

PRÁCTICA 1

Toma de contacto con el software de prácticas





ÍNDICE

1	OBJETIVOS.....	3
2	ARRANCANDO MINIX3 SOBRE LA MÁQUINA VIRTUAL QEMU.....	3
2.1	Descripción del entorno de prácticas	3
2.2	Arranque de MINIX3 sobre la máquina virtual QEMU	4
2.3	Intercambio de archivos entre Windows y MINIX.....	10
2.4	Intercambio en caliente de archivos entre Windows y MINIX.....	14
2.5	Uso de un terminal a través de la línea de comunicación serie (TCP/IP).....	21
3	MODIFICACIÓN PUNTUAL DEL NÚCLEO DE MINIX.....	27
4	INCORPORACIÓN DE UNA BIBLIOTECA AL NÚCLEO DE MINIX.....	36
5	ANEXOS.....	42
5.1	Manual del editor “elvis”	42
5.2	Manual del editor “mined”	43

1 OBJETIVOS

Los objetivos de esta práctica son:

- Familiarizarse con el software de prácticas.
- Familiarizarse con el trasvase de ficheros desde Windows a MINIX usando comandos básicos de UNIX que se estudian en la asignatura de [Taller de Sistemas Operativos](#).
- Recompilar MINIX 3.1.2a tras introducir cambios puntuales en el código fuente de su [núcleo \(/usr/src/kernel/\)](#) tanto en lenguaje C ([main.c](#)) como en ensamblador ([mpx386.s](#)).

2 ARRANCANDO MINIX3 SOBRE LA MÁQUINA VIRTUAL QEMU

El emulador [QEMU](#) simula una máquina (virtual) con arquitectura IA32 de manera eficiente, ejecutando el código 80x86 directamente sobre la CPU anfitriona. Sobre esta máquina virtual pueden arrancarse diversos sistemas operativos como: Linux, MINIX, Microsoft Windows, etc. QEMU es software abierto, pero existen productos similares de tipo comercial como [VMware Workstation Player](#).

Una forma de arrancar un sistema operativo como MINIX sobre una máquina virtual QEMU es utilizar una imagen de disco –un fichero preparado especialmente para representar byte a byte el contenido de un disco– de tal forma que ante cualquier fallo los efectos destructivos ocasionados queden reducidos al contenido de dicho fichero sin afectar al resto del sistema que se ejecuta directamente sobre el equipo físico.

2.1 Descripción del entorno de prácticas

Los alumnos que lo deseen podrán realizar las prácticas sobre su propio ordenador personal portátil con Windows 10 (64 bits), con el fin de aprovechar de manera mas cómoda, eficiente y conveniente el software de prácticas al ejecutarlo desde su disco duro interno.

Las prácticas se realizarán en el aula y/o CIC de la ETSISI. En el CIC están disponibles ordenadores personales con el sistema operativo Windows 10 (64 bits). Los alumnos, por el hecho de estar matriculados, disponen de una cuenta personal con nombre de usuario y contraseña para entrar en cualquiera de los ordenadores del CIC. Los alumnos que hayan cursado en primero la asignatura de TSO (Taller de Sistemas Operativos) conocen perfectamente el procedimiento. En caso de cualquier problema con la cuenta deberán dirigirse a la ventanilla sita en el pasillo del CIC, donde serán atendidos por el personal encargado de resolver ese tipo de incidencias.

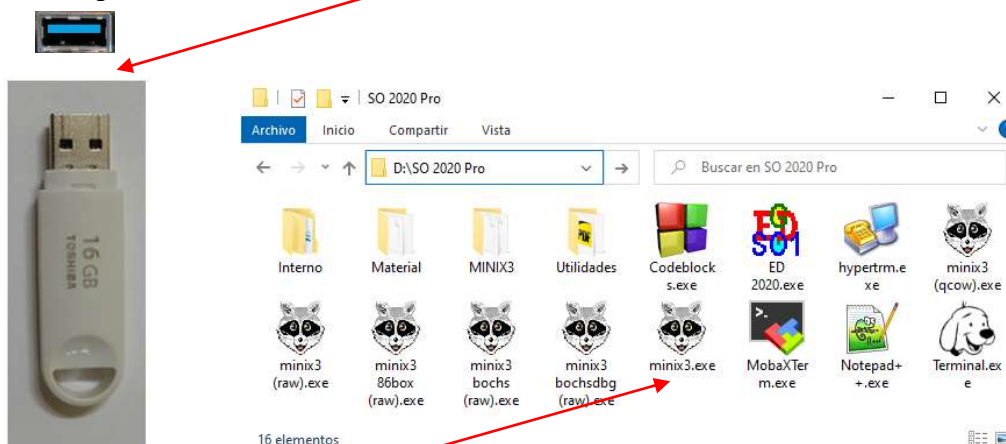
El alumno sin portátil deberá llevar a las sesiones de laboratorio un pendrive personal donde podrá guardar el software de prácticas y los programas que vaya realizando. Se proporcionará a los alumnos un fichero comprimido ([SO 2010 Pro.zip](#)) con todo lo necesario para la realización de las prácticas, de manera que puedan trabajar sobre ellas tanto dentro como fuera de los laboratorios del CIC. Para la entrega de prácticas se utilizará a veces la plataforma Moodle:

<https://moodle.upm.es/titulaciones/oficiales/login/login.php>

en la parte correspondiente al curso “Sistemas Operativos”.

2.2 Arranque de MINIX3 sobre la máquina virtual QEMU

- Encender el equipo del laboratorio (o portátil) y entrar en Windows introduciendo el nombre de usuario y contraseña. Si se usa portátil vaya al directorio con el software de prácticas. En otro caso conectar el pendrive que contiene el software de prácticas, en uno de los puertos USB.



- Pichar sobre el icono **minix3.exe**, que lanza la máquina virtual [qemu](#).

