

# LordOS: um desktop GNU/Linux privado e rápido, construído sobre Debian e Wayland

Documento técnico: projeto, modelo de segurança e estado atual

Versão **0.8.0 Beta** Autores: **Pablo Miranda, Federico Pereira** · iPERFEX

Data: **18 de outubro de 2026**

## Resumo

O LordOS é uma distribuição GNU/Linux livre baseada no Debian 13 que traz um ambiente de desktop próprio, construído a partir dos seus componentes fundamentais sobre Debian e Wayland. Em vez do GNOME ou do KDE, ele tem seu próprio servidor gráfico e compositor Wayland escritos em C ( `lord-server` ), seu próprio motor de interface (React rodando sobre o motor JavaScript QuickJS, com layout, pintura e texto em código C nativo), seus próprios apps nativos e seus próprios serviços de segurança: um lançador de apps em sandbox, um broker de permissões, um armazenamento de permissões por app, um firewall por app aplicado pelo kernel, assinatura de pacotes e um cofre de senhas criptografado. Ele reaproveita o que é maduro e amplamente revisado: o kernel Linux, o Debian e seus pacotes, Mesa, PipeWire, o protocolo Wayland e algumas bibliotecas (FFmpeg, Poppler, libarchive, OpenSSL, Monocypher) confinadas em pequenos processos ajudantes em sandbox.

Este documento descreve o projeto e o estado atual da versão **0.8.0 Beta** (outubro de 2026), o primeiro lançamento público, para um leitor técnico que quer conferir o projeto em vez de acreditar nele. Cada afirmação aponta para um arquivo no repositório público. Os números são os medidos nos registros de teste do projeto e indicam onde foram medidos. O que está projetado mas não construído está marcado como tal.

## 1. Visão

### 1.1 O problema

Um desktop Linux moderno é montado a partir de muitos projetos independentes: um servidor gráfico ou compositor, um toolkit, um gerenciador de sessão, um daemon de configurações, um chaveiro, um gerenciador de rede, uma interface de firewall e os apps. Cada um é bom no que faz, mas é nas emendas entre eles que as coisas ficam lentas, inconsistentes ou inseguras. Três emendas em especial motivaram o LordOS:

- **Peso.** Um desktop que desenha sua interface com um motor web, ou que carrega vários toolkits ao mesmo tempo, precisa de centenas de megabytes antes de o usuário abrir qualquer coisa. O LordOS quer ser confortável em máquinas modestas e num Raspberry Pi 5.
- **Confiança nos apps.** Num desktop Linux tradicional, todo programa que o usuário roda pode ler a pasta pessoal inteira, gravar o microfone, abrir qualquer conexão de rede e ler a área de transferência. O sandbox

existe, mas costuma ser um complemento que o próprio desktop não entende.

- **Visibilidade.** O usuário raramente vê qual app está falando com qual servidor, e mais raramente ainda consegue dizer "este app não pode usar a internet" de um jeito que o app não consiga contornar.

## 1.2 Princípios

O projeto segue uma lista curta de regras, escritas em `AGENTS.md` e `docs/ARCHITECTURE.md` :

1. **Ser dono das partes que o usuário toca; reaproveitar a fundação.** O servidor gráfico, o compositor, o motor de interface, os apps e os serviços de segurança são código do LordOS. O kernel, os pacotes do Debian, Mesa, PipeWire e o protocolo Wayland são reaproveitados.
2. **Uma política de dependências rígida.** X11, wlroots, libdrm, libinput, WebKit, GTK e Qt não são usados dentro do LordOS. Bibliotecas que leem dados não confiáveis só são linkadas por ajudantes em sandbox, nunca pelo compositor.
3. **A interface não executa comandos.** O JavaScript do LordOS não tem API para executar programas arbitrários. Há três exceções estreitas e sem argumentos (o shell de login do Terminal, abrir um app `.desktop` instalado e iniciar um app `.lpk` instalado no seu sandbox).
4. **Root só por portas fechadas.** Tudo o que precisa de root passa por uma tabela fixa de ações validadas ou por um pequeno serviço dedicado com protocolo fechado.
5. **O caminho pela CPU sempre tem que funcionar.** A composição na GPU é uma otimização, nunca um requisito.
6. **O compositor nunca espera.** Ajudantes, apps e serviços falam com o `lord-server` por sockets não bloqueantes; um app travado trava só a si mesmo.
7. **Verificar num sistema real.** Cada marco é conferido numa máquina virtual com capturas de tela reais antes de ser dado como pronto, e compatibilidade que não foi testada não é anunciada.

## Arquivos-fonte

- [AGENTS.md](#)
- [docs/ARCHITECTURE.md](#)

## 2. Arquitetura

### 2.1 Camadas

## As camadas do LordOS

👤 VOCÊ

### Apps

Apps incluídos e o tema · apps .lpk isolados · apps Linux (GTK, Qt, navegadores)



🏠 NOSSO

### Motor de interface do LordOS

React sobre QuickJS, layout, pintura e texto nativos em C. Usado pelos apps do LordOS; apps Linux não passam por ele



🏠 NOSSO

### Servidor gráfico e compositor Wayland do LordOS

lord-server, em C: KMS sem libdrm, evdev sem libinput, composição na CPU ou GPU. Apps Linux e apps .lpk isolados se conectam aqui como clientes Wayland



≡ REUTILIZADO

### Espaço de usuário do Debian 13

dpkg e pacotes, Mesa (GPU opcional), PipeWire, libwayland, libxkbcommon, systemd



⚙️ KERNEL

### Kernel Linux

DRM/KMS, evdev, Landlock, seccomp, cgroups v2, eBPF

👤 Você

🏠 Nosso

≡ Reutilizado

⚙️ Kernel

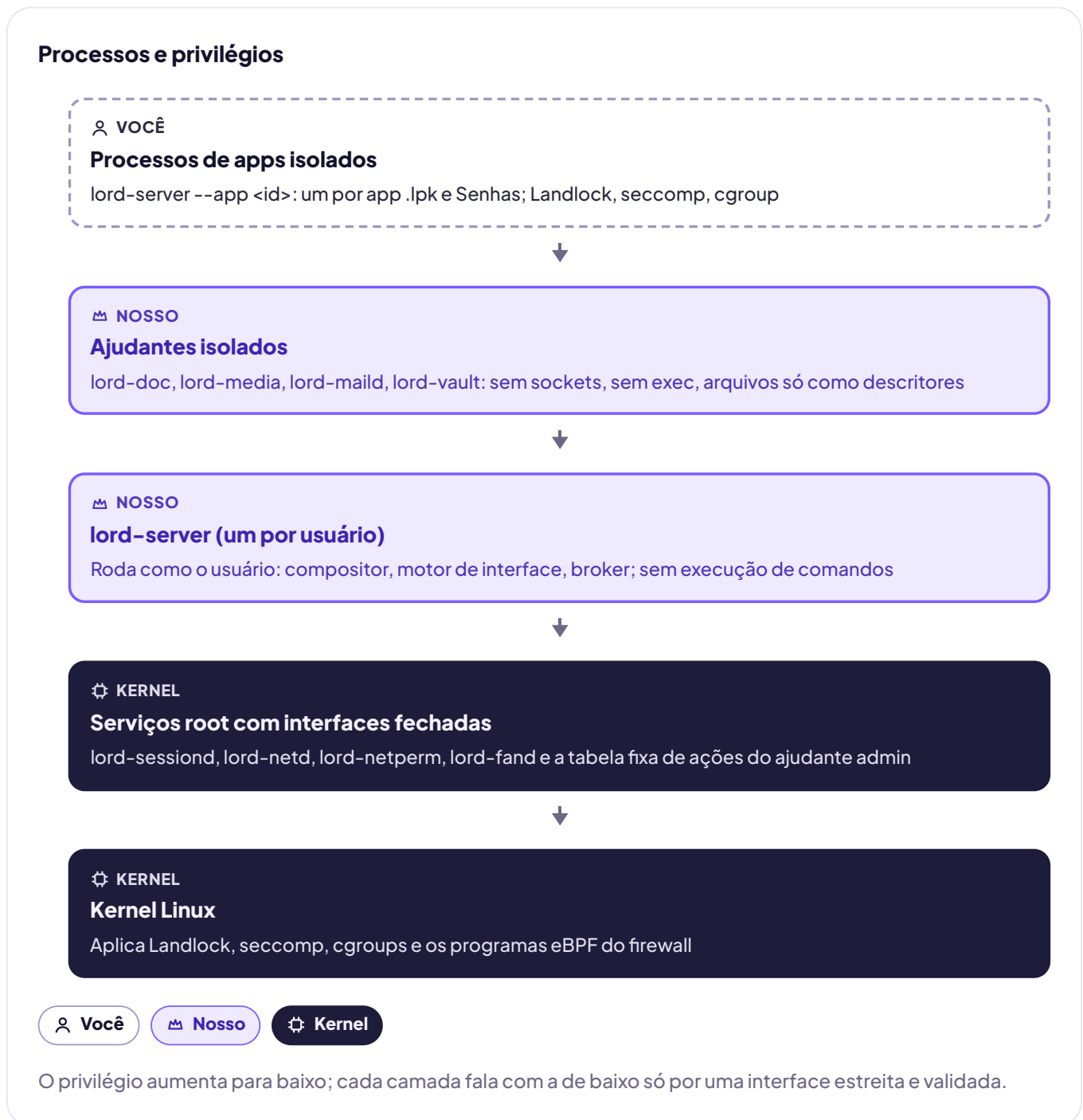
As camadas violeta são código próprio do LordOS; as de baixo são reaproveitadas do Debian e do Linux.

De baixo para cima:

- O **kernel Linux** fornece DRM/KMS para as telas, evdev para a entrada e os recursos de segurança sobre os quais o LordOS se apoia: Landlock, seccomp-bpf, cgroups v2 e eBPF. Em PCs é o kernel do Debian; no Raspberry Pi 5 é a compilação do LordOS do kernel do Raspberry Pi com Landlock ativado (\$7.4).
- O **Debian 13** fornece o espaço de usuário: **dpkg** e o repositório de pacotes, systemd, Mesa, PipeWire, **libwayland**, **libxkbcommon**, CUPS, BlueZ e o resto.
- O **lord-server** é o servidor gráfico e compositor Wayland. Ele controla telas e entrada diretamente, faz a composição e serve o protocolo Wayland para os apps Linux.

- O **motor de interface** mora dentro do **lord-server** : o QuickJS roda a interface em React, e código C nativo faz layout, pintura e texto.
- Os **apps** são de três tipos: os apps incluídos e o tema, que rodam no contexto JavaScript do **lord-server** ; os apps **.lpk** instalados, cada um no seu próprio processo em sandbox com sua própria instância do motor; e os apps Linux, que se conectam ao compositor como clientes Wayland comuns.

## 2.2 Processos e privilégios



Um LordOS em execução tem poucos processos, com papéis bem separados:

| Processo                                                                                            | Roda como                    | Papel                                                                                                                               |
|-----------------------------------------------------------------------------------------------------|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <code>lord-sessiond</code>                                                                          | root                         | tela de login, sessões, empréstimo da tela e da entrada para a sessão visível, área de trabalho remota                              |
| <code>lord-netd</code>                                                                              | root                         | a rede do sistema: DHCP, Wi-Fi, VPN; só as ações <code>net-*</code>                                                                 |
| <code>lord-netperm</code>                                                                           | root (capacidades limitadas) | regras de rede por app, e as restrições de dispositivos e Bluetooth dos apps Linux, aplicadas com programas eBPF de cgroup e de LSM |
| <code>lord-fand</code>                                                                              | root (sem capacidades)       | controle das ventoinhas pela temperatura                                                                                            |
| <code>lord-server</code>                                                                            | o usuário                    | um por sessão: compositor, motor de interface, broker                                                                               |
| <code>lord-server --app &lt;id&gt;</code>                                                           | o usuário, em sandbox        | um por app <code>.lpk</code> em execução (e Senhas)                                                                                 |
| <code>lord-doc</code> , <code>lord-media</code> , <code>lord-maild</code> , <code>lord-vault</code> | o usuário, em sandbox        | ajudantes que processam dados não confiáveis ou segredos                                                                            |
| tela de login                                                                                       | <code>lordos-greeter</code>  | desenha a interface de login; nunca verifica senhas por conta própria                                                               |

Cada processo root tem uma interface estreita e documentada. A §6.3 descreve cada uma.

