Enhanced Linked List Support in GNU-EFI

Finnbarr P. Murphy

(fpm@fpmurphy.com)

GNU-EFI enables you to develop (U)EFI applications for IA-64 (IPF), IA-32 (x86) and x86_64 platforms using the GNU GCC toolchain and the (U)EFI development environment.

The current support for double linked lists in GNU-EFI is old and not very useful. Moreover, it is based on a series of macros rather than actual functions. I needed additional functionality for a small EFI command line utility that I was writing and so I decided to try and improve double linked list support in GNU-EFI.

Here is the contents of the original header, i.e. */usr/include/efi/efilink.h*, for GNU-EFI v3.0t:

```
#ifndef _EFI_LINK_H
#define _EFI_LINK_H
/*++
Copyright (c) 1998  Intel Corporation
Module Name:
    link.h (renamed efilink.h to avoid conflicts)
Abstract:
    EFI link list macro's
Revision History
--*/
#ifndef EFI_NT_EMUL
//
// List entry - doubly linked list
//
typedef struct _LIST_ENTRY {
    struct _LIST_ENTRY  *Flink;
    struct _LIST_ENTRY  *Blink;
} LIST_ENTRY;
#endif
//
//  VOID
//  InitializeListHead(
//      LIST_ENTRY *ListHead
//      );
//
#define InitializeListHead(ListHead) \
    (ListHead)->Flink = ListHead;    \
    (ListHead)->Blink = ListHead;
//
//  BOOLEAN
//  IsListEmpty(
//      PLIST_ENTRY ListHead
//      );
//
#define IsListEmpty(ListHead) \
    ((ListHead)->Flink == (ListHead))
//
//  VOID
//  RemoveEntryList(
//      PLIST_ENTRY Entry
//      );
//
#define _RemoveEntryList(Entry) {       \
```



```
        LIST_ENTRY *_Blink, *_Flink;    \
        _Flink = (Entry)->Flink;        \
        _Blink = (Entry)->Blink;        \
        _Blink->Flink = _Flink;         \
        _Flink->Blink = _Blink;         \
        }
#if EFI_DEBUG
    #define RemoveEntryList(Entry)                      \
        _RemoveEntryList(Entry);                        \
        (Entry)->Flink = (LIST_ENTRY *) BAD_POINTER;    \
        (Entry)->Blink = (LIST_ENTRY *) BAD_POINTER;
#else
    #define RemoveEntryList(Entry)      \
        _RemoveEntryList(Entry);
#endif
//
//  VOID
//  InsertTailList(
//      PLIST_ENTRY ListHead,
//      PLIST_ENTRY Entry
//      );
//
#define InsertTailList(ListHead,Entry) {\
    LIST_ENTRY *_ListHead, *_Blink;     \
    _ListHead = (ListHead);             \
    _Blink = _ListHead->Blink;          \
    (Entry)->Flink = _ListHead;         \
    (Entry)->Blink = _Blink;            \
    _Blink->Flink = (Entry);            \
    _ListHead->Blink = (Entry);         \
    }
//
//  VOID
//  InsertHeadList(
//      PLIST_ENTRY ListHead,
//      PLIST_ENTRY Entry
//      );
//
#define InsertHeadList(ListHead,Entry) {\
    LIST_ENTRY *_ListHead, *_Flink;     \
    _ListHead = (ListHead);             \
    _Flink = _ListHead->Flink;          \
    (Entry)->Flink = _Flink;            \
    (Entry)->Blink = _ListHead;         \
    _Flink->Blink = (Entry);            \
    _ListHead->Flink = (Entry);         \
    }
//  VOID
//  SwapListEntries(
//      PLIST_ENTRY Entry1,
//      PLIST_ENTRY Entry2
//      );
//
// Put Entry2 before Entry1
//
#define SwapListEntries(Entry1,Entry2) {\
    LIST_ENTRY *Entry1Flink, *Entry1Blink;     \
    LIST_ENTRY *Entry2Flink, *Entry2Blink;     \
    Entry2Flink = (Entry2)->Flink;             \
    Entry2Blink = (Entry2)->Blink;             \
    Entry1Flink = (Entry1)->Flink;             \
    Entry1Blink = (Entry1)->Blink;             \
    Entry2Blink->Flink = Entry2Flink;       \
    Entry2Flink->Blink = Entry2Blink;        \
    (Entry2)->Flink = Entry1;               \
    (Entry2)->Blink = Entry1Blink;          \
    Entry1Blink->Flink = (Entry2);            \
    (Entry1)->Blink = (Entry2);             \
```



```
    }
//
//  EFI_FIELD_OFFSET - returns the byte offset to a field within a structure
//
#define EFI_FIELD_OFFSET(TYPE,Field) ((UINTN)(&amp;(((TYPE *) 0)->Field)))
//
//  CONTAINING_RECORD - returns a pointer to the structure
//      from one of it's elements.
//
#define _CR(Record, TYPE, Field)  \
    ((TYPE *) ( (CHAR8 *)(Record) - (CHAR8 *) &amp;(((TYPE *) 0)->Field)))
#if EFI_DEBUG
    #define CR(Record, TYPE, Field, Sig)     \
        _CR(Record, TYPE, Field)->Signature != Sig ?        \
            (TYPE *) ASSERT_STRUCT(_CR(Record, TYPE, Field), Record) : \
            _CR(Record, TYPE, Field)
#else
    #define CR(Record, TYPE, Field, Signature)   \
        _CR(Record, TYPE, Field)
#endif
//
// A lock structure
//
typedef struct _FLOCK {
    EFI_TPL     Tpl;
    EFI_TPL     OwnerTpl;
    UINTN       Lock;
} FLOCK;
#endif
```

