گرافیک در لینوکس، از X تا Wayland

پوست‌اندازی پنگوئن‌ها

» نویسنده: ایوان جنکینز » منبع: آرس تکنیکا » ترجمه: احمد شریف‌پور

در اوایل دهه ۸۰ میلادی، باب شیفلر (BobScheifler) دانشمند علوم کامپیوتر MIT شروع به پیاده‌سازی اصول یک سیستم پنجره‌سازی (WindowingSystem) جدید کرد. او تصمیم گرفت، این سیستم را X بنامد. زیرا این سیستم در واقع نمونه ارتقا یافته سیستم گرافیکی W بود که به‌صورت معمول در سیستم‌عامل V مورد استفاده قرار می‌گرفت. حتی خود باب هم در آن زمان تصور نمی‌کرد که سیستم پنجره‌سازی X که او و محققان همکارش در حال به‌وجود آوردن آن بودند تا ایجاد انقلابی در صنعت کامپیوتر پیش خواهد رفت. X تقریباً در تمامی سیستم‌عامل‌های مبتنی بر یونیکس به واسط گرافیکی استاندارد تبدیل شد، زیرا قابلیت‌ها و مفاهیمی که در آن ارائه شده بود، بسیار برتر از سایر رقبا بود. تنها چندین سال زمان لازم بود تا سیستم پنجره‌سازی X در کل جامعه یونیکس مورد پذیرش قرار گیرد. در این مقاله نگاهی خواهیم داشت به نحوه توسعه مجموعه عظیم گرافیک‌های لینوکس؛ از شکل‌گیری سیستم اولیه کلاینت/سرور X تا تلاشی مدرن که Wayland نامیده می‌شود.

آنچه X را این چنین متمایز می‌سازد، خود داستانی افسانه مانند است. X نخستین واسط گرافیکی بود که از راه‌حلی توزیع‌شده و مبتنی بر شبکه بهره می‌برد. X سروری که روی یکی از ماشین‌های اشتراک زمانی (Time Sharing) در حال اجرا بود، می‌توانست تصویر مربوط به پنجره‌های متعلق به هریک از کلاینت‌های محلی را تولید کند. X یک پروتکل نمایش روی شبکه را به گونه‌ای تعریف کرد که پنجره‌های یک ماشین می‌توانستند در یک ماشین راه‌دور دیگر به نمایش درآیند. در واقع بنا بود X در چنین محیط شبکه‌ای مورد استفاده قرار گیرد و این پروتکل کاملاً مستقل از سخت‌افزار بود. کلاینت‌های X که در حال اجرا روی یکی از انواع یونیکس بودند، می‌توانستند صفحه نمایش خود را از طریق شبکه به یک پلتفرم یونیکسی کاملاً متفاوت ارسال کنند. X همچنین ظاهر و نمای محیط را کاملاً از خود سرور مجزا کرد. به این ترتیب، پروتکل X تنها دستگاه‌های اشاره‌گر (نظیر ماوس) و المان‌های ابتدایی پنجره‌ها را تعریف می‌کرد و ظاهر واسط گرافیکی را به مجموعه‌ابزار مورد استفاده در ابزارک‌ها (widget toolkit)، مدیرهای پنجره (window managers) و محیط‌های دسکتاپ واگذار می‌کرد.