## 2.3 O que é nosso e o que é reaproveitado

| Nosso (neste repositório)                                                                                 | Reaproveitado                                                                           |
|-----------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------|
| Servidor gráfico e compositor Wayland (C)                                                                 | Kernel Linux                                                                            |
| Motor de interface: nós nativos, layout flexbox, rasterizador, shaping de texto, bidi, métodos de entrada | Debian 13 e seus pacotes; <code>dpkg</code>                                             |
| Renderizador React (configuração de host para o <code>react-reconciler</code> )                           | React 19.3.1, <code>react-reconciler</code> 0.34 (empacotados no tema)                  |
| Apps nativos e temas                                                                                      | QuickJS 2025-04-26 (incorporado como código-fonte)                                      |
| Sandbox, broker, permissões, assinatura, serviço de firewall, cofre                                       | Mesa (opcional), PipeWire, <code>libwayland</code> , <code>libxkbcommon</code>          |
| Gerenciador de sessão, serviço de rede, serviço de ventoinhas                                             | systemd, systemd-resolved, BlueZ, CUPS, wpa_supplicant, OpenVPN, o daemon do strongSwan |
| Motor de pacotes Debian (OpenPGP, xz, HTTP, resolvidor)                                                   | FFmpeg, Poppler, libarchive, OpenSSL, Monocypher (só nos ajudantes)                     |
| Driver de touchpad, emulador de terminal, interpretador WebAssembly                                       | stb (fontes e imagens), fontes sob a OFL                                                |

O LordOS conversa com vários serviços do Debian pelos protocolos deles, com clientes próprios: D-Bus (BlueZ, perfis de energia, data e hora), IPP (CUPS), o protocolo do PulseAudio (PipeWire) e o Secret Service. `libdbus`, `sd-bus`, `libcups`, `libpulse`, `libbpf` e `libapt` não são usados.

## 2.4 Serviços do sistema

- **Rede ( `lord-netd` ).** A rede pertence ao sistema, não a uma sessão. O `lord-sessiond` inicia o `lord-netd` no boot, com caminho fixo e sem argumentos, para que a máquina tenha endereço antes de alguém fazer login. Ele roda só as ações fechadas `net-*` : seu próprio cliente DHCP e código netlink, Wi-Fi pelo `wpa_supplicant` do Debian, WireGuard pelo código netlink próprio do LordOS, OpenVPN, IPsec/IKEv2 pelo `charon` do strongSwan controlado via VICI com cliente próprio do LordOS e, opcionalmente, túneis OpenConnect, PPTP e compatíveis com Fortinet. Ele só fala com os ajudantes root das sessões, por um socket que aceita apenas uid 0.
- **Ventoinhas ( `lord-fand` ).** Um serviço root que lê a temperatura da CPU a cada 2 s e escreve o ciclo das ventoinhas a partir de um perfil, desde o início do boot. Ele só controla ventoinhas que a própria varredura encontrou: o conector de ventoinha do Raspberry Pi 5 e controladores de placa-mãe de uma lista permitida. Ventoinhas de notebook, de GPU e desconhecidas nunca são escritas. Ao parar, ou depois de uma falha, ele restaura as configurações originais do kernel. Não roda programas, não tem rede, nem capacidades, nem nós de dispositivo.
- **Configurações de hardware.** Telas (modos vindos do KMS com nomes do EDID, reversão de 15 segundos com "manter estas configurações?", persistência por tela, rotação, Luz Noturna), Bluetooth pelo BlueZ,

bateria e perfis de energia, ociosidade e tampa, data e hora, notificações dos apps Linux e apps padrão. As mudanças chegam como uevents do kernel e sinais D-Bus; nada fica consultando em loop.

## Arquivos-fonte

- [server/src/main.c](#)
- [server/src/admin.c](#)
- [server/src/net/netd.c](#)
- [server/src/fan/fand.c](#)
- [server/src/hw](#)

## 3. O compositor

### 3.1 Telas sem libdrm

O `lord-server` configura os modos de tela com os ioctls DRM do kernel diretamente ( `kms.c` ): enumera conectores, CRTCs e planos, aloca buffers e troca a página no retraço vertical. Ele escolhe a placa que tem conectores, e não simplesmente `card0` , o que importa em placas onde o controlador de vídeo e a GPU são dispositivos separados (no Raspberry Pi 5, `vc4` e `v3d` ).

### 3.2 Duas formas de montar um quadro

- **Caminho pela CPU.** A interface e as janelas dos apps são copiadas para um canvas, e um buffer "dumb" é apresentado com uma troca de página. Não precisa de nada além do kernel. É o padrão quando não há placa com 3D, e `LORD_GPU=0` o força (o modo de segurança do menu de boot o usa).
- **Caminho pela GPU.** GBM, EGL e GLES2 do Mesa ( `server/src/gpu/gpu.c` ). A interface vira uma textura, e só a região alterada é enviada a cada quadro. As janelas dos apps não são copiadas para o canvas da interface: a interface deixa um buraco transparente onde cada janela fica, e as texturas das janelas são desenhadas por baixo, de modo que um app que só muda o próprio conteúdo não provoca repintura da interface. Os apps podem entregar buffers de GPU ( `linux-dmabuf` ) sem cópias, e um app em tela cheia com um buffer adequado é exibido diretamente (scanout direto). O cursor usa o plano de cursor do hardware.

O caminho é escolhido na inicialização: uma placa sem 3D (simplifiedrm, bochs, virtio-gpu sem virgl e outras) usa a CPU sem nem carregar o Mesa; um renderizador OpenGL por software também volta para a CPU, porque compor com ele custa mais que o caminho de CPU próprio do LordOS. Se um quadro na GPU falha, a GPU é desligada e o caminho pela CPU assume.

**Estado:** o caminho pela GPU foi exercitado com o renderizador por software do Mesa em máquinas virtuais. Os testes em placas AMD e Nvidia reais estão pendentes.

### 3.3 Janelas e entrada

- **Gerenciamento de janelas em C.** Mover e redimensionar rodam em C, com um evento por quadro e sem JavaScript, e a janela sendo movida é desenhada a partir da sua própria camada. As quatro bordas e os cantos redimensionam, com cursores próprios do LordOS.

- **Entrada sem libinput.** Teclados, mouses e mesas digitalizadoras são lidos do evdev. Os layouts de teclado vêm do `libxkbcommon` (todos os layouts XKB, variantes e compose).
- **Touchpad e tela sensível ao toque.** O LordOS tem seu próprio driver multitoque ( `server/src/tp` ): aceleração do ponteiro, toque para clicar e clique com dois dedos, rejeição da palma, rolagem cinética e natural, pinça e gestos de três dedos. Os gestos chegam tanto à interface do LordOS quanto aos apps Linux ( `zwp_pointer_gestures_v1` ). Ele é testado com gestos gravados no `make test` .

### 3.4 Compatibilidade Wayland

O compositor ( `server/src/wl/wl.c` ) usa apenas `libwayland-server` e código gerado a partir do XML dos protocolos. Ele oferece: `wl_compositor` , `wl_subcompositor` , `wl_shm` , `wl_output` (um por tela), `xdg_wm_base` com popups, `wl_seat` (ponteiro, teclado, toque), `wl_data_device_manager` (uma área de transferência compartilhada com o LordOS), `zxdg_decoration_manager_v1` , o protocolo de decoração do lado do servidor do KDE (para que apps GTK 3 como o Firefox recebam a moldura do tema), `zxdg_output_manager_v1` , `wp_fractional_scale_v1` , `wp_viewporter` , `zwp_linux_dmabuf_v1` v4, `wp_presentation` , `zwp_relative_pointer_manager_v1` , `zwp_pointer_constraints_v1` , `zwp_pointer_gestures_v1` e `wp_cursor_shape_manager_v1` .

As janelas Linux entram no gerenciador de janelas do SDK como qualquer outra, com título, foco, maximizar, minimizar e ícone no Dock. Apps que pedem decoração do lado do servidor recebem a moldura do tema; apps que desenham a própria (GTK 4) a mantêm, e a sombra delas é recortada. Apps GTK 4 e libadwaita seguem o modo claro/escuro, a cor de destaque e a ordem dos botões por meio de uma implementação de `org.freedesktop.portal.Settings` que o próprio `lord-server` serve no barramento da sessão; o GTK 3 segue pelo keyfile do GSettings.

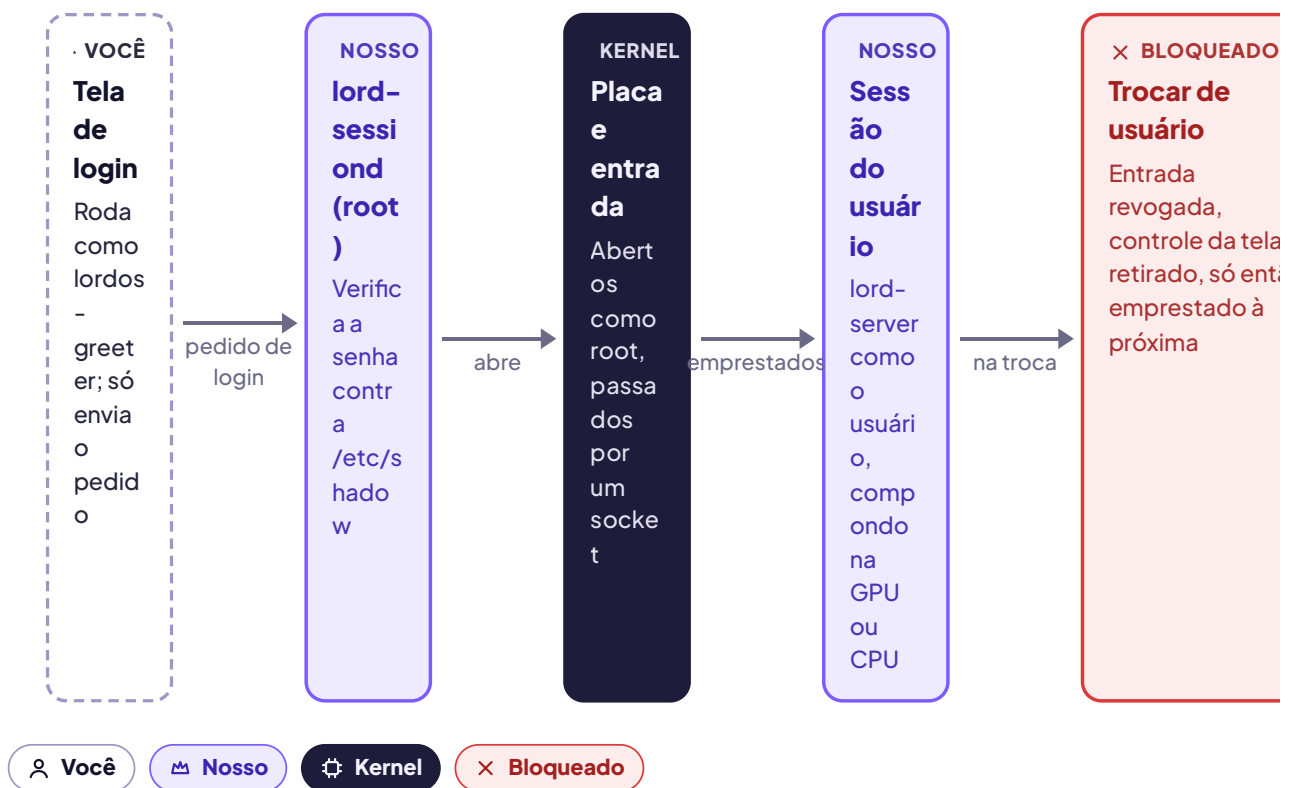
**O Xwayland ainda não é suportado:** o LordOS não roda apps que só falam X11 (o Xwayland está planejado antes da 1.0, §10.3). Também estão pendentes: arrastar e soltar, a seleção primária, os métodos de entrada do LordOS dentro dos apps Linux ( `text-input` ), ativação ( `xdg-activation` ), sincronização explícita e ícones de app em SVG. Nenhum protocolo de captura de tela é oferecido aos apps.

