Linux grafika od zahteve do prikaza

Matic Poženel

Junij 2024

1 Uvod

Po predmetu Vgrajeni sistemi sem postal precej navdušen nad programiranju mikroprocesorjev, predvsem zaradi enostavnosti programiranja. Pomnilniško preslikani naslovi omogočajo preprosto interakcijo z vhodno-izhodnimi napravami (V/I napravami) - potrebno je poznati le specifikacijo in pomen bitov v njenih registrih. Ob programiranju na višjem nivoju (native aplikacije z grafičnim vmesnikom) se mi je zato začelo porajati vprašanje - zakaj je potrebno za vsak programski jezik mukotrpno iskati grafični vmesnik, ki bi zadovoljil potrebe programerja? Še več - ali bi morda lahko spisal svoj grafični vmesnik, ki bi bil prilagojen mojim potrebam? Konec koncev je za delovanje računalnika zadolžen procesor in ta more na tak ali drugačen način sporočiti zaslonu, naj na koordinati (X,Y) aktivira piksel z neko določeno barvo.

2 Grafični sklad

Grafični sklad je nabor programskih komponent, ki nadzirajo delovanje naprav (npr. gonilniki) in nudijo vmesnik za programiranje (API) z namenom prikaza grafičnih elementov na izhodni napravi. Za grafični sklad sicer ni predpisane enotne oblike, a večinoma so plasti, ki ga sestavljajo, razporejene na sledeč način (nivo najbližje strojni opremi je na lestvici najnižje):

1. Aplikacijska plast (Application layer)
2. Interoperacijske plasti (Interoperation layer)
   2.1. Plast za nadzor namizja (Desktop Management Layer)
   2.2. Plast za nadzor oken (Window Management Layer)
3. Predstavitvene plasti (Presentation Layers)
   3.1. Kompozicijska plast (Compositing Layer)
   3.2. Orodna plast (Widget Toolkit Layer)
   3.3. Upodobitvena plast (Rendering Layer)

4. Prikazna plast (Display Layers)
   4.1. Gonilniška plast (Device Driver Layer)
   4.2. Strojna plast (Hardware Layer)

2.1 Prikazna plast

2.1.1 Strojna plast

V strojni plasti najdemo strojno opremo, torej fizične naprave, ki jih uporabljajo višje plasti

2.1.2 Gonilniška plast

Gonilniška plast vsebuje gonilnike, ki omogočajo delovanje bodisi specifičnih prikaznih naprav (video pomnilnika, GPE, zaslona ipd.), bodisi z neko pomnožico le-teh.

2.2 Predstavitvene plasti

2.2.1 Upodobitvena plast

Splošno pravilo upodobitvene plasti je, da se po nekem algoritmu iz navodil, matematičnih funkcij in podobnih abstraktnih konstruktov pridobi sliko (tj. neko grafično celoto), ki se bo izrisala na zaslon, in sicer prek posebnega programa - upodobitelja. Tu nastane razkorak med 2D in 3D grafiko, saj se je potrebno, na primer, odločiti, s katerim principom se bodo izrisovala okna na zaslon. Čeprav se morda zdi, da bi lahko za vse probleme uporabljali le 3D upodabljanje (saj lahko na 2D gledamo kot na poseben primer 3D), je to računsko veliko bolj intenzivno in se na starejši strojni opremi ne obnese tako dobro, kot klasično 2D upodabljanje.

2.2.2 Orodna plast

Orodna plast večinoma skrbi za izris grafičnih kontrolnih elementov (gumbi, meniji, vnosna polja itd.), ki jih uporablja nadzornik oken. Ta plast je večinoma kar združena z upodobitveno plastjo. Grafični kontrolni elementi pa morajo (večinoma) delovati s kompozitorjem.

2.2.3 Kompozicijska plast

V kompozicijski plasti se nahaja poseben program, imenovan kompozitor, ki skrbi za ustrezno razporeditev (kompozicijo) grafičnih elementov iz spodnjih plasti na zaslonu. Za 2D grafiko to plast nemalokrat kar enači z upodobitveno (z upodobitvene plasti), pri 3D grafiki pa je skoraj obvezna.

2.3 Medoperacijske plasti

2.3.1 Plast za nadzor oken

Grafično procesiranje in prikaz , kot sta bila razložena do te točke, se odvijata tako rekoč na enem platnu, poimenovanem "okno". Upravljalec oken jih lahko izriše več in določi, na katerem se bo prikazala posamezna upodobljena komponenta. Prav tako je zadolžen za razporejanje oken, morebitno prekrivanje ipd. Tu se že srečamo z nekoliko bolj poznanimi (tržnimi) imeni, kot so AwesomeWM, qtile, i3wm in drugimi.

