



Multiuser

# Contents

2	Terminology and concepts . . . . .	2
3	“user” vs. “uid” . . . . .	2
4	Trusted components . . . . .	3
5	System services . . . . .	3
6	User services . . . . .	3
7	Multi-seat (logind seats) . . . . .	4
8	Fast user switching . . . . .	4
9	Requirements . . . . .	4
10	Distinguishing between privacy levels in user-specific data . . . . .	6
11	Authentication . . . . .	6
12	General use-cases . . . . .	7
13	Existing multi-user models . . . . .	9
14	Switchable profiles without privacy . . . . .	9
15	Typical desktop multi-user . . . . .	10
16	Android 4.2+ . . . . .	12
17	Multi-user support in the Tizen 3 automotive platform . . . . .	13
18	Approach . . . . .	14
19	The principle of least-astonishment . . . . .	15
20	Levels of protection between users . . . . .	15
21	User accounts: representing users within the system . . . . .	16
22	Creating and managing user accounts . . . . .	18
23	Graphical user interface and input . . . . .	20
24	Switching between users . . . . .	23
25	Preserving “core” functionality across user-switching . . . . .	24
26	Returning to previous state . . . . .	26
27	Application ownership and installation . . . . .	27
28	Summary of recommendations . . . . .	27

This document describes how multiple users are expected to use the Apertis system, and works mostly as a guide and recommendations to help designing the system. It is intended to act as an “umbrella” document covering the multi-user topic in general, and will be supplemented by more concrete documents describing particular use-cases and recommendations for how those use-cases can be addressed.

At the time of writing, there is one such document, “Multiuser Design: Transactional Switching”. Please see the [current design documents](#)<sup>1</sup>.

The driving force behind having a multi-user system is to allow customization of the system. A car may have multiple drivers and passengers who would be frustrated by customizations done by each other to the system’s look and feel and even to data such as playlists. Having multiple users allows each to customize their own interface.

<sup>1</sup><https://jwd.pages.apertis.org/apertis-website/concepts/>

42 Depending on OEM and consumer requirements, multi-user systems can poten-  
43 tially also provide personal files and online accounts for each user.

## 44 Terminology and concepts

### 45 “user” vs. “uid”

46 In a Unix system, users are typically identified by a numeric user ID, often  
47 abbreviated “uid”. A uid can represent a person, a system facility, multiple  
48 people, or even an application (as in Android).

49 Because these do not correspond 1:1 in some designs, it is important to be clear  
50 which one is under discussion. In this document, the jargon term *uid* or *user*  
51 *ID* is used to refer to a Unix user identifier, while *user* or *person* is used to refer  
52 to a human using the system.

53 *User account* refers to any abstract representation of the user within the system.  
54 This is most commonly a uid, matching the original Unix design. However,  
55 systems can exist with multiple uids per user account, such as Android, in  
56 which each (user account, app) pair has a uid. Conversely, systems can exist  
57 with multiple user accounts sharing a uid, such as SteamOS (in which one uid  
58 runs the Steam Big Picture UI, and users log in to it with separate Steam  
59 accounts).

60 The canonical form of a Unix uid is numeric, but for ease of reference, a short  
61 lower-case textual *username* may be used to refer to a uid. For example, it is  
62 common to talk about system users named “root” and “backup”, but the real  
63 identities of these users within the system are the corresponding numeric uids  
64 0 and 34; the usernames are merely for convenience and mnemonic value.

### 65 Trusted components

66 A *trusted* component is a component that is technically able to violate the  
67 security model (i.e. it is relied on to enforce a privilege boundary), such that  
68 errors or malicious actions in that component could undermine the security  
69 model. The *trusted computing base* is the set of trusted components. This is  
70 independent of its quality of implementation – it is a property of whether the  
71 component is relied on in practice, and not a property of whether the component  
72 is *trustworthy*, i.e. safe to rely on. For a system to be secure, it is necessary  
73 that all of its trusted components must be trustworthy.

74 One subtlety of Apertis’ app-centric design is that there is a trust boundary  
75 between applications even within the context of one user. As a result, a multi-  
76 user design has two main layers in its security model: system-level security that  
77 protects users from each other, and user-level security that protects a user’s  
78 apps from each other. Where we need to distinguish between those layers, we  
79 will refer to the *TCB for security between users* or the *TCB for security between*  
80 *apps* respectively.

## 81 **System services**

82 A *system service* is a service that, conceptually, runs on behalf of the whole  
83 computer or car, without a division between users. In designs where each user  
84 has a distinct uid, system services run under a system uid, either root (the most  
85 privileged uid) or a special unprivileged uid per service or group of services;  
86 they do not run with the uid of any particular user.

87 This term does not necessarily imply anything about whether the service is  
88 considered to be “part of the operating system”, or whether it is part of a pre-  
89 installed or user-installable application bundle as discussed in the Applications  
90 Design document. However, because system services can accept requests from  
91 multiple users, any system service that will handle users’ private data must be  
92 trusted to impose a privilege boundary.

93 Examples of system services commonly present in Linux systems include Conn-  
94 Man, NetworkManager, BlueZ, udisks and the D-Bus system bus.

## 95 **User services**

96 A *user service* is a service that runs on behalf of a particular user. In designs  
97 where each user has a distinct uid, each user’s user services typically run under  
98 that same uid; in designs like SteamOS where all users share a single generic  
99 uid representing “all users”, user services would typically share that same uid.

100 Examples of user services commonly present in Linux systems include dconf,  
101 gvfs, Tracker, Tumbler and the D-Bus session bus.

102 Identifying a process as a user service is independent of whether it is treated  
103 as part of the Apertis platform and independent of any particular application  
104 (such as the user services mentioned above), or treated as part of an application  
105 bundle (“agents” associated with apps).

## 106 **Multi-seat (logind seats)**

107 In the context of a multi-user system, a *seat* is a collection of display and input  
108 devices, optionally linked to other devices such as a USB socket or optical drive,  
109 intended to be used by one user at a time. Typical PCs only offer one seat, but a  
110 second graphics adapter, often connected via USB, can be used to add additional  
111 seats (a *multi-seat* system).

112 This jargon term is commonly used in Linux system services such as systemd-  
113 logind, the older ConsoleKit, and GDM. In the context of a car, it should be  
114 noted that it does not necessarily correspond precisely to the car’s seats: for  
115 instance, in the common layout that places a single “head unit” touchscreen  
116 between the driver and front passenger, that touchscreen and any USB sockets  
117 adjacent to it would be treated as a single seat. If, for example, additional  
118 touchscreens were added behind the front seats for use by rear passengers, that  
119 would be a multi-seat system with 3 seats (front, rear left, rear right).

120 Apertis uses systemd-logind as a core system service, so where disambiguation  
121 is needed, we will refer to this as a *logind seat*.

## 122 **Fast user switching**

123 Many operating systems have the concept of *fast user switching*, which is de-  
124 scribed in “**Fast user switching**”: **switching user without logging out**. Following  
125 common usage, this document reserves the term “fast user switching” to refer  
126 to that particular multi-user model, even if some other model might be equally  
127 fast or faster in practice.

## 128 **Requirements**

129 Apertis is currently designed as a single-user system. There is one GUI session  
130 with full access to all preferences, apps and data, and a set of apps and user  
131 services with varying levels of sandboxing and privilege separation from each  
132 other within that session, running on top of system services whose privileges  
133 vary. The high-level requirement for this document is that this should be ex-  
134 panded to support multiple GUI users, each with their own private data and  
135 user services, running on top of similar system services.

