Escribir Programas con NCURSES

Eric S. Raymond y Zeyd M. Ben-Hamlim

CONTENIDOS

Introducción

Este documento es una introducción para programar con Curses. No es una referencia exhaustiva para la Interfaz de Programación de Aplicaciones de Curses (API); este papel esta cubierto por las páginas del manual de curses. Mejor dicho, pretende ayudar a programadores de C facilitándoles el uso del paquete.

Este documento está dirigido a programadores de aplicaciones en C que no están especialmente familiarizados con *Ncurses.* Si usted es un especialista programador de *Curses*, no necesita leer las secciones de Interfaz_de_Ratón, Depuración, Compatibilidad_con_versiones_anteriores, Avisos_Consejos_Y_Trucos. Esto le llevará con velocidad a las características y rasgos especiales de la implementación de *Ncurses.* Si usted no tiene tanta experiencia, continúe leyendo.

El paquete *Curses* es una subrutina de librería para el terminal independiente de pintura de pantalla y manejo de sucesos de entrada que presenta un modelo de pantalla de alto nivel al programador, ocultando diferencias entre los diferentes tipos de terminales y haciendo optimización automática de la salida de una pantalla llena de texto en otra. *Curses* utiliza un terminal de información, que es un formato de base de datos que puede describir las capacidades de miles de terminales diferentes.

El Curses API puede parecerse a los terminales arcaicos de los entornos UNIX cada vez más dominados por X, Motif y Tcl/Tk.

Sin embargo, UNIX todavía soporta líneas TTY y X soporta xterm(1); El Curses API tiene la ventaja de : a) respaldo-portabilidad de la celda del carácter de un terminal, y b) simplicidad. Para una aplicación que no necesite gráficos mapas de bits y fuentes múltiples, una implementación usando curses será típicamente un gran negocio más simple y menos caro que uno usando una herramienta X.

Breve historia de Curses

Históricamente, el primer antecesor de curses fueron las rutinas escritas para proporcionar el manejo de pantalla para el juego ?rogue?; esto utilizaba las ya existentes capacidades de la base de datos de capacidades del terminal (termcap) para describir capacidades del terminal. Las rutinas fueron abstraídas dentro de una librería documentada y lanzadas por primera vez con las versiones tempranas de BSD UNIX.

System III UNIX de Bell Labs constaba de una librería curses reescrita y mucho más mejorada. Esta incluía el formato del terminal de información(terminfo). Este terminal se basa en la base de datos de capacidades del terminal (termcap) de Berkeley, pero contiene mejoras y extensiones. Cadenas de caracteres con capacidades parametrizadas fueron introducidas, haciendo posible describir múltiples atributos de vídeo, colores y manejar muchos mas terminales no usuales que con el terminal anterior (termcap). En los posteriores lanzamientos de AT&T System V , curses desarrolla el uso de mas facilidades y ofrece más capacidades, llegando incluso más allá que las curses de BSD en poder y flexibilidad.

Alcance de este documento

Este documento describe ncurses, una implementación gratuita de las curses API de

System V con algunas extensiones claramente marcadas. Incluye las siguientes características de las curses de System V:

· Soporte para múltiple atributo de pantalla (highlight) de pantalla (curses de BSD solo manejaban unos atributos de pantalla de salida, normalmente video-inverso).

· Soporte de dibujo de líneas y cajas usando caracteres de formularios.

· Reconocimiento de teclas de funciones en la entrada.

· Soporte de color.

· Soporte de blocs (pads) (ventanas con el largo de la pantalla en las que la pantalla o una subventana define una ventana de vista).

Además, este paquete realiza el uso de las características de insertar y borrar líneas y caracteres de terminales mas equipados, y determina cómo para optimizar utilizan estas características sin ayuda del programador. Esto permite combinaciones arbitrarias de atributos de vídeo para ser presentados, incluso en terminales que dejan ?magic cookies? en la pantalla para marcar cambios en los atributos.

El paquete ncurses puede también capturar y usar eventos de un ratón en algunos entornos (notablemente, xterm bajo el sistema de ventanas X). Este documento incluye consejos para el uso del ratón.

El paquete ncurses fue creado por Pavel Curtis. La persona que mantiene originalmente el paquete es Zeyd Ben-Halim <zmbenhal@clark.net>. Eric S. Raymon esr@snark.thyrus.com escribió muchas de las nuevas características en las versiones posteriores a 1.8.1 y escribió la mayor parte de esta introducción. Las personas que lo mantienen ahora primario actuales son Thomas Dickey <dickey@clark.net> y Juergen Pfeifer >Juergen.Pfeifer@T-Online.de>

Este documento describe también la librería de extensiones Paneles, similarmente producida en la capacidad Paneles de SVr4. Esta librería te permite asociar almacenamiento secundario con cada pila o superficie de ventanas sobrelapadas, y proporciona operaciones de movimiento de ventanas alrededor de la pila y cambios en su visibilidad de modo natural (manejando ventanas solapadas).

Finalmente, este documento describe en detalle las librerías de Menú y Formulario (forms), también copiadas de System V, que soporta fácil construcción y secuencias de menús y formularios rellenables. Este código fue contribuido al proyecto por Jügen Pfeifer.

Terminología

En este documento, la siguiente terminología es utilizada con consistencia razonable:

Ventana

Estructura de datos que describe un subrectángulo de la pantalla (posiblemente la pantalla entera). Puede escribir en la ventana como si fuera una pantalla en miniatura, haciendo el barrido independiente de otras ventanas en la pantalla física.

Pantallas

Subconjunto de ventanas que son tan grandes como el terminal de pantalla, por ejemplo, comienzan en la esquina izquierda superior y abarca hasta la esquina izquierda inferior. Una de estas, stdscr, es suministrada automáticamente para el programador.

Pantalla Terminal

La idea del paquete de que presentación de terminal aparece actualmente, por ejemplo, cual ve el usuario ahora. Esto es una pantalla especial.

Notas sobre esta traducción

El documento original del que se ha realizado esta traducción se llama "Writing Programs with ncurses" cuyos autores, tal y como se explica en otros apartados, fueron Eric S. Raymond y Zeyd M. Ben-Hamlim. Este documento se puede encontrar en diversas direcciones de internet, a continuación escribimos algunas de estas:

> http://bat710.univ-lyon1.fr/~ascil/ncurses/
>
> http://www.aaronsrod.com/freemoney/ncurses-intro.html
>
> http://aotech1.tuwien.ac.at/~dusty/ncurses-intro.html

Otras direcciones interesantes que contiene documentos sobre la librería ncurses, incluso documentos con el mismo titulo que este pero que en realidad no son igual que este, son:

> http://www.ecks.org/docs/ncurses-hack.html (Documento de ncurses para hackers)

Esta traducción ha sido realizada por Patricia Martínez Cano, alumna de la Facultad de Informática de la Universidad de Murcia, bajo la coordinación de Juan Piernas Cánovas. La publicación de esta traducción ha sido autorizada por Eric S. Raymond

La Librería Curses

Una descripción de Curses

Compilar programas utilizando Curses

Para utilizar la librería, es necesario tener ciertos tipos y variables definidas. Por ello, el programador debe tener una línea:

#include <curses.h>

al principio del programa. El paquete de pantalla utiliza la librería estándar I/O, por consiguiente <curses.h> incluye <stdio.h>. >curses.h> también incluye <termios.h>, <termio.h> o <sgtty.h> dependiendo de su sistema. Es redundante (pero inofensivo) para el programador incluirlas, también. Al enlazar con curses usted necesita tener -lncurses en su LDFLAGS o en la línea de comando. No se necesita para otras librerías.

Actualización de pantalla

Para actualizar la pantalla óptimamente, es necesario para las rutinas saber que aspecto tiene la pantalla actual y que aspecto quiere el programador que tenga después. Para este propósito, una tipo de estructura de datos llamada ?window? se define como la que describe la imagen de una ventana para las rutinas, incluyendo su posición de comienzo en la pantalla (la coordenada (y,x) de la esquina superior izquierda) y su tamaño. Una de estas (llamada curscr, para la pantalla actual) es una imagen de pantalla de cómo el terminal actual aparece. Otra pantalla (llamada stdscr, para pantalla estándar) es suministrada por defecto para hacer cambios en ella.

Una ventana es puramente una representación interna. Se utiliza para construir y almacenar una imagen potencial de una porción del terminal. No lleva necesariamente una relación con que esta realmente en el terminal; es mas como un cuaderno de apuntes o un buffer escrito.

Para hacer que la sección de la pantalla física correspondiente a una ventana refleje el contenido de la estructura de la ventana, la rutina refresh() (o wrefresh() si la ventana no es stdscr) es llamada.

Una sección física dada puede estar sin el alcance de un numero de ventanas solapadas. Además, se pueden realizar cambios en las ventanas en cualquier orden, sin poner atención en la eficiencia. Entonces, el programador puede efectivamente decir ?haz que parezca esto?, y dejar a la implementación del paquete determinar el camino mas eficiente de repintar la pantalla.

Ventanas estándar y convenciones de nombramiento de Funciones

Como se ha indicado anteriormente, las rutinas puedes usar distintas ventanas, pero dos son dadas automáticamente: curscr, la cual sabe el aspecto del terminal, y stdscr, la cual es el aspecto que el programador quiere que tenga el terminal después. El usuario no debe nunca realmente acceder directamente a curscr. Los cambio deben ser realizados a través de API, y entonces la rutina de refresco (refresh()) o wrefresh) será llamada.

Muchas funciones son definidas para utilizar stdscr como pantalla por defecto. Por ejemplo, para añadir un carácter a stdsc, se realiza una llamada a addch() con el carácter deseado como argumento. Para escribir en una ventana diferente el uso de la rutina waddch() (para una ventana específica ,?w?indow-specific addch()) es permitido. Esta convención de prefijar los nombres de las funciones con una ?w? cuando se aplican a ventanas específicas es consistente. Las únicas rutinas que no sigue esto son aquellas en las que una ventana debe ser siempre especificadas.

Para mover las coordenadas actuales (y,x) de un punto a otro, se proporcionan las rutinas move() y wmove().

Sin embargo, es a veces deseable para el primer movimiento y después actuar con alguna operación de entrada salida(I/O). Para evitar torpezas, muchas de las rutinas de entrada salida pueden ser precedidas por el prefijo ?mv? y la coordenada (y,x) deseada como argumento de la función. Por ejemplo, la llamada

move(y,x);

addch(ch);

puede ser reemplazada por

mvaddch(y,x,ch);

y

wmove(win,y,x);

waddch(win,ch);

puede ser reemplazada por

mvwaddch(win,y,x,ch);

Nótese que el puntero de descripción de la ventana (win) va después de las coordenadas añadidas (y,x). Si la función requiere de un puntero de ventana, siempre será pasado como primer parámetro.

Variables

La librería curses presenta algunas variables que describen las capacidades del terminal.

Tipo nombre descripción

---

int LINES numero de líneas en el terminal

int COLS numero de columnas en el terminal

La librería curses.h : además introduce algunos constantes definidas y tipos de uso general:

bool

Tipo booleano, realmente es un `char' (e.g., bool doneit;)

TRUE

Flag booleano `true' (1).

FALSE

Flag booleano `false' (0).

ERR

Flag de error devuelto por rutinas en un fallo (-1).

OK

Flag de error devuelto por rutinas cuando las cosas acaban con éxito.

Uso de la Librería

Ahora describimos como utilizar realmente el paquete de pantalla. En esto, nosotros asumimos que toda la actualización, lectura, etc. es realizada por stdscr.

Estas instrucciones trabajaran en una ventana, suministrando usted los cambios en los nombres y parámetros como se ha mencionado arriba.

