MEMORIA P1 y P2.

Practica 1.

Para la práctica 1 usaremos la configuración por defecto de todo y aplicaremos los siguientes comandos:

- cd /usr/src/linux-2.4.20-8
- make mrproper /*configura el entorno*/
- make menuconfig (y guardamos sin modificar nada) /*crea el archivo .config*/

- [modificar la variable linux_banner] (archivo /init/version.c) para que nos muestre un mensaje personalizado al iniciar Linux de nuevo y podamos ver que la recompilación ha sido correcta.

- make dep /*crea los archivos de dependencias entre componentes .depend.*/
- make clean /*eliminará los pasos no finalizados de construcciones previas para que no interfieran con la actual*/
- make bzImage /*construye la parte estática. Su misión es compilar el nuevo kernel.*/
- make modules /*construye los módulos de carga dinámica para aquellas partes que se indicaron en la configuración con “m”*/
- make modules_install /*mueve los módulos a su ubicación final, es decir, /lib/modules/<versión kernel>/kernel/<tipo de modulo>*/

/*Reubicar el kernel en el directorio adecuado*/
- cp System.map /boot/System.map-2.4.20-8 (¿Sobreescribir?--> SI)
- cp arch/i386/boot/bzImage /boot/vmlinuz-2.4.20-8 (¿Sobreescribir?--> SI)

Una vez que iniciamos de nuevo linux (con COD: linuxlocal), comprobamos que se ha cargado correctamente la imagen con la siguiente orden:

cat ../var/log/dmesg | grep “Algo que pusieramos en la variable linux_banner”

Practica 2. Bloque I

Hay que tener en cuenta los errores consideraciones del guión de prácticas:

- Para el primer bloque, la función “llevamos()” tiene que llamarse “consultar()”.
- Las rutas relativas pueden ser a partir de (son equivalentes):
  - /usr/src/linux-2.4
  - /usr/src/linux-2.4.20-8
- El módulo cliente admite argumentos (veremos como se hace en el código).
- La variable xtime está declarada en include/linux/sched.h y no en /kernel/sched.h

Este es el código de los archivos que vamos a usar en este primer bloque:

Acumulador.c

```
#define MODULE
#include <linux/module.h>
#include <linux/time.h>

extern struct timeval xtime;
static int sum=0;

void acumular (int i){

        sum = sum+i;

}

int consultar(){

        return sum;
}


int init_module(){

        sum = 0;
        printk (“Modulo acumulador insertado en %d\n”,xtime.tv_sec);
        return 0;
}

int cleanup_module(){

        consultar();
        printk (“Modulo extraido en %d\n”,xtime.tv_sec);
        printk (“Modulo acumulador ha acumulado %d\n”, consultar());
        return 0;
}
```

Cliente.c

```
#define MODULE
#include <linux/module.h>
#include <linux/time.h>

extern struct timeval xtime;
MODULE_PARM (ac, “i”);


int init_module(){

        printk (“Modulo cliente insertado en %d\n”, xtime.tv_sec);
        acumular (ac);
        return 0;
}

int cleanup_module(){

        printk (“Suma = %d\n”, consultar());
        printk (“Modulo cliente extraido en %d\n”, xtime.tv_sec);
        return 0;
}
```

Para cargar estos módulos hay que hacer lo siguiente:

- Compilar los módulos con:
    - gcc –I /usr/src/Linux-2.4/incude –c acumulador.c –o acumulador
    - gcc –I /usr/src/Linux-2.4/incude –c cliente.c –o cliente

- Tener en cuenta que primero tiene que insertarse el módulo acumulador y luego el módulo cliente.
- Insertar el acumulador con:
    - insmod –f acumulador
- Insertar el módulo cliente con:
    - insmod –f cliente ac=”valor_que_queramos_acumular”

Practica 2. Bloque II.

