

Proceso de creación de módulos

Para crear los módulos fue necesario escribir un archivo en c para el módulo de cpu y para el módulo de memoria. Los cuales se describen a continuación:

[memo_201504399.c](#)

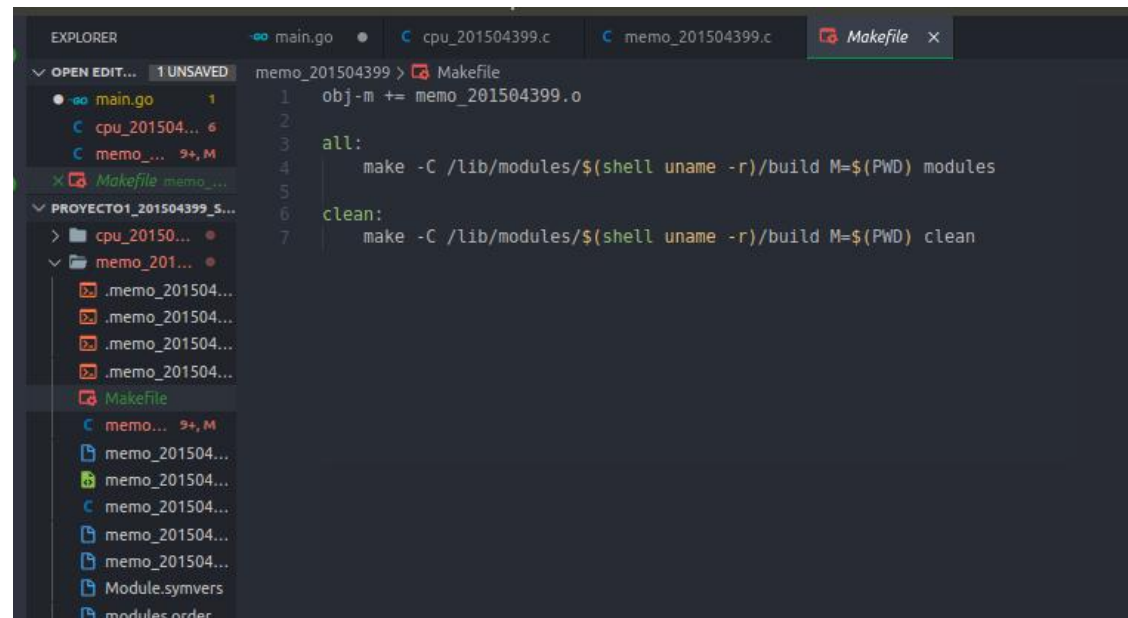
```
main.go • cpu_201504399.c • memo_201504399.c • inf
memo_201504399.c > C memo_201504399.c > [ej] inf
1 #include <linux/proc_fs.h>
2 #include <linux/seq_file.h>
3 #include <asm/uaccess.h>
4 #include <linux/hugetlb.h>
5 #include <linux/module.h>
6 #include <linux/init.h>
7 #include <linux/kernel.h>
8 #include <linux/fs.h>
9 #include <linux/swap.h>
10 #include <asm/page.h>
11 #include <linux/mmzone.h>
12 #include <linux/cpumask.h>
13 #include <linux/kernel_stat.h>
14
15 #define BUFSIZE 150
16
17 MODULE_LICENSE("GPL");
18 MODULE_DESCRIPTION("Escribir informacion de la memoria ram.");
19 MODULE_AUTHOR("201504399");
20
21 struct sysinfo inf;
22 static int escribir_archivo(struct seq_file * archivo, void *v){
23     si_meminfo(&inf);
24     long total_memoria = (inf.totalram * 4 );
25     long memoria_libre = (inf.freeram * 4);
26     long buffer = (inf.bufferram);
27     long cached= (global_node_page_state(NR_FILE_PAGES) * 2 )- inf.bufferram ;
28     long memoria_utilizada = total_memoria - (memoria_libre+buffer +cached);
29     long porcentaje =(memoria_utilizada *100)/total_memoria);
30     //Total Memory - (Free + Buffers + Cached)
31
32     seq_printf(archivo, "{\n");
33     seq_printf(archivo, "\tTotal: \"%lu\", \n", total_memoria /1024); //total memoria ram
34     seq_printf(archivo, "\tUso: \"%lu\", \n", memoria_utilizada/1024); // total memoria consumida
35     seq_printf(archivo, "\tPorcentaje: \"%li\" \n", porcentaje); //porcentaje de consumo
36     seq_printf(archivo, " } \n");
37     return 0;
38 }
```