Aquí hay un programa ejemplo para motivar la discusión:

```
#include <curses.h>

#include <signal.h>

static void finish(int sig);

main(int argc, char *argv[])

{

/* actualice aquí sus estructuras de datos no son de curses */

(void) signal(SIGINT, finish); /* organizar interrupciones para terminar */

(void) initscr(); /* inicializar la librería curses */

keypad(stdscr, TRUE); /* permitir el mapeo de teclado */

(void) nonl(); /* decir a curses no hacer NL->CR/NL a la salida */

(void) cbreak(); /* coger los caracteres de entrada uno cada vez, no esperar por ellos
\n */

(void) noecho(); /* no hacer el eco de entrada */

if (has_colors())

{

start_color();

/*

* Asignación de colores simples, todos los necesarios normalmente.

*/

init_pair(COLOR_BLACK, COLOR_BLACK, COLOR_BLACK);

init_pair(COLOR_GREEN, COLOR_GREEN, COLOR_BLACK);

init_pair(COLOR_RED, COLOR_RED, COLOR_BLACK);

init_pair(COLOR_CYAN, COLOR_CYAN, COLOR_BLACK);
```

```
init_pair(COLOR_WHITE, COLOR_WHITE, COLOR_BLACK);

init_pair(COLOR_MAGENTA, COLOR_MAGENTA, COLOR_BLACK);

init_pair(COLOR_BLUE, COLOR_BLUE, COLOR_BLACK);

init_pair(COLOR_YELLOW, COLOR_YELLOW, COLOR_BLACK);

}

for (;;)

{

int c = getch(); /* refresco, aceptar una tecla pulsada en la entrada */

/* procesar el comando y la tecla pulsada */

}

finish(0); /* Ya lo hemos realizado */

}

static void finish(int sig)

{

endwin();

/* hacer el wrapup no-curses aqui */

exit(0);

}
```

Comenzar

Para utilizar el paquete de pantalla, las rutinas deben conocer las características del terminal, y el espacio para curscr y stdscr debe estar asignado. La función initscr() realiza ambas cosas. Puesto que debe asignar espacio para las ventanas, puede provocar un desbordamiento cuando intente hacerlo. Esto ocurre en raras ocasiones, initscr() podrá terminar el programa con un mensaje de error. Initscr() debe siempre ser referenciada antes de las rutinas que afectan a las ventanas que son utilizadas. Si esto no es así, el programa podrá hacer un volcado del núcleo tan pronto como curscr o stdscr sean referenciadas. Sin embargo, es normalmente mejor esperar que se la llame cuando usted este seguro de que la necesitará, así como chequear para comprobar errores en la inicialización. Las rutinas de cambios de estado como nl() y cbreak() deben ser referenciadas después de iniscr().

Una vez que las ventanas de pantalla han sido situadas en memora, usted puede establecerlas en su programa. Si quiere permitir a una pantalla hace el barrido, use scrollok(). Si quiere que el cursor este en la izquierda después del ultimo cambio, utilice leaveok(). Si esto no se realiza, refresh() moverá el cursor a la coordenada

actual (y,x) de la pantalla después de actualizarla.

Usted puede crear nuevas de ventanas por si mismo utilizando las funciones newwin(), derwin(), y subwin(). La rutina delwin() le permitirá deshacerse de ventanas antiguas. Todas estas opciones descritas anteriormente pueden ser aplicadas a cualquier ventana.

Salida (Output)

Ahora que las cosas han sido establecidas, querrá en realidad actualizar el terminal. Las funciones básicas utilizadas para cambiar lo que ira en una ventana son addch() y move(). addch() añade un carácter a las coordenadas (y,x) actuales. move() cambia las coordenadas (y,x) actuales a donde quiera usted que estén. Devuelve un ERR si intenta mover fuera de la ventana. Como se menciona arriba, puede combinar las dos en mvaddch() para hacer ambas cosas a la vez.

Las otras funciones de salida, como addstr() y printw(), todas llaman a addch() para añadir caracteres a la ventana.

Después de que haya puesto en la ventana lo que usted quería allí, cuando quiera que la parte del terminal cubierta por la ventana tenga ese aspecto, debe llamar a refresh(). Para optimizar el hecho de que se encuentren los cambios, refresh() asume que parte de la ventana no cambiada desde el ultimo refresh() de la ventana no ha sido actualizado en el terminal, por ejemplo, que no ha refrescado la parte del terminal con la ventana solapada. Si no es este caso, la rutina touchwin() se suministra para hacer que parezca que la ventana entera ha sido cambiada, de este modo realiza refresh() chequeando la subsección entera del terminal para los cambios.

Si llama a wrefresh() con curscr como argumento, hará que la pantalla parezca como curscr piensa que debe parecer. Esto es útil para implementar un comando que redibuje la pantalla en caso de desorden.

Entrada (Input)

La función complementaria de addch() es getch() que, si el eco esta activo, llamara a addch() para sacar el carácter por pantalla. Por ello el terminal necesita saber que hay en el terminal a todas horas, si los caracteres están siendo sacados por pantalla, el tty debe estar en modo crudo o cocinado (cbreak). Por ello inicialmente el terminal ha activado el eco y esta en modo cocinado, uno o el otro ha sido cambiado antes de llamar a getch(); de otro modo, la salida del programa será impredecible

Cuando necesita aceptar una línea orientada a la entrada en una ventana, las funciones wgetstr() y semejantes están disponibles. Hay incluso una función wscanw() que puede hacer scanf()(3)- estilo de análisis multi-campo en la entrada de la ventana. Estas funciones pseudo-lineas-orientadas activan el eco mientras se ejecutan.

El código ejemplo anterior utiliza la llamada a keypad(stdscr, TRUE) para permitir el soporte de mapeado de teclas-funciones. Con esta característica, el código getch() ve el flujo de entrada como secuencias de caracteres que corresponden a flechas y teclas de funciones. Estas secuencias son devueltas como valores pseudo-carácter. Los valores de #define devueltos son escuchados en la curses.h. El mapeo de secuencias

a valores #define es determinado mediante capacidades de teclas en la correspondiente entrada del terminal de información.

Uso de caracteres de Formularios

La función addch() (y algunas otras, incluyendo box() y border()) pueden aceptar algunos argumentos pseudo-caracteres que son definidos especialmente por ncurses. Estos son los valores #define establecidos en el encabezado de curses.h; mire allí la lista completa (busque el prefijo ACS_).

Las más útil de las definiciones ACS son los caracteres de dibujo de formularios. Puede utilizar estos para dibujar cajas y gráficos simples en la pantalla. Si el terminal no tiene estos caracteres, curses.h los convertirá a un reconocible (aunque feo) conjunto de caracteres ASCII.

Atributos de caracteres y color

El paquete ncurses soporta los atributos de pantalla incluyendo salida normal, video-inverso, subrayado(underline), y parpadeo (blink). Además soporta color, que es tratado como otra forma de atributo de pantalla.

Los atributos de pantalla están codificados, internamente, como bits altos de los tipos de pseudo-caracteres(chtype) que curses.h utiliza para representar el contenido de una celda de la pantalla. Vea el encabezado del fichero curses.h para completar la lista de valores de mascara de atributos de pantalla (busque el prefijo A_).

Hay dos modos de hacer los atributos de pantalla. Uno es el lógico- o poner el valor del atributo de pantalla que desee en el argumento del carácter de una llamada addch(), o otra llamada de salida, que tome un argumento chtype. El otro es poner el valor actual del atributo. Esto esta lógicamente relacionado con el atributo de pantalla que especifique en el primer formulario. Usted hace esto con las funciones attron(), attroff(), y attrset(); vea las páginas del manual para mas detalles. El color es una clase especial de luminosidad.

El paquete realmente piensa en términos de pares de colores, combinaciones de colores de primer y segundo plano. El modo sample establece ocho pares de colores, todos con garantía disponible en oscuro. Nótese que cada par de colores es , en efecto, llamado con el nombre de su color de primer plano. Cualquier otro rango de ocho valores no conflictivos podrían haber sido utilizados como primeros argumentos de los valores de init_pair().

Una vez que haya hecho un init:pair() que cree N pares de colores, puede utilizar COLOR_PAIR(N) como un atributo de pantalla que invoca a una particular combinación de color. Nótese que COLOR_PAIR(N), para la constante N, es por si mismo una constante en tiempo de compilación y puede ser utilizada en la inicialización.

Interfaz de Ratón

La librería curses además suministra una interfaz de ratón. Nota: esta capacidad es original de ncurses, no es parte de XSI Curses estándar, ni de la versión 4 de System v, ni de curses BSD. Por ello, recomendamos que envuelva el código relacionado con el ratón en una #ifdef utilizando la macro característica

NCURSES_MOUSE_VERSION de esta forma no será compilado ni enlazado en sistemas no-ncurses.

Actualmente, la recogida de eventos del ratón solo trabaja bajo xterm. En el futuro, ncurses detectara la presencia de gpm(), una versión gratuita de Alessandro Rubini que es un servidor de ratón para sistemas Linux, y acepta la capturación de eventos a través de él.

La interfaz de ratón es muy simple. Para activarla, puede utilizar la función mousemask(), pasando como primer argumento el bit de mascara que especifica el tipo de evento que usted quiere que su programa reconozca. Devolverá la máscara de bits de los eventos que realmente se convertirán en visibles, los cuales difieren del argumento si el dispositivo del ratón no es capaz de reconocer algunos de los tipos de eventos que usted ha especificado.

Una vez que el ratón esta activo, el comando de su bucle de aplicación debe observar el valor devuelto de KEY-MOUSE a través de wgetch(). Cuando vea esto el evento de ratón recogido ha sido introducido en la cola. Para sacarlo de ella, utilice la función getmouse() (debe hacer esto antes de la próxima wgetch(), de otro modo otro evento de ratón puede venir y hacer el primero inaccesible.

Cada llamada a getmouse() rellena una estructura (la dirección que le pasara) con el dato del evento del ratón. El dato del evento incluye zero-origin , las coordenadas de la celda del carácter de la pantalla relativa del puntero de ratón. Además incluye una mascara de evento. Los bits en esta mascara se les dará un valor, correspondiendo al tipo de evento que haya sido reconocido.

La estructura de ratón contiene dos campos adicionales que podrían ser significativo en el futuro como interfaces ncurses hacia nuevas formularios de dispositivo puntero. Además de las coordenadas x e y , hay un hueco para una coordenada z; esto seria útil con las pantalla táctiles que devuelven un parámetro de presión o duración . Hay además un campo de dispositivo ID, que podría ser utilizado para distinguir entre diferentes punteros de dispositivos.

La clase de eventos visibles puede ser cambiada en cualquier momento a través de mousemask(). Los eventos pueden ser reconocidos incluyendo pulsación, liberación, simple, doble y triple clic (puede establecerse el máximo de botón pulsado para los clics). Si no permite los clics, serán reconocidos como pares de pulsado- liberación. En algunos entornos, la mascara de eventos incluye bits de reconocimiento del estado de ?shift?, ?alt? y teclas de control de teclado durante los eventos.

También se incluye una función para chequear si ha caído un evento de ratón dentro de ventanas dadas. Puede utilizarla para ver si una ventada dada puede considerar un evento de ratón relevante a ella.

Debido a que el reconocimiento de eventos de ratón no estará disponible en todos los entornos, seria poco aconsejable construir aplicaciones ncurses que requieran el uso de ratón. Mas bien, debería utilizar el ratón como una alternativa para comandos punto y disparo que su aplicación normalmente aceptaría por teclado. Dos de los juegos test en las distribuciones ncurses (bs y knight) contiene código que ilustra como puede realizarse eso.

Lea la pagina de manual para curs_mouse(3X) para mas detalles de ls funciones de la interfaz de ratón.

Finalización

Para limpiar después de las rutinas ncurses, se suministra la rutina endwin(). Esta restaura los modos de tty a como estuvieran cuando initscr() fue llamada por primera vez, y mueve el cursor abajo a la esquina izquierda. Por esto en cualquier momento después de la llamada a initscr, endwin() debería ser referenciada antes de la salida.

Descripción de Funciones

Describiremos a continuación con detalle el comportamiento de varia funciones de curses importantes, como complemento de las paginas de descripción del manual.

Inicialización y Wrapup

Initscr(): La primera función referenciada debería ser siempre initscr(). Esta determinara el tipo de terminal e inicializa las estructuras de datos de curses. initscr () además decide que la primera llamada a refresh() limpiará la pantalla. Si se produce un mensaje de error se escribe en el error estándar y el programa termina su ejecución. Por otra parte devuelve un puntero a stdscr. Unas pocas funciones deben ser llamadas antes de initscr (slk_init(), filter(), ripofflines(), use_env(), y si esta utilizando múltiples terminales , newterm()).

Endwin(): Su programa debe siempre referenciar a endwin() antes de finalizar o salir del programa. Esta funcion restaurara los modos tty, movera el cursor al al esquina de abajo izquierda de la pantalla, inicializa el terminal a modo no visual. Referenciando a refresh() o doupdate() despues de un escape temporal del programa restaurara la pantalla ncurses que habia antes del escape.

Newterm(type,ofp, ifp) : Un programa que sale a mas de un terminal utiliza newterm() en vez de initscr(). newterm() debe ser referenciado una vez por cada terminal: Devuelve una variable de tipo SCREEN* que debe ser salvada como la referencia a un terminal. Los argumentos son el tipo del terminal (una cadena de caracteres) y los puntero a fichero (FILE) para la salida y entrada del terminal. Si el tipo es NULL entonces la variable de entorno $TERM se utiliza. Endwin() debe llamar una vez en el tiempo de los toques finales para cada terminal abierto utilizando esta función.

Set_term() : Esta función se utiliza para cambiar a un terminal diferente previamente abierto con newterm(). La pantalla de referencia para el nuevo terminal se pasa como parámetro. El terminal anterior es devuelto por la función. Todas las demás llamadas afectan solo al terminal actual.

Delscreen(sp): Es la inversa de newterm(); deslocaliza las estructuras de datos asociadas con la referencia SCREEN dada.

