Comenzando en clip

El presente documento, fue desarrollado en base al trabajo realizado por Sergio A. Carrasco L., quien esta realizando un trabajo excelente y lo publica en breves mail, este documento es una compilación de estos.
Gustavo Carlos Asborno

Indice

EXTENSIONES DEL LENGUAJE Y FUNCIONES

CLIP es un compilador de Clipper, en otras palabras, incluye la sintaxis estándar de Clipper. Debido a esto, consulte los manuales de Clipper para comenzar a programar. Sólo las extensiones del lenguaje serán consideradas aquí.
En la práctica debemos agregar que la sintaxis, está más cercana al Clipper 5.3 (para que lo usen como referencia). Además, aquí sólo se mencionarán estas características extendidas pues no vale la pena su traducción. Si se desea hacer uso de ellas y saber explícitamente como usarlas, debe consultarse el manual oficial de CLIP en inglés, que trata de este tema en el capítulo 10. Si aún no lo obtienes, puedes bajarlo de:

ftp://ftp.itk.ru/pub/clip/clip-doc-en-html.tgz en formato HTML

ftp://ftp.itk.ru/pub/clip/clip-doc-en-pdf.tgz en formato PDF

Instrucción SWITCH :

```
[DO] SWITCH <expression>
        CASE <const11>[,<const12>,...]
                <statements>,...
        [ CASE <const21>[,<const22>,...] ]
                <statements>,...
        [ OTHERWISE ]
                <statements>,...
END[SWITCH]
```

Nuevo operador ":=@" :

```
a := "a" ; b := @a ; ? a,b          // "a","a"
```

Soporte estilo FoxPro para acceso de arreglos :

```
aVar := {1,2,3,4}
? aVar(nIndex)
```

Soporte de llamadas a bloques de código vía función:

```
cb := {|x1,x2|qout(x2,x2),"return value"}
? cb(1,2)
```

Constantes hexadecimales:

```
0xff        //255 en decimal
```

Aritmética racional:

```
//Compilador convierte a doble presición, perdiendo presición
x := 12345678901234567890123456789O
```

```
//Forma correcta de usar constantes de números grandes.
set(_SET_RATIONAL,.t.) ó SET RATIONAL ON
x:=val("12345678901234567890123456789O")
x:=val("1234.567890")
x:=val("1/3")
```

La función STR ahora soporta los parámetros "len" y "dec" :

```
? str(val("7/3"),1000,990)
```

Añadida función RSTR que da el resultado en forma de numerador/denominador :

? rstr(val("7/3")+val("5/6"))   // 19/6

Arreglos asociativos:

declare m[5]
m[1]=11; m[2]=12; ....

Instrucción FOR ... IN :

```
FOR <element> IN <associative_array>
   <statements>...
NEXT
```

ó

```
FOR <key> IN <associative_array> KEYS
   <statements>...
NEXT
```

Cadenas como arreglos:

string[<position>, |<length>|]

Modelo orientado a objetos :
Debido al modelo OO y la compilación a un programa C, existe la posibilidad de escribir las clases estándar Tbrowse y Get en Clipper mismo. Al mismo tiempo, la eficiencia de estas clases NO es peor que aquellas escritas en puro C.

Algunas de las CLASES predefinidas de CLIP, son las siguientes:

CODBDEPOSITORY	CODBDICTIONARY	CODBIDLIST
CODBLIST	DATETIME	FIND
HISTORY	LISTITEM	MEDIT
QUEUE	SORTEDARRAY	STACK
TEXTEDIT		

Las siguientes CLASES son para uso con la librería gráfica "libclip-ui.so":

UIBUTTON	UIBUTTONBAR	UICHECKBOX
UICHILDWINDOW	UICHOICE	UICOLOR
UICOMBOBOX	UIDOCUMENT	UIDRIVER
UIEDIT	UIEDITTEXT	UIFONT
UIFORM	UIFRAME	UIGRID
UIHBOX	UIIMAGE	UILABEL
UIMAINWINDOW	UIMENU	UIMENUCHECKEDITEM
UIMENUITEM	UIPOPUPMENU	UISPLITTER
UISTATUSBAR	UITABLE	UITIMER
UITOOLBAR	UITOOLBUTTON	UITREE
UIVBOX	UIWINDOW	UIWORKSPACE

Otras CLASES:

XFL_FORMS	XMLTAG

FUNCIONES CLIP Ordenadas alfabéticamente (no traducidas)
(* sólo se incluirá su nombre y descripción breve en el instructivo en PDF *)

GENERANDO NUESTRO CHARSETS

Linux y por consecuencia CLIP, vienen preparados para su uso internacional. Esto significa que podemos adaptarlo, en teoría, a cada uno de nuestros requerimientos nacionales. Para esto se vale principalmente, de dos herramientas: "keymaps" y "charsets".

Charsets: Traducido generalmente como "juego o tabla de caracteres", es un archivo donde aparecen los 256 códigos ASCII en una columna y su correspondiente codificación a caracteres en otra. Generalmente en los primeros 128 no hay problemas, en los segundos 128 es donde se marca la diferencia. Este archivo es la base para interpretar, mostrar o imprimir nuestros caracteres. En general para el ámbito latinoamericano se usa la tabla "cp850" (codepage 850).

La nacionalización de una aplicación CLIP, incluye dispositivos de pantalla, teclado, constantes de cadena y funciones embebidas.

Los dispositivos de pantalla son la consola, el terminal y el emulador de terminal. Uno puede configurar el entorno por medios estándar que el S.O. proporciona, por ej. cargado de fonts, terminal dependiente de BIOS, o por controladores de teclado/pantalla.

