Ryan Mansfield

10/5/2016

Csc139

Professor Dais

Deliverable 1: Assignment 1

1. Purpose

The purpose of the assignment is to learn how to create kernel modules using the loadable kernel module structure. The assignment walked through the creation, the running and the removal of modules in kernel space. The secondary purpose of the assignment is to learn how to use kernel development APIs by implementing a linked list using kernel libraries

2) Theory

The loadable kernel module is a common operating system implementation. This allows for the addition of kernel code to an operating system without the need for compiling the entire operating system. The design uses principles from object orientation to divide the kernel into module structures that can be added or removed very simply. The Debian 8 operating system provided allows for such modules to exist.

The linked list is a data structure with the keynote feature being the ability to add and removed nodes. The list used in the Linux kernel is circular, and each element of a list is referred to as a list head. The API <linux/list.h> allows for list functions to easy create, add to, traverse, and delete from lists.

The aforementioned concepts will be utilized in the following problem statement. The goal is to create 4 entries into a linked list and print those to the kernel message system. Then the entries are to be sorted using the <linux/list\_sort.h> library with the sorted result printed to the kernel console. The final step is to remove the list and deallocate the memory.

3) Implementation

The following objectives were implemented in C, since that is the language the libraries are written in. The following code is the source code for the kernel module code.

#include <linux/init.h>

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/list.h>

#include <linux/list\_sort.h>

#include <linux/slab.h>

static LIST\_HEAD(birthday\_list); //this will create a static list head variable

int compare(void \*priv, struct list\_head \*a, struct list\_head \*b);

//prototype for compare function as defined in list\_sort header

void addlist(char \*name, int month, int day, int year );

//function that adds elements to the list

void traverse(void);

//traverse function to print results

void sortlist(void \*priv );

//prototype for the sort function

struct birthday

{

char \*name;

int month;

int day;

int year;

struct list\_head list; //creates a list head

};

/\* This function is called when the module is loaded. \*/

//this is where the init code will go for the list

int simple\_init(void)

{

printk(KERN\_INFO "Loading Module\n");

addlist("Alice",1,9,1999);

addlist("Bob",3,9,1978);

addlist("Mallory",12,9,1958);

addlist("Nancy",6,9,2003);

addlist("Kate",3,8,1978);

traverse();

printk(KERN\_INFO "This is the presorted list\n");

sortlist(NULL);

traverse();

printk(KERN\_INFO "This is the sorted list\n");

return 0;

}

int compare( void \*priv, struct list\_head \*a, struct list\_head \*b)

{

if( a != NULL && b != NULL ) //checks to see if the pointers are null

{

struct birthday \*tempa = container\_of(a, struct birthday, list);

/\*the above function takes the list head pointer a and creates a birthday struct around the list pointer and points a at it. \*/

struct birthday \*tempb = container\_of(b, struct birthday, list);

int montha = tempa->month;

int monthb = tempb->month;

int daya = tempa->day;

int dayb = tempb->day;

int yeara = tempa->year;

int yearb = tempb->year;

//setting stack variables so heap isnt perpetually accessed

if(yeara < yearb) //year comparison

{

//return a -1 to sort list saying a should go first

return -1;

}

else if(yeara > yearb)

{

//return a 1 to sort list saying b should go first

return 1;

}

else //they are equivalent

{ //month comparison

if(montha < monthb)

{

return -1;

}

else if( montha > monthb)

{

return 1;

}

else

{ //day comparison

if(daya < dayb)

{

return -1;

}

else if(daya > dayb)

{

return 1;

}

else //the equivolency case

{

return 0;

}

}

}

}

else

{

printk(KERN\_INFO "Error: one of the pointers is null");

return 0; //error code for null pointer reference

}

}

void addlist( char \*name, int month, int day, int year)

/\*function that adds to li

st\*/

{

struct birthday \*tmp ;//creating a temp birthday struct

tmp = kmalloc(sizeof(\*tmp), GFP\_KERNEL);//dynamically allocates mem

if(name)

{

tmp->name = name;

tmp->month = month;

tmp->day = day;

tmp->year = year;

//thats it for the basic data

//now to define the list head field and add a tail

INIT\_LIST\_HEAD(&tmp->list); //init head at address of list

list\_add\_tail(&tmp->list,&birthday\_list);

//adds tail at new list head in respect to old list head

}

else

{

printk(KERN\_INFO "There was no name provided. Element was not added");

}

}

void traverse(void)//traverse the list

{

struct birthday \*ptr;

list\_for\_each\_entry(ptr, &birthday\_list, list)

{

printk(KERN\_INFO "NAME = %s\n", ptr->name);

printk(KERN\_INFO "MONTH = %d\n", ptr->month);

printk(KERN\_INFO "DAY = %d\n", ptr->day);

printk(KERN\_INFO "YEAR = %d\n", ptr->year);

printk(KERN\_INFO "====================================\n");