Realizar la salida al Terminal

Refresh() y wrefresh(win): estas funciones deben ser referenciadas para realmente poner la salida en el terminal, como otras rutinas solo manipulan estructuras de

datos. Wrefresh() copia la ventana nombrada al terminal físico de la pantalla, teniendo en cuenta que ya esta preparada para actualizaciones. refresh() hace el refresco de stdscr(). A menos que leaveok() haya sido activado el cursor físico del terminal se deja en la localización del cursor de la ventana.

Doupdate() y wnoutrefresh(win): estas dos funciones permiten múltiples actualizaciones con mas eficiencia que wrefresh. Para utilizarlas, es importante entender como trabaja curse. Además de todas las estructura de las ventanas, curses mantiene dos estructura de datos que representan el terminal de pantalla: una pantalla fisica describe que hay en realidad en la pantalla, y una pantalla virtual, que describe lo que el programador quiere que haya en pantalla. Wrefresh trabaja con la primera copia de la ventana nombrada en el terminal virtual (wnoutrefrsh()), y entonces llama a la rutina para actualizar la pantalla (doupdate()). Si el programador desea la salida de varias ventanas a la vez, una serie de llamadas a wrefresh() resultarán en llamadas alternativas a wnoutrefresh(), causando varias explosiones a la salida en la pantalla. Llamando a wnoutrefrsh() para cada ventana, entonces es posible llamar a doputate() una vez, resultando en un único estallido a la salida, con pocos caracteres transmitidos en total (esto además evita visualizar un molesto parpadeo en cada actualización).

Acceso a las Capacidades de bajo nivel

Setupterm(term,filenum,errret): esta rutina es referenciada para inicializar la descripción del terminal, sin inicializar las estructuras de pantalla de curses o cambiar los bits de modo de los driver de tty. Term es la cadena de caracteres que representa el nombre del terminal que se utiliza. Filenum es el descriptor de fichero UNIX del terminal que se usa para la salida. erret es un puntero a un entero, en el que se devuelve la indicación de un éxito o un fallo. Los valores devueltos pueden ser 1(si todo esta bien), 0 (finalización no adecuada) o -1(algún problema localizando la base de datos del terminal de información).

El valor de term puede ser un NULL, que hará que el valor de TERM sea utilizado en el entorno. El puntero errret puede también ser un NULL, que significa que el código de error no se quiere. Si errret tiene el valor por defecto, y algo va mal, setpterm() escribirá por pantalla un mensaje de error apropiado y saldrá, antes de regresar. Por esto, un programa simple puede llamar a setupterm(0,1,0) y no preocuparse de inicializar errores. Después de la llamada a setupterm(), la variable global cur_term esta establecida como puntero de la estructura actual de las capacidades del terminal. Llamando a setupterm() para cada terminal, y salvando y restaurando cur_term, es posible para un programa utilizar dos o mas terminales a la vez. Setupterm() además restaura los nombres de la sección de la descripción del terminal en la tabla ttytype() de caracteres globales. En consecuencia, llama a setupterm() que sobrescribirá esta tabla, si podrá tenerla guardada para usted si necesita que lo este.

Depuración

NOTA: Estas funciones no forman parte de las curses API estándar.

Trace(): esta función se usa para específicamente establecer el nivel de traza. Si el nivel de traza es no-cero, la ejecución de su programa generara un fichero llamado ?trace? en el directorio actual de trabajo conteniendo el documento de las acciones de la librería. Un nivel de traza mayor permite un documento mas detallado (y prolijo)

- vea los comentarios unidos a las definiciones TRACE_ en el fichero curses.h para mas detalles. (Además es posible establecer un nivel de traza asignando el nivel de traza a la variable de entorno NCURSES_TRACE).

_tracef(): esta función puede ser utilizada para realizar la salida de su propia información de depuración. Solo esta disponible si realiza el enlace con _lncurses_g. Puede ser utilizada de la misma manera que printf(),solo produce la salida a una nueva linea despues de terminar los argumentos. La salida va a un fichero llamado trace en el directorio actual.

Los registros de traza pueden ser difíciles de interpretar debido a que hay en ellos un volumen completo del volcado de memoria. Esto es un escrito llamado ?tracemunch? incluido con la distribución de ncurses que puede aliviar este problema de alguna manera; esto compacta las secuencias largas de operaciones similares en líneas simples de pseudo-operaciones. Estas pseudo-ops pueden ser distinguidas por el hecho de que se las nombra con letras mayúsculas.

Avisos, Consejos Y Trucos

Las paginas del manual son una completa referencia para esta librería. En el resto de este documento, discutidos varios métodos útiles que no son tan obvios en las descripciones de las paginas del manual.

Algunas notas de precaución

Si se encuentra a si mismo pensando si necesita utilizar noraw() o nocbreak(), piense de nuevo y actúe con cautela. Probablemente sea mejor elegir utilizar getstr() o una de sus semejantes en modo cocinado. Las funciones noraw() y nocbreak() intentan restaurar el modo cocinado, pero pueden terminar aporreando algunos bits de control establecidos antes de comenzar su aplicación. Además, pueden siempre haber estado poco documentadas, y estar casi rompiendo la utilidad de la aplicación con otras librerías curses.

Lleve en mente que refresh() es un sinónimo de wrefresh (stdscr), y no intente mezclar el uso de stdscr con el uso de ventanas declaradas con newwin(); una llamada a refresh() puede quitarlas de la pantalla. La manera correcta de manejar esto es usar subwin(), o no tocar stdscr y embaldosar su pantalla con ventanas declaradas que usted después llamara a wnoutrefresh() en alguna parte de su bucle de eventos de programa, con una llamada doupdate para hacer estallar el repintado actual.

Usted estará menos cerca de tener problemas si diseña sus trazados de pantalla para usar embaldosados mejor que ventanas solapadas. Históricamente, el soporte curses para ventanas solapadas ha sido flojo, frágil y documentado pobremente. La librería ncurses no es todavía una excepción a esta regla.

Hay una librería gratuita llamada paneles que se incluye en la distribución de ncurses que hace un bonito trabajo de dar fuerza a las características de las ventanas solapadas.

Intente evitar utilizar las variables globales LINES y COLS. Utilice getmaxyx() en el contexto de stdscr en su lugar. Razón: su código puede ser llevado para ejecutarse un

entorno con cambios de tamaño de las ventanas, en el caso de que varias ventanas puedan ser abiertas con diferentes tamaños.

Abandonar temporalmente el modo *Ncurses*

Algunas veces querrá escribir un programa que se utilice la mayor parte del tiempo en modo pantalla, pero ocasionalmente vuelva a modo cocinado. Una razón común para esto es que permita la salida del entorno (shell-out). Esta conducta es fácil de realizar con ncurses.

Para dejar el modo ncurses, llame a endwin() como si estuviera intentando terminar el programa. Esto hará que la pantalla vuelva al modo cocinado; puede hacer su salida del entorno(shell-out). Cuando quiera volver a al modo ncurses, simplemente llame a refresh() o doupdate().Esto repintara la pantalla.

Hay una funcion booleana, isendwin(), cuyo código puede utilizarse para testear si el modo ncurses de pantalla esta activo. Devuelve TRUE en el intervalo entre una llamada a endwin() y el siguiente refresh(), FALSE de otra forma.

Aquí hay un código de ejemplo para la salida del entorno:

addstr("Shelling out...");

def_prog_mode(); /* salvar los modos tty actuales */

endwin(); /* restaurar los modos tty originales */

system("sh"); /* ejecutar el entorno(shell) */

addstr("returned.\n"); /* preparando el mensaje de retorno */

refresh(); /* restaurar los modos salvados, repintar la pantalla */

Uso De *Ncurses* Bajo Xterm

Una operación de cambio de tamaño en X envía SIGWINCH a la aplicación que se ejecuta bajo xterm. La librería ncurses no captura la señal, porque no puede en general saber como quiere que la pantalla sea repintada. Tendrá que escribir el manejador de SIGWINCH usted mismo.

La forma mas fácil de codificar su manejador de SIGWINCH es haber hecho un endwin, seguida de un refresco y un repintado de pantalla que usted codifique. El refresco recogerá un nuevo tamaño de pantalla del entorno xterm.

Manipulación de Múltiples Terminales

La función initscr() en realidad llama a la función llamada newterm() para hacer la mayor parte de su trabajo. Si esta escribiendo un programa que abra múltiples terminales, utilice newterm directamente.

Por cada llamada, tendrá que especificar el tipo de terminal y un par de punteros a ficheros; cada llamada devolverá una referencia a pantalla, y stdscr establecerá en la ultima asignada. Cambiara de pantallas con la llamada a set_term. Note que también

tendrá que llamar a def_shell_mode y def_prog_mode para cada tty por si mismo.

Prueba de las capacidades del terminal

Algunas veces usted querría escribir programas que testeen la presencia de varias características andes de decidir si va a entrar en el modo ncurses. Una forma fácil de hacer esto es llamar a setupterm(), entonces utilice las funciones tigerrflag(), tigetnum(), y tigetstr() para hacer el test.

Un caso particularmente útil de esto surge normalmente cuando quiere consultar si un tipo de terminal dado debería ser tratado como ?smart (accesible mediante el cursor) o ?stupid?. El modo correcto de consultar esto es miras si el valor devuelto de tigetstr ("cup") es no nulo. Alternativamente, puede incluir el fichero term.h y testear el valor de la macro cursor_address.

Sintonización para la velocidad

Utilice la familia de funciones addchstr() para un repintado del texto mas rápido cuando sepa que el texto no contiene ningún carácter de control. Intente hacer cambios en los atributos poco frecuentes en sus pantallas. ¡No utilice la opción immedok()!

Aspectos especiales de *Ncurses*

Cuando ejecute en PC-clonicos, ncurses ha realzado el soporte para caracteres de gama medio altos de IBM (hight-half) y ROM. El modo de vídeo A_ALTCHARSET , permite visualizar ambos medio-altos gráficos ACS y los gráficos PC ROM 0-31 que normalmente se interpretan como caracteres de control.

La funcion wresize() permite cambiar el tamaño de una ventana en un lugar.

Compatibilidad con versiones anteriores

A pesar de nuestros mejores esfuerzos, hay algunas diferencias entre ncurses y la (no documentada) conducta de implementaciones anteriores de curses. Estas se presentan en la documentación de API para ambigüedades u omisiones.

Refresco de ventanas superpuestas

Si define dos ventanas A y B que se superponen, y entonces alternativamente escriben deprisa y las refrescan, los cambios hechos en la región solapada bajo versiones históricas de curses estaban normalmente documentadas sin precisión.

Para comprender por qué esto es un problema, recuerde que las actualizaciones de pantalla son calculadas entre dos representaciones de la visualización entera. La documentación dice que cuando usted refresque una ventana, primero es copiada a la pantalla virtual, y entonces los cambios se calculan para actualizar la pantalla física (y se aplican al terminal). Pero "copiado" no es muy especifico, y diferencias sutiles en como el copiado de trabajos pueden producir diferentes conductas en el caso donde dos ventanas solapadas son cada una refrescadas en intervalos impredecibles.

Lo que ocurre a las regiones de sobrelapado depende de que hace wnoutrefresh() con sus argumentos -- que porción de la ventana del argumento se copia a la pantalla

virtual. Algunas implementaciones hacen "cambios de copia", copiando solo posiciones en la pantalla que han sido cambiadas (o han sido marcadas como cambiadas con wtouch() y semejantes). Algunas implementaciones hacen "copia integra" copiando todas las posiciones de pantalla a la pantalla virtual si o si no han sido cambiadas.

La librería ncurses por sí misma no ha sido siempre consistente en su puntuación. Debido a una traba, las versiones 1.8.7 a 1.9.8 a hacían la copia integra. Las versiones 1.8.6 y anteriores, y 19.9 y más recientes, hacen la copia de cambios.

Para implementaciones más comerciales de curses, no se documenta y no se conoce con seguridad (por lo menos no para los que mantienen ncurses) si hacen la copia de cambios o la copia integra. Sabemos que la versión System V 3 curses tiene lógica en parecer que realiza la copia de cambios, pero la lógica de alrededor y las representaciones de datos son suficientemente complejas, y nuestro conocimiento lo suficientemente indirecto, que es difícil saber si esto es así. No esta claro que pretende decir la documentación de SVr4y XSI estándar. La XSI Curses estándar escasamente menciona wnourefresh(); los documentos SVr4 parecen estar describiendo la copia integra, pero es posible con algún esfuerzo y trabajo duro leerlas comprobando la otra manera.

Debería por ello ser poco sensato confiar en el comportamiento de los programas que podrían haber estado enlazados con otras implementaciones de curses. En su lugar, puede hacer un explícito touchwin() antes de la llamada a wnoutrefresh() para garantizar una copia integra de los contenidos a algún sitio.

El modo realmente limpio de manejar esto es utilizar la librería paneles. Si, cuando quiera una actualización de pantalla, usted hace update_panels(), hará todo las llamadas necesarias a wnourefrsh() para cualquier orden de pila de paneles que haya definido. Entonces puede realizar un doupdate() y habrá un simple estallido de la Entrada-Salida física que hará todas sus actualizaciones.

