Konfigurierbares Menü System für Eingebettete Systeme
Configurable Menu System for Embedded Applications

Roman Pollak

20. Januar 2010

1 Zusammenfassung

Bei Arbeiten mit kleinen System, die man mit eine 2 Zeilen Character LCD Display bedienen möchte, kommt man immer wieder an den Punkt, wo man etwas Konfigurieren, Einstellen etc. möchte. Hier wird ein kleines aber sehr an die Voraussetzungen Konfigurierbares Menü System beschrieben.

2 Einleitung

Bei einem Hobby Projekt, brauchte ich ein Menü System, mit dem man verschiedene Einstellungen Konfigurieren kann. Dies sollte mit einem 2-4 Zeiligem LCD Display (HD78440 typ) möglich sein, mit einem Rotary Encoder oder mit drei Tasten. Auf der Suche, bin ich nur auf wenig solche Systeme gestoßen, die keine grafischen Ausgabe brauchen. Also blieb mir nichts anderes, als selber in die Tasten zu hauen :).
Was sollte es den können? Viele mögen sich noch erinnern an die alten Laser Drucker. Viele hatten ein kleines LCD Display, über das man Einstellungen wie die Serielle Schnittstellen Baud rates, 1/2 Stopp Bits etc. ändern konnte. Und das alles mit ein paar tasten ohne Maus, touch Screen.
Bei meinem Projekt (ein Sternen Himmel, mit Sonne, Erde, Planeten und natürlich den Sternen) sollte man einstellen können über ein Menü, welches Mode gerade eingestellt ist (Tag, Nacht, Fancy:) oder soll die Sonne aufgehen oder untergehen. Ich wollte das dies alles in dem Programm Speicher eines AVR Mega32 Platz hat und sehr klein ist. Da ich ein Rotary encoder einem Taste verwendet habe, musste dies reichen um es zu bedienen. Allerdings... Im hinter Kopf hatte ich bereits ein anderes Projekt, an dem ich weiter arbeiten möchte, Scope Clock, auf dem es auch grafische Ausgabe möglich ist.

Daraus ergaben sich diese Anforderungen :

- Verwendbar für gross und klein Systeme (von AVR bis SPARC)
- Nicht verwendete Elemente resp. Programm Teile werden gar nicht erst Compiliert
- Selbstständig Arbeitend dh. wird ein Radio Button geändert, wird keine Callback Routine aufgerufen, sondern wird direkt die Variable geändert
- Verwendung bei Character basierenden wie auch Grafischen Displays resp. Ausgabe
- Geringe Speicher bedarf
- Portable ...
- Alles Tools (compiler, precompiler etc.) sollte auf Unix/Linux vorhanden resp. lauf bar sein.

3 Realisation

Ich habe viel gesucht resp. modern gesagt gegoogled, jedoch wenig wirklich brauchbares gefunden. Jedoch es gab mir Ideen und Anhaltspunkte fuer die Daten Strukturen. Dann kam mir im Sinn, das ich ja ein Buch habe, das genau User Interfaces in Embedded Applikationen behandelt. Im Oreilly Buch Yacc&Lex wird auch ein solches System beschrieben.
Also Editor anwerfen und los "C"-len...

3.1 Implementierung

Das wichtigste wohl am ganzem Projekt ist, die dahinter sich versteckende Daten Struktur.

```
struct menu_item {
char *title ;
  struct menu_item *next;       /*next item */
  struct menu_item *prev;       /*prev item */
  struct menu_item *up;

int type;
union {
int (*func) ();
int *val;
struct {
  int cnt;
 int *val;
 struct item *items;
}item;
  struct {
    int min,max;
    int *val;
  }cval;
  struct menu_item *down;
}u;
};
```

Diese Struktur ist Kern, des ganzes Menü Systems. In dem "title" steht natürlich der Titel des Menüüü Punktes wie "Files" oder "Einstellungen". Die Menü Punkte innerhalb des gleichen Menü (aka alles was in "File"), werden mit verkette Liste (next,prev) verbunden. Dies vereinfacht später, uns zu "bewegen" innerhalb des Menüs mit Up/Down. Den wir müssen nur einen Zeiger schieben und schon ist der nächste Menü Punkt aktiv. Der "up" Zeiger, zeigt an das darüber liegende Menü, zb. sind wir im "Einstellungen->Serielle Schnittstelle->Baud Rate" Zeigt der "up" auf Serielle Schnittstelle. So können wir schnell wieder zurück ins darüber liegende Menü, wenn wir eine "Back" Taste haben. Bei Drei Tasten müssen wir anders vorgehen, auf das ich später eingehen möchte. Um euch nicht im Dunkeln zu lassen,das geht auch :).
In dem "type" steht um was für eine Art des Elementes handelt es sich. Ist es ein Untermenü oder eine Radio Liste, Check Liste oder möchte man eine Zahl ein-