Al principio del arranque tomará control el **BIOS** (concretamente [SeaBIOS](#)), quien tras inicializar la máquina virtual y detectar la presencia del disco duro, procederá a la carga de su **MBR** en memoria, cuyo código procederá a la carga del **PBR** de la partición activa. El código del **PBR** procederá a su vez a la carga del **monitor de arranque** cediéndole el control. En ese momento se visualizará la siguiente ventana:

```
QEMU (minix3 [qcow]) - Press Ctrl+Alt+G to release grab
Machine View
SeaBIOS (version rel-1.13.0-48-gd9c812dda519-prebuilt.qemu.org)

iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+01F8F390+01EEF390 CA00

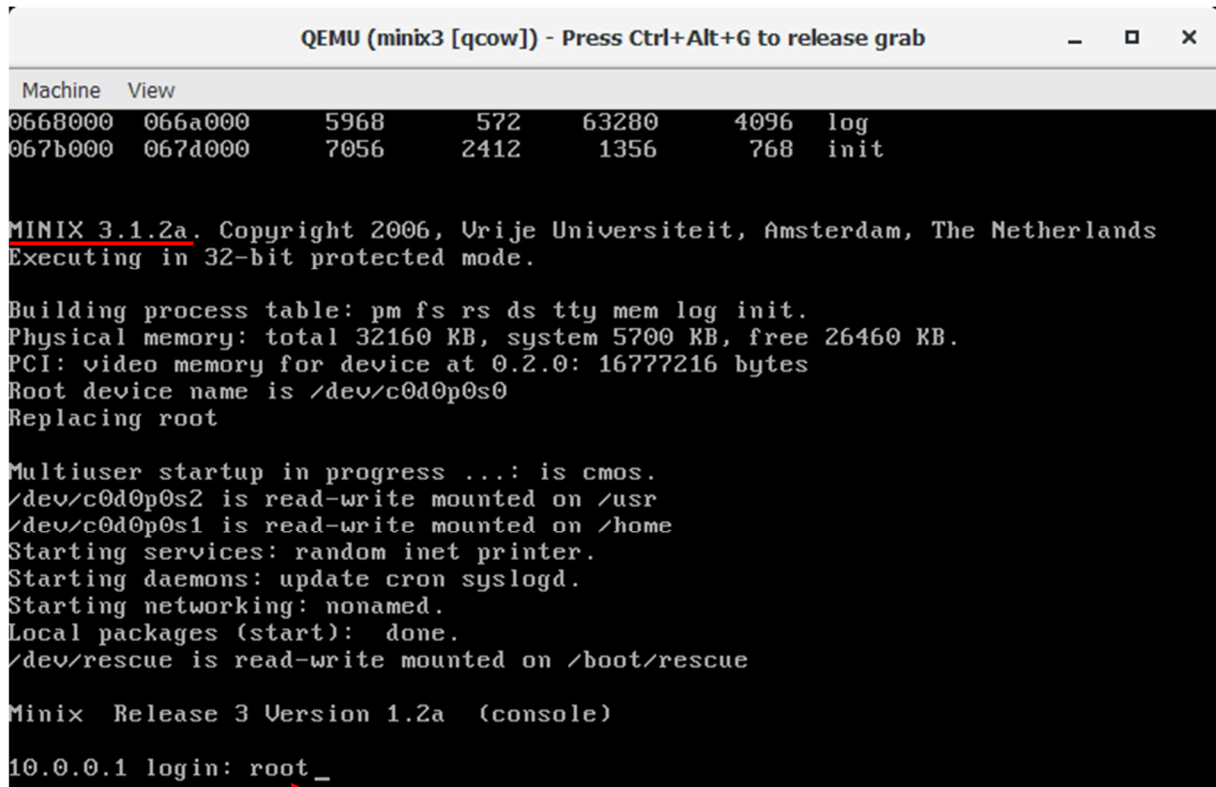
Booting from Hard Disk...
--- Welcome to MINIX 3. This is the boot monitor. ---

By default, MINIX 3 will automatically load in 3 seconds.
Press ESC to enter the monitor for special configuration.

Hit a key as follows:

 1 Start MINIX 3 (requires at least 16 MB RAM)
 2 Start Small MINIX 3 (intended for 8 MB RAM systems)
 3 Start Custom MINIX 3
```

- Las opciones que nos muestra el **monitor de arranque** de MINIX 3 nos permiten seleccionar la imagen del núcleo de MINIX que deseamos arrancar. En nuestro caso, en tanto en cuanto no modifiquemos el núcleo, sólo tendremos una imagen, la original **/boot/image/3.1.2a**, que es la que arrancará en 3 segundos sin necesidad de indicar nada. Tras presionar la tecla 1 (o esperar 3 segundos) se muestra el arranque de la [versión 3.1.2a](#) (04/05/2006) de MINIX. [Nota: la versión actual es la [3.3.0](#)]



```
QEMU (minix3 [qcow]) - Press Ctrl+Alt+G to release grab
Machine  View
0668000  066a000    5968    572   63280    4096  log
067b000  067d000    7056   2412    1356    768  init

MINIX 3.1.2a. Copyright 2006, Urije Universiteit, Amsterdam, The Netherlands
Executing in 32-bit protected mode.

Building process table: pm fs rs ds tty mem log init.
Physical memory: total 32160 KB, system 5700 KB, free 26460 KB.
PCI: video memory for device at 0.2.0: 16777216 bytes
Root device name is /dev/c0d0p0s0
Replacing root

Multiuser startup in progress ...: is cmos.
/dev/c0d0p0s2 is read-write mounted on /usr
/dev/c0d0p0s1 is read-write mounted on /home
Starting services: random inet printer.
Starting daemons: update cron syslogd.
Starting networking: nonamed.
Local packages (start): done.
/dev/rescue is read-write mounted on /boot/rescue

Minix Release 3 Version 1.2a (console)
10.0.0.1 login: root_
```

- Entrar como usuario **root** (no se nos pedirá adicionalmente ninguna palabra clave). Teclee el comando **id** para conocer la identidad del usuario que ha abierto esta sesión en el equipo (virtual). Teclee el comando **version** para mostrar el número de versión. Teclee el comando **last** para asegurarse de que es la primera sesión que se abre en esa instalación de MINIX.
- Podemos pasar a ver MINIX en pantalla completa tecleando la combinación de teclas de qemu: **Ctrl + Alt + F** (*full screen*). Repitiendo esa combinación de teclas volvemos a la visualización en ventana. Esto también puede conseguirse con la opción *View* (**Alt + V**) de la barra de menús desplegables de qemu:

```

QEMU (minix3 [qcow])
Machine View
/dev/re... ed on /boot/rescue
Minix (console)
10.0.0.
To inst... man' with the install CD still in the
drive. r you have installed it, login as root
and typ... mation about configuring X Windows, see
www.min...
If you... emory to run X Windows, standard MINIX 3
support... nals. Just use ALT+F1, F2, F3 and F4 to
navigat...
To get... t /etc/motd.
# id
uid=0(r...
# last...
root console Sat Aug 29 23:32 still logged in
reboot ~ Sat Aug 29 23:32
wtmp begins Sat Aug 29 23:32
#
  