Antecedentes del Borrado

Si ha estado utilizando una versión muy antigua de ncuses (1.8.7 o anterior) le sorprendería su comportamiento para las funciones de borrado. En versiones anteriores, las áreas borradas de una ventana eran rellanadas con un modificado en blanco de los atributos actuales (como habían sido establecidos por wattrset(), wattron(), wattroff() y semejantes).

En versiones nuevas, esto no es así, En cambio, los atributos de los blancos borrados son normales a menos y hasta que se modifiquen con las funciones bkgset() o wbkdset().

Este cambio de actuación ajusta ncurses al System V versión 4 y a la XSI Curses estándar.

Ajuste con XSI Curses

La librería ncurses pretende ser un ajuste de nivel base con la XSI Curses estándar de X/Open. Muchas características de nivel extendido (de hecho, casi todas las características que no conciernen directamente con caracteres anchos e

internacionalización) son también soportados.

Un efecto del ajuste XSI es el cambio en la conducta descrita en Antecedentes_de_Borrado-Compatibilidad_con_versiones_anteriores.

También, ncurses reúne los requerimientos XSI que cada puntero de entrada de macro tiene una función correspondiente y seria enlazada (y será un prototipo chequeado) si la definición de la macro esta desactivada con #undef.

Librería Paneles

La librería ncurses provee por si misma un buen soporte de visualizaciones de pantalla en el que las ventanas son embaldosadas (no solapadas). En casos mas generales en los que las ventanas se solaparían, tiene que utilizar series de llamadas wnoutrefresh() seguidas de doupdate(), y ser cuidadoso acerca del orden en que hace los refrescos de ventana. Tiene que ser de abajo-arriba,, de otra manera partes de las ventanas que serian oscurecidas se mostraran a través.

Cuando su diseño de interfaz es tal que las ventanas se zambullen profundamente en la pila de visibilidad o surgen a lo mas alto en tiempo de ejecución, el resultado reserva-mantenimiento puede ser tedioso y difícil de conseguir bien. De aquí la librería paneles.

La librería paneles aparece primero en AT&t system V. La versión documentada aquí es el código gratuito distribuido con ncurses.

Compilación con la Librería Paneles

Sus módulos que utilicen paneles deben importar las definiciones de la librería paneles con

#include <panel.h>

y deben ser enlazadas explícitamente con la librería paneles utilizando un argumento -lpanel. Note que deben también enlazar la librería ncurses con -lncurses. Muchos compiladores son de dos pasos y aceptaran cualquier orden, pero es una buena practica poner -lpanel primero y -lncurses en segundo lugar.

Descripción de Paneles

Un objeto panel es una ventana que es implícitamente tratada como una parte de una superficie que incluye todos los demás objetos panel. La superficie tiene un orden implicito de abajo-arriba de visibilidad. La librería paneles incluye una funcion de actualizacion (analoga a refrsh()) que visualiza todos los paneles en la superficie en el orden apropiado para resolver superposiciones. La ventana estándar, stdscr, es considerada por debajo de todos los paneles.

Detalles de las funciones de paneles están disponibles en las paginas del manual. Nosotros nos acercaremos a los atributos de pantalla.

Usted crea un panel de una ventana llamando a new_panel() con un puntero de ventana. Este entonces llegara a la parte superior de la superficie. La ventana del

panel estará disponible como el valor de la llamada a panel_window() con el puntero del panel como argumento.

Puede borrar el panel (quitarlo de la superficie) con del_panel. Esto no descolocara la ventana asociada; tiene que hacer esto por sí mismo. Puede reemplazar una ventana de panel con una ventana diferente llamando a replace_window. La nueva ventana seria de diferente tamaño; el código del panel recomputerizara todas las superposiciones. Esta operación no cambia la posición del panel en la superficie.

Dos funciones (top_panel(), bottom_panel()) se suministra para reestructurar la superficie. La primera pone su ventana argumento en lo mas alto de la superficie; la segunda la envía a la parte baja. Ambas operaciones dejan la localización de pantalla, contenido y tamaño sin cambiar.

La función update_panels() hace todas las llamadas a wnoutrefresh necesarias para preparar doupdate()(que debe llamar usted mismo, después).

Normalmente, querrá llamar a update_panels() y doupdate() justo antes de aceptar un comando a la entrada, una vez en cada dicho de iteración con el usuario. Si llama a update_panels() después de cada y todas las escrituras en panel, generara muchos refrescos innecesarios y parpadeos de pantalla.

Paneles, Salida, y la Pantalla Estándar

No debe mezclar las operaciones wnout refrsh() o wrefresh() con el código de paneles; funcionara solo si la ventana argumento esta en el panel mas alto o no oscurecido por otros paneles.

La ventana stdscr es un caso especial. Se considera por debajo de todos los paneles. Debido a los cambios a paneles puede oscurecer parte de stdscr, aunque, podría llama a update_panels() antes de doupdate() incluso cuando solo quiera cambiar stdscr.

Note que wgetch llama automáticamente a wrefresh. Por ello, contestando a la entrada de la ventana de panel, necesita estar seguro de que el panel es totalmente no oscurecido.

No hay en el presente una forma de visualizar los cambios de un panel oscurecido sin repintar todos los paneles.

Escondiendo Paneles

Es posible quitar un panel de la superficie temporalmente; utilice hide_panel para esto. Utilice show_panel() para hacerlo visible de nuevo. La función predicado panel_hidden testea si un panel esta oculto o no.

El código panel_update ignora los paneles ocultos. No puede hacer top_panel() o bottom_panel() con un panel oculto. Otras operaciones de paneles se pueden aplicar.

Otras Características Diversas

Es posible guiar la superficie utilizando las punciones panel_above() y panel_below. Dado un puntero de panel, devuelven un panel por encima o por debajo de este panel.

Dado NULL, devuelven el panel mas abajo y el panel de mas arriba.

Cada panel tiene un puntero de uso asociado, no utilizado por el código del panel, al que puede adjuntar un dato de aplicación. Ver la pagina de documentación de set_panel_userptr() y panel _userptr para mas detalles.

Librería Menú

Un menú es una pantalla de visualización que asiste al usuario para elegir alguno de los subconjunto de un conjunto dado de cosas. La librería menú es una extensión de curses que soporta programación fácil de las jerarquías de menús con una uniforme pEro flexible interfaz.

La librería menú aparece por primera vez en AT&T system V. La versión documentada aquí es un código distribuido gratuitamente con ncurses.

Compilación con la Librería Menú

Sus módulos que utilicen menú deben importar las declaraciones de la librería menú con

#include <menu.h>

y debe enlazarse explícitamente con la librería menú utilizando el argumento -lmenu. Nótese que deben también enlazar la librería ncurses con -lncurses. Muchos compiladores son de dos pasos y aceptaran cualquier orden, pero es todavía mejor poner -lmenu primero y -lncurses después.

Descripción de Menús

Los menús creados por esta librería consisten en una colección de objetos incluyendo una parte que es cadena de caracteres como nombre y una parte que es una cadena de caracteres como descripción. Para hacer menús, usted crea grupos de esto objetos y los conecta con la estructura de los objetos de menú.

El menú puede entonces ser asociado, esto es escrito a una ventana asociada. Realmente, cada menú tiene dos ventanas asociadas; una ventana de contenido en la que el programador puede escribir títulos o bordes, y una subventana en la que los objetos de menú propiamente dichos son visualizados. Si esta subventana es demasiado pequeña para visualizar todos los objetos, será un punto de visualización listado con la colección de objetos.

Un menú puede también ser quitado de la asociación (esto es, no visualizado), y finalmente liberado para hacer el almacenamiento asociado con el y con los objetos disponible para reutilización.

El flujo general de control de un programa menú aparece así:

Inicializar curses.

Crear los objetos de menú, utilizando new_item().

Crear el menú utilizando new_menu().

Asociar el menú utilizando menu_post()

Refresco de pantalla.

Procesar respuesta de usuario a traves de un bucle de entrada.

Quitar la asociación del menú utilizando menu_unpost().

Liberar el menú, utilizando free_menu().

Liberar los objetos utilizando free_item().

Finalizar curses.

Selección de Objetos

Los menús pueden ser multi-valor o (por defecto) valor-simple (vea la pagina del manual menu_opts(3x) para ver como cambiar el valor por defecto. Ambos tipos siempre tienen un objeto actual.

Del menú valor-simple puede leer el valor seleccionado simplemente observando el objeto actual. Del multi-valor, cogerá el actual seleccionado saltando a través de los objetos aplicando la función predicado item_value(). Su código de procesamiento de menú puede utilizar la funcion set_item_value() para poner banderas a los objetos de la selección.

Los objetos menú pueden ser hechos no seleccionables utilizando set_item_opts() o item_opts_off() con el argumento O_SELECTABLE . Esto es la única opción hasta ahora definida para menús, pero es una buena practica codificar como si otros bits de opciones estuvieran activos.

Visualización de Menú

La librería menú calcula un tamaño mínimo de visualización de su ventana, basado en las siguientes variables:

El numero y máxima longitud de los objetos menús.

Si la opcion O_ROWMAJOR esta activa.

Si la visualización de descripciones esta activa.

Cualquiera que el formato de menú pueda haber sido establecido por el programador.

La longitud de la cadena mascara de caracteres de menú utilizada para los atributos de pantalla de objetos seleccionados.

La función set_menu_format() te permite establecer el tamaño máximo del punto de visualizacion (viewport) de la pagina de menú que se utilizara para visualizar los objetos de menú. Puede recuperar cualquier formato asociado con un menú con menu_format(). El formato por defecto es lineas=16, columnas=1.

La pagina actual de menú puede ser menor que el tamaño del formato. Esto depende

del numero de objeto y el tamaño y si O_ROWMAJOR esta activo. Esta opción (por defecto) provoca que los objetos menú sean visualizados en modelo ?raste-scan?, así que si mas de un objeto se ajustara horizontalmente la primera pareja de objetos lado con lado en la línea superior. La alternativa es visualizar la mayor columna, que intenta poner primero diferentes objetos en la primera columna.

Como se menciona anteriormente, un formato de menú no suficientemente ancho permite que todos los objetos se ajusten en la pantalla resulten en un visualizado de menú que es verticalmente paginable. Puede paginarlo con peticiones al dispositivo de menú, que será descrito en la sección Procesamiento_de_la_entrada_de_Menú.

Cada menú tiene una cadena de caracteres de marcas para visualmente seguir de cerca los objetos seleccionados; vea las pagina del manual menu_mark(3x) para mas detalles. La longitud de la cadena de caracteres de marcas también influye en el tamaño de la pagina de menú.

La función scale_menu() devuelve el tamaño mínimo de visualización que el código de menú computeriza de todos estos factores. Hay otras visualizaciones de atributos de menú incluyendo un atributo de selección, un atributo para objetos seleccionables, un atributo para objetos no seleccionables y un carácter de relleno usado para separar el nombre del texto del objeto de la descripción del texto. Esto tiene valores por defecto razonables que la librería le permite cambiar (vea la pagina de manual menu_attribs(3x)).

Ventanas de Menú

Cada menú tiene, como se menciona antes, un par de ventanas asociadas. Ambas ventanas son pintadas cuando el menú es situado y borrado cuando el menú es quitado de su situación.

El exterior o la estructura de la ventana no es de otra manera tocada por otras rutinas de menú. Existe luego el programador puede asociar un titulo, un borde, o quizás el texto de ayuda con el menú y tener refresco apropiado o borrado e tiempo de asociación/ no asociación. La parte interna de la ventana o subventana esta donde la pagina de menú actual es visualizada. Por defecto, ambas ventanas son stdscr. Puede establecerlas con la función de menu_win(3x). Cuando llama a menu_post(), puede escribir el menú en su subventana. Cuando llama a menu_unpost(), puede borrar la subventana, sin embargo, ninguna de estas realmente modifica la pantalla. Para hacer esto, llame a wrefresh() o alguna equivalente.

Procesamiento de la entrada de Menú

El bucle del código del procesamiento de menú llamaría a menu_driver() repetidamente. El primer argumento de esta rutina es un puntero de menú; el segundo es un código de comando de menú. Usted debería escribir una rutina de entrada-captura que mapee los caracteres de entrada al código de los comandos de menú, y pase su salida a menu_driver(). El código de comando de menú esta completamente documentado en menu_driver(3x).

El grupo más simple de código de comando es REQ_NEXT_ITEM, EQ_PREV_ITEM, REQ_FIRST_ITEM, REQ_LAST_ITEM, REQ_UP_ITEM, REQ_DOWN_ITEM, REQ_LEFT_ITEM, REQ_RIGHT_ITEM. Estos cambia los objetos seleccionados

actualmente. Estas solicitudes pueden causar listado de la pagina de menú si solo se visualiza una parte.

