

## Proceso de creación de módulos

Para crear los módulos fue necesario escribir un archivo en c para el módulo de cpu y para el módulo de memoria. Los cuales se describen a continuación:

### [memo\\_201504399.c](#)

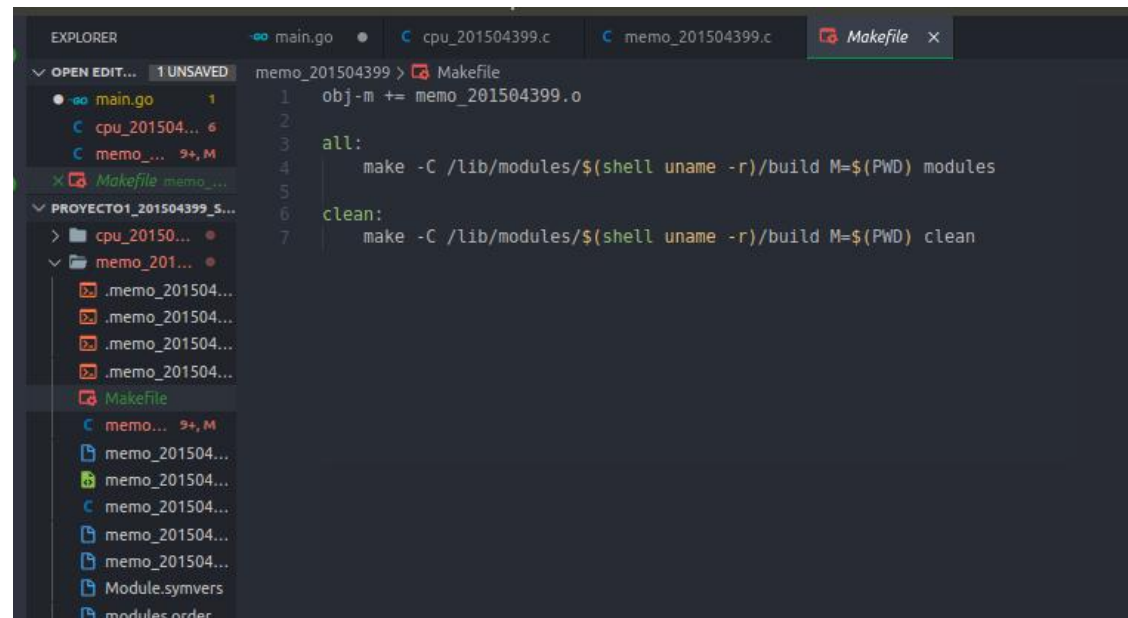
```
main.go • cpu_201504399.c • memo_201504399.c • inf
memo_201504399.c > C memo_201504399.c > [ej] inf
1 #include <linux/proc_fs.h>
2 #include <linux/seq_file.h>
3 #include <asm/uaccess.h>
4 #include <linux/hugetlb.h>
5 #include <linux/module.h>
6 #include <linux/init.h>
7 #include <linux/kernel.h>
8 #include <linux/fs.h>
9 #include <linux/swap.h>
10 #include <asm/page.h>
11 #include <linux/mmzone.h>
12 #include <linux/cpumask.h>
13 #include <linux/kernel_stat.h>
14
15 #define BUFSIZE 150
16
17 MODULE_LICENSE("GPL");
18 MODULE_DESCRIPTION("Escribir informacion de la memoria ram.");
19 MODULE_AUTHOR("201504399");
20
21 struct sysinfo inf;
22 static int escribir_archivo(struct seq_file * archivo, void *v){
23     si_meminfo(&inf);
24     long total_memoria = (inf.totalram * 4 );
25     long memoria_libre = (inf.freeram * 4);
26     long buffer = (inf.bufferram);
27     long cached= (global_node_page_state(NR_FILE_PAGES) * 2 )- inf.bufferram ;
28     long memoria_utilizada = total_memoria - (memoria_libre+buffer +cached);
29     long porcentaje =(memoria_utilizada *100)/total_memoria);
30     //Total Memory - (Free + Buffers + Cached)
31
32     seq_printf(archivo, "{\n");
33     seq_printf(archivo, "\tTotal: \"%lu\", \n", total_memoria /1024); //total memoria ram
34     seq_printf(archivo, "\tUso: \"%lu\", \n", memoria_utilizada/1024); // total memoria consumida
35     seq_printf(archivo, "\tPorcentaje: \"%li\" \n", porcentaje); //porcentaje de consumo
36     seq_printf(archivo, " } \n");
37     return 0;
38 }
```