```

- Puede comprobarse que estamos en un sistema de tipo UNIX tecleando comandos como: [pwd](#) para ver el directorio en el que se está; [ls](#), o [ls -la](#) para ver el contenido del directorio; [who](#) para saber quién está en el sistema; [cd](#) para moverse por la jerarquía de directorios; [ps](#), [ps -e](#), [ps -ef](#) o [top](#) para ver qué procesos están ejecutándose, etc.
- Es posible que en algún momento el teclado empiece a funcionar mal, escribiéndose símbolos raros en vez de los caracteres correspondientes a las teclas presionadas. En ese caso pulse la combinación de teclas **Ctrl + Alt**, y el teclado volverá a responder.
- Con la combinación de teclas **Ctrl + Alt + 2** puede conmutarse al [monitor de qemu](#) donde pueden ejecutarse comandos (del monitor) como: [help](#), [stop](#), [cont](#), [quit](#) o [info](#).

```

QEMU (minix3 [qcow])
Machine View
compat_monitor0 console
QEMU 5.1.0 monitor - type 'help' for more information
(qemu) help stop
stop -- stop emulation
(qemu) help cont
clcont -- resume emulation
(qemu) help quit
qlquit -- quit the emulator
(qemu) help info registers
info registers [-a] -- show the cpu registers (-a: all - show register info for
all cpus)
(qemu) help info block
info block [-n] [-v] [device] -- show info of one block device or all block devi
ces (-n: show named nodes; -v: show details)
(qemu) █
  
```

- En el monitor de qemu (**Ctrl + Alt + F**) tecleando el comando **info registers** podemos visualizar el contenido de los registros hardware del ordenador que configuran el estado de funcionamiento del procesador:

```

QEMU (minix3 [qcow])
Machine  View
ES =0018 00007000 0000ba97 00409300 DPL=0 DS [-WA]
CS =0030 00001000 00005e6f 00409a00 DPL=0 CS32 [-R-]
SS =0018 00007000 0000ba97 00409300 DPL=0 DS [-WA]
DS =0018 00007000 0000ba97 00409300 DPL=0 DS [-WA]
FS =000d 00007000 0000bfff 0040b300 DPL=1 DS [-WA]
GS =000d 00007000 0000bfff 0040b300 DPL=1 DS [-WA]
LDT=0078 0000a120 00000027 00408200 DPL=0 LDT
TR =0040 00009ca0 00000067 00408900 DPL=0 TSS32-au1
GDT= 00009d08 000003b7
IDT= 0000814c 000003bf
CR0=00000011 CR2=00000000 CR3=00000000 CR4=00000000
DR0=00000000 DR1=00000000 DR2=00000000 DR3=00000000
DR6=ffff0ff0 DR7=00000400
EFER=0000000000000000
FCW=037f FSW=0000 IST=01 FTW=00 MXCSR=00001f80
FPR0=0000000000000000 0000 FPR1=0000000000000000 0000
FPR2=0000000000000000 0000 FPR3=0000000000000000 0000
FPR4=0000000000000000 0000 FPR5=0000000000000000 0000
FPR6=0000000000000000 0000 FPR7=0000000000000000 0000
XMM00=00000000000000000000000000000000 XMM01=00000000000000000000000000000000
XMM02=00000000000000000000000000000000 XMM03=00000000000000000000000000000000
XMM04=00000000000000000000000000000000 XMM05=00000000000000000000000000000000
XMM06=00000000000000000000000000000000 XMM07=00000000000000000000000000000000
(qemu) █

```

En concreto se aprecia que el [registro de control 0 \(CR0\)](#) tiene el valor 0x00000011 y el [registro base de la tabla de páginas \(CR3\)](#) tiene el valor 0x00000000. Eso significa que el procesador está trabajando en [modo protegido](#) de 32 bits (las máquinas actuales son todas ya de 64 bits) utilizando [segmentación](#) pero no [paginación](#) (que si ya tiene MINIX versión 3.3.0). Además los [segmentos](#) correspondientes a código, datos y pila del sistema operativo tienen DPL 0, ejecutándose por tanto en el anillo con máximo nivel de privilegio, con drivers en el anillo de nivel de privilegio 1 y programas de usuario en el anillo de mínimo nivel de privilegio (3).

- Para volver a MINIX hay que teclear **Ctrl + Alt + 1**.
- Ya en MINIX y ubicados en el directorio de entrada, **/root**, vamos a crear un fichero de texto que denominaremos **nombre**, cuyo contenido sea una línea con el nombre y los apellidos del alumno, seguido de su número de matrícula con formato **xx0999**. Para crear un fichero en MINIX se recomienda utilizar el editor de texto [mined](#) que se utiliza de manera parecida a editores de Windows [teclea **mined nombre**. **Ctrl-X** para salir de mined]. También puede utilizarse el editor [elvis](#), tipo **vi** (aunque sobre MINIX 3.1.2a+qemu+Windows quizás dé algún problema). Si se desea pueden consultarse los anexos para ver una breve descripción de ambos editores. Alternativamente utilice para la consulta el manual en línea [man](#) con los comandos: **man mined** o **man elvis**.
- Comprobar que se ha creado el fichero correctamente. Para ello, mostrarlo por pantalla con el comando: **cat nombre**.

```

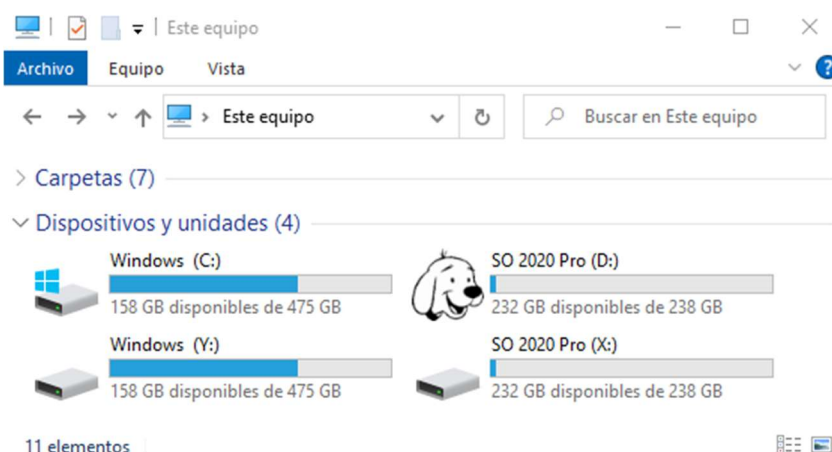
QEMU (minix3 [qcow])
Machine  View
~
~
~
~
~
:wq
# cat nombre
Pedro Pablo Lopez Rodriguez bp0999
# ls -l
total 14
-rw-r--r-- 1 bin  operator   592 Nov 14 2005 .ashrc
-rw-r--r-- 1 bin  operator   300 Apr 22 2005 .ellepro.b1
-rw-r--r-- 1 bin  operator  5979 Apr 22 2005 .ellepro.e
-rw-r--r-- 1 bin  operator    44 Apr 22 2005 .exrc
-rw-r--r-- 1 bin  operator   304 Aug 12 2005 .profile
-rw-r--r-- 1 bin  operator  2654 Oct 11 2005 .vimrc
-rw-r--r-- 1 root operator    35 Aug 30 10:06 nombre
# wc nombre
   1   5   35 nombre
# halt
Local packages (down): done.
Sending SIGTERM to all processes ...
MINIX will now be shut down ...
d0p0s0>off