همزمان با پیشرفت توسعه X زیر نظر باب شیفلر و با حمایت مالی MIT، تولیدکنندگان بیشتری به آن علاقه‌مند شدند. رهبران بازار در آن زمان، یعنی شرکت‌هایی مانند DEC، موفق به دریافت مجوزی رایگان برای دسترسی به کد منبع X در جهت بهبود آن شدند. در این زمان بود که اتفاقی عجیب و نادر رخ داد. گروهی از تولیدکنندگان از MIT درخواست کردند تا در اعمال نوعی مدیریت در جهت حفظ یکپارچگی کد X همکاری کند. آن‌ها می‌خواستند، X در تمام جهان و توسط تمام گروه‌ها و شرکت‌ها قابل استفاده باشد. MIT با این پیشنهاد موافقت کرد و کنسرسیوم X در MIT به سرعت تشکیل شد و کدهای منبع X به‌صورت کامل و به همراه بهینه‌سازی‌های DEC منتشر شدند. انتشار کدهای X به واقع رویدادی غیرمعمول بود. جامعه تولیدکنندگان دریافته بودند که X به کالایی بسیار ارزشمند تبدیل شده است و این برای همه بهتر بود که آن را از تملک و کنترل شدن توسط یک شرکت واحد حفظ کنند. شاید باز شدن کدهای منبع X با اهمیت‌ترین رویداد در داستان شکل‌گیری X باشد و کنسرسیوم X در MIT هنوز هم حقوق مالکیت معنوی X را برای خود حفظ کرده است. یکی از توسعه‌دهندگان رده بالایی که توسط کنسرسیوم X استخدام شد، کیت پاکارد بود (شکل ۲) که در سال ۱۹۸۸ مأمور بازنویسی هسته سرور X شد. همان‌طور که در ادامه خواهیم دید، پاکارد در توسعه مجموعه گرافیک‌های لینوکس نقش برجسته‌ای خواهد داشت.

هرچند X حاکم مطلق مجموعه‌های گرافیک لینوکس و یونیکس بود، اما این نرم‌افزار مملو از امکانات و همه‌جا حاضر، سرانجام قربانی موفقیت خود شد. با رواج استفاده از لینوکس در دهه ۹۰ میلادی، X در پیکربندی‌هایی که سرور مستقل X و کلاینت آن هر دو روی یک کامپیوتر رومیزی در حال اجرا بودند، مورد استفاده قرار گرفت و به این شکل تقریباً به همراه تمام توزیع‌های لینوکس عرضه می‌شد. شفافیت شبکه‌ای (Network Transparency: این اصطلاح برای سیستم ها و نرم‌افزارهایی به کار می‌رود که برخورد آن‌ها با منابع محلی و منابع موجود روی شبکه یکسان است) X در چنین پیکربندی تک سیستمی، دیگر کاربردی نداشت و این قابلیت که زمانی بسیار برجسته و پرطرفدار بود، اکنون بار اضافه‌ای را به ترسیم‌های گرافیکی و ویدیویی تحمیل می‌کرد.

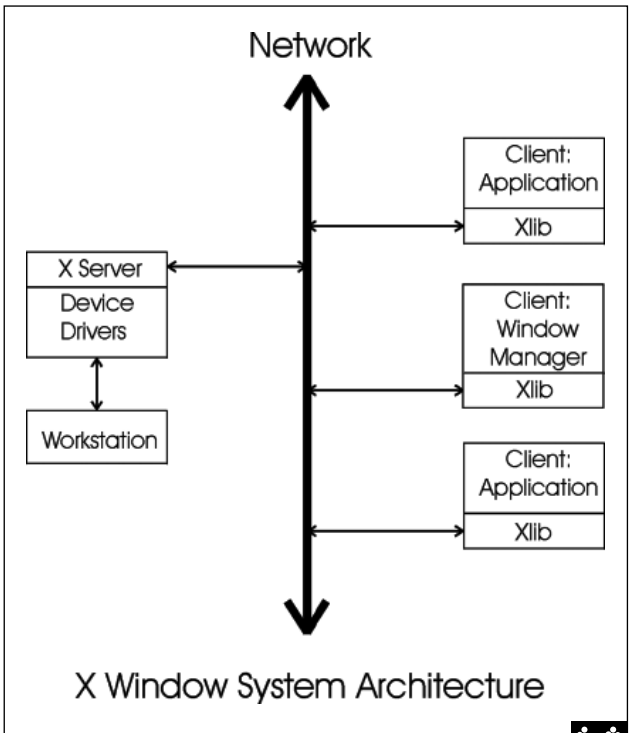
با افزایش میزان فروش کامپیوترهای شخصی در این دوره، پیچیدگی و برتری‌های سخت‌افزارهای گرافیکی اختصاصی (dedicated) به تدریج شروع به پیشی‌گرفتن از قابلیت‌های X کردند. روند پیشرفت و توسعه سخت‌افزارهای جدید و بهسازی شده مورد استفاده در کارت‌های گرافیکی بسیار شدید بود که این روند هم‌اکنون نیز حفظ شده است.