Algunas funciones Clipper (tales como upper, lower, isalpha, isdigit), pseudográficos, coeficientes de peso para indexación y comparación, etc., tienen características nacionales. Se dispone de la utilidad "gen_tbl" para la creación de un archivo con estas especificaciones.

CLIP posee especiales seteos para poder usar datos de DBF para distintos usuarios con diferentes "charsets" simultáneamente:

set("DBF_CHARSET","cp866")

O en forma de comando:

set dbf charset to cp866

Donde cp866 es el nombre del archivo "tbl" (tabla) con las descripciones de las características nacionales.

Puede darse que los datos son almacenados en una DBF con un "charset", un programa los opera con otro y un tercero, los muestra a su vez con un set distinto.

También existe la posibilidad de setear un "charset" para la impresora:

set("PRINTER_CHARSET","cp866")

O en su forma de comando:

set printer charset to cp866

¿Cómo crearlo?

Los "charsets" (set de caracteres) son usados para recodificar las entradas y salidas de un terminal, para transacciones con terminales pseudográficos, para determinar ordenamientos y, para recodificación de archivos y bases de datos en las finalizaciones de entradas y salidas.

Al igual que los "keymaps", CLIP usa su propio formato para almacenar "charsets", pero cuenta con una utilidad para la generación estándar de mapeos "charset-a-unicode".

Para añadir nuevos "charset":

1) Copia el apropiado mapeo de "charset" desde /usr/share/consoletrans/ (de cualquier distribución):

$ cp cp850.sfm.gz $CLIPROOT/chartsets/

2) Cambiarse al directorio y descomprimir cambiandole el nombre:

$ cd $CLIPROOT/charsets/

$ zcat cp850.sfm.gz > cp850.uni

Advertencia: Algunos archivos de mapeo unicode, proporcionan mapeo sólo para parte de los 256 símbolos en el "charset". Para tales casos probablemente, debes agregar manualmente los caracteres faltantes.

3) Usa la utilidad gen_tbl, para la generación del archivo *.tbl:

$ zcat UnicodeData-2.1.8.txt.gz | $CLIPROOT/bin/gen_tbl cp850.uni > p850.tbl

4) Setea la variable CLIP_CHARSET, en tu archivo de configuración ".profile" o ".bash_profile" :

xport CLIP_CHARSET=cp850

Este seteo fue probado y me funcionó perfectamente. Falta verlo al imprimir y en la ordenación de claves de índices.

GENERANDO NUESTRO KEYMAPS

Keymaps: Traducido como "mapa de caracteres o teclas", también se corresponde con un archivo donde se define el comportamiento del controlador del teclado (como se debe interpretar las pulsaciones de las teclas).

En el caso del teclado en modo SCAN, su diseño y codificación son hechos por medio de la creación de un archivo "keymap" (mapa de teclas) específico para CLIP, que viene con descripciones predefinidas para controladores de teclado.

Para que una aplicación CLIP pueda ser capaz de usar el modo SCAN, es necesario generar la descripción del "keymap" deseado usando la utilidad "genmap.sh", para obtener nuestro mapa en el directorio "$CLIPROOT/keymaps/".

La configuración del mapa de teclado para ambiente de consola ("terminal") en GNU/linux, se realiza con uno de los siguientes comandos:

 # kbdconfig (Otras distribuciones)
 # dpkg-reconfigure console-data (Debian)
(ver también "install-keymap" y "loadkey")

Producto de esta acción, se selecciona un mapa desde *"/usr/share/keymaps/i386/qwerty/" y se almacena* como *"/etc/console/*.kmap.gz".*

Los "keymaps" son usados en modo de teclado directo, por ahora posible sobre

consola linux y, sobre algunos tipos de terminales que pueden ser switcheados a modo SCAN por secuencias de escape.

CLIP usa el mismo "keymap" del kernel de linux y, el procesamiento de SCAN es tomado prestado desde los fuentes del núcleo. Esto significa, que puedes fácilmente añadir nuevos "keymaps" y modificarlos cuando lo requieras.

¿Cómo crearlo?

1) Copiar el "keymap" deseado desde "/usr/share/keymaps/i386/qwerty/" o desde "/etc/console/" (puedes usar cualquier distribución linux):

cp /etc/console/boottime.kmap.gz $CLIPROOT/keymaps/

2) Cambiarse de directorio y descomprimir:

cd $CLIPROOT/keymaps/
gunzip boottime.kmap.gz

3) Usar "genmap.sh" para la generación de un "keymap" leible para CLIP:
./genmap.sh $CLIPROOT/keymaps/boottime.kmap > es-cp850

Advertencia: El parámetro para genmap.sh, debe incluir el "path" completo.

4) Debo decirle a CLIP que ocupe mi mapa de teclado, en mi archivo de configuracion ".profile" o ".bash_profile", debo setear CLIP_KEYMAP=es-cp850.

NOTA: No es necesario que lo generes. Si tienes configurado tu linux en español, CLIP tomará esas definiciones por defecto. Yo hice la prueba de cambiar el teclado y al iniciar la aplicación, me envía un mensaje de error sobre su uso en conflicto con la terminal. Debe de haber algún seteo que me faltó por definir (?). NO OLVIDAR que el ambiente gráfico (xterminal), maneja su propia configuración.