136 This section contains a list of general requirements applicable to many multi-  
137 user systems.

- 138 • Multiple users should be able to use the system. Depending on the specific  
139 set of requirements, this could involve concurrent use, or one user at a time.
- 140 • When the user logs in to a newly started system, they should find the same  
141 applications they had left open last time they shut down the system, and  
142 in the same state. See **Returning to previous state** for discussion of this  
143 topic.
- 144 • Some data is private to each user. Depending on the specific set of re-  
145 quirements, this could include:
  - 146 – Settings
  - 147 – Address book
  - 148 – Browser history
  - 149 – Application icons
  - 150 – Arrangement of icons in the app launcher
  - 151 – Account data for web services
  - 152 – Playlists
- 153 • Some data is shared between users. Depending on the specific set of  
154 requirements, this could include:

- 155           – Applications (from the store)
- 156           – Media library (music, videos)
- 157       • Depending on the specific set of requirements, switching users at runtime
- 158           could be supported. Where it exists, this shall be performed with a smooth
- 159           transition, with no visual flickering. User switching should not take more
- 160           than 5 seconds. See [Switching between users](#) for discussion of this topic.
- 161       • A subset of features are considered to be core functionality, and must
- 162           not be disturbed by switching between users: they must remain available
- 163           before, during and after any transition between users. The set of core
- 164           functionality could vary by device; in this document we mainly use music
- 165           playing and navigation as examples of this category. See [Preserving “core”](#)
- 166           [functionality across user-switching](#) for further discussion of this topic.
- 167       • The subset of features that are not disturbed while switching between
- 168           users must not be limited to functionality that is considered to be “part of
- 169           the operating system”. For example, it should be possible to place a user-
- 170           installable player for a third-party music streaming service such as Spotify
- 171           or last.fm in this category. Again, see [Preserving “core” functionality](#)
- 172           [across user-switching](#).
- 173       • Depending on the specific set of requirements, peripheral hardware devices
- 174           such as USB storage devices and paired Bluetooth devices could either
- 175           be shared across the entire system, or specific to a user. If they are
- 176           shared, then they must be accessible to all users, with all users able to
- 177           unmount/eject them.
- 178       • The authentication and user-switching user interface should not distract
- 179           the driver more than is necessary; for instance, they should not ask security
- 180           or authentication questions unless a decision is strictly required.
- 181       • The user privileges of the system should be visually obvious: if users have
- 182           selected different personalizations such as colour schemes or themes, then
- 183           the display should use a particular user’s theme whenever it is acting on
- 184           behalf of that user, and at no other time. This limits the risk that users
- 185           will encounter undesired privacy consequences resulting from misunder-
- 186           standing the system’s privacy model.

## 187 **Distinguishing between privacy levels in user-specific data**

188 There are several possible categories of user-specific data.

189 Some user-specific data is private. For instance this might include email, brows-  
 190 ing history, social media feeds. (Alice should not be able to read Bob’s email,  
 191 history, social media feeds and so on unless Bob has allowed it.) Meanwhile,  
 192 some user-specific data is sensitive because it allows acting on someone else’s  
 193 behalf. (If Alice is logged-in to Amazon, Bob should not be able to buy things

194 using her account.) Private and sensitive data are interchangeable from a sys-  
195 tems perspective: they must be accessible by that user, and only by that user.

196 However, some data is only user-specific for convenience or organization; it isn't  
197 important whether other users are able to read it, as long as it doesn't make  
198 their own actions less convenient.

199 For instance, the set of apps that are visible in menus might be one example of  
200 user-specific data that does not necessarily need to be treated as private. If Alice  
201 has installed apps for social media networks that Bob doesn't use, they shouldn't  
202 appear in Bob's menus — but if Bob specifically looks for them, perhaps in an  
203 Android-Settings-like “storage usage” view, it might be considered acceptable  
204 that he can see what Alice installed.

205 Another possibility for sharing data is that playlists within a shared media  
206 library could appear as an unobtrusive “Bob's playlists” folder in other users'  
207 menus, if desired.

208 As discussed in [Levels of protection between users](#), the level of privacy and  
209 integrity protection between users can vary according to OEM and consumer  
210 requirements; this could influence how user-specific data is categorized.

## 211 Authentication

212 We assume that the HMI provides a way for users to identify and authenticate  
213 themselves to a trusted HMI component, for instance by:

- 214 • presence of a unique physical key
- 215 • presence of a personal item such as a phone with Near-Field Communica-  
216 tion support
- 217 • a password or lock-screen gesture
- 218 • face or fingerprint recognition
- 219 • simply selecting a user from a menu (choice of user, but no meaningful  
220 authentication, similar to one of the cases described in [Switchable profiles](#)  
221 [without privacy](#))

222 The exact authentication mechanism depends on manufacturer and user require-  
223 ments, and is outside the scope of this document: this document only assumes  
224 that an identification/authentication mechanism exists as part of the operating  
225 system, and does not rely on specific properties of that mechanism.

## 226 General use-cases

227 While this document does not go into the specifics of more elaborate use-cases,  
228 there are a few simpler use-cases which should be considered by any concrete  
229 multi-user design within the framework established by this document. In some

230 cases these use-cases could be considered and rejected, if a particular design's  
231 requirements put them out of scope.

## 232 **First use**

233 Alice uses the car for the first time. The system recognises that she has not  
234 used it previously and so there is no saved state.

235 **a. First use:** The system starts in some default state, for instance at a main  
236 menu or with a default application such as a media player running.

## 237 **Individual use: preferences and state restored**

238 Alice and Bob share a car, and have separate keys. Alice has configured the  
239 display for a red UI theme; she uses the car on Monday, listens to a podcast while  
240 she drives, and has the email app open in the background. Bob has configured  
241 the UI for a blue theme. He uses the car on Tuesday, and reads the BBC News  
242 website in the browser app while stopped at motorway services.

243 **a. Last-used mode:** The next time Alice starts the car and authenticates  
244 as herself (see [Authentication](#)), the podcast and email apps should resume in  
245 the same state they were in when she shut the system down on Monday, and  
246 the HMI configuration should reflect her preferences (the red theme should be  
247 used, etc.). Similarly, the next time Bob authenticates as himself, the BBC  
248 News website should be displayed in the browser app as it was when he shut  
249 the system down on Tuesday, and the blue theme should be used.

250 **b. Privacy between non-concurrent users:** If the system is configured  
251 to provide protection between users, then Alice's private data should not be  
252 available to Bob and vice versa. For instance, Bob's web browsing history and  
253 social media accounts should not be available when Alice starts the web browser,  
254 even if Alice deliberately looks for them.

## 255 **User switching**

256 **a. User switching:** Bob is currently using the HMI to read Twitter, and Alice  
257 wants to check her email. Neither is currently driving. Alice should be able  
258 to authenticate in some way (see [Authentication](#)), switching the HMI to have  
259 Alice as its current user. When she has finished, Bob should be able to switch  
260 the HMI back so he is the current user again, and continue to read Twitter.

261 **b. Privacy during user switching:** after switching from Bob's user account  
262 to Alice's, Bob should be able to go away, knowing that Alice cannot access  
263 his Twitter feed. When Alice has finished and hands back control to Bob, she  
264 should be able to know that Bob cannot access her email.

265 In existing multi-user systems like those described in section 4, this  
266 is typically implemented by leaving Bob's user account in a "locked"



267 state after he transfers control to Alice, and vice versa, requiring  
268 re-authentication before resuming use.

## 269 **Guest mode**

270 Greg, a guest, is in Diana's car.

271 **a. Unauthenticated guest session:** If Diana has enabled it (or if it is enabled  
272 by default and Diana has not disabled it), Greg should be able to start a guest  
273 session that can access public information and the Web, play music from the  
274 car's music library, etc. without authentication.

275 **b. Owner's privacy:** Greg should not be able to access Diana's private data  
276 (or the private data of any other user of the system).