ظهور نقشه‌های جدول ترجمه (Translation Table Maps)

در سال ۲۰۰۴ برخی از توسعه‌دهندگان لینوکس به طرز فزآینده‌ای از سرعت آهسته توسعه X ناامید شده بودند. آن‌ها OpenGL را در اختیار داشتند، یک API رندر تصویر که در سال ۱۹۹۲ و از ابتدا برای تولید گرافیک دوبعدی و سه‌بعدی توسعه داده شده بود (OpenGL درواقع از نتیجه کارهای شرکت منحل شده سیلیکون گرافیکس منشعب شده بود). اما پس از سال‌ها تلاش برای افزودن قابلیت‌های سه بعدی به X جهت تعامل با قطعات گرافیکی، هنوز حتی فراخوان یک عملیات OpenGL ساده نیز از طریق لایه X امکان‌پذیر نبود.

پس از آن و در سال ۲۰۰۷ نور امید شروع به درخشیدن کرد. توماس هل‌استورم، اریک آن‌هولت و دیوید ایرلای ماجول مدیریت حافظه‌ای را توسعه داده و آن را نقشه جدول ترجمه یا به اختصار TTM نامیده بودند. TTM به منظور جابه‌جا کردن بافرهای حافظه مختص قطعات گرافیکی، بین حافظه آن قطعات و حافظه سیستم طراحی شده بود. استقبال گسترده جامعه لینوکس از این ماجول نشان از اهمیت زیاد آن داشت. چنین قابلیتی این امید را به ارمغان آورد که در نهایت کسی کار روی معضل ایجاد یک API برای مدیریت نیازهای سه بعدی برنامه‌های گرافیکی را شروع خواهد کرد. راهبرد این بود که بافر حافظه به یک شیء درجه اول تبدیل شود و به برنامه‌ها اجازه داده شود بافرهای حافظه مربوط به محتوای گرافیکی را تخصیص داده (Allocate) و آن‌ها را دستکاری کنند. TTM بافرهای تمام برنامه‌ها را روی ماشین میزبان مدیریت خواهد کرد و همگام‌سازی بین GPU و CPU را به عهده خواهد گرفت. این کار با استفاده از یک حصار (Fence) انجام خواهد شد. این حصار تنها یک سیگنال بود که نشان می‌داد، عملیات GPU روی بافر داده به اتمام رسیده است و بنابراین کنترل آن بافر می‌تواند دوباره به برنامه سازنده آن بازگردانده شود.

باید صادقانه اعتراف کرد، TTM تلاشی جاه‌طلبانه برای استانداردسازی نحوه دسترسی برنامه‌ها به GPU و همین‌طور یک برنامه مدیریت حافظه تمام و کمال برای همه درایورهای ویدیویی در دنیای لینوکس بود. به‌طور



شکل ۱ شمای کلی معماری سیستم پنجره‌سازی X

شکل ۲ کیت پاکارد – توسعه دهنده هسته X

خلاصه TTM سعی داشت، تمام عملیاتی را که ممکن بود یک برنامه گرافیکی به آن احتیاج داشته باشد، فراهم آورد. عوارض جانبی چنین رویکردی، حجم بسیار زیاد کدها بود. API ایجاد شده TTM بسیار بزرگ و سنگین بود، درحالی‌که هر درایور اپن‌سورس منحصر به‌فردی تنها به زیرمجموعه کوچکی از فراخوان‌های این API نیاز داشت. یک API بزرگ به معنای سردرگمی توسعه‌دهندگانی بود که مجبور بودند در این مجموعه بزرگ دست به انتخاب بزنند. بزرگ‌ترین شکایت نیز این بود که TTM مشکلاتی در زمینه سرعت و کارایی داشت که شاید به مکانیسم حصار بندی آن و سیستم ناکارآمد کپی کردن بافرها مربوط می‌شد. TTM می‌توانست تأمین‌کننده بسیاری از نیازهای برنامه‌ها باشد، اما اجازه نداشت که آهسته و کند باشد.