Hay solicitudes explícitas de paginación que también cambia el objeto actual (porque la localización seleccionada no cambia, pero el objeto lo hace). Estos son REQ_SCR_DLINE, REQ_SCR_ULINE, REQ_SCR_DPAGE, y REQ_SCR_UPAGE.

La REQ_TOGGLE_ITEM selecciona o quita la selección del objeto actual. ES utilizada en menus multi-valor; si la utiliza con O_ONEVALUE , cogera un error devuelto (E_REQUEST_DENIED).

Cada menú tiene un buffer patrón asociado. La lógica de menu_driver() intenta acumular caracteres ASCII imprimibles pasado al buffer; cuando concuerda un prefijo con el nombre de un objeto, este objeto (o el próximo que concuerde) es seleccionado. Si añadiendo un carácter producido no hay nueva concordancia, ese carácter es borrado del buffer patrón, y menu_driver() devuelve E_NO_MATCH.

Algunas solicitudes cambian el buffer patrón directamente: REQ_CLEAR_PATTERN, REQ_BACK_PATTERN, REQ_NEXT_MATCH, REQ_PREV_MATCH. La ultimas dos se utilizan cuando la entrada del buffer patrón concuerda con mas de un objeto en un menú multi_valor.

Cada paginación con éxito o solicitud de navegación de objeto limpia el buffer patrón. Además es posible establecer el buffer patrón explícitamente con set_menu_pattern().

Finalmente, las solicitudes de dispositivo de menú de la constante MAX_COMMAND se consideran comandos de aplicación especifica. El código de menu_driver() las ignoran y devuelve E_UNKNOWN_COMMAND.

Otros Aspectos Diversos

Varias opciones de menú pueden afectar el procesamiento y la aparición visual y entrada de procesamiento de menús. Ver menu_opts(3x) para mas detalles.

Es posible cambiar el objeto actual del código de aplicación; esto es útil si quiere escribir sus propias solicitudes de navegación. Es además posible explícitamente establecer la líneas superior del visualizado de menú. Ver mitem_current(3x). Si su aplicación necesita cambiar el cursor subventana de menú por alguna razón, pos_menu_cursor() restaurara a la localización actual para continuar con el procesamiento del dispositivo de menú.

Es posible establecer enlaces para ser llamados en la inicialización del menú en tiempo de los últimos toques finales, y cuando se de que los objetos seleccionados cambien. Ver menu_hook(3x).

Cada objeto, en cada menú, tiene asociado un puntero de uso en el que puede manejar la aplicación de datos. Ver mitem_userptr(3x) y menu_userptr(3x).

Librería Formulario(Forms)

La librería formulario (form) es una extensión de curses que soporta programación fácil en formularios de pantalla para entrada de texto y control de programa.

La librería formulario apareció por primera vez en AT&T System V. La versión documentada aquí es un código gratuito distribuido con ncurses.

Compilación con la librería Formularios (Forms)

Sus módulos que utilicen formularios deben importar las declaraciones de la librería con

#include <form.h>

y debe ser enlazado explícitamente con la librería formularios utilizando el argumento -lform. Note que debe también enlazarse con la librería ncurses con -lncurses. Muchos compiladores son de dos pasos y aceptaran cualquier orden, pero todavía es una buena practica poner -lform primero y -lncurses después.

Descripción de Formularios

Un formulario es una colección de campos; cada campo puede ser una etiqueta(texto aclaratorio) o una localización de datos de entrada. Formularios Largos pueden ser segmentados en paginas; cada entrada a una nueva pagina limpia la pantalla.

Para hacer formularios, crea grupos de campos y los conecta con objetos estructura de formularios; la librería formulario realiza hace esto relativamente sencillo.

Una vez definido, un formulario puede ser asociado, es decir escrito en una ventana asociada. En realidad, cada formulario tiene dos ventanas asociadas; una ventana de contenido en la que el programador puede escribir títulos y bordes, y una subventana en la que los campos de formularios apropiados son visualizados.

Como el formulario de usuario rellena el formulario asociado, las teclas de navegación y edición permiten movimiento entre campos, las teclas de edición permiten modificación de los campos, y el texto sencillo añadiduras o cambios de datos en el campo actual. La librería form le permite (el diseñador de formularios) ligar cada tecla de navegación y de edición a una tecla pulsada aceptada por los campos de curses que pueden tener condiciones de validación en ellas, así que chequean la entrada de datos por tipo y valor. La librería formulario suministra un rico conjunto de campos de tipos predefinidos, y hace relativamente fácil definir nuevos.

Una vez que su transacción es completada (o abortada), un formulario puede ser quitada de su asociación (esto es, ya no esta visualizado), y finalmente liberado hace que el almacenamiento asociado con el y sus objetos disponibles puedan reulilizarse.

El flujo general de control de un programa con formularios aparece asi:

1. Inicializar curses.

2. Crear los campos de formularios, utilizando new_field().

3. Crear el formulario utilizando new_form().

4. Asociar el formulario utilizando form_post().

5. Refrescar la pantalla.

6. Procesar las solicitudes del usuario a través de un bucle de entrada.

7. Quitar la asociación del formulario utilizando form_unpost().

8. Liberar el formulario, utilizando free_form().

9. Liberar los campos utilizando free_field().

10. Terminar curses.

Note que esto se parece mucho a un programa menú; la librería formulario maneja tarea que son en muchos sentidos semejantes, y su interfaz fue obviamente diseñada para parecerse a la Librería_Menú cada vez que sea posible.

En programas con formularios, sin embargo, el ?proceso de solicitudes de usuarios? es de alguna manera mas complicado que para menús. Además de las operaciones de navegación como menú, el bucle de dispositivo de menú tiene que soportar edición de campos y validación de datos.

Crear y Liberar Campos y Formularios

La función básica para crear campos es new_field():

FIELD *new_field(int height, int width, /* nuevo tamaño de campo */

int top, int left, /* esquina superior izquierda */

int offscreen, /* numero de líneas de pantalla */

int nbuf); /* numero de buffers de trabajo */

Los objetos menú ocupan siempre una línea simple, pero los campos de formulario pueden tener múltiples líneas. Por esto new_field() requiere que usted especifique la profundidad y altura (los dos primeros argumento, que ambos deben ser mayores que cero).

Debe especificar la localizaron del campo de la esquina superior izquierda en la pantalla (el tercer y cuarto argumento, que debe ser cero o mayor). Note que estas coordenadas son relativas a la subventana de la formulario, que coincidirá con stdscr por defecto pero no necesita ser stdscr si ha hecho una llamada explícita a set_form_window().

El quinto argumento permite especificar el numero de líneas fuera de pantalla. Si este es cero, el campo entero será siempre visible. Si es no cero, el formulario se podrá paginar, con solo una pantalla entera (inicialmente la parte de arriba) visible y a un tiempo dado. Si hace el campo dinámico y crece tanto que no se ajuste a la pantalla, el formulario se podrá paginar incluso si el argumento fuera de pantalla (offscreen) era inicialmente cero.

La librería formularios localiza un buffer de trabajo por campo; el tamaño de cada buffer es ((altura+offscreen)*anchura +1, un carácter para cada posición en el campo mas un carácter de fin NUL. El sexto argumento es un numero adicional de buffers de datos para localizar por el campo; su aplicación puede utilizarlos para sus propios

propósitos.

FIELD *dup_field(FIELD *field, /* campo a copiar */

int top, int left); /* localización de la nueva copia */

La función dup_field() duplica un campo existente en una nueva localización. El tamaño y la informularioción de los buffers son copiadas; algunos banderas de atributos y bits de estados no son (ver la función form_fiel_new(3x) para mas detalles).

FIELD *link_field(FIELD *field, /* campo a copiar */

int top, int left); /* localización del nuevo campo */

La funcion link_field() también duplica un campo existente en una nueva localización. La diferencia con dup_field() es que dispone para que los nuevos campos de buffer sean compartidos con el antiguo.

Aparte del uso obvio de hacer campos editables de dos diferentes paginas de formularios, enlazando campos dando un formulario de cortar en etiquetas dinámicas. Si declara varios campos ligados a un original, y entonces los hace inactivos, los cambios desde el original serán propagados a los campos enlazados.

Como con campos duplicados, los campos enlazados tienen bits de atributos separado del original.

Como puede adivinar, todas estas localizaciones de campos devuelven NULL si la localización del campo no es posible debido a un error de memoria insuficiente o argumentos fuera del limite.

Para conectar los campos de un formulario, utilice

FORM *new_form(FIELD **fields);

Esta función espera ver un valor NULL- una tabla de punteros de campos. Dichos campos están conectados a una nueva localización de objeto formulario; su dirección es devuelta (o NULL si la localización falla).

Note que new_field() no copia la tabla de punteros a un almacenamiento privado; si modifica el contenido de la tabla de punteros durante el procesamiento de los formularios, muchas cosas extrañas de diversas maneras pueden ocurrir. También note que cualquier campo dado puede ser conectado a una formulario.

Las funciones free_field() y free_form están disponibles para liberar objetos campos y formularios. Es un error no intentar liberar un campo conectado a una formulario, pero no viceversa; por ello, usted generalmente liberara sus objetos formularios primero.

Cambiar Atributos de Campos (Fields)

Cada campo de formulario tiene un numero de localización y atributos de tamaño asociados con el. Hay otros atributos de campo utilizado para controlar la

visualización y edición de un campo. Algunos(por ejemplo, el bit O_STATIC) envuelve suficientes complicaciones para ser cubierto en secciones propias posteriormente. Nosotros cubrimos las funciones utilizados para coger y establecer diferentes atributos básicos aquí.

Cuando un campo se crea, los atributos no especificados por la función new_field se copian desde un campo de sistema invisible. En el ajuste de atributos y cambios de funciones, el argumento NULL se toma para referirse a este campo. Los cambio persiste como valores por defectos hasta que sus aplicaciones de formularios terminan.

Cambio de tamaño y localización de Datos

Puede recuperar los tamaños de los campos y localizaciones a través de

int field_info(FIELD *field, /* campo desde el que se cambia */

int *height, *int width, /* tamaño del campo */

int *top, int *left, /* esquina superior izquierda */

int *offscreen, /* numero de líneas fuera de pantalla */

int *nbuf); /* numero de buffer de trabajo */

Esta función es un tipo de inversa de new_fiel(); en vez de establecer los atributos de tamaño y localización de un nuevo campo, se cambian desde uno existente.

Cambiar la localización de un Campo (Field)

Si es posible mover una localización de campo en pantalla:

int move_field(FIELD *field, /* campo a alterar */

int top, int left); /* nueva esquina superior izquierda */

Puede, por supuesto, preguntar la localización actual a través de fiel_info().

El Atributo de Justificación

Los campos de una línea puede estar no-justificado, justificados a la derecha, justificados a la izquierda, o centrados. Aquí decimos como manipular este atributo:

int set_field_just(FIELD *field, /* campo a modificar */

int justmode); /* modo para establecer */

int field_just(FIELD *field); /* cambiar el modo de campo */

Los valores de modo aceptados y devueltos por estas funciones son macros preprocesadas are preprocessor macros NO_JUSTIFICATION, JUSTIFY_RIGHT,

JUSTIFY_LEFT, o JUSTIFY_CENTER.

Visualización de atributos de Campo (Field)

Para cada campo, puede establecer el atributo de primer plano para caracteres introducidos, un atributo de segundo plano para el campo entere, y un carácter comodín para la porción de campo no rellenada. Puede también controlar paginación del formulario.

Este grupo de cuatro atributos de campo controlan la apariencia visual de un campo en la pantalla, sin afectar de ninguna forma al dato en el buffer de campo.

int set_field_fore(FIELD *field, /* campo a modificar */

chtype attr); /* atributo a establecer */

chtype field_fore(FIELD *field); /* campo a consultar */

int set_field_back(FIELD *field, /* campo a modificar */

chtype attr); /* atributo a establecer */

chtype field_back(FIELD *field); /* campo a consultar */

int set_field_pad(FIELD *field, /* campo a alterar */

int pad); /* carácter comodín a establecer */

chtype field_pad(FIELD *field);

int set_new_page(FIELD *field, /* campo a modificar */

int flag); /* TRUE para forzar una nueva pagina */

chtype new_page(FIELD *field); /* campo en cuestión */

Los atributos a establecer y devueltos por las primeras cuatro funciones son valores de atributos de visualización normales de curses(3x). (A_STANDOUT, A_BOL, A_REVERSE, etc). El bit de pagina de un campo controla si esta siendo visualizado al comenzar de un nuevo formulario.

Bits de Opciones de Campo (Field)

Hay también una gran colección de bits de opciones de campo que puede establecer el control de varios aspectos de procesamiento de formularios. Puede manipularlos con esta funciones:

int set_field_opts(FIELD *field, /* campo a modificar */

int attr); /* atributo a establecer */

int field_opts_on(FIELD *field, /* campo a modificar */

int attr); /* atributos a activar */

int field_opts_off(FIELD *field, /* campo a modificar */

int attr); /* atributos a desactivar */

int field_opts(FIELD *field); /* campo a consultar */

Por defecto todos estas opciones están activas. Aquí presentamos los bits de opciones disponibles:

O_VISIBLE

Controla si el campo es visible en pantalla. Puede utilizarse durante el procesamiento del formulario para ocultar o mostrar los campos.