The following code extends the functionality of the current support for linked lists and replaces all the macros with actual functions. It is based on the the EDK2 source code, specifically *.../src/edk2/MdePkg/Library/BaseLib/LinkedList.c*.

Here is the contents of the new version of */usr/include/efi/efilink.h*:

```


#ifndef _EFI_LINK_H
#define _EFI_LINK_H
#ifndef EFI_NT_EMUL
//
// List entry - doubly linked list
//
typedef struct _LIST_ENTRY {
    struct _LIST_ENTRY  *ForwardLink;
    struct _LIST_ENTRY  *BackLink;
} LIST_ENTRY;
#endif
LIST_ENTRY *
InitializeListHead ( LIST_ENTRY * );
LIST_ENTRY *
InsertHeadList ( LIST_ENTRY *ListHead, LIST_ENTRY *Entry);
LIST_ENTRY *
InsertTailList ( LIST_ENTRY *ListHead, LIST_ENTRY  *Entry);
LIST_ENTRY *
GetFirstNode ( LIST_ENTRY  *List);
LIST_ENTRY *
GetNextNode ( LIST_ENTRY *List, LIST_ENTRY *Node);
LIST_ENTRY *
GetPreviousNode ( LIST_ENTRY *List, LIST_ENTRY *Node);
BOOLEAN
IsListEmpty ( LIST_ENTRY *ListHead);
BOOLEAN
IsNull ( LIST_ENTRY  *List, LIST_ENTRY *Node);
BOOLEAN
```



```
IsNodeAtEnd ( LIST_ENTRY *List, LIST_ENTRY *Node);
LIST_ENTRY *
SwapListEntries ( LIST_ENTRY *FirstEntry, LIST_ENTRY  *SecondEntry);
LIST_ENTRY *
RemoveEntryList ( LIST_ENTRY *Entry);
//
//  EFI_FIELD_OFFSET - returns the byte offset to a field within a structure
//
#define EFI_FIELD_OFFSET(TYPE,Field) ((UINTN)(&amp;(((TYPE *) 0)->Field)))
//
//  CONTAINING_RECORD - returns a pointer to the structure
//      from one of it's elements.
//
#define _CR(Record, TYPE, Field)  \
    ((TYPE *) ( (CHAR8 *)(Record) - (CHAR8 *) &amp;(((TYPE *) 0)->Field)))
#if EFI_DEBUG
    #define CR(Record, TYPE, Field, Sig)     \
        _CR(Record, TYPE, Field)->Signature != Sig ?        \
            (TYPE *) ASSERT_STRUCT(_CR(Record, TYPE, Field), Record) : \
            _CR(Record, TYPE, Field)
#else
    #define CR(Record, TYPE, Field, Signature)   \
        _CR(Record, TYPE, Field)
#endif
//
// A lock structure
//
typedef struct _FLOCK {
    EFI_TPL     Tpl;
    EFI_TPL     OwnerTpl;
    UINTN       Lock;
} FLOCK;
#endif
```

The *FLOCK* typedef has nothing to do with the linked list code but it was in the original header so I just left it there. It really should be moved to some other GNU-EFI header.

