

## SO2 - LABORATORIO 2 - MINIX

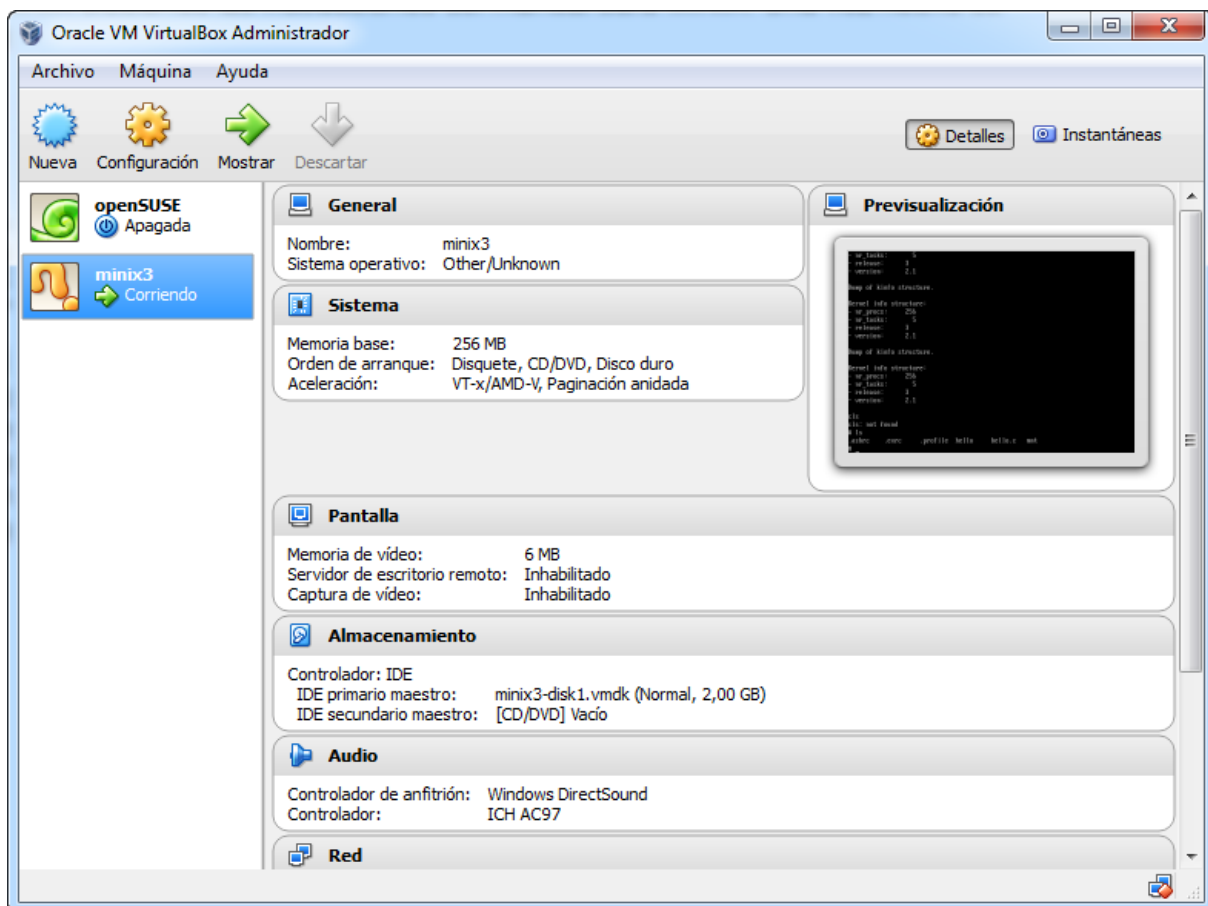
### 1-Objetivos

Los principales objetivos de este Laboratorio son:

- Introducir en el kernel código para contar los mensajes enviados por los procesos.
- Hacer visible a los procesos de usuario, a través del IS (Servidor de Información), los contadores de mensajes enviados por los procesos.
- Mostrar por pantalla el número de mensajes enviados por los procesos, utilizando para ello la tecla de función F9 que no está en uso.

### Procedimiento

\*Lo primero a realizar es configurar Virtual Box e importar la imagen virtualizada de MINIX siguiendo las instrucciones del manual para MINIX. Una vez hecho se arranca con la imagen recién importada.