277 **c. Guest's privacy:** Greg's browser history, Facebook authentication token,  
278 etc. should not be available to subsequent guests. For instance, the system  
279 could temporarily allocate space for Greg's user-specific data, then discard it  
280 and terminate all guest processes as soon as Greg logs out, returning to default  
281 settings for the next guest.

282 **d. Guest is restricted:** Greg should not be able to add or delete music, install  
283 or remove apps, or similar actions.

## 284 **Borrowing the car**

285 Diana lends her car to David, giving him her key.

286 If the system is configured to consider a key as sufficient authentication for a  
287 user, then it cannot be expected to protect Diana from malicious action by  
288 David. However, if the system is configured to require secondary authentication  
289 such as a password, PIN or lock-screen swipe pattern, then David will not be  
290 able to use Diana's account.

291 **a. Can create a new account:** Even though David and Diana are using  
292 the same key, David should be able to create a new account that saves his  
293 preferences, and switch to it.

## 294 **Existing multi-user models**

295 This chapter describes the conceptual model, user experience and design ele-  
296 ments used in various non-Apertis operating systems' support for multiple users,  
297 because it might be useful input for decision-making. Where available, it also  
298 provides some details of the implementations of features that seem particularly  
299 interesting or relevant.

## 300 **Switchable profiles without privacy**

301 The simplest multi-user model can be found in platforms such as Windows 95  
302 and the Sony PlayStation 3. In these systems, certain settings and other pieces

303 of application data (such as documents and saved games) are stored separately  
304 for each user, but there is no privacy or protection between users: each user can  
305 easily access other users' accounts.

306 One variant of this is where no authentication is required to access a different  
307 account, as on the PlayStation 3: a user selects their name from a list, and  
308 there is nothing preventing them from selecting a different user's name instead.  
309 Similarly, an unauthorized user can identify themselves as any authorized user  
310 and gain access.

311 Another variant of this is where there is meaningful authentication (e.g. a login  
312 step with a password), but authenticating as *any* user is sufficient to access *all*  
313 users' private files. For instance, Windows 95 offered login authentication, but  
314 did not support filesystems with user-level permissions. As a result, unautho-  
315 rized users were prevented in principle (in practice, the login step was easily  
316 circumvented), but each authorized user had the technical capability to read  
317 and write any other user's files by navigating to the appropriate directory.

318 Both variants of this model are simple to implement, and provides straightfor-  
319 ward semantics. Their disadvantage is that they do not meet typical privacy  
320 expectations for a modern operating system: users can impersonate one another,  
321 read each other's private files, and even alter each other's private files. As such,  
322 it is only suitable for an environment in which every user of the system fully  
323 trusts every other user of the system (and, for the first variant, everyone with  
324 physical access to the system).

325 We anticipate that these simple use-cases will be appropriate for some, but not  
326 all, Apertis systems: for example, they might be appropriate for a family car  
327 where the installed apps do not handle particularly sensitive information. In  
328 other Apertis systems, stronger privacy/protection between users is likely to be  
329 required.

### 330 **Typical desktop multi-user**

331 Many modern desktop/laptop operating systems (such as the Windows NT  
332 series, Mac OS X, and various open source desktop environments on Linux and  
333 BSD platforms) have a similar model for how multiple users are handled. Apertis  
334 shares many software components with the GNOME 3 desktop environment (as  
335 used in, for instance, Debian GNU/Linux and Fedora Linux), so we will use  
336 GNOME on Linux as our primary example of this type of environment.

337 On Unix-derived systems such as Linux and Mac OS X, each user account is  
338 typically represented by one Unix uid, corresponding to their intended use in  
339 all Unix systems.

### 340 **Basic multi-user: log out, log in as another user**

341 The most basic form of multi-user support is considerably older than graphical  
342 user interfaces, and is implemented in most current desktop/laptop operating

343 systems. The system boots to a login prompt at which the user can choose their  
344 user account (for instance by choosing from a list or by typing its name), and  
345 authenticate in some way (typically with a password, but many authentication  
346 mechanisms are possible).

347 Each user has their own set of data files and configuration. To provide privacy  
348 between user accounts, the system tracks the ownership of user files, and either  
349 denies access to other users' files by default, or can be configured to do so.

350 To switch between users, the first user must log out, ending their session; this  
351 typically also terminates most or all of their user services. Ending their session  
352 presents another login prompt, at which the second user can log in.

353 In a typical implementation on Linux systems with the X11 windowing system,  
354 a system service (a “display manager”, such as GNOME’s GDM) starts an X  
355 display and uses it to show the graphical login prompt. When the first user  
356 logs in, their uid is granted access to the X display, which is taken over by  
357 their session. At the end of their session, the display manager terminates the X  
358 server, and starts a new X server for the next login prompt.

359 Systems which offer this model can easily support the simpler models from  
360 **Switchable profiles without privacy** as trivial cases of this model: they can  
361 implement the PlayStation 3-like model by omitting the authentication step  
362 after choosing a user, or the Windows 95-like model by giving each authorized  
363 user access permissions for other users' files.

#### 364 **“Fast user switching”: switching user without logging out**

365 A refinement of the above model for systems with enough memory is to offer  
366 more than one parallel login session, with one active login session and any  
367 number of inactive sessions. This is commonly referred to as *fast user switching*.

368 Again, most current desktop/laptop operating systems offer this in some form.  
369 The first user chooses a “Switch User...” option from a menu; this optionally  
370 locks the first user’s session (for instance by locking their screensaver), and  
371 switches to a login prompt at which the second user can log in. To switch back,  
372 the second user uses “Switch User...” to access another login prompt, at which  
373 a third user can log in, and so on. Several users can share the system, with  
374 up to one active session and any number of inactive sessions (limited by system  
375 RAM, and optionally an arbitrary limit on the number of users).

376 If the user logging in at the login prompt already has a login session, then the  
377 system detects that, and instead of starting a new session, it switches back to  
378 the existing session, automatically unlocking the screensaver if required. When  
379 a user logs out, their session is replaced by a login prompt at which any user  
380 can log in.

381 Designers typically treat this model as a superset of the simpler model in **Basic**  
382 **multi-user: log out, log in as another user**: in practice, implementations of “fast

383 user switching” also offer the non-concurrent log-out/log-in arrangement as a  
384 trivial case. Similarly, as in **Basic multi-user: log out, log in as another user**,  
385 implementations of this model can easily support the models from **Switchable**  
386 **profiles without privacy** as trivial cases.

387 In GNOME’s GDM display manager, the first session takes over the X server  
388 originally used for the login prompt, the same as in **Basic multi-user: log out,**  
389 **log in as another user**:: this runs on a Linux virtual console, traditionally tty7.  
390 The “Switch User...” option causes the display manager to run a new X server  
391 on a different virtual console, typically tty8, and switch to it; the second user’s  
392 session takes over that X server, and so on, allocating a new virtual console  
393 and running a new X server each time. If a user logs out, the display manager  
394 remains on the same virtual console, but runs a new X server for the login  
395 prompt. If the user logging in at the login prompt already has a login session,  
396 instead of taking over that X server for a new session, the display manager  
397 switches to the appropriate virtual console for the existing session. The X  
398 server with the login prompt remains in the background, and is re-used the  
399 next time a login prompt is required, instead of starting a new X server: for  
400 example, a system where three users Alice, Bob and Chris repeatedly switch  
401 between their accounts would reach a “steady state” with four X servers on four  
402 virtual consoles (corresponding to Alice, Bob, Chris, and the login prompt).

