

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



FACULTAD DE INGENIERÍA

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA


Diego Antonio Momotic Montesdeoca
2013-18633

Guatemala, diciembre de 2020

Manual creación de módulos

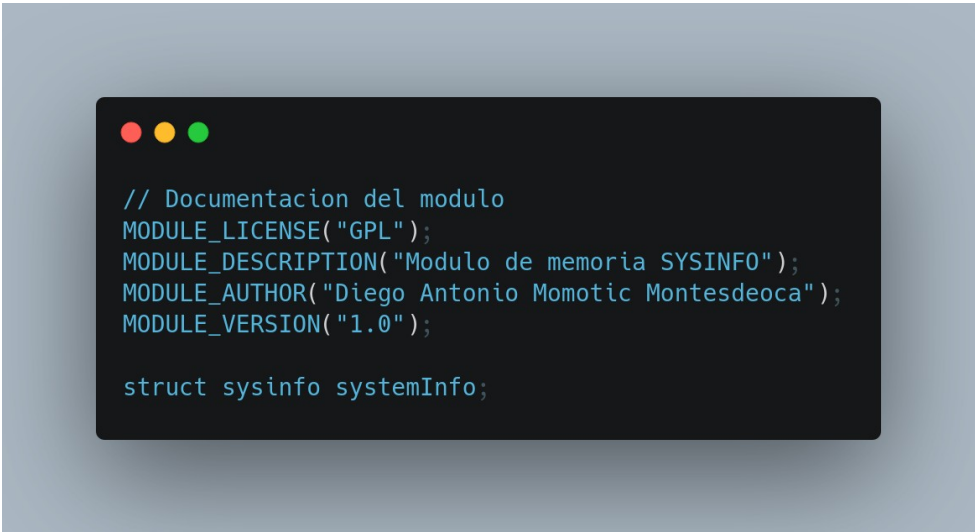
Modulo RAM

- 1) Importamos librerías que serán utilizadas para obtener la información de la memoria RAM.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains a list of kernel headers to be included in the module.

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/seq_file.h>
#include <linux/hugetlb.h>
#include <linux/proc_fs.h>
#include <linux/sys.h>
```

- 2) Documentamos modulo y hacemos uso del struct sysinfo.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It contains module documentation macros and a struct definition.

```
// Documentacion del modulo
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Modulo de memoria SYSINFO");
MODULE_AUTHOR("Diego Antonio Momotic Montesdeoca");
MODULE_VERSION("1.0");

struct sysinfo systemInfo;
```

- 3) Registramos los métodos encargados de escribir y leer la información que será escrita en el directorio /proc, es importante mencionar que en este ejemplo la información es escrita como un string en formato JSON ya que será leída desde el servidor con la ayuda de Go para su posterior procesamiento en la página web con la ayuda de JavaScript. Y al haber guardado la información en formato JSON, nos facilitamos la lectura y recepción de la data tanto en el servidor como en el cliente.

```
static int write_and_read(struct seq_file *file, void *v){
    unsigned long total_memoria, memoria_libre, memoria_uso, porcentaje_uso;

    si_meminfo(&systemInfo);

    total_memoria = (systemInfo.totalram * systemInfo.mem_unit) / (1024 * 1024);
    memoria_libre = (systemInfo.freeram * systemInfo.mem_unit) / (1024 * 1024);
    memoria_libre = memoria_libre + 2900; //Cache memory (puede variar en cada PC)
    memoria_uso = total_memoria - memoria_libre;
    porcentaje_uso = (100 * memoria_uso) / total_memoria;

    //Lo guardo como JSON para que en el servidor sea mas facil la lectura del archivo
    seq_printf(file, "{");
    seq_printf(file, "\"memoriaTotal\":%lu,", total_memoria);
    seq_printf(file, "\"memoriaLibre\":%lu,", memoria_libre);
    seq_printf(file, "\"memoriaEnUso\":%lu,", memoria_uso);
    seq_printf(file, "\"porcentajeEnUso\":%lu", porcentaje_uso);
    seq_printf(file, "}");
    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
};
```

- 4) Creamos y registramos los métodos encargados de la creación del archivo “memo_201318633” en el directorio /proc

```
static int __init inicio(void)
{
    proc_create("memo_201318633", 0, NULL, &fops);
    printk(KERN_ALERT "          MODULO INSERTADO          \n");
    printk(KERN_ALERT "          Carnet 201318633          \n\n");
    return 0;
}

static void __exit salida(void)
{
    remove_proc_entry("memo_201318633", NULL);
    printk(KERN_ALERT "          MODULO REMOVIDO          \n");
    printk(KERN_ALERT "          SISTEMAS OPERATIVOS 1          \n\n");
}

module_init(inicio);
module_exit(salida);
```

- 5) Creamos nuestro archivo Makefile que será el encargado de la creación del modulo como tal para su posterior incorporación al kernel.