And here is the corresponding source code file, *efilink.c,* which implements the double linked list functionality:

```


#include <efi.h>
#include <efilib.h>
LIST_ENTRY *
InitializeListHead (
   LIST_ENTRY               *ListHead
  )
{
  ListHead->ForwardLink = ListHead;
  ListHead->BackLink = ListHead;
  return ListHead;
}
LIST_ENTRY *
InsertHeadList (
  LIST_ENTRY               *ListHead,
  LIST_ENTRY               *Entry
  )
{
  Entry->ForwardLink = ListHead->ForwardLink;
  Entry->BackLink = ListHead;
  Entry->ForwardLink->BackLink = Entry;
  ListHead->ForwardLink = Entry;
  return ListHead;
}
LIST_ENTRY *
```



```
InsertTailList (
  LIST_ENTRY               *ListHead,
  LIST_ENTRY               *Entry
  )
{
  Entry->ForwardLink = ListHead;
  Entry->BackLink = ListHead->BackLink;
  Entry->BackLink->ForwardLink = Entry;
  ListHead->BackLink = Entry;
  return ListHead;
}
LIST_ENTRY *
GetFirstNode (
  LIST_ENTRY          *List
  )
{
  return List->ForwardLink;
}
LIST_ENTRY *
GetNextNode (
   LIST_ENTRY          *List,
   LIST_ENTRY          *Node
  )
{
  return Node->ForwardLink;
}
LIST_ENTRY *
GetPreviousNode (
   LIST_ENTRY          *List,
   LIST_ENTRY          *Node
  )
{
  return Node->BackLink;
}
BOOLEAN
IsListEmpty (
   LIST_ENTRY          *ListHead
  )
{
  return (BOOLEAN)(ListHead->ForwardLink == ListHead);
}
BOOLEAN
IsNull (
   LIST_ENTRY          *List,
   LIST_ENTRY          *Node
  )
{
  return (BOOLEAN)(Node == List);
}
BOOLEAN
IsNodeAtEnd (
  LIST_ENTRY          *List,
  LIST_ENTRY          *Node
  )
{
  return (BOOLEAN)(!IsNull (List, Node) &amp;&amp;&amp;&amp; List->BackLink == Node);
}
LIST_ENTRY *
SwapListEntries (
  LIST_ENTRY               *FirstEntry,
  LIST_ENTRY               *SecondEntry
  )
{
  LIST_ENTRY                   *Ptr;
  if (FirstEntry == SecondEntry) {
    return SecondEntry;
  }
  //
```



```
  // Ptr is the node pointed to by FirstEntry->ForwardLink
  //
  Ptr = RemoveEntryList (FirstEntry);
  //
  // If FirstEntry immediately follows SecondEntry, FirstEntry will be placed
  // immediately in front of SecondEntry
  //
  if (Ptr->BackLink == SecondEntry) {
    return InsertTailList (SecondEntry, FirstEntry);
  }
  //
  // Ptr == SecondEntry means SecondEntry immediately follows FirstEntry,
  // then there are no further steps necessary
  //
  if (Ptr == InsertHeadList (SecondEntry, FirstEntry)) {
    return Ptr;
  }
  //
  // Move SecondEntry to the front of Ptr
  //
  RemoveEntryList (SecondEntry);
  InsertTailList (Ptr, SecondEntry);
  return SecondEntry;
}
LIST_ENTRY *
RemoveEntryList (
  LIST_ENTRY          *Entry
  )
{
  Entry->ForwardLink->BackLink = Entry->BackLink;
  Entry->BackLink->ForwardLink = Entry->ForwardLink;
  return Entry->ForwardLink;
}
```

I maintained the coding style used in the EDK2 codebase.