403 Once two or more users have logged in, this model provides very rapid switching  
404 between them: none of their applications or user services need to be terminated  
405 or restarted. It also eliminates any loss of transient “context” such as notifica-  
406 tions or window positions, without needing to implement state-saving. However,  
407 it uses a significant amount of memory: because inactive users’ applications are  
408 not terminated, two alternating users could need up to twice as much memory  
409 as a single user. Similarly, because the inactive users’ applications are not ter-  
410 minated or paused, merely disconnected from input and display devices, they  
411 can continue to consume other resources, such as CPU time and network band-  
412 width: a misbehaving application in Alice’s session can cause Bob’s session to  
413 appear slow.

#### 414 **Multi-user desktops with multi-seat support**

415 Some systems, in particular the systemd-logind component used in Apertis, can  
416 be used to extend the model in **Basic multi-user: log out, log in as another**  
417 **user** by offering several so-called “seats” as defined in **Multi-seat logind seats**.  
418 A logind seat is a collection of display and input devices intended to be used by  
419 a single user, offering the equivalent of section **Basic multi-user: log out, log in**  
420 **as another user** independently on each logind seat. Similarly, a system can offer  
421 “fast user switching” ( **“Fast user switching”: switching user without logging**  
422 **out**) on some or all of the available logind seats.

423 GNOME’s GDM display manager switches between virtual consoles on the first  
424 logind seat, in exactly the same way as section **“Fast user switching”: switching**

425 **user without logging out.** On the second and subsequent logind seats, it behaves  
426 as described in **Basic multi-user: log out, log in as another user**, with this logind  
427 seat's X server remaining visible regardless of the current virtual console, and  
428 does not offer "fast user switching".

## 429 **Android 4.2+**

430 Recent versions of Android have gained multi-user support, initially for tablets  
431 only, then extended to phones in Android 5.

432 When first started, **Android 4.2<sup>2</sup>** shows a prompt for setting up the first user  
433 account. The first user account is special in that it is considered the administra-  
434 tor for the device, and can thus create, remove and assign permissions to other  
435 users.

436 Android uses separate Unix user account IDs (uids) for separating applications  
437 from each other, so any communication or sharing between applications was  
438 already mediated by the Linux kernel and other trusted parts of the Android  
439 system software. The multi-user design simply allocates a block of uids to each  
440 user, one uid per (user, application) pair: for example, the first user (user  
441 number 0) might receive uids u0a123 and u0a45 for two of their apps, and user  
442 number 1 might receive uids that include u1a67.

443 Because applications are already isolated from one another by their differing  
444 uids, all interaction between apps is mediated by trusted processes, so those  
445 trusted processes were adapted to take the user into account when deciding  
446 permissions. Similarly, because apps conventionally use Android-specific APIs  
447 to access user data, adapting those Android-specific APIs to take the user into  
448 account is straightforward: an application making an API call that previously  
449 listed *all* online service accounts will now only be told about the appropriate  
450 user's online service accounts.

451 Authentication is through the usual means used by Android: each user gets their  
452 custom lock screen and, depending on that user's settings, types in a PIN, a  
453 password or a pattern connecting dots in a grid for logging in. Icons representing  
454 all users are shown in the current user's lock screen, so user switching is a matter  
455 of locking the screen (which can be done through the 'quick settings' menu,  
456 available in the status bar) and tapping the desired user.

457 From a user interface perspective, this resembles "**Fast user switching**": **switch-**  
458 **ing user without logging out** on typical desktop operating systems. However, as  
459 an implementation detail, each user's apps are terminated when user switching  
460 occurs, so the actual implementation is closer to the "log out / log back in"  
461 model (section **Basic multi-user: log out, log in as another user**).

462 Some settings are global to the device, including Wi-Fi networks. All users  
463 can change these settings, apparently, and those changes will affect every other

---

<sup>2</sup><http://developer.android.com/about/versions/jelly-bean.html#android-42>

464 user. User settings and data are kept separate from each other's. The list of  
465 applications in the user's launcher is separate for each user, but application files  
466 are only downloaded the first time a user asks that application to be installed,  
467 to save space.

468 Because Android provides custom API for everything the application does, the  
469 storage and reading of data and settings for each user is done automatically by  
470 their APIs. That means applications did not have to be modified for supporting  
471 multi-user: the fact that they already use Android APIs to obtain directory  
472 paths and save files ensures that they are saved to the proper place.

### 473 **Multi-user support in the Tizen 3 automotive platform**

474 The multi-user architecture designed for Tizen 3 in an automotive environment  
475 was [presented](#)<sup>3</sup> at FOSDEM 2015.

476 At a conceptual level, Tizen applications can either be installed system-wide or  
477 for a particular user. Guest users can only use system-wide applications; it was  
478 not clear from the presentation whether only preinstalled applications can be  
479 system-wide, or whether separate installable applications can also be installed  
480 system-wide. If installed for a particular user, the application's files are copied  
481 into that user's home directory, contrasting with the centralized app storage  
482 used "behind the scenes" in this design document and in Android.

483 The Tizen model is designed for a "multi-seat" environment as described in  
484 [Multi-user desktops with multi-seat support](#), where several sets of grouped de-  
485 vices (a display, its attached touchscreen input device, and perhaps USB sockets  
486 and/or a headphone jack located near that display) are all attached to the same  
487 computer as peripherals; this is an attractive model if the system is powerful  
488 enough to provide acceptable performance on all seats, but comes with higher  
489 performance requirements than some of the potential classes of requirements  
490 addressed by this document. In particular, there is a focus on the ability to  
491 move concurrent applications seamlessly from one screen to another, following  
492 a user who moves from one seat to another.

493 In the Tizen model, all users share a single compositor, which manages all  
494 seats' displays and input devices, resulting in the compositor being required to  
495 act as part of the TCB for security between users (see [Trusted components](#)). As  
496 discussed further in [Graphical user interface and input](#), we do not recommend  
497 this approach while using X11 for GUI services.

498 There is a single privileged user in the Tizen system, and only that user can  
499 configure certain shared resources such as wireless networking and Bluetooth.  
500 This seems an unnecessarily limiting model for a car that might be shared  
501 between two or more primary drivers, for example in a family. It is intended  
502 that this user will eventually be able to launch applications on seats that are  
503 currently in use by other users.

---

<sup>3</sup>[https://fosdem.org/2015/schedule/event/embedded\\_multiuser/](https://fosdem.org/2015/schedule/event/embedded_multiuser/)

504 The API model in Tizen appears to involve system services such as the media  
505 server and thumbnail generation service not only acting on behalf of users to  
506 fulfill requests, but running as ‘root’ so that the same application can write  
507 directly into multiple users’ home directories. We recommend avoiding this  
508 practice: it puts all of those services into the TCB for each layer of the secu-  
509 rity model (security between users, security between apps and security between  
510 system services), greatly increasing the amount of security-sensitive code in the  
511 system and the potential impact of a bug or security flaw.

512 The presentation mentioned adding the user ID as an explicit parameter in IPC  
513 (inter-process communication) calls from applications to system services so that  
514 the system service will act on behalf of the appropriate user. This could be made  
515 to work securely by verifying that the actual user ID matches the one in the IPC  
516 call, but is a potentially dangerous approach: if a naive implementation trusts  
517 the given parameter and does not verify it, a malicious application could easily  
518 subvert that implementation. We recommend avoiding “user ID” parameters  
519 in APIs: if the service can determine the user ID in a secure way, then the  
520 parameter is unnecessary, and if it cannot, this approach brings the calling  
521 application into the TCB for security between users (with the practical result  
522 that all or nearly all applications would end up in the TCB, greatly increasing  
523 the system’s attack surface).

## 524 **Approach**

525 Because this document does not define precise requirements or use-cases for  
526 the system, this section outlines multiple possible approaches to several design  
527 questions. The choice between these approaches must be made based on concrete  
528 requirements.

