201504002



Faculta de Ingeniería

Sistemas Operativos 1

Módulos del Kernel

Módulo de Memoria (sysinfo)

Para el módulo de memoría se utilizó sysinfo para obtener la información del sistema.

Luego se imprimió en un archivo para el módulo un JSON que contiene la memoria total, la memoria libre y el porcentaje de memoria que este posee.

El memo_open se utiliza para abrir el archivo y luego file_operations se sobreescribe para que escriba en el módulo de memoria que se está creando

Luego se utiliza printk para escribir en el buffer de /proc

```
Modulos > Memory > C memo_201504002.c > ۞ memo_show(seq_file *, void *)
     static int memo_open(struct inode *inode, struct file *file){
     return single_open(file, memo_show, NULL);
     static const struct file_operations memo fops = {
      .owner = THIS MODULE,
     .open = memo_open,
     .read = seq_read,
     .llseek = seq_lseek,
     .release = single_release,
      static int __init memo_init(void) {
          printk(KERN INFO "201504002\n");
          proc create("memo 201504002", 0, NULL, &memo fops);
         return 0;
      static void __exit memo_exit(void) {
         remove_proc_entry("memo_201504002", NULL);
          printk(KERN_INFO "Sistemas Operativos 1\n");
     module init(memo init);
     module_exit(memo_exit);
```

Módulo de CPU (task_struct)

Se utilizó el struct task_struct para obtener el proceso

Luego utilizando un for_each_process se recorrieron todos los tasks padres.

Se dividieron los estados de los procesos de la siguiente manera:

- R (Ejecutando)
- T (Detenido)
- S (Suspendido)
- Z (zombie)
- X (Otro)

```
s home.component.ts C cpu_201504002.c X
char user[16];
char estado;
int proc ejec = 0:
int proc_susp = 0;
int proc_stop = 0;
int proc_zombie = 0;
int total;
struct task_struct *task;
struct task_struct *task_child;
seq_printf(m,"{\n\"procesos\":[");
for_each_process(task){
    if(task->state == TASK_RUNNING){
        estado = 'R';
    }else if((task->state == TASK_STOPPED)||(task->state == TASK_TRACED)){
       estado = 'T';
    proc_stop++;
}else if((task->state == TASK_INTERRUPTIBLE)||(task->state == TASK_UNINTERRUPTIBLE)){
        estado = 'S';
        proc_susp++;
    }else if(task->state == EXIT_ZOMBIE){
```

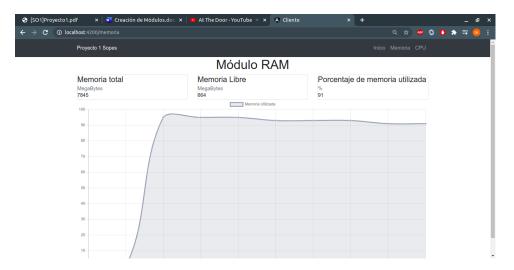
Luego se escribió el JSON del proceso.

Una vez se lee el padre, se verifica si tiene hijos con list_for_each y luego se realiza el mismo procedimiento de los estados.

```
usuario = task->cred->uid.val;
if( usuario == 0){
    strcpy(user,"root");
}else{
    strcpy(user,"inti");
}
//imprimir proceso padre
seq_printf(m,"{\n \"pid\":\"%d\",\n \"nombre\":\"%s0\",\n \"usuario\":\"%s\",\n \"estado\":\"%c
task->comm,
user,
estado);
list_for_each(list,&task->children){
    task_child=list_entry(list,struct task_struct,sibling);

if(task_child->state == TASK_RUNNING){
    estado = 'R';
    proc_ejec++;
}else if((task_child->state == TASK_STOPPED)||(task->state == TASK_TRACED)){
    estado = 'T';
    proc_stop++;
}else if((task_child->state == TASK_INTERRUPTIBLE)||(task->state == TASK_UNINTERRUPTIBLE)){
    estado = 'S';
    proc_susp++;
}else if(task_child->state == EXIT_ZOMBIE){
    estado = 'Z';
    proc_zombie++;
}
```

Memoria



Referencias

https://www.elconspirador.com/2014/12/21/crear-un-proceso-en-un-modulo-del-kernel/

 $\frac{https://blog.sourcerer.io/writing-a-simple-linux-kernel-module-\\d9dc3762c234?gi=433ef60d45cc}{d9dc3762c234?gi=433ef60d45cc}$