

Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Ingeniería en ciencias y sistemas
Sistemas Operativos 1
Ing. Sergio Mendez

Proyecto 1

Alan Joel Morataya Escobar 201700944
Julio Estuardo Gomez Alonzo 201504042

Creación de módulos

- ⑩ En este caso llamaremos al fichero como `cpu_201700944_201504042.c`

Esto incluye todos los módulos del kernel

Incluye la macros `__init` y `__exit`

Incluye la macros `KERNEL_INFO`

- ⑩ Definimos 3 estructuras que nos permitiran obtener la info de los procesos
- ⑩ Task_struct nos permitira obtener la informacion de cada proceso y acceder a todos.
- ⑩ List_head nos permitira iterar en la lista de procesos del sistema e ir obteniendo cada uno de ellos.

10 El metodo escribirarchivo es el que nos permitira crear un archivo nuevo en la carpeta proc del sistema y con la instruccion seq_printf escribiremos lo que deseamos que contenga dicho archivo

```
static int escribirarchivo(struct seq_file *archivo, void *v) {
    seq_printf(archivo, "-----\n");
    seq_printf(archivo, "|                                     |\n");
    seq_printf(archivo, "| Alan Joel Morataya Escobar         |\n");
    seq_printf(archivo, "| 201700944                          |\n");
    seq_printf(archivo, "| Julio Estuardo Gomez Alonzo       |\n");
    seq_printf(archivo, "| 201504042                          |\n");
    seq_printf(archivo, "|                                     |\n");
    seq_printf(archivo, "|                               Proyecto 1 |\n");
    seq_printf(archivo, "|                               CPU         |\n");
    seq_printf(archivo, "| (char [54])-----\n");
    seq_printf(archivo, "-----\n");
    seq_printf(archivo, "-----LISTA DE PROCESOS -----\n");
}
```

⑩ Esto pertenece al mismo metodo escribirarchivo y en esta parte se procede a recorrer todos los procesos del sistema con los structs que se declararon anteriormente, accedemos a sus atributos y procedemos a escribirlos en el archivo y asi mismo hacer lo mismo con los hijos de cada proceso si es que los tuviera.

```
for_each_process( task ) {
    seq_printf(archivo, "\nPID: %d\n", task->pid);
    seq_printf(archivo, "NOMBRE: %s\n", task->comm);
    seq_printf(archivo, "ESTADO: %li\n", task->state);

    list_for_each(list, &task->children)
    {
        child = list_entry(list, struct task_struct, sibling);
        seq_printf(archivo, "\tNOMBRE DEL HIJO: %s PID:%d ESTADO: %li\n", child->comm, child->pid, child->state);
    }
}

return 0;
}
```

➤ En esta estructura,

observamos la llamada a la función OS2_open, esta función, devuelve la instrucción single_open que se ejecuta cada vez que llamamos al proceso, como parametro se define el codigo que mostrara nuestro proceso que en este caso es el metodo “escribirarchivo”

```
static int hello_proc_open(struct inode *inode, struct file *file) {
    // seq_printf(file, "-----LISTA DE PROCESOS ----- \n");
    return single_open(file, escribirarchivo, NULL);
}
```

⑩ A continuacion se define la configuracion del proceso, es decir las operaciones que realizara

```
static const struct file_operations hello_proc_fops = {
    .open = hello_proc_open,
    .read = seq_read
};
```

dicho proceso con .open indicamos el metodo a ejecutar al abrir dicho proceso

- ⑩ Definimos las funciones de inicio y fin del del procedimiento, observamos las que en ambos utilizamos un nombre en común `cpu_201700944_20150402` que sera el nombre del proceso que se creara y se eliminara, la instrucción `proc_create` le enviamos la direccion de memoria de la estructura que contiene la configuración del proceso

```
static int inicio(void)
{
    //al cargar imprimimos el carnet
    proc_create("cpu_201700944_20150402", 0, NULL, &hello_proc_fops);
    printk(KERN_INFO "Nombre: Alan Joel Morataya Escobar\n");
    printk(KERN_INFO "Nombre: Julio Estuardo Gomez Alonzo\n");
    return 0;
}

//esto con rmmmod
static void salir(void)
{
    //al salir imprimo el nombre del curso
    remove_proc_entry("cpu_201700944_20150402", NULL);
    printk(KERN_INFO "Curso: Sistemas Operativos 1\n");
    // return 0;
}
```

- ⑩ Definimos la informacion basica del modulo asi como los metodos a ejecutar al iniciar el modulo y al finalizarlo, en este caso se llamara al metodo “inicio” al iniciar el modulo y el metodo “salir” al finalizarlo

```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Alan201700944");
MODULE_DESCRIPTION("Modulo para mostrar la informacion del CPU");

module_init(inicio);
module_exit(salir);
```

Ya con el codigo escrito procedemos a crear un archivo `MAKEFILE` que contendra las instrucciones para que se pueda crear correctamente nuestro modulo, el archivo `MAKEFILE` contendra un codigo como el siguiente:

```
Makefile
obj-m += cpu_201700944_20150402.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
```