در این‌جا بود که کیت پاکارد دوباره وارد میدان شد. او در سال ۲۰۰۸ اعلام کرد، کار روی جایگزینی برای TTM در حال پیشرفت است. در آن زمان کیت برای اینتل کار می‌کرد و به کمک اریک آن‌هولت و با استفاده از درس‌هایی که از توسعه TTM آموخته بود، آن را بازنویسی کرد. این API جدید قرار بود GEM (سرنام Graphics Execution Manager) نامیده شود. غالب توسعه‌دهندگانی که این مطلب را می‌خوانند، به احتمال می‌توانند حدس بزنند که پس از آن‌چه اتفاقی رخ داد. زیرا توسعه‌دهندگان باتجربه می‌دانند، تنها چیزی که از داشتن شانس حل یک مشکل بزرگ از طریق نوشتن مجموعه‌ای عظیم از کدها بهتر است، امکان انجام دوباره آن است. GEM برتری‌های فراوانی بر TTM داشت. یکی از مهم‌ترین آن‌ها این حقیقت بود که API آن بسیار سبک‌تر بود و مفهوم مشکل‌آفرین حصاربندی از میان برداشته شده بود. کیت و اریک وظیفه قفل‌کردن بافرهای حافظه را از API خارج کرده و آن را بر عهده برنامه‌ها گذاشتند. این کار برای GEM این آزادی را به ارمغان آورد که بتواند بر مدیریت حافظه تحت کنترل GPU تمرکز کند و بتواند به کنترل محتوای اجرایی قطعه ویدیویی بپردازد. هدف این بود که تمرکز از مدیریت حافظه با جابه‌جایی بافرها برداشته شده و به مدیریت فراخوان‌های تابع ()ioctl در درون هسته منتقل شود. نتیجه نهایی این بود که GEM بیش از آن‌که یک مدیر حافظه باشد، به یک API جریان محور (streaming API) تبدیل شد.

GEM به برنامه‌ها این امکان را می‌داد که بافرهای حافظه را به اشتراک بگذارند. بنابراین لازم نبود، تمام محتویات فضای حافظه GPU دوباره بارگذاری شود. در یادداشت انتشار نسخه اصلی GEM آمده است:

«GEM مکانیسم ساده‌ای را برای مدیریت داده‌های گرافیکی و کنترل چرخه اجرا در درون سیستم عامل لینوکس فراهم می‌آورد. با استفاده از بسیاری از زیرسیستم‌های موجود هسته، GEM این کار را با حجم به نسبت کمی از کد انجام می‌دهد.»

معرفی GEM در می‌سال ۲۰۰۸ گامی امیدوارکننده در جهت پیشرفت مجموعه گرافیکی لینوکس بود. GEM سعی نداشت نقش همه چیز را برای همه برنامه‌ها بازی کند. به عنوان مثال، اجرای دستورات مربوط به GPU را به درایورهای مخصوص قطعات گرافیکی واگذار کرد. چون کیت و اریک در اینتل کار می‌کردند، برای آن‌ها طبیعی بود که GEM را مختص درایورهای اپن‌سورس اینتل طراحی کنند. آن‌ها امید داشتند، GEM تا حدی پیشرفت کند که از سایر درایورها نیز پشتیبانی کرده و از این طریق به‌طرز مؤثری سه تولیدکننده بزرگ GPU را تحت پوشش قرار دهد.

به هرحال تطابق یافتن GEM با درایورهای قطعات غیراینتلی به کندی پیش می‌رفت. برخی شواهد حاکی از این بود که درایور AMD با نمونه‌ای از «مدیر گرافیک TTM نزدیک به GEM» (GEMified TTM manager) سازگار شده بود و این نشانه‌ای بود از مقاومت در برابر انتقال کامل کدها به فضای GEM. دیگر GEM در خطر بود و آن خطر تبدیل شدن به رقابتی با تنها یک شرکت کننده بود.