```

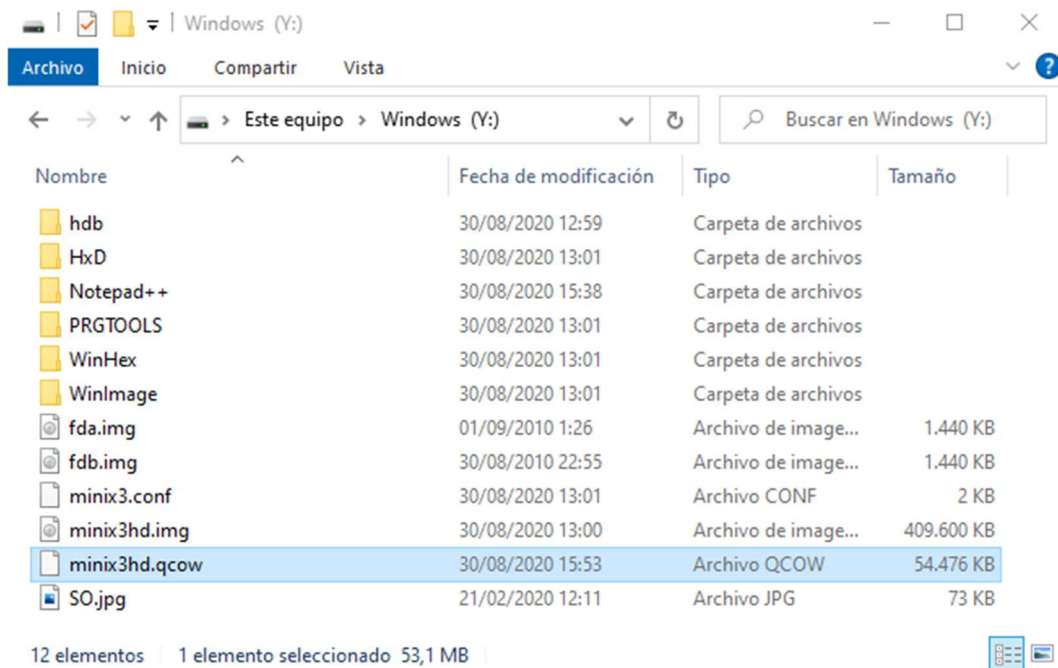
- Ahora vamos a salir del sistema y comprobar que no hemos perdido los cambios que hemos realizado. Para ello, tecleamos el comando **halt (shutdown)**. Nos aparecerá un “prompt” nuevo con la forma “d0p0s0>” que corresponde al *bootloader* (en modo real de 16 bits) de MINIX presente en la subpartición 0 (s0) de la partición 0 (p0) del disco 0 (d0) del controlador 0 (c0) del sistema. Tecleamos el comando **off** del *bootloader* para dar por terminada la sesión QEMU (si falla, cerrar la ventana con el ratón **[X]**).



En Windows (pulse la tecla de Windows si en algún momento qemu tiene retenido el ratón) si abrimos el icono del escritorio correspondiente al equipo veremos que aparece una unidad nueva, normalmente **Y:**. Además si ejecutamos el software de prácticas desde un directorio en vez de desde la raíz de un pendrive, puede aparecer otra unidad, normalmente **X:**. En algunos casos estas unidades pueden tomar como nombre letras diferentes de X e Y.



La unidad X: permite hacer referencia de manera abreviada a la carpeta base donde se encuentra el software de prácticas. Cuando dicha carpeta sea la raíz de un pendrive utilizaremos la letra de la unidad, D: en nuestro caso, en vez de X:). La unidad Y: corresponde a un subdirectorio del directorio temporal (-so-), que va a contener entre otras cosas el fichero imagen de disco, **minix3hd.qcow** o **minix3hd.img**, donde está instalado MINIX y de donde arranca la máquina virtual. La unidad Y: también contendrá algunas aplicaciones y utilidades que se descomprimirán en ese subdirectorio del directorio temporal.

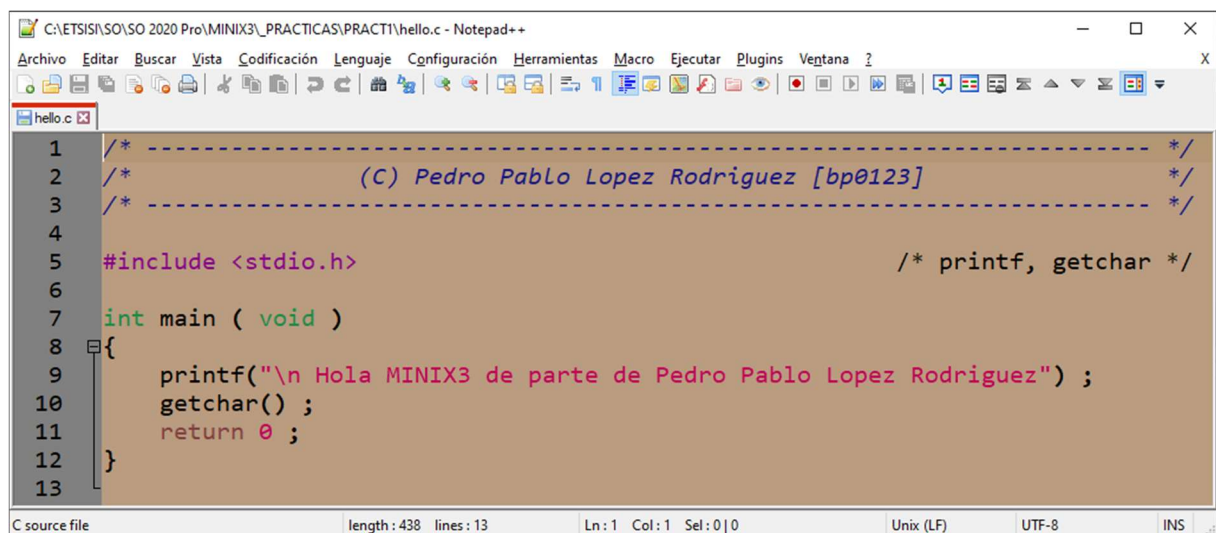


El fichero imagen de disco **Y:\minix3hd.qcow** se creó la primera vez que se ejecutó el programa **X:\minix3.exe** y contiene todos los ficheros de MINIX. Además el fichero de imagen **minix3hd.qcow** ha sido modificado por las acciones que acabamos de realizar. En concreto, esa imagen contiene ahora el nuevo fichero **/root/nombre** con el nombre completo del alumno y su número de matrícula. Cuando termine la sesión de prácticas, si queremos llevarnos el trabajo realizado, **debemos copiar esa imagen a una memoria USB**, ya que cuando apaguemos el equipo del laboratorio la imagen **Y:\minix3hd.qcow** podría borrarse, pues realmente se encuentra en un subdirectorio del directorio temporal de Windows. Si en otra sesión posterior queremos seguir trabajando a partir de donde lo habíamos dejado, debemos copiar al directorio **-so-** del directorio temporal de Windows la imagen de la sesión anterior (teclear **%TMP%\-so-** en el campo de dirección de cualquier ventana para acceder a ese directorio).

