Manual técnico

Consideraciones

El proyecto fue realizada en:

- En Arch Linux
- 3Gb de memoria RAM
- 4Gb de memoria SWAP
- 75GB de almacenamiento interno
- Procesador intel core i5 7gen
-

Estructura de un módulo del Kerenel Linux

para hacer un módulo de kernel linux es importante tener claras la funciones inicio y las funciones de salida, las cuales lucen de la siguiente manera

```
static int __init inicio(void){
    printk(KERN_INFO "Modulo iniciado");
    return 0;
}
```

y

```
static void __exit  fin(void){
    printk(KERN_INFO "Modulo finalizado");
}
```

respectivamente. En donde simplemente imprimimos un mensaje para indicar que el módulo inicio y que el módulo finaliza, sin embargo en la función de inicio y de fin es conveniente llamar a las funciones o colocar el código que deseemos para que el módulo ejecute su función de la mejor manera.

Luego es importante indicar que hemos codificado como el inicio del módulo y la salida del módulo, se hace de la siguiente manera:

```
module_init(inicio);
```

y

```
module_exit(fin);
```

respectivamente.

Módulo Procesos:

Para obtener información del módulo de los procesos se hace uso de una estructura llamada task_struc de la siguiente manera:

```
struct task_struct *task;
```

dicha estructura tiene varios miembros pero los utilizados para la práctica son:

Para obtener el identificador del proceso:

```
task->pid
```

Para obtener el nombre del proceso:

```
task->comm
```

para obtener el identificador del usuario que creó el proceso, se utiliza la función from_kuid para que obtenga la información solicitada desde esa estructura y obtenemos el id del usuario por medio de task_uid mandandole como parámetro el proceso del que queremos saber el usuario.

```
from_kuid(&init_user_ns, task_uid(task))
```

Para obtener el número identificador del estado en el que se encuentra el proceso actual hacemos uso del atributo state:

```
task->state
```

Para obtener el nombre como tal del estado del proceso se creó la siguiente funcion:

```
get_state_name(task->state))
```

La cual contiene lo siguiente:

```
char* get_state_name(long state){
    char* string="";
    switch(state){
        case 0:
        string="TASK_RUNNING";
        break;
        case 1:
        string="TASK_INTERRUPTIBLE";
        break;
        case 2:
        string="TASK_UNINTERRUPTIBLE";
        break;
        case 4:
        string="__TASK_STOPPED";
        break;
        case 8:
        string="__TASK_TRACED";
        break;
        case 16:
```

```
        string="EXIT_DEAD";
        break;
        case 32:
        string="EXIT_ZOMBIE";
        break;
        case 64:
        string="TASK_PARKED";
        break;
         case 128:
        string="TASK_DEAD";
        break;
         case 256:
        string="TASK_WAKEKILL";
        break;
         case 512:
        string="TASK_WAKING";
        break;
        case 1024:
        string="TASK_NOLOAD";
        break;
        case 2048:
        string="TASK_NEW";
        break;
        case 4096:
        string="TASK_STATE_MAX";
        break;
        default:
            string="";
            break;
    }
    return string;
}
```

para obtener la cantidad de cpu consumido por cada proceso se utiliza el siguiente atributo del proceso:

```
task->cpu
```

Para obtener los hijos de un proceso se hace por medio del atributo children:

```
 task->children
```

para lo siguiente utilizaremos dos variables las cuales son:

```
struct task_struct *child_tasks;
```

y

```
struct list_head *list;
```

dicho atributo puede ser recorrido por medio de un list for each

```
list_for_each(list, &task->children)
```

dónde list seria basicamente nuestro cursor del ciclo y task->children es el atributo sobre el cual iteramos.

ahora necesitaremos obtener la estructura para cierta entrada la cual se hace por medio del list_entry:

```
child_tasks = list_entry(list, struct task_struct, sibling);
```

donde el primer argumento es el puntero de la lista, el segundo es el tipo de datos y el tercero es el miembro de la estructura que se refiere a la lista y child_task es donde se guardará el task_struct que obtendremos.

ahora obteniendo la estructura de los procesos hijos se puede recorrer de la siguiente manera:

```
for_each_process( child_tasks ){
 print(”id: %d, nombtre:%s",child_tasks->pid,child_tasks->comm);
}
```

for_each_process recorre todos los procesos contenidos en la estructura ingresada, que en este caso son los procedimientos hijos.