O_ACTIVE

Controla si el campo esta activo durante el procesamiento del formulario (por ejemplo visitado por teclas de navegación de formularios). Puede utilizarse para hacer etiquetas o campos derivados con valores alterables de buffer por las aplicaciones de formularios, no el usuario

O_PUBLIC

Controla si el dato es visualizado durante la entrada de campo. Si esta acción esta desactivada en un campo, la librería aceptara y editara datos en este campo, pero no será visualizado y el cursor visible del campo no se moverá. Puede desactivar el bit O_PUBLIC para definir campos con contraseña.

O_EDIT

Controla si el dato del campo puede ser modificado. Cuando esta opción es desactivada todas las respuestas editables excepto REQ_PREV_CHOICE y REQ_NEXT_CHOICE fallaran. Tales campos de solo lectura pueden ser útiles para mensajes de ayuda.

O_WRAP

Controla el limpiado de palabra en campo multi-linea. Normalmente, cuando algún carácter de una (separado por espacio) palabra alcanza el final de la línea actual, la palabra entera será llevada a la siguiente línea (asumiendo que hay una). Cuando esta opción esta desactivada, la palabra será cortada por la mitas del fin de línea.

O_BLANK

Controla el espaciado del campo. Cuando esta opción esta activa, introduciendo un carácter en la primera posición del campo borra el campo entero (excepto el carácter recién introducido).

O_AUTOSKIP

Controla el omitir automáticamente el próximo campo cuando este está completo. Normalmente, cuando el formulario de usuario intentan escribir mas datos en un campo que se ajustaran, la localización de editado salta al siguiente campo. Cuando esta opción esta desactivada, el cursor de usuario manejara el final del campo. Esta opción se ignora en campos dinámicos que no han alcanzado su tamaño limite.

O_NULLOK

Controla si la Validación_de_Campo es aplicada a campos en blanco. Normalmente, no lo es; el usuario puede dejar un campo en blanco sin invocar al normal chequeo de validación a la salida. Si esta opción esta desactivada en un campo, salir de él invocará al chequeo de validación.

O_PASSOK

Controla si la validación ocurre en cada salida, o solo después de que un campo es modificado. Normalmente lo ultimo es verdad. Estableciendo O_PASSOK puede n ser útiles si su función de validación de campo puede cambia r durante el procesamiento de formularios.

O_STATIC

Controla si el campo esta ajustado a sus dimensiones iniciales. Si lo desactiva, el campo será dinámico y se extenderá para ajustar el dato introducido.

Una opción de campo no puede ser modificada mientras el campo esta seleccionado actualmente. Sin embargo, las opciones pueden no ser modificadas en campos situados que no son los actuales.

Los valores de las opciones son mascaras de bits y pueden ser compuesta con lógicas-o de manera obvia.

Estados de Campo (Field)

Cada campo tiene una bandera de estado, que se establece a FALSE cuando el campo es creado y TRUE cuando el valor en el buffer del campo cambia. Esta bandera puede ser consultado su valor e inicializado directamente:

int set_field_status(FIELD *field, /* campo a modificar */

int status); /* modo de inicialización */

int field_status(FIELD *field); /* modo de cambio de campo */

Inicializando esta bandera bajo el control del programa puede ser útil si utiliza el mismo formulario repetidamente, buscando campos modificados cada vez.

Llamando a field_status() en un campo no seleccionado actualmente para la entrada devolverá un valor correcto. Llamando a field_status() en un campo que es seleccionado actualmente por la entrada no tiene necesariamente que dar un valor de estado de campo correcto, debido a que el dato introducido no es necesariamente copiado al buffer cero antes de salir del chequeo de validación. Para garantizar que el valor de estado devuelto refleja la realidad, llamar a field_status() en una rutina de chequeo de validación de salida, desde la inicialización o terminación del campo o de la forma, o justo después de un requerimiento de REQ_VALIDATION haya sido procesado por el dispositivo de la forma.

Puntero de Campo (Field) para Usuario

Cada estructura de campo contiene una ranura para un puntero de carácter que no es utilizada por la librería formularios. Es necesario para ser utilizada por las aplicaciones para almacenar datos privados para el campo. Puede manipularlo con:

int set_field_userptr(FIELD *field, /* campo a modificar */

char *userptr); /* modo a establecer */

char *field_userptr(FIELD *field); /* modo de cambio para el campo */

(Correctamente, este puntero de usuario de campo debe tener un tipo (void*). El tipo (char*) es guardado por la compatibilidad con System V).

Es valido establecer el puntero de usuario con el campo por defecto (con una llamada a set_field_userptr() pasándole un puntero de campo NULL). Cuando un nuevo campo se crea, el puntero de usuario de campo por defecto es copiado para inicializar el nuevo puntero de usuario de campo.

Campos (Field) Variables de Tamaño

Normalmente, un campo es ajustado al tamaño especificado en el tiempo de creación. Si, sin embargo, desactiva el bit O_STATIC , se convertirá en dinámico y automáticamente se reajustara su tamaño para acomodar el dato como es introducido. Si el campo tiene extra buffers asociados con él, crecerán a lo largo con el buffer principal de entrada.

Un campo dinámico de una línea tendrá una altura fija pero una profundidad variable, paginación horizontal para visualizar datos dentro del área de campo como originalmente.

Validación de Campo (Field)

Por defecto, un campo aceptara cualquier dato que ajustase en su buffer de entrada. Sin embargo, es posible adjuntar un tipo de validación a un campo. Si hace esto, cualquier intento de dejar el campo mientras contenga datos que no concuerden con el tipo de validación fallara. Algunos tipos de validación también tiene un chequeo de validez de carácter para cada vez que un carácter es introducido en el campo.

Un chequeo de validación no se le llama cuando la funcion set_field_buffer() modifica el buffer de entrada, ni cuando ese buffer es cambiado a través de un campo enlazado.

La librería formulario suministra un rico conjunto de tipo s de validación predefinidos , y le da la capacidad de definir propios tipos por usted mismo. Puede examinar y cambiar los atributos de validación con las siguientes funciones:

int set_field_type(FIELD *field, /* campo a modificar */

FIELDTYPE *ftype, /* tipo asociado */

...); /* argumentos adicionales */

FIELDTYPE *field_type(FIELD *field); /* campo de aplicación */

El tipo de validación de un campo es considerado un atributo del campo. Como con otros atributos de campo, también haciendo set_field_type() con un campo NULO cambiara el sistema por defecto por la validación de nuevos campo creados.

Aquí mostramos los tipo de validación predefinidos:

TYPE_ALPHA

Este tipo de campo acepta datos alfabéticos; no espacios en blanco, no dígitos no caracteres especiales (es chequeado a tiempo de la entrada de caracteres). Esto es establecido con:

Int set_field_type(FIELD *field, /* campo a modificar */

TYPE_ALPHA, /* tipo asociado */

Int width); /* máxima profundidad de campo */

El argumento de profundidad establece una profundidad mínima de dato. Normalmente querrá establecer esto a la profundidad de campo; si es mayor que la profundidad del campo, el chequeo de validación fallara siempre. Una profundidad mínima de cero hace opcional el campo de terminación.

TYPE_ALNUM

Este campo acepta datos alfabéticos y dígitos; no espacios en blanco, no caracteres especiales (es chequeado a tiempo de la entrada de caracteres). Se establece con:

Int set_field_type(FIELD *field, /* campo a modificar */

TYPE_ALNUM, /* tipo asociado */

Int width); /* máxima profundidad de campo */

El argumento de profundidad establece una profundidad mínima de campo. Como con TYPE_ALPHA , normalmente querrá establecer esto con el campo de profundidad; si es mayor que la profundidad del campo, el chequeo de validación fallara siempre. Una profundidad mínima de cero hace opcional el campo de terminación.

TYPE_ENUM

Este tipo permite restringir los valores del campo entre un conjunto especificado de valores de cadenas de caracteres (por ejemplo, las dos letra del código postal para los estados de U.S.). Se establece con:

Int set_field_type(FIELD *field, /* campo a modificar */

TYPE_ENUM, /* tipo asociado */

Char **valuelist; /* lista de valores posibles */

Int checkcase; /* ¿case-sensitive? */

Int checkunique); /* ¿debe especificarse unívocamente? */

El parámetro de la lista de valores debe ser un puntero a NULL- lista terminada de cadenas de caracteres validos. El argumento checkcase, si es verdadero, hace la comparación con la cadena de caracteres case-sensitive.

Cuando para el usuario existe un campo TYPE_ENUM, el proceso de validación intenta completar el dato en el buffer a una entrada valida. Si una cadena de elección completa ha sido introducida, es por supuesto valida. Pero es posible también introducir un prefijo de una cadena valida y haberla completado para usted.

Por defecto, si introduce un prefijo así y une mas de un valor en la lista de cadenas, el prefijo será completado por el primer valor que concuerde. Pero el argumento checkunique, si es verdadero, requiere que el prefijo concuerde para ser único para que sea validado.

Las solicitudes de entrada REQ_NEXT_CHOCE y REQ_PREV_CHOICE pueden ser particularmente útiles con estos campos.

TYPE_INTEGER

Este tipo de campo acepta un entero. Se establece como se explica a continuación:

Int set_field_type(FIELD *field, /* campo a modificar */

TYPE_INTEGER, /* tipo asociado */

Int padding; /* lugares para ceros de relleno */

Int vmin, int vmax); /* rango valido */

Los caracteres validos consiste en un encabezado signo menos o dígitos opcionales. El rango de chequeo se realiza en la salida. Si el rengo máximo es menor o igual que el mínimo, el rango se ignora.

Si el valor sobrepasa el rango de chequeo, se rellena con tantos dígitos cero de rastreo como sea necesario hasta encontrar el argumento de relleno.

Un valor de buffer TYPE_INTEGER puede convenientemente ser interpretado con la función atoi(3) de la librería C.

TYPE_NUMERIC

Este tipo de campo acepta un entero. Se establece como se explica a continuación:

Int set_field_type(FIELD *field, /* campo a modificar */

TYPE_NUMERIC, /* tipo asociado */

Int padding; /* lugares para ceros de relleno */

double vmin, double vmax); /* rango valido */

Caracteres validos consisten en un signo menos de encabezado y dígitos opcionales, posiblemente incluyendo un punto decimal. Si su sistema soporta puntos a caracteres decimales locales utilizados debe ser el definido por su local. El rango de chequeo es

realizado a la salida. Si el rango máximo es menor o igual que el mínimo, el rango se ignoran.

Si el valor sobrepasa el rango de chequeo, se rellena con tantos dígitos cero de rastreo como sea necesario hasta encontrar el argumento de relleno.

Un valor de buffer TYPE_NUMERIC puede convenientemente ser interpretado con la funcion atof(3) de la librería C.

TYPE_REGEXP

Este tipo de campo acepta un dato concordante que corresponde con una expresión regular. Se establece como se especifica a continuación:

int set_field_type(FIELD *field, /* campo a modificar */

TYPE_REGEXP, /* tipo asociado */

char *regexp); /* expresion para comprobar */

La sintaxis para la expresión regular es la de regcomp(3). El chequeo de la concordancia de la expresión regular es realizado a la salida.

Manipulación del Buffer Field Directo

El atributo principal de un campo es lo que contiene su buffer. Cuando un formulario ha sido completado, su aplicación normalmente necesita saber el estado de cada buffer de campo. Puede averiguar esto con:

char *field_buffer(FIELD *field, /* campo a consultar */

int bufindex); /* numero de bufferes a revisar */

Normalmente, el estado del buffer con numero cero para cada campo se establece por el usuario editando acciones en ese campo. A veces es útil establecer el valor de buffer de numero cero (o algún otro) desde si aplicación:

int set_field_buffer(FIELD *field, /* campo a modificar */

int bufindex, /* numero de buffer a cambiar */

char *value); /* cadena del valor a establecer */

Si el campo no es lo suficientemente grande y no puede ser restablecido su tamaño a un tamaño mayor para contener el valor especificado, el valor se truncara para que se ajuste.

Llamando a field_buffer() con un puntero nulo dará un error. Llamando a field_buffer() en un campo que no es el seleccionado actualmente por la entrada devolverá un valor correcto. Llamando a field_buffer() con un campo que no es actualmente seleccionado por la entrada puede no ser necesario dar un valor de buffer de campo correcto, porque el dato introducido no es necesariamente copiado al buffer cero antes de la salida del chequeo de validación. Para garantizar que el valor de buffer devuelto refleja en la realidad de la pantalla, llame a field_buffer en (1) una salida de la rutina

de del chequeo de validación (2) desde la inicialización del campo o del formulario o la terminación, o (3) justo después de que una solicitud de REQ_VALIDATION haya sido procesada por el dispositivo de formularios.