1. Se importaron las librerías necesarias.
2. Se creó la función escribir archivo para poder escribir en el módulo la información de la ram, se estructuró como formato json. Los datos a cargar son: total de memoria ram, el cual se obtiene de la propiedad totalram de la estructura sysinfo, para obtener la memoria utilizada se aplicó la fórmula  $MemoriaUtilizada = MemoriaTotal - (MemoriaLibre + Buffer + Cached)$

```
39
40 static int al_abrir(struct inode *inode, struct file *file){
41     return single_open(file, escribir_archivo, NULL);
42 }
43
44 static struct file_operations operaciones =
45 {
46     .open = al_abrir,
47     .read = seq_read
48 };
49
50 int iniciar(void){ //modulo de inicio
51     proc_create("memo_201504399", 0, NULL, &operaciones);
52     printk(KERN_INFO "%s", "Cargando modulo.\n");
53     printk(KERN_INFO "%s", "201504399\n");
54
55     return 0;
56 }
57
58 void salir(void){ //modulo salida
59
60     remove_proc_entry("memo_201504399", NULL);
61     printk(KERN_INFO "%s", "Removiendo modulo.\n");
62     printk(KERN_INFO "Sistemas Operativos 1.\n");
63 }
64
65 module_init(iniciar);
66 module_exit(salir);
67
```

3. El método iniciar crea el módulo con el nombre memo\_201504399, se ejecuta el método al\_abrir, el cual llama al método de escritura del archivo e imprime en el buffer el mensaje Cargando módulo y el número de carnet.
4. El método salir borra el módulo e imprime los mensajes en el buffer.

Para poder crear el archivo .ko se necesita el archivo make file que crea los archivos necesarios para crear el archivo tipo módulo (ko), el archivo makefile cuenta con dos comandos, make clean para borrar todos los archivos y make all para crearlos.



