Rendszer figyelése Conky-val, 2. rész

PClinuxOS Magazine – 2015. július

Írta: Peter Kelly (critter)

A múlt hónapban megismerkedtünk a conky alapjaival, de most valami ütőset akarunk, ezért Lua-val és Cairo-val egészítjük ki.

Lua és Cairo

Lua és Cairo úgy hangozhat, mint egy kedves pár neve, akiket a nyaralás során ismertél meg, és e-mail címet cseréltetek, de ez nem az a pár.

A Lua egy nyílt forráskódú szkriptnyelv, amit teljes egészben a brazil Pontifical Catholic University of Rio de Janeiro dolgozott és fejlesztett ki. Lua az írása nagy „L”-lel. A Cairo egy fejlett, 2D grafikus könyvtár. Conky képes együttműködni a Lua-val néhány trükkel és használ néhány Lua-ra jellemző függvényt.

Mindez annyit tesz, hogy írni kell néhány Lua szkriptet a Conky-nak, ami talán felhasznál egy-két függvényt a Cairo grafikus könyvtárból. Ne aggódj, ez egyáltalán nem olyan bonyolult. Ha valóban nem akarsz továbblépni, számos on-line elérhető konfigurációs fájl van, ami Lua szkripteket használ és készen vár, hogy saját céljaidra felhasználd. De olvass tovább, talán tanulsz valamit, ami tetszeni fog.

Miért Lua és Cairo?

Ideje elmagyarázni, hogy mit akarnak csinálni itt. A Conky magában képes néhány szép kimenetet előállítani, de a beépített függvényei korlátozzák. Felvértezve más szkript-nyelvek alkalmazásának képességével elérhetővé válik az adott nyelv sok képessége. A Lua egy fejlett és gazdag eszköztárú szkriptelő nyelv. Emellett megszünteti az alap Conky számos korlátozását. Hogy milyen mértékben teszi, függ attól, hogy mennyire messzire mész el. Emellett, a Lua képes importálni külső könyvtárak függvényeit és a Cairo grafikus könyvtárak olyanok, amik néhány nagyon hasznos rutint használnak egységes rendszerben. Mind a Lua, mind a Cairo túl nagy, hogy áttekintsük, itt én csak érintem a felszínt. Bemutatom az alapvető alkalmazást néhány példával és rátok hagyom a további felfedezést.

A Conky-nak nem kell a Lua és a Lua-nak sem a Cairo, de Conky + Lua + Cairo = ÜTŐS.

Lua használata Conky-val

A Lua szkriptelő nyelv, és utasításokat olyan sima szövegfájlból olvas ki, amiket – nem meglepően – szkriptnek (leírás) hívunk. Ezért a Conky-nak hozzá kell férnie a Lua szkripthez, amit az alábbi két sornak a Conky beállítófájljában a TEXT szakasz elé történő beírásával érünk el. Az első sor:

lua_load az_útvonal_a_te_lua_szkriptedhez (A Lua szkript .lua végződésű.)

Közli a Conky-val, hogy milyen Lua szkriptet töltsön be. Természetesen meg kell adnod a Lua szkript útvonalát. A bemutató céljára használt Lua szkript a következő

/home/felhasználó/demo.lua

A második sor:

lua_draw_hook_pre conky_demo_mag

Közli a Conky-val, hogy a szkript melyik függvényét kell végrehajtani. A szkriptben a függvény nevének conky_ kezdetűnek kell lennie, mint fent, itt eltekinthetünk most ettől.

A ***lua_draw_hook_pre demo_mag*** szintén működik.

Változtasd még ezt is meg:

„***update_interval 5”-öt „update_interval 1***”-re, vagy egy megfelelő értékre, ami használható eredményt ad.

Ahhoz, hogy megfelelő méretű képernyőnk legyen, add hozzá ezt a sort:

minimum_size 400 400

Mint mindig, a Lua-Conky integráció változatos és sokféle lehet, de ez az eljárás elég egyszerű és nálam működik.

Ha csak ezt a két első sort adod hozzá a sablonodhoz, akkor hagyd a következő text szakaszt üresen és mentsd újra talán .conkyrc_lua néven, működni fog. Természetesen több mindent adhatsz a TEXT szakaszhoz és alkalmazhatod a Conky és a Lua kimenetét együtt, de most legyen csak egyszerű.