2.3.2 Plast za nadzor namizja

Upravljalec namizja je večinoma tista komponenta, na katero pomislimo, ko govorimo o grafičnih uporabniških vmesnikih (GUI-jih). Tudi tu srečamo poznana imena, kot so KDE, Gnome, XFCE, Cinnamon, MATE in drugi.

2.4 Aplikacijska plast

Aplikacijska plast je interna programu samemu.

V operacijskem sistemu Linux

3 Strojna oprema

Brez vhodno/izhodnih (V/I) naprav nam računalniki bore malo koristijo. Če poenostavimo: želimo si imeti nek način vnosa podatkov v pomnilnik, ter nek način pridobivanja podatkov iz pomnilnika tako, da bodo človeku berljivi - to pomeni, da želimo podatke pridobiti v grafični (besedilo, vizualizacije...), zvočni ali pa v kaki drugi, zaenkrat precej futuristični obliki. Ker se v pričujoči predstavitvi osredotočamo na grafično obliko, bomo izmed perifernih naprav zagotovo potrebovali zaslon. Sam način vhoda in izbira vhodnih perifernih naprav za potrebe te predstavitve ni bistvena (razen v določenih delih). Vprašanje se torej glasi - kako pripravimo procesor, da nam na zaslon izriše nekaj pikslov? Še bolje: kako to ukažemo Linux jedru[1]?

3.1 Fizični medpomnilnik okvirjev (frame buffer device)

Če preskočimo prikaze, osnovane na znakovnih celicah[2], so Linuxovi sistemi v začetku podpirali medpomnilnike okvirjev (MO, frame buffer). Sprva je bila to fizična naprava, ki je, kot je moč razbrati iz imena, hranila naslednji okvir (sliko oz. polje pikslov), ki se bo izrisal na zaslon v naslednji iteraciji. Tako kot sodobne grafične kartice je bil na napravi vgrajen vmesnik, na katerega je bil preko ustreznega kabla priključen zaslon. Na tej točki v zgodovini so imele uporabniške aplikacije možnost neposrednega pisanja v registre MO - po eni strani so tako oblikovale željen okvir, po drugi strani pa so imele možnost nastavljanja načina delovanja naprave oz. njen mode (ang. mode-setting). To

je imelo nesrečen stranski vpliv - MO je bilo moč poganjati izven njegovih zmožnosti, kar je povzročilo fizično škodo ne le na njem, temveč tudi na zaslonu[3].

3.2 Grafične procesne enote

Uporaba MO pa je bila poleg teh varnostnih hib precej neučinkovit pristop za upodabljanje 3D grafike. Upodabljanje, ki sloni skoraj izključno na linearni algebri (matričnih operacijah), je bilo še vedno prepuščeno procesorju. Tako so se začele pojavljati prve grafične procesne enote (GPE, angl. GPU).

Grafične procesne enote sestojijo iz enakih komponent, kot centralne procesne enote: oboje imajo aritmetično-logične enote, nadzorne enote in več nivojev predpomnilnikov.[4]

Razlikujeta pa se v številu in zmogljivosti posameznih komponent, in nenazadnje seveda v samem namenu. CPE je splošnonamenski - s serijskim izvajanjem lahko stori vse, a morda nekoliko počasneje (npr. množenje matrik). GPE je namenjen predvsem za linearno algebro in paralelno računanje. Sestavljen je iz velikega števila jeder, ki računajo vzporedno druga z drugo. Hitrost grafičnih procesnih enot merimo v številu operacij v plavajoči vejici na sekundo (FLOPS, floating point operations per second), danes navadno v teraflopsih.

3.3 Grafične kartice

GPE je lahko integriran (na istem vezju kot CPE), ali pa je na voljo kot samostojna naprava, ki ji pravimo grafična kartica. Integriran GPE si deli pomnilnik s CPE-jem, grafična kartica pa vsebuje svoj pomnilnik, imenovan video pomnilnik (VRAM), ki je namenjen izključno GPE in se po zasnovi nekoliko razlikuje od SDRAM-a (pogosto je to SGRAM oz. GDDR SDRAM, Graphics double data rate synchronous dynamic random-access memory). Na grafični kartici najdemo še vmesnik, na katerega lahko priključimo zaslon - ta je bil doslej povezan bodisi z MO-jem ali pa matično ploščo.

GPE na VRAM-u shranjuje različne elemente grafičnega upodabljanja (npr. teksture)[5] in pa medpomnilnik okvirjev ter FIFO vrsta ukazov. MO pošilja okvirje na vmesnik za zaslon, uporabniške aplikacije pa izstavljajo ukaze na FIFO vrsto in konfigurirajo napravo.