MOUSE

El mouse es soportado en los siguientes casos:

a) En consola, en modo terminal estándar usando el servicio "gpm".
b) Sobre terminales X-Window (gráficos) con TERM=xterm ó TERM= terminales rxwt.
c) Sobre otros terminales X-Window, en el caso que se haya definido la variable XTERM_MOUSE=ON

d) Desde el emulador "stelnet" (Windows), con la variable de ambiente XTERM_MOUSE=ON.
e) Desde consola a través del protocolo "ssh" sobre un Servidor (se requiere el paquete trans_1.2).
f) Sobre un servidor Windows.

PROGRAMACIÓN ORIENTADA A OBJETOS EN CLIP

INTRODUCCIÓN AL MODELO OO

Primero que todo, algunas palabras sobre el modelo OO que forma parte de CA-Clipper. Este se basa en un arreglo común y corriente y, cualquier llamada similar a obj:atributo u obj:metodo() , resultará en una situación de búsqueda del primer elemento que coincida con el nombre del atributo o método. Tal búsqueda es ejecutada linealmente y es practicamente análoga a:

ascan(obj, { |x| x[1] == "atributo"})

Esta función reduce grandemente la eficiencia del modelo OO basado en Clipper puro. Esto, por supuesto, no es más que una explicación simplificada, pero el sentido es el mismo que se ha descrito.

Nosotros creemos que está claro ahora, el propósito del por qué se realizó una asociación con los arreglos. El modelo OO basado en esta asociación es más rápido de esta forma.

Al mismo tiempo, no necesita expresiones similares a ...
obj:=TClass(nombre_clase):new()
en la clase Tclass misma, lo cual incrementa la eficiencia del modelo OO.

¿Cómo podrías crear tu propia clase? Esto es muy simple de realizar:

```
//------------------------------
function MiClaseNueva()

obj:=map()                    // Genero un objeto vacío
clone(MyClass2New(),obj)      // Adopto la estructura de MyClass2
clone(MyClass3New(),obj)      // Adopto la estructura de MyClass3 (se adiciona)
obj:attribute1:=0                  // Si existen coincidencias de atributos o
métodos,
obj:attribute2:=date()        // los elementos de la última clase prevalecen
obj:method1:=@func1()          // El "metodo1" se convierte en una referencia a
                              // función
obj:method2:=@func2()         // Estas funciones deben ser definidas en el mismo
                              // archivo ".prg" como estáticas
                               // Si los métodos han sido adoptados desde otras
                              // clases,ellos serán reasignados a las clases
                              //indicadas
return obj                    // Retornamos un objeto listo.

//-------------------------

static function func1

::attribute1++
return NIL

static function func2(self)

self:attribute1--
return self
```

También, nosotros hemos querido agregar dos simples reglas:

1.- Un atributo es creado cuando algo, incluido NIL, le es asignado.

2.- En cualquier momento en tiempo de ejecución, el método puede asignar o reasignar cualquier función anunciada en este módulo como función estática, o puede adoptar esta función desde otro objeto, como una usual asignación de valores:

miObj1:metodo1:= miObj2:métodos

¿En qué forma pueden ser usados los objetos ? Como en CA-Clipper, o aún mas

sencillamente:

```
obj := MiClaseNueva()
obj := metodo1()
? obj:atributo1
```

En un objeto, el método "destroy()", puede ser utilizado, pero esto no es lo suficientemente destructor, lo cual es usual en lenguajes de tercera generación. Existe una variable local "obj" a la cual se le asigna un objeto. Al abandonar la función, esta variable (como era local) con todos sus datos es destruida. Ahora, consideremos el caso de un objeto que tiene el siguiente atributo:

```
obj:hFile := fopen( "archivo" )
```

Cuando es destruido "obj", es necesario cerrar "hFile", pero el compilador no sabe esto. El compilador (mejor dicho la máquina virtual) sólo conoce que en "hFile" hay un número y sólo destruirá el número, pero el archivo permanecerá abierto. Sólo para tales propósitos se usará un método "mi_destroy()", y éste será llamado (si es que existe) antes de destruir la variable "obj".

```
static function mi_destroy()
fclose(::hFile)
return
```

CONTROL DEL CAMBIO DE ATRIBUTOS

Si es necesario controlar cambios de los atributos de un objeto, debe usarse el método "modify()" e invocar la función mapmodify(obj, .t.) .

El método modify() es llamado antes de cambiar el valor de cualquier atributo de un objeto. Dos parámetros son pasados a modify(): el código "hash" del atributo a ser cambiado y el nuevo valor a ser asignado. Por ejemplo, modify() debería retornar el valor a asignar al atributo:

```
// En nuestro programa

obj := MyObj()
obj:attr1 := "adios"
? obj:attr1,obj:attr2   // hola mundo
                        // En el prg de creación de la clase
function MyObj()
local obj := map()
obj:attr1 := ""
obj:attr2 := "mundo"
obj:modify := @mymodify()
mapmodify(obj,.t.)      //Activa (.t.) o desactiva (.f.) el modo de control
                        //para el cambio de atributos.
return obj

static function mymodify(self,hash,value)
if hash == hash_ATTR1 .and. value == "adios"
         return "hola"  //  Si el atributo vale "adios", le reasigna "hola".
endif
return value
```

RECUPERANDO Y/O REACTIVANDO OBJETOS

CLIP es capaz de almacenar datos de cualquier tipo en los campos MEMO, incluyendo objetos (los convierte en cadenas). Pero no hay forma de almacenar directamente los métodos (las funciones), pues éstos pueden ser cambiados.

Pero existe una manera. Para poder realizar el proceso de recuperación de los objetos con sus correspondientes métodos, se debe proceder de la siguiente forma:

Los datos son decodificados. Si los datos son del tipo objeto y éste tiene el atributo CLASSNAME, entonces la función "recover&(var:CLASSNAME)(var)" es llamada. Esta función cumple con reasignar los métodos a este objeto.