Most kell egy demo.lua nevű Lua szkript. A Cairo könyvtárak a grafikájukat úgy állítják elő, hogy egy forrásgrafikát maszkok és útvonalak alkalmazásával bemásolnak a Cairo felületére, ez adja a végeredményt. Ez nem egy túl pontos leírása a Cairo rajzolási modellnek, de elég egyszerű ahhoz, hogy a további vizsgálatokhoz felhasználjuk.

Az első demó szöveges grafikát használ forrásként. A Lua szkript a fölösleges szóközöket eldobja, ezért az olvashatósághoz szükséges szóközöket használd. Ez az alap Lua szkript, sorszámozással.

```
1 require 'cairo'
2 function conky_demo_mag()
3 if conky_window == nil then return end
4 local cs = cairo_xlib_surface_create(conky_window.display, conky_window.drawable, conky_window.visual, conky_window.width, conky_window.height)
5 cr = cairo_create(cs)
6 -- Start of output
7 cairo_select_font_face (cr, "Liberation Sans", CAIRO_FONT_SLANT_NORMAL, CAIRO_FONT_WEIGHT_NORMAL);
8 cairo_set_font_size (cr, 24)
9 cairo_set_source_rgba (cr,1,0,0,1)
10 cairo_move_to (cr,80,200)
11 cairo_show_text (cr,"PCLinuxOS Magazine")
12 cairo_stroke (cr)
13 -- End of output
14 cairo_destroy(cr)
15 cairo_surface_destroy(cs)
16 cr=nil
17 end
```

A kimenete egy sima szöveges fájl. A legtöbb célra a csak a 6. és 13. sor közötti kódot kell megváltoztatni.

Az 1. sor mondja a Lua-nak, hogy szkript futásához Cairo könyvtárak kellenek.

A 2. sortól kezdődik a függvény leírása, amit a Conky hív majd meg.

A 3. sor ellenőrzi a Conky ablak meglétét. Ha nincs, a függvény kilép és mivel a függvényt semmi sem követi, a szkript is kilép.

A 4. sor sokkal érdekesebb. A local azt jelenti, hogy ami követi az csak ehhez a helyi függvényhez tartozik és a szkript többi része nem módosítja, még ha névütközés lenne is. Ezt egy Cairo objektumfelület paraméteres definíciója következik, ami a Conky ablakát használja megjelenítésre; engedélyezi, hogy a Conky ablakba rajzoljunk; láthatóvá teszi a grafikát; és a felületet a Conky ablaka szélességével és magasságával egyenlőre méretezi át.

Ez lehetővé teszi a Conky relatív koordinátáinak használatát. A felület típusát a cs objektumhoz irányítja, ez ahogy hivatkozunk rá.

Az 5. sor egy cs típusú felületet hoz létre és a cr változóhoz köti.

A 6. egy megjegyzés sor és nem dolgozza fel. Minden a „--” után megjegyzésnek számít.

A 7. sor állítja be a cr felületen használandó betűtípust. A típus „Liberation Sans”. A betű dőlését a CAIRO_FONT_SLANT_NORMAL (normál) Cairo konstans határozza meg. Egy további lehetőség a CAIRO_FONT_SLANT_ITALIC (dőlt). Betű vastagság a CAIRO_FONT_WEIGHT_NORMAL (normál) konstans által megadott. Másik opció a CAIRO_FONT_WEIGHT_BOLD (félkövér).

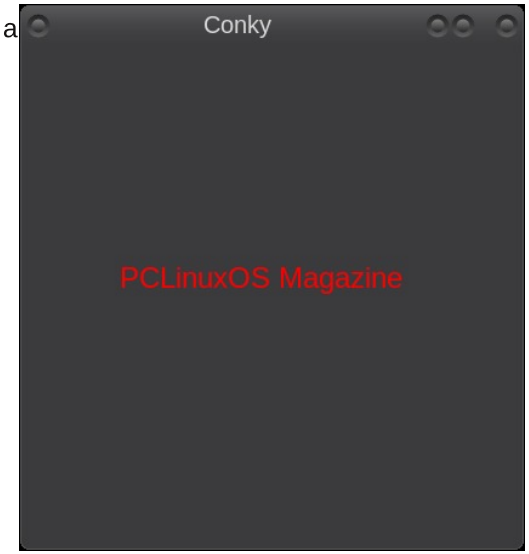
A 8. sor állítja be a betűméretet.