Hay que tener en cuenta los errores consideraciones del guión de prácticas:

- Omitir #include <sys/types.h>
- Cambiar #include<sys/mman.h> por #include</usr/include/sys/mman.h>
- Comentar las líneas 30 y 38 del archivo /usr/include/sys/mman.h
    - #typedef __off_t off_t;
    - #typedef __mode_t mode_t
- Podemos omitir #include<linux/malloc.h>
- Usar las funciones get___32 en lugar de get___ Esto es así porque si ejecutamos strace whoami, podemos ver la traza de ejecución de la orden whoami y ver que llamadas al sistema intervienen. En este caso concreto, es las llamada geteuid32(),

El código del módulo que vamos a crear es el siguiente:

Changewhoami.c

```
#define MODULE
#define __KERNEL__
#include <linux/module.h>
#include <linux/kernel.h>
#include <asm/unistd.h>
#include <sys/syscall.h>
#include <asm/fcntl.h>
#include <linux/types.h>
#include <linux/dirent.h>
#include </usr/include/sys/mman.h>
#include <linux/string.h>
#include <linux/fs.h>
/*#include <linux/malloc.h>*/

extern void* sys_call_table[]; /*para acceder a la tabla de llamadas del sistema*

/*Establecemos los id originales*/
int (*orig_geteuid)();
int (*orig_getuid)();
int (*orig_getgid)();
int (*orig_getegid)();

/*Asigno una id que nunca va a ser la mia. Con la orden cat/etc/passwd podemos ver el nombre de usuario, la contraseña encriptada, el id y el id del grupo al que pertenece dicho usuario. Asi que podemos seleccionar uno de esa lista que sabemos que nunca va a ser el nuestro*/

int hacked_geteuid(){

        return 47;
}
```

```
int hacked_getuid(){

	return 47;

}

int hacked_getgid(){

	return 47;

}

int hacked_getegid(){

	return 47;

}

/*Al llamar al módulo se hackea la id*/

int init_module(void){

   orig_geteuid = sys_call_table[SYS_geteuid32];
   sys_call_table[SYS_geteuid32] = hacked_geteuid;

   orig_getuid = sys_call_table[SYS_getuid32];
   sys_call_table[SYS_getuid32] = hacked_getuid;


   orig_getgid = sys_call_table[SYS_getgid32];
   sys_call_table[SYS_getgid32] = hacked_getgid;


   orig_getegid = sys_call_table[SYS_getegid32];
   sys_call_table[SYS_getegid32] = hacked_getegid;


}

/*Reestablecer los id originales cuando se retira el módulo*/

void cleanup_module(void){

    sys_call_table[SYS_geteuid32] = orig_geteuid;
    sys_call_table[SYS_getuid32] = orig_getuid;
    sys_call_table[SYS_getgid32] = orig_getgid;
    sys_call_table[SYS_getegid32] = orig_getegid;

}
```

En esta práctica también es necesario recompilar el kernel, pero teniendo en cuenta que para la realización de las prácticas, vamos a usar los equipos de la Escuela que tienen instalada distribución Red Hat. Esta distribución, por motivos de seguridad, no exporta la tabla de llamadas al sistema.
Para solventar este inconveniente deberemos exportarla nosotros mismos. Los pasos a seguir son:

- Editamos /usr/src/linux-2.4/kernel/ksyms.c y añadimos la línea:
  - EXPORT_SYMBOL(sys_call_table)
  -
- Recompilamos el kernel de linux con las mismas ordenes que en la práctica 1:

  - cd /usr/src/linux-2.4.20-8
  - make mrproper /*configura el entorno*/
  - make menuconfig (y guardamos sin modificar nada) /*crea el archivo .config*/

  - [modificar la variable linux_banner] (archivo /init/version.c) para que nos muestre un mensaje personalizado al iniciar Linux de nuevo y podamos ver que la recompilación ha sido correcta.

  - make dep /*crea los archivos de dependencias entre componentes .depend.*/
  - make clean /*eliminará los pasos no finalizados de construcciones previas para que no interfieran con la actual*/
  - make bzImage /*construye la parte estática. Su misión es compilar el nuevo kernel.*/
  - make modules /*construye los módulos de carga dinámica para aquellas partes que se indicaron en la configuración con “m”*/
  - make modules_install /*mueve los módulos a su ubicación final, es decir, /lib/modules/<versión kernel>/kernel/<tipo de modulo>*/

  /*Reubicar el kernel en el directorio adecuado*/
  - cp System.map /boot/System.map-2.4.20-8 (¿Sobreescribir?--> SI)
  - cp arch/i386/boot/bzImage /boot/vmlinuz-2.4.20-8 (¿Sobreescribir?--> SI)

- Arrancar con COD: linuxlocal
- Compilar el módulo changewhoami:
  - gcc –I /usr/src/Linux-2.4/incude –c changewhoami.c –o fakeid
- Cargar el módulo:;
  - insmod –f fakeid

Ahora cuando hagamos un whoami, nos debe salir el id que hemos asignado en el programa anterior.