```
obj-m += memo_201318633.o

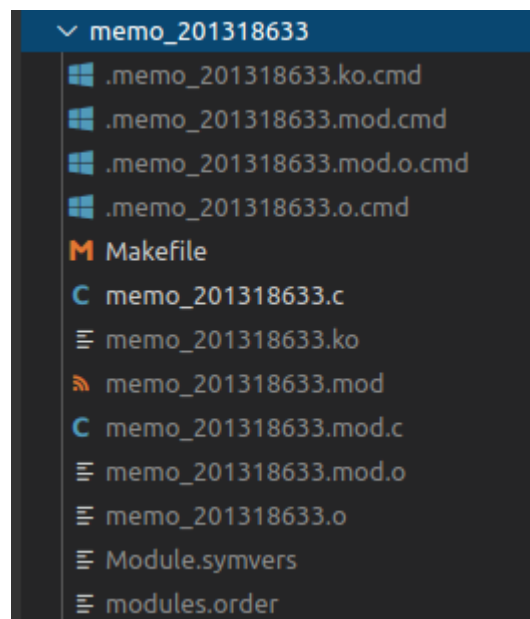
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

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

6) Ahora debemos ejecutar desde nuestra consola el comando

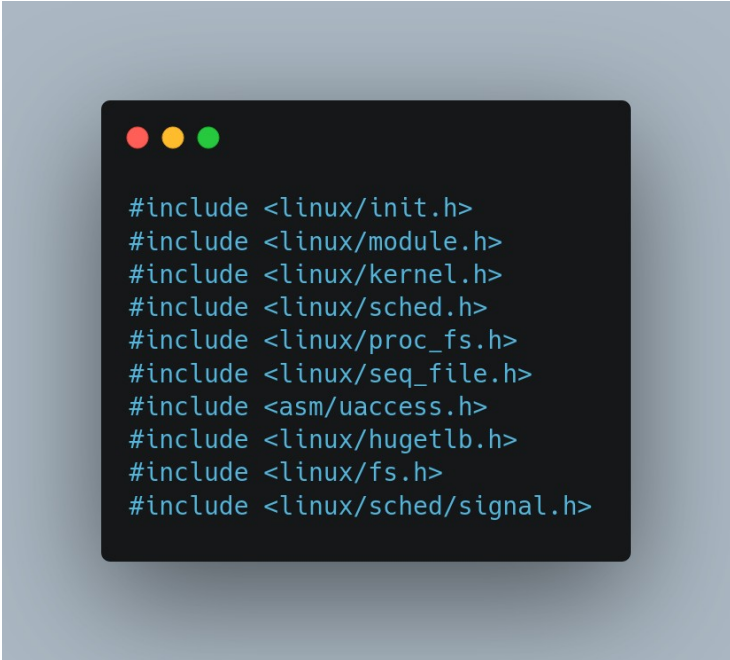


7) Si no hay ningún error, podremos observar en el mismo directorio nuestro archivo “memo_201318633.ko” que representa el módulo que debemos insertar al kernel por medio del comando “insmod”



Modulo CPU

- 1) Importamos librerías que serán utilizadas para obtener la información de los procesos del sistema.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It displays a list of kernel headers to be included in a C program.

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#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>
```

- 2) Documentamos el módulo y hacemos uso de los structs “task_struct” y “list_head”, quienes contienen la información que necesitamos de los procesos y sus respectivos hijos.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It displays module documentation macros and struct declarations for a kernel module.

```
// Documentacion del modulo
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Modulo de CPU task_struct");
MODULE_AUTHOR("Diego Antonio Momotic Montesdeoca");
MODULE_VERSION("1.0");

struct task_struct *task_list;
struct task_struct *task_child;
struct list_head *list;

char buffer[256];
```

- 3) Se inicializan los contadores necesarios para llevar el control del estado de los procesos y se utiliza una función que retorna un string con el estado del proceso basado en un dato de tipo int.

