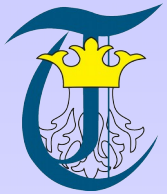


Universitatea *Transilvania* Brașov
Facultatea de Inginerie Electrică și Știința Calculatoarelor
Departamentul de Electronică și Calculatoare

# Sisteme de operare distribuite și în timp real

– curs –
master SECI anul I

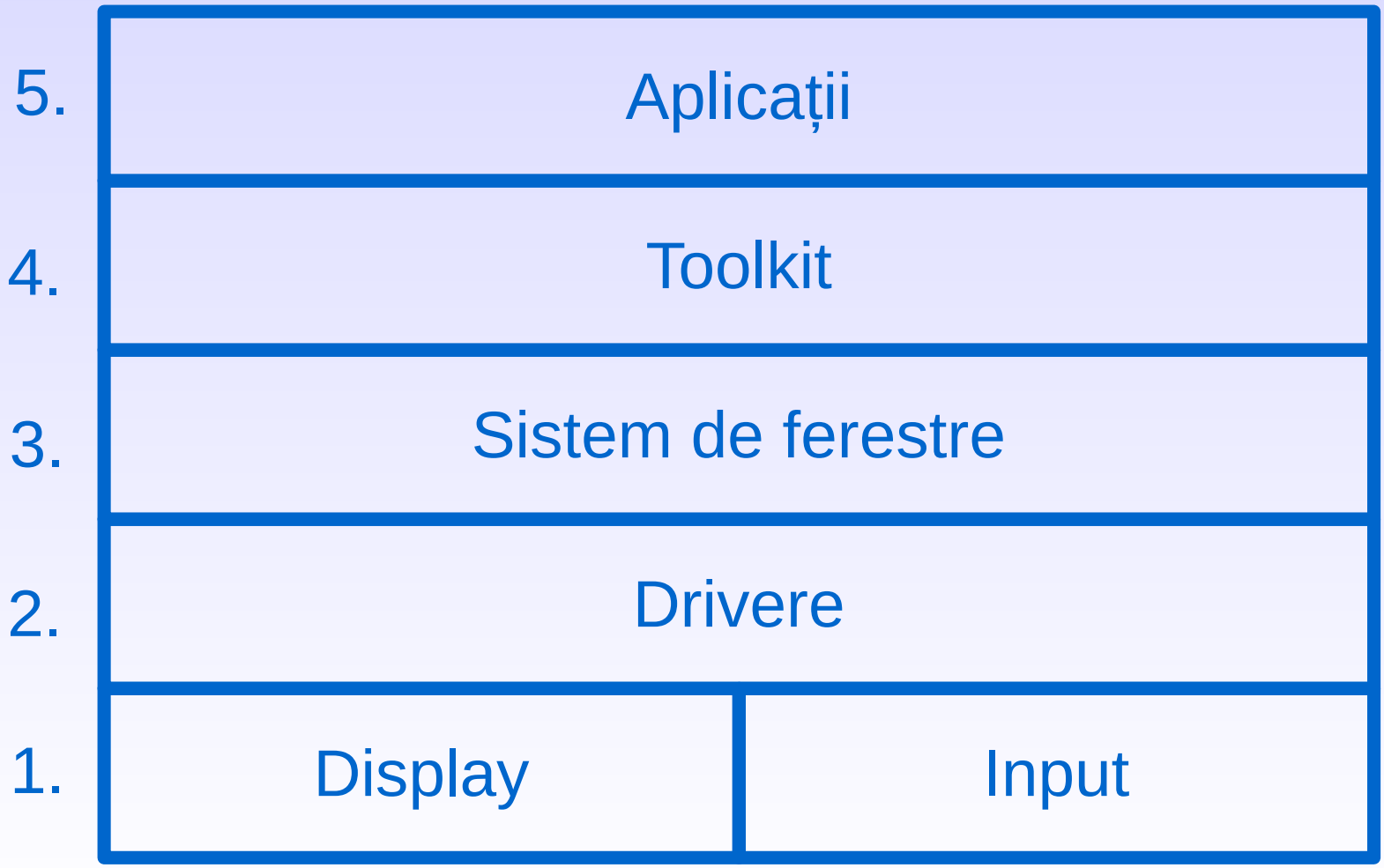
ș.l. dr. ing. Kertész Csaba-Zoltán

# Grafică încorporată

- sistemele embedded oferă din ce în ce mai mult și o interfață grafică către utilizator
  - folosire mai ușoară, mai intuitivă
  - tendințe de piață
- interfețele grafice pot fi
  - cu LED-uri sau LCD-uri dedicate (interfațare simplă din drivere)
  - avansate: LCD grafic (monocrom sau color), TFT, OLED, touchscreen
    - aceste interfețe au nevoie de sisteme grafice care abstractizează conținutul afișat de hardware