[Ejercicio] En este punto el alumno debe salvar la imagen **Y:\minix3hd.qcow** en su pendrive, debe borrar el directorio temporal de Windows y debe cerrar su sesión actual (sin apagar el ordenador). A continuación vuelva a entrar en Windows, copie el fichero **minix3hd.qcow** salvado en el pendrive, al directorio **%TMP%\-so-** y vuelva a arrancar la máquina virtual como se hizo antes. Compruebe que el fichero **/root/nombre** que creó sigue existiendo con el mismo contenido que se le dio cuando se creó. Compruebe con el comando **last** que no se trata de la primera sesión que se ha abierto en el sistema. **[fin Ej]**

2.3 Intercambio de archivos entre Windows y MINIX

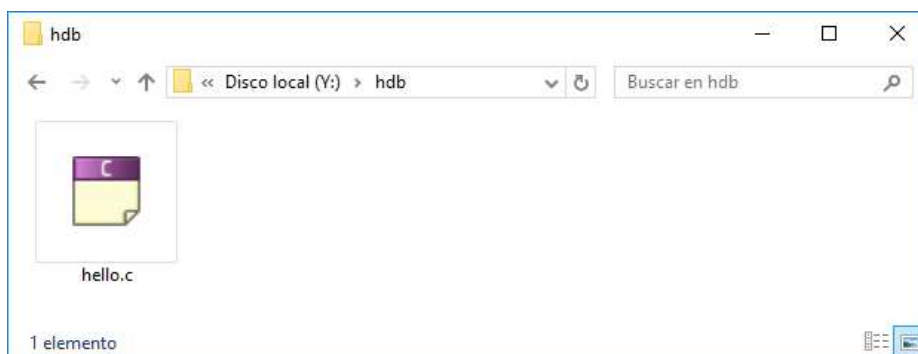
Hemos visto que podemos crear nuevos ficheros en MINIX editándolos por ejemplo con **mined**. Ahora lo que queremos es editar un fichero en Windows y llevarlo a MINIX. Vamos a abrir el programa **X:\MINIX3_PRACTICAS\PRACT1\hello.c** con el editor **Notepad++**, pinchando sobre el correspondiente icono **X:\Notepad++.exe** presente en la raíz de la unidad base.



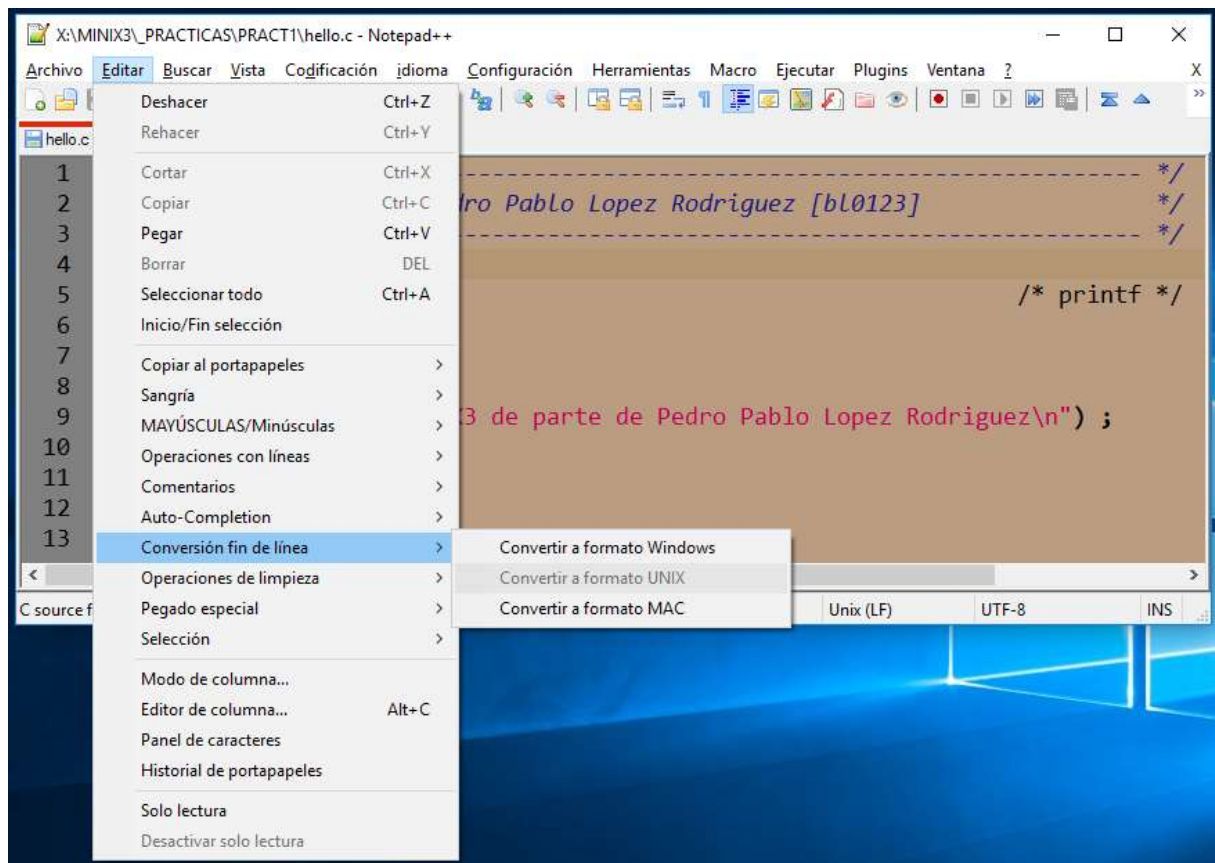
```
C:\ETSIS\ISO\SO 2020 Pro\MINIX3\_PRACTICAS\PRACT1\hello.c - Notepad++
Archivo  Editar  Buscar  Vista  Codificación  Lenguaje  Configuración  Herramientas  Macro  Ejecutar  Plugins  Ventana  ?
hello.c x
1  /* ----- */
2  /*           (C) Pedro Pablo Lopez Rodriguez [bp0123]           */
3  /* ----- */
4
5  #include <stdio.h>                                           /* printf, getchar */
6
7  int main ( void )
8  {
9      printf("\n Hola MINIX3 de parte de Pedro Pablo Lopez Rodriguez" ) ;
10     getchar() ;
11     return 0 ;
12 }
13
C source file          length : 438  lines : 13          Ln : 1  Col : 1  Sel : 0 | 0          Unix (LF)          UTF-8          INS
```

- Modifique el fichero **hello.c** sustituyendo el nombre que aparece, por el nombre completo del alumno y poniendo **putchar('\n') ;** antes del **return(0)**.