## 529 **The principle of least-astonishment**

530 One valuable general design principle is that, when a user carries out an action,  
531 it should be easy to predict the outcome. In the context of a multi-user system,  
532 this implies various more concrete principles, such

- 533 • sharing should not occur when a user would not expect it to; this “over-  
534 sharing” is likely to lead to users distrusting the system and being unwilling  
535 to store private data in it, even if that would be advantageous
- 536 • sharing should occur when a user would expect it to; if it does not, users  
537 will be inconvenienced by having to copy data manually between different  
538 contexts
- 539 • performing a similar action in different contexts should have a similar  
540 result

## 541 **Levels of protection between users**

542 There is a spectrum of possible sets of requirements for privacy and integrity  
543 protection between users: a strongly protected model similar to the one detailed  
544 in section **Typical desktop multi-user**, a model with no protection at all as  
545 described in **Switchable profiles without privacy**, or anything in between (e.g.  
546 with protection between users in general, but certain categories of data explicitly  
547 shared).

548 The desired level of protection depends on the user, but we could also decide  
549 that Apertis will only support a subset of the possible range, and an OEM could  
550 decide that they will only support a subset of the range allowed by Apertis.

551 In use-cases that involve differently-privileged users, the desired level of protec-  
552 tion might vary between users within a system: for instance, the main users of  
553 a car might opt for a setup in which switching from one main user to another  
554 does not require authentication, but switching from a “guest” user to a main  
555 user does.

556 For each set of requirements, we aim to minimize the “friction” in switching  
557 between users, subject to whatever minimum is imposed by the requirements –  
558 stronger privacy and integrity protection comes with a higher minimum “fric-  
559 tion”. For example, if users are to be protected from each other, then switching  
560 between users must include an authentication step, whereas if there is no ef-  
561 fective protection (privilege boundary) between users, switching between users  
562 merely requires choosing the desired user account.

563 As a general design principle, design documents for concrete use cases should ad-  
564 dress the “strongest” supported protection between users, because that imposes  
565 the most difficult privacy/integrity requirements. Secondly, they should con-  
566 sider the “weakest” supported protection between users, because that imposes  
567 the most general sharing requirements: ideally, this is just a trivial case of the  
568 high-privacy version, with some of the “pain points” omitted, but it does in-  
569 troduce new requirements for the ability to pass data between users. All other  
570 levels of privacy/integrity protection can be represented as somewhere between  
571 those extremes.

572 As a compromise plan if we find situations that cannot be solved in a higher-  
573 privacy model, it is possible to relax our requirements to declare the highest-  
574 privacy use cases to be out of scope.

## 575 **User accounts: representing users within the system**

576 There are three possible approaches to representing users in a Linux system.

## 577 **Sharing one uid between all users**

578 In this approach, all user applications and user services run under the same  
579 uid. The system defines its own proprietary “user account” concept, and all



580 components that access user-specific data must ensure that they access the  
581 correct user's data, disallowing access to other users' data if appropriate.

582 This has the potential to make transitions between users very easy: the "cur-  
583 rent user" is simply a variable within each application or service. However,  
584 it places a great deal of trust on each of these components, including every  
585 third-party (user-installable) application that accesses user-specific data. If the  
586 system's security model is that users can be protected from each other, then in  
587 effect, all of these components are included in the trusted computing base; if  
588 the requirements do not include protection between users, then distinguishing  
589 between users is not required for security, but is still required for correctness. In  
590 practice, we anticipate that not every component would discriminate between  
591 users correctly.

592 This approach also has practical problems for the re-use of existing open source  
593 components, which assume the traditional use of one uid per user. Having to  
594 modify all of these components, with a complex change that is unlikely to be ac-  
595 cepted by their upstream developers, would significantly reduce the competitive  
596 advantage derived from their use.

597 As a result of these disadvantages, we do not recommend this approach for  
598 Apertis. It would only be viable if all of the following are true:

- 599     • users are not protected from each other, and this will not change in future  
600       development
- 601     • user-specific data is minimal, only needs to be accessed via Apertis-specific  
602       APIs, and this will not change in future development
- 603     • it is not considered to be a significant problem if third-party applications  
604       and services do not consistently distinguish between users, and this will  
605       not change in future development

606 An additional consideration for this approach is that it potentially alters a large  
607 number of interfaces (such as D-Bus method calls) to have a parameter for the  
608 user account to be affected. If changing requirements result in switching to  
609 the "one uid per user" or "many uids per user" models in future, such that the  
610 correct user account is implicit in the uid, then this vestigial parameter will  
611 remain in the interface, making the interface more complex than is required.

612 If the form of the additional parameter resembles the numeric or string form of  
613 a uid, then this could even lead to security issues, for instance if a component  
614 trusts the explicit user-account parameter and ignores the actual uid.

615 If this approach is taken, then we recommend reducing the confusion caused  
616 by naming the additional parameter something more similar to "profile" than  
617 "user". If the system is later extended to have one uid per user, rendering the  
618 parameter vestigial, we recommend giving it a neutral, constant value that does  
619 not match any user account name, such as "default".

## 620 **One uid per user**

621 The traditional Unix design which motivated the uid concept is that each user  
622 account is represented by one numeric uid.

623 Because each process (i.e. each application or service) starts with a particular  
624 uid, and processes without administrative privileges cannot change their uid  
625 while running, this approach requires that user-switching involves starting new  
626 processes for the new user.

627 The major advantage of this approach is that it is how the existing components  
628 in the system, including the Linux kernel, are designed to operate. In particular,  
629 the Linux kernel provides privacy and integrity protection between uids.

630 We recommend this approach for Apertis.

## 631 **Multiple uids per user**

632 Android uses a design involving multiple uids per user, one per app or set of  
633 related apps, as described in [Android 4.2+](#). This allows the Linux kernel's  
634 privacy and integrity features to be used to protect apps from other apps, even  
635 within a user session. However, in Apertis, this advantage is redundant, since  
636 we already use a different kernel feature (AppArmor) to provide privacy and  
637 integrity protection between apps.

638 The major disadvantage of this approach is that it requires every interaction  
639 between dissimilar apps to be mediated by a system-level component. Within  
640 the context of Android, this is not a problem, since Android applications and  
641 services are expected to use Android-specific APIs in any case. However, Apertis  
642 re-uses existing open source components where appropriate; these components  
643 would have to be modified to cope with crossing privilege boundaries when they  
644 communicate with different uids, which, as in the “one shared uid” approach,  
645 would reduce the value of re-using these components.

646 We do not recommend this approach for Apertis.

## 647 **Creating and managing user accounts**

648 Based on the description of desired use case scenarios, Collabora understands  
649 the main means of identifying and authenticating a user will be through their  
650 own personal car key. This means a key with an unique ID will have to be issued  
651 to each user of the car.

652 Because most cars require the key to remain inserted while the car is in use, if  
653 runtime user-switching is required, a secondary form of authentication is likely  
654 to be required. This could be done via a password (or equivalent, such as a PIN  
655 or touchscreen swipe pattern), via biometrics such as fingerprint, face or voice  
656 recognition, or by verifying possession of a near-field communication device such  
657 as a mobile phone.

658 As previously noted, depending on manufacturer and consumer requirements,  
659 there is the possibility of simpler authentication schemes for less privacy-  
660 conscious users; for instance, a manufacturer or consumer could choose to relax  
661 the security model to one where a car key is sufficient to authenticate as any  
662 registered user selected from a menu.

663 A registration process will be required, to associate authentication tokens with  
664 user accounts: one way this could work is detailed in this section.