# Sisteme grafice

- interfața dintre aplicații și display



# Sisteme grafice

- hardware-ul legat de interfața grafică:
  - ecran LCD, OLED, etc, cu organizare pe pixeli
  - input: touchscreen, ...
- SO include drivere pentru aceste dispozitive
  - permite afișarea pe pixeli
- sistem de ferestre
  - sistem ce oferă suport pentru afișarea formelor geometrice (linii, suprafețe), a fonturilor, și combinarea acestora provenind de la diferite aplicații

# Sisteme grafice

- toolkit-urile grafice oferă un API către aplicații
  - permite interfațare ușoară între programe și sistemul de ferestre
  - poate oferi portabilitate peste diferite sisteme de ferestre
- aplicațiile grafice
  - de obicei folosesc API-ul oferit de toolkit pentru a desena pe ecran
  - mai multe aplicații pot să deseneze deodată pe ecran
  - în unele cazuri aplicațiile pot accesa direct driverul grafic pentru desenare accelerată

# Sistemul grafic X

- sistemul Linux pe desktop folosește sistemul grafic X
  - Xfree86, Xorg
  - încorporează driverele pentru ecran (VGA, SVGA) și pentru dispozitive de input (tastatură, mouse, ... )
  - are o arhitectură client-server
    - serverul X oferă interfața API către drivere
    - aplicațiile se conectează la acest server ca niște clienți, folosind un protocol IPC (socketi) prin intermediul căreia accesează API-ul oferit de server
    - X poate fi extins și pe rețea
    - window manager este un client privilegiat (tratează operațiile asupra ferestrelor asociate aplicațiilor)

# X pe sisteme embedded

- arhitectura X nu este concepută pentru sisteme încorporate
  - sistemele embedded nu beneficiază de arhitectura orientată pe rețea
  - X prezintă multe dependințe către alte servere (X font server) sau aplicații (X resource manager, X window manager) care măresc excesiv dimensiunea sistemului
  - X a fost scris pentru calculatoare de uz general performante, incluzând multe operații care necesită putere de procesare mare
- există variante reduse de X, de ex. nano-X

# Suport kernel pentru aplicații grafice

- interfețele către dispozitive grafice de obicei încorporează un frame buffer
  - frame buffer conține o reprezentare pe pixeli a imaginii ce trebuie afișate
  - este realizat ca un buffer de memorie de dimensiunea w×h byte ( sau ×3 sau ×4 byte), ușor accesibil prin operații cu pointeri
  - kernelul include suport pentru frame-buffere (începând cu 2.2) ce permit desenare directă într-un frame-buffer, care după aceea va fi transmis către driverul de display (timp în care se poate scrie într-un alt frame-buffer)

# Folosirea frame-bufferului

- în cazul aplicațiilor simple, acestea pot accesa direct frame-bufferul din kernel fără a fi nevoie de un sistem grafic (de gen X)
- aplicațiile pot deschide /dev/fb, și scrie datele în ele (reprezentând pixeli)
  - pentru acces mai rapid se poate folosi o mapare în memorie a fișierului cu mmap()
- în cazul aplicațiilor mai complexe, care au nevoie de acces multiplu către ecran (de ex. mai multe aplicații scriu pe ecran) folosirea directă a frame-bufferului nu este fezabilă

# Toolkit-uri pentru grafică embedded

- folosirea frame-bufferului din kernel este cea mai avantajoasă pentru sisteme embedded
- există toolkituri care implementează un API ușor de folosit și cu resurse minime ce permit crearea aplicațiilor grafice ce folosesc frame-buffer
  - nano-X: implementează un API similar cu X, dar care are nevoie de resurse mult mai mici
  - directfb: un API foarte subțire, oferind și grafică accelerată
  - Qt/Embedded: un API aproape identic cu cel din Qt permițând o portabilitate crescută

# Nano-X

- un proiect open-source special destinat pentru aplicații embedded
- are un footprint mic: 100k code, 50-250k ram
- oferă o interfață către driverele de display, mouse, tastatură și touchscreen
- suport pentru formate de pixel de 1, 2, 4, 8, 16, 24, 32 bit (ușor de folosit pe diferite displayuri)
- oferă două interfețe API: Microwindows (similar cu win32 sdk) și Nano-X (similar cu Xlib)
- foarte configurabil la nivel de compilare