Na tej točki (v zgodovini in v predstavitvi) še vedno obstaja problem neposrednega dostopa do grafične strojne opreme s strani programov. Tako se lahko zgodi, da v MO piše več aplikacij naenkrat; enako velja za konfiguracijo naprave[6]. Nalogo reševanja tega problema nosi jedro operacijskega sistema.

4 *Primer*: uporaba virtualnega medpomnilnika okvirjev v Linuxu

Ker je MO zastarel in ni več v uporabi, ga danes na sistemih Linux ne uporabljamo več na enak način (tj. vsaj ne z napravo fbdev[7])[8].

Kljub temu obstaja virtualni medpomnilnik okvirjev (VMO), ki je še vedno uporabljen v današnjih Linux sistemih. S perspektive programerja gre za navadno datoteko oz napravo /dev/fb∗, kjer znak ∗ ponazarja številko VMO - na večini sodobnih distribucij, kot je na primer Linux Mint, obstaja le /dev/fb0. Tega lahko tudi kot programerji neposredno uporabimo. Poglejmo si primer na omenjenem operacijskem sistemu. Predpogoj za uporabo fb0 je, da se iz grafičnega vmesnika prestavimo v enega izmed teletipkalnikov[9] (teletypewriter). Okenski sistem X ob zagonu namreč prevzame absoluten nadzor nad VMO in nam tako onemogoči neposredno rabo.

Po opravljenem preklopu v na primer tty2 lahko preprosto prevedemo in poženemo naslednjo kodo:[10]

Strukturi fb_fix_screeninfo in fb_var_screeninfo[11] nosita fiksne in spremenljive informacije o zaslonu. Fiksne morda niso tako zanimive s perspektive programiranja grafike (so pa morda zanimive s perspektive razvoja operacijskih sistemov[12]), razen polja line_length, ki nosi število pikslov v eni vrstici. Druga struktura je veliko bolj zanimiva:

Telo funkcije main() torej pridobi fiksne in spremenljive podatke. Slednji vsebujejo tudi x in y resolucijo zaslona, odmik na x in na y ter število bitov na piskel. Iz tega lahko izračunamo velikost enega okvirja zaslona v bajtih in določimo pomnilniški naslov (v VMO!), kamor bomo v i-ti iteraciji postavili i-ti piksel. Seveda moramo prej VMO še preslikati v pomnilniški prostor CPE, kar storimo s sistemskim klicem mmap. Sedaj lahko VMO tretiramo kot binarno datoteko - podobno, kot bi tretirali BMP datoteko. V dvojni for zanki se pomikamo piksel za pikslom in vsakič spremenimo količino rdeče, zelene in modre barve (ter prosojnost). Na koncu le čakamo na vnos nekega znaka, da se nam program ne zapre prehitro.

Če si ogledamo še funkcijo get_screeninfo(), ki pridobi podatke, lahko opazimo, da to stori prek sistemskih klicev ioctl (). Ti bodo ponovno prišli v upoštev v poglavju DRM in DRI. Za argumente vzamejo datotečni deskriptor, strojnoodvisen ukaz in dodatne argumente (v tem primeru strukturo, kamor se zapišejo pridobljeni podatki.)

5 Grafični sklad operacijskega sistema Linux

Jedro Linux operacijskega sistema nima orodij, ki bi omogočala izris kompleksnih grafičnih elementov na zaslonu, kot so okna, animacije in podobno. To pomeni, da se implementacija grafike razlikuje od distribucije do distribucije. V

mnogih, kot so Ubuntu, Mint, openSUSE in Fedora, je določena implementacija grafičnega sklada že vgrajena v samo distribucijo, v lažjih ("lightweight") distribucijah, kot sta Arch in Gentoo, pa mora uporabnik sam dodati gonilnike, upravljalca oken, kompozitor in druge gradnike.

5.1 Linux jedro in gonilniki naprav

5.2 Grafični API

Prva komponenta v grafičnem skladu operacijskega sistema Linux je grafični gonilnik, ki ga pogosto imenujemo kar grafični API. Zadolžen je za prevedbo visokonivojskih ukazov, ki so na voljo programerju, v strojno kodo, razumljivo grafičnemu procesorju. V to kategorijo spadajo grafični API-ji, kot so OpenGL, Vulkan in Direct3D.