}

}

void sortlist(void \*priv)

{

list\_sort(priv, &birthday\_list, compare); //calls the sort list

}

/\* This function is called when the module is removed. \*/

void simple\_exit(void)

{

printk(KERN\_INFO "Removing Module\n");

struct birthday \*point,\*next;

list\_for\_each\_entry\_safe(point,next,&birthday\_list,list)

{

list\_del(&point->list);//deletes the element off of the list

kfree(point); //frees the kmalloc space

}

printk(KERN\_INFO "the list has been cleared and the space is free\n");

}

/\* Macros for registering module entry and exit points. \*/

module\_init( simple\_init );

module\_exit( simple\_exit );

MODULE\_LICENSE("GPL");

MODULE\_DESCRIPTION("Simple Module");

MODULE\_AUTHOR("SGG");

The goal of this implementation was to modularize and add security. Since this is a kernel file, a makefile is required. The source for the make file is show below.

//MakeFILE code

obj-m += simple.o

all:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

4) Results

There were some difficulties in obtaining results. There was a warning the make command made about comparing null, and although it was a warning, it would keep the module hanging when the addlist() function was called. The problem was that when the kernel module hangs, there is no good way to exit out of it. The only safe choice was to reset the VM. This happened a few times until the comparison was rectified upon closer inspection.

Another issue encountered was including the wrong headers. Since this is a kernel file, there cannot be any user libraries. However, I made the mistake of trying to use <stdlib> for the exit() function. The same issue, but reversed, happened early in the process when I attempted to use gcc to compile the kernel file. The gcc doesn’t have access to those files, so it could not find the header files for the Linux kernel API.

The screenshots are posted below in Figures 1.1,1.2, and 1.3.