### 665 **Registering the users**

666 After the car has been bought, the owner is provided with a number of keys,  
667 one of each is handed to each user. Each user in turn will follow the following  
668 procedure:

- 669 1. User inserts the key and starts the vehicle
- 670 2. The Apertis system starts up and recognizes that the key is unregistered
- 671 3. A wizard is displayed to register the new user
- 672 4. The user enters whatever information is needed to set up their user ac-  
673 count, such as their name
- 674 5. The user is given the option of registering a password or other authen-  
675 tication tokens to be used for keyless authentication (for user switching,  
676 mainly)
- 677 6. Alternatively the wizard can continue from here on to register email and  
678 web accounts the user may be interested in

679 In case there are more users than keys available, new keys will need to be  
680 acquired.

### 681 **The first user to be registered is special**

682 It's important that at least one user be able to perform administrative tasks,  
683 such as wiping out all of the data, removing users, and so on. One practical  
684 solution to this is that the first user to be registered is considered special and  
685 be able to perform these tasks and is also able to give these privileges to other  
686 users as they see fit, so that more users would be able to perform administrative  
687 tasks.

688 One analogy used in the security literature is that the system “imprints” on  
689 the first user seen, in the same way that a duckling imprints on its parent. A  
690 refinement of this model is that deleting all users resets the system to a state  
691 in which the next user created will be privileged, the so-called “[resurrecting](https://www.cl.cam.ac.uk/~fms27/duckling/)  
692 [duckling](https://www.cl.cam.ac.uk/~fms27/duckling/)<sup>4</sup>” model.

---

<sup>4</sup><https://www.cl.cam.ac.uk/~fms27/duckling/>

693 Frank Stajano and Ross Anderson. *The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks*. In B. Christianson, B.  
694 Crispo and M. Roe (Eds.). *Security Protocols, 7th International Workshop Proceedings*, Lecture Notes in Computer Science, 1999.  
696

## 697 **Premium segment considerations**

698 Markets which are targeted by Apertis system will be segmented. Upper seg-  
699 ment cars do not necessarily require the key to be kept in the ignition while the  
700 car is on. For those kinds of systems, the system could use key proximity as  
701 authentication factor, so it would allow login for all users whose keys are in the  
702 car.

## 703 **Possible trade-offs and their consequences**

704 As discussed previously, the authentication system is one of the problematic  
705 areas that might need trade-offs. The main means of authentication being con-  
706 sidered at the moment is the car key owned by a user.

707 The fact that most cars require the key to remain in the ignition barrel to  
708 keep the car working makes it impossible for a different user to log in. This  
709 indicates the need for an alternate authentication method, such as a password,  
710 which would probably need to be registered with the system when the users first  
711 register themselves by using the key.

712 Should that solution be deemed not good enough, then disallowing user switch-  
713 ing at runtime will be considered, requiring the car to be turned off and on with  
714 a different key for logging in with another user.

## 715 **Graphical user interface and input**

716 As of May 2015, the graphics layer of Apertis is based on the Mutter window  
717 manager/compositor, with an Apertis plugin added to provide the desired UX,  
718 all running on the X display server. However, the intention is to migrate from  
719 X to Wayland for display access in the near future. The X Window System was  
720 designed for the more trusting environment of 1980s academic computing, and  
721 does not provide an effective security boundary between applications (for exam-  
722 ple, applications can eavesdrop on other applications' input events and output  
723 frames); in the context of a multi-user system which might require differently-  
724 privileged windows to share a display, this is a compelling reason to prefer  
725 Wayland.

726 This section explores several potential models for managing input and output.

727 The basic infrastructure component for Wayland is a *compositor*, which is re-  
728 sponsible for mapping application-supplied surfaces (windows) into the visible  
729 display, routing input events to those surfaces, and applying any visual effect  
730 with a larger scope than an individual application, such as animated transitions  
731 between applications.

732 In the current design proposal for switching single-user Apertis to Wayland, the  
733 compositor is a Wayland version of Mutter, with a version of the Apertis UX  
734 plugin that has similarly been adapted for Wayland; this design is analogous to  
735 GNOME 3's Shell, which also uses the Mutter libraries for window management  
736 and compositing under either X or Wayland. One alternative that has been  
737 considered is to use the Wayland-specific Weston compositor instead of Mutter,  
738 again with a plugin or extension to provide the desired UX. From the perspective  
739 of this document, either Mutter or Weston is viable, and neither is preferred  
740 over the other from a multi-user perspective.

741 The Wayland compositor is part of the TCB for security between apps: it is  
742 responsible for imposing a boundary between the apps that communicate with  
743 it, and preventing them from carrying out undesired actions such as reading  
744 each other's input or taking screenshots of each other's windows. Depending  
745 on the design and implementation, it may also need to be part of the TCB for  
746 security between users.

#### 747 **Single compositor**

748 One possible model is to have a single compositor which starts on boot, runs un-  
749 til shutdown, and is directly responsible for compositing all application surfaces.  
750 This model would be appropriate if there is only one uid shared by all users as  
751 described in section [Sharing one uid between all users](#), since in that model there  
752 is no OS-level isolation between user accounts in any case. It could potentially  
753 also be used in a design where each user has their own uid, by running the  
754 compositor with a non-user-specific uid.

755 The major disadvantage of this situation is that it places the user-level com-  
756 positor into a trusted position: it would become part of the trusted computing  
757 base for separation between users (see [Trusted components](#)). Mutter is not typ-  
758 ically used like this, and has not been designed or audited for this use. Other  
759 compositors would need to be carefully checked for safety for this use. As a  
760 general design principle, the less code is in the trusted computing base (for  
761 any given layer of security), the better; this conflicts with the user-level com-  
762 positor's broad role in mediating between apps, including animated transitions,  
763 copy/paste functionality, on-screen keyboard handling and so on.

#### 764 **Nested compositors**

765 Another possible approach is to make use of *nested compositors*. In this model,  
766 a *system compositor* starts on boot, runs until shutdown, and is responsible for  
767 compositing surfaces provided by system-level components. Instead of surfaces  
768 supplied by applications, the system compositor would primarily be responsible  
769 for compositing surfaces supplied by one or more *session compositors*, and rout-  
770 ing input events to an appropriate session compositor: in effect, it treats the  
771 session compositors like ordinary applications.

772 The system compositor would run under a system (non-user-specific) uid, while

773 the session compositors would run under an appropriate uid for their respective  
774 users.

775 We do not recommend this approach. This design was suggested during early  
776 upstream design work on Wayland, but is now strongly discouraged by Wayland  
777 developers. One major issue is in dealing with input events. Mediating every  
778 input event through two layers of compositor would increase latency, limiting  
779 responsiveness, so it is desirable to grant user sessions direct access to input  
780 events; but granting direct access to session compositors nested inside a system  
781 compositor is problematic, and would cause conflicts between the roles of the  
782 system compositor and the systemd-logind service.

783 Another reason to prefer other models is the increased complexity of the system  
784 as a whole in this model.

## 785 **Switching between compositors**

786 The traditional design for user-switching in X, as described in [Basic multi-user:  
787 log out, log in as another user](#) and [“Fast user switching”: switching user without  
788 logging out](#), is to start a new X server for each user session and switch between  
789 them, for instance by using the Linux kernel’s “virtual console” facility, or by  
790 dynamically attaching/detaching the X servers to the video device. It would be  
791 possible to do the equivalent in a Wayland environment, by running multiple  
792 session compositors, switching access to the video output between them, and  
793 not having a system compositor.

794 In this model, the transition between users would involve systemd-logind revok-  
795 ing the old session compositor’s control over the display (“DRM master” status)  
796 and over input devices, and giving control to the new session compositor. This  
797 could be done at any point in the transition: before, after or during an animated  
798 transition.

799 The major disadvantage of this design is that switching between virtual consoles  
800 is an all-or-nothing operation: the system either displays a frame from one  
801 compositor or a frame from another, but it cannot combine two (for instance  
802 by overlaying them, with transparent regions). It is also not instantaneous, and  
803 would have to be disguised by having a transition where several consecutive  
804 frames are allowed to be the same.