### 3.5 Várias telas e HiDPI

Todas as telas conectadas são controladas ao mesmo tempo, estendidas ou espelhadas, num único espaço lógico, com o arranjo lembrado pela identidade EDID e suporte a conectar e desconectar a quente. Cada tela é pintada na sua própria escala (100, 125, 150, 175 ou 200 %), então texto e vetores são rasterizados na resolução real dela. Os apps Linux recebem um `wl_output` por tela e escala fracionária. Testado com duas saídas virtuais (1280×800 e 2560×1600) nos dois caminhos; monitores reais, painéis de notebook e hubs DisplayPort não foram testados.

### 3.6 Sessões e login

## Emprestar a tela a uma sessão



A sessão visível usa a GPU sem nunca ser root; na troca, o acesso é revogado antes que a próxima receba os dispositivos.

O **lord-sessiond** é o gerenciador de sessões próprio do LordOS:

- **A tela de login** é um **lord-server** rodando como o usuário sem privilégios **lordos-greeter**, com um tema especial. Ela envia pedidos de login ao **lord-sessiond**; **as senhas são sempre verificadas pelo processo root** contra o **/etc/shadow**, nunca pelo JavaScript da tela de login. Tentativas que falham esperam 1, 2, 4... segundos, até 5 minutos.
- **Uma sessão por usuário**, todas podendo existir ao mesmo tempo (troca rápida de usuário). Cada uma é um **lord-server** que começa como root e abandona os privilégios logo em seguida.
- **Empréstimo da tela.** O **lord-sessiond** abre a placa de vídeo e os dispositivos de entrada como root e os empresta à sessão que está no monitor, por um socket Unix. Numa troca, ele os revoga: **EVIOCREVOKE** faz os descritores de entrada da sessão anterior pararem de funcionar, e **DROP\_MASTER** retira o controle da tela, antes de qualquer coisa ser entregue à próxima sessão. A sessão visível continua compondo na GPU sem nunca ser root. Uma sessão fora do monitor desenha em memória, e só se alguém a estiver vendo remotamente.
- **Bloquear a tela realmente tira a sessão do monitor**; não é uma camada desenhada por cima. Se a sessão do usuário travar, a tela de login continua funcionando.
- **Boot.** Um tema de GRUB do LordOS desenha o menu de boot, e um tema de Plymouth do LordOS desenha a tela de abertura (a coroa com uma barra de progresso). A passagem não tem nenhum quadro preto nem texto de console: o Plymouth para de desenhar e solta o controle da tela enquanto seu último quadro continua

visível, o `lord-sessiond` assume o controle da tela e mostra o primeiro quadro real da tela de login, e só então o Plymouth termina, mantendo a tela de abertura até ela ser substituída.

**Área de trabalho remota.** O mesmo serviço oferece o desktop a um navegador na rede local, em `https://<endereço>:5673` (5673 soletra "LORD" num teclado de telefone). Ela vem **desligada por padrão** e é ativada em Ajustes → Compartilhar. Usa TLS 1.2 ou superior com um certificado P-256 autoassinado cuja impressão digital os Ajustes mostram para comparação, uma política de segurança de conteúdo rígida, verificação de mesma origem, esperas por IP após logins com falha e um cookie de sessão `Secure`, `HttpOnly` e `SameSite=Strict`. A página de login nunca lista as contas de usuário. O processo que atende uma conexão abandona o root assim que sabe de quem é a sessão. As atualizações de tela vão em PNG quando a imagem está parada e em JPEG durante o movimento. O áudio está projetado, mas não construído. O acesso pela internet não é previsto; recomenda-se uma VPN.

## Arquivos-fonte

- [server/src/kms.c](#), [server/src/gpu/gpu.c](#), [server/src/outputs.c](#)
- [server/src/input.c](#), [server/src/tp](#), [server/src/ui/grab.c](#)
- [server/src/wl/wl.c](#), [server/src/ui/portal.c](#)
- [server/src/remote/sessiond.c](#), [server/src/remote/monitor.c](#), [server/src/remote/sess.c](#)
- [tools/boot/plymouth](#), [tools/iso/grub](#), [docs/SESSIONS.md](#)

## 4. O motor de interface

### 4.1 Não é um navegador

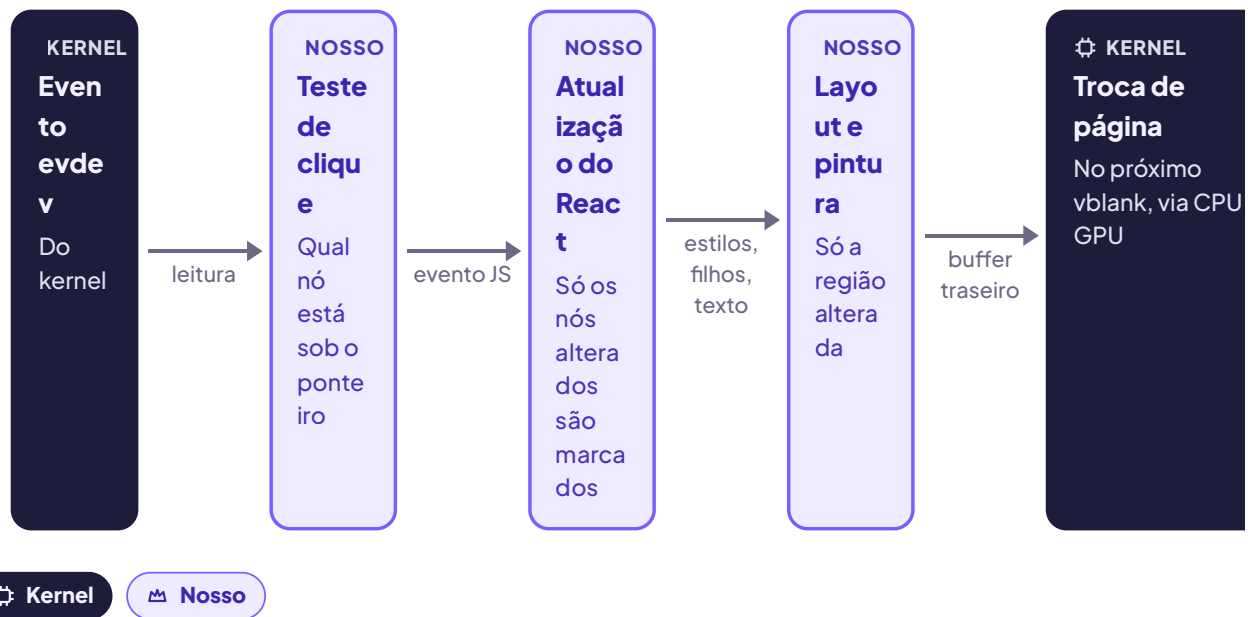
A interface do LordOS é escrita em JavaScript com React, mas não há motor web por baixo. WebKit ou Chromium custariam de 100 a 300 MB de RAM por processo e não seriam código do LordOS. Em vez disso, o tema e os apps usam um renderizador React ( `runtime/index.js` ) que cria e atualiza **nós nativos** em C: `View`, `Text`, `Image`, além de superfícies para janelas Wayland e mídia. Os estilos são um subconjunto do CSS (flexbox, cores, bordas arredondadas e por lado, sombras, desfoque, transformações, transições) aplicado por código C que o projeto controla. O modelo é parecido com o de um framework móvel que renderiza React em views nativas, só que aqui o lado "nativo" é do próprio LordOS.

### 4.2 As peças

- **JavaScript:** QuickJS 2025-04-26, um pequeno motor ES2023 em C, incorporado como código-fonte e embutido no `lord-server`.
- **React:** React 19.3.1 com `react-reconciler` 0.34, empacotados com esbuild em cada tema. O tema padrão com todos os apps pesa 367 KB.
- **Layout:** implementação própria de flexbox do LordOS ( `layout.c` ), com quebra de linha, bordas lógicas de início/fim e espelhamento da direita para a esquerda.
- **Pintura:** rasterizador 2D próprio do LordOS, com retângulos arredondados com antialiasing, sombras em cache, desfoque em resolução reduzida e rastreamento de regiões alteradas: só a região que mudou é repintada. Em ARM, laços NEON opcionais aceleram cópias e máscaras, verificados como idênticos bit a bit ao código C.

- **Texto:** os glifos são rasterizados com `stb_truetype`. O shaping é código OpenType próprio do LordOS (substituições GSUB e posicionamento GPOS, emoji colorido), com bidi Unicode completo, junção do árabe, shaping índico e o Universal Shaping Engine (409 de 409 frases de teste idênticas ao HarfBuzz em `tests/hb_compare.py`), quebra de linha CJK, separação de palavras em tailandês e fonte de reserva por escrita. HarfBuzz, FreeType, fontconfig, fribidi e ICU não são linkados pelo `lord-server`.
- **Métodos de entrada:** coreano (hangul), japonês (kana e SKK) e chinês (pinyin), em `server/src/ime.c`, sem ibus nem fcitx.

#### De um movimento do mouse a um novo quadro



O motor repinta só o que mudou e troca a página no próximo retraço vertical.

## 4.3 Como o JavaScript chega ao sistema

O lado JavaScript chama serviços nativos pelas APIs `lord.*` implementadas em C (`server/src/ui/js_*.c`): `lord.fs` (arquivos), `lord.settings`, `lord.net`, `lord.pkg`, `lord.media`, `lord.mail`, `lord.vault`, `lord.fw`, `lord.monitor`, `lord.fan` e outras. Toda chamada valida seus argumentos em C. As regras são rígidas:

- O `lord-server` roda sem root, e **nenhuma API executa comandos arbitrários**.
- O `lord.fs` só escreve na pasta pessoal e nas montagens do próprio usuário, não tem "apagar" genérico (só mover para a Lixeira) e recusa as pastas que o sistema lê para iniciar ou instalar algo, ou que guardam decisões de segurança: `~/.local/share/applications`, `~/.config/autostart`, a pasta de apps instalados, `settings.json`, o arquivo de permissões, as regras do firewall, o arquivo de contas de e-mail e a pasta do cofre.
- Tudo o que precisa de root passa pelo `admin.c`: uma tabela fechada de ações com argumentos validados do lado root, caminhos de programa fixos e configuração gerada pelo LordOS, nunca texto do usuário passado a um shell. As senhas vão pela entrada padrão, nunca pela linha de comando.
- Trabalho longo nunca bloqueia um quadro: ajudantes e serviços são consultados pelo laço principal sem espera.

## 4.4 Idiomas

A interface vem em inglês (o padrão), espanhol e português, e novos idiomas entram como pacotes de idioma, sem mudar código. O texto passa pelo `t( 'namespace.key' )` do SDK, com plurais e formatos regionais. A interface inteira se espelha para idiomas da direita para a esquerda, e a região ( `LANG / LC_*` ) é repassada aos apps Linux. `node tools/i18n-check.mjs` confere que toda chave existe em todos os idiomas e que nenhuma marca de terceiros aparece na interface.