Esta característica puede ser usada para enviar objetos como cadenas ("strings") vía correo electrónico o TCP. A continuación un ejemplo de su uso:

```
// Creo el objeto y lo manipulo

x:=asdfNew()
            // Visualizo la ejecución de los métodos, veo si funciona.
? "x:m1",x:m1()
? "x:m2",x:m2()

// Convierto el objeto -> cadena
y:=var2str(x)    // También podría ser: campo->memo_campo:=x

? "y=",y      // Visualizo la cadena resultado de la conversión.

//Reconvierto cadena -> objeto
z:=str2var(y)     // _recover_asdf() es llamada automáticamente
                  // También podría ser: z:=campo->memo_campo
? "z=",z
? "z:m1",z:m1()    // Verificar si funcionan los métodos
? "z:m2",z:m2()
return

//-------- Creación de la clase  (Esto en un solo prg)--------

function asdfNew()
local o:=map()
o:classname := "ASDF"
o:a1 := "asdf"
o:a2 := "qwer"
_recover_asdf(o)
return o

function _recover_asdf(o)
o:m1 :=@asdf_1()
o:m2 :=@asdf_2()
? "recuperando"
return o

static function asdf_1
? "asdf_1",::a1
return ::a1

static function asdf_2
```

```
? "asdf_2",::a2
return ::a1

//-------------------- Fin prg --------------------
```

OPERADORES DE SOBRECARGA PARA OBJETOS

CLIP soporta sobrecarga (implementación) de ciertas operaciones con los objetos (lógicas y matemáticas), previa inclusión del método en la creación de la clase. Las operaciones que pueden ser sobrecargadas y sus correspondientes métodos operador, son listados en la siguiente tabla :

*** Tabla NO incluida en texto plano ***

(Rogamos disculpar las molestias y remitirse al PDF del tema)
(Operaciones: +,-,*,/,%,^,|,&,=,==,!=,<,>,<=,>=)

A continuación, un ejemplo del uso de la sobrecarga de operaciones al usar objetos:

```
//---- En el prg que lo usa ------

oCar1 := newBMW( )
oCar2 := newKAMAZ( )
? oCar1 > oCar2            // .F. (El "peso" del Kamaz es mayor)
oCar := oCar1 + oCar2
? oCar:model              //"Scrap"
? oCar:weight             // 10000

// 10000 kilos del chatarra de fierro.

//----- En el prg que Crea la Clase ----


function newCar()
local obj := map()
obj:model := ""
obj:weight := 0
obj:operator_gt := @car_gt()     //Aquí defino que hacer cuando uso ">"
obj:operator_add := @car_add()   //Aquí defino que hacer cuando uso "+".
return obj


static function car_gt(car)
return ::weight > car:weight

static function car_add(car)
local obj := newCar()
obj:model := "Scrap"
obj:weight := ::weight+car:weight
return obj

//--------------

function newBMW( )
local obj := newCar()            // Adopta la clase "Car"
obj:model := "BMW"
obj:weight := 2000
return obj
```

```
function newKAMAZ( )
local obj := newCar()           // Adopta la clase "Car"
obj:model := "KAMAZ"
obj:weight := 8000
return obj
```

CONCLUSIÓN

Debido al diseño del modelo OO y su compilación a un programa "C", existe la posibilidad de escribir las clases estándar Tbrowse y Get en el mismo lenguaje Clipper. Al mismo tiempo, la eficiencia de estas clases no es peor que aquellas escritas en puro "C" para CA-Clipper.

SECUENCIA DE INICIO DE UN PROGRAMA CLIP

Cuando un programa se inicia, el sistema de ejecución de CLIP, chequea la existencia de algunos archivos y trata su contenido de acuerdo a las asignaciones de ciertas variables de ambiente.

El primer archivo útil es "$CLIPROOT/lang/$LANG". Este es un script que carga valores predeterminados para ciertas variables de ambiente relacionadas con el lenguaje. Si LANG está vacío, CLIP chequeará también la variable CHARSET. Por ejemplo, veamos el contenido del archivo ru_RU.KOI8-R:

```
#CLIP_KEYMAP se usa en modo de teclado "raw" (directo).
CLIP_KEYMAP=ru-koi8-r
```

El próximo archivo es $CLIPROOT/term/$TERM, donde TERM es la variable que identifica el tipo de terminal usado y lo asocia a un archivo, que contiene un script para el comportamiento del teclado. Por ejemplo, veamos en contenido de $CLIPROOT/term/linux-console:

```
# switchea sobre la consola linux un teclado "raw" (directo).
CLIP_SCANMODE=IOCTL
```

TECLADO

Las aplicaciones CLIP tienen dos tipos de controladores de teclado: el modo ANSI y el modo SCAN.

En el modo ANSI, los códigos de las teclas son aceptados desde el terminal y dos modificadores Ctrl-J y Ctrl-K son usados para extender las posibilidades.

Un modificador cambia el código de la tecla, para las aplicaciones CLIP. Por ejemplo: presionando la tecla "1" se genera el código 49, presionando "Ctrl-J + 1" se genera el código 376, presionando "Ctrl-K + 1" también genera el código 376. El modificador Ctrl-J trabaja de la misma forma que ALT y, el modificador K trabaja de la misma forma que CTRL, pero no con todas las teclas.