The screenshot shows a code editor with a dark theme. The Explorer panel on the left shows a project structure with files like main.go, cpu\_201504399.c, memo\_201504399.c, and a Makefile. The main editor area shows the content of the Makefile, which includes rules for building the module (obj-m += memo\_201504399.o) and cleaning the build directory (clean: make -C /lib/modules/\$(shell uname -r)/build M=\$(PWD) clean).

```
1 obj-m += memo_201504399.o
2
3 all:
4     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
5
6 clean:
7     make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Después de ejecutar make all, se puede cargar el módulo con el comando **sudo insmod memo\_201504399.ko** y con el comando **dmesg** se pueden ver los mensajes del buffer. Y para descargar el módulo se utiliza el comando **sudo rmmod memo\_201504399**

### cpu\_201504399.c

El archivo cpu\_201504399.c tiene los mismos métodos y funciones que el archivo memo.

En la función escribir archivo escribe dentro del módulo cpu\_201504399 la información de todos los procesos del cpu, para esto se necesitaron 2 task\_struct uno para procesos padre y otro para hijos y un list\_head para realizar la lista de procesos hijos.

Para verificar los procesos fue necesario utilizar un for\_each\_process el cual revisa los procesos padre y escribe en el archivo todas sus propiedades, para revisar si tiene hijos se realizó un list\_for\_each.

La información se estructuró en formato json y los hijos son un arreglo de formato json dentro de esa estructura.

El código se muestra a continuación:

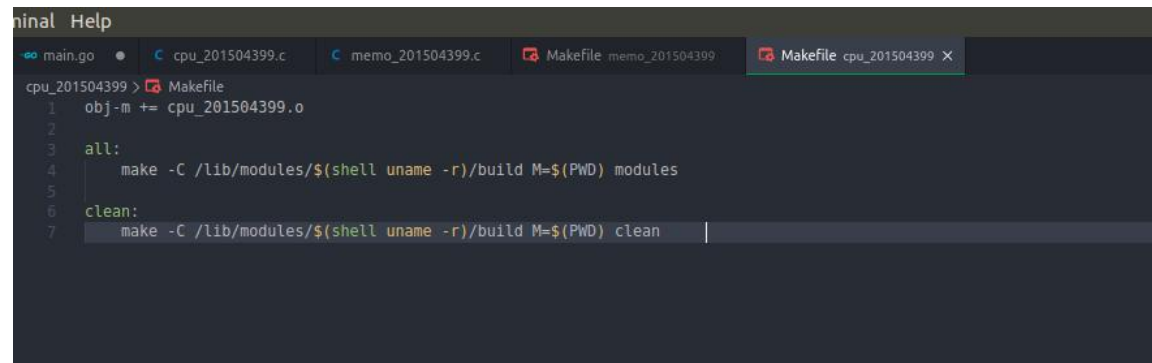
```
main.go • cpu_201504399.c x memo_201504399.c Makefile
cpu_201504399 > C cpu_201504399.c > for_each_process(task)
12 struct task_struct *task; //estructura definifa en sched.h para tareas /procesos
13 struct task_struct *task_child; //estructura necesaria para iterar a través de tareas secundarias
14 struct list_head *list; //estructura necesaria para recorrer la lista en cada tarea -> estructura de hijos.
15
16 static int escribir_archivo(struct seq_file * archivo, void *v){
17     int contador =0;
18     int contadoraux=0;
19     #define Convert(x) ((x) << (PAGE_SHIFT - 10))
20     seq_printf(archivo, " [ ");
21
22     for_each_process(task){
23         if(contador>0){
24             seq_printf(archivo," , \n");
25         }
26         seq_printf(archivo," {\n");
27         seq_printf(archivo, "\t\"PID\": \"%d\",", task->pid);
28         seq_printf(archivo, "\t\"PROCESO\": \"%s\",", task->comm);
29         seq_printf(archivo, "\t\"ESTADO\": \"%ld\",", task->state);
30         if(task->mm!=NULL){
31             seq_printf(archivo, "\t\"MEMORIA\": \"%ld\",", Convert(get_mm_rss(task->mm)));
32         } else {
33             seq_printf(archivo, "\t\"MEMORIA\": \"%0\",");
34         }
35         seq_printf(archivo, "\t\"USUARIO\": \"%d\",", __kuid_val(task->real_cred->uid));
36
37         seq_printf(archivo, "\t\"HIJOS\": [ ");
38         contadoraux=0;
39         list_for_each(list, &task->children){
40             if(contadoraux>0){
41                 seq_printf(archivo," , \n");
42             }
43             task_child = list_entry(list, struct task_struct, sibling);
44             seq_printf(archivo, "\t\t{\n");
45             seq_printf(archivo, "\t\t\t\"PID\": \"%d\",", task_child->pid);
46             seq_printf(archivo, "\t\t\t\"PROCESO\": \"%s\",", task_child->comm);
47             seq_printf(archivo, "\t\t\t\"ESTADO\": \"%ld\",", task_child->state);
48             if(task_child->mm!=NULL){
49                 seq_printf(archivo, "\t\t\t\"MEMORIA\": \"%ld\",", Convert(get_mm_rss(task_child->mm)));
50             } else {
```