A 9. sor a színt határozza meg. A formátum vörös, zöld, kék alfa és az értékek 0-tól 1-ig lehetnek a 0-tól 255-ig helyett, amihez inkább hozzászoktunk. A Lua képes röptében konvertálni, amennyiben olyan kifejezést használunk mint cairo_set_source_rgba (cr,234/255,0,107/255,1).

A 10. sor mozgatja a beviteli pontot.

A 11. sor adja meg a megjelenítendő szöveget.

Végül a 12. sor rajzolja meg a szöveget.



A 13. sor ismét csak megjegyzéseket tartalmaz.

A 14-16. sorok a kilépés előtti tisztogatást végzik el.

A ***conky -d .conkyrc-lua*** futtatása ezt eredményezi (jobbra). Működik, de hiányzik belőle a plusz. Semmit sem csinálunk valami extra nélkül.

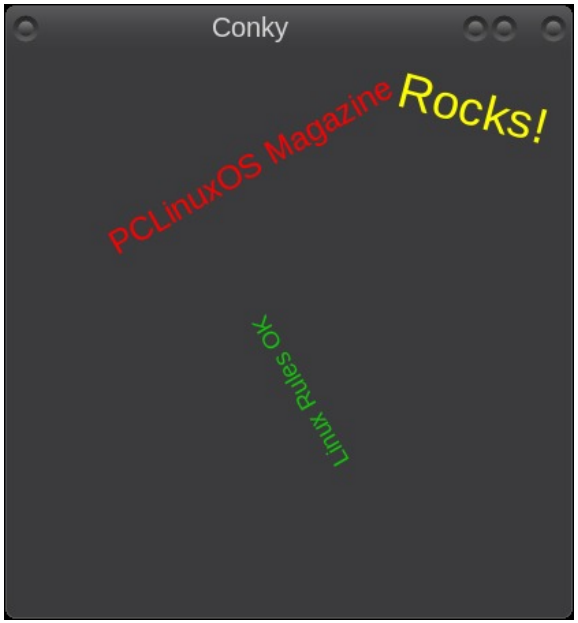
Vedd észre, Lua és Cairo használata esetén lehet egy kis késleke-dés, mielőtt bármi megjelenne a

Conky-ban, különösen amikor conky változókat dolgoz fel (lásd később). A terminálban figyelmeztetések jelenhetnek meg, ez normális.

Írd át a szkript két megjegyzés sora közötti részét az alábbiak szerintire:

```
cairo_select_font_face (cr, "Liberation Sans", CAIRO_FONT_SLANT_NORMAL, CAIRO_FONT_WEIGHT_NORMAL)
cairo_set_font_size (cr, 24)
cairo_set_source_rgba (cr,1,0,0,1)
cairo_move_to (cr,80,150)
cairo_rotate (cr, -math.pi / 6)
cairo_show_text (cr,"PCLinuxOS Magazine")
cairo_set_font_size (cr, 36)
cairo_set_source_rgba (cr,1,1,0,1)
cairo_move_to (cr,210,210)
cairo_rotate (cr, math.pi / 4)
cairo_show_text (cr,"Rocks!")
cairo_set_font_size (cr, 18)
cairo_set_source_rgba (cr,19/255,197/255,11/255,1)
cairo_move_to (cr,333,210)
cairo_rotate (cr, -3 * math.pi / 4)
cairo_show_text (cr,"Linux Rules OK")
cairo_stroke (cr)
```

Jegyezd meg, hogy a cairo_stroke (cr) hívás csak egyszer kell a három szövegblokkhoz, de nincs mindig így. Kétség esetén írd be a sort. Futtasd a Conky-t ismét és a fentihez hasonló képernyőt kell kapnod. Haladás!



De ismét csak készíthettem volna forgó szövegképeket bármilyen grafikus alkalmazással és megjeleníthettem volna a „normál” Conky-val képként.

A nagy ugrás akkor jön, amikor „élő” szöveget jelenítesz meg ehhez hasonló formában, statikus szöveg helyett. Megjeleníthetsz szöveget, ami ered a rendszer lekérdezéséből, szöveget, ami például jelenti az akkumulátor aktuális töltöttségi állapotát. Nagyon könnyű megcsinálni.

Vagyis, ezt fogjuk a legközelebb tenni.

