Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Laboratorio de Sistemas Operativos 1

Módulos de Kérnel Linux

Integrantes:
Randolph Estuardo Muy – 201314112
Sergio De Los Rios – 201213282

Módulos

Para la creación de módulos es necesario tener en cuenta el ambiente de trabajo:

Sistema operativo : Ubuntu 18.04
Lenguaje de programación : C
Arquitectura del sistemas Operativo: 64 bits

Modulo de Memoria (SysInfo)

La finalidad de la creación de este modulo es mostrar información de la memoria Ram de nuestro Ordenador, ya que es importante llevar el control sobre la memoria consumida actualmente por nuestro ordenador, así como mostrar datos mas atendibles como su porcentaje de uso.

A continuación se muestran una serie de pasos para la creación del modulo.

Paso1:

-Importar librerías necesarias
-hay que tener en cuenta que la librería que brinda información de la memoria Ram es: **<sys/sysinfo.h>** pero hay ocasiones en que el lenguaje C da error en la importación de dicha librería. Es por eso que se puede hacer utilización de la información sin necesidad de importarla

```
#include <linux/module.h>    /* Todos los modulos lo necesitan */
#include <linux/kernel.h>    /* Ofrece la macro KERN_INFO */
#include <linux/init.h>      /* Ofrece las macros de inicio y fin */
#include <linux/seq_file.h>  //Esta libreria funciona para el seq_file el cual escribe y lee
#include <linux/hugetlb.h>   //si_meminfo
#include <linux/proc_fs.h>   //proc_create
#include <linux/sys.h>
```

Paso 2:

Se necesita la creación de 2 métodos los cuales son inicio y salida los cuales son ejecutados con **module_init** y **module_exit** respectivamente

Inicio: es la creación del archivo **memo_201314112_201213282** en la carpeta **/proc**, ademas cuando se hace consulta de los módulos cargados con **dmesg** puede mostrarse información que brindan los módulos por lo que se mostrara la información de los estudiantes.

Salida: este método se encarga de realizar una acción cuando un modulo fue descargado. Por lo que en esta ocasión se imprimirá un mensaje de alerta con el nombre: “Sistemas Operativos 1”.

```
static int __init inicio(void)
{
        proc_create("memo_201314112_201213282", 0, NULL, &fops); //creacion del archivo
        printk(KERN_ALERT "Carnets: 201314112_201213282\n");
        return 0;
}

static void __exit salida(void)
{
        printk(KERN_ALERT "Sistemas Operativos 1\n");
}


// se Indica cuales son las funciones de inicio y fin
module_init(inicio);
module_exit(salida);
```

Paso3:

es necesario mostrar documentacion del modulo creado por lo que estos metodos se encargan de documentar al nuevo modulo

```
// Documentacion del modulo
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Modulo de Pruebas SOPES1");
MODULE_AUTHOR("Randolph_Muy-201404243___Sergio_De_los_Ríos-201213282");
```

Paso 4:

En este paso se realiza toda la lógica de obtener información de la memoria Ram ademas de escribirlo físicamente en el archivo el cual es el nuevo modulo creado.
Tener en cuenta la utilización de **sysinfo** el cual se hablo al principio de este modulo, por lo que no fue necesaria su importación, ademas se maneja con un struct con identificador **si.**
El método **write_and_read** se encarga de escribir en el archivo.
El método que se encarga de parametrizar que archivo lee y escribe es **fops** es cual es un struct de tipo **file_operations** es cual es encargado de buscar el la información que se le pasen a sus atributos **.open** y **.read** .

```
#define BUFSIZE 150
struct sysinfo si;

static int write_and_read(struct seq_file *s,void *v)
{
	si_meminfo(&si);
	int32_t total_memoria,memoria_libre;
	total_memoria = (si.totalram * si.mem_unit)/(1024 * 1024);
	memoria_libre = (si.freeram * si.mem_unit)/(1024*1024);
	seq_printf(s, " ------- Info Est1 ----- \n");
	seq_printf(s, " Nombre: Randolph Muy\n");
	seq_printf(s, " Carnet: 201314112\n");
	seq_printf(s, " ------- Info Est2------ \n");
	seq_printf(s, " Nombre: Sergio De los Ríos\n");
	seq_printf(s, " Carnet: 201213282\n");
	seq_printf(s, " ---------------------- \n");
	seq_printf(s, " Memoria Total : \t %d MB\n",total_memoria);
	seq_printf(s, " Memoria Libre : \t %d MB \n",memoria_libre);
	seq_printf(s, " Memoria en uso: \t %i %%\n", (memoria_libre * 100)/total_memoria) ;
	return 0;
}

static int open_file(struct inode *inode, struct file *file) {
	return single_open(file, write_and_read, NULL);
}

static struct file_operations fops =
{
	.open = open_file,
	.read = seq_read
};
```

Modulo de CPU (task struct)

Este modulo es el encarga de mostrar información sobre el procesos del sistema operativo.