## 2-.TOMA DE CONTACTO CON EL SERVIDOR DE INFORMACIÓN (IS)

*MINIX 3 ofrece la posibilidad de obtener diferentes visualizaciones de información por pantalla a través de las teclas de función F1-F12. La tecla F1 se usa para obtener un volcado con información acerca de la tabla de procesos y la tecla F2 para obtener información sobre los mapas de memoria de los procesos. La tecla F9 no tiene asignada ninguna funcionalidad.*

Entrando en el fichero `dmp.c` localizado en el directorio `/usr/src/servers/is/` podemos observar las distintas funciones y a qué teclas se encuentran arraigadas. Para que F9 tenga la misma función que F10 no hace falta más que definirlo con la misma función.

```
* The entry points into this file are
* map_unmap_fkeys:  register or unregister function key maps with TTY
* do_fkey_pressed:   handle a function key pressed notification
*/

#include "inc.h"
#include <minix/vm.h>

struct hook_entry {
    int key;
    void (*function)(void);
    char *name;
} hooks[] = {
    { F1,  proctab_dmp, "Kernel process table" },
    { F3,  image_dmp,  "System image" },
    { F4,  privileges_dmp, "Process privileges" },
    { F5,  monparams_dmp, "Boot monitor parameters" },
    { F6,  irqtab_dmp,  "IRQ hooks and policies" },
    { F7,  kmessages_dmp, "Kernel messages" },
    { F8,  vm_dmp,      "VM status and process maps" },
    { F9,  kenv_dmp,    "Kernel parameters" },
    { F10, kenv_dmp,    "Kernel parameters" },
    { SF1, mproc_dmp,   "Process manager process table" },
    { SF2, sigaction_dmp, "Signals" },
}
```

Nos cambiamos al directorio `/usr/src/releasetools/` y tecleamos "make install". Al volver arrancar comprobamos si la función fue añadida.

```
Dump of kinfo structure.

Kernel info structure:
- nr_procs:    256
- nr_tasks:    5
- release:     3
- version:     2.1

Oprimiendo F10 Dump of kinfo structure.

Kernel info structure:
- nr_procs:    256
- nr_tasks:    5
- release:     3
- version:     2.1

Oprimiendo F9 Dump of kinfo structure.

Kernel info structure:
- nr_procs:    256
- nr_tasks:    5
- release:     3
- version:     2.1
```

### 3 NÚMERO TOTAL DE MENSAJES ENVIADOS POR LOS PROCESOS

#### 3.1 Definir y mantener el array de contadores de mensajes.

*Registrar la antigüedad de un proceso, para llevar la cuenta de cuántos mensajes envía cada proceso declararemos un array del mismo tamaño que la tabla de procesos y de tipo "long" ya que, como veremos más adelante, pueden llegar a registrarse muchos envíos de mensajes. A este array le denominaremos numMensajes y se definirá en el fichero **/usr/src/kernel/proc.h**.*

El contador de mensajes enviados por un proceso debe incrementarse cada vez que se llame al procedimiento mini\_send que se encuentra en el fichero **/usr/src/kernel/proc.c**. Si quisiéramos hacer un control más exacto de los mensajes enviados, deberíamos tener en cuenta cuáles son realmente recibidos pero, por simplicidad, vamos a mantener la solución indicada.

Primero editamos el fichero **proc.h** localizado en **usr/src/kernel/** y definimos el array numMensajes.

```
#define END_PROC_ADDR (&proc[NR_TASKS + NR_PROCS])

#define proc_addr(n)      (&(proc[NR_TASKS + (n)]))
#define proc_nr(p)        ((p)->p_nr)

#define isokprocn(n)       ((unsigned) ((n) + NR_TASKS) < NR_PROCS + NR_TASKS)
#define isemptyn(n)        isemptyp(proc_addr(n))
#define isemptyp(p)        ((p)->p_rts_flags == RTS_SLOT_FREE)
#define iskernelp(p)        ((p) < BEG_USER_ADDR)
#define iskerneln(n)        ((n) < 0)
#define isuserp(p)         isusern((p) >= BEG_USER_ADDR)
#define isusern(n)          ((n) >= 0)
#define isrootsysn(n)       ((n) == ROOT_SYS_PROC_NR)

#ifndef __ASSEMBLY__

EXTERN struct proc proc[NR_TASKS + NR_PROCS]; /* process table */
EXTERN long numMensajes[NR_TASKS + NR_PROCS]; /* array definido */
int mini_send(struct proc *caller_ptr, endpoint_t dst_e, message *m_ptr,
               int flags);

#endif /* __ASSEMBLY__ */

#endif /* PROC_H */
```