Előbb jobb, ha elmagyarázok néhányat azok közül az új, vagy megváltoztatott sorok közül. Sok sort adtam hozzá, de többsége ismétlése a korábbiaknak.

Az első új sor a

cairo_rotate (cr, -math.pi / 6).

Forgatja a kimenetet a „math.pi/6” számítás eredményével. A Lua radiánnal méri a szöget, ami 3 órától indul, vagy egy kör középpontjához képest keleti irányban indul és az óra járásának megfelelő irányba fordul. A Lua-nak nagy matematikai függvénykészlete van és a -math.pi a pi (π) negatív értékét adja vissza. Mivel egy kör 2π radián, ez 30° az óra járásával ellentétesen. Legtöbb embernek a fokok használata megfelel, rávesszük a Lua-t ennek számolására.

cairo_rotate (cr,-30 * (math.pi / 180))

Tipp: hogy biztosan a várt eredményt kapjuk, használhatod a print parancsot kimenet megjelenítésére a terminálban.

print (-30 * (math.pi / 180))

Ez egy általános célú hibakereső módszer, de math használata esetén nem értékelhető.

A szöveg második sora 15°-os forgatást jelent, ami forgatások összeadódnak. A szöveg elhelyezéséhez a 210,210-es koordinátákat kellett használni (hibák és próbálkozások során találtam rá), de ez egy 400 pixeles négyzet alakú ablakban nem lehet 210,210, ez nagyjából a közepe lehet. Hogy használható koordináta rendszerünk legyen, vissza kell forgatni 0°-ra, végrehajtani az utasítások fordítását, majd szükség szerint forgatni. Ez következik utána

```
cairo_rotate (cr, 30 * (math.pi / 180))    -- Forgatás vissza, ahol voltunk.
cairo_move_to (cr, 283,40)
cairo_rotate (cr, 15 * (math.pi / 180))
```

Ezek már sokkal elfogadhatóbb koordináták.

Kicsit rendbe szedhetjük a szkriptet úgy, hogy a Cairo konstansainak a sablonunkban a kimenet kezdete előtt értéket adunk, azaz a 6. sor előtt.

f_noslant=CAIRO_FONT_SLANT_NORMAL
f_italic=CAIRO_FONT_SLANT_ITALIC
f_nobold=CAIRO_FONT_WEIGHT_NORMAL
f_bold=CAIRO_FONT_WEIGHT_BOLD

Majd valahogy így hívd meg azokat:

cairo_select_font_face (cr, "Liberation Sans", f_noslant, f_nobold);

A Lua, hogy conky változótól vegyen át értéket, a conky_parse() függvényt alkalmazza. Ezeket a sorokat illeszd be a Lua szkriptbe közvetlenül a cairo_stroke (cr) sor elé.

cairo_select_font_face (cr, "Junkyard", f_noslant, f_bold)
cairo_set_font_size (cr, 38)
cairo_set_source_rgba (cr,19/255,197/255,208/255,1)
cairo_rotate (cr, 120 * (math.pi / 180))
cairo_move_to (cr,130,400)
cairo_rotate (cr, -60 * (math.pi / 180))
cairo_show_text (cr,"Akkutöltés "..conky_parse("${battery_percent}".."%"))

A kimeneti sor különböző részeinek összefűzésére két pontot .. használunk.

Most, ha a lentihez hasonló képernyőképet látsz, tudod hogyan készült.

Rajzolás

Kicsit játszottunk szövegekkel Lua-ban, de a Cairo egy erőteljes grafikus függvénykönyvtár. Cairo-val vonalakat, íveket és négyszögeket rajzolhatunk. A Cairo képes olyan bonyolultabbakra, mint pl. a függvényeket megrajzolni.

A Cairo rajzolatai köbös Bezier függvények, néha útvonalaknak, vagy spline-oknak nevezik. Ezek rendelkeznek meghatározott kezdő- és végponttal, illetve különféle köztes pontokkal, amik között halad, nem feltétlenül érintve azokat. A köztes pontok meghatározzák a görbe alakját ezért hívják ezeket parametrikus görbéknek. A Bezier-görbéket a korlátlan méretezhetőség jellemzi és mint grafikai elemeket, kezdetben autókarosszériák készítéséhez fejlesztették ki. Ezek Conky-ban nem feltétlenül túl hasznosak és nagyon komplex téma lévén nem kívánom tárgyalni a görbéket ebben a cikkben.