## Arquivos-fonte

- [runtime/index.js](#), [sdk/index.js](#)
- [server/src/ui/js.c](#), [server/src/ui/node.c](#), [server/src/ui/layout.c](#), [server/src/ui/paint.c](#)
- [server/src/ui/shape.c](#), [server/src/ui/otf.c](#), [server/src/ui/bidi.c](#), [server/src/ime.c](#)
- [server/src/ui/js\\_sys.c](#), [server/src/fsops.c](#)

## 5. Aplicativos

### 5.1 Apps nativos

Cerca de trinta apps nativos vêm com o LordOS, todos apps React sobre o mesmo motor e o mesmo SDK: Arquivos, Ajustes, Terminal, Loja, Editor de Texto, Notas, Calculadora, Monitor, Visualizador de PDF, Fontes, Emoji e Símbolos, Scanner, Fotos, Player, Capturas de Tela, Câmera, Gravador, Senhas, Discos, Backups, Logs, Relógio, Ajuda, E-mail, Calendário, Contatos, Lembretes, Sobre e o instalador. Alguns destaques:

- **Terminal:** emulador compatível com xterm escrito pelo LordOS em C, com abas, painéis divididos, perfis, imagens embutidas e Sixel, e sessões que sobrevivem a um reinício do `lord-server`.
- **Monitor:** CPU, memória, discos, processos, refrigeração e conexões de rede por app num mapa-múndi, com o país consultado numa base local. A thread de amostragem só existe enquanto o app está aberto.
- **Discos:** leitor SMART próprio do LordOS, formatação e pendrives inicializáveis verificados com SHA-256.
- **Backups:** snapshots com hard links e criptografia opcional, rodando com baixa prioridade.

### 5.2 Apps `.lpk` instaláveis

Os apps nativos de terceiros são empacotados como `.lpk`: um tar comprimido com gzip contendo um manifesto, o `app.js` empacotado, um ícone obrigatório e traduções e recursos opcionais. Um app não traz React nem o SDK; usa os do sistema, o que o mantém pequeno (o app de exemplo Fractal pesa 5 KB). Eles são instalados sem root em `~/.local/share/lordos/apps/<id>/`. O instalador recusa links, dispositivos, caminhos absolutos, `..`, caracteres estranhos, mais de 2.000 arquivos ou 64 MB, e substitui a versão anterior de forma atômica. Cada um roda isolado (§6.4).

### 5.3 A Loja e os pacotes Debian

A Loja mostra os apps do Debian por categoria, com busca, além de apps `.lpk` e arquivos `.deb` locais, e sempre mostra um plano antes de instalar ou remover. Os pacotes Debian são tratados pelo **motor de pacotes próprio do LordOS** ( `server/src/pkg` ): ele lê as fontes do apt, verifica as assinaturas do `InRelease` com código OpenPGP próprio (RSA e Ed25519), recusa índices expirados, reaproveitados (replay), de outra suíte ou com

dados depois da assinatura, confere cada índice e cada `.deb` com os valores SHA-256 do índice assinado e resolve as dependências (planos idênticos aos do apt em 12 casos de teste). Só o último passo é do Debian: o `dpkg` desempacota e configura, e continua sendo o único que escreve no seu banco de dados, então o `apt` segue funcionando. O LordOS não tem cliente TLS próprio, então as fontes `https://` são baixadas por `http://`; a integridade depende das assinaturas e dos hashes, mas os nomes dos pacotes baixados ficam visíveis na rede.

## 5.4 Apps Linux

Apps Linux com um arquivo `.desktop` aparecem no lançador de apps. A abertura passa por `lord.apps.launch(id)`: a interface só pode indicar um arquivo `.desktop` instalado; o `lord-server` o relê, confere, monta o `argv` a partir da linha `Exec` conforme a especificação e o executa com `execve`, sem shell nem argumentos de quem chamou, dentro do seu próprio escopo de cgroup pertencente ao root. Chromium e Firefox vêm pré-instalados na ISO e foram verificados com a moldura do LordOS. **Os apps Linux não ficam em sandbox por padrão**: eles rodam como o usuário, como em qualquer Debian. Ganham atribuição, o firewall por app e as restrições por app que o dono definir (§6.6), incluindo uma visão opcional "sem acesso aos seus arquivos" aplicada pelo lançador.

## 5.5 WebAssembly

O LordOS inclui seu próprio interpretador WebAssembly em C (`server/src/wasm`), exposto como o global padrão `WebAssembly`, para que um app nativo possa trazer módulos compilados de C, C++, Rust ou Zig. Ele cobre o WebAssembly 2.0 sem SIMD (o MVP mais extensão de sinal, conversões saturadas, multi-valor, memória em bloco e tipos de referência), passa em 26.290 de 26.290 asserções da suíte de testes oficial na VM e transforma todo erro num trap em vez de uma falha. No exemplo do projeto, ele rodou de 3 a 5 vezes mais rápido que o mesmo código em JavaScript.

## 5.6 Instalar o LordOS

- **ISO live** para PCs de 64 bits, com instalador gráfico (UEFI e BIOS).
- **Imagem para Raspberry Pi 5**, para cartão SD ou NVMe (§7.4).
- **Num Debian 13 já instalado**: `curl -fsSL https://lordos.org/install.sh | sh`. O script pergunta antes de mudar qualquer coisa, baixa o `SHA256SUMS` e verifica sua assinatura Ed25519 com a chave de release embutida no script, recusa continuar sem uma assinatura válida, confere o SHA-256 do `.deb`, instala com o apt e configura o sistema como a ISO. Os arquivos do release são publicados no GitHub Releases.

## Arquivos-fonte

- [apps](#), [server/src/lpk.c](#), [tools/lpk.mjs](#)
- [server/src/pkg](#), [server/src/apps.c](#)
- [server/src/wasm/wasm.c](#), [server/src/ui/js\\_wasm.c](#)
- [tools/install/install.sh](#)

## 6. Segurança

### 6.1 Modelo de ameaças

O LordOS é projetado contra estas ameaças:

- **Um app instalado malicioso ou comprometido** que quer os arquivos, as chaves, o microfone, a câmera, a área de transferência ou a rede do usuário.
- **Documentos e mensagens maliciosos:** PDFs, arquivos compactados, fontes, imagens, vídeos e e-mails feitos para explorar um leitor.
- **Um app Linux que quer mandar dados para fora**, ou escapar das regras definidas para ele.
- **Um atacante na rede local** tentando usar a área de trabalho remota ou adulterar downloads.
- **Pacotes e releases adulterados.**

Fora do modelo, por enquanto: um kernel ou uma conta root comprometidos; ataques físicos; canais laterais; e apps Linux que o dono não restringiu, que rodam como o usuário, como em qualquer Debian, e podem ler os arquivos dele. Os apps Linux são confinados apenas pelas restrições que o dono define para eles (§6.6), e só quando são abertos pelo lançador de apps.

### 6.2 Privilégio mínimo

- O `lord-server` roda como o usuário. Numa sessão gerenciada, ele nunca segura os dispositivos de vídeo ou de entrada por conta própria: o `lord-sessiond` os empresta e os revoga.
- Bibliotecas que leem dados não confiáveis só são linkadas por ajudantes: FFmpeg e libva pelo `lord-media` ; Poppler e libarchive pelo `lord-doc` ; OpenSSL pelo `lord-maild` e pelo `lord-sessiond` ; Monocypher pelo `lord-vault` .
- Os ajudantes recebem arquivos e dispositivos como descritores abertos pelo `lord-server` ; nunca abrem caminhos.
- O compositor nunca espera por um ajudante ou por um app, então uma falha ou um travamento fica contido.

### 6.3 Os processos root e suas interfaces fechadas

| Processo                                        | Aceita                                                                                                                                                                                          | Nunca faz                                                           |
|-------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------|
| ajudante <code>admin.c</code>                   | uma tabela fixa de ações, com argumentos validados como root                                                                                                                                    | ações genéricas, texto do usuário num shell                         |
| <code>lord-sessiond</code>                      | pedidos de login da tela de login; JSON curto de sessão ( $\leq 4$ KB)                                                                                                                          | deixar a tela de login verificar senhas; ler pixels como root       |
| seu processo filho de posicionamento em cgroups | <code>enter &lt;scope&gt; &lt;limits&gt;</code> só para o próprio processo que pede, vindo da folha <code>server</code> de uma sessão e rodando um binário da pasta do <code>lord-server</code> | posicionar processos arbitrários; devolver a árvore ao usuário      |
| <code>lord-netd</code>                          | ações <code>net-*</code> vindas apenas de pares root                                                                                                                                            | ações que não são de rede; rodar qualquer coisa vinda de uma sessão |
| <code>lord-netperm</code>                       | <code>RELOAD</code> sem dados                                                                                                                                                                   | rodar programas; receber argumentos; abrir sockets de internet      |
| <code>lord-fand</code>                          | <code>STATUS</code> para qualquer um; <code>PROFILE</code> , <code>CURVE</code> , <code>FORCE</code> para administradores                                                                       | indicar caminhos ou dispositivos; rodar programas; usar a rede      |

## 6.4 O que é isolado, e o sandbox de apps

Nem todo programa no LordOS fica em sandbox. Isto é exatamente o que vale para cada tipo:

| Tipo                             | Processo                                              | Sandbox                                                                                                                                           | Escopo de cgroup              | Broker e permissões                                      | Firewall e restrições (§6.6)                                                                                    |
|----------------------------------|-------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------|----------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| Apps <code>.lpk</code> e Senhas  | próprio ( <code>lord-server --app &lt;id&gt;</code> ) | Landlock, seccomp, sem capacidades                                                                                                                | próprio, do root, com limites | cada chamada é conferida em C                            | regras de rede e as oito restrições que não são de rede, no broker                                              |
| Os demais apps incluídos         | dentro do <code>lord-server</code>                    | não (fazem parte do desktop)                                                                                                                      | o do <code>lord-server</code> | sem broker; câmera, microfone e tela são conferidos em C | como "LordOS": regras de rede; câmera, microfone, tela e notificações                                           |
| Apps Linux pelo lançador de apps | próprio                                               | nenhum por padrão; com "sem acesso aos seus arquivos", namespaces próprios de usuário, montagem e PID mais Landlock, a partir da próxima abertura | próprio, do root              | não                                                      | regras de rede; câmera, microfone, USB, sockets Bluetooth, arquivos e notificações; não a área de transferência |
| Programas do Terminal            | filhos do shell do Terminal                           | não                                                                                                                                               | o do <code>lord-server</code> | não                                                      | como "LordOS": regras de rede                                                                                   |

Um programa iniciado de outro jeito, fora do lançador de apps, não está em nenhum escopo de app e não recebe regras por app. Flatpak e bubblewrap não são usados.

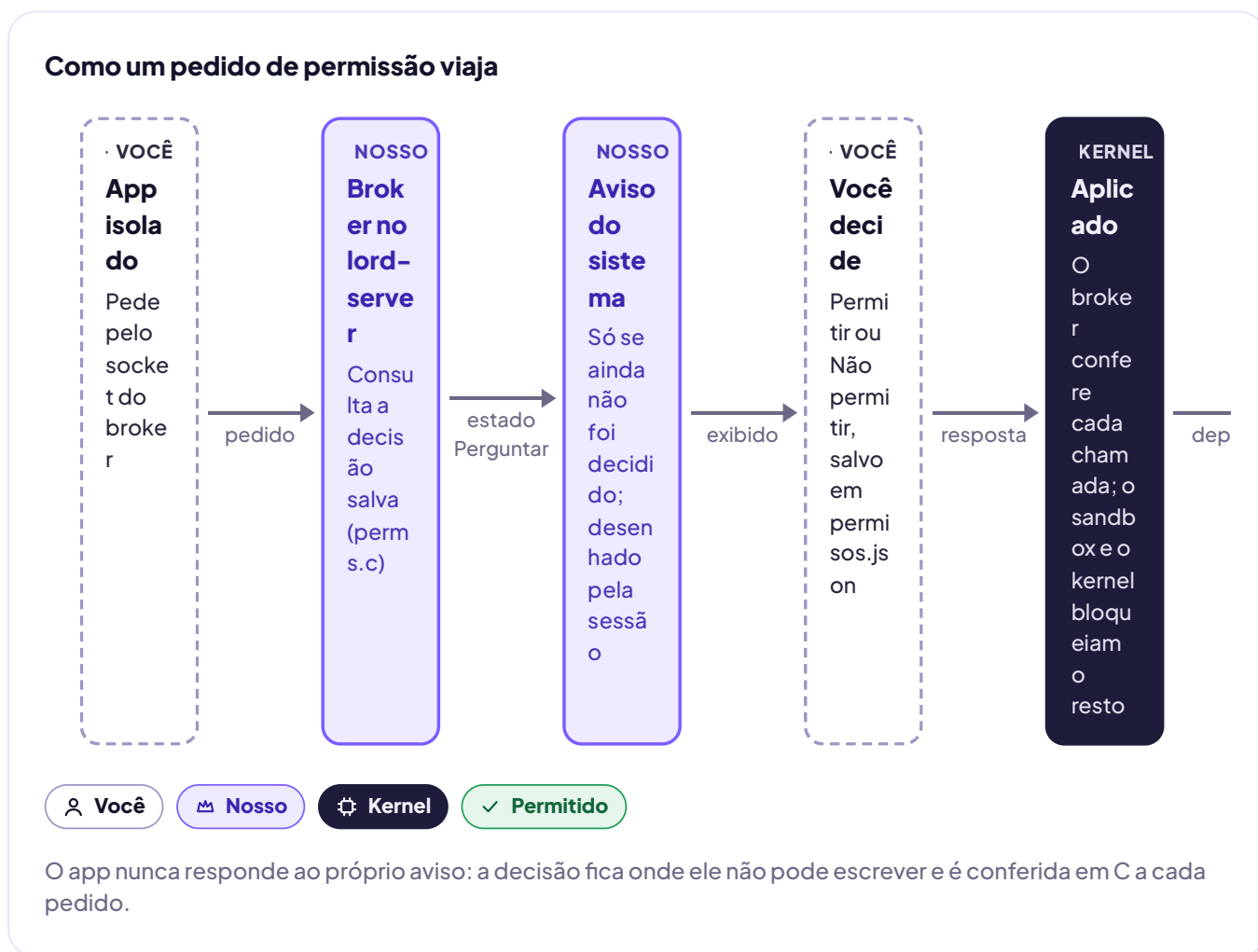
Cada app `.lpk` , e o app Senhas, roda como `lord-server --app <id>` : o mesmo binário em "modo app", iniciado pelo `lord-server` com um id de pacote validado como único argumento. O processo se conecta ao compositor como cliente Wayland e abre o socket do broker, depois aplica o sandbox e só então carrega o código do app ( `server/src/sec/sandbox.c` ):