5.2.1 OpenGL in grafični gonilniki

OpenGL in njegova "nadgradnja", Vulkan, sta standarda oz. specifikaciji[14]. V prvi vrsti je namenjen izdelovalcem grafičnih kartic, saj po teh navodilih izdelajo gonilnike za svojo opremo. Tako ne OpenGL-a in Vulkana ne moremo opredeliti kot "knjižnjici", temveč kot "set navodil", katerim zadošča več gonilnikov in implementacij za klienta.

Projekt Nouveau je primer zbirke odprtokodnih gonilnikov za Nvidia grafične kartice izdelanih s pomočjo Gallium3D API-ja. To pomeni, da so vsi Nouveau gonilniki po OpenGL standardu, za dostop do njih pa uporabimo knjižnice na višjih nivojih, kot je Mesa.

5.2.2 Mesa

Mesa je ena izmed bolj poznanih klientskih implementacij OpenGL (in Vulkan) standarda. Gre za dejansko grafično knjižnico, ki deluje z večino gonilnikov današnjih grafičnih kartic. Upravljalca oken X.org in Wayland uporabljata OpenGL standard, zaradi česa oba uporabljata Mesa implementacijo. Prvi ponuja vmesnik med okenskim sistemom X in OpenGL sceno.

5.2.3 Povezava med upravljalci oken in OpenGL

Standard OpenGL je namenjen upodabljanju 2D in 3D grafike in tako ne nudi funkcij za okna. Z uporabo GLX podaljška (GLX extension) okenskega sistema X lahko posamezno OpenGL sceno vežemo na neko X okno. Na Windows sistemih lahko namesto GLX uporabimo vmesnik WGL.

Ker odvisnost od upravljalca oken oteži programiranje prenosljivih aplikacij, so avtorji standarda OpenGL zasnovali še vmesnik EGL, ki povezuje poljubnega upravljalca oken z upodobitvenim API-jem.

5.3 DRM in DRI

Tako OpenGL kot upravljalec oken izdata nek nabor ukazov odvisnih od konkretne GPE. Te je torej potrebno nekako spraviti od programske rešitve do dejanske strojne opreme. Za to skrbi posebna komponenta Linux jedra, neposredni upodobitveni upravljalec (DRM, direct rendering manager), ki prejete ukaze posreduje GPE. Do njega lahko v Linux sistemih dostopamo preko knjižnice libdrm.

DRM uporablja sistemske klice ioctl , da alocira pomnilnik in nastavi parametre GPE. Sistemski klici ioctl [16] so namenjeni za vhodno-izhodne operacije za različne naprave - v tem primeru za GPE.

Visokonivojske aplikacije torej uporabljajo orodje Mesa, ki je zmožno prek knjižnice libdrm (z uporabo sistemskih klicev) direktno komunicirati z Linuxovim jedrom

Vsaka visokonivojska aplikacija se na orodje Mesa poveže posredno prek okenskega sistema X. Da lahko ta posodobi prikaz, se poveže neposredno z DRM DRI je ogrodje, ki večim uporabniškim programom omogoča hkratni dostop do grafične strojne opreme brez povzročanja konfliktov.

5.3.1 KMS

Za vso grafiko, ki ne uporablja upravljalca oken, uporabimo Linuxov Kernel Mode Setting (KMS). Gre za podsistem, ki nam omogoča, da preko libdrm knjižnice neposredno konfiguriramo strojno opremo prek ioctl sistemskih klicev. KMS torej uporabimo le v primeru, ko kompozitor in upravljalec oken nista na voljo (npr. zaklenjen zaslon).

5.4 Okenski sistem X

Okenski sistem X (na kratko: X) je odprtokoden okenski sistem, najdemo pa ga na večini današnjih linux sistemov. Organiziran je v zvezo odjemalec-strežnik in tako omogoča tudi oddaljen dostop. X je eden izmed najbolj osnovnih gradnikov grafičnih uporabniških vmesnikov, zadolžen pa je tudi za rokovanje z dogodki (event handling) in za olepšave (visual decorations)

5.4.1 X11

X11 je protokol, ki definira izmenjavo sporočil v strežniški arhitekturi sistema X. Če se strežnik in klient nahajata na isti napravi, se sporočila izmenjajo z UNIX vtiči. X11 je razširljiv - nove funkcionalnosti lahko dodajamo brez spreminjanja protokola samega.

5.4.2 Xlib in XCB

Xlib (X library) je knjižnica, namenjena implementaciji na strani klienta. Uporabljajo jo druga orodja, kot sta GTK+ in Qt za izdelavo grafičnih vmesnikov za aplikacije.

