Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Sistemas Operativos 2
Catedráticos: Ing. Cesar Rolando Batz Saquimux
Auxiliar: Franklin Estuardo Verlázquez Fuentes

Manual Técnico

Práctica 1

Wendy Lucía Mazariegos Samayoa	2007 14966
Mario Josue Alvarado Lima	2011 23848
Guillermo Rene Medinilla Hernández	2012 13233

Guatemala, 24 de agosto de 2020

Módulos

Módulo CPU

Bibliotecas

```
#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/sched.h>        // pr_info

#include <linux/sched/signal.h> // for_each_process

#include <linux/proc_fs.h>

#include <linux/seq_file.h>

#include <linux/init.h>

#include <linux/uaccess.h>

#include <linux/slab.h>

#include <linux/fs.h>

#include <linux/sysinfo.h>

#include <linux/mm.h>

#include <linux/swap.h>

#include <linux/user_namespace.h>

#include <linux/cred.h>

#include <linux/mm_types.h>
```

Declaración de bibliotecas que son necesarias para la obtención y manipulación de los datos de los módulos de kernel.

Variables

```
struct task_struct *procesos
```

Esta variable contendrá todos los procesos para obtener la información.

```
size_t total_de_procesos = 0;
```

Esta variable llevará el conteo de todo los procesos, es inicializada en 0 para ir incrementando en uno cada proceso que es analizado.

```
size_t procesos_en_ejecucion = 0;
```

Esta variable llevará el conteo de todos los procesos que están en estado de ejecución, es inicializada en 0 para ir incrementando en uno cada proceso que esté en estado de ejecución.

```
size_t procesos_suspendidos = 0;
```

Esta variable llevará el conteo de todos los procesos que están en estado de suspendido, es inicializada en 0 para ir incrementando en uno cada proceso que esté en estado de suspendido.

```
size_t procesos_detenidos = 0;
```

Esta variable llevará el conteo de todos los procesos que están en estado de detenido, es inicializada en 0 para ir incrementando en uno cada proceso que esté en estado de detenido.

```
size_t procesos_zombies = 0;
```

Esta variable llevará el conteo de todos los procesos que están en estado de zombie, es inicializada en 0 para ir incrementando en uno cada proceso que esté en estado de zombie.

```
size_t memoria = 0;
```

Esta variable llevará el conteo del total de memoria que está utilizando cada proceso, es inicializada en 0 para ir incrementando según la memoria que utilice cada proceso analizado.

Lógica

Se escribe un archivo en formato json por lo que se le dará la estructura de un objeto con elementos procs que contendrá un arreglo con objetos donde cada objeto tendrá información del proceso analizado, esta información será id, nombre, usuario, estado, memoria utilizada, porcentaje memoria utilizada. El objeto principal también tendrá elementos de total de memoria consumida, total de procesos, procesos en ejecución, procesos suspendidos, procesos detenidos y procesos zombies.

Para cada proceso de la variable procesos se debe de obtener la ram utilizada. Preguntamos si el campo mm de procesos existe y si el campo total_vm del campo mm de procesos existe, si existe asignamos a la variable ram el valor de total_vm en caso contrario asignamos un valor por defecto de 0. Se incrementa el valor de la variable memoria el valor de ram del proceso que se está analizando.

Se utilizan los campos state y exit_state de procesos, el campo state nos indica si el proceso está en estado de inejecutable en caso tenga valor de -1, si tiene valor de 0 significa que el proceso se está ejecutando y si tiene un valor mayor a 0 el proceso se detuvo. En caso se haya detenido el proceso utilizamos el campo exit_state que nos indica en qué estado se detuvo, si tiene un valor de 16 significa que está en estado zombie y si tiene un valor de 32 está suspendido el proceso, de lo contrario el proceso se detuvo.

Para obtener el id del proceso se utiliza el campo pid de procesos, para obtener el nombre del proceso se utiliza el campo comm de procesos, para obtener el usuario se utiliza el campo euid del campo cred de procesos.