Vonalak

Vonal megrajzolásához a cairo_move_to való a kiinduló pontként és a cairo_line_to függvény egy vonalszakasz megrajzolásához. A cairo_line_to függvény ismételhető több vonalszakasz rajzolásához és a cairo_close_path való az alakzat lezárására. A vonalvastagságot a cairo_set_line_width segítségével lehet vezérelni. A vastagság a kiinduló-, vagy csúcspont körül egyenletesen oszlik el. A vastagság többes vonalszakaszokban nem változtatható. A vonalvastagság a cairo_stroke függvények között állandó marad. Nyílt végű vonalakhoz zárásra ezek egyike használható

CAIRO_LINE_CAP_BUTT (sarkos), ***CAIRO_LINE_CAP_ROUND*** (kerekített), vagy ***CAIRO_LINE_CAP_SQUARE*** (szögletes)

és lezáratlan vonalak esetén a tetőpontoknál a csomópontok ezeket a formákat vehetik fel

CAIRO_LINE_JOIN_MITER (ferde), ***CAIRO_LINE_JOIN_BEVEL*** (szögletes), vagy ***CAIRO_LINE_JOIN_ROUND*** (kerek)

Ezeket a változókat a sablonfájlodban meghatározhatod. Ha egy zárt sokszöget rajzolsz, kitöltheted a cairo_stroke-ot cairo_fill-re cserélve. Ekkor a vonalvastagságot 0-ra állítja. A következő egy hatszöget rajzol meg.

```
cairo_set_line_width (cr,4)
cairo_move_to (cr,300,200)
cairo_line_to (cr, 250,113)
cairo_line_to (cr, 150,113)
cairo_line_to (cr, 100,200)
cairo_line_to (cr, 150,287)
cairo_line_to (cr, 250,287)
cairo_close_path (cr)
cairo_stroke (cr)
```

Ennyit rövid áttekintésként a vonalak használatáról Lua-ban Cairo könyvtárakat használva.

Hogy valami hasznosabbat csináljunk, mivel a Cairo íveket képes rajzolni, készítek sebességmérőt a processzormagokhoz. Előbb nézzük át az ívek használatát.

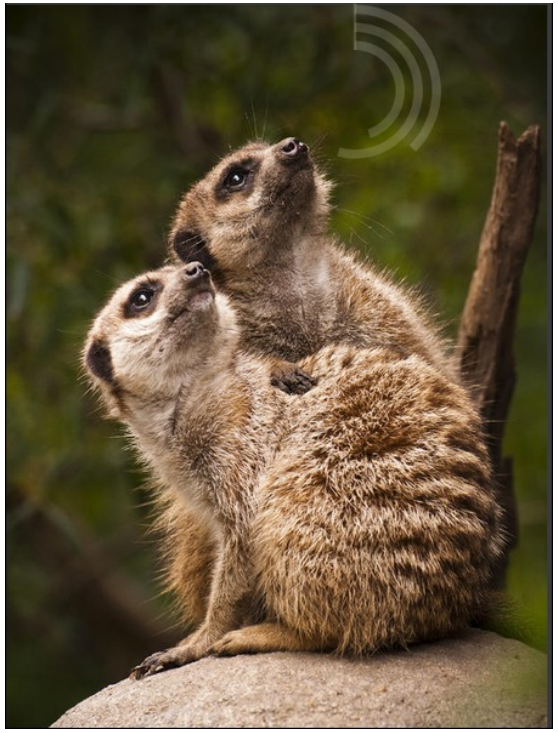
Ívek

Ív lehet egy kör része, vagy ha kezdő és végpont egybeesik, akkor egy teljes kör. Összetett ábrákhoz az ívek keverhetők vonalakkal. Egy ív megrajzolásához meg kell határozni a középpont helyét, a sugarát, illetve az ív kiinduló és a vége szögét. Be kell még állítani a vastagságot, a színt és az ív végének típusát, noha erre vannak alapértékek.

Két cairo parancs van az ívekre:

cairo_arc	rajzolás óramutató járásának megfelelő irányba
cairo_arc_negative	rajzolás óramutató járásával ellentétes irányba

Mindkettő rendelkezik a következő paraméterekkel: felületi hivatkozás (a felületre cr-rel hivatkozunk), középpont x,y koordinátái, induló szög radiánban és zárószög radiánban.