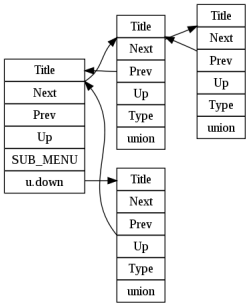

Abbildung 1: Beispiel der Verkettung der Menü Punkte mit Untermenü

geben. Die dahinter stehende "union" verpackt alle zusätzlichen Informationen, die man für die Elemente benötigt. Wie zb. ist es quasi ein Link zum Untermenü, steht im type "SUB_MENU" und "down" zeigt auf die entsprechende Untermenü Liste (siehe 3.1).

Listing 1: Definition der Menü Typen

```
#define BACK_MENU  1
#define EXIT_MENU  2
#define RADIO_MENU  3
#define SUB_MENU  4
#define CHECK_MENU  5
#define FUNC_MENU   6
#define VAR_MENU    7
#define SETV_MENU   8
```

Das Listing (1) zeigt die (z.Z) vorgegebene Menü Typen.
Der "BACK_MENU" ist ein ein Pseudo Typ, den er ist dafür bestimmt, wenn man nur drei tasten hat in einem System, um in dem Menü Baum zurück zum oberen Menü zu gelangen. Als "title" sollte hier etwas wie "<<Back" oder "Zurück" stehen. Verwendet man im Ziel System vier taste, kann man auf diesen Typ verzichten, den das "up" steht bei jedem so oder so, wie wir später sehen werden.
Der "EXIT_MENU" typ, ist auch ein Pseudo typ, der es uns ermöglicht ganz das Menü zu verlassen.
"RADIO_MENU" und "CHECK_MENU" sind sehr ähnlich und doch mit grossem unterschied. Die alten Radio's hatten ja grosse tasten, bei den immer nur eine taste gedrückt war. Druckte man eine andere Taste, so springt die gedruckte raus und die, die man gedrückt hat wurde eingerastet. Dies macht man mit "RADIO_MENU" auch. Man hinterlegt in der "Menu_item" Struktur den Zeiger auf die variable, die geändert werden soll. Nun man möchte die Werte die in diese variable geschrieben werden ja nicht sehen, sondern viel mehr das was ein Mensch versteht. Dafür muss man eine zweite Struktur hinterlegen, die den entsprechenden Text und die dazu gehörenden variablen werte hat. Beim betätigen der entspr. Taste, wird der diese Wert automatisch in die variable geschrieben. Noch etwas... Ist schon ein werte in der Variable hinterlegt resp. steht etwas drin was den Werten entspricht, die per Menü System definiert sind, sieht man das.

Beim "CHECK_MENU" ist der unterschied darin, dass mehrere punkte selektiert werden können, in dem man nicht die variable als ganze setzt, sondern nur ein entsprechendes Bit. Dies ist wichtig zu wissen, den die Werte müssen immer 2^x sein!!! Bei beiden Typen wird eine Item Struktur verwendet um die Namen und Werte entsprechend zu verwalten (siehe 2).

Listing 2: Definition der Menu Typen

```
struct item {
char *title;
int val;
};
```

Diese wird als eine Liste deklariert wie im Listing 3 gezeigt wird.

Listing 3: Definition der Menu Typen

```
char item_txt_4_1[] PROGMEM ="Ab";
char item_txt_4_2[] PROGMEM ="Auf";
struct item item_s_4[] PROGMEM = {
{ item_txt_4_1,-1},
{ item_txt_4_2,1},
};
```

Auf dem Display, wird man nur ein "Ab" oder "Auf" sehen, jedoch in die variable wird ein -1 oder 1 geschrieben. Ein Hinweis, diese sind als "int" deklariert.
Der "FUNC_MENU" ist dazu gedacht, um eine Funktion aufzurufen.
"VAR_MENU" zeigt der Inhalt einer variable.
"SETV_MENU" kann man eine Variable aus Zahl setzten. Die Idee dahinter ist, das man in der union u.cval den minimal und maximal wert setzten kann und die variable bewegt sich irgendwo dazwischen, wie zb. min=0, max=100.