- **Landlock** (ABI até 6): acesso somente leitura ao pacote, ao runtime compartilhado de apps, às fontes e às bibliotecas; leitura e escrita só em `~/.local/share/lordos/appdata/<id>` ; escopo restrito para sockets Unix abstratos e sinais.
- **seccomp-bpf**: uma lista de chamadas de sistema permitidas, em todas as threads (TSYNC), sem `ptrace` , `mount` , `kexec` , `bpf` , `perf_event_open` , `keyctl` , carregamento de módulos ou `execve` . Novos sockets Unix e `socketpair` são recusados, porque o Landlock até a ABI 6 não filtra `connect` para sockets Unix com nome; isso mantém o app longe do barramento da sessão, do serviço de impressão e do socket Wayland.
- **Sem privilégios**: `no_new_privs` , um conjunto de capacidades vazio e o bounding set descartado.
- **cgroup v2**: `app-lordos-lpk-<id>.scope` , criado pelo root, com `memory.high` de 192 MB, `memory.max` de 512 MB e um núcleo de CPU.

- **Rede:** sem a permissão `network` (declarada e permitida), o seccomp recusa sockets de internet e o Landlock recusa connect e bind TCP. Namespaces de rede não são usados, porque criar um exige privilégios que o `lord-server` não tem.

Se faltar um recurso do kernel, o app inicia mesmo assim e o log diz qual parte está faltando. O app de teste propositalmente malicioso ( `tests/lpk-evil` ) falha em tudo o que tenta; o `build/sandbox-test` passa em 14 de 14 verificações para apps e 14 de 14 com rede. Medido numa VM: cerca de 13,7 MB de RSS por app isolado, inicialização mediana de 106 ms e 0,89 ms para aplicar o sandbox.

## 6.5 O broker, as permissões e o diálogo de arquivos



O app isolado só fala com o `lord-server` por um socket `SOCK_SEQPACKET`. Cada pedido é conferido em C: acesso a arquivos só dentro da pasta de dados do próprio app; leitura da área de transferência só com a permissão `clipboard`; pedidos de permissão respondidos pela decisão salva ou, se não houver, por um **aviso desenhado pelo sistema** (*Permitir / Não permitir*); câmera e microfone só através do `lord-server`.

- **As decisões** ficam salvas por usuário em `~/.config/lordos/permisos.json`, fora do alcance do app. A consulta é  $O(1)$ .
- **O diálogo de arquivos é o powerbox próprio do LordOS** ( `sdk/powerbox.jsx` ), não o `xdg-desktop-portal`: o `lord-server` o desenha, abre o arquivo escolhido e passa só aquele descritor. A única interface de portal de desktop que o LordOS implementa é `org.freedesktop.portal.Settings`, para o tema do GTK.

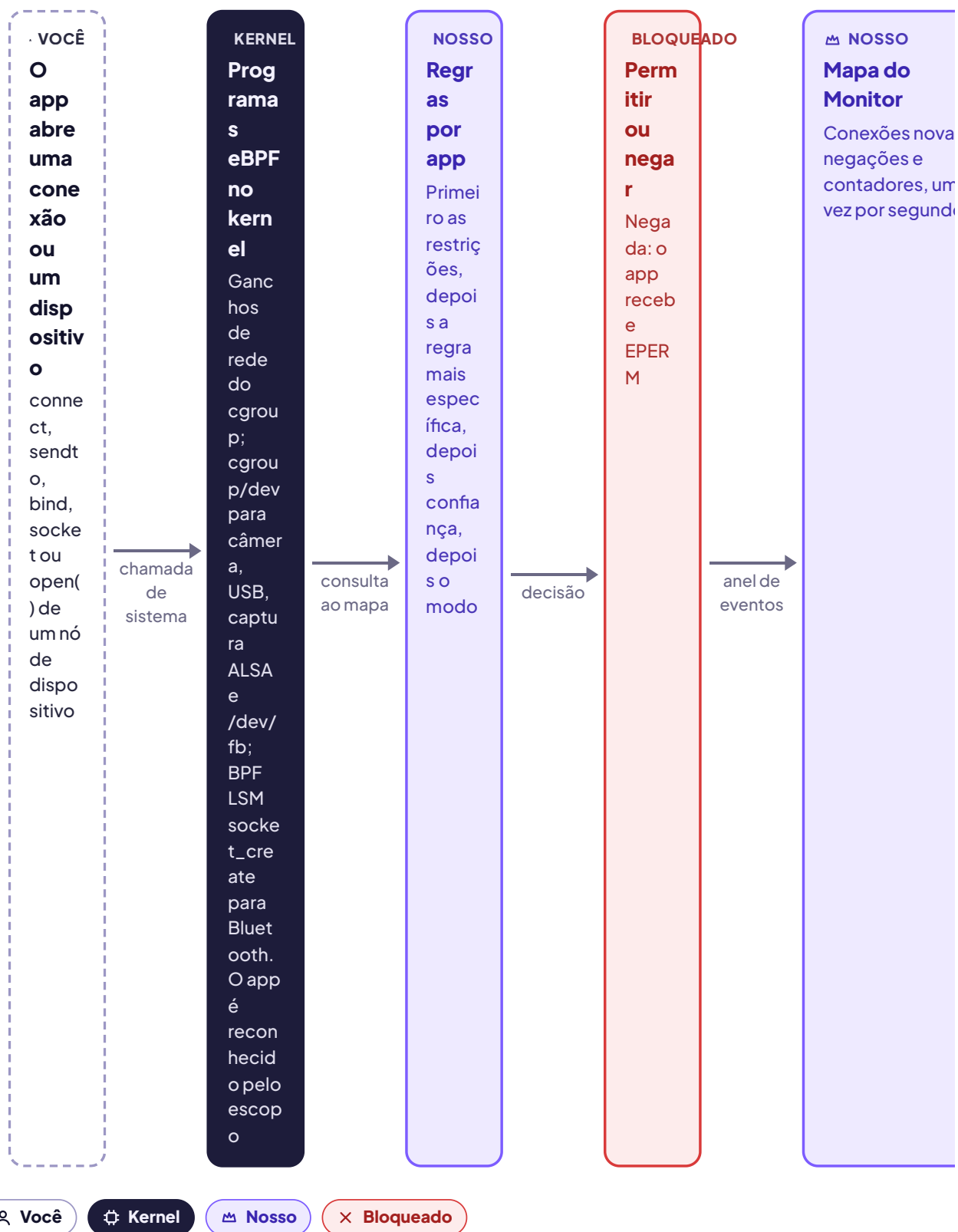
- **Microfone:** o `lord-server` é dono do fluxo do PipeWire e dá ao app um anel compartilhado somente leitura.
- **Câmera:** o `lord-server` abre o dispositivo e o entrega só ao `lord-media`; o app recebe uma visão somente leitura dos quadros decodificados e precisa confirmar cada um, e o `lord-server` valida cada confirmação.
- **Revogação:** negar a câmera ou o microfone nos Ajustes os interrompe na hora (o `lord-media` do app é encerrado; o fluxo do PipeWire é fechado). A permissão `network` é decidida na inicialização.
- **Chaveiro:** os pedidos ao chaveiro de um app isolado levam um dono que o `lord-server` define a partir do id verificado do app.

**Restrições valem mais que permissões.** Uma restrição do firewall (§6.6) vence qualquer decisão salva: o pedido é recusado sem aviso, mesmo que o app já tenha tido permissão, e restringir um app em execução tira dele a câmera e o microfone na hora, como revogar.

**Limites.** Não há "permitir uma vez". Algumas permissões declaráveis (Bluetooth, abrir apps, notificações) ainda não têm chamada no broker, então os apps isolados simplesmente não podem usar esses recursos; escutar numa porta depende da permissão de rede. Um app Linux rodando como o usuário poderia editar o arquivo de permissões, a menos que esteja restrito com "sem acesso aos seus arquivos".

## 6.6 O firewall por app, suas restrições e o Monitor

## O caminho de uma conexão ou de um dispositivo



A decisão acontece no kernel quando abre uma conexão, um socket ou um dispositivo; nada roda por pacote.

O `lord-netperm` aplica regras por app no kernel. Os apps Linux abertos pelo lançador de apps rodam em `app-lordos-<id>.scope`, os apps `.lpk` em `app-lordos-lpk-<id>.scope`, e o resto da sessão conta como `lordos`.

Todos os escopos pertencem ao root. O serviço carrega sete programas de rede ( `connect4/6` , `sendmsg4/6` , `bind4/6` , `sock_create` ), um programa de dispositivos ( `cgroup/dev` ) e um programa LSM por cgroup sobre `socket_create` no cgroup do `lordos.service` , descobre o app de cada processo pelo seu ancestral de cgroup e devolve permitir ou negar ( `EPERM` ). Os programas são compilados dentro do binário e carregados com o carregador próprio do LordOS pela chamada de sistema `bpf()` , que também lê o BTF do kernel para achar o gancho LSM; libbpf e bcc não são usados. Se o kernel recusar o programa de dispositivos ou o LSM, os programas de rede continuam funcionando e o serviço avisa.