هم TTM و هم GEM از طریق یکپارچه شدن با X برای دستیابی به قطعه گرافیکی و اجرای عملیات مختص GPU سعی داشتند مشکل شتاب‌دهی سه‌بعدی در مجموعه گرافیک لینوکس را حل کنند. هردوی آن‌ها تلاش می‌کردند به انبوه کتابخانه‌های گرافیکی نظیر OpenGL (که به X وابسته است)، Qt (وابسته به X) و +GTK (آن هم وابسته به X) سر و سامانی بدهند. مشکل این جا بود که X بین تمام این کتابخانه‌ها و هسته قرار داشت و هسته تنها مسیر دسترسی به درایورهای قطعات و در نهایت GPU است.

X قدیمی‌ترین جنگ‌جوی میدان است و اصرار دارد با همه نبرد کند. X چندمیلیون خط کد دارد، اما بسیاری از آن‌ها در گذشته‌های دور نوشته شده‌اند، یعنی در زمانی که GPUها وجود نداشتند، ترانزیستورهای قابل برنامه‌نویسی مختص انجام سایه‌زنی، چرخش و ترجمه رأس‌ها (vertexes) وجود نداشتند. سخت‌افزار هیچ تصوری از oversampling و interpolation برای کاهش میزان aliasing نداشت و حتی قادر نبود فضاهای رنگی بسیار دقیق را به‌وجود آورد. به این ترتیب، زمان استراحت این جنگ‌جوی سالخورده فرا رسیده بود.

Wayland: مدیر نمایش جدید

در سال ۲۰۰۸ یک مهندس نرم‌افزار به نام کریستین هوگزبرگ (Kristian Hogsberg)، در حال رانندگی در حومه شهر بوستون بود. شاید مقصد وی رسیدن به محل کار یا بازگشت به خانه بود. مهندسان نرم‌افزار در دنیایی به شدت درون‌گرا زندگی می‌کنند. آن‌ها بیشتر روز را صرف حل مسائل پیچیده، شکستن مسائل به بخش‌های ریزتر و بازسازی دوباره آن می‌کنند. اغلب هنگامی که حواس آن‌ها پرت می‌شود، ذهن ناخودآگاه آن‌ها شروع به فعالیت‌های عجیب و غریب می‌کند و ایده‌های نیمه پخته را با الگوریتم‌های نیمه کاره پیوند می‌زند. این اتفاق ممکن است در نامربوط‌ترین مکان‌ها، مثلاً در حمام، آشپزخانه و در حین پختن غذا یا رانندگی رخ دهد. هنگامی که هوگزبرگ در حال عبور از روستای کوچک ویلند (Wayland) در ماساچوست بود، ایده‌ای که در ذهنش جوانه زده بود، متبلور شد. بهتر است این ایده را از زبان خود وی بشنویم:

«ایده اصلی این بود که تمام پنجره‌ها دوباره به مسیر دیگری هدایت (redirect) شوند، ما می توانیم تمام عملیات رندر را در سمت کلاینت انجام دهیم و یک نقطه ارجاع (Handle) بافر را به سرور ارسال کنیم و مدیر ترکیب‌بندی (Compositing Manager) در سرور نمایش اجرا خواهد شد. یکی از اهداف این کار اجرای یک سرور X روی Wayland است. ابتدا در

حالت تمام صفحه (مانند Xnest) و سپس در حالت بدون والد (rootless). زیرا X به این زودی میدان را خالی نخواهد کرد.»

ایده او نوشتن یک مدیر نمایش کاملاً جدید و وادار کردن آن به ارسال مستقیم خروجی سه بعدی به هسته و در نتیجه پشت سر گذاشتن X بود. یکی از کلاینت‌های این مدیر نمایش جدید خود X خواهد بود. همان‌گونه که گفته شد این مدیر پنجره جدید برای بزرگداشت نام روستایی که این ایده حین عبور از آن به ذهن هوگزبرگ خطور کرده بود، Wayland نامیده شد.