El modo de trabajo de los teclados más avanzados es SCAN. En este modo, el terminal pasa al servidor códigos de teclas pulsadas y liberadas. Todas las combinaciones de teclas están disponibles en el caso de poseer correctas descripciones de "keymap" (mapas de teclas). Además hay posibilidad de agregar código de símbolos, usando "Alt + teclas". En estos casos se usan las descripciones de diseño de consola ubicados en "/usr/share/keymaps/i386/qwerty/*". Damos por hecho que te estás moviendo en

la plataforma i386 y con teclados cuyo orden físico, debajo de los números, es "qwerty...".

Para trabajar con un terminal con capacidades SCAN, uno puede usar la consola de linux o nuestro emulador de terminal (bajo Windows) Stelnet (bajarlo de ftp://ftp.itk.ru/pub/telnet).

Para poder usar las capacidades avanzadas de Stelnet (Windows), hay que ejecutarlo de la siguiente forma:

stelnet -s -t linux-stelnet nombre_pc_servidor

Y lanzar la aplicación CLIP como sigue:

```
export TERM=linux-stelnet
mi_programa_clip
export TERM=linux
```

TERMINALES MODO TEXTO EN CLIP

INTRODUCCIÓN

Cuando vamos a desarrollar una aplicación Clip, uno de los elementos más importantes es la interfaz de usuario, lo que va a apreciar el usuario por pantalla. En el caso del mundo DOS, estábamos acostumbrados a verlo en una representación de "caracteres" por filas y columnas, ya sea de 25*80 ó 50*80.

En una primera etapa, supondremos que nuestras aplicaciones "texto" Clipper serán portadas a "texto" Clip. Lo normal en linux, por defecto, es que las terminales virtuales vengan configuradas como 25 filas por 80 columnas y en modo emulación "linux".

Además debemos mantener en mente, que nuestra idea es que algunos equipos Windows (que nunca faltan) se conecten a un Servidor Central, por lo que deberán usar un "cliente" para esa conexión. Aquí viene el detalle, debemos estar conscientes de los modos de configuración de éste cliente para que esa relación sea compatible.

Por ejemplo, existe un cliente para conexión remota segura (para Windows) llamado Putty. Este simula una terminal grafica de linux (xterminal), que se puede configurar para emulación modo "linux" (entre otras) y variable número de filas y columnas.

Al portar o programar en Clip, debemos tener en cuenta esta potencial variabilidad de interfaz de salida. Por eso se recomienda encarecidamente, evitar el uso de posicionamiento fijo y recurrir a las funciones de posicionamiento relativo: maxrow() y maxcol().

Si aún después de todas estas recomendaciones, aún tenemos ganas de cambiar la apariencia de nuestra terminal de texto linux, entonces lean lo que viene a continuación.

TIPS PARA CAMBIAR RESOLUCION

(1) Usando nuestro cargador de arranque.

LILO: Editamos el archivo "/etc/lilo.conf" y agregamos o modificamos:

vga=modo

GRUB: Editamos el archivo "/boot/grub/menu.lst" y agregamos o modificamos:

```
title            Debian GNU/Linux, kernel 2.4.27-2-386
root              (hd0,5)
kernel           /boot/vmlinuz-2.4.27-2-386 root=/dev/hda6 vga=modo ro
initrd           /boot/initrd.img-2.4.27-2-386
savedefault
boot
```

Estas característica, permiten el uso de varios modos de video especiales soportados por el video de la BIOS. Debido a esto, la selección está limitada al tiempo de buteo y trabaja sólo sobre maquinas 386.

Aquí en el fondo, lo que estamos haciendo es ocupar una area de memoria RAM para uso de VIDEO, algo como convertir nuestra terminal de texto en gráfica. Tanto es así, que en estas condiciones hasta podemos colocar imágenes o ver películas en este ex inhóspito ambiente.
A esta capacidad, de alto nivel, se le denomina "frame buffer". Existen algunas tablas para que se guíen en lo que deben colocar en "modo" (argumento de VGA). No olvidar que esta capacidad debe estar incluida en nuestro kernel, nosotros lo único que hacemos es "activarla".

Modos (como constantes para filas y columnas):
normal *80x25*
extended ó ext *80x50*
ask Menú de selección.
Modos (para filas y columnas):
0x0f00 estándar 80x25
0x0f01 estándar con font de 8 puntos: 80x43 sobre EGA, 80x50 sobre VGA
0x0f02 VGA 80x43 (VGA switcheado a 350 líneas "scan" con font de 8 puntos)
0x0f03 VGA 80x28 (estándar VGA "scans", pero con font de 14 puntos)
0x0f04 Deja el corriente modo de video
0x0f05 VGA 80x30 (480 "scans", 16-puntos font)
0x0f06 VGA 80x34 (480 "scans", 14-puntos font)
0x0f07 VGA 80x60 (480 "scans", 8-puntos font)
0x0f08 Modificar gráficos

Si por alguna razón LILO no soporta el modo en forma hexadecimal, se lo pueden dar en formato decimal. Sería del 3840 al 3848.

Todas estas configuraciones dependen mucho del harware que tengan. Yo las probé en 3 equipos, y en solo 1 me resultó. En los otros 2 cambiaba la resolución al principio pero luego se reseteaba a normal.

