unlike LVM2, which creates snapshots in the form of block devices that can be
524 mounted, Btrfs creates snapshots in the form of subvolumes, which are represented as subdirectories.

526 Even though snapshots are displayed in a subdirectory they are not “owned” by
527 that subvolume. Snapshots and subvolumes are identical in Btrfs, and are first-class citizens with respect to other subvolumes. This means that the default
528 subvolume can be set at any time. The change will be made the next time the
529

²⁸<https://btrfs.wiki.kernel.org/>

²⁹[https://en.wikipedia.org/wiki/Disk_buffer#Force_Unit_Access_\(FUA\)](https://en.wikipedia.org/wiki/Disk_buffer#Force_Unit_Access_(FUA))

530 filesystem is mounted. All the subvolumes, except the top-level subvolume, can
531 also be deleted; irrespective of their relationships with each other.

532 **Repair and recovery**

533 If, for any reason, the root node or the superblock gets corrupted and the filesys-
534 tem cannot be mounted, mounting in recovery mode will make btrfs check the
535 superblock (or alternate superblocks if the superblock is also corrupted) for
536 alternate roots from previous transactions. This is possible because all modifi-
537 cations to the Btrfs trees are done in a CoW manner and existing roots are not
538 deleted. The filesystem stores the last four roots as a backup for the recovery
539 option.

540 **Checksumming**

541 The header of every chunk of space in Btrfs has space for 32-bytes of check-
542 sums of the chunk itself. In addition, there is a checksum tree which maintains
543 checksums for each block of data. Since the data as well as the metadata blocks
544 are referenced in the checksum tree, all information in the filesystem is check-
545 summed.

546 Currently, Btrfs uses the CRC-32 checksum algorithm, but there are plans to
547 upgrade that, and add the option to set the checksum algorithm when the
548 filesystem is created.