Operating Systems Assignment 4 - Linux Kernel Modules

August 30, 2015

Kernel Data Structures

Goal:

To modify the kernel module so as to make use of kernel linked-list data structure. To create a linked list containing five *struct birthday* elements. Traverse the linked list and output its contents to the kernel log buffer. In the module exit point, delete the elements from the linked list and return the free memory back to the kernel.

Implementation:

OS Version: Ubuntu 14.04

Linux Kernel version: 3.13.0-62-generic

The linux kernel provides several data structures. To implement the above mentioned task, I use a circular, doubly linked list. It is available in *<linux/list.h>*.

- Create a structure birthday containing elements day, month and year.

  struct birthday {

  int day;

  int month;

  int year;

  struct list_head list;

  }

  list_head structure embeds the linked list within the nodes that comprise the list. The *list_head* structure holds two members *next* and *prev*, these point to next and previous elements of the list. This data structure can now be managed with a series of *macro* functions.

Insert elements into the linked list:

- Firstly declare a list_head object, which can be used as a reference to the head of the list by using *LIST_HEAD()* macro.

  static LIST_HEAD(birthday_list);

This macro defines and initializes the variable birthday_list, which is of type *struct list_head.*

- Now, create and initialize instances of the *struct birthday* as follows:

  struct birthday *person;

  person = *kmalloc*(sizeof(*person), *GFP_KERNEL*);

Assign day, month and year of person to some value (Eg: person->day = 2). Here *kmalloc()* is used for allocating kernel memory. *GFP_KERNEL* flag indicates routine kernel memory allocation.

- Initialize the head of the linked list by *INIT_LIST_HEAD*(&person->list);

Add this element to the end of the linked list by :

list_add_tail(&person->list, &birthday_list);

- The other four elements of the linked list can be created exactly as above and include these elements to the end of the linked list by using *list_add_tail()*.

```
person = kmalloc(sizeof(*person), GFP_KERNEL);
person->day = dy;
person->month = mnth;
person->year = yr;
list_add_tail(&person->list, &birthday_list);
dy++;
mnth++;
yr++;
```

Traverse the linked list:

The linked list can be traversed by using list_for_each_entry() Macro which accepts three parameters:

- A pointer to the birthday structure.
- A pointer to the head of the list (ie: *birthday_list*).
- The name of the variable containing the list_head structure (ie *list* in our case).

list_for_each_entry(ptr, &birthday_list, list);

Remove elements from the linked list:

To remove all the elements from the linked list, remove each individual element as the list is being traversed. This can be done by using *list_for_each_entry_safe()* which has same functionality as *list_for_each_entry()*, additionally it has an argument which maintains the value of the next pointer of the item being deleted. *list_del*(struct list_head **element*) can be used for removing *element* from the list. This also maintains the structure of the reminder of the list.

kfree() is used to give back the kernel memory which was allocated using *kmalloc()*. Releasing allocated memory when it is no longer needed is essential to prevent memory leaks.

```
list_for_each_entry_safe(ptr, next, &birthday_list, list) {
list_del(&ptr->list); //deleting the element
kfree(ptr); //free the memory allocated to it
}
```

How to run?

- Create a Makefile to run this program. Run by giving the command make.

```
obj-m := linked.o
KVERSION := $(shell uname -r)
KDIR := /lib/modules/$(KVERSION)/build
PWD := $(shell pwd)
default:
$(MAKE) -C $(KDIR) M=$(PWD) modules
clean:
$(MAKE) -C $(KDIR) M=$(PWD) clean
install:
$(MAKE) -C $(KDIR) M=$(PWD) modules_install
```

- Once it is succesfully run, it will generate linked.ko, linked.mod.c, modules.order, Module.symvers etc.
- To load or insert the module => insmod linked.ko
- To list all the modules, give lsmod
- To unload or remove the module => rmmod linked.ko
- dmesg will display all the messages which we have written.

List of modules after loading the module linked

```
akilesh@akilesh-ubuntu:~/Desktop/OSasn4/Linked$ sudo insmod linked.ko
akilesh@akilesh-ubuntu:~/Desktop/OSasn4/Linked$ lsmod
Module                  Size  Used by
linked                 12655  0
vmnet                  51316  13
vmw_vsock_vmci_transport    26278  0
vsock                  34903  1 vmw_vsock_vmci_transport
vmw_vmci               62966  1 vmw_vsock_vmci_transport
vmmon                  84056  0
ctr                    13049  2
ccm                    17773  2
bnep                   19624  2
rfcomm                 69160  0
snd_hda_codec_hdmi     46368  1
snd_hda_codec_realtek    65812  1
uvcvideo               80885  0
videobuf2_vmalloc      13216  1 uvcvideo
videobuf2_memops       13362  1 videobuf2_vmalloc
videobuf2_core         40664  1 uvcvideo
videodev              134688  2 uvcvideo,videobuf2_core
samsung_laptop         14486  0
intel_rapl             18773  0
x86_pkg_temp_thermal    14205  0
intel_powerclamp       14705  0
coretemp               13435  0
snd_hda_intel          56531  3
kvm_intel             143187  0
snd_hda_codec         193017  3 snd_hda_codec_realtek,snd_hda_codec_hdmi,snd_hda_intel
kvm                   455843  1 kvm_intel
snd_hwdep              13602  1 snd_hda_codec
snd_pcm               102099  3 snd_hda_codec_hdmi,snd_hda_codec,snd_hda_intel
crct10dif_pclmul       14289  0
crc32_pclmul           13113  0
ghash_clmulni_intel    13216  0
snd_page_alloc         18710  2 snd_pcm,snd_hda_intel
cryptd                 20359  1 ghash_clmulni_intel
arc4                   12608  2
snd_seq_midi           13324  0
snd_seq_midi_event     14899  1 snd_seq_midi
ath9k                 164164  0
```

Screen shot of kernel log