Im Source code, findet Ihr alle wichtigen hilf Routinen und um solches Menü System zu bauen. ok ok ok .. Programmieren. Entsprechendes Beispiel ist auch dabei, das man auf einem Linux System austesten kann.

3.2 Menü Generator

Obvoll diese Daten Strukturen sind recht einfach und zuverlässig sind, verliert man schon nach ein paar Menü punkten die Übersicht. Dies brachte mich auf die Idee, einen möglichst einfachen Menü Generator zu erstellen. Der ein für Menschen relativ übersichtlichen Text mit ein paar key Wörtern, in die entsprechende Struktur übersetzt und im ganzen Menü System verlinkt.

```
<?xml version="1.0" ?>

<menus>
 <menu type="radio" variable="mode">
 <title>Mode</title>
 <item val="1">Stern Sim.</item>
 <item val="2">Alle</item>
 <item val="3">Nacht</item>
```

```
 <item val="4">Tag</item>
 </menu>

 <menu type="sub">
 <title>Simulation</title>
  <menu type="radio" variable="ab_auf">
   <title>Auf-/Abgang</title>
   <item val="-1">Ab</item>
   <item val="1">Auf</item>
  </menu>

  <menu type="radio" variable="auto_v">
    <title>Auto Mode</title>
    <item val="1">Auto</item>
    <item val="0">Manual</item>
   </menu>

 <menu type="change" min="0" max="100" variable="auto_v">
    <title>Auto Mode</title>
 </menu>
</menu>

<menu type="exit" >
 <title>Exit</title>
 </menu>

</menus>
```

Listing 3.2 zeigt solch ein xml Menü file. Anhand diese Vorgaben, wird ein C header file Listing 3.2 erstellt.

```
/* Menu configuration as defined
// [Mode] type [radio]
// [Simulation] type [sub]
//  [Auf-/Abgang] type [radio]
//  [Auto Mode] type [radio]
//  [Auto Mode] type [change]
// [Exit] type [exit]
 Menu configuration as defined */


char item_txt_2_1[] PROGMEM ="Stern Sim.";
char item_txt_2_2[] PROGMEM ="Alle";
char item_txt_2_3[] PROGMEM ="Nacht";
char item_txt_2_4[] PROGMEM ="Tag";
struct item item_s_2[] PROGMEM = {
{ item_txt_2_1,1},
{ item_txt_2_2,2},
{ item_txt_2_3,3},
{ item_txt_2_4,4},
};
char item_txt_4_1[] PROGMEM ="Ab";
char item_txt_4_2[] PROGMEM ="Auf";
struct item item_s_4[] PROGMEM = {
{ item_txt_4_1,-1},
{ item_txt_4_2,1},
};
char item_txt_5_1[] PROGMEM ="Auto";
char item_txt_5_2[] PROGMEM ="Manual";
struct item item_s_5[] PROGMEM = {
{ item_txt_5_1,1},
{ item_txt_5_2,0},
};
char menu_text_2[] PROGMEM="Mode"; /*forward declaration **Mode** */
char menu_text_3[] PROGMEM="Simulation"; /*forward declaration **Simulation** */
char menu_text_4[] PROGMEM="Auf-/Abgang"; /*forward declaration **Auf-/Abgang** */
char menu_text_5[] PROGMEM="Auto Mode"; /*forward declaration **Auto Mode** */
char menu_text_6[] PROGMEM="Auto Mode"; /*forward declaration **Auto Mode** */
char menu_text_7[] PROGMEM="Exit"; /*forward declaration **Exit** */
char menu_text_8[] PROGMEM="<<BACK"; /*forward declaration **<<BACK** */
struct menu_item menu_item_2; /*forward declaration **Mode** */
```

```
struct menu_item menu_item_3; /*forward declaration **Simulation** */
struct menu_item menu_item_4; /*forward declaration **Auf-/Abgang** */
struct menu_item menu_item_5; /*forward declaration **Auto Mode** */
struct menu_item menu_item_6; /*forward declaration **Auto Mode** */
struct menu_item menu_item_7; /*forward declaration **Exit** */
struct menu_item menu_item_8; /*forward declaration **<<BACK** */


/* Menu definition for Mode */
struct menu_item menu_item_2 PROGMEM = {
 .title=menu_text_2,
 .next=&menu_item_3,
 .prev=0,
 .up=0,
 .type=RADIO_MENU,
 .u.item.cnt=4,
 .u.item.items= item_s_2,
 .u.item.val=&mode,
};


/* Menu definition for Simulation */
struct menu_item menu_item_3 PROGMEM = {
 .title=menu_text_3,
 .next=&menu_item_7,
 .prev=&menu_item_2,
 .up=0,
 .type=SUB_MENU,
 .u.down=&menu_item_4,
};


/* Menu definition for Auf-/Abgang */
struct menu_item menu_item_4 PROGMEM = {
 .title=menu_text_4,
 .next=&menu_item_5,
 .prev=&menu_item_8,
 .up=&menu_item_3,
 .type=RADIO_MENU,
 .u.item.cnt=2,
 .u.item.items= item_s_4,
 .u.item.val=&ab_auf,
};


/* Menu definition for Auto Mode */
struct menu_item menu_item_5 PROGMEM = {
 .title=menu_text_5,
 .next=&menu_item_6,
 .prev=&menu_item_4,
 .up=&menu_item_3,
 .type=RADIO_MENU,
 .u.item.cnt=2,
 .u.item.items= item_s_5,
 .u.item.val=&auto_v,
};


/* Menu definition for Auto Mode */
struct menu_item menu_item_6 PROGMEM = {
 .title=menu_text_6,
 .next=0,
 .prev=&menu_item_5,
 .up=&menu_item_3,
 .type=SETV_MENU,
 .u.cval.val=&auto_v,
 .u.cval.min=0,
 .u.cval.max=100,
};


/* Menu definition for Exit */
struct menu_item menu_item_7 PROGMEM = {
 .title=menu_text_7,
 .next=0,
 .prev=&menu_item_3,
 .up=0,
 .type=EXIT_MENU,
};


/* Menu definition for <<BACK */
struct menu_item menu_item_8 PROGMEM = {
 .title=menu_text_8,
 .next=&menu_item_4,
 .prev=0,
 .up=&menu_item_3,
 .type=BACK_MENU,
};
```