- **As regras** aceitam domínios exatos, curingas, redes IPv4/IPv6, `lan` , `internet` , portas, protocolos e direção. As restrições (sem internet, sem LAN, sem rede, sem conexões de entrada) vêm primeiro; depois vence a regra mais específica, com negar ganhando de permitir em especificidade igual; depois os apps confiáveis; depois o modo (silencioso, o padrão, ou desligado). O `npcompile.c` pré-calcula isso em mapas de hash e de maior prefixo correspondente. Um teste roda o código de decisão do kernel contra o decisor de referência em 480.000 conexões aleatórias: O diferenças.
- **Os nomes de domínio** vêm só do resolvedor do sistema: a interface de monitoramento do systemd-resolved ou, sem ele, respostas da porta 53 dos servidores configurados, lidas por um socket de pacotes que só recebe. Respostas que um app recebeu nos próprios sockets nunca são levadas em conta. Por isso, um app que resolve nomes por conta própria com DNS criptografado (DNS sobre HTTPS dentro de um navegador, por exemplo) não é alcançado pelas regras de domínio: uma regra "permitir só example.com" continua bloqueando-o, mas uma regra "bloquear example.com" não o impede. As regras por endereço e por rede, o padrão do app e suas restrições continuam valendo, e os Ajustes dizem isso abaixo das regras.
- **As regras na interface.** Ajustes → Privacidade → Regras de rede por app mostra o padrão de cada app ("Permitir tudo" ou "Bloquear tudo exceto as regras") e suas regras na ordem em que vencem, com quando cada uma coincidiu pela última vez; a folha que adiciona uma regra confere o que é digitado enquanto se digita. Em Monitor → Rede, "Permitir / Bloquear este domínio" num destino de um app abre a mesma folha, preenchida com o nome que o resolvedor do sistema deu para aquele endereço.
- **Fugas.** A árvore de cgroups pertence ao root, então um app não consegue se mover. Um processo que pede ao gerenciador systemd do usuário uma nova unidade ( `systemd-run --user` ) cai sob `user@<uid>.service` ; o `lord-netperm` anexa os mesmos programas ali e submete esses processos à união mais estrita das restrições dos apps restritos daquele usuário em execução. Testado: todas as tentativas de fuga negadas, outros usuários não afetados.
- **Custo:** cerca de 0,5 µs por connect UDP, TCP dentro do ruído, nenhuma mudança na vazão, já que nada roda por pacote.
- **O Monitor** mostra as conexões de cada app (a partir de `sock_diag` ) e os destinos que o firewall informou há pouco, os bytes, os destinos por país a partir de uma base local e um botão de Regra por app. Nenhum endereço sai da máquina para consultas: sem DNS reverso, sem geolocalização online.

**Restrições além da rede.** Além das quatro restrições de rede (sem internet, sem LAN, sem rede, sem conexões de entrada), o dono pode restringir de um app a câmera, o microfone, a tela, o USB, os arquivos, a área de transferência, o Bluetooth e as notificações. Cada uma é aplicada onde o recurso é entregue, conferida contra o escopo do app e nunca contra um nome que o app dá a si mesmo, e vence qualquer permissão concedida em `permisos.json` .

## Onde cada restrição é aplicada

 NOSSO

### Broker no lord-server

Apps .lpk e Senhas: as oito restrições, recusadas sem aviso



 NOSSO

### Lançador de apps

Apps Linux, sem acesso aos seus arquivos: namespaces próprios de usuário, montagem e PID, uma home em tmpfs e Landlock, a partir da próxima abertura



 NOSSO

### Servidor de notificações (notifyd)

Notificações de apps Linux: a identidade do remetente vem do barramento; se algo falha, é descartada



 NOSSO

### Guarda do PipeWire (pwguard)

Microfone e câmera pelo PipeWire: identidade tirada das credenciais do socket, ao vivo



 KERNEL

### Programas eBPF do lord-netperm no kernel

Ganchos de rede; cgroup/dev para câmera, USB, captura ALSA e /dev/fb; BPF LSM socket\_create para sockets Bluetooth

 Nosso

 Kernel

Cada restrição é conferida onde o recurso é entregue, contra o escopo do app, que é do root, nunca contra um nome que o app dá a si mesmo.

| Restrição             | Apps .lpk e Senhas                                | Apps incluídos ("LordOS")  | Apps Linux pelo lançador de apps                                                                                   |
|-----------------------|---------------------------------------------------|----------------------------|--------------------------------------------------------------------------------------------------------------------|
| câmera                | broker<br>( <code>brokercam.c</code> )            | <code>js_media.c</code>    | <code>cgroup/dev</code> nos nós V4L2, e o guarda do PipeWire                                                       |
| microfone             | broker                                            | <code>js_media.c</code>    | o guarda do PipeWire, e <code>cgroup/dev</code> nos nós de captura ALSA                                            |
| tela                  | broker (sem API de captura no modo app)           | <code>js_media.c</code>    | por projeto: sem protocolo de captura, sem servidor X, sem portal de captura de tela; <code>/dev/fb*</code> negado |
| USB                   | broker (a chave de segurança de Senhas)           | não oferecida              | <code>cgroup/dev</code> : nós USB diretos, hidraw e serial USB                                                     |
| arquivos              | broker (seletor de arquivos, gravações da câmera) | não oferecida              | o lançador: namespaces próprios, uma home em tmpfs e Landlock, a partir da próxima abertura                        |
| área de transferência | broker                                            | não oferecida              | não disponível                                                                                                     |
| Bluetooth             | broker (sem API de Bluetooth no modo app)         | não oferecida              | programa BPF LSM sobre <code>socket_create</code> : sem sockets <code>AF_BLUETOOTH</code>                          |
| notificações          | a permissão <code>notifications</code>            | <code>sdk/notify.js</code> | <code>notifyd</code> , pela identidade verificada de quem chama                                                    |

Para os apps isolados, o broker ( `sec/restrict.c` ) é o único ponto de aplicação: um pedido restrito é recusado sem aviso, e restringir um app em execução tira dele a câmera e o microfone na hora ( `tests/broker_restrict_test.c` cobre cada restrição, com e sem ela). A linha "LordOS" é o próprio desktop (Arquivos, copiar e colar, Discos, os ajustes de Bluetooth), então ela não oferece arquivos, USB, área de transferência nem Bluetooth; em vez disso, restringem-se apps individuais. Para os apps Linux:

- **Dispositivos.** No cgroup v2, o controlador de dispositivos é um programa BPF. O do `lord-netperm` recusa `open()` e `mknod()` das classes de dispositivo que o app tem restritas (V4L2, USB direto, hidraw, serial USB, nós de captura ALSA, `/dev/fb*` ) e permite todo o resto. Vale a partir do próximo `open` ; um descritor já aberto continua utilizável.
- **Sockets Bluetooth.** O gancho de sockets inet nunca vê `AF_BLUETOOTH` , então um programa LSM por cgroup sobre `socket_create` os recusa.
- **Arquivos.** Logo antes do `execve` , o filho do lançador entra em namespaces próprios de usuário, montagem e PID, sem privilégios. A pasta pessoal é um tmpfs vazio com só as pastas próprias do app (derivadas do seu id de desktop, nunca de campos que um arquivo `.desktop` controla) e o arquivo com que ele foi aberto; a pasta de runtime tem só os sockets do Wayland e do som, então o barramento de sessão e o systemd do usuário ficam fora de alcance; depois o Landlock limita o resto. Vale a partir da próxima abertura. Apps GTK, Firefox e Chromium rodam sem suas integrações de D-Bus.

- **Notificações.** O `notifyd` pede ao barramento o PID de quem chama (só conta uma resposta enviada pelo próprio barramento), acha o escopo do app e descarta a notificação de um app restrito; um erro ou uma resposta atrasada também a descartam (falha fechada).
- **Microfone e câmera pelo PipeWire.** O guarda do PipeWire ( `sec/pwguard.c` ), o cliente de PipeWire próprio do `lord-server` , identifica cada cliente pelo PID que o PipeWire tirou das credenciais do socket e, para um app restrito, esconde os nós de captura e destrói seus streams de captura em menos de um segundo, incluindo uma gravação já em andamento. Enquanto um app restrito roda, um cliente cuja identidade não pode ser estabelecida é tratado como restrito.

**Limites.** O modo "perguntar" está projetado, mas hoje se comporta como o silencioso. O ICMP só é bloqueado por "sem rede"; conexões a uma porta já aberta não são conferidas. As restrições para apps Linux não cobrem:

- o BlueZ via D-Bus: um app com o Bluetooth restrito ainda pode pedir ao `bluetoothd` , pelo barramento do sistema, que procure, pareie e conecte;
- o barramento do sistema, que continua ao alcance mesmo com "sem acesso aos seus arquivos";
- a área de transferência;
- apps iniciados fora do lançador de apps, que não estão em nenhum escopo de app;
- os efeitos colaterais da união do `user@` : enquanto um app restrito roda, os serviços do próprio usuário ficam sujeitos à mesma união (o áudio de chamadas de um headset, que o PipeWire abre a cada chamada, para durante esse tempo);
- uma janela de poucos milissegundos em que um cliente novo de PipeWire de um app restrito tem todas as permissões, ou em que um stream de captura pode receber áudio antes de ser destruído.

## 6.7 Ajudantes que processam dados não confiáveis



O `lord-doc` (PDF, arquivos compactados, fontes, imagens do scanner), o `lord-media` (vídeo, áudio, fotos, câmera), o `lord-maile` (e-mail, calendários, contatos) e o `lord-vault` (segredos) compartilham um único sandbox reforçado, `sandbox_apply_helper()` :

- sem `socket` , `socketpair` , `connect` , `bind` , `listen` , `accept` ; sem `execve` ; `clone` só para threads; seccomp em todas as threads, e se o seccomp não puder ser aplicado o ajudante encerra;
- arquivos só por descritores passados pelo `lord-server` ; ioctls só por classe (DRM, V4L2, DMA-BUF), quando necessário;
- resultados devolvidos em memória compartilhada selada, validados pelo `lord-server` .

O `lord-media` nem sequer recebe o socket do PipeWire; o áudio passa por um anel compartilhado cujo fluxo pertence ao `lord-server` . O `lord-maile` não consegue criar sockets: ele pede ao `lord-server` , que só se conecta aos servidores do arquivo de contas (e, depois que o usuário pede para carregar imagens, só a endereços públicos nas portas 80/443 por dez minutos) e entrega o socket já conectado. O `build/sandbox-test helper` passa em 17 de 17 verificações.

## 6.8 Assinatura de pacotes e níveis de confiança

Os pacotes `.lpk` trazem um manifesto com o SHA-256 de cada arquivo e uma assinatura Ed25519 sobre ele. SHA-256 e Ed25519 são código próprio do LordOS (o Ed25519 segue a aritmética do TweetNaCl), testados com os vetores do NIST e da RFC 8032 (55 de 55). A verificação acontece em C na instalação e no carregamento. A Loja mostra o nível de confiança: **LordOS** (chave oficial, compilada dentro do binário para que nenhum arquivo possa acrescentar outra), **Autor verificado** (listado no catálogo assinado), **Assinado por <nome>** ou **Não assinado**, com um aviso. As atualizações precisam ser assinadas com a mesma chave. Um catálogo assinado traz uma lista de revogação. Se você editar um app instalado, o LordOS o marca como "Modificado por você", pode restaurar o original ou pode compartilhar sua versão assinada com sua própria chave, dando crédito ao original.

**Estado:** na 0.8.0 Beta, os pacotes oficiais ainda são assinados com uma chave de desenvolvimento, até a chave definitiva substituí-la. Um repositório de apps do LordOS assinado para a Loja ainda não foi construído.

## 6.9 Assinatura dos releases

Cada arquivo de release (o `.deb` , a ISO e a imagem do Raspberry Pi) aparece com seu SHA-256 no `SHA256SUMS` , e o `SHA256SUMS` é assinado com a chave de release do LordOS (Ed25519). A chave pública é `tools/release/lordos-release.pub.pem` , também embutida no instalador para Debian. Sua impressão digital (SHA-256 da chave pública em DER) é:

```
c7155b8e6bbbb180f907381994d06af76d333016cd6f3facf67cadd053ea108e
```

A chave privada fica só na máquina do mantenedor, nunca no repositório nem numa máquina de build. Cada release também traz uma lista de materiais de software (SBOM) SPDX 2.3 por arquitetura, listada no `SHA256SUMS` e portanto coberta pela assinatura, e suas notas saem do `CHANGELOG.md` , mantido no formato Keep a Changelog. A §11 mostra como conferir um download.

## 6.10 O cofre

O `lord-vault` guarda as senhas e o chaveiro da sessão em arquivos `.lvault` : uma chave de cofre aleatória de 32 bytes, envolvida uma vez por método de desbloqueio (senha com Argon2id, com custo calibrado por

máquina; FIDO2 `hmac-secret` com o cliente CTAP2 próprio do LordOS; arquivo-chave; senha mais arquivo-chave; chave de recuperação); conteúdo selado com XChaCha20-Poly1305, tendo o cabeçalho inteiro como dados associados; anexos selados separadamente; gravações atômicas com backup. Todo byte invertido e todo truncamento são detectados (1.165 de 1.165 no teste). Os segredos ficam numa área de memória travada com `mlock`, excluída dos core dumps e apagada ao ser liberada, num processo que não pode ser despejado (dump). A janela de Senhas roda no seu próprio processo em sandbox, então os valores revelados nunca entram no JavaScript do desktop. O chaveiro abre no login com a senha que o `lord-sessiond` verificou e atende os apps Linux pela implementação própria do Secret Service do LordOS. Chaves de segurança de hardware reais ainda não foram testadas.