یک ایده، چیزی جز یک ایده نیست. هر روز ایده‌های درخشان زیادی به ذهن بسیاری از افراد خطور می‌کند، اما این ایده‌ها در هیاهوی زندگی ناپدید می‌شوند. هوگزبرگ روی کتابخانه‌های رندر کار می‌کرد و به احتمال به این فکر می‌کرد که دنیایی که در آن برنامه‌ها بتوانند به طور مستقیم و بدون دخالت X سالخورده، با GPU تعامل داشته باشند چه دنیای هیجان‌انگیزی خواهد بود. او تصمیم گرفت که کدها را بنویسد و این رؤیا را به واقعیت تبدیل کند. به گفته کیت پاکارد، سرور اولیه وی ظرف مدت دو هفته شروع به کار کرد.

یکی از قابلیت‌های کلیدی Wayland استفاده از APIهای رندری است که هیچ وابستگی به X ندارند. برای حفظ سازگاری، خود سرور X نیز به کلاینتی برای wayland تبدیل شد و تمام رندرهای X به طور مستقیم به wayland خورانده خواهد شد. بسته wayland نیز درست همانند همتای قبلی‌اش X تنها یک پروتکل را تعریف می‌کند. معماری wayland، با توانایی کار در کنار X راهی ساده را برای مهاجرت کلاینت‌های کنونی X و حتی کلاینت‌هایی که برای آینده و بر مبنای X طراحی شده بودند، فراهم می‌آورد. سرور X می‌تواند همانند سابق به فعالیت ادامه دهد و از تمام کلاینت‌های قدیمی پشتیبانی کند.

مدیر نمایش wayland؛ اگر بتوانیم آن را این‌گونه بنامیم، از نیروی مدیر اجرای GEM، evdev (درایورهای ورودی) و تعویض حالت هسته یا kms (سرنام kernel switching mode) که پیش از این در هسته لینوکس وجود داشتند استفاده می‌کند. Wayland مدیر ترکیب‌بندی خودش را دارد که در تضاد کامل با X قرار دارد. زیرا X برای کار با تغییرات بافر حافظه بر سیستم‌های ترکیب‌بندی خارجی تکیه می‌کند.

Wayland از DRI2 نیز بهره می‌برد. هم کامپوزیتور wayland و هم کلاینت‌های wayland یک handle برای دسترسی به یک بافر حافظه اشتراکی از وضعیت واقعی صفحه نمایش در اختیار دارند. کلاینت از طریق DRI2 رندر می‌شود و از APIهای سه بعدی استفاده می‌کند. زمانی‌که بافر توسط کلاینت تغییر کرد، کامپوزیتور Wayland نسخه‌ای را که از میزکار در اختیار دارد، به روز می‌کند و صفحه را دوباره ترسیم (Redraw) می‌کند.

به یقین رویکرد wayland بسیاری از مسائلی را که به‌صورت سنتی در X با مشکل مواجه می‌شدند، حل خواهد کرد. این پروژه به سادگی انسان را هیجان‌زده می‌کند؛ نبوغی در پس این کدها نهفته است و پروژه توسط اینتل و ردهت پشتیبانی می‌شود.

اما موانع اندکی باید از سر راه برداشته شوند تا لینوکس بتواند ادعا کند که مجموعه گرافیکی سه بعدی مدرنی در اختیار دارد و دو مانع بزرگ در این میان توسعه درایورهای اپن‌سورسی است که بتوانند با کارت‌های گرافیکی ان‌ویدیا و AMD کار کنند. سومین تولیدکننده بزرگ کارت‌های گرافیک، یعنی اینتل در این میان از موقعیت خوبی برخوردار است، زیرا ماجول هسته GEM با در نظرگرفتن درایورهای اینتل نوشته شده و Wayland پیشاپیش با این درایورها سازگار است.

اما چه کسی درایورهای اپن‌سورس AMD و ان‌ویدیا را به روز خواهد کرد؟ توسعه درایورهای اپن‌سورس در لینوکس، به‌ویژه برای کارت‌های گرافیکی، همواره یکی از کابوس‌های توسعه‌دهندگان‌بوده است. چنین کاری معمولاً به‌واسطه سروکار داشتن با مشخصات سخت‌افزاری ناقص یا حتی عدم وجود مشخصات سخت‌افزاری، همواره به یک نتیجه ختم خواهد شد: مهندسی معکوس دستگاه موردنظر.

