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

Manual de creación de módulos de Kernel

Juan José Hernández Rodríguez
201612378
15 de diciembre de 2021

Módulos

El kernel de Linux es de tipo monolítico y una de las funciones que tenemos para poder agregar características al kernel es con módulos, que son fragmentos de código o binarios que pueden ser cargados al kernel según se necesite. Para la aplicación se realizaron dos módulos, uno para obtener información de todos los procesos activos en el sistema y otro para obtener el estado de la memoria.

Estructura básica de los módulos utilizados

La estructura de cada módulo tiene una base similar por no decir igual, de manera muy básica los módulos cuentan con las importaciones de las librerías de los headers del kernel de Linux, un método de escritura, una estructura de operaciones del módulo, un método de inicia que se ejecuta al cargar el módulo al kernel, un método de salida al eliminar el modulo del kernel y la información la licencia autor y descripción. Ambos módulos cuentan con los métodos de inicio y salida, estos despliegan un mensaje personalizado en el log del kernel.

```c
static int __init memoMod_init(void)
{
    struct proc_dir_entry *entry;
    entry = proc_create("memo_201612378", 0777, NULL, &ops_memo);
    if (!entry)
    {
        return -1;
    }
    else
    {
        printk(KERN_INFO "201612378\n");
    }
    return 0;
}

static void __exit memoMod_exit(void)
{
    remove_proc_entry("memo_201612378", NULL);
    printk(KERN_INFO "Sistemas operativos 1\n");
}
```

Se utilizo el siguiente enlace para la documentación de las librerías utilizadas, se trabajó sobre un kernel de Linux 5.11.0

https://elixir.bootlin.com/linux/latest/source/include/linux/sched.h

Módulo de memoria

El módulo de memoria crea una estructura para almacenar los valores de la estructura sysinfo y se obtienen los datos de la memoria de la estructura. Todos los valores son retornados en bytes por lo que se pasaron a megabytes para un manejo más cómodo de los datos, con la función seq_printf() escribimos una línea para el archivo que se creara sobre la ruta /proc al momento de montar el módulo. En este modulo simplemente se obtienen los datos y se escriben dentro del archivo para leerlos desde el servidor mas adelante.

```c
static int escribir_memo(struct seq_file *m, void *v)
{
    si_meminfo(&inf);

    long memoriaTotal = inf.totalram*(unsigned long)inf.mem_unit/(1024*1024);
    long memoriaLibre = inf.freeram*(unsigned long)inf.mem_unit/(1024*1024);
    long memoriaCompartida = inf.sharedram*(unsigned long)inf.mem_unit/(1024*1024);
    long memoriaBuffer = inf.bufferram*(unsigned long)inf.mem_unit/(1024*1024);
    long memoriaUsada = memoriaTotal - (memoriaLibre + memoriaCompartida);

    seq_printf(m, "{\n");
    seq_printf(m, "\"memTotal\": %lu,\n",memoriaTotal);
    seq_printf(m, "\"memLibre\": %lu,\n",memoriaLibre);
    seq_printf(m, "\"memCompartida\": %lu,\n",memoriaCompartida);
    seq_printf(m, "\"memBuffer\": %lu,\n",memoriaBuffer);
    //seq_printf(m, "}\n");
    return 0;
}
```

Módulo de CPU

El modulo de CPU difiere del modulo de memoria porque en este tenemos que iterar los procesos que están actualmente en el sistema, fuera de eso es básicamente la misma idea: obtener valores de la estructura task_struct.

Se crearon 3 structs para iterar los procesos y los hijos de estos, además de varias variables contadoras que servirán para guardar el numero de procesos de cada tipo y el numero de procesos totales que están corriendo dentro del sistema.

```c
struct task_struct *tareas;
struct task_struct *hijos;
struct list_head *head;

int totalprocesos = -1;
int totalejecucion = -1;
int totalsuspendido = 0;
int totaldetenido = 0;
int totalzombie = -1;
int totalunknow = -1;
```

Para obtener los valores de cada proceso simplemente se recorre la lista de procesos y se toman los atributos de struct que vamos a necesitar, sin embargo se hicieron validaciones sobre el estado del proceso para escribir dentro del archivo el nombre del estado para un reporte mas entendible.

```
for_each_process(tareas)
{
    totalprocesos++;
    seq_printf(m, "{\n");
    seq_printf(m, "\"Pid\" :%ld,\n", tareas->pid);
    seq_printf(m, "\"Nombre\": \"%s\",\n", tareas->comm);
    seq_printf(m, "\"Usuario\": %d,\n", tareas->cred->uid);
    //seq_printf(m, "\"Estado\": %ld,\n", tareas->state);

    if(tareas->state == 0)
    {
        seq_printf(m, "\"Estado\": \"Running\",\n");
        totalejecucion = totalejecucion+1;
    }
    else if(tareas->state == 1)
    {
        seq_printf(m, "\"Estado\": \"Suspended\",\n");
        totalsuspendido = totalsuspendido+1;
    }
```

Para cada proceso se recorre la lista de sus hijos y de ellos se obtienen sus datos, la estructura de cada proceso es la misma de tipo task_struct. Por ultimo se escriben todos los contadores dentro del archivo para leerlos más adelante con el servidor.

```
    seq_printf(m, "\"Hijos\": [\n");
    list_for_each(head, &tareas->children)
    {
        hijos = list_entry(head, struct task_struct, sibling);
        seq_printf(m, "{\n");
        seq_printf(m, "\"Pid\" :%ld,\n", hijos->pid);
        seq_printf(m, "\"Nombre\": \"%s\"\n", hijos->comm);
        seq_printf(m, "},\n");

    }
    seq_printf(m, "{");
    seq_printf(m, "}\n");
    seq_printf(m, "]\n");
    seq_printf(m, "},\n");
}

if(totalprocesos == -1)totalprocesos = 0;
if(totalejecucion == -1)totalejecucion = 0;
if(totalsuspendido == -1)totalsuspendido = 0;
if(totaldetenido == -1)totaldetenido = 0;
if(totalzombie == -1)totalzombie = 0;
if(totalunknow == -1)totalunknow = 0;
```