Atributos de Formularios (Forms)

Como con los atributos de campo, los atributos de formularios heredan un valor por defecto desde la estructura del formulario por defecto del sistema. Este valor puede ser cambiado o establecido o estas funciones utilizando un argumento de puntero de formulario a NULL.

El atributo principal de un formulario es su lista de campos. Puede preguntar su valor y cambiar esta lista con:

int set_form_fields(FORM *form, /* formulario a modificar */

FIELD **fields); /* campos a conectar */

char *form_fields(FORM *form); /* campos del formulario */

int field_count(FORM *form); /* cuenta de campos conectados */

El segundo argumento de set_form_field() puede ser un NULL- tabla de punteros de campo terminados como el requerido por new_form(). En este caso, los campos antiguos del formulario están desconectados pero no liberados (y elegidos para ser conectados a otros formularios), entonces los nuevos campo esta conectados.

Puede también esta a nulo, en cuyo caso los campo antiguo están desconectados (y no liberados) pero nuevos no están conectados.

La función field_count() simplemente cuenta el numero de campo conectados a un formulario dado. Devuelve -1 si el argumento de puntero de formulario es NULL.

Control de Visualización de Formularios (Forms)

En resumen de esta sección, puede ver que la visualización del formulario normalmente comienza por la definición de su tamaño (y campos), representándola, y refrescando la pantalla. Hay un paso oculto antes de la representación, que es la asociación del formulario con una ventana estructurada (en realidad, un para de ventanas) en las que serán visualizadas. Por defecto, la librería formularios asocia cada formulario con la pantalla completa stdscr.

Haciendo este paso explícito, puede asociar el formulario con la estructura de ventana declarada en su pantalla de visualización. Esta puede ser útil si quiere adaptara la visualización del formulario a diferentes tamaños de pantalla, dinámicamente formularios embaldosados en la pantalla, o utilizar un formulario como parte de una planificación de interfaz manejada con Paneles.

Las dos ventanas asociadas con cada formulario tiene las mismas funciones como sus análogas en la Librería_Menú. Ambas ventanas son pintadas cuando el formulario es asociado y borrado cuando el formulario es quitado de su asociacion.

La salida o estructura de ventana no es de otra manera tocada por las rutinas de

formularios. Existe para que el programador pueda asociar un titulo, un borde, o quizás texto de ayuda con el formulario y tenerla refrescado apropiadamente o borrado a tiempo de asociación / sin asociación. La ventana de entrada o subventana esta donde la pagina de formulario actual es en realidad visualizada.

Para declarar sus propias estructuras de ventanas para una forma, necesitara saber el tamaño del rectángulo limite de la forma. Puede conseguir esta información con:

int scale_form(FORM *form, /* formulario a consultar */

int *rows, /* líneas de formularios */

Las dimensiones del formulario se pasan en las asociaciones apuntadas por los argumentos. Una vez que tiene esta información, puede usarla para declarar ventanas, entonces usar una de esta funciones:

int set_form_win(FORM *form, /* formulario a modificar */

WINDOW *win); /* estructura de ventana a conectar */

WINDOW *form_win(FORM *form); /* estructura de ventana de formulario */

int set_form_sub(FORM *form, /* formulario a modificar */

WINDOW *win); /* subventana de formulario a conectar */

WINDOW *form_sub(FORM *form); /* subventana de formulario */

Note que las operaciones de curse, incluyendo refresh(), en la forma, podrían estar hechas en la estructura de ventana, no en la subventana de forma.

Es posible chequear desde su aplicación si todas los campo paginables están en realidad visualizadas dentro de la subventana de forma.

Utilice estas funciones:

int data_ahead(FORM *form); /* formulario a consultar */

int data_behind(FORM *form); /* formulario a consultar */

La función data_ahead() devuelve TRUE si (a) el campo actual esta actualizado y no ha visualizado el dato de la derecha, (b) el campo actual es multi-linea y hay datos de fuera de la pantalla debajo de él.

La función data_behind() devuelve TRUE si la primara posición de carácter(esquina superior izquierda) esta fuera de la pantalla (no siendo visualizada).

Finalmente, hay una función para restaura el cursor de ventana del formulario al valor esperado por el dispositivo del formulario:

Int pos_form_cursor(FORM*)

Si su aplicación cambia el cursor de ventana de la forma, llame a esta función antes de manejar el control de vuelta al dispositivo de formularios para re-sincronizarlo.

Entrada en el Dispositivo de Formularios (Forms)

La función form_driver() maneja las respuestas de entrada virtuales para la navegación del formulario, editando, y validando solicitudes, justamente como menu_driver lo hace para menús (vea la sección Procesamiento_de_la_entrada_de_Menú).

int form_driver(FORM *form, /* formulario para pasar a la entrada */

int request); /* código de solicitud de formulario */

Su función de virtualización de entrada necesita tomar la entrada y entonces convertirla a un carácter alfanumérico (que es tratado como un dato introducido en el campo actualmente seleccionado), o una solicitud de procesamiento de la forma.

El dispositivo del formulario suministra enlaces (a través de las funciones de validación de entrada y de terminación del campo) con los que su código de aplicación puede chequear que entrada es tomada por el dispositivo que concuerde con el que era esperado.

Petición de Navegación de Página

Estas solicitudes causan que el nivel de pagina se mueva a través del formulario, provocando la visualización de una nueva pantalla de formulario.

REQ_NEXT_PAGE

Mover a la siguiente pagina del formulario.

REQ_PREV_PAGE

Mover a la anterior pagina del formulario.

REQ_FIRST_PAGE

Mover a la primera pagina del formulario.

REQ_LAST_PAGE

Mover a la ultima pagina del formulario.

Estas solicitudes tratan la lista como cíclica; esto es, REQ_NEXT_PAGE desde la ultima pagina va a la primera, y REQ_PREV_PAGE desde la primera pagina va a la ultima.

Petición de Navegación Inter-Campo (Field)

Estas peticiones manejar la navegación entre campos de la misma pagina.

REQ_NEXT_FIELD

Mover al siguiente campo.

REQ_PREV_FIELD

Mover al campo anterior.

REQ_FIRST_FIELD

Mover al primer campo.

REQ_LAST_FIELD

Mover al ultimo campo.

REQ_SNEXT_FIELD

Mover al siguiente campo ordenado.

REQ_SPREV_FIELD

Mover al anterior campo ordenado.

REQ_SFIRST_FIELD

Mover al primer campo ordenado..

REQ_SLAST_FIELD

Mover al ultimo campo ordenado.

REQ_LEFT_FIELD

Mover al campo de la izquierda.

REQ_RIGHT_FIELD

Mover al campo de la derecha.

REQ_UP_FIELD

Mover al campo de arriba.

REQ_DOWN_FIELD

Mover al campo de abajo.

Estas peticiones tratan la lista de campos en la pagina como cíclica; esto es REQ_NEXT_FIELD desde el ultimo campo va al primero, y REQ_PREV_FIELD desde el primer campo va al ultimo. El orden de los campo para esto (y las solicitudes REQ_FIRST_FIELD y REQ_LAST_FIELD) es simplemente el orden de los punteros de campo en la tabla del formulario (como se inicializa con new_form() o set_form_fields()).

También es posible atravesar los campos como si hubieran sido clasificados en orden de posición en pantalla, así la secuencia va de izquierda a derecha y de arriba abajo. Para hacer esto, utilice el segundo grupo de cuatro peticiones de movimiento

ordenado.

Finalmente, es posible moverse entre campos utilizando direcciones visuales arriba, abajo, derecha e izquierda. Para llevar a cabo esto, utilice el tercer grupo de cuatro solicitudes. Note, sin embargo, que la posición del formulario para los propósitos de estas peticiones es la esquina superior izquierda.

Por ejemplo, suponga que tiene un campo multi-linea B, y dos campos de línea simple A y C en la misma línea con B, con A ala izquierda de B y C a la derecha de B. Una REQ_MOVE_RIGHT desde A ira a B solo si A,B, y C todos comparten la misma primera línea; de otra forma pasaran por alto B hacia C.

Petición de Navegación Intra-Campo (Field)

Estas peticiones conducen el movimiento del cursor del editor dentro el campo seleccionado actualmente.

REQ_NEXT_CHAR

Mover al siguiente carácter.

REQ_PREV_CHAR

Mover al carácter anterior.

REQ_NEXT_LINE

Mover a la siguiente línea

REQ_PREV_LINE

Mover a la línea anterior.

REQ_NEXT_WORD

Mover a la siguiente palabra.

REQ_PREV_WORD

Mover a la palabra anterior.

REQ_BEG_FIELD

Mover al comienzo del campo.

REQ_END_FIELD

Mover al final del campo.

REQ_BEG_LINE

Mover al principio de la línea.

REQ_END_LINE

Mover al final de la línea.

REQ_LEFT_CHAR

Mover a la izquierda en el campo.

REQ_RIGHT_CHAR

Mover a la derecha en el campo.

REQ_UP_CHAR

Mover hacia arriba en el campo.

REQ_DOWN_CHAR

Mover hacia abajo en el campo.

Cada palabra se separa del carácter anterior y del siguiente con un espacio en blanco. El comando para mover al principio y el final de la línea o del campo busca el primer o el ultimo carácter de relleno en sus rangos.

Petición de Paginar

Los campos que son dinámicos y han crecido y los campos explícitamente creados con líneas fuera de pantalla se pueden paginar. Los campos de una línea se pueden paginar horizontalmente; los campos multi-linea se pueden paginar verticalmente. La mayor parte de la paginación se provoca editando y con el movimiento intra-campo (la librería pagina el campo para conservar el cursor visible). Es posible explicitar la petición de paginación con las siguientes solicitudes:

REQ_SCR_FLINE

Pagina verticalmente una línea hacia delante.

REQ_SCR_BLINE

Pagina verticalmente una línea hacia atrás.

REQ_SCR_FPAGE

Pagina verticalmente una pagina hacia delante..

REQ_SCR_BPAGE

Pagina verticalmente una pagina hacia delante..

REQ_SCR_FHPAGE

Pagina verticalmente media pagina hacia delante..

REQ_SCR_BHPAGE

Pagina verticalmente media pagina hacia delante..

REQ_SCR_FCHAR

Pagina horizontalmente un carácter hacia delante.

REQ_SCR_BCHAR

Pagina horizontalmente un carácter hacia atrás.

REQ_SCR_HFLINE

Pagina horizontalmente una profundidad de campo hacia delante.

REQ_SCR_HBLINE

Pagina horizontalmente una profundidad de campo hacia atrás.

REQ_SCR_HFHALF

Pagina horizontalmente media profundidad de campo hacia delante.

REQ_SCR_HBHALF

Pagina horizontalmente media profundidad de campo hacia atrás.

Para los propósitos de paginación, una pagina de un campo es la altura de su parte visible.

Petición de Editado de Campos (Field)

Cuando pase un carácter ASCII al dispositivo de los formularios, es tratado como una petición de añadir el carácter al buffer de datos del campo. Si esto es un inserción o un reemplazamiento depende del modo de edición del campo (inserción es por defecto).

Las siguientes peticiones soportan el editado del campo y cambiar el modo de edición:

REQ_INS_MODE

Establecer el modo de inserción.

REQ_OVL_MODE

Establecer el modo de sobrelapamiento.

REQ_NEW_LINE

Petición de nueva línea (mire arriba para una explicación).

REQ_INS_CHAR

Insertar el espacio de localización de carácter.

REQ_INS_LINE

Insertar una línea en blanco en la localización del carácter.

REQ_DEL_CHAR

Borrar un carácter donde este el cursor.

REQ_DEL_PREV

Borrar la palabra anterior al cursor.

REQ_DEL_LINE

Borrar una línea donde este el cursor.

REQ_DEL_WORD

Borrar una palabra donde este el cursor.

REQ_CLR_EOL

Limpiar hasta el final de la línea.

REQ_CLR_EOF

Limpiar hasta el final del campo.

REQ_CLEAR_FIELD

Limpiar el campo entero.

El comportamiento de las peticiones REQ_NEW_LINE y REQ_DEL_PREV es complicada y controlada en mayor parte por un par de opciones de formularios. Los casos especiales son provocados cuando el curso esta al principio de un campo, o en la ultima línea del campo.

Primero, consideramos REQ_NEW_LINE:

La conducta normal de REQ_NEW_LINE en el modo de inserción es para romper la línea actual, en la posición del cursor de edición, insertando la parte de línea actual después del cursor como una nueva línea siguiente a la actual y moviendo el cursor al principio de la nueva línea (puede pensar esto como si insertara una nueva línea en el buffer de campo).

La conducta normal de REQ_NEW_LINE en modo de sobrelapado es limpiar la línea actual de la posición del cursor de edición al final de la línea. El cursor es entonces movido al principio de la siguiente línea.