X.org Foundation danes odsvetuje razvoj aplikacij z neposredno uporabo X knjižnic, temveč priporoča različna orodja (dve omenjeni zgoraj). Za nizkonivojski razvoj prav tako odsvetujejo uporabo Xlib knjižnice[17], priporočajo pa njeno "novejšo verzijo" oz. alternativo XCB (X C Binding). Razlogi so manjša velikost, večja odpornost na napake programerja, boljša podpora za multithreading, več razširitev in nižje latence[18]

5.4.3 X.org server

X.org je strežniška implementacija okenskega sistema X. Strežnik navadno zažene upravljalec prikaza ali pa ga ročno zaženemo prek ukaznega poziva.

5.5 Cairo

Cairo je knjižnica, ki skrbi izključno za izrisovanje vektorske grafike. Podpira tudi izrisovanje na X11 platna preko knjižnice Xlib. Cairo knjižnice večinoma ne uporabljamo neposredno, temveč prek orodij, kot je GTK+.

5.5.1 Pixman

Pixman je deljena knjižnica med X strežnikom in knjižnico Cairo. Nudi algoritme za rasterizacijo, podporo za gradiente in drugo.

5.6 Kompozitor

Kompozitor je program, ki nudi grafične olepšave okenskemu sistemu. To stori z uporabo medpomnilnika oken - vanj za vsakega X klienta shrani vse okvirje, ki jih klient ustvari. Sedaj lahko tem okvirjem doda različne olepšace, kot so sence, prosojnost in zameglitev, tranzicije med okni in celo vertikalna sinhronizacija (V-sync).

5.7 Wayland

Okenski sistem X je že precej star (prva verzija je zaživela leta 1984) in posledično postaja neroden za uporabo, obenem pa njegova zasnova ni varna, saj ne kriptira oz. ne ščiti komunikacij med klientom in strežnikom, kar omogoča prisluškovanje in potencialno razkritje zaupnih informacij.

Wayland je naslednik okenskega sistema X. Gre za komunikacijski protokol, ki na mesto strežnika postavlja kar kompozitor, ki pa igra tudi vlogo upravljalca oken. Tako komunikacija s klienti (uporabniškimi aplikacijami) ne poteka več preko centralnega strežnika, ločenega kompozitorja, od tu nazaj do strežnika in naposled do Linux jedra, temveč klienti komunicirajo direktno s kompozitorjem, ta pa nadalje direktno z jedrom. Posledično Wayland omogoča, da menjavo oziroma izbiro med vrsto različnih kompozitorjev, sprogramiranih za to arhitekturo.

Kot je že moč razbrati, Wayland ponuja preprosto nadaljno vzdrževanje, razvijanje in razširjevanje.

5.7.1 XWayland

Zaradi lažje kompatibilnosti v času, ko še vedno prevlada prikazni strežnik X, lahko uporabimo orodje XWayland, ki pod implementacijo Wayland arhitekture vrine še X strežnik in tako omogoči, da novejši protokol uporabijo tudi aplikacije, ki niso namenjene zanj.

5.8 GUI orodja

GUI orodje (tudi GUI knjižnica) vsebuje vse potrebne funkcionalnosti za izdelavo grafičnih uporabniških vmesnikov, kot so rokovalniki dogodkov, prizori (scene) in interaktivni elementi (widgets).

V večini primerov so GUI orodja le ovojne knjižnice nizkonivojskih knjižnic, kot sta Xlib in XCB. Nekateri imajo celo implementirane lastne označevalne jezike, sisteme za dogodke in končne avtomate.

5.8.1 GTK+

GIMP Toolkit je GUI orodje za izdelavo večplatformskih aplikacij z uporabniškim grafičnim vmesnikom.

5.8.2 Qt

Qt je GUI orodje za izdelavo večplatformskih aplikacij z uporabniškim vmesnikom, ki ponuja knjižnico interaktivnih elementov, za razliko od GTK+ orodja pa lahko za izdelavo vmesnikov uporabimo aplikacijo QtDesigner in integrirano razvojno okolje (IDE) QtCreator.

5.9 Vloga OpenGL-a

Čeprav je na tej točki že razvidno, da na Linuxu ne obstaja namensko orodje ali knjižnica za izdelavo uporabniških grafičnih vmesnikov, lahko opazimo, da (prej ali slej) vsi grafični ukazi v Linuxovem grafičnem skladu uporabijo OpenGL. To pomeni, da lahko kot Linuxovo namensko orodje za grafiko uporabimo kar OpenGL. Primer programa, ki deluje na opisan način, je Blender, ki se ne zanaša na nizkonivojske knjižnice ali orodja, temveč ima svojo knjižnico, popolnoma osnovano na standardu OpenGL.

6 Nekaj primerov

7 Viri