805 For some UX designs, this would not matter. For example, if a designer specifies  
806 that the first user’s session should “fade out” to a black screen or some sort of  
807 “please wait...” placeholder, or move off-screen, then the system could switch to  
808 a matching frame in the new compositor, wait for the switch to occur, and have  
809 the second user’s session “fade in” or move in from off-screen. Similarly, if the  
810 UX for user-switching involves a menu from which the new user is chosen, then  
811 that menu could be used as a fixed point around which to anchor the transition.

812 However, if the desired transition has the two users’ sessions overlap – for in-  
813 stance, a full-screen cross-fade from one to the other, or any animated movement

814 that has both sessions exist on-screen at the same time – then it would be dif-  
815 ficult to achieve these effects in this design without essentially copying a static  
816 screen-capture of one session into the other session. Similarly, if the desired  
817 transition has smooth movement from beginning to end – for example, smooth  
818 horizontal scrolling with the conceptual model that the other user’s session is  
819 “just off-screen” – then the only practical points at which to do the virtual con-  
820 sole switch would be at the very beginning or at the very end; either way, this  
821 would likely result in a few frames of non-responsiveness at a time when the  
822 user might reasonably expect the system to be responsive.

823 Copying a screen-capture of one session into the other session is also a potential  
824 privacy risk, since it results in the screen contents crossing the trust boundary:  
825 it would be technically possible for the second user’s session to save the captured  
826 image.

## 827 **Switching between compositors with a system compositor**

828 Because Wayland does not require clearing the framebuffer during switching,  
829 another possible approach would be to use a system-level compositor without  
830 nesting, used for transitions, and optionally for startup and shutdown. At any  
831 given moment, either the system-level compositor or a session compositor would  
832 be active (have control over input and output), but never both.

833 In this model, as in [Switching between compositors](#), the transition between users  
834 would involve systemd-logind revoking the old session compositor’s control over  
835 the display (“DRM master” status) and over input devices; however, instead of  
836 immediately giving control to the other session, instead it would give control to  
837 a special-purpose system-level compositor which would perform the transition,  
838 and then in turn hand over to the new session. This system-level compositor  
839 could capture the current screen contents as a starting point for the animated  
840 transition, if desired; as in [Switching between compositors](#), the screen contents  
841 would cross a privilege boundary, but unlike [Switching between compositors](#),  
842 the other side of the privilege boundary in this design is a trusted process.

843 The new session compositor could be started without direct access to the display  
844 (it would not yet be the “DRM master”), and instructed to draw its initial state  
845 into a buffer; recent Linux kernel enhancements mean that it could use in-GPU  
846 processing and memory for this drawing operation, without having control over  
847 what is displayed. The system-level compositor would use that buffer as the  
848 endpoint of its animated transition. On completing the transition, it would  
849 instruct systemd-logind to grant full display and input access to the new session  
850 compositor.

851 As a result of its role in user-switching, the system-level compositor used for  
852 the transition would potentially be part of the TCB for security between users.  
853 However, its functionality would be minimal: because it would not be active  
854 during normal use, only during transitions, it would not necessarily need to

process input at all, and its output handling would be limited to performing the animation from the old to the new screen contents.

## Switching between users

If runtime switching between users is required, there is a spectrum of possible approaches.

At one extreme is the simplest form of the approach described in section 4.2.1, where we terminate all of the newly inactive user’s apps and user services (anything that is user-specific), and only non-user-specific processes (system services) continue to run. That has the lowest possible memory and CPU overhead: there is going to be a small amount of overhead during the necessary “grace period” while we let the inactive user’s apps save their state before killing them, but this is minimized.

At the opposite extreme is the “fast user switching” as described in section 4.2.2, in which the inactive user’s entire session, including GUI apps, user services, games, and infrastructure components such as the window manager and X server (or session compositor) continue to run, with the only difference being that they are disconnected from the input and display hardware. That has considerable overhead: in the worst case, where we assume that system services are negligible when compared with per-user components, switching between two users could double the memory and CPU consumption.

We can choose various points along that spectrum depending on OEM and customer requirements. If we can terminate all of the inactive user’s apps and the majority of their user services, the result is close to the first extreme – for example, this could be based on an “agents continue to run across user-switching” flag in the app manifest, perhaps implemented as an Android-style “permission”. App-store curators could carry out more thorough validation on services that request that flag, to ensure that they will not have an adverse performance impact.

If we can terminate all of their apps but must leave *all* of their user services running, we get closer to the second extreme. The closer we are to the second extreme, the higher our hardware requirements for a given performance level will be.

If we terminate at least some of the newly inactive user’s processes, a second axis of variation is how much overlap we are prepared to tolerate between the sessions: to allow those processes to save their current state, a “grace period” will be required between notifying those processes that they must exit, and actually terminating them.

One approach is to disallow overlap entirely, and not start the transition until the inactive user’s session has completely ended, with a “please wait...” message while their processes shut down. However, this maximizes latency and user-visible disruption. To reduce the time required to switch between users, it



896 might be desirable for these processes to continue to run concurrently for a  
897 short time, in parallel with starting the newly active user’s session. There is  
898 a trade-off here: the more CPU time is consumed by the newly inactive user’s  
899 processes, the less is available to display a smooth animated transition to the  
900 newly active user and launch *their* processes. This could be mitigated by de-  
901 prioritizing the CPU and bandwidth consumption of the inactive user’s apps, at  
902 the cost of extending the necessary “grace period” for a given amount of state-  
903 saving activity: for example, if an app’s state-saving procedure would normally  
904 take 50% of the CPU for 0.1 seconds, throttling that app to 5% of the CPU  
905 would make its shutdown take 1 second.

## 906 Preserving “core” functionality across user-switching

907 If user-switching during use is supported, then certain features of the system  
908 must continue to work during and after the user switching operation.

909 For example, navigation-related notifications (notifying the driver that they  
910 should turn off their current route soon, that the speed limit will change soon,  
911 etc.) are time-sensitive, and it would be reasonable to require that these noti-  
912 fications are not interrupted or delayed, even if user switching takes place just  
913 before or even during the notification.

914 Further examples of background features that might be in the category that  
915 must not be interrupted include media playback (if the driver is listening to  
916 music, it would be reasonable to require that playback is not stopped or dis-  
917 rupted by user switching, although interrupting “now playing...” notifications  
918 might still be acceptable) and incoming phone or VoIP calls.

919 These features cannot be assumed to be a fixed part of the operating system: for  
920 example, it should be possible to have uninterrupted media playback via a third-  
921 party audio streaming app, such as one for last.fm or Spotify, or uninterrupted  
922 VoIP call notifications for a third-party VoIP implementation.

923 Conversely, essential operating system features such as preinstalled or non-  
924 removable apps are not necessarily all in the category of features that must  
925 continue to work during user-switching: for example, incoming email notifica-  
926 tions are less time-critical than calls, and it is likely to be acceptable for them  
927 to be paused during user-switching.

928 There are several possible approaches to keeping these features working across  
929 a user-switch. Depending on the concrete requirements and use cases, we could  
930 choose one of these approaches for the whole system, or choose some combination  
931 of them for different apps and services.

932 As mentioned briefly above, there is the potential for a subtle distinction be-  
933 tween components where an interruption to notifications is unacceptable (for  
934 instance, navigation or incoming calls might be in this category), and compo-  
935 nents where an interruption to functionality is unacceptable, but an interruption  
936 to notifications is allowed (or even desirable).