Sin embargo, REQ_NEW_LINE al principio de un campo, o en la ultima línea del campo, en cambio hace un REQ_NEXT_FIELD. Si la opción O_NL_OVERLOAD esta desactivada, esta acción especial también esta desactivada.

Ahora vamos a considerar REQ_DEL_PREV:

La conducta normal de REQ_DEL_PREV es borrar el carácter anterior. Si el modo de

inserción esta activo, y el cursor esta al comienzo de la línea, y el texto de la línea no se ajusta a la anterior, en cambio añade el contenido de la línea actual a la anterior y borrar la línea actual (puede pensar esto como el borrado de una nueva línea del buffer de campo).

Sin embargo, REQ_DEL_PREV al principio de un campo es tratado como una REQ_PREV_FIELD.

Si la opción O_BS_OVERLOAD esta desactivada, esta acción especial esta desactivada y el dispositivo de los formularios devuelve un E_REQUEST_DENIED.

Vea las Opciones de Formularios para la discusión de cómo establecer y limpiar las opciones de overload.

Petición de Orden

Si el tipo de nuestro campo esta ordenado, y tiene funciones asociadas de conseguir el siguiente y el valor previo del tipo desde un valor dado, hay peticiones que pueden conseguir este valor dentro del buffer de campo:

REQ_NEXT_CHOICE

Coloca el valor sucesor al valor actual en el buffer.

REQ_PREV_CHOICE

Coloca el valor antecesor al valor actual en el buffer.

De los tipos de campos ya implementados, solo TYPE_ENUM tiene construido una función de sucesor y antecesor. Cuando defina un tipo de campo por usted mismo (vea Tipos_de_validación_del_Usuario), puede asociar sus propias funciones de orden.

Comandos de la Aplicación

Las peticiones de formularios son representadas como enteros por encima del valor de curses mayor que KEY_MAX y menor que o igual a la constante MAX_COMMAND. Si su rutina de virtulización de la entrada devuelve un valor superior a MAX_COMMAND, el dispositivo de formularios lo ignorara.

Cambiar Enlaces en los Campos(Field)

Es posible establecer enlaces de funciones para ser ejecutados en cualquier momento que el campo actual o el formulario cambien. Aquí tiene unas funciones que aportan esto:

typedef void (*HOOK)(); /* puntero a una función que devuelve void */

int set_form_init(FORM *form, /* formulario a modificar */

HOOK hook); /* inicialización del enlace */

HOOK form_init(FORM *form); /* formulario a consultar */

int set_form_term(FORM *form, /* formulario a modificar */

HOOK hook); /* enlace de terminación */

HOOK form_term(FORM *form); /* formulario a consultar */

int set_field_init(FORM *form, /* formulario a modificar */

HOOK hook); /* inicialización del enlace */

HOOK field_init(FORM *form); /* formulario a consultar */

int set_field_term(FORM *form, /* formulario a modificar */

HOOK hook); /* enlace de terminación */

HOOK field_term(FORM *form); /* formulario a consultar */

Estas funciones le permiten establecer o testear cuatro enlaces diferentes. En cada conjunto de funciones, el segundo argumento debería ser y una dirección de una función de enlace. Esta funciones difieren solo en el tiempo de llamar al enlace.

form_init

Este enlace se le llama cuando el formulario es asociado; también, justo después de cada operación de cambio de pagina.

field_init

Este enlace se le llama cuando el formulario es asociado; también, después de cambios en el campo.

field_term

Este enlace se le llama después de la validación de campo; esto es, justo antes de que el campo sea modificado. Es también llamado cuando el formulario no esta asociado.

form_term

Este enlaces es llamado cuando el formulario es quitado de su asociación; también, justo antes de cada operación de cambio de pagina.

Llamadas a estos enlaces puede ser provocados

1.Cuando las peticiones de edición de usuario son procesadas por el dispositivo de los formularios.

2.Cuando la pagina actual es modificada por una llamada a set_current_field().

3.Cuando la pagina actual es modificada por una llamada a set_form_page().

Ver Comandos de Cambios de Campos para la discusión de los dos últimos casos.

Puede establecer enlaces por defecto para todos los campos pasando un conjunto de

funciones con NULL como primer argumento.

Puede desactivar cualquiera de estos enlaces estableciéndolos a NULL, como valor por defecto.

Comandos de cambios de Campos (Field)

Normalmente, la navegación a través del formulario será dada por las peticiones de la entrada de usuario. Pero algunas veces es útil ser capaz de mover el foco para editar y visualizar bajo el control de nuestra aplicación, o preguntar que campo esta actualmente. Las siguientes instrucciones ayudan a acabar esto:

int set_current_field(FORM *form, /* formulario a modificar */

FIELD *field); /* campo a cambiarse */

FIELD *current_field(FORM *form); /* formulario a consultar */

int field_index(FORM *form, /* formulario a consultar */

FIELD *field); /* campo para coger el indice */

La función field_index() devuelve un índice del campo dado de la tabla de campos de formularios (la tabla pasada a new_form() o set_form_fields()).

El campo actual inicial de un formulario es el primer campo activo en su pagina. La función set_form_fields() resetea esto.

Es posible también moverse alrededor por las paginas.

int set_form_page(FORM *form, /* formulario a modificar */

int page); /* pagina a la que ir (0-originen) */

int form_page(FORM *form); /* volver a la pagina actual del formulario */

La pagina inicial de un formulario creado nuevo es 0. La función set_form_field() resetea esto.

Opciones de Formularios (Forms)

Como con los campos, los formularios pueden tener bits de opciones de control. Pueden ser cambiados o testeados con estas funciones:

int set_form_opts(FORM *form, /* formulario a modificar */

int attr); /* atributo a establecer */

int form_opts_on(FORM *form, /* formulario a modificar */

int attr); /* atributos a activar */

int form_opts_off(FORM *form, /* formulario a modificar */

int attr); /* atributos a desactivar */

int form_opts(FORM *form); /* formulario a consultar */

Por defecto, todas las opciones esta activadas. Aquí están disponibles algunos bits de opciones:

O_NL_OVERLOAD

Permite el sobrecargamiento de REQ_NEW_LINE como se describe en Peticiones de Edición. El valor de esta opción es ignorado en campo dinámicos que no han alcanzado su tamaño limite; estos no tienen ultima línea, por lo que las circunstancia para provocar un REQ_NEW_FIELD nunca se alcanzan.

O_BS_OVERLOAD

Permite el sobrecargamiento de REQ_DEL_PREV como se describe en Peticiones de Edición.

Los valores de opción son mascaras de bits y puede estar compuestas con lógica- o en el modo obvio.

Tipos de validación del Usuario

La librería form le da la capacidad para definir tipo de validación de usuario por usted mismo. Mas aun, los argumentos adicionales opcionales de set_field_type efectivamente le permiten parametrizar los tipos de validación. Muchos de las complicaciones en la interfaz de los tipos de validación tienen que ver con el manejo de argumentos adicionales en las funciones de validación e usuario.

Tipos de Uniones

El modo más simple de crear tipos de dato de usuario es componerlos desde otros preexistentes:

FIELD *link_fieldtype(FIELDTYPE *type1,

FIELDTYPE *type2);

Esta función crea un tipo de campo que aceptara cualquier valor legal para sus argumentos de tipo campo (que pueden estar predefinidos o definidos por el programador). Si una llamada a set_field_type() después requiere argumentos, el nuevo tipo compuesto espera todos los argumentos para el primer tipo, que todos los argumentos para el segundo. Ordenar funciones (ver Petición_de_Orden) asociadas con los componentes de los tipos trabajaran en la composición; lo que hace es chequear la función de validación para el primer tipo, entonces por el segundo, para figurar que tipo que contiene el buffer debería ser tratado.

Nuevos Tipos de Campos

Para crear un tipo de campo que cumpla los requisitos, necesita especificar una o las dos cosas siguientes:

Una función de validación de carácter, para chequear cada carácter como es introducido.

Una función de validación de campo para aplicarla a la salida del campo.

Aquí tiene como hacer esto:

typedef int (*HOOK)(); /* puntero a una función que devuelve un entero */

FIELDTYPE *new_fieldtype(HOOK f_validate, /* validador de campo */

HOOK c_validate) /* carácter validador */

int free_fieldtype(FIELDTYPE *ftype); /* tipo a liberar */

Al menos uno de los argumentos de new_fieldtype() debe ser un no NULL. El dispositivo de los formularios llamara automaticamente la nueva función de validación en los puntos apropiados del procesando un campo del nuevo tipo.

La función free_fieldtype() deslocaliza el argumento fieldtype, liberando todo el almacenamiento asociado con el.

Normalmente, un validador de campo es llamado cuando el usuario intenta deja el campo. Su primer argumento es un puntero de campo, desde el cual puede coger el buffer de campo y testearlo. Si la función devuelve TRUE, la operación termina con éxito; si devuelve FALSE, el cursor de edición permanece en el campo.

Un validador de carácter coge el carácter pasado como primer argumento. También devolvería TRUE si el carácter es valido, FALSE en otro caso.

Argumentos de Funciones de validación

Sus funciones de validación de campo- y de carácter- se les pasara también un segundo argumento. Este segundo argumento es la dirección de la estructura (que llamara a una pila) construida desde cualquiera de los argumentos de tipos específicos de campo pasados a set_field_type(). Si estos argumentos no están definidos para el tipo de campo, este argumento de puntero de pila será NULL.

Para arreglar que estos argumentos sean pasados a sus funciones de validación, debe asociar un conjunto pequeño de funciones de almacenamiento-dirección con el tipo. El dispositivo de formularios utilizara esta pila sintetizada desde los argumentos posteriores de cada argumento de set_field_type(), y un puntero a la pila será pasado a las funciones de validación.

Aquí esta como usted hace la asociación:

typedef char *(*PTRHOOK)(); /* puntero a la función devolviendo (char *) */

typedef void (*VOIDHOOK)(); /* puntero a la función devolviendo void */

int set_fieldtype_arg(FIELDTYPE *type, /* tipo a modificar */

PTRHOOK make_str, /* hacer la estructura desde argumentos */

PTRHOOK copy_str, /* hacer copia de la estructura */

VOIDHOOK free_str); /* liberar almacenamiento de la estructura */

Aquí esta como el enlace almacenamiento-dirección se usa:

make_str

Esta función se le llama desde set_field_type(). Coge un argumento, una va_list de argumentos de tipo especifico pasados para set_field_type(). Se espera devolver un puntero de pila a una estructura de datos que encapsule estos argumentos.

copy_str

Esta función se le llama desde las funciones de la librería form que localizan nuevas instancias de campos. Se espera coger un puntero de pila, copiar la pila as almacenamiento localizado, y devolver la dirección de la pila copiada.

free_str

Esta función se le llama desde las rutinas de campo- y tipo-deslocalizacion en la librería. Toma argumento de un puntero de pila, y se espera que libere el almacenamiento de la pila.

Las funciones make_str y copy_str pueden devolver NULL para mostrar fallo de localización. Las rutinas de librería las llamaran y devolverán una indicación de error cuando esto ocurra. De este modo, sus funciones de validación nunca deberían ver un puntero de fichero NULL y no necesita chequear especialmente por ello.

Funciones de orden para Tipos de Usuario

Algunos tipos de campo de usuario son simplemente ordenados en la misma manera bien definida que TYPE_ENUM lo esta. Por cada tipo, es posible definir unas funciones de sucesor y predecesor para soportar las peticiones REQ_NEXT_CHOICE y REQ_PREV_CHOICE. Aquí esta como:

typedef int (*INTHOOK)(); /* puntero para la función que devuelve un entero */

int set_fieldtype_arg(FIELDTYPE *type, /* tipo a modificar */

INTHOOK succ, /* coger el valor del sucesor */

INTHOOK pred); /* coger el valor del predecesor */

Los argumentos de sucesor y predecesor serán cada uno pasados con dos argumentos; un puntero de campo, y un puntero de pila (como para las funciones de validación). Se espera que se utilice la función field_buffer() para leer el valor actual, y set_field_buffer() en el buffer para establecer el valor siguiente o previo. El enlace puede devolver TRUE para indicar éxito (un valor siguiente o previo legal esta establecido) o FALSE para indicar fallo.

Evitar Problemas

La interfaz para definir tipos de usuario es complicada y delicada. Mejor que intentar crear un tipo de usuario enteramente para cumplir los requisitos, puede comenzar estudiando el código fuente de la librería para cualquier tipo predefinido que se parezca lo mas posible a lo que usted quiere.

Utilice el código como modelo, y desarrollarlo hacia lo que realmente quiera. Evitara muchos problemas y molestias de esta manera. El código en la librería ncurses ha sido específicamente exento del copyright del paquete para soportar esto.

Si el tipo de usuario define funciones de orden, tiene que hacer algo intuitivo con el campo en blanco. Una convención útiles es hacer el sucesor de un campo en blanco al valor mínimo del tipo, y su predecesor al máximo.