Ennyi. Az ívek egyszerűek, mivel szögeket elfogadnak. Ha tudod, hogy mit figyelsz meg, akkor nehézségek nélkül elkészítheted a Conky képernyődet. Ebből a célból én átméreteztem a Conky ablakát a rajz befogadására és utána ezekkel a sorokkal egészítettem ki a Lua sablonomat:

```
cairo_set_source_rgba (cr,220/255,218/255,213/255,0.2)
cairo_set_line_width (cr, 10)
cairo_arc (cr,390,100,90,-90 * (math.pi/180),45 * (math.pi/180))
cairo_stroke (cr)
cairo_arc (cr,390,100,70,-90 * (math.pi/180),105 * (math.pi/180))
cairo_stroke (cr)
cairo_arc (cr,390,100,50,-90 * (math.pi/180),72 * (math.pi/180))
cairo_stroke (cr)
cairo_set_line_width (cr, 3)
cairo_move_to (cr, 390,55)
cairo_line_to (cr, 390,0 )
cairo_stroke (cr)
```

Adtam némi átlátszóságot az ívhez. A bemutatáshoz a szögértékek véletlenszerűek, de könnyen beolvashatóak Conky változóból.

Egy új típusú műszer

Tipp: használj olyan szerkesztőt, ami kiemeli az összetartozó zárójeleket (mint Kate, vagy Geany).

Előbb rakjuk fel a mutatót. A kódot csak egy magra írom le. A mutatónak van egy fix középpontja, ami körül fordul el a mért értéknek megfelelően. A legalacsonyabb pont a bal alsó negyedben a legmagasabb pedig a jobb alsó negyedben van. A vízszintes alatti 45°-os értéket választottam mindkét oldalon, ami 0% esetén -255°-nak, 100% esetén 45°-nak felel meg, 270° a teljes elfordulás, vagy 2,7° a CPU aktivitás minden 1%-ához. A vonalak nem kezelik a fokokat, ezért némi számolás kell a végpontok kiszámításához.

A változóknál szükséges értékeket tárolni fogom, ez a globális változtatásokat sokkal egyszerűbbé teszi:

A CPU-terhelést ezzel lehet megállapítani: ***cpu1=((conky_parse("cpu cpu1")))***

A szög a következő ***angle1=((cpu1 * 2.7) -225)***

A középpont beállítása ***ctrx1,ctrx2=100,200***. Ez meghatározza mind az x. mind az y értékét.

Az adott mutató hossza id ***p_len p_len=60***.

A mutató végpontjának x koordinátája a mutatóhossz szorozva a szög koszinuszával, az y értéke pedig a szinuszával. A Lua elvégzi nekünk a számítást.

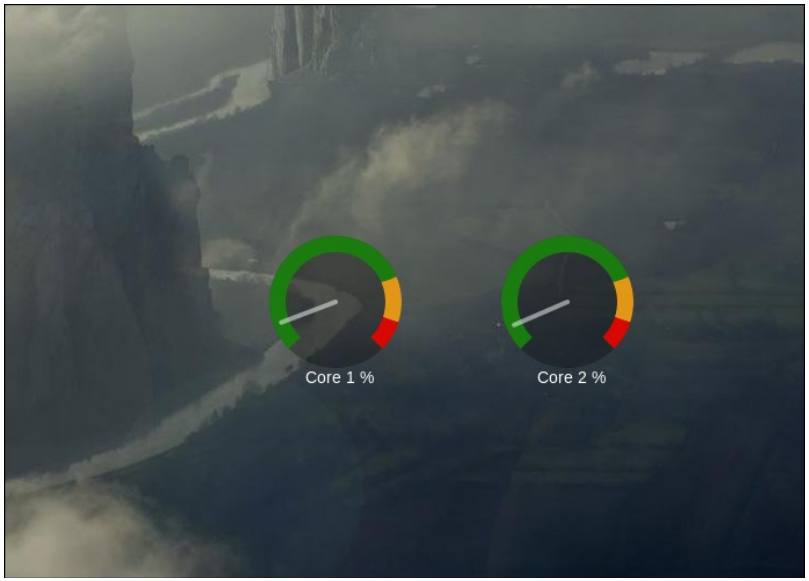
xval1=ctrx1 + (p_len * (math.cos ((angle1 * (math.pi / 180)))))
yval1=ctry1 + (p_len * (math.sin ((angle1 * (math.pi / 180)))))

