Universidad San Carlos de Guatemala
Ingeniería en Ciencias y Sistemas
Vacaciones Diciembre 2021
Laboratorio Sistemas Operativos 1

Manual del proceso de creación de los módulos

Audrie Annelisse del Cid Ochoa
201801263
3012930900101

Manual del proceso de creación de los módulos

Para la obtención de información de memoria, cpu y procesos, se hizo uso de los siguientes módulos: cpu_201801263 y memo_201801263.

En ambos módulos se utilizaron funciones de inicio y salida similares, variando en ciertos detalles como nombres y datos requeridos.

Como primer paso nos encontramos con la inserción del módulo, para esto se utilizó el macro "**module_init"** el cual requirió de las siguientes librerías:

```
//Header obligatorio de todos los modulos
#include <linux/module.h>
//Header para usar KERN_INFO
#include <linux/kernel.h>

//Header para los macros module_init y module_exit
#include <linux/init.h>
```

La función definida a ejecutar al momento de insertar el módulo fue "**_insert"**

```
static int _insert(void)
{
    //Creando Modulo en /procs
    proc_create("memo_201801263", 0, NULL, &operaciones);
    printk(KERN_INFO "201801263\n");
    return 0;
}
```

Esta función se ayudó con el sistema de archivos proc, del cual utilizamos **"proc_create"** el cual permitió la creación del archivo virtual que utilizamos para la carga y obtención de la información de cpu, memoria y procesos.

Según el módulo la información vario respecto a la impresión de información como mensaje, en el caso del CPU, se imprimió el nombre del estudiante y para la ram se imprimió el carné. En esta misma función se declaró el nombre que llevaría cada módulo.

Para el módulo con la información de la ram, se asignó como nombre **"memo_201801263"** y para el módulo con la información del CPU, se asignó como nombre **"cpu_201801263".**

Como siguiente paso se definió la estructura "**proc_ops"** de la siguiente forma:

```c
//Si el kernel es 5.6 o mayor se usa la estructura proc_ops
static struct proc_ops operaciones =
{
    .proc_open = al_abrir,
    .proc_read = seq_read
};
```

En el siguiente paso es donde se procede a insertar la información dentro del módulo, para cada módulo se realizó un procedimiento diferente, a continuación se detallará cada uno.

CPU.c

Para el módulo de CPU, se utilizó la estructura "**task_struct"**, esta permitió obtener la información necesaria respecto a los procesos existentes.

```c
//Funcion que se ejectua cada vez que se lee el archivo con el comando CAT
static int escribir_archivo(struct seq_file *archivo, void *v)
{
    unsigned long memoria_total;
    // unsigned long memoria_proceso;
    // unsigned long porcentaje_memoria;
    unsigned long rss;

    si_meminfo(&inf);
    memoria_total= (inf.totalram*inf.mem_unit)/(1024*1024);
    //Obtener el listado de procesos en ejecución
    for_each_process(proceso){


        if (proceso->mm){
            rss=get_mm_rss(proceso->mm);
        }
        seq_printf(archivo, "\"Proceso\": \"%s\",\n \"PID\": \"%d\",\n \"Usuario\": \"%d\",\n \"RamB\": \"%8li\",\n \"Memoria_TotalM\": 
        list_for_each(hijos, &(proceso->children)){


            proceso_hijo=list_entry(hijos, struct task_struct, sibling);
            if (proceso_hijo->mm){
            rss=get_mm_rss(proceso_hijo->mm);
        }
            seq_printf(archivo, "\t\"Proceso_Hijo\": \"%s\",\n \"PID\": \"%d\",\n \"Usuario\": \"%d\",\n \"RamB\": \"%8li\",\n \"Estado\
        }
    }

    return 0;
}
```

De esta estructura utilizamos los siguientes atributos:

- **comm:** Permitió obtener el nombre del proceso
- **pid:** Permitió obtener la identificación del proceso
- **state:** Permitió obtener el estado en el que se encontraba el proceso

RAM.c

Para el módulo de Ram, se utilizó la estructura “**sysinfo”**, ya que esta cuenta con atributos que permiten obtener la información que se requería.

Data Fields

__kernel_long_t	uptime
__kernel_ulong_t	loads [3]
__kernel_ulong_t	totalram
__kernel_ulong_t	freeram
__kernel_ulong_t	sharedram
__kernel_ulong_t	bufferram
__kernel_ulong_t	totalswap
__kernel_ulong_t	freeswap
__u16	procs
__u16	pad
__kernel_ulong_t	totalhigh
__kernel_ulong_t	freehigh
__u32	mem_unit
char	_f [20-2 *sizeof(__kernel_ulong_t)-sizeof(__u32)]

```c
//Funcion que se ejecuta cada vez que se lee el archivo con el comando CAT
static int escribir_archivo(struct seq_file *archivo, void *v)
{
    unsigned long memoria_total;
    unsigned long memoria_libre;
    unsigned long memoria_consumida;
    unsigned long memoria_cache;
    unsigned long memoria_compartida;
    unsigned long porcentaje;
    //Lleno mi estructura con los datos de memoria ram
    si_meminfo(&inf);
    memoria_total= inf.totalram*inf.mem_unit;
    memoria_libre= inf.freeram*inf.mem_unit;
    memoria_cache= (inf.bufferram*9*inf.mem_unit);
    memoria_compartida= inf.sharedram*inf.mem_unit;
    memoria_consumida= (inf.totalram-(((inf.bufferram*9))+inf.freeram))*inf.mem_unit;

    porcentaje= (((memoria_consumida/(1024*1024))+440)/(memoria_total/(1024*1024)))*100;

    // seq_printf(archivo,"{\n");
    // seq_printf(archivo, "\"Memoria_Total\": \"%8li\", \n", memoria_total/(1024*1024));
    // seq_printf(archivo, "\"Memoria_Libre\": \"%8li\", \n", memoria_libre/(1024*1024));
    // seq_printf(archivo, "\"Memoria_cache\": \"%8li\", \n", (memoria_cache/(1024*1024))-400);
    // seq_printf(archivo, "\"Memoria_Compartida\": \"%8li\" \n", memoria_compartida/(1024*1024));
    // seq_printf(archivo, "}");

    seq_printf(archivo, "%8li ", memoria_total/(1024*1024));
    seq_printf(archivo, " %8li \n", memoria_libre/(1024*1024));
    return 0;
}
```

De esta estructura utilizamos los siguientes atributos:

- **totalram**: Para obtener el total de ram y bytes
- **freeram:** Para obtener el total de ram libre

Estos datos obtenidos fueron escritos en el módulo para utilizar más adelante desde el servidor.