```
//Contadores
unsigned int total = 0;
unsigned int ejecucion = 0;
unsigned int suspendidos = 0;
unsigned int detenidos = 0;
unsigned int zombies = 0;
unsigned int desconocido = 0;

int contador1 = 0;
int contador2 = 0;

char *get_task_state(long state)
{
    total++;
    switch (state)
    {
        case TASK_RUNNING:
            ejecucion++;
            return "RUNNING";
        case TASK_INTERRUPTIBLE:
            suspendidos++;
            return "SLEEPING";
        case 4:
            zombies++;
            return "ZOMBIE";
        case 8:
            detenidos++;
            return "STOPPED";
        case 1026:
            suspendidos++;
            return "SLEEPING";
        default:
            desconocido++;
            return "UNKNOWN";
    }
}
```


- 4) Al igual que en módulo de RAM, se crean los métodos encargados de la escritura de la información dentro del archivo que será creado en el directorio /proc y nuevamente se almacena la información en formato JSON para facilitar su manipulación durante la lectura desde el servidor para su posterior impresión en el cliente web.

```
static int write_and_read(struct seq_file *file, void *v)
{
    total = 0;
    ejecucion = 0;
    suspendidos = 0;
    detenidos = 0;
    zombies = 0;
    desconocido = 0;

    contador1 = 0;

    //Lo guardo como JSON para que en el servidor sea mas facil la lectura del archivo
    seq_printf(file, "{");
    seq_printf(file, "\"procesos\":"); //Atributo procesos
    seq_printf(file, "["); //Inicio de arreglo procesos
    for_each_process(task_list)
    {
        if(contador1 > 0){
            seq_printf(file, ",");
        }
        seq_printf(file, "{");
        seq_printf(file, "\"id\":%d,", task_list->pid);
        seq_printf(file, "\"proceso\":\"%s\",", task_list->comm);
        // seq_printf(file, "\"status\":%d,", task_list->state);
        seq_printf(file, "\"status\":\"%s\",", get_task_state(task_list->state));

        seq_printf(file, "\"hijos\":");
        seq_printf(file, "[");
        contador2 = 0;
        list_for_each(list, &task_list->children)
        {
            if(contador2 > 0){
                seq_printf(file, ",");
            }
            task_child = list_entry(list, struct task_struct, sibling);
            seq_printf(file, "{");
            seq_printf(file, "\"id\":%d,", task_child->pid);
            seq_printf(file, "\"proceso\":\"%s\",", task_child->comm);
            seq_printf(file, "\"status\":\"%s\",", get_task_state(task_child->state));
            seq_printf(file, "}");
            contador2++;
        }
        seq_printf(file, "]");
        seq_printf(file, "}");
        contador1++;
    }
    seq_printf(file, "],"); //Cierre arreglo procesos
    seq_printf(file, "\"total\":%d,", total);
    seq_printf(file, "\"ejecucion\":%d,", ejecucion);
    seq_printf(file, "\"suspendidos\":%d,", suspendidos);
    seq_printf(file, "\"detenidos\":%d,", detenidos);
    seq_printf(file, "\"zombies\":%d,", zombies);
    seq_printf(file, "\"desconocido\":%d", desconocido);
    seq_printf(file, "}"); //Cierre objeto global
    return 0;
}

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

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


- 5) Debemos crear y registrar los métodos encargados de la creación del archivo "cpu_201318633" dentro del directorio /proc.

```
static int inicio(void)
{
    proc_create("cpu_201318633", 0, NULL, &fcpu);
    printk(KERN_ALERT "                MODULO INSERTADO           \n");
    printk(KERN_ALERT "                Diego Antonio Momotic Montesdeoca       \n\n");
    return 0;
}

static void salida(void)
{
    remove_proc_entry("cpu_201318633", NULL);
    printk(KERN_ALERT "                MODULO REMOVIDO           \n");
    printk(KERN_ALERT "                DICIEMBRE 2020           \n\n");
}

module_init(inicio);
module_exit(salida);
```

- 6) Debemos crear un archivo Makefile con la configuración necesaria.

```
static int inicio(void)
{
    proc_create("cpu_201318633", 0, NULL, &fcpu);
    printk(KERN_ALERT "                MODULO INSERTADO           \n");
    printk(KERN_ALERT "                Diego Antonio Momotic Montesdeoca       \n\n");
    return 0;
}

static void salida(void)
{
    remove_proc_entry("cpu_201318633", NULL);
    printk(KERN_ALERT "                MODULO REMOVIDO           \n");
    printk(KERN_ALERT "                DICIEMBRE 2020           \n\n");
}

module_init(inicio);
module_exit(salida);
```

- 7) Finalmente podremos ejecutar el comando “make all” y si no hubieron problemas, veremos nuestro archivo “cpu_201318633.ko” que será el módulo a insertar en el kernel con la ayuda del comando “insmod”

