Sistemas Operativos 1	**Raul Alexander Xiloj Lopez**
Auxiliar Fernando Mazariegos	**201612113**

Documentación

Módulos

Los módulos fueron desarrollados en el lenguaje de programación “C”.

- Para compilarlo y generar los archivos necesarios entre ellos el .ko (que es el que se introduce en el kernel) se utilizó el comando make
- Para montar un módulo en el kernel se utilizo el comando `sudo insmod [nombre].ko`
- Para desmontar un módulo del kernel se utilizó el comando `sudo rmmod [nombre].ko`

RAM

Para el módulo de la memoria Ram se hizo uso del struct **sysinfo** que obtiene la información de la RAM como por ejemplo, el total de ella, la memoria libre. Se guardó en formato json para tener una mayor facilidad de presentarlo en el frontend ya que se trabajó con JS.

```
15  struct sysinfo inf;
16
17  static int proc_ram_data(struct seq_file * file, void *v){
18      si_meminfo(&inf);
19      unsigned long totalram = (inf.totalram*4);
20      unsigned long freeram = (inf.freeram*4);
21      seq_printf(file, "{\n");
22      seq_printf(file,"\"total_memo\": %lu,\n",totalram/1024);
23      seq_printf(file,"\"free_memo\": %lu,\n", freeram/1024);
24      seq_printf(file,"\"used_memo\": %lu\n", ((totalram - freeram)*100)/totalram);
25      seq_printf(file, "}\n");
26      return 0;
27  }
```

Sistemas Operativos 1	Raul Alexander Xiloj Lopez
Auxiliar Fernando Mazariegos	201612113

CPU

Para el módulo del CPU se utilizó el struct **task_struct** para obtener los procesos del sistema (los que se encuentran en ejecución, suspendidos, zombie y detenidos). De igual manera se guardó en formato json para tener una mayor facilidad de presentarlo ya que solo se parsea.

```
static int proc_cpu_msg(struct seq_file * file, void *v){
    int running = 0, sleeping = 0, zombie = 0, stopped = 0;
    seq_printf(file, "{\n\"processes\":[ ");
    for_each_process(task){//Marco para iterar a traves de cada tarea en SO
        seq_printf(file, "{");
        seq_printf(file,"\"pid\":%d,\n",task->pid);
        seq_printf(file,"\"name\":\"%s\",\n",task->comm);
        seq_printf(file, "\"user\": %d,\n",task->cred->uid);
        seq_printf(file,"\"state\":%ld,\n",task->state);

        seq_printf(file,"\"children\":[");
        list_for_each(list, &task->children){
            task_child = list_entry(list, struct task_struct, sibling);
            seq_printf(file,"%d,",task_child->pid);
        }
        seq_printf(file,"]");

        switch(task->state){
            case 0:
                running++;
                break;
            case 1:
            case 1026:
                sleeping++;
                break;
            case 4:
                zombie++;
                break;
            case 5:
                stopped++;
                break;
            default:
                break;
        }
        seq_printf(file, "},\n");
    }
    seq_printf(file, "],\n");
    seq_printf(file, "\"running\":%d,\n",running);
    seq_printf(file, "\"sleeping\":%d,\n",sleeping);
    seq_printf(file, "\"zombie\":%d,\n",zombie);
    seq_printf(file, "\"stopped\":%d,\n",stopped);
```

Sistemas Operativos 1	**Raul Alexander Xiloj Lopez**
Auxiliar Fernando Mazariegos	**201612113**

Server

El servidor se realizó en el lenguaje de programación “go” y fue de gran ayuda ya que con él se obtuvo de manera fácil el porcentaje de utilización del cpu. Y la comunicación en tiempo real se realizó por medio de websocket.

Se realizó leyendo el archivo “/proc/stat” el cual contiene información sobre la actividad del kernel y en este caso información sobre cpu.

```go
func getCPU() (idle, total uint64) {
    contents, err := ioutil.ReadFile("/proc/stat")
    if err != nil {
        return
    }
    lines := strings.Split(string(contents), "\n")
    for _, line := range lines {
        fields := strings.Fields(line)
        if fields[0] == "cpu" {
            numFields := len(fields)
            for i := 1; i < numFields; i++ {
                val, err := strconv.ParseUint(fields[i], 10, 64)
                if err != nil {
                    fmt.Println("Error: ", i, fields[i], err)
                }
                total += val //tally up all the numbers to get total ticks
                if i == 4 {
                    idle = val
                }
            }
            return
        }
    }
    return
}
```

```go
func readerCPU(ws *websocket.Conn) {
    for { //for loop that run forever (like a while(true))

        messageType, _, err := ws.ReadMessage()
        if err != nil {
            log.Println(err)
            return
        }

        for {
            idle0, total0 := getCPU()
            time.Sleep(1 * time.Second)
            idle1, total1 := getCPU()

            idleTicks := float64(idle1 - idle0)
            totalTicks := float64(total1 - total0)
            cpuUsage := 100 * (totalTicks - idleTicks) / totalTicks
            fmt.Printf("CPU usage is %f%% ", cpuUsage)
            ws.WriteMessage(messageType, []byte(strconv.FormatFloat(cpuUsage, 'f', 2, 64)))
            time.Sleep(1 * time.Second)
        }

    }
}
```

Sistemas Operativos 1	Raul Alexander Xiloj Lopez
Auxiliar Fernando Mazariegos	201612113

FRONTEND