In the following example, I use this new linked list functionality to maintain a sorted list of UEFI boot hot key variables. This utility first checks to see if boot hot keys are supported and, if they are, list out any hot keys that are present as global variables. See section 3.1.6 of the UEFI Specification v2.3.1 Errata C for further details.

```


//
//  Copyright (c) 2013  Finnbarr P. Murphy.   All rights reserved.
//
//  Examine BootOptionSupport global variable and list boot hotkeys if available
//        Note - does not list contents of hotkeys.
//
#include <efi.h>
#include <efilib.h>
// from TianoCore .../MdePkg/Include/Base.h
#define BASE_CR(Record, TYPE, Field)  ((TYPE *) ((CHAR8 *) (Record) - (CHAR8 *) &amp;(((TY
PE *) 0)->Field)))
#define VarBootOptionSupport      L"BootOptionSupport"
#define EFI_BOOT_OPTION_SUPPORT_KEY   0x00000001
#define EFI_BOOT_OPTION_SUPPORT_APP   0x00000002
#define EFI_BOOT_OPTION_SUPPORT_COUNT 0x00000300
#define EFI_SHELL_INTERFACE_GUID \
   (EFI_GUID) {0x47c7b223, 0xc42a, 0x11d2, {0x8e,0x57,0x00,0xa0,0xc9,0x69,0x72,0x3b}}
#define BUFSIZE 8
typedef struct {
    LIST_ENTRY Link;
    CHAR16     Name[BUFSIZE];
} HOTKEY_LIST_ENTRY;
STATIC HOTKEY_LIST_ENTRY HotKeyList;
```

```
typedef union {
    struct {
        UINT32 Revision : 8;
        UINT32 ShiftPressed : 1;
        UINT32 ControlPressed : 1;
        UINT32 AltPressed : 1;
        UINT32 LogoPressed : 1;
        UINT32 MenuPressed : 1;
        UINT32 SysReqPressed : 1;
        UINT32 Reserved : 16;
        UINT32 InputKeyCount : 2;
    } Options;
    UINT32 PackedValue;
} EFI_BOOT_KEY_DATA;
typedef struct _EFI_KEY_OPTION {
    EFI_BOOT_KEY_DATA KeyData;
    UINT32 BootOptionCrc;
    UINT16 BootOption;
} EFI_KEY_OPTION;
typedef enum {
    ARG_NO_ATTRIB         = 0x0,
    ARG_IS_QUOTED         = 0x1,
    ARG_PARTIALLY_QUOTED  = 0x2,
    ARG_FIRST_HALF_QUOTED = 0x4,
    ARG_FIRST_CHAR_IS_ESC = 0x8
} EFI_SHELL_ARG_INFO_TYPES;
struct _EFI_SHELL_ARG_INFO {
    UINT32 Attributes;
} __attribute__((packed)) __attribute__((aligned (1)));
typedef struct _EFI_SHELL_ARG_INFO EFI_SHELL_ARG_INFO;
struct _EFI_SHELL_INTERFACE {
    EFI_HANDLE           ImageHandle;
    EFI_LOADED_IMAGE    *Info;
    CHAR16             **Argv;
    UINTN                Argc;
    CHAR16             **RedirArgv;
    UINTN                RedirArgc;
    EFI_FILE            *StdIn;
    EFI_FILE            *StdOut;
    EFI_FILE            *StdErr;
    EFI_SHELL_ARG_INFO  *ArgInfo;
    BOOLEAN              EchoOn;
} __attribute__((packed)) __attribute__((aligned (1)));
typedef struct _EFI_SHELL_INTERFACE EFI_SHELL_INTERFACE;
static EFI_STATUS
get_args(EFI_HANDLE image, UINTN *argc, CHAR16 ***argv)
{
    EFI_STATUS Status;
    EFI_SHELL_INTERFACE *shell;
    EFI_GUID gEfiShellInterfaceGuid = EFI_SHELL_INTERFACE_GUID;
    Status = uefi_call_wrapper(BS->OpenProtocol, 6,
                               image, &amp;gEfiShellInterfaceGuid,
                               (VOID **)&amp;shell, image, NULL,
                               EFI_OPEN_PROTOCOL_GET_PROTOCOL);
    if (EFI_ERROR(Status))
        return Status;
    *argc = shell->Argc;
    *argv = shell->Argv;
    Status = uefi_call_wrapper(BS->CloseProtocol, 4, image,