937 For a possible example of the second category, consider music playback, on a  
938 system where a visual notification is triggered when the current track changes.  
939 Suppose we switch the current user from Alice to Bob at 12:00:00, at which time  
940 track 1 is 2 seconds from ending, and the animated transition takes 4 seconds.  
941 It seems reasonable to expect that track 1 must continue to play until 12:00:02,  
942 and it also seems reasonable to expect that track 2 must start at 12:00:02 and  
943 continue to play smoothly. However, it is not necessarily a requirement that the  
944 “now playing track 2” notification cannot be delayed until Bob’s session becomes  
945 fully available at 12:00:04; indeed, this might be considered more desirable than  
946 having it interrupt the animated transition.

## 947 **System services**

948 System services (as defined by **System services**) continue to run regardless of  
949 what is happening in user sessions, so one possible approach is to put “core”  
950 functionality in system services. These could be anywhere from highly privileged  
951 to entirely unprivileged; the distinction here is only that they are independent  
952 of user accounts.

953 For example, network management services such as ConnMan are highly-  
954 privileged system services, whereas the Avahi name-resolution and service  
955 discovery service is system-wide but unprivileged.

956 If this approach is to be used for third-party installable applications, then we  
957 will need to ensure that third-party application bundles can provide system  
958 services, in a way that does not allow those third-party application bundles to  
959 compromise the overall security of the system.

960 For components that deal with user-specific data, making the component into  
961 a system service requires that the component is trusted to provide the correct  
962 privilege separation: for example, if the component has access to multiple users’  
963 private data, it should not reveal one user’s private data to another user unless  
964 the system’s security model allows this to happen.

965 As a general design principle to avoid circular dependencies and unnecessarily  
966 tightly-coupled components, lower layers should not rely on higher layers. Sys-  
967 tem services are at a low layer in the stack, so they should not initiate commu-  
968 nication with user services or users’ graphical sessions. One common approach  
969 to this is to have a component inside each user session whose role is to provide  
970 the user interface for a “headless” system service, separating backend logic and  
971 system-level configuration (the system service) from user interface presentation  
972 and per-user configuration (the user part).

## 973 **User services continuing to run**

974 User services (as defined by **User services**) are inherently per-user. If the end  
975 of a user’s login session terminates their GUI applications but leaves some or  
976 all of their user services running, this could increase system load (as noted

977 in section [Switching between users](#)), but would make user services a suitable  
978 implementation for features that must run uninterrupted. This could apply  
979 either in general, or with restrictions (for example, some subset of the inactive  
980 user’s user-services could continue to run, perhaps according to a “flag” in their  
981 associated app manifests).

## 982 **Distinguishing between the driver and other users**

983 Because the driver is the primary user of the system, one possible refinement  
984 of this requirement would be to say that core functionality associated with the  
985 driver cannot be interrupted, and must retain its ability to display notifications,  
986 but that switching may interrupt functionality associated with other users. This  
987 would limit the additional system load from multiple users: the maximum set  
988 of processes running at a given time would be one non-driver’s full session, plus  
989 whatever subset of the driver’s processes are considered to be necessary.

## 990 **Agents**

991 The Apertis design has the concept of “agents”, which are lightweight back-  
992 ground processes running on behalf of a user. Depending on the precise re-  
993 quirements for agents, they could be implemented as system services, or as user  
994 services, or divided between those two categories.

## 995 **Returning to previous state**

996 Saving and restoring the state of the session is a hard problem in general. Some  
997 platforms, such as Android, made it a central piece of their application life cycle  
998 management and built it right into the application support for the platform. The  
999 fact that Android and iOS have custom platform layers allows them to make  
1000 this viable.

1001 Collabora is not aware of any deployment of OS-level freezing and thawing of  
1002 processes at the moment, but such a strategy could be investigated in the future  
1003 for usage in Apertis. For now, having the application itself care about saving  
1004 and restoring state, even if supported by some high level API, seems to be  
1005 the more realistic approach. More discussion about this can be found in the  
1006 [Applications design](#)<sup>5</sup> document.

## 1007 **Application ownership and installation**

1008 In current app-store platforms such as Apple, Google Play, Steam or PlayStation  
1009 Store, if you buy an app, it is associated with your personal account (Apple,  
1010 Google, etc.) and can be downloaded to any device associated with that account,  
1011 subject to some limits. This is one possible approach to how apps are deployed  
1012 on Apertis.

---

<sup>5</sup><https://jwd.pages.apertis.org/apertis-website/concepts/applications/>

1013 To avoid wasting space with duplicate application installations, current app-  
1014 store implementations with multi-user support, such as Android, have chosen  
1015 to install applications system-wide. If Apertis apps are, conceptually, installed  
1016 per-user, then we recommend implementing this by keeping a list of apps per  
1017 user, and merely hiding apps from users who have not “installed” that app. If  
1018 the user acquires an app that another user has already installed, the system  
1019 could behave as though it was freshly downloaded, but in fact just stop hiding  
1020 the system-wide app from the current user: from the user’s perspective, this is  
1021 indistinguishable from a very fast download and installation.

1022 Another potential conceptual model is to treat apps as more like car accessories.  
1023 You could, for instance, buy a car with metallic paint, or add alloy wheels  
1024 later; when you sell the car, the feature goes with it. Applying this model to  
1025 applications, it could be possible to buy a car with the social media app bundle  
1026 preinstalled, or add the media streaming bundle later, and have the apps go  
1027 with the car when it is sold. In some respects, this is the more natural model  
1028 from the implementation point of view: we do not recommend duplicating the  
1029 app’s executable code and resources, regardless of whether it is conceptually  
1030 installed per-user.

1031 Whichever of these approaches is taken, choosing whether ownership/licensing  
1032 of the app follows the car or the purchaser is primarily a matter for the app  
1033 store implementation, not the multi-user design.

## 1034 Summary of recommendations

1035 As discussed in [User accounts representing users within the system](#), Collabora  
1036 recommends representing each user account as a Unix user ID (uid). The first  
1037 user to be registered in a new system must be able to perform administration  
1038 tasks such as system updates, application installation, creation of new users  
1039 and setting up permissions – that is discussed in [Creating and managing user  
1040 accounts](#).

1041 There is a range of possible approaches to switching between users, discussed in  
1042 section [Switching between users](#). This document does not recommend a particu-  
1043 lar choice from that range, since it depends on the available hardware resources  
1044 and the system’s use-cases and requirements. For budget-limited designs with  
1045 significant hardware limitations, we should consider terminating most user-level  
1046 processes while switching to reduce concurrency, or if this is not acceptable, opt  
1047 to leave user-switching unsupported; for premium models with more capable  
1048 hardware, the more resource-expensive “fast user switching” approach can be  
1049 considered.

1050 In [Preserving “core” functionality across user-switching](#) we outline various pos-  
1051 sible approaches to ensuring that “core functionality” is not interrupted by a  
1052 user switch. Services that need to stay running after a user switch should have  
1053 their background functionality split from their UIs; they can either run as a

1054 different Unix user account ID – a “system service” – or be a specially flagged  
1055 “user service” that is not terminated with the rest of the session.

1056 In [Returning to previous state](#), Collabora recommends that applications should  
1057 be handling saving and restoring of their state themselves, potentially supported  
1058 by helper SDK APIs, which means only applications written with Apertis in  
1059 mind would work. That recommendation comes from the fact that there is no  
1060 solution that would work for all applications.

1061 Ways of having a smooth visual transition when switching users are discussed  
1062 in section [Graphical user interface and input](#). Collabora recommends revisiting  
1063 this topic after Apertis’ graphical user interface and input processing has been  
1064 switched from X to Wayland; our provisional recommendation is to implement  
1065 a hand-off procedure between compositors running under the appropriate user  
1066 ID, either with ( [Switching between compositors with a system compositor](#) or  
1067 without (section [Switching between compositors](#)) an intermediate switch to a  
1068 system compositor.