Ha beállítottuk a vonal színét, vastagságát és végpontja típusát, kész vagyunk.

```
cairo_set_source_rgba (cr,1,1,1,1)
cairo_set_line_width (cr,4)
cairo_set_line_cap (cr,CAIRO_LINE_CAP_ROUND)
ctrx1,ctry1=100,200
cpu1=(conky_parse("${cpu cpu1}"))
cairo_move_to (cr, ctrx1,ctry1)
angle1=((cpu1 * 2.7) -225)
xval1=ctrx1 + (80 * (math.cos ((angle1 * (math.pi / 180)))))
yval1=ctry1 + (80 * (math.sin ((angle1 * (math.pi / 180)))))
cairo_line_to (cr, xval1, yval1)
cairo_stroke (cr)
```

Most akkor a műszer külső íve jön. Ennek a vonalat leíró Lua szkript sora előtt kell lennie, mivel azt szeretnénk, hogy a vonal az ív „fölött" jelenjen meg. A használni kívánt színek: zöld, sárga és vörös, mutatva a normális, az emelt és a veszélyes szintet. Kell még fekete háttér. Az ív közepe egyezik a mutatóéval, ezért az a változó újrahasznosítható. Az induló szög -225, az emelt indítását 75%-os CPU terhelésben határoztam meg, a veszélyes szintet pedig 90%-ban. A veszélyes szint vége természetesen a 100%. A vonalvastagság valahol a 14 pixelt meg kell haladja. Mivel az ív középpontja egyezik a mutató középpontjával és a mutatóhossz pedig a sugárral, ezeket az értékeket újra hasznosíthatjuk.

```
cairo_set_line_width (cr,14)
cairo_set_source_rgba (cr,0,0,0,0.3)
cairo_arc (cr,ctrx1,ctry1,p_len + 7,0,(2*math.pi))
cairo_fill (cr)
cairo_set_source_rgba (cr,27/255,124/255,16/255,1)
cairo_arc (cr,ctrx1,ctry1,p_len,-225 * (math.pi/180),((75 * 2.7)-225) * (math.pi/180))
cairo_stroke (cr)
cairo_set_source_rgba (cr,226/255,152/255,22/255,1)
cairo_arc (cr,ctrx1,ctry1,p_len,((75 * 2.7)-225) * (math.pi/180),((90 * 2.7) * (math.pi/180))
cairo_stroke (cr)
cairo_set_source_rgba (cr,217/255,8/255,0,1)
cairo_arc (cr,ctrx1,ctry1,p_len,((90 * 2.7) * (math.pi/180),45 * (math.pi/180))
cairo_stroke (cr)
```



Téglalapok

Noha vonalakkal négyszög rajzolható, olyan gyakoriak, hogy a Cairo külön függvénnyel – cairo_rectangle – rendelkezik azok elkészítésére. A szükséges argumentumok a bal alsó sarok x,y koordinátái, a szélessége és a magassága. Ebből elképzelheted, hogy vonalas műszer készítése viszonylag egyszerű feladat.

```
cairo_select_font_face (cr, "Liberation Sans", f_noslant, f_nobold)
cairo_set_font_size (cr, 18)
core1=(conky_parse("${cpu cpu1}"))
core2=(conky_parse("${cpu cpu2}"))
cairo_translate (cr, 100,200)
cairo_set_source_rgba (cr,1,1,1,1)
cairo_rectangle (cr,0,0,30,-100)
cairo_stroke (cr)
cairo_set_source_rgba (cr,0,1,0,1)
cairo_rectangle (cr,0,0,30,-core1)
cairo_fill (cr)
cairo_set_source_rgba (cr,1,1,1,1)
cairo_move_to (cr,-5,25)
cairo_show_text (cr,"core1")
cairo_move_to (cr,22,-25)
cairo_rotate (cr, -90 * (math.pi / 180))
cairo_show_text (cr,core1.." %")
```

Itt használtam egy új függvényt a beillesztési pont mozgatására, a cairo_translate-et.

Ez kétségtelen növeli a Lua-Cairo együttműködésben rejlő lehetőségeket. A határ a saját képzelőerőnk, amit remélhetően sikerül beindítani ezekkel a rövid bemutatókkal.