Ya con todos los datos necesarios se puede escribir cada proceso y las variables de memoria en el archivo en formato de json.

Módulo Árbol CPU

Bibliotecas

```
#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/sched.h>        // pr_info

#include <linux/sched/signal.h> // for_each_process

#include <linux/proc_fs.h>

#include <linux/seq_file.h>

#include <linux/init.h>

#include <linux/uaccess.h>

#include <linux/slab.h>

#include <linux/fs.h>

#include <linux/sysinfo.h>

#include <linux/mm.h>

#include <linux/swap.h>

#include <linux/user_namespace.h>

#include <linux/cred.h>

#include <linux/mm_types.h>
```

Declaración de bibliotecas que son necesarias para la obtención y manipulación de los datos de los módulos de kernel.

Variables

```
struct task_struct *task;
```

Esta variable contendrá todos los procesos “padres” para obtener la información.

```
struct task_struct *task_child;
```
Esta variable contendrá todos los procesos “hijos” para obtener la información.

```
struct list_head *list;
```
Esta variable contendrá todos los procesos hijos de cada proceso padre para recorrer los procesos hijos y poder asignarle el proceso hijo a task_child.

Lógica

Se escribe un archivo en formato json con un arreglo de objeto donde cada objeto principal puede o no tener hijos y los hijos es un arreglo con objetos, el objeto principal tendrá información de id y nombre del proceso.

Para obtener el id se utiliza el campo pid de la variable task y para obtener el nombre se utiliza el campo comm de la variable task.

Para cada proceso de la variable task se debe de obtener la información de id y nombre, posteriormente se debe de recorrer la lista de hijos de cada proceso de la variable task por medio del campo children que se le asigna a la variable list.

Posteriormente se le debe de asignar cada proceso hijo a la variable task_child para obtener la información de id y nombre..

Se debe

Módulo RAM

Biblioteca

```
#include <linux/init.h>

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/sys.h>

#include <linux/mm.h>

#include <linux/proc_fs.h>

#include <linux/seq_file.h>
```

Declaración de bibliotecas que son necesarias para la obtención y manipulación de los datos de los módulos de kernel.

Variables

```
int factorBytes;
```

Variable que contendrá el factor a multiplicar la ram total y ram utilizada.

```
long total_ram;
```

Variable que contendrá el total de la ram.

```
long free_ram;
```

Variable que contendrá la ram disponible a utilizar.

```
int used_ram;
```

Variable que contiene la ram utilizada.

```
struct sysinfo info;
```

Variable que se utiliza para obtener la información de la ram.

Lógica

Se escribe la información en formato de json donde contendrá un solo objeto con los campos de total ram en KB, ram disponible en KB, ram usada en KB.

Se utiliza la función si_meminfo donde se envía por parámetro por referencia a la variable info que es de tipo sysinfo, está variable al regresar tendrá la información de la ram y se utilizará el campo mem_unit de info para obtener el factor de bytes, el campo totalram donde dirá la cantidad total de ram a este valor se el debe de multiplicar el factor de bytes y dividir dentro de 1000 para tener el valor de KB, el campo freeram donde dirá la cantidad de ram libre a este valor se le debe de multiplicar el factor de bytes y dividir dentro de 1000 para tener el valor de KB y para calcular used_ram se resta la cantidad de total_ram con free_ram.

Go

Bibliotecas

```
import (
    "bufio"
    "encoding/json"
    "fmt"
    "io/ioutil"
    "log"
    "net/http"
    "os"
    "os/exec"
    "path/filepath"
    "strconv"
    "strings"
    "time"
)
```

Bibliotecas para la utilización de lectura y manipulación de los datos de los módulos escritos en c.