Para llevar el fichero **hello.c** a MINIX, debemos tener apagada la máquina virtual MINIX y en ese estado copiar el fichero **hello.c** al directorio **Y:\hdb** (creado al arrancar MINIX).



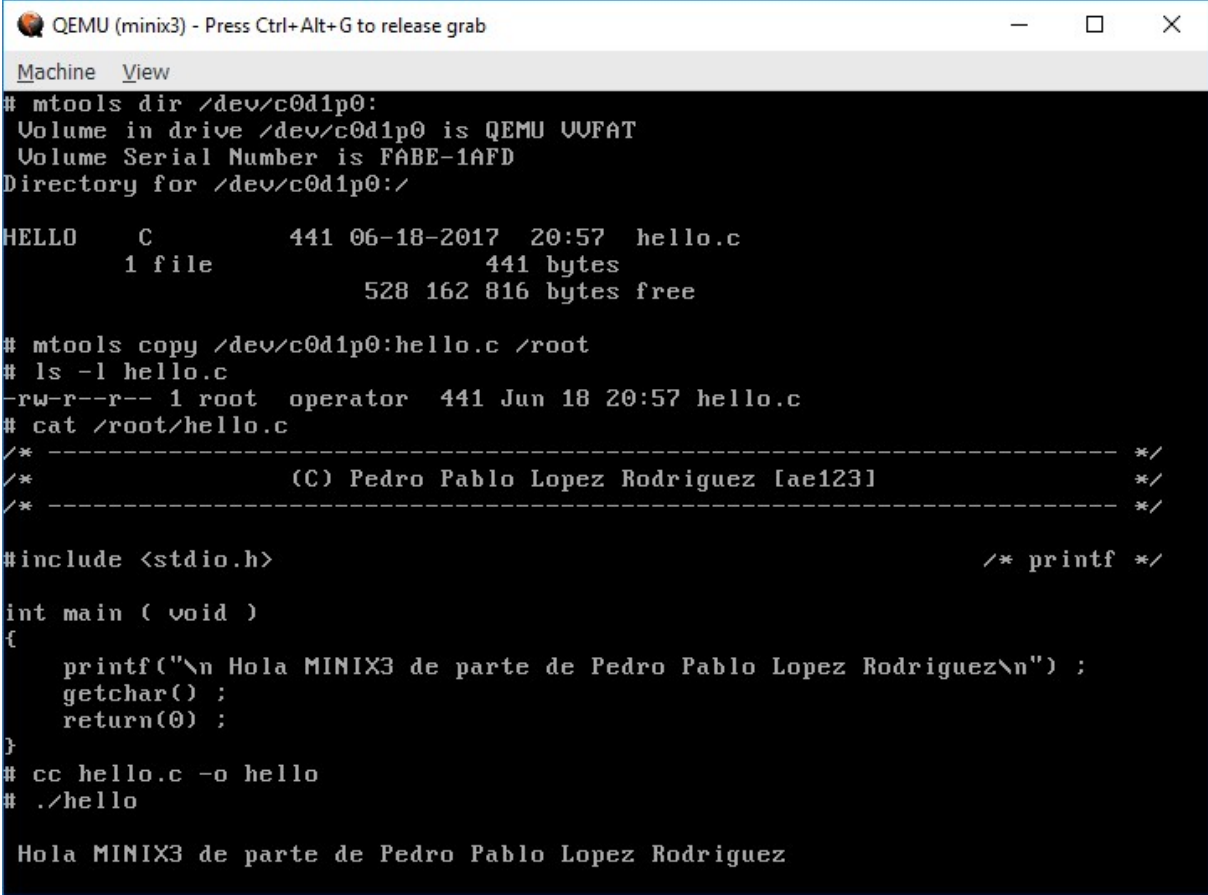
Un detalle importante es que en el Notepad++ antes de salvar el fichero **hello.c** hay que tener seleccionada la opción de conversión del fin de línea a formato UNIX, ya que en Windows los finales de línea se codifican con los caracteres ascii **'\r'** (retorno de carro) y **'\n'** (salto de línea), pero en UNIX el final de línea se codifica sólo con el carácter **'\n'**. Si no se hiciera la conversión podríamos tener problemas a la hora de compilar el programa.



A continuación encendemos la máquina virtual normalmente y, ya en MINIX, ejecutamos los siguientes comandos (ver [mtools](http://www.gnu.org) en <http://www.gnu.org>, # mtools (a secas) muestra un help)

```
# mtools dir /dev/c0d1p0:
# mtools type /dev/c0d1p0:/hello.c
# mtools copy /dev/c0d1p0:/hello.c /root
# ls -l /root/hello.c
# cat /root/hello.c
# cc hello.c -o hello
# ./hello
```