El sistema que me resultó en todos, es el que se basa en cambiar la resolución de acuerdo a la siguiente tabla:

HEXA	640x480	800x600	1024x768	1280x1024
256	0x301	0x303	0x305	0x307
32k	0x310	0x313	0x316	0x319
64k	0x311	0x314	0x317	0x31A
16M	0x312	0x315	0x318	0x31B

DECI	640x480	800x600	1024x768	1280x1024
256	769	771	773	775
32k	784	787	790	793

```
64k |    785  |    788  |    791   |    794
16M |    786  |    789  |    792   |    795
----+---------+---------+----------+----------
```

En un monitor de 15 pulgadas, la resolución de 640x480 equivale a 80x30 y la de 800x600 a 80x37 columnas x filas. Ustedes practiquen todo lo que quieran. Si han bajado los fuentes del núcleo, echen una mirada al archivo "/usr/src/linux/Documentation/svga.txt".

(2) Usando una emulación por software.

resizecons : Haría lo mismo que "resize", no lo probé.

resize : Lo único que hace es modificar las variables de ambiente COLUMNS y LINES. En la práctica no es muy útil.

svgatextmode : Este si que es de verdad. En debian está en los repositorios (al menos de Sarge), así que es llegar y bajarlo. Posee un archivo de configuración "/etc/TextConfig" bien autoexplicativo. No es llegar y ejecutarlo desde consola, primero se deben indagar los valores posibles viendo su archivo de configuración. Por ejemplo:

```
# SVGATextMode 80x34x9
```

Probé varios modos, y todos me funcionaron. Lo bueno es que posee una variedad muy grande de configuración, incluyendo los tipos y tamaños de fonts:

Filas : 25, 28, 29, 30, 32, 33, 34, 40, 43, 50, 60
Columnas : 40, 80, 100, 116, 132

console-tools : este paquete es un clon de kbd y contiene varias herramientas. La más util y sencilla, es para cambiar el font de las letras del terminal. Se debe modificar el archivo "/etc/console-tools/config" con lo siguiente:

SCREEN_FONT=<fuente>

Donde "fuente" es el nombre del archivo sin la extensión, ubicado en "/usr/share/consolefonts". Generalmente son archivos *.psf.gz.

LA VARIABLE TERM

Para linux en general, existe la variable de entorno denominada TERM . Linux sabe que tipo de terminal estamos utilizando, o mejor dicho, que tipo de "emulación de terminal" estamos usando (si usamos nuestras consolas virtuales), así los programas toman el valor de esta variable para ver en la base de datos "terminfo", cuales son las capacidades de la terminal en cuestión.

Si digitamos."echo $TERM" en cualquiera de las terminales virtuales, veremos que aparece la palabra "linux", ya que es el tipo de emulación usado por defecto por los Pcs. Muy relacionadas están también las variables LINES y COLUMNS, cuyo significado es bastante obvio.

TERMCAP Y TERMINFO

No todas las terminales tienen las mismas capacidades, es decir, no todas manejan los mismos caracteres de escape y demases, si el sistema operativo (SO) envía un caracter o secuencia que un terminal no reconoce, pues obviamente éste no va a hacer nada, y tal vez en el mejor de los casos, es posible que algo se desconfigure en la pantalla (¿?).

Este tipo de situación fue muy común, aún después de haber realizado una serie de estandarizaciones ANSI e ISO.

Para hacer frente a este problema de estándares, se creó una base de datos llamada "termcap", es decir "Teminal Capabilities" (Capacidades de un Terminal), la cual fue sucedida por otra base de datos, más nueva, llamada "terminfo" (Información de Terminales).

Estas bases de datos se encuentran en algunos archivos del sistema, por ejemplo en Debian Sarge están ubicadas en "/etc/terminfo/*" ó "/usr/lib/terminfo/*" , que es un directorio donde están ordenados por abecedario las rutinas de bajo nivel sobre definiciones de los principales tipos de terminales y cuenta con una sección para cada tipo de terminal que se esté usando. Así el sistema sabe de que es capaz el terminal y no envía señales equívocas. Los programas de aplicación utilizan esta base de datos, mediante el llamado a ciertas librerias C. Para más información, en un terminal digitar "man terminfo".

CONFIGURANDO TERMCAP PARA CLIP
En el caso de Clip, este usa la base de datos "terminfo". Una copia del sistema se instala en el directorio $CLIPROOT/etc/terminfo, al momento de la compilación e instalación de Clip. Si defines la variable de ambiente TERMCAP en tu archivo de configuración ".bash_profile" o similar, se pueden usar algunas de las definiciones del archivo "$CLIPROOT/etc/termcap" en ambiente de texto. Por ejemplo:

```
trueansi    ansi|xenix      cons25r     xenix8    AT386|at386    AT386c|at386c
xterm       xterm-          std         xterms     vt100     vt200   vt340
pc3|ibmpc3    pc3a|ibmpc3a      pc3nc         console|linux|linux-koi8|linux-c|
linux-koi8r|fslinux
con80x25 con80x28  con80x43 con80x50  con100x37 con100x40 con132x43     ct120
```

Cada una de éstas etiquetas viene seguida por una configuración que hace uso de la base de datos (a bajo nivel) "terminfo". Para una descripción detallada de todas las opciones, ver "man termcap" o
http://www.linuxinfor.com/english/man5/termcap.html .

Todos los nombres de los "terminales avanzados" tienen que estar registrados en la base de datos
"/etc/terminfo/". Las tablas-unicódigo estándar de linux localizadas en "/usr/share/consoletrans" son usadas para la traducción de las salidas del terminal.

Las tablas-unicódigo necesarias, han sido desempacadas y puestas en el directorio $CLIPROOT/charsets. La distribución de Clip viene con algunas tablas preexistentes. Por el momento NO seteen la variable TERMCAP, más

---

MANEJADORES DE BASES DE DATOS REEMPLAZABLES (RDD)

INTRODUCCIÓN
RDD es la abreviación inglesa del término "Replaceable Database Drivers" (Manejadores de Bases de Datos Reemplazables), que son usados para procesar distintos formatos de base de datos usando un controlador intercambiable. RDD es una capa intermedia, que traduce las llamadas del API de CLIP (relacionadas con bases de datos) a operaciones de archivo de bajo nivel.