1. Se importaron las librerías necesarias.
2. Se creó la función escribir archivo para poder escribir en el módulo la información de la ram, se estructuró como formato json. Los datos a cargar son: total de memoria ram, el cual se obtiene de la propiedad totalram de la estructura sysinfo, para obtener la memoria utilizada se aplicó la fórmula $MemoriaUtilizada = MemoriaTotal - (MemoriaLibre + Buffer + Cached)$

```
39
40 static int al_abrir(struct inode *inode, struct file *file){
41     return single_open(file, escribir_archivo, NULL);
42 }
43
44 static struct file_operations operaciones =
45 {
46     .open = al_abrir,
47     .read = seq_read
48 };
49
50 int iniciar(void){ //modulo de inicio
51     proc_create("memo_201504399", 0, NULL, &operaciones);
52     printk(KERN_INFO "%s", "Cargando modulo.\n");
53     printk(KERN_INFO "%s", "201504399\n");
54
55     return 0;
56 }
57
58 void salir(void){ //modulo salida
59
60     remove_proc_entry("memo_201504399", NULL);
61     printk(KERN_INFO "%s", "Removiendo modulo.\n");
62     printk(KERN_INFO "Sistemas Operativos 1.\n");
63 }
64
65 module_init(iniciar);
66 module_exit(salir);
67
```

3. El método iniciar crea el módulo con el nombre memo_201504399, se ejecuta el método al_abrir, el cual llama al método de escritura del archivo e imprime en el buffer el mensaje Cargando módulo y el número de carnet.
4. El método salir borra el módulo e imprime los mensajes en el buffer.

Para poder crear el archivo .ko se necesita el archivo make file que crea los archivos necesarios para crear el archivo tipo módulo (ko), el archivo makefile cuenta con dos comandos, make clean para borrar todos los archivos y make all para crearlos.



The screenshot shows a code editor with a dark theme. The Explorer panel on the left shows a project structure with files like main.go, cpu_201504399.c, memo_201504399.c, and a Makefile. The main editor area shows the content of the Makefile, which includes rules for building the module (obj-m += memo_201504399.o) and cleaning the build directory (clean: make -C /lib/modules/\$(shell uname -r)/build M=\$(PWD) clean).

```
1 obj-m += memo_201504399.o
2
3 all:
4     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
5
6 clean:
7     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Después de ejecutar make all, se puede cargar el módulo con el comando **sudo insmod memo_201504399.ko** y con el comando **dmesg** se pueden ver los mensajes del buffer. Y para descargar el módulo se utiliza el comando **sudo rmmod memo_201504399**

cpu_201504399.c

El archivo cpu_201504399.c tiene los mismos métodos y funciones que el archivo memo.

En la función escribir archivo escribe dentro del módulo cpu_201504399 la información de todos los procesos del cpu, para esto se necesitaron 2 task_struct uno para procesos padre y otro para hijos y un list_head para realizar la lista de procesos hijos.

Para verificar los procesos fue necesario utilizar un for_each_process el cual revisa los procesos padre y escribe en el archivo todas sus propiedades, para revisar si tiene hijos se realizó un list_for_each.

La información se estructuró en formato json y los hijos son un arreglo de formato json dentro de esa estructura.

El código se muestra a continuación:

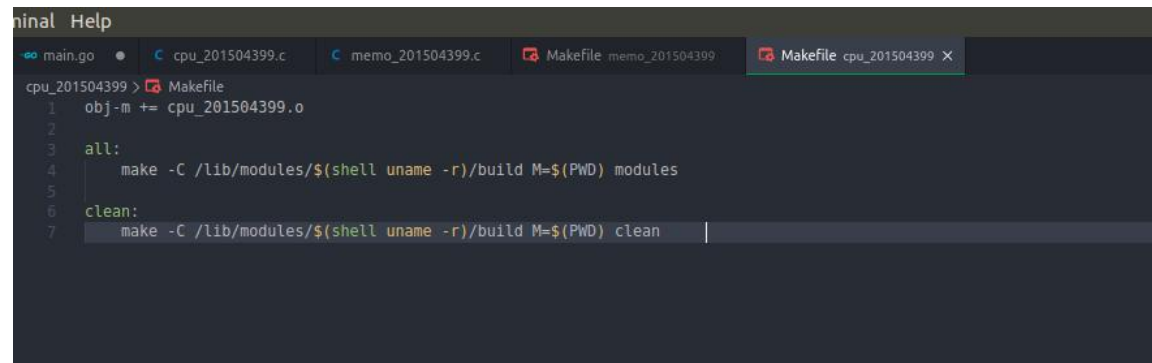
```
main.go • cpu_201504399.c x memo_201504399.c Makefile
cpu_201504399 > C cpu_201504399.c > for_each_process(task)
12 struct task_struct *task; //estructura definifa en sched.h para tareas /procesos
13 struct task_struct *task_child; //estructura necesaria para iterar a través de tareas secundarias
14 struct list_head *list; //estructura necesaria para recorrer la lista en cada tarea -> estructura de hijos.
15
16 static int escribir_archivo(struct seq_file * archivo, void *v){
17     int contador =0;
18     int contadoraux=0;
19     #define Convert(x) ((x) << (PAGE_SHIFT - 10))
20     seq_printf(archivo, " [ ");
21
22     for_each_process(task){
23         if(contador>0){
24             seq_printf(archivo," , \n");
25         }
26         seq_printf(archivo," {\n");
27         seq_printf(archivo, "\t\"PID\": \"%d\\\",", task->pid);
28         seq_printf(archivo, "\t\"PROCESO\": \"%s\\\",", task->comm);
29         seq_printf(archivo, "\t\"ESTADO\": \"%ld\\\",", task->state);
30         if(task->mm!=NULL){
31             seq_printf(archivo, "\t\"MEMORIA\": \"%ld\\\",", Convert(get_mm_rss(task->mm)));
32         } else {
33             seq_printf(archivo, "\t\"MEMORIA\": \"0\\\",");
34         }
35         seq_printf(archivo, "\t\"USUARIO\": \"%d\\\",", __kuid_val(task->real_cred->uid));
36
37         seq_printf(archivo, "\t\"HIJOS\": [ ");
38         contadoraux=0;
39         list_for_each(list, &task->children){
40             if(contadoraux>0){
41                 seq_printf(archivo," , \n");
42             }
43             task_child = list_entry(list, struct task_struct, sibling);
44             seq_printf(archivo, "\t\t\"PID\": \"%d\\\",", task_child->pid);
45             seq_printf(archivo, "\t\t\"PROCESO\": \"%s\\\",", task_child->comm);
46             seq_printf(archivo, "\t\t\"ESTADO\": \"%ld\\\",", task_child->state);
47             if(task_child->mm!=NULL){
48                 seq_printf(archivo, "\t\t\"MEMORIA\": \"%ld\\\",", Convert(get_mm_rss(task_child->mm)));
49             } else {
50                 seq_printf(archivo, "\t\t\"MEMORIA\": \"0\\\",");
51             }
52             seq_printf(archivo, "\t\t\"USUARIO\": \"%d\\\",", __kuid_val(task_child->real_cred->uid));
53             seq_printf(archivo, "\t\t} \n");
54             contadoraux++;
55         }
56         seq_printf(archivo," ] \n");
57         contador++;
58         seq_printf(archivo,"} \n");
59     }
60     seq_printf(archivo, " ] ");
61     #undef k
62     return 0;
63 }
```