ojo con el ":" final
podría omitirse /dev



```
QEMU (minix3) - Press Ctrl+Alt+G to release grab
Machine View
# mtools dir /dev/c0d1p0:
Volume in drive /dev/c0d1p0 is QEMU UUFAT
Volume Serial Number is F4BE-1AFD
Directory for /dev/c0d1p0:/

HELLO      C          441 06-18-2017  20:57  hello.c
          1 file                441 bytes
                          528 162 816 bytes free

# mtools copy /dev/c0d1p0:hello.c /root
# ls -l hello.c
-rw-r--r-- 1 root  operator  441 Jun 18 20:57 hello.c
# cat /root/hello.c
/* ----- */
/*                (C) Pedro Pablo Lopez Rodriguez [ae123]                */
/* ----- */

#include <stdio.h>                                /* printf */

int main ( void )
{
    printf("\n Hola MINIX3 de parte de Pedro Pablo Lopez Rodriguez\n") ;
    getchar() ;
    return(0) ;
}
# cc hello.c -o hello
# ./hello

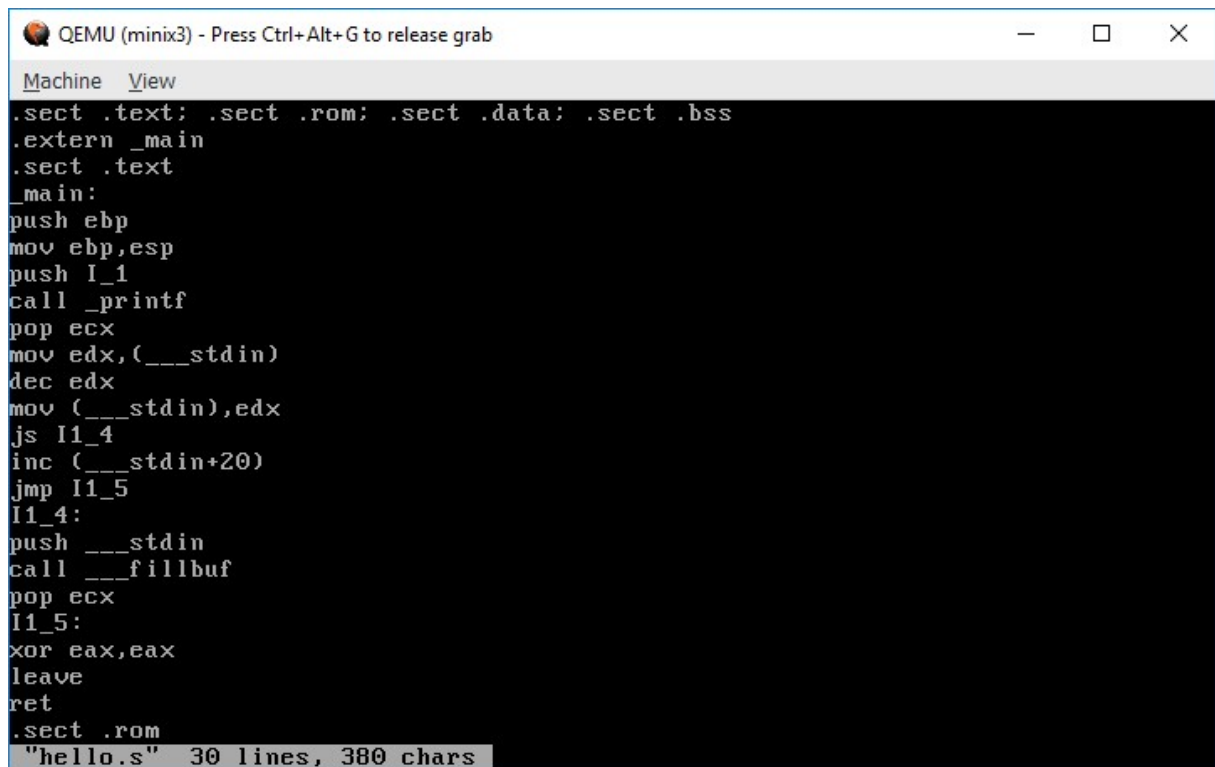
Hola MINIX3 de parte de Pedro Pablo Lopez Rodriguez
```

Vemos que hemos conseguido copiar **hello.c** al directorio **/root**, hemos comprobado que la copia se ha hecho bien y hemos conseguido compilar y ejecutar dicho programa. Hemos utilizado el compilador de C (cc) incluido en MINIX (**make hello** no funciona sin Makefile).

El resultado de la compilación de **hello.c** es el fichero ejecutable de MINIX **hello**. Podríamos haber compilado **hello.c** generando primero un fichero en ensamblador **hello.s** y ensamblando luego ese fichero para generar el ejecutable. En ese caso deberíamos haber ejecutado los comandos:

```
# cc -S hello.c
# cc hello.s -o hello
# ./hello
```

Si tecleamos el comando **ls -l hello*** veremos que además de **hello.c** (fichero fuente en C) tenemos los ficheros **hello.s** (fichero con instrucciones de ensamblador) y **hello** (fichero ejecutable). Podemos ver el contenido de **hello.s** con **cat** o con el editor **mined** o **elvis**:

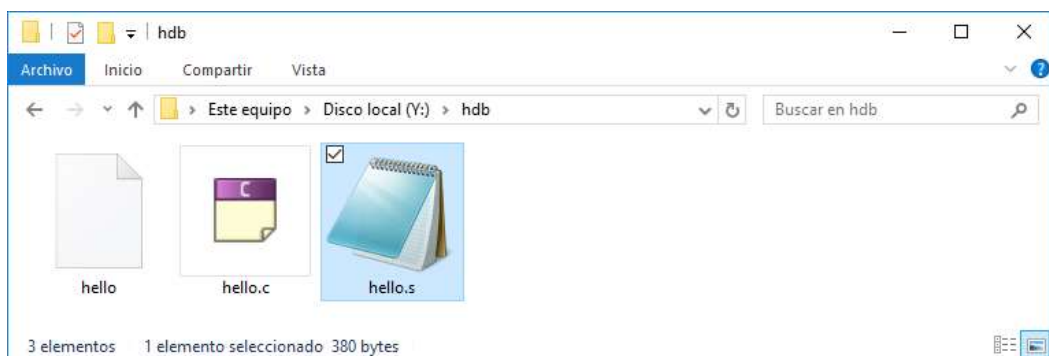


```
QEMU (minix3) - Press Ctrl+Alt+G to release grab
Machine View
.sect .text; .sect .rom; .sect .data; .sect .bss
.extern _main
.sect .text
_main:
push ebp
mov ebp,esp
push I_1
call _printf
pop ecx
mov edx,(__stdin)
dec edx
mov (__stdin),edx
js I1_4
inc (__stdin+20)
jmp I1_5
I1_4:
push __stdin
call __fillbuf
pop ecx
I1_5:
xor eax,eax
leave
ret
.sect .rom
"hello.s" 30 lines, 380 chars
```

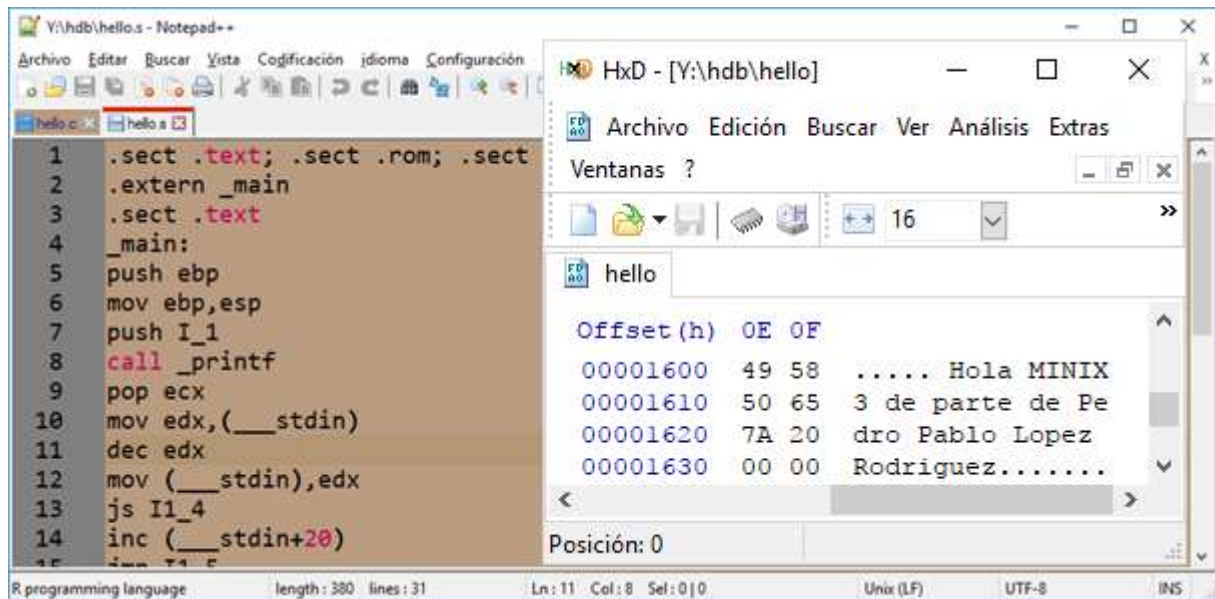
Para llevar los ficheros resultantes de la compilación de **hello.c** a Windows, lo único que hay que hacer es utilizar los correspondientes comandos de las [mtools](#) (ver `mtools -h`):

```
# mtools dir /dev/c0d1p0:
# mtools copy /root/hello.s /dev/c0d1p0:.
# mtools copy /root/hello /dev/c0d1p0:.
# mtools dir /dev/c0d1p0:
```