La arquitectura usada en CLIP difiere de aquella usada en Clipper. CLIP RDD (desde el punto de vista de Clipper) está formado de tres diferentes subcontroladores: (a) el controlador de la tabla, (b) el controlador del

índice y (c) el controlador del memo. Estos controladores pueden ser fácilmente combinados en cualquier orden.

Además, CLIP posee herramientas adicionales de desarrollo. Un conjunto de funciones RDD que operan con los "descriptores" de bases de datos, en vez de "áreas de trabajo".

Algunas de las más interesantes características del subsistema CLIP-RDD, son las siguientes:

* Optimizador de consultas (tecnología similar a Rushmore/MachSIx).
* Campos VARCHAR (almacenamiento compacto de cadenas de hasta 64KB de longitud).
* Campos VARIANT, que permiten almacenar datos de cualquier tipo básico xBase (CHARACTER, NUMERIC, DATE, DATETIME ó LOGIC).
* Almacenamiento de datos de cualquier tipo (incluyendo objetos), en campos memo.
* Ponga a funcionar sus propios filtros.
* MEMOPACK (empaquetamiento de archivos memo) y flexible FTP (reutilización de bloques descartados que reducen el abultamiento del archivo memo).
* Soporte de SET RELATION con "scope" (especie de filtro con límites de acuerdo a un índice). Sólo los registros de las tablas hijas que cumplen la condición son visibles.
* Ponga a funcionar sus propios índices (comportamiento personalizado).
* Bloqueo de multiples registros (a la vez).
* Gatilladores ó disparadores ("triggers") de eventos para bases de datos.
* Edición de registros de restauraciones no actualizadas ("rollbacks").
* Tablas temporales (que son borradas al cierre -CLOSE-, automáticamente).

CARACTERISTICAS DE LOS RDDs

Existen los siguientes submanejadores:

Manejadores de tablas: DBF, FOX y VFP.

Manejadores de índices: NTX, CTX, CDX e IDX.

Manejadores de memos: DBT y FTP

Los siguientes RDDs (según Clipper) están constituidos por estos submanejadores:

DBFNTX: DBF + NTX + DBT

DBFCTX: DBF + CTX + DBT

DBFCDX: FOX + CDX + FPT

DBFIDX: FOX + IDX + FPT

SIXCDX: FOX + CDX + FPT (sinónimo de DBFCDX)

FPCDX: VFP + CDX + FPT

MANEJADORES DE TABLAS

CLIP discrimina el tipo de tabla a ser abierta usando la firma del archivo, p.e. la tabla es abierta exitosamente aún si el RDD especificado es diferente. Pero esto no es aplicable a los archivos memo e índices.

Los manejadores DBF y FOX son mayoritariamente lo mismo. La única diferencia es la firma del archivo para tablas con memo (0xF5 para FOX y 0x83 para DBF). El manejador VFP opera con tablas "Visual FoxPro". La tabla VFP tiene un encabezado distinto y proporciona algunas posibilidades que no existen en el usual formato DBF, p.e. la posibilidad de creación de campos binarios y nulos (el 5° y 6° elemento en la descripción de un campo, usado con DBCREATE() y DBSTRUCT()). Todos los manejadores de tablas, soportan todos los tipos de campos descritos a continuación:

Tipo de campo	Identificador(es)	Tamaño en tabla	Descripción
CHARACTER	'C'	1 ... 65534	Cadena de caracteres
VARCHAR[a]	'V' [b]	1 ... 65534	Cadena de caracteres de longitud variable
NUMERIC	'N', 'F' [c]	1 ... 20	Enteros o fracciones
CURRENCY	'Y'[c]>	8	Cantidades monetarias
DOUBLE	'B'[c]>	8	Números de doble presición punto flotante
INTEGER	'I'[c]>, 'V(4)'[b]>	4	Valores enteros
DATE	'D', 'V(3)'[b]>	8(3 bytes,V(3))	ato cronológico: año, mes y día
DATETIME	'T'[c]>	8	Dato cronológico: año, mes, día y hora
LOGICAL	'L'	1	Valor lógico de Verdadero o falso
MEMO	'M'	10 (4 con VFP)	Dato de cualquier tipo [d] sin límite de tamaño
BLOB[e]	'P'[c]>, 'G'[c]>	10 (4 con VFP)	Dato binario cualquier tipo [d]> sin límite tamaño
VARIANT[f]	'X'	10 ... 127	Dato de cualquier tipo xBase

Notas:

a. El tipo VARCHAR permite almacenar caracteres de cualquier tamaño. Para esto, los primeros 6 bytes del <tamaño_del_campo> son almacenados en el archivo DBF mismo y el resto es almacenado en el archivo MEMO. Al momento de indexar, sólo aquellos bytes almacenados en la DBF son usados.

b. Prestado de la librería Six.

c. Prestado de VFP.

d. Disponible con el manejador memo FPT. Sólo CHARACTER con DBT.

e. El tipo BLOB es siempre binario, sin importar el estado de binario en la descripción del campo pasado a la función DBCREATE().

f. El tipo VARIANT permite almacenar datos de cualquier tipo xBase. Las cadenas de caracteres pueden ser de tamaño 2 del total <tamaño_de_campo> (dos bytes son reservados para almacenar el tipo y longitud). Los campos VARIANT pueden ser indexados. El orden de los valores de varios tipos en un índice, es el siguiente: CHARACTER, LOGICAL, DATE, DATETIME y NUMERIC.