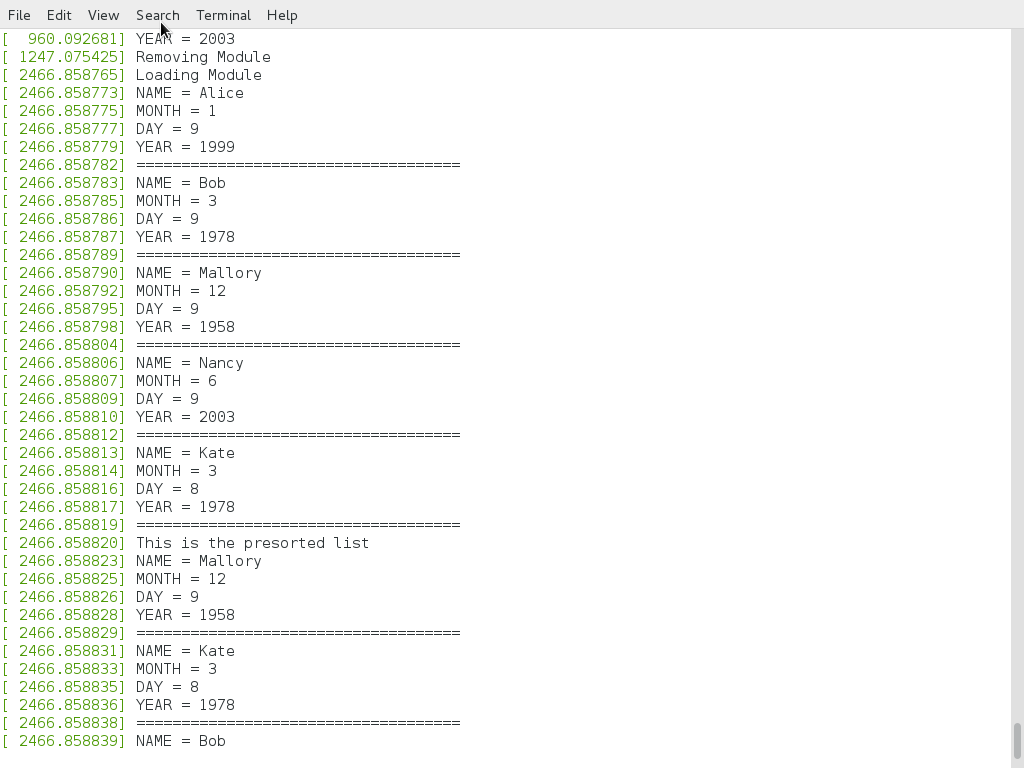


FIGURE 1.1: TRAVERSING THE LIST

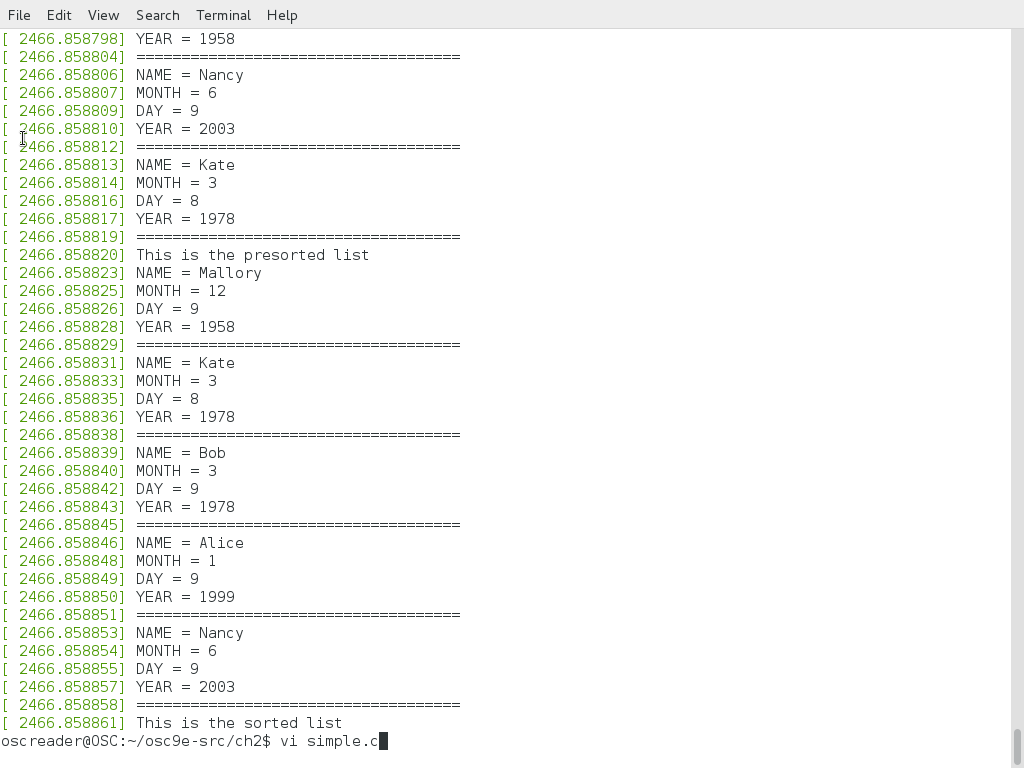


FIGURE 1.2: TRAVERSING THE SORTED LIST

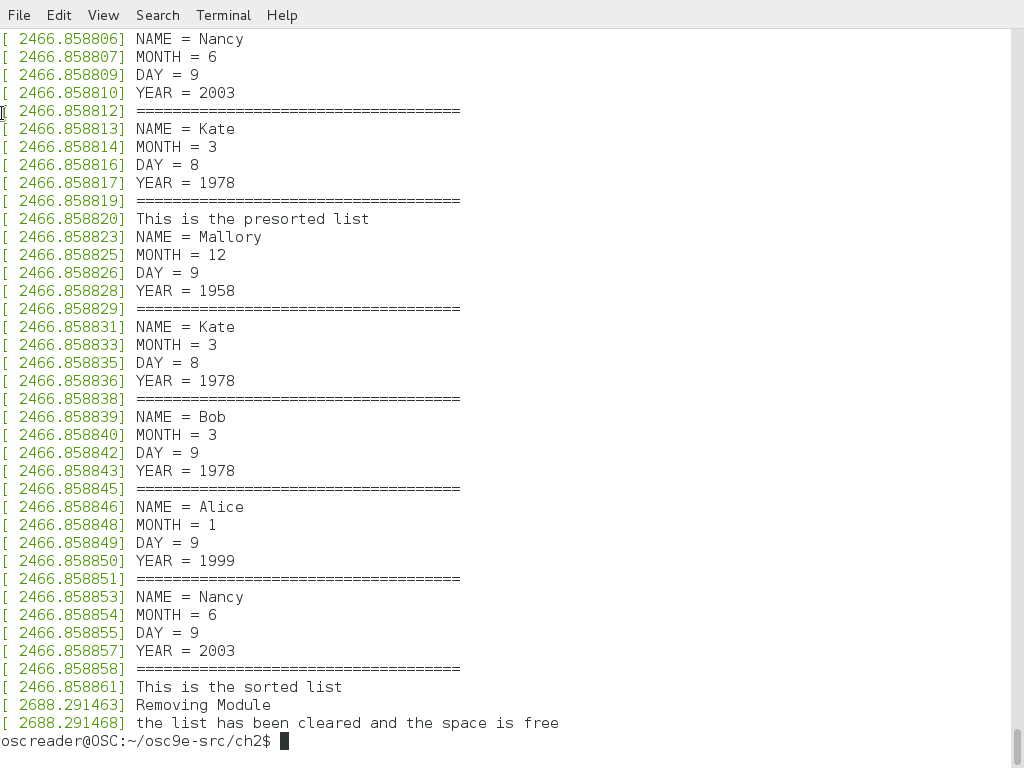


FIGURE 1.3: REMOVING THE MODULE