```
33     seq_printf(archivo, "\t\t\"MEMORIA\": \"0\\\",");
34 }
35 seq_printf(archivo, "\t\"USUARIO\": \"%d\\\",", __kuid_val(task->real_cred->uid));
36
37 seq_printf(archivo, "\t\"HIJOS\": [ ");
38 contadoraux=0;
39 list_for_each(list, &task->children){
40     if(contadoraux>0){
41         seq_printf(archivo," , \n");
42     }
43     task_child = list_entry(list, struct task_struct, sibling);
44     seq_printf(archivo, "\t\t\"PID\": \"%d\\\",", task_child->pid);
45     seq_printf(archivo, "\t\t\"PROCESO\": \"%s\\\",", task_child->comm);
46     seq_printf(archivo, "\t\t\"ESTADO\": \"%ld\\\",", task_child->state);
47     if(task_child->mm!=NULL){
48         seq_printf(archivo, "\t\t\"MEMORIA\": \"%ld\\\",", Convert(get_mm_rss(task_child->mm)));
49     } else {
50         seq_printf(archivo, "\t\t\"MEMORIA\": \"0\\\",");
51     }
52     seq_printf(archivo, "\t\t\"USUARIO\": \"%d\\\",", __kuid_val(task_child->real_cred->uid));
53     seq_printf(archivo, "\t\t} \n");
54     contadoraux++;
55 }
56 seq_printf(archivo," ] \n");
57 contador++;
58 seq_printf(archivo,"} \n");
59 }
60 seq_printf(archivo, " ] ");
61 #undef k
62 return 0;
63 }
```

María de Los Angeles Herrera Sumalé
201504399

El método iniciar crea el módulo con el nombre `cpu_201504399`, se ejecuta el método `al_abrir`, el cual llama al método de escritura del archivo e imprime en el buffer el mensaje Cargando módulo y el número de carnet.
El método salir borra el módulo e imprime los mensajes en el buffer.

Para poder crear el archivo `.ko` se necesita el archivo make file que crea los archivos necesarios para crear el archivo tipo módulo (ko), el archivo makefile cuenta con dos comandos, `make clean` para borrar todos los archivos y `make all` para crearlos.



```
terminal Help
cpu_201504399 > Makefile
1  obj-m += cpu_201504399.o
2
3  all:
4      make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
5
6  clean:
7      make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Después de ejecutar `make all`, ya se puede cargar el módulo con el comando **`sudo insmod cpu_201504399.ko`** y con el comando **`dmesg`** se pueden ver los mensajes del buffer. Y para descargar el módulo se utiliza el comando **`sudo rmmod cpu_201504399`**