Nos trasladamos al fichero main.c y damos valor inicial 0 al array numMensajes.

```

Máquina Ver Dispositivos Ayuda
tty: ignoring unrecognized escaped scancode 0x5b
Tiene la opción autocaptura de teclado habilitada. Esto causará que la máquina virtual capture automáticamente el teclado cada
#include "arch_proto.h"

#ifdef CONFIG_SMP
#include "smp.h"
#endif
#ifdef USE_WATCHDOG
#include "watchdog.h"
#endif
#include "spinlock.h"

/* dummy for linking */
char *** _penviron;

/* Prototype declarations for PRIVATE functions. */
static void announce(void);

void bsp_finish_booting(void)
{
    int i;
    for(int j=0; j<sizeof(NR_TASKS); j++){
        numMensajes[j]=0;
    }
}

```

Por último, en el apartado, tocamos el procedimiento mini\_send localizado en proc.c para incrementar el contador cada vez que llaman a dicho procedimiento.

```

{
/* Send a message from 'caller_ptr' to 'dst'. If 'dst' is blocked waiting
 * for this message, copy the message to it and unblock 'dst'. If 'dst' is
 * not waiting at all, or is waiting for another source, queue 'caller_ptr'.
 */
    register struct proc *dst_ptr;
    register struct proc **xpp;
    int dst_p;
    ;
    dst_p = _ENDPOINT_P(dst_e);
    dst_ptr = proc_addr(dst_p);
    numMensajes[0]=numMensajes[0]+1;
    if (RTS_ISSET(dst_ptr, RTS_NO_ENDPOINT))
    {
        return EDEADSRCDST;
    }

    /* Check if 'dst' is blocked waiting for this message. The destination's
     * RTS_SENDING flag may be set when its SENDREC call blocked while sending.
     */
    if (WILLRECEIVE(dst_ptr, caller_ptr->p_endpoint)) {
        int call;
proc.c: 1900 lines, 57250 characters.
#

```

Vamos al directorio `/usr/src/releasetools` y tecleamos: `make install`. Luego reiniciamos para comprobar que todo funciona con normalidad.

```
APIC disabled, using legacy PIC

MINIX 3.2.1. Copyright 2012, Urije Universiteit, Amsterdam, The Netherlands
MINIX is open source software, see http://www.minix3.org
Initiating legacy i8253 timer
CPU 0 freq 2562 MHz
kernel: selecting amd syscall ipc style
Started UFS: 8 worker thread(s)
Root device name is /dev/c0d0p0s0
/dev/c0d0p0s0: clean
/dev/c0d0p0s0 is mounted on /
none is mounted on /proc
/dev/c0d0p0s2: clean
/dev/c0d0p0s1: clean
size on /dev/imgrd set to 0kB
Multiuser startup in progress ...
Starting hotplugging infrastructure... done.
Starting services: random lance inet printer ipc vbox.
Starting daemons: update cron syslogd.
Starting networking: dhcpd nonamed.
Local packages (start): done.
Minix Release 3 Version 2.1 (console)

10.0.2.15 login: Mon Nov 28 17:54:41 GMT 2016
*****
```