```
33         seq_printf(archivo, "\t\t\t\"MEMORIA\": \"%0\",");
34     }
35     seq_printf(archivo, "\t\"USUARIO\": \"%d\",", __kuid_val(task->real_cred->uid));
36
37     seq_printf(archivo, "\t\"HIJOS\": [ ");
38     contadoraux=0;
39     list_for_each(list, &task->children){
40         if(contadoraux>0){
41             seq_printf(archivo," , \n");
42         }
43         task_child = list_entry(list, struct task_struct, sibling);
44         seq_printf(archivo, "\t\t{\n");
45         seq_printf(archivo, "\t\t\t\"PID\": \"%d\",", task_child->pid);
46         seq_printf(archivo, "\t\t\t\"PROCESO\": \"%s\",", task_child->comm);
47         seq_printf(archivo, "\t\t\t\"ESTADO\": \"%ld\",", task_child->state);
48         if(task_child->mm!=NULL){
49             seq_printf(archivo, "\t\t\t\"MEMORIA\": \"%ld\",", Convert(get_mm_rss(task_child->mm)));
50         } else {
51             seq_printf(archivo, "\t\t\t\"MEMORIA\": \"%0\",");
52         }
53         seq_printf(archivo, "\t\t\t\"USUARIO\": \"%d\",", __kuid_val(task_child->real_cred->uid));
54         seq_printf(archivo, "\t\t\t} \n");
55         contadoraux++;
56     }
57     seq_printf(archivo," ] \n");
58     contador++;
59     seq_printf(archivo,"} \n");
60
61 }
62 seq_printf(archivo, " ] ");
63 #undef k
64 return 0;
65 }
66
```

María de Los Angeles Herrera Sumalé  
201504399

El método iniciar crea el módulo con el nombre `cpu_201504399`, se ejecuta el método `al_abrir`, el cual llama al método de escritura del archivo e imprime en el buffer el mensaje Cargando módulo y el número de carnet.  
El método salir borra el módulo e imprime los mensajes en el buffer.

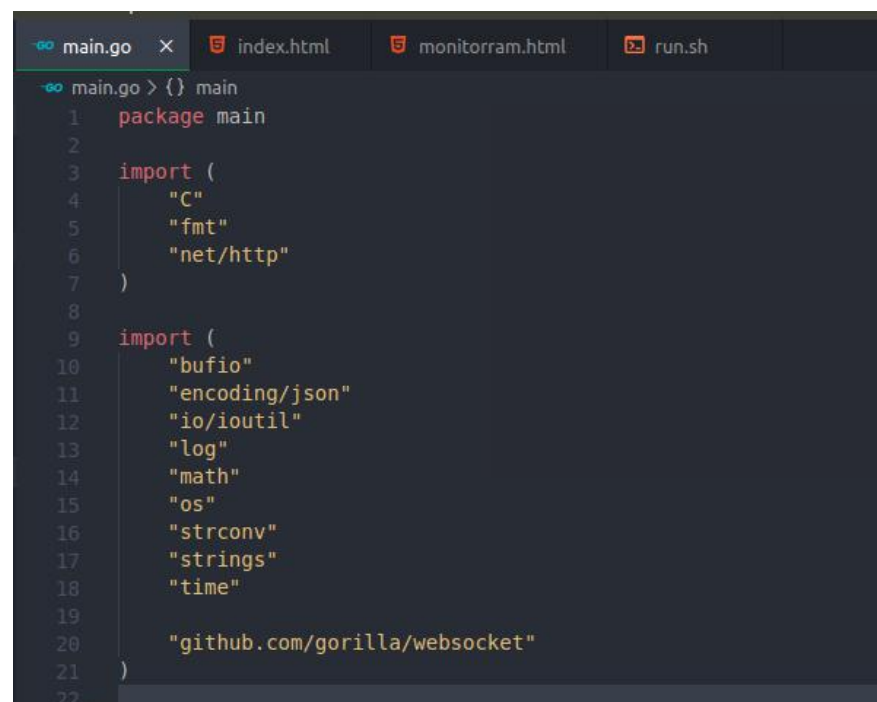
Para poder crear el archivo `.ko` se necesita el archivo make file que crea los archivos necesarios para crear el archivo tipo módulo (ko), el archivo makefile cuenta con dos comandos, `make clean` para borrar todos los archivos y `make all` para crearlos.



```
terminal Help
cpu_201504399 > Makefile
1  obj-m += cpu_201504399.o
2
3  all:
4      make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
5
6  clean:
7      make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Después de ejecutar `make all`, ya se puede cargar el módulo con el comando **`sudo insmod cpu_201504399.ko`** y con el comando **`dmesg`** se pueden ver los mensajes del buffer. Y para descargar el módulo se utiliza el comando **`sudo rmmod cpu_201504399`**

## GO



```
main.go > {} main
1  package main
2
3  import (
4      "C"
5      "fmt"
6      "net/http"
7  )
8
9  import (
10     "bufio"
11     "encoding/json"
12     "io/ioutil"
13     "log"
14     "math"
15     "os"
16     "strconv"
17     "strings"
18     "time"
19
20     "github.com/gorilla/websocket"
21 )
22
```