Wie man sieht, sind alle “next”,”prev” schon richtig verlinkt und “up”s zeigen auf die richtigen Menü punkte, so wie die “down”s sind schon verlinkt. Hier komm ich zurück auf das “drei tasten Problem”! Vielleicht ist es euch aufgefallen, es gibt bei den Submenüs ein Punkt “<<BACK”. Dies ist ein Pseudo Punkt, der Angezeigt wird und auf das darüber liegende Menü verweist. Mit dieser kleinen “Hilfe” kommt man zurück von einem Submenü. Bei Systemen mit Vier tasten (“back”,”Up”,”Down”,”forward”) muss dieser Code nicht generiert werden. Dies kann man bei Menü Generator aus Option mit geben.

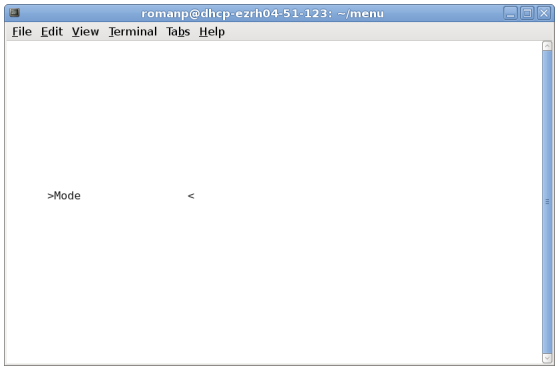

Abbildung 2: Screenshoot von Menü Systems unter Linux

Der Menü Generator selber, ist mit GNU Awk mit XML Extension geschrieben (http://home.vrweb.de/~juergen.kahrs/gawk/XML/ erhältlich oder ganz einfach nach “xml gawk” googlen). Aufgerufen wird das ganze mit “xgawk -f menu.awk input.xml”. Der Output wird auf dem standart output raus gegeben. Mit “xgawk -f menu.awk -v back_arg=1 input.xml” wird bei jedem Submenü auch ein “Back” erstellt. Dies ist vor allem bei Menüs mit drei Tasten von Vorteil.