درایورهای Nouveau نمونه مناسبی از این درایورها است. در ابتدا از سوی توسعه‌دهندگان ان‌ویدیا اعلام شد، هیچ برنامه‌ای برای پشتیبانی از Wayland وجود ندارد. اما کار در جامعه لینوکس آغاز شده بود؛ گاهی با حمایت سازندگان و گاهی بدون حمایت آن‌ها. در پروژه Nouveau یک درایور ان‌ویدیا از طریق روند مهندسی معکوس با قدرت تمام در حال توسعه است. برنامه‌ای به نام Renouveau (سرنام Reverse Engineering for nouveau) این عملیات را به ترتیب انجام خواهد داد:

- ثبت محتویات رجیسترهای MMIO دستگاه
- انجام پاره‌ای ترسیمات گرافیکی
- ثبت مقادیر جدید رجیسترها

پس از آن تفاوت بین مقادیر ثبت شده حافظه، به‌صورت یک فایل متنی به سرورهای ftp مربوط به Renouveau فرستاده می‌شود، جایی‌که فایل‌ها برای آنالیزهای بعدی در دسترس توسعه‌دهندگان قرار می‌گیرد.

در این زمینه AMD بسیار مهربان‌تر از ان‌ویدیا ظاهر شده است. در طی چند سال گذشته، AMD یک تیم توسعه درایور برای نوشتن درایورهای اپن‌سورس برای سخت‌افزارهای خودش تشکیل داده است. آن‌ها همچنین به‌صورت دوره‌ای مشخصات سخت‌افزارهایشان را منتشر می‌کنند تا توسعه اپن‌سورس بتواند به کار خود ادامه دهد. نام این درایور fglx (سرنام FireGL and Radeon for X) است و جامعه لینوکس می‌تواند به‌روز رسانی‌های دوره‌ای (معمولاً ماهانه) را از AMD دریافت کند. Wayland توسعه‌ای جدید و امیدوارکننده برای وضعیت فعلی مجموعه گرافیکی لینوکس است. به تازگی اوبونتو نیز وارد عمل شده و اعلام کرده آن‌ها نیز در حال برنامه‌ریزی برای استفاده از Wayland به همراه مدیر پنجره جدید خودشان، یعنی یونیتی هستند. اینتل نیروی عظیمی را صرف پشتیبانی و حمایت از توسعه GEM کرده و توسعه‌دهندگان Wayland را به کار گرفته است تا اطمینان حاصل کند که کارایی و سرعت درایورها همپای معماری گرافیکی آن‌ها پیش خواهد رفت.

باقیمانده دنیای گرافیک، به شدت قطبی شده است. AMD و ان‌ویدیا درگیر رقابتی همه‌جانبه برای کسب سهم بیشتری از بازار هستند و به نظر نمی‌رسد، نیازهای جامعه اپن‌سورس در رده‌های نخست فهرست برنامه‌های آن‌ها جایگاهی داشته باشد.

جامعه لینوکس برپایه همکاری و تشریک مساعی بنا نهاده شده است. از طریق این همکاری و نگرش باز، در طی سال‌ها به شدت رشد کرده است. به نظر می‌رسد که مجموعه گرافیک‌های لینوکس آماده جهشی بزرگ شده است. تولید کنندگان سخت‌افزارهای گرافیکی آزادند که میزان مشارکت خود را به دلخواه انتخاب کنند و این بسیار تعجب‌آور است که آن‌ها چقدر نسبت به این امر بی‌میل هستند.

آیا هیچ تولیدکننده‌ای به‌ویژه اگر ادعا داشته باشد که محصولاتش قدرتمندترین سخت‌افزارهای دنیا هستند، نمی‌خواهد که کاربران بهترین تجربه ممکن را از کار با محصولاتش کسب کنند؟ با عدم انتشار اطلاعات آیا آن‌ها واقعاً به خط تولید خود زیان نمی‌رسانند؟ تنها یک چیز مسلم است؛ شما این کار را «همکاری» نمی‌نامید.