Para el servidor utilicé el lenguaje de programación Go, importando librerías para peticiones http, utilizar formato json, abrir archivos, tiempo para manejar la comunicación y por último la librería para los websocket, los cuales son un canal de comunicación que siempre estarán enviando la información, con cierto tiempo se espera.



```
main.go x index.html monitorram.html run.sh
main.go > {} main > cpuStruct
22
23 type memoria struct {
24     Total    string `json:"Total"`
25     Uso      string `json:"Uso"`
26     Porcentaje string `json:"Porcentaje"`
27 }
28
29 type datos struct {
30     Ejecucion  string `json:"Ejecucion"`
31     Suspendidos string `json:"Suspendidos"`
32     Detenidos  string `json:"Detenidos"`
33     Zombie     string `json:"Zombie"`
34     Total      string `json:"Total"`
35 }
36
37 type hijo struct {
38     PID      string `json:"PID"`
39     PROCESO  string `json:"PROCESO"`
40     ESTADO  string `json:"ESTADO"`
41     MEMORIA  string `json:"MEMORIA"`
42     USUARIO  string `json:"USUARIO"`
43 }
44
45 type proceso struct {
46     PID      string `json:"PID"`
47     PROCESO  string `json:"PROCESO"`
48     ESTADO  string `json:"ESTADO"`
49     MEMORIA  string `json:"MEMORIA"`
50     USUARIO  string `json:"USUARIO"`
51     HIJOS    []hijo `json:"HIJOS"`
52 }
53
54 type objcpu struct {
55     ListaProcesos []proceso `json:"ListaProcesos"`
56     General       []datos  `json:"General"`
57 }
58
59 type cpuStruct struct {
60     Porcentaje float64 `json:"porcentaje"`
61 }
```

Structs necesarios para obtener los datos de los respectivos módulos, cpu y memoria ram.

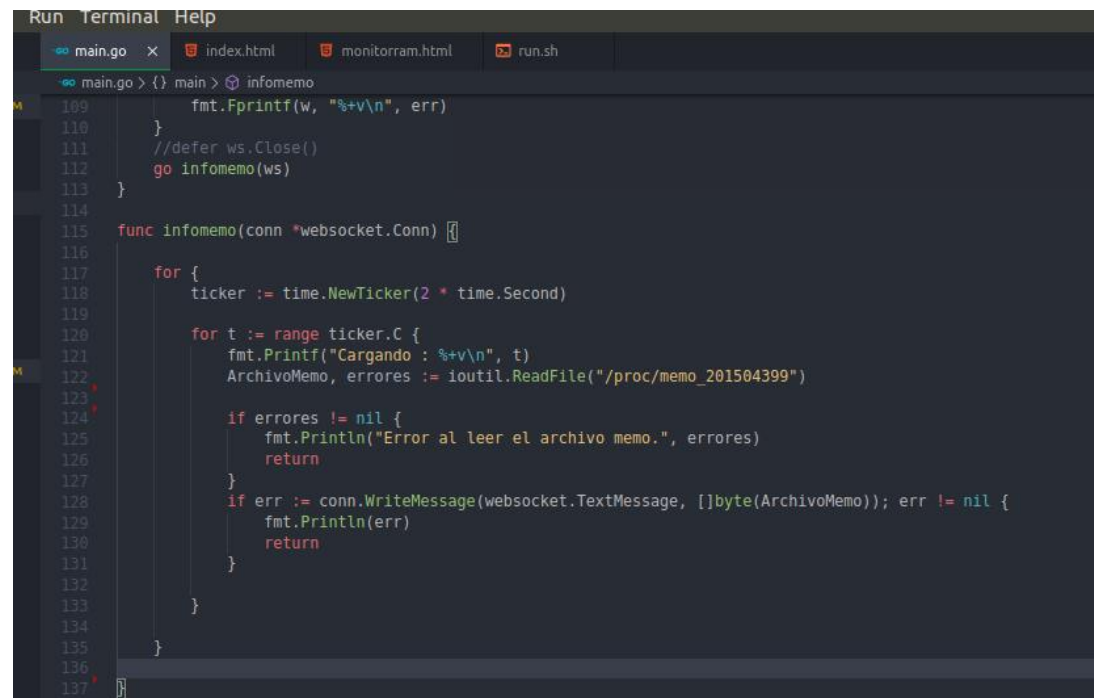
```
Terminal Help
main.go x index.html monitorram.html run.sh
main.go > {} main > cpuStruct
67
68 func Upgrade(w http.ResponseWriter, r *http.Request) (*websocket.Conn, error) {
69     actualiza.CheckOrigin = func(r *http.Request) bool { return true }
70
71     // websocket connection
72     ws, err := actualiza.Upgrade(w, r, nil)
73     if err != nil {
74         log.Println(err)
75         return ws, err
76     }
77     // returnswebsocket connection
78     return ws, nil
79 }
80 func Upgrade2(w http.ResponseWriter, r *http.Request) (*websocket.Conn, error) {
81     actualiza.CheckOrigin = func(r *http.Request) bool { return true }
82
83     // websocket connection
84     ws, err := actualiza.Upgrade(w, r, nil)
85     if err != nil {
86         log.Println(err)
87         return ws, err
88     }
89     // returnswebsocket connection
90     return ws, nil
91 }
92 func Upgrade3(w http.ResponseWriter, r *http.Request) (*websocket.Conn, error) {
93     actualiza.CheckOrigin = func(r *http.Request) bool { return true }
94
95     // websocket connection
96     ws, err := actualiza.Upgrade(w, r, nil)
97     if err != nil {
98         log.Println(err)
99         return ws, err
100    }
101    // returnswebsocket connection
102    return ws, nil
103 }
104
105 > func serveWs(w http.ResponseWriter, r *http.Request) {--
```