Tras cerrar MINIX, podemos acceder a esos ficheros en Windows moviéndonos a la carpeta **Y:\hdb** (que no es más que la carpeta `%TMP%\-so-\hdb`):



Por ejemplo podemos visualizar el fichero **hello.s** con el editor Notepad++.

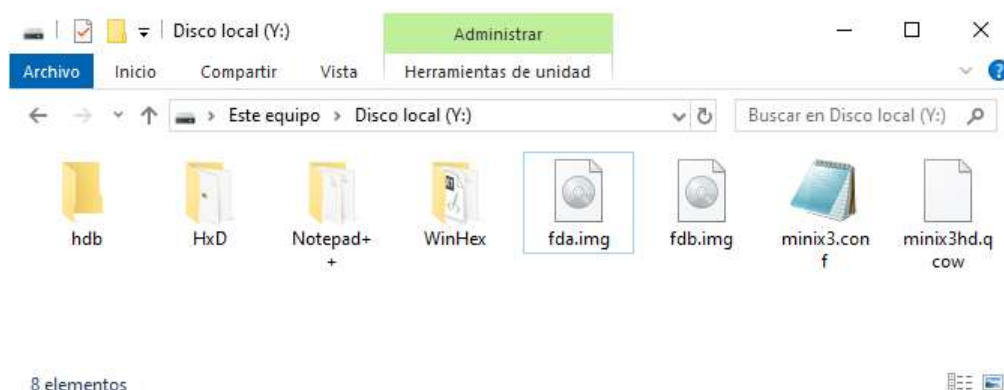


[Ejercicio] Intente ejecutar en Windows el fichero ejecutable **hello** obtenido de la compilación de **hello.c**, describa lo que sucede y razone por qué es eso lo que sucede. También inspeccione el fichero **hello** con el editor hexadecimal [HxD](#) (en X:\Utilidades) para ver su contenido. [fin Ej]

De lo anterior concluimos que somos capaces de llevar ficheros desde Windows a MINIX y viceversa, gracias a la carpeta compartida **Y:\hdb**. Sin embargo el alumno tiene que tener en cuenta que **este método tiene sus limitaciones**. Si MINIX está en funcionamiento y copiamos en Windows un nuevo fichero a **Y:\hdb** (o simplemente modificamos en Windows un fichero de **Y:\hdb**) ese nuevo fichero no será accesible desde MINIX. Dicho de otro modo MINIX sólo puede acceder a los ficheros que estaban inicialmente en **Y:\hdb** o a los que MINIX ha copiado o modificado en **/dev/c0d1p0**. Por tanto en Windows antes de copiar un fichero a **Y:\hdb** debemos asegurarnos de que esté apagada la máquina virtual MINIX.

2.4 Intercambio en caliente de archivos entre Windows y MINIX

Si estamos en MINIX y necesitamos copiar ficheros de Windows sin tener que apagar la máquina virtual, podemos recurrir a trasvasar los ficheros en imágenes de disquete. Si miramos en la unidad **Y:** veremos que contiene dos ficheros: **fda.img** y **fdb.img**.



Los ficheros **fda.img** y **fdb.img** son ficheros de tamaño 1,44 MB que corresponden a imágenes de [disquete](#) estándar de 3,5 pulgadas, doble cara y alta densidad, con 80 cilindros, 2 pistas por cilindro, 18 sectores por pista y 512 bytes por sector. Como $80 \times 2 \times 18 \times 512 \text{ Bytes} = 1440 \text{ KBytes} = 1,44 \text{ MB}$ es obvio que ese es el motivo del tamaño de **fda.img** y **fdb.img**. Los disquetes son unidades de almacenamiento removibles. Inicialmente **fda.img** está insertado en la unidad de disquete **A:** y **fdb.img** está insertado en la unidad de disquete **B:** de la máquina virtual. Esto puede comprobarse en el monitor de qemu (**Ctrl + Alt + 2**) introduciendo el comando **info block**, que muestra el estado de las unidades de bloques:

```

QEMU (minix3)
Machine  View
compat_monitor0 console
QEMU 2.9.0 monitor - type 'help' for more information
(qemu) info block
ide0-hd0 (#block142): Y:\minix3hd.img (qcow)
  Cache mode:      writeback

floppy0 (#block380): Y:\fda.img (raw)
  Removable device: not locked, tray closed
  Cache mode:      writeback

floppy1 (#block544): Y:\fdb.img (raw)
  Removable device: not locked, tray closed
  Cache mode:      writeback

ide0-hd1 (#block1170): json:{"driver": "raw", "file": {"fat-type": 0, "dir": "Y:\
\shdb", "driver": "vfat", "floppy": false, "rw": true}} (raw)
  Cache mode:      writeback

ide1-cd0: [not inserted]
  Removable device: not locked, tray closed

sd0: [not inserted]
  Removable device: not locked, tray closed
(qemu)

```

Tras volver a MINIX con **Ctrl + Alt + 1**, y puesto que **fda.img** y **fdb.img** están formateados con sistemas de ficheros MSDOS ([FAT12](#)), podemos ver con las **mtools** cuál es el contenido actual de las unidades de disquete, utilizando los comandos:

```

# mtools dir /dev/fd0:          # mtools dir /dev/fd1:
# mtools dir a:                # mtools dir b:
# mtools type a:\HOLA_A.TXT    # mtools type b:\HOLA_B.TXT
# mtools copy a:\HOLA_A.TXT /root  # mtools copy b:\HOLA_B.TXT /root