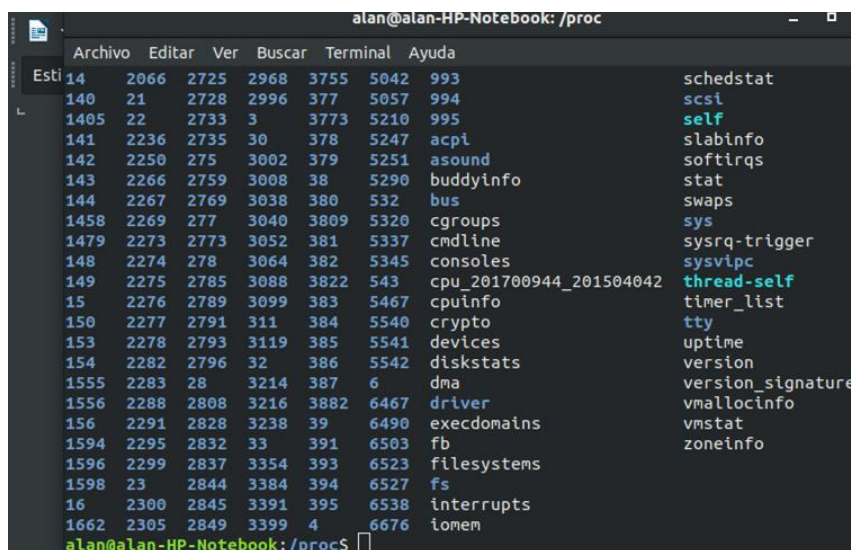
buscamos la carpeta `/lib/modules/$(Shell uname -r)/build`, es decir la versión del kernel que estamos utilizando, y ejecutamos el código.

Compilamos el archivo con el comando `make` y se creará un archivo con el nombre de nuestro módulo más la extensión `.ko` en este caso `cpu_201700944_20150402.ko`

Con el comando `insmod` y el nombre del módulo procedemos a instalar el módulo

```
alan@alan-HP-Notebook:~/Escritorio/partel_proyectoIsopes$ cd cpu
alan@alan-HP-Notebook:~/Escritorio/partel_proyectoIsopes/cpu$ sudo insmod cpu_201700944_20150402.ko
[sudo] contraseña para alan:
alan@alan-HP-Notebook:~/Escritorio/partel_proyectoIsopes/cpu$
```

- 10 Accedemos a `/proc` con el comando `cd /proc`



Se puede observar que ya se creó un proceso con el nombre del módulo que instalamos.

Con el comando `cat` + nombre del proceso, desplegaremos la información del archivo que creó nuestro módulo, en este caso serán todos los procesos que están en ejecución en el sistema, así como sus hijos.
Comando: `cat cpu_201700944_20150402`

⑩ Con el

```

alan@alan-HP-Notebook: /proc
Archivo Editar Ver Buscar Terminal Ayuda
ESTADO: 1026

PID: 6715
NOMBRE: bash
ESTADO: 1
    NOMBRE DEL HIJO: cat PID:7146 ESTADO: 0

PID: 6817
NOMBRE: code
ESTADO: 1
    NOMBRE DEL HIJO: code PID:6836 ESTADO: 1
    NOMBRE DEL HIJO: code PID:6853 ESTADO: 1
    NOMBRE DEL HIJO: code PID:6871 ESTADO: 1
    NOMBRE DEL HIJO: bash PID:6951 ESTADO: 1

PID: 6836
NOMBRE: code
ESTADO: 1
    NOMBRE DEL HIJO: cpptools PID:6897 ESTADO: 1

PID: 6853
NOMBRE: code
ESTADO: 1

```

comando sudo rmmod

cpu_201700944_201504042.ko procedemos a eliminar el proceso y por ende ya no existira en el directorio proc.

- ⑩ Para el modulo de memoria ram son exactamente los mismo pasos, unicamente cambia el contenido que se escribira en el archivo que se creara en el proc.
 - ✦ Utilizaremos el struct sysinfo para obtener la informacion del sistema, en este caso la informacion de nuestra memoria ram, con el atributo totalram obtendremos la memoria total de nuestro sistema y con el atributo freeram la memoria libre de nuestro sistema y procederemos a escribir dicha informacion en el archivo.

```

struct sysinfo datos; //struct para manejar la informacion del usuario
//esto con insmod

static int escribirarchivo(struct seq_file *arch, void *v) {
    si_meminfo(&datos);
    long memoriatotal=(datos.totalram *4);
    long memorialibre=(datos.freeram *4);

```

```

seq_printf(arch,"-----\n");
seq_printf(arch,"| Laboratorio Sistemas Operativos 1 | \n");
seq_printf(arch,"| Escuela de vacaciones Junio 2020 | \n");
seq_printf(arch,"| Alan Joel Morataya Escobar | \n");
seq_printf(arch,"| 201700944 | \n");
seq_printf(arch,"| Julio Estuardo Gomez Alonzo | \n");
seq_printf(arch,"| 201504042 | \n");
seq_printf(arch,"| | \n");
seq_printf(arch,"| Proyecto 1 | \n");
seq_printf(arch,"| MEMORIA RAM | \n");
seq_printf(arch,"| | \n");
seq_printf(arch,"-----\n");
seq_printf(arch,"Sistema Operativo: Ubuntu 18.04\n");
seq_printf(arch,"Memoria total: \t %8lu MB \n",memoriatotal/1024); //falta pasarlo a megas
seq_printf(arch,"Memoria libre: \t %8lu MB \n",memorialibre/1024);
seq_printf(arch,"Memoria en uso: \t %ld %%\n",((memoriatotal-memorialibre)*100)/memoriatotal);
return 0;
}

```