### 3.2 Hacer que el núcleo pueda suministrar al IS el array de contadores.

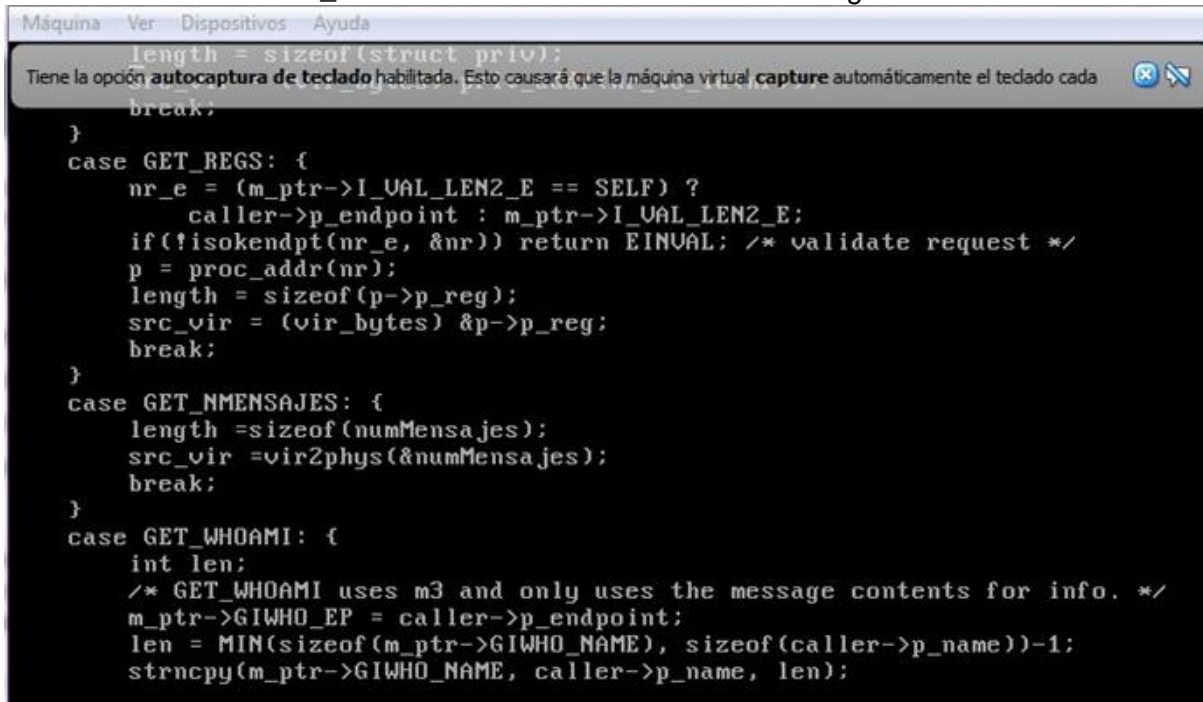
*Para que el array de contadores, residente en el espacio del núcleo, pueda ser reclamado por el IS, el núcleo ofrece una llamada al sistema “**sys\_getinfo**” que ya proporciona mucha información del sistema y que podemos ampliar para que ahora también permita recuperar el array de contadores de mensajes.*

*Los servicios de información disponibles a través de “**sys\_getinfo**” pueden consultarse en el fichero `/usr/src/include/minix/com.h` bajo el epígrafe “Field names for SYS\_GETINFO”, que define constantes como `GET_PROCTAB` para indicar que se desea reclamar la información de la tabla de procesos del núcleo. Nosotros definiremos una nueva constante que llamaremos `GET_NMENSAJES`.*

Para definir la constante GET\_NMENSAJES nos dirigimos a `/usr/src/include/minix/` en donde editamos el archivo `com.h`.

```
# define GET_RANDOMNESS 3 /* get randomness buffer */
# define GET_MONPARAMS 4 /* get monitor parameters */
# define GET_KENV 5 /* get kernel environment string */
# define GET_IRQHOOKS 6 /* get the IRQ table */
# define GET_PRIVTAB 8 /* get kernel privileges table */
# define GET_KADDRESSES 9 /* get various kernel addresses */
# define GET_SCHEDINFO 10 /* get scheduling queues */
# define GET_PROC 11 /* get process slot if given process */
# define GET_MACHINE 12 /* get machine information */
# define GET_LOCKTIMING 13 /* get lock()/unlock() latency timing */
# define GET_BIOSBUFFER 14 /* get a buffer for BIOS calls */
# define GET_LOADINFO 15 /* get load average information */
# define GET_IRQACTIDS 16 /* get the IRQ masks */
# define GET_PRIV 17 /* get privilege structure */
# define GET_HZ 18 /* get HZ value */
# define GET_WHOAMI 19 /* get own name, endpoint, and privileges */
# define GET_RANDOMNESS_BIN 20 /* get one randomness bin */
# define GET_IDLETS 21 /* get cumulative idle time stamp counter */
# define GET_CPUINFO 23 /* get information about cpus */
# define GET_REGS 24 /* get general process registers */
# define GET_NMENSAJES 25 /* DEFINIMOS NUEVA CONSTANTE_
#define I_ENDPT m7_i4 /* calling process (may only be SELF) */
#define I_VAL_PTR m7_p1 /* virtual address at caller */
#define I_VAL_LEN m7_i1 /* max length of value */
```