Struct

```
type ArbolInfo struct {
    id      int
    nombre  string
    hijos   []ArbolInfo
}
```

Representa al árbol de procesos donde están escritos los padres e hijos.

```
type ProcInfo struct {
    Pid                         int
```

```
    Nombre                      string
    Usuario                     int
    Estado                      string
    MemoriaUtilizada            int
    PorcentajeMemoriaUtilizada float64
}
```

Representa el estado de un proceso del sistema operativo.

```
type cpuDatos struct {
    Procs       []ProcInfo
    Memoria     int
    Total       int
    Ejecucion   int
    Suspendidos int
    Detenidos   int
    Zombies     int
}
```

Json escrito en el módulo de cpu_grupo18.

```
type ProcsInfo struct {
    Running  int
    Sleeping int
    Stopped  int
    Zombie   int
    Other    int
    Total    int
}
```

Representa la sumatoria del sistema operativo según su estado.

```
type MemoInfo struct {
    TotalKb     uint64
    AvailableKb uint64
    UsedKb      uint64

    TotalMb     float64
    AvailableMb float64
    UsedMb      float64
}
```

Representa el estado de la ram.

```
type MemoDatos struct {
   TotalKb      uint64
   AvailableKb uint64
   UsedKb       uint64
}
```
Representa los datos del módulo de mem_grupo18.

Funciones

```
func main() {
```
En esta función se encuentran las direcciones de las páginas url que están disponibles para obtener información obtenida y procesada de los módulos de kernel escritos en c.

url	función
/arbol	arbolHandler
/procsinfo	procsInfoHandler
/procs	procsHandler
/proc	procHandler
/cpu	cpuHandler
/killproc	KillProcHandler
/memo	memoHandler

Se especifica dónde va a estar servido el front-end de producción en este caso en una carpeta llamada static, se utilizará la dirección base / y el puerto a escuchar será el 8080.

```
func killProcHandler(w http.ResponseWriter, r *http.Request) {
```
Dentro del request se obtiene el parámetro pid que será el proceso a eliminar. Se utiliza la función GetProcInfo dónde se envía como parámetro por referencia procInfo que será la variable que almacenará la información del proceso a eliminar y el parámetro pid que será el parámetro que se desea eliminar. En caso exista error se reporta un http.Error indicando el error. Se utiliza el comando exec.Command donde se envía el parámetro a ejecutar en este caso kill y el parámetro a eliminar. En caso exista error se reporta un http.Error indicando el error. Se genera un json a partir de la estructura del proceso, en caso exista error se reporta un http.Error indicando el error, y se envía como respuesta de que todo se realizó correctamente.

```
func procsInfoHandler(w http.ResponseWriter, r *http.Request) {
```

Se utiliza la función GetcpuDatos y se asigna el resultado a una variable de tipo cpuDatos, en caso de error se reporta un http.Error indicando el error. Se le asigna los valores correspondiente a una variable de tipo ProcsInfo y por último se crea un json a partir de esa estructura, en caso de error se reporta un http.Error indicando el error, y se envía como respuesta de que todo se realizó correctamente.

```
func procHandler(w http.ResponseWriter, r *http.Request) {
```

Se obtiene del request un parámetro que es el identificador que se desea obtener la información, se utiliza la función GetProcInfo donde se envía una variable por referencia de tipo procInfo y el identificador del proceso que se desea obtener información, en caso de error se reporte un un http.Error indicando el error, se convierte un json utilizando json.Marshal, en caso de error se reporta un http.error de lo contrario se responde con el json indicando que todo salió bien.

```
func arbolHandler(w http.ResponseWriter, r *http.Request) {
```

Se indica la dirección del módulo que se desea abrir en este caso es /proc/cpu_arbol_grupo18 se utiliza la biblioteca ioutil y la función ReadFile para cargar el archivo, en caso de error se reporta un http.error en caso todo esté correcto se envía como respuesta el json encontrado en el archivo.

```
func memoHandler(w http.ResponseWriter, r *http.Request) {
```