# Nano-X

- nucleul nano-X oferă suport pentru desenarea liniilor, cercurilor și poligoanelor pe ecran, pentru afișarea imaginilor de tip BMP, JPG și PNG și un motor de fonturi capabil să afișeze fonturi True-Type și Type1
- nano-X poate fi rulat într-un mod similar cu X: există un server nano-X la care aplicațiile se conectează prin IPC, dar poate fi legat și împreună cu aplicații pentru a înlătura apelurile IPC

# Nano-X

- complexitatea programării crescută
  - ex.: redimensionarea unei ferestre

```
if (net_wm_moveresize)
  {
      //if _NET_WM_MOVERESIZE is supported
      //create an X Event
      XEvent xev;
      xev.xclient.type = ClientMessage;
      xev.xclient.message_type = net_wm_moveresize;
      xev.xclient.display = x11Info().display();
      xev.xclient.window = winId();
      xev.xclient.format = 32;
      xev.xclient.data.l[0] = event->globalPos().x(); //x_root
      xev.xclient.data.l[1] = event->globalPos().y(); //y_root
      xev.xclient.data.l[2] = 8;                      //direction
      xev.xclient.data.l[3] = Button1;                //button
      xev.xclient.data.l[4] = 0;                      //source indication
      //release the mouse to the window manager
      XUngrabPointer(x11Info().display(), CurrentTime);
      //send event
      XSendEvent(x11Info().display(), QX11Info::appRootWindow(x11Info().screen()), False,
                 SubstructureRedirectMask | SubstructureNotifyMask, &xev);
      return;
  }
```

# Qt/Embedded

- este toolkit-ul Qt destinat pentru Linux embedded (apelând direct kernelul fără alte sisteme de ferestre)
- sistemul se bazează pe nucleul Qt: este scris în C++ și are un footprint relativ mare până la 9MB (necomprimat)
- permite o portare foarte ușoară a aplicațiilor, practic toate aplicațiile Qt rulează nemodificat și pe Qt/Embedded

# Qt/Embedded

- codul sursă devine mai simplă și mai inteligibilă
  - ex.: redimensionarea unei ferestre

```
setGeometry(geometry().translated(event->globalPos() - m_lastPos));
```

- codul compilat însă devine mult mai mare și consumă mai multe resurse
- recomandat numai pentru sisteme puternice, care au nevoie de experiență utilizator sofisticat

# Embedded Wizard

- pentru sisteme cu resurse limitate
  - footprint redus
  - suport pentru nuclee simple de RTOS
  - poate fi creat o singură imagine cu toată aplicația
- oferă suport pentru mai multe platforme embedded (de performanțe acceptabile pentru ux)
  - există suport și pentru portarea pe platforme noi
- costuri unice de dezvoltare relativ mari
- versiune de evaluare pentru microcontrollere STM32Fx

# Embedded Wizard

- limbaj de programare dedicat – chora
  - orientat pe obiecte
  - are primitive pentru lucrul cu interfețe grafice: puncte, dreptunghiuri, culori
  - poate include cod nativ C
  - posibilitate de reconfigurare prin metadate integrate în cod
  - are un generator de cod asociat care creează cod C automat din fișierele sursă chora
  - codul generat se leagă împreună cu o bibliotecă dedicată platformei și poate fi compilat direct pentru a crea aplicația pe target

# Embedded Wizard

- design grafic și prototipare vizuală a aplicației
  - se creează interfața grafică din elemente de bază (puncte, linii, dreptunghiuri, imagini, etc...)
  - comportamentul interfeței poate fi vizualizat chiar în editor
- depanare ușoară a aplicației grafice pe calculatorul gazdă
  - sistem de prototipare care rulează partea grafică a aplicației independent de platformă
  - urmărire în timp real a tuturor variabilelor interne și a tuturor obiectelor folosite

# Limbajul chora

- bazat pe C (fără caracteristicile limbajului considerate periculoase – v. MISRA)
  - nu are pointeri
  - vectori statici
- orientat pe obiecte
  - moștenire
  - încapsulare
- garbage collector automat

# Limbajul chora

- primitive de date
  - int8, int16, int32, uint8, uint16, uint32
  - bool
  - float
  - point
  - rect
  - color
  - char, string

# Limbajul chora

- clase
  - variabile
  - proprietăți (metode setter și getter asociate)
  - metode
  - sloturi (pot fi notificate asincron)
- metadate
  - profiluri (compilare condiționată, diferite platforme)
  - stiluri (skinning automat)
  - generator optimizat (va genera cod exlusiv pentru părțile folosite)

# Toolkit-ul Mosaic

- include funcționalități de bază pentru crearea aplicațiilor grafice
  - widgeturi de bază (View, Group, liste, butoane, imagini, icoane, text, etc...)
  - evenimente (timer-e, apăsare de taste, touch)
  - resurse (bitmap-uri rasterizate, font-uri)
  - grafică 3D