MANEJADORES DE INDICES

La siguiente tabla, resume la disponibilidad de características claves a través de los manejadores de índices. Recordemos antes, dos conceptos importantes:

ORDER: Se usa para identificar cada uno de los índices ("tag") contenidos dentro de un único archivo físico denominado "bag" (saco o bolsa de "Tags") .
TAG: Representa la etiqueta o identificación de una clave índice dentro de un archivo contenedor ("Bag"). Este archivo "bag" (saco), puede almacenar de uno a varios índices ("tags") a la vez.

Item	NTX	CTX	IDX	CDX
Administración de "order" (soporte de "tags")	No	Si	No	Si
Número de "tags" por archivo bag"	1	63	1	Ilimitado
Número de archivos "bags" por área de Trabajo	Ilimit.	Ilimit.	Ilimit.	Ilimitado
Índices condicionales (claúsula FOR)	Si	Si	Si	Si
Índices temporales (parciales) (claúsula WHILE)	Si	Si	Si	Si
Uso de "Descendiendo" vía claúsula DESCENDING	Si	Si	Si	Si
Uso de "Único" vía la claúsula UNIQUE	Si	Si	Si	Si
Soporte de las claúsulas EVAL y EVERY	Si	Si	Si	Si
Índices estructurados (compuestos)	No	Si	No	Si
Máxima longitud de la expresión clave (bytes)	256	256	255	255
Máxima longitud para la condición FOR (bytes)	256	256	255	255
Soporte de relaciones "scope" (filtrar rango)	Si	Si	Si	Si
Optimización de las claúsulas FILTER y FOR (tecnología MachSIx/Rushmore)	No	No	Si	Si

MANEJADORES DE MEMOS

Existen dos manejadores de archivos memo, DBT y FPT. Los archivos memo DBT son usados con los RDDs DBFNTX y DBFCTX. El formato del archivo DBT es totalmente compatible con Clipper.

Los archivos memo FPT usados con los RDDs DBFCDX y DBFIDX, poseen un formato más poderoso que puede almacenar no sólo datos de texto, si no que cualquier otro dato. El archivo FPT puede almacenar cualquier tipo de dato CLIP, incluyendo objetos, datos binarios (p.e. imágenes), etc.

TRADUCCION DE MENSAJES

El compilador de CLIP, almacena todas las constantes de cadena definidas como [cadena_de_datos] en los subdirectorios "$CLIPROOT/locale.po*".

Por ejemplo para las personas de habla hispana, podríamos usar los siguientes archivos ubicados en "$CLIPROOT/locale.po/ISO-8859-1/", para la traducción de los mensajes:

clip.po
cliprt.po
sys.po

De manera de que si existe la posibilidad, de hacer traducciones a otros lenguajes. Al momento de ejecutarse una aplicación, CLIP lee la variable de ambiente LANG y obtiene las apropiadas constantes desde los archivos antes mencionados.

Gettext y utilidades para los archivos de traducción "po".

Gettext es un paquete común, ampliamente usado para los mensajes de localización. El paquete tiene dos partes principales: las utilidades de extracción y preparación del mensaje y, una librería para el manejo de mensajes en tiempo de ejecución.

CLIP usa algunas de las utilidades de "gettext" para la extracción de mensajes

y preparación de archivos de mensajes, pero usa código propio para el manejo de los mensajes en tiempo de ejecución.
CLIP tiene numerosas utilidades para hacer el manejo de mensajes más fácil:

Ruta para obtener mensajes locales:

Extracción de mensajes y creación del archivo "pot" (PO-planTilla). Para la extracción desde fuentes C, se puede usar el comando "xgettext" desde el paquete "gettext". CLIP automáticamente extrae los mensajes encerrados entre paréntesis cuadrados "[]" y los almacena en el archivo $CLIPROOT/locale/pot/<nombremódulo>/<nombrearchivo>.pot . El <nombremódulo> es determinado por CLIP desde el nombre del directorio de trabajo en uso, o puede ser especificado por la variable de ambiente CLIP_MODULE. El <nombrearchivo> se obtiene del nombre del archivo compilado.

Usar el script "clip_msgmerge" para mezclar los archivos pot desde el directorio $CLIPROOT/locale.pot con los archivos po traducidos en el directorio $CLIPROOT/locale.po. El "clip_msgmerge" mezcla archivos para todos los locales en el directorio $CLIPROOT/locale.po/<local>. Para añadir un nuevo lical, necesitas crear un directorio apropiado, por ejemplo; $CLIPROOT/locale.po/ru_RU.KOI8-R para usar las locales de ru_RU.KOI8-R.

Editar los archivos creados po, con cualquier editor de texto, o usar un editor especializado en archivos po, similar al kbabel de KDE.

NOTA: Para la definición del "charset" en el archivo po, por favor chequea el campo "Content-Type: test/plain, charset=ASCII" en la primera entrada del archivo. Debes especificar un valor válido de "charset" usado. Por ejemplo "Content-Type: test/plain, charset=KOI8-R"). Esto es muy importante para corregir el manejo de los mensajes.

Y el último paso es invocar el script "clip_msgfmt", para formatear mensajes dentro del archivo de mensajes compilado, en el directorio "$CLIPROOT/locale.mo/".

Ahora CLIP, en tiempo de ejecución, obtendrá los mensajes necesarios desde estos archivos y los sustituirá en tu programa.