Después, editamos el archivo `do_getinfo.c` ubicado en `/usr/src/kerne/system/` e incluimos el nuevo caso `case GET_NMENSAJES` como se indica en la imagen:



```
length = sizeof(struct priv);
break;
}
case GET_REGS: {
    nr_e = (m_ptr->I_VAL_LEN2_E == SELF) ?
        caller->p_endpoint : m_ptr->I_VAL_LEN2_E;
    if(!isokendpt(nr_e, &nr)) return EINVAL; /* validate request */
    p = proc_addr(nr);
    length = sizeof(p->p_reg);
    src_vir = (vir_bytes) &p->p_reg;
    break;
}
case GET_NMENSAJES: {
    length = sizeof(numMensajes);
    src_vir = vir2phys(&numMensajes);
    break;
}
case GET_WHOAMI: {
    int len;
    /* GET_WHOAMI uses m3 and only uses the message contents for info. */
    m_ptr->GIWHO_EP = caller->p_endpoint;
    len = MIN(sizeof(m_ptr->GIWHO_NAME), sizeof(caller->p_name))-1;
    strncpy(m_ptr->GIWHO_NAME, caller->p_name, len);
```

\*Por último en esta sección incluimos un sinónimo corto para la función "sys\_getinfo"



para ligarla a GET\_NMENSAJES, anteriormente definida.

```
#define sys_getloadinfo(dst) sys_getinfo(GET_LOADINFO, dst, 0,0,0)
#define sys_getmachine(dst) sys_getinfo(GET_MACHINE, dst, 0,0,0)
#define sys_getcpuinfo(dst) sys_getinfo(GET_CPUINFO, dst, 0,0,0)
#define sys_getproctab(dst) sys_getinfo(GET_PROCTAB, dst, 0,0,0)
#define sys_getprivtab(dst) sys_getinfo(GET_PRIVTAB, dst, 0,0,0)
#define sys_getproc(dst,nr) sys_getinfo(GET_PROC, dst, 0,0, nr)
#define sys_getrandomness(dst) sys_getinfo(GET_RANDOMNESS, dst, 0,0,0)
#define sys_getrandom_bin(d,b) sys_getinfo(GET_RANDOMNESS_BIN, d, 0,0,b)
#define sys_getimage(dst) sys_getinfo(GET_IMAGE, dst, 0,0,0)
#define sys_getirqhooks(dst) sys_getinfo(GET_IRQHOOKS, dst, 0,0,0)
#define sys_getirqactids(dst) sys_getinfo(GET_IRQACTIDS, dst, 0,0,0)
#define sys_getmonparams(v,v1) sys_getinfo(GET_MONPARAMS, v,v1, 0,0)
#define sys_getschedinfo(v1,v2) sys_getinfo(GET_SCHEDINFO, v1,0, v2,0)
#define sys_getpriv(dst, nr) sys_getinfo(GET_PRIV, dst, 0,0, nr)
#define sys_getidlepsc(dst) sys_getinfo(GET_IDLETSC, dst, 0,0,0)
#define sys_getregs(dst,nr) sys_getinfo(GET_REGS, dst, 0,0, nr)
#define sys_getnmessages(dst) sys_getinfo(GET_NMENSAJES,dst,0,0,0)
int sys_getinfo(int request, void *val_ptr, int val_len, void *val_ptr2,
int val_len2);
int sys_whoami(endpoint_t *ep, char *name, int namelen, int
*priv_flags);

/* Signal control. */
int sys_kill(endpoint_t proc_ep, int sig);
```

### 3.3 Conectar la nueva funcionalidad a la tecla de función F9

*Para comprobar finalmente si todo lo realizado funciona de manera adecuada, conectaremos la función hecha en los pasos anterior a la tecla F9 la cual, al oprimirla debería de darnos la cantidad de mensajes realizados al momento de pulsar dicha tecla.*

Definimos el prototipo para la nueva función en `/usr/src/servers/is` al modificar `proto.h`.

```
/* Function prototypes. */

/* main.c */
int main(int argc, char **argv);

/* dmp.c */
void map_unmap_fkeys(int map);
int do_fkey_pressed(message *m);
void mapping_dmp(void);
void vm_dmp(void);

/* dmp_kernel.c */
void proctab_dmp(void);
void procstack_dmp(void);
void privileges_dmp(void);
void image_dmp(void);
void irqtab_dmp(void);
void kmessages_dmp(void);
void monparams_dmp(void);
void kenv_dmp(void);
void nmessages_dmp(void);

/* dmp_pm.c */
void mproc_dmp(void);
proto.h: unmodified: line 1
```

Cuando debemos modificar `dmp_kernel.c` para agregar la función que ha de imprimir la

cantidad total de mensajes puede ocurrir que nuestro array numMensajes no sea definido causando error a la hora de compilar.