Se utiliza la función GetMemInfo para obtener la información del módulo de memoria y se le asigna el resultado a memoInfo de tipo MemoInfo, en caso de error se reporta un http.error indicando el error, se realiza un casteo a float64 los parámetros AvailableKb de memoInfo y TotalKb luego se divide en 1000 para tener la información en MB se le asigna a memoInfo en AvailableMb y TotalMB respectivamente y para usedMb en memoInfo se realiza una resta de TotalMB y AvailableMB. Se obtiene un json del struct utilizando json.Marshal en caso de error se reporta un http.error indicando el error de lo contrario se envía el json en respuesta indicando que todo funcionó correctamente.

```
func cpuHandler(w http.ResponseWriter, r *http.Request) {
```

Se utiliza la función getCPUSample para obtener los resultados del cpu y se le asigna a la variable idle0 y total0 posteriormente se espera 1 segundo para volver obtener los resultados de cpu y se le asigna en esta ocasión a idle1 y total1, luego se obtiene el diferencial de idle restando idle1 con idle0 casteando el resultado a float64, luego se obtiene la diferencia de total restando total1 con total0 casteando el resultado a float64 y por último el uso de cpu se obtiene calculando la resta de totalticks con idleticks dividido entre totalticks.

```
func getCPUSample() (idle, total uint64) {
```

Se utiliza la función ReadFile de la biblioteca ioutil para leer el archivo /proc/stat en caso de error se reporta un http.error indicando el error, se realiza un split al contenido por saltos de

líneas. se recorre el arreglo resultante por cada línea, a cada línea se obtienen los campos por medio de la función Field y se pregunta si el campo en la posición 0 es cpu en caso es verdadero se obtiene el número de campos y se recorre el número de campos encontrados se obtiene el valor de cada campo con parseUnit a 64 se incrementa el total con val y se pregunta si la iteración es igual a 4 en caso sea cierto idle toma el valor de idle.

```
func GetMemInfo() (MemoInfo, error) {
```

Se declara las variables err de tipo error, la variable mem de tipo MemoDatos, la variable m de tipo MemoInfo, se utiliza la variable GetMemDatos y el resultado se asigna a mem y err, en caso que haya error se retorna una estructura vacía de MemoInfo y el error de lo contrario se obtiene los valores de AvailableKb, TotalKb y usedKb de mem y se le asigna a AvailableKb, TotalKb y UsedKb de m respectivamente, se retorna m y err.

```
func GetMemDatos() (MemoDatos, error) {
```

Se utiliza la variable path para almacenar la dirección del archivo a cargar en este caso será la dirección /proc/mem_grupo18 se utiliza la función ReadFile de la biblioteca ioutil para cargar el archivo a file si en caso haya error se retorna una estructura MemoDatos vacía y el error de lo contrario se declara la variable mem de tipo MemoDatos y se utiliza json.Unmarshal para obtener la estructura a partir de un json en caso de error se retorna una estructura vacía de MemoDatos y el error. De caso contrario se retorna la variable mem y nil en lugar de algún error.

```
func GetcpuDatos() (cpuDatos, error) {
```

Se utiliza la variable path para almacenar la dirección del archivo a cargar en este caso será la dirección /proc/cpu_grupo18 se utiliza la función ReadFile de la biblioteca ioutil para cargar el archivo a file si en caso haya error se retorna una estructura cpuDatos vacía y el error de lo contrario se declara la variable mem de tipo cpuDatos y se utiliza json.Unmarshal para obtener la estructura a partir de un json en caso de error se retorna una estructura vacía de cpuDatos y el error. De caso contrario se retorna la variable cpu y nil en lugar de algún error.

Front-end y máquina

Para el front-end utilizamos angular versión 9 debido a que levanta de forma rápida y sencilla una página funcional y lista para modificar los componentes a nuestro gusto. Usamos Angular material para darle el estilo y comportamiento a ciertas secciones de la página. Se crearon varias url según la página podría obtener la información por ejemplo esta /procs, /cpu, /memo, /procsinfo, /killproc y /arbol, pero toda la información solicitada está en /.

La máquina que se está utilizando es de google cloud es una C2 standard-4 con 4 cpu virtuales y 16Gb de ram, 100 GB de almacenamiento y un sistema operativo linux distribución ubuntu 18.04, es necesario habilitar el puerto http específicamente el 8080.