Paso 1:

Importar las librerías a utilizar.

```
#include <linux/init.h>         // Incluido para __init y __exit macros
#include <linux/module.h>    // Incluido para todos los modulos de kernel
#include <linux/kernel.h>    // Incluido para KERN_INFO
#include <linux/sched.h>
#include <linux/proc_fs.h>
#include <linux/seq_file.h>
#include <asm/uaccess.h>
#include <linux/hugetlb.h>
#include <linux/fs.h>
#include <linux/sched/signal.h>
```

Paso 2:

Se crear la información del modulo, así como los structs que servirán para obtener la información de los procesos. El struct task_list nos servirá para listar los procesos padre y el task_child lo utilizaremos para guardar los procesos hijos, y el list_head para poder identificar los procesos hijos del proceso padre.

```
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Escribir informaciond del cpu");
MODULE_AUTHOR("Sergio De los Rios");
    struct task_struct *task_list;
    struct task_struct *task_child;
    struct list_head *list;
```

Paso 3:

Se crean los métodos que irán al cargar y descargar el modulo y se especifica cuales serán. Al cargar el modulo se utiliza printk para mostrar en la terminal la información de los estudiantes y al descargar el nombre del curso. Y al cargar el modulo se tiene que crear o sobrescribir un archivo con los nombres y carnets de los estudiantes así como los procesos del cpu.

```
static int testmodulo_init(void)
{
    proc_create("cpu_201213282_201314112", 0, NULL, &fcpu);
    // unsigned int process_count = 0;
    // printk(KERN_INFO "---Inicio-----\n");
    // for_each_process(task_list) {
    //      printk(KERN_INFO "PID: %d\tPROCESS: %s\tSTATE: %ld\n",task_list->pid, task_list->comm, task_list->state);
    //      process_count++;
    //      list_for_each(list, &task_list->children){
    //          task_child = list_entry( list, struct task_struct, sibling );
    //          printk(KERN_INFO "hijo:\tPID: %d\tPROCESS: %s\tSTATE: %ld\n",task_child->pid,task_child->comm, task_child->state);
    //     }
    //     printk("-----------------------------------------------------");    /*for aesthetics*/
    // }
    // pr_info("Number of processes:%u\n", process_count);
    printk(KERN_INFO "Sergio De los Rios-201213282\nRandolph Muy-201314112\n");//numero de carnet
 //Aqui iria el codigo a ejecutar
   return 0;    // Si el retorno no es 0
                //quiere decir que el modulo no se ha podido cargar
}

static void testmodulo_cleanup(void)
{
   printk(KERN_INFO "Sistemas Operativos 1\n"); //se loga en el /var/log/messages
}
module_init(testmodulo_init);
module_exit(testmodulo_cleanup);
```

Paso 4:

Escribimos en el archivo la información de los estudiantes seguido de la información que nos importa de los procesos, en este caso su id, su nombre, su estado y después los hijos si es que este proceso los tiene.

```
static int escritura_archivo(struct seq_file *s,void *v)
{
    seq_printf(s, "  Nombre: Sergio De los Ríos.\n");
    seq_printf(s, "  Carnet: 201213282.\n");
    seq_printf(s, "  Nombre: Randolph Muy.\n");
    seq_printf(s, "  Carnet: 201314112.\n");
    unsigned int process_count = 0;
    seq_printf(s, "---Inicio-----\n");
    for_each_process(task_list) {
         seq_printf(s, "PID: %d\tPROCESS: %s\tSTATE: %ld\n",task_list->pid, task_list->comm, task_list->state);
         process_count++;
         list_for_each(list, &task_list->children){
                task_child = list_entry( list, struct task_struct, sibling );
                seq_printf(s, "hijo:\tPID: %d\tPROCESS: %s\tSTATE: %ld\n",task_child->pid,task_child->comm, task_child->state);
        }
        seq_printf(s,"-------------------------------------------------");    /*for aesthetics*/
    }
    pr_info("Number of processes:%u\n", process_count);
    return 0;
}
static int open_file(struct inode *inode, struct  file *file) {
  return single_open(file, escritura_archivo, NULL);
}

static struct file_operations fcpu =
{
    .open = open_file,
    .read = seq_read
};
```

Como ejecutar y crear el modulo

Para la creación del modulo en la carpeta /proc se listan una serie de pasos:

Paso 1:
crear una carpeta con el archivo con extensión .c el cual contendrá la información que nos interesa del modulo.
Se recomienda también que la carpeta contenga un archivo con el nombre **Makefile** el cual nos servirá como script para hacer que la compilacion de los comandos para que sea mas simple. Ademas de abrir una consola y se ubique el directorio en donde estos archivos están creados.

A continuación se muestra la información dentro del archivo **Makefile**:

este comando debe contener el nombre del archivo (modulo) con extensión .o
obj-m += memo_201314112_201213282.o

- Este comando se ejecutara cuando en consola ejecutemos **make all** , lo cual crea una serie de archivos importantes que nos servirán para la carga y descarga del modulo.
all:
make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

- Este comando eliminara todos los archivos creados por **make all** por lo que en consola se debe escribir: **make clean.**

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

Paso 2:

cuando se ejecute **make all** y los archivos sean crados el se podra notar que en el directorio se crearon nuevos archivos, y el que nos interesa el el archivo que termina con extesion .**ko**

memo_201314112_201213282.ko

Este archivo debemos hacer la carga con el siguiente comando (se necesitan permisos de usuario **root**):

sudo insmod cpu_201213282_201314112.ko

Con la ejecución de este comando se carga el modulo al kernel si puede ser observado con el coamndo **dmesg** el cual nos mostrar información del modulo, ademas si se listan los módulos con **lsmod** se podrá observar el nombre del modulo listado.

Paso 3:

En este paso es observar la información del modulo por lo que en la consola de linux necesitamos ubicarnos en la carpeta **/proc** por lo que si listamos su contenido podremos observar el nombre del nuevo modulo, por lo que ahora para poder observar su contenido basta con ejecutar **cat cpu_201213282_201314112** lo cual mostrara información programada dentro del modulo

Este paso trata de descargar o eliminar el modulo de los modulos activos. Etonces para este paso se ejecuta el siguiente comando (con permisos de usuario root):

sudo *rmmod* cpu_201213282_201314112

Este comando necesita el nombre del modulo para poder ser descargado.

Ahora para limpiar todos los archivos creados en la carpeta basta con ejecutar el comando ***make clean*** y automáticamente eliminara todos los archivos que fueron necesarios para la compilación del modulo.