Métodos para actualizar la conexión del websocket, retorn la nueva conexión.

```
Run Terminal Help
main.go x index.html monitorram.html run.sh
main.go > {} main > cpuStruct
162 }
163 func infocpu(conn *websocket.Conn) {
164
165     for {
166         ticker := time.NewTicker(2 * time.Second)
167
168         for t := range ticker.C {
169             fmt.Printf("Cargando : %v\n", t)
170             Archivocpu, err := ioutil.ReadFile("/proc/cpu_201504399")
171             //listaprocesos := []proceso{}
172             objetocpu := objcpu{}
173             err = json.Unmarshal(Archivocpu, &objetocpu)
174             if err != nil {
175                 fmt.Println("Error al leer el archivo cpu.", err)
176                 return
177             }
178             fmt.Println(objetocpu.General)
179
180             if err := conn.WriteMessage(websocket.TextMessage, []byte(Archivocpu)); err != nil {
181                 fmt.Println(err)
182                 return
183             }
184
185         }
186     }
187 }
188 //time.Sleep(2000 * time.Millisecond)
189 }
```

María de Los Angeles Herrera Sumalé  
201504399

Método que obtiene los datos del cpu, leyendo el módulo ya cargado en el proc, enviándolo como objeto Json a través de la ruta en la que se comunicará el websocket.

A screenshot of a Go IDE window titled 'Run Terminal Help'. The editor shows a Go file named 'main.go' with tabs for 'main.go', 'index.html', 'monitortram.html', and 'run.sh'. The code defines a function 'infomemo' that takes a 'websocket.Conn' as an argument. Inside the function, a ticker is set to 2 seconds. A range loop iterates over the ticker, printing 'Cargando' and reading a file from '/proc/memo\_201504399'. It then checks for errors and writes the data to the websocket connection. The function is called from the 'main' function.

```
109     fmt.Fprintf(w, "%v\n", err)
110 }
111 //defer ws.Close()
112 go infomemo(ws)
113 }
114
115 func infomemo(conn *websocket.Conn) {
116     for {
117         ticker := time.NewTicker(2 * time.Second)
118         for t := range ticker.C {
119             fmt.Printf("Cargando : %v\n", t)
120             ArchivoMemo, errores := ioutil.ReadFile("/proc/memo_201504399")
121
122             if errores != nil {
123                 fmt.Println("Error al leer el archivo memo.", errores)
124                 return
125             }
126             if err := conn.WriteMessage(websocket.TextMessage, []byte(ArchivoMemo)); err != nil {
127                 fmt.Println(err)
128                 return
129             }
130         }
131     }
132 }
133
134
135
136
137
```

Método que obtiene los datos de la memoria, leyendo el módulo ya cargado en el proc, enviándolo como objeto Json a través de la ruta en la que se comunicará el websocket.

A screenshot of a Go IDE window showing the 'main' function. The code sets up an HTTP server on port 3000, serving static files from the 'Frontend' directory. It also registers a websocket endpoint at '/memo' and calls 'serveWs'. The server is started with 'ListenAndServe' on port 3000.

```
241 }
242 func main() {
243     fmt.Println("Puerto 3000")
244     fs := http.FileServer(http.Dir("./Frontend"))
245     http.Handle("/", fs)
246     http.HandleFunc("/memo", serveWs)
247     go otro()
248     go otro2()
249     log.Fatal(http.ListenAndServe(":3000", nil))
250 }
251
```

Para levantar el servidor le indicamos la ruta donde se encuentra el archivo css , el cual está en la carpeta Frontend, junto con las páginas html utilizadas para este proyecto.  
El servidor está disponible en el puerto 3000