                               &amp;gEfiShellInterfaceGuid, image, NULL);
    return Status;
}
BOOLEAN
IsHotKeyVariable (CHAR16 *Name, EFI_GUID *Guid)
{
    EFI_GUID GlobalVariableGuid = EFI_GLOBAL_VARIABLE;
    if (CompareGuid (Guid, &amp;GlobalVariableGuid) ||
        (StrSize (Name) != sizeof (L"Key####")) ||
```



```
        (StrnCmp (Name, L"Key", 3) != 0)) {
        return FALSE;
    }
    return TRUE;
}
INT16
CompareOptionNumbers(CHAR16 *Name1, CHAR16 *Name2)
{
    UINT16  OptionNumber1 = 0;
    UINT16  OptionNumber2 = 0;
    UINTN   Index;
    for (Index = 3; Index < 7; Index++) {
       if ((Name1[Index] >= L'0') &amp;&amp; (Name1[Index] <= L'9')) {
           OptionNumber1 = OptionNumber1 * 10 + Name1[Index] - L'0';
       } else if ((Name1[Index] >= L'A') &amp;&amp; (Name1[Index] <= L'F')) {
           OptionNumber1 = OptionNumber1 * 10 + Name1[Index] - L'A';
       }
    }
    for (Index = 3; Index < 7; Index++) {
       if ((Name2[Index] >= L'0') &amp;&amp; (Name2[Index] <= L'9')) {
           OptionNumber2 = OptionNumber2 * 10 + Name2[Index] - L'0';
       } else if ((Name2[Index] >= L'A') &amp;&amp; (Name2[Index] <= L'F')) {
           OptionNumber2 = OptionNumber2 * 10 + Name2[Index] - L'A';
       }
    }
    if (OptionNumber1 > OptionNumber2)
        return 1;
    if (OptionNumber1 < OptionNumber2)
        return -1;
    return 0;
}
static void
Usage(void)
{
    Print(L"Usage: showhotkeys [-v | --verbose] [-n | --nosort ]\n");
}
EFI_STATUS
efi_main (EFI_HANDLE image, EFI_SYSTEM_TABLE *systab)
{
    EFI_STATUS Status = EFI_SUCCESS;
    EFI_GUID Guid = EFI_GLOBAL_VARIABLE;
    EFI_GUID curGuid= NullGuid;
    UINTN Attr = EFI_VARIABLE_NON_VOLATILE | EFI_VARIABLE_BOOTSERVICE_ACCESS | EFI_VARIABL
E_RUNTIME_ACCESS;
    UINTN DataValue = 0;
    UINTN DataSize = 4;
    UINTN argc;
    UINTN Size, CurSize;
    CHAR16 **argv;
    CHAR16 *Name, *val;
    HOTKEY_LIST_ENTRY *HotKeyVar;
    HOTKEY_LIST_ENTRY *Node, *PrevNode;
    int Result, Verbose=0, NoSort=0;
    InitializeLib(image, systab);
    Status = get_args(image, &amp;argc, &amp;argv);
    if (EFI_ERROR(Status)) {
        Print(L"ERROR: Parsing command line arguments: %d\n", Status);
        return Status;
    }
    if (argc == 2) {
        if (!StrCmp(argv[1], L"--help") ||
            !StrCmp(argv[1], L"-h") ||
            !StrCmp(argv[1], L"-?")) {
            Usage();
            return Status;
        } else if (!StrCmp(argv[1], L"--verbose") ||
            !StrCmp(argv[1], L"-v")) {
                Verbose=1;
```



```
        } else if (!StrCmp(argv[1], L"--nosort") ||
            !StrCmp(argv[1], L"-n")) {
                NoSort=1;
        }
    }
    if (argc == 3) {
        if (!StrCmp(argv[1], L"--verbose") ||
            !StrCmp(argv[1], L"-v")        ||
            !StrCmp(argv[2], L"--verbose") ||
            !StrCmp(argv[2], L"-v")) {
                Verbose=1;
        }
        if (!StrCmp(argv[1], L"--nosort") ||
            !StrCmp(argv[1], L"-n")       ||
            !StrCmp(argv[2], L"--nosort") ||
            !StrCmp(argv[2], L"-n")) {
                NoSort=1;
        }
    }
    Status = uefi_call_wrapper(RT->GetVariable, 5,
                  VarBootOptionSupport, &amp;Guid, Attr, &amp;DataSize, &amp;DataValue);