## 6.11 Janelas protegidas

Uma janela marcada como protegida (como a de Senhas) é pintada de preto em todo caminho que copia pixels para fora: capturas de tela, gravações, miniaturas, a área de trabalho remota e os menus e a tela cheia do Wayland. O monitor real a mostra normalmente. Os apps Linux nem têm protocolo de captura de tela, para começo de conversa.

## 6.12 Divulgação e normas

Vulnerabilidades são relatadas como descreve o `SECURITY.md`, com um contato legível por máquina em `https://lordos.org/.well-known/security.txt` (RFC 9116). Os processos do LordOS estão **alinhados com, não certificados em**, ISO/IEC 29147 e ISO/IEC 30111 (divulgação e tratamento de vulnerabilidades), ISO/IEC 5230 (conformidade de licenças do OpenChain: o inventário em `THIRD_PARTY.md`, os avisos de licença e o SBOM) e ISO/IEC 18974 (garantia de segurança do OpenChain: o SBOM mais as atualizações de segurança do Debian para os pacotes de que o LordOS depende). O lordos.org publica as páginas legais e de confiança (política de segurança, assinatura de código, licenças, privacidade, ciclo de suporte, normas) e diz o que ainda falta: sem autocertificação formal, sem cruzamento automático do SBOM com bases de vulnerabilidades, sem SBOM das imagens completas e sem auditoria de terceiros.

## Arquivos-fonte

- [server/src/sec/sandbox.c](#), [server/src/sec/brokersrv.c](#), [server/src/sec/perms.c](#), [server/src/sec/cgroup.c](#)
- [server/src/netperm/np.bpf.c](#), [server/src/netperm/netperm.c](#), [server/src/fw/rules.c](#)
- [server/src/sec/restrict.c](#), [server/src/sec/pwguard.c](#), [server/src/hw/notifyd.c](#), [server/src/apps.c](#), [server/tests/broker\\_restrict\\_test.c](#)
- [server/src/sec/sign.c](#), [server/src/pkg/crypto.c](#), [tools/release/sign-release.sh](#), [tools/release/make-sbom.sh](#)
- [server/src/vault/vault.c](#), [server/src/vault/vfile.c](#)
- [server/src/media/capture.c](#)
- [SECURITY.md](#), [docs/COMPLIANCE.md](#), [CHANGELOG.md](#)

## 7. Desempenho

Todos os números abaixo foram medidos pelo projeto e estão documentados nos arquivos citados. A maioria vem de máquinas virtuais; as medições em hardware real estão indicadas como tal.

## 7.1 Memória e repouso

| Medição                                                                               | Valor                                                  | Onde                                      |
|---------------------------------------------------------------------------------------|--------------------------------------------------------|-------------------------------------------|
| Sistema inteiro, ISO live, em repouso depois de um minuto (4 GB, 4 núcleos)           | 465 MB em uso, CPU 99 % ociosa                         | VM117,<br><a href="#">docs/ISO.md</a>     |
| Sistema inteiro, instalado a partir da ISO, em repouso                                | 487 MB em uso, CPU 99 % ociosa                         | VM120,<br><a href="#">docs/ISO.md</a>     |
| Boot, ISO live com UEFI                                                               | 16,3 s (4,9 s de kernel + 11,4 s de espaço de usuário) | VM117,<br><a href="#">docs/ISO.md</a>     |
| <code>lord-server</code> sem placa com 3D (Mesa não carregado)                        | cerca de 32 MB de RSS                                  | <a href="#">docs/GPU.md</a>               |
| Uma sessão de usuário ( <code>lord-server</code> ), testes de área de trabalho remota | cerca de 50 MB; tela de login 41 MB                    | VM118,<br><a href="#">docs/REMOTE.md</a>  |
| Um app <code>.lpk</code> isolado                                                      | 13,7 MB de RSS                                         | VM122,<br><a href="#">docs/PRIVACY.md</a> |

## 7.2 Desenho

- Arrastar uma janela numa tela de 1280×800 pelo caminho da CPU: cerca de 6,7–6,9 ms de trabalho por quadro, por volta de 120 quadros por segundo (VM118, [docs/DISPLAYS.md](#) ).
- A mesma janela numa tela de 2560×1600 a 150 %: cerca de 5,1 ms por quadro.
- Vídeo: pintar direto do quadro decodificado quando só um vídeo opaco mudou levou o `lord-server` de 42 % para 13 % de um núcleo com um vídeo 1080p em janela, e de 92 % para 15 % em tela cheia ( [docs/ROADMAP.md](#) , mídia).
- O Monitor fechado não custa nada: a thread de amostragem não existe até ele ser aberto ( [docs/MONITOR.md](#) ).

## 7.3 GPU

Nas VMs só havia um renderizador por software disponível. Com ele, o caminho pela CPU sai mais barato que a composição "na GPU", e é por isso que o LordOS usa o caminho pela CPU quando o único renderizador é por software. O benefício real do caminho pela GPU (composição e apresentação quase de graça, tela cheia sem cópias) é esperado em placas reais, mas **hardware AMD e Nvidia real ainda não foi medido**.

## 7.4 O Raspberry Pi 5

O Raspberry Pi 5 é a única placa suportada. Resultados numa placa real (16 GB, cartão SD, sem monitor), de [docs/RASPBERRY.md](#) :

| Medição                                       | Valor                                                                                                             |
|-----------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| Compilar todos os binários do LordOS na placa | 1 min 02 s, 0 avisos                                                                                              |
| <code>make -C server test</code>              | 23 aprovados, 0 falhas, 0 pulados                                                                                 |
| Cópia e máscara com NEON vs. C puro           | 9,90 → 4,33 ms e 11,09 → 5,49 ms por tela (2,0–2,3×), idênticos bit a bit                                         |
| Em repouso na tela de login, sem monitor      | <code>lord-server</code> 36 MB de RSS, 0,6 % de CPU; sistema com 450 MB de 16 GB; 38–43 °C, sem limitação térmica |
| Gerar a imagem SD na placa                    | 8 min 18 s; 503 MB comprimida                                                                                     |
| Compilar o kernel do LordOS na placa          | cerca de 50 min por variante                                                                                      |

**O kernel do LordOS.** O kernel do próprio Raspberry Pi não inclui o Landlock. Sem ele, o teste hostil do ajudante de documentos falha em 5 das suas 25 verificações de sistema de arquivos e o teste do sandbox não consegue rodar. O LordOS compila o kernel do Raspberry Pi a partir do código-fonte e da configuração padrão dele, mudando só três opções (Landlock ligado, a lista de LSMs, o sufixo de versão), e o fixa para que um kernel mais novo do repositório não o substitua. A imagem também ativa o cgroup de memória, que o Raspberry Pi desliga no boot.

**Ventoinhas.** No Pi 5 num gabinete de teste, o `lord-fand` manteve a ventoinha desligada a 50–53 °C durante uma carga de 4 threads (o gabinete de alumínio funciona como dissipador), a levou a 55 % (4.500 RPM) e à velocidade máxima (7.400 RPM) com curvas de teste e restaurou as configurações do próprio kernel depois de `SIGTERM` e `kill -9` ( `docs/FANS.md` ).

**Ainda não verificado na placa:** tudo o que envolve um monitor conectado (configuração de modo, composição na GPU na tela, taxa de quadros), a sessão de desktop e o boot da imagem gerada.

## 7.5 PCs modernos

Ainda não há medições em PCs reais no repositório. A ISO é testada em máquinas virtuais (UEFI e BIOS); está planejado testá-la numa máquina AMD real a partir do pendrive live.

## Arquivos-fonte

- [docs/ISO.md](#), [docs/RASPBERRY.md](#), [docs/FANS.md](#), [docs/DISPLAYS.md](#)
- [tools/rpi/build-kernel.sh](#), [tools/rpi/build-img.sh](#)
- [server/src/canvas.c](#)

## 8. Experiência visual

- **Um guia de design.** Os apps seguem um guia comum ( `docs/DESIGN.md` ): uma grade de espaçamento de 8 px, uma escala tipográfica enxuta, alturas de controle consistentes e uma única ação na cor de destaque por

tela. Os tokens são exportados pelo SDK ( `spacing` , `typeScale` , `controlSize` ), e cada app passou por rodadas de acabamento documentadas, com antes e depois, nos modos claro e escuro, da direita para a esquerda e em janelas pequenas.

- **Temas.** Um tema é um desktop completo escrito em React: barra de menus, Dock, lançador de apps, Central de Controle e moldura das janelas, além dos seus tokens. O LordOS traz um tema padrão com barra de menus e Dock, e um tema Clássico com barra de tarefas e menu iniciar, que hoje é um modelo e vira uma segunda aparência completa antes da 1.0 (§10.3). Os temas trocam ao vivo; as configurações persistem.
- **Claro e escuro, cores de destaque.** Os dois modos e a cor de destaque se aplicam ao vivo aos apps nativos, aos apps isolados e, pelo portal e pelo GSettings, aos apps GTK.
- **Direita para a esquerda.** A interface inteira se espelha para árabe, hebraico e escritas parecidas, incluindo os apps isolados, que seguem o idioma da sessão ao vivo.
- **Lumo.** O mascote do LordOS é o Lumo, uma pequena raposa celestial, usado em telas de boas-vindas, estados vazios e material de marca, com um conjunto fixo de poses definido no guia da marca.
- **Acessibilidade hoje:** uma opção de "reduzir movimento" que desliga as animações decorativas, atalhos de teclado em toda parte e telas escaláveis até 200 %. Um leitor de tela, zoom e alto contraste estão no roadmap e **ainda não foram construídos**.

## Arquivos-fonte

- [docs/DESIGN.md](#), [docs/LORDOS-BRAND.md](#)
- [themes](#), [sdk/controls.jsx](#), [sdk/i18n.js](#)

## 9. Extensibilidade

- **O SDK** ( `@lordos/sdk` ) é a única coisa que os apps importam: janelas, configurações, tokens do tema, controles, arquivos, mídia, notificações, chaveiro, impressão e mais. Os apps não importam nada dos temas, então qualquer app funciona em qualquer tema.
- **Escrever um app** é uma pasta com `manifest.json` e `index.jsx`. `node tools/lpk.mjs <folder>` empacota sem React nem o SDK, acrescenta o ícone e as traduções e assina se houver uma chave; `keygen` , `sign` , `verify` e `catalog` cobrem o resto. Os apps podem incluir módulos WebAssembly e desenhar pixels diretamente com `<Image pixels>` .
- **Os temas** seguem um contrato fixo ( `Desktop` , `MenuBar` , `Dock` , `Launcher` , `ControlCenter` , `WindowFrame` , tokens), documentado em `themes/README.md` , com o tema Clássico como modelo.
- **A interoperabilidade com Linux** se dá por padrões: Wayland, o portal de configurações, o Secret Service, o serviço de notificações, os apps padrão do `mimeapps.list` e os arquivos `.desktop` .
- **Automação:** hoje não há API de scripts ou automação além do SDK. Integrações com assistentes de IA (por exemplo, pelo Model Context Protocol) **não existem no LordOS**; são apenas uma possível direção futura e teriam de respeitar a mesma regra de que o JavaScript não executa comandos.