La solución está escrita en la imagen en donde volvemos a definirla en el mismo archivo dmp\_kernel.c. Una vez hecho esto, el error desaparece.

```
/* Declare some local dump procedures. */
static char *proc_name(int proc_nr);
static char *s_traps_str(int flags);
static char *s_flags_str(int flags);
static char *p_rts_flags_str(int flags);

/* Some global data that is shared among several dumping procedures.
 * Note that the process table copy has the same name as in the kernel
 * so that most macros and definitions from proc.h also apply here.
 */
struct proc proc[NR_TASKS + NR_PROCS];
long numMensajes[NR_TASKS + NR_PROCS];
struct priv priv[NR_SYS_PROCS];
struct boot_image image[NR_BOOT_PROCS];

extern struct minix_kerninfo *_minix_kerninfo;

/*=====
 *                               kmessages_dmp                               *
 *=====*/
void kmessages_dmp()
{
    struct kmessages *kmess; /* get copy of kernel messages */
    static char print_buf[KMESS_BUF_SIZE+1]; /* this one is used to print */
}
```

En la siguiente imagen se imprime el primer elemento del array, solo buscamos saber la cantidad de mensajes en total. Si quisieramos separarlos por procesos, la función debería variar imprimiendo cada elemento.

Al ser del tipo long, hace falta el índice "%ld" para mostrarlo en pantalla. Caso contrario se precipita a un error.

```
if ((r = sys_getmachine(&machine)) != OK) {
    printf("IS: warning: couldn't get copy of kernel machine struct: %d\n",
r);
    return;
}

printf("Dump of kinfo structure.\n\n");
printf("Kernel info structure:\n");
printf("- nr_procs:      %3u\n", kinfo.nr_procs);
printf("- nr_tasks:      %3u\n", kinfo.nr_tasks);
printf("- release:        %.6s\n", kinfo.release);
printf("- version:        %.6s\n", kinfo.version);
printf("\n");
}
void nmensajes_dmp(){
    printf(" %ld", numMensajes[0]);
}

/*=====
 *                               s_flags_str                               *
 *=====*/
static char *s_flags_str(int flags)
```

Por último queda configurar la tecla F9 para que llamen a la función nmensajes\_dmp. Al igual que en la primera sección del trabajo, esto se logra sin más complicaciones como la mostrada a continuación.



```
char *name;
} hooks[] = {
    { F1,   proctab_dmp, "Kernel process table" },
    { F3,   image_dmp,  "System image" },
    { F4,   privileges_dmp, "Process privileges" },
    { F5,   monparams_dmp, "Boot monitor parameters" },
    { F6,   irqtab_dmp,  "IRQ hooks and policies" },
    { F7,   kmessages_dmp, "Kernel messages" },
    { F8,   vm_dmp,      "VM status and process maps" },
    { F9,   nmensajes_dmp, "Mensajes"},
    { F10,  kenv_dmp,    "Kernel parameters" },
    { SF1,  mproc_dmp,   "Process manager process table" },
    { SF2,  sigaction_dmp, "Signals" },
    { SF3,  fproc_dmp,   "Filesystem process table" },
    { SF4,  dtab_dmp,    "Device/Driver mapping" },
    { SF5,  mapping_dmp, "Print key mappings" },
    { SF6,  rproc_dmp,   "Reincarnation server process table" },
    { SF8,  data_store_dmp, "Data store contents" },
    { SF9,  procstack_dmp, "Processes with stack traces" },
};

/* Define hooks for the debugging dumps. This table maps function keys
 * onto a specific dump and provides a description for it.
 */
```

### Prueba:

Una vez guardado todos los cambios (yendo a `/usr/src/releasetools` y allí escribir `make install`). Reiniciamos para comprobar que el contador funciona de forma correcta al oprimir F9.

```
10.0.2.15 login: root
Last login: Mon Nov 28 21:41:20 2016 on console
Copyright (c) 2012, Urije Universiteit, Amsterdam, The Netherlands

To install additional packages, run 'pkgin'.

To install packages from the online package repository, if you have a
working network connection from MINIX: first do a 'pkgin update' to
update the list of available packages, and then do a 'pkgin' to get a
list of commands. For example, 'pkgin install vim' installs the 'vim'
package, and 'pkgin available' will list all available packages.

To install packages from the installation CD: same, but use pkgin_cd.
To switch to the online repository, do 'pkgin update' again. To install
all packages, do pkgin_all.

MINIX 3 supports multiple virtual terminals. Just use ALT+F1, F2, F3
and F4 to navigate among them.

For more information on how to use MINIX 3, see the wiki:
http://wiki.minix3.org.

#
# 22108_
```