    if (Status != EFI_SUCCESS)  {
         if (Status != EFI_NOT_FOUND)
             Print(L"ERROR: GetVariable failed: %d\n", Status);
         return Status;
    } else {
         // Print(L"%04x\n", DataValue);
         if (DataValue &amp;&amp; EFI_BOOT_OPTION_SUPPORT_KEY) {
            if (Verbose == 1)
                Print(L"Boot manager supports application launching using keys\n");
         } else {
            if (Verbose == 1)
                Print(L"Boot manager does not support application launching using keys\n")
;
            return Status;
         }
    }
    InitializeListHead(&amp;HotKeyList.Link);
    CurSize = BUFSIZE;
    Name = AllocateZeroPool(CurSize);
    // loop through all NVRAM variables and find any hotkeys
    while (TRUE) {
        Size = CurSize;
        Status = uefi_call_wrapper(RT->GetNextVariableName, 3, &amp;Size, Name, &amp;curGu
id);
        if (Status ==  EFI_NOT_FOUND)
            break;
        if (Status == EFI_BUFFER_TOO_SMALL) {
            Name = ReallocatePool(Name, CurSize, Size);
            CurSize = Size;
            Status = uefi_call_wrapper(RT->GetNextVariableName, 3, &amp;Size, Name, &amp;c
urGuid);
        }
        if (Status != EFI_SUCCESS) {
            Print(L"ERROR: GetNextVariableName failed: %d\n", Status);
            return Status;
        }
        if (IsHotKeyVariable (Name, &amp;curGuid)) {
            HotKeyVar = AllocateZeroPool(sizeof(HOTKEY_LIST_ENTRY));
            if (HotKeyVar == NULL) {
                Print(L"ERROR: Out of memory resources\n");
                return EFI_OUT_OF_RESOURCES;
            }
            StrCpy(HotKeyVar->Name, Name);
            // place new hotkey entry into correct position in sorted list if sort enables
            InsertHeadList(&amp;HotKeyList.Link, &amp;HotKeyVar->Link);
            if (NoSort == 0) {
                for (Node = (HOTKEY_LIST_ENTRY *)GetFirstNode (&amp;HotKeyList.Link),
```



```
                    PrevNode = (HOTKEY_LIST_ENTRY *)GetFirstNode (&amp;HotKeyList.Link);
                    !IsNull (&amp;HotKeyList.Link, &amp;Node->Link);
                    Node = (HOTKEY_LIST_ENTRY *)GetNextNode (&amp;HotKeyList.Link, &amp;N
ode->Link))
                {
                    Result = CompareOptionNumbers(PrevNode->Name, Node->Name);
                    if (Result > 0) {
                        Node = (HOTKEY_LIST_ENTRY *) SwapListEntries (&amp;PrevNode->Link,
 &amp;Node->Link);
                    } else if (Result < 0) {
                        break;
                    }
                }
            }
        }
    }
    FreePool(Name);
    // output the (sorted) list of hotkeys
    while (!IsListEmpty(&amp;HotKeyList.Link)) {
        Node = (HOTKEY_LIST_ENTRY *)GetFirstNode(&amp;HotKeyList.Link);
        HotKeyVar = BASE_CR(Node, HOTKEY_LIST_ENTRY, Link);
        Print(L"%s\n", HotKeyVar->Name);
        RemoveEntryList(&amp;Node->Link);
        FreePool(Node);
    }
    return Status;
}
```

You can find all the source code, including a Makefile for the *showhotkeys* utility on GitHub.

I plan to work with the GNU-EFI maintainer, Nigel Croxon, to get this enhanced functionality into the next release of GNU-EFI.

Enjoy!