## Arquivos-fonte

- [sdk](#), [apps/README.md](#), [themes/README.md](#)
- [docs/NATIVE-APPS.md](#), [docs/WASM.md](#), [examples/apps/fractal](#)

## 10. Roadmap e limitações

### 10.1 Onde as coisas estão

A 0.8.0 Beta é o primeiro lançamento público; a 0.7.0 foi um build interno e nunca foi publicada. Os marcos M1 a M9 estão completos: o servidor gráfico, o motor de interface, o SDK e os temas, o terminal, idiomas e escritas complexas, compatibilidade com apps Linux, pacotes e serviços do sistema, sessões, isolamento e o firewall, e os conjuntos de apps do M9 (documentos e impressão, mídia, apps de sistema, senhas, configurações de hardware, e-mail e calendário). Antes do lançamento, as restrições do firewall que não são de rede passaram a ser aplicadas para todos os tipos de app, as regras por domínio ganharam interface nos Ajustes e no Monitor, o boot ganhou uma tela de abertura e um tema de GRUB do LordOS, e os releases ganharam um SBOM assinado e um registro de mudanças. A ISO final foi gerada e testada em máquinas virtuais em 18 de outubro de 2026.

### 10.2 Limitações conhecidas

- Hardware real: GPUs AMD e Nvidia, monitores reais, notebooks (bateria, brilho da tela, tampa), controladoras Bluetooth, impressoras e scanners, chaves FIDO2 e provedores de e-mail reais não foram testados.
- O Raspberry Pi 5 é a única placa suportada, com o build do LordOS do seu kernel com Landlock; não foi testado com monitor, e a imagem gerada ainda não foi inicializada.
- Os apps Linux não ficam em sandbox por padrão; ainda não há Xwayland.
- Firewall: sem modo "perguntar"; as regras por domínio não veem apps que resolvem nomes com o próprio DNS criptografado.
- As restrições para apps Linux não cobrem o BlueZ via D-Bus, o barramento do sistema, a área de transferência nem apps iniciados fora do lançador de apps; a união do `user@` também sujeita os serviços do próprio usuário enquanto um app restrito roda; um cliente novo de PipeWire tem uma janela de poucos milissegundos antes de o guarda restringi-lo (§6.6).
- A chave de assinatura de `.lpk` ainda é uma chave de desenvolvimento, e ainda não há um repositório assinado de apps do LordOS.
- Permissões: sem "permitir uma vez"; o uso de câmera e microfone pelos apps Linux não é mostrado.
- O Secure Boot precisa estar desligado; a assinatura com um shim está pendente.
- Os downloads de pacotes usam HTTP (assinados e com hash).
- A área de trabalho remota não tem áudio.
- Acessibilidade (leitor de tela, zoom, alto contraste) não foi construída.

### 10.3 Próximos passos

A seguir, uma imagem do Raspberry Pi 5 inicializada e testada na placa real. Antes da 1.0, como planejado em `docs/ROADMAP.md` :

- **Xwayland**, iniciado sob demanda, com o compositor como gerenciador de janelas X e os apps X11 com a mesma identidade por app, o mesmo firewall e as mesmas restrições dos apps Wayland;
- **o tema Clássico como segunda aparência completa**: barra de tarefas, menu iniciar com busca, bandeja, ajustes rápidos, painel de notificações e encaixe de janelas, com os mesmos apps com outra pele, feito só com design e recursos próprios do LordOS.

Depois da 1.0: um app de virtualização para desktop construído sobre KVM e QEMU; um contêiner Android com LineageOS e apps da Loja do LordOS; o modo "perguntar" do firewall; acessibilidade; e um vídeo de demonstração completo.

## Arquivos-fonte

- [docs/ROADMAP.md](#), [CHANGELOG.md](#)

# 11. Como verificar o projeto você mesmo

## 11.1 Compile

Numa máquina de desenvolvimento com Node.js, empacote os temas, o SDK e os apps:

```
npm install && npm run build
```

No Debian 13, compile os binários e rode os testes:

```
sudo apt-get install -y --no-install-recommends build-essential
make -C server -j4
make -C server test          # every self-contained suite
make -C server test-full    # adds the helpers, the sandbox and the HarfBuzz comparison
```

O `make test` falha se alguma suíte falhar; uma suíte cuja ferramenta ou cujos dados não estão instalados informa "skipped" em vez de passar em silêncio. Os testes do sandbox precisam de um kernel com Landlock. A ISO e a imagem do Raspberry Pi são geradas como descrito em [docs/ISO.md](#) e [docs/RASPBERRY.md](#).

## 11.2 O que os testes cobrem

- **Terminal e texto:** o emulador VT, o shaping OpenType comparado ao HarfBuzz, o bidi Unicode comparado ao arquivo de teste oficial, os métodos de entrada, a separação de palavras em tailandês comparada à libthai.
- **Segurança:** o sandbox (app, rede, ajudante), a assinatura (vetores do NIST e da RFC 8032), o estado dos pacotes, as regras de caminhos de extração contra arquivos compactados maliciosos, as proteções do `lord.fs`, as regras do firewall (com 20.000 mutações de JSON e 30.000 operações aleatórias contra uma referência linear), o leitor de DNS (200.000 pacotes aleatórios), o compilador dos mapas do kernel (480.000 conexões) e o teste de integração com o kernel (18 de 18 numa VM).
- **Leitores:** e-mail, MIME, sanitizador de HTML, iCalendar, vCard e XML (30.953 verificações com fuzzing por mutação), o formato do cofre, os importadores, QR, CBOR, serialização D-Bus, o leitor da base de países, journal, TZif e IPP.
- **Motor e hardware:** layout e pintura, NEON vs. C, o driver de touchpad com gestos gravados, as ventoinhas em árvores sysfs simuladas, o arranjo de telas, o som e a suíte de especificação do WebAssembly.

Na geração da ISO final, o `make -C server test` informou 23 aprovados, 0 falhas, 1 pulado (dados de bidi não instalados).

## 11.3 Confira um release

Baixe o `SHA256SUMS`, o `SHA256SUMS.sig`, os arquivos do release e a chave pública ( `tools/release/lordos-release.pub.pem` ). Confira que a impressão digital da chave é `c7155b8e6bbbbb180f907381994d06af76d333016cd6f3facf67cadd053ea108e` e então, com o OpenSSL 3:

```
base64 -d SHA256SUMS.sig > SHA256SUMS.sig.bin
openssl pkeyutl -verify -pubin -inkey lordos-release.pub.pem -rawin -in SHA256SUMS -sigfile
sha256sum -c --ignore-missing SHA256SUMS
```

O SBOM de cada arquitetura ( `lordos_<version>_<arch>.spdx.json` ) também está no `SHA256SUMS`, então a mesma conferência o cobre.

## 11.4 Confira um sistema em execução

- `systemctl status lord-netperm lord-fand lordos` mostra os serviços; o teste da ISO confirmou os sete programas de rede anexados. Se o kernel recusar o programa de dispositivos ou o LSM, o `lord-netperm` informa `"devices":false` ou `"bluetooth":false`.
- `cat /proc/<pid>/status | grep -E 'Seccomp|NoNewPrivs'` num processo `lord-server --app` mostra o sandbox.
- `systemd-cgls` mostra a árvore `lordos-session-<uid>` pertencente ao root, com um escopo por app.
- Área de trabalho remota: `ss -ltn` não mostra nada na porta 5673 até você ativá-la em Ajustes → Compartilhar.

## 12. Licença e créditos

O código do LordOS é distribuído sob a **licença MIT**. Os componentes de terceiros mantêm suas próprias licenças, listadas em `THIRD_PARTY.md`: QuickJS e stb (incorporados como código-fonte), Monocypher (CC0/BSD-2), a lista de palavras da EFF (CC BY 3.0 US), a lista de senhas comuns (MIT), os dados de países do DB-IP Lite (CC BY 4.0), Natural Earth (domínio público) e as fontes Inter, JetBrains Mono e Plus Jakarta Sans (OFL). Cada release traz um SBOM SPDX do que o pacote inclui e daquilo de que depende ( `docs/COMPLIANCE.md` ).

O LordOS é criado pelo grupo **iPERFEX**: **Pablo Miranda** e **Federico Pereira**.

## 13. Glossário

- **Broker**: o único socket pelo qual um app isolado pede ao `lord-server` qualquer coisa que toque nas coisas do usuário; cada pedido é conferido em C.
- **cgroup (v2)**: um mecanismo do kernel que agrupa processos para contabilizar e limitar recursos. O LordOS dá a cada app seu próprio grupo, um "escopo", e o usa para identificar o app no firewall.
- **Compositor**: o programa que combina as janelas de todos os apps na imagem final da tela.
- **DRM/KMS**: as interfaces do kernel para dispositivos gráficos e para configurar modos de tela.
- **eBPF**: pequenos programas verificados que rodam dentro do kernel; o LordOS os anexa aos cgroups para decidir sobre conexões de rede, abertura de dispositivos e sockets Bluetooth.

- **evdev**: a interface do kernel para dispositivos de entrada.
- **LSM (BPF LSM)**: os ganchos de segurança do kernel; com o LSM **bpf**, um pequeno programa anexado por cgroup pode recusar uma operação, como criar um socket Bluetooth.
- **Namespace**: um recurso do kernel que dá a um processo sua própria visão de algo (usuários, montagens, ids de processo); o LordOS usa namespaces sem privilégios para os apps Linux restritos dos arquivos do usuário.
- **Ajudante**: um pequeno processo do LordOS que lida com dados não confiáveis ou segredos num sandbox mais rígido ( **lord-doc**, **lord-media**, **lord-maild**, **lord-vault** ).
- **Landlock**: um recurso do kernel que permite a um processo sem privilégios restringir quais arquivos e portas de rede ele, e seus filhos, podem usar.
- **.lpk**: o formato de pacote do LordOS para apps nativos instaláveis.
- **.lvault**: o formato de arquivo do cofre criptografado do LordOS.
- **Guarda do PipeWire**: o cliente de PipeWire próprio do **lord-server** que esconde os dispositivos de captura dos apps Linux restritos e interrompe seus streams de captura.
- **Powerbox**: um diálogo de arquivos desenhado pelo sistema que entrega ao app só o arquivo escolhido pelo usuário.
- **QuickJS**: um pequeno motor JavaScript escrito em C.
- **SBOM**: uma lista de materiais de software, a listagem legível por máquina dos componentes que um release contém, no formato SPDX.
- **Escopo**: um cgroup para um app, como **app-lordos-lpk-<id>.scope**.
- **seccomp-bpf**: um filtro do kernel que restringe quais chamadas de sistema um processo pode fazer.
- **Secret Service**: o padrão D-Bus do freedesktop que os apps Linux usam para guardar segredos.
- **Wayland**: o protocolo entre os apps gráficos e o compositor.

## 14. Para ler mais

| Capítulo              | Documentos em docs/                                                                                                              |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------|
| 1. Visão              | ARCHITECTURE.md, LORDOS-BRAND.md                                                                                                 |
| 2. Arquitetura        | ARCHITECTURE.md, NETWORK.md, FANS.md, SETTINGS.md, SYSTEM.md                                                                     |
| 3. Compositor         | GPU.md, DISPLAYS.md, WINDOWS.md, TRACKPAD.md, WAYLAND.md, SESSIONS.md, REMOTE.md                                                 |
| 4. Motor de interface | ARCHITECTURE.md, LANGUAGES.md, TERMINAL.md                                                                                       |
| 5. Aplicativos        | NATIVE-APPS.md, PACKAGES.md, WASM.md, FILES.md, MEDIA.md, DOCUMENTS.md, MAIL.md, ISO.md, INSTALL-DEBIAN.md                       |
| 6. Segurança          | PRIVACY.md, FIREWALL.md, MONITOR.md, SIGNING.md, PASSWORDS.md, MEDIA.md, REMOTE.md, COMPLIANCE.md, e <a href="#">SECURITY.md</a> |
| 7. Desempenho         | ISO.md, RASPBERRY.md, FANS.md, DISPLAYS.md, GPU.md                                                                               |
| 8. Experiência visual | DESIGN.md, LORDOS-BRAND.md, LANGUAGES.md, DOCK.md                                                                                |
| 9. Extensibilidade    | NATIVE-APPS.md, WASM.md, e <a href="#">apps/README.md</a> , <a href="#">themes/README.md</a>                                     |
| 10. Roadmap           | ROADMAP.md, RELEASING.md e <a href="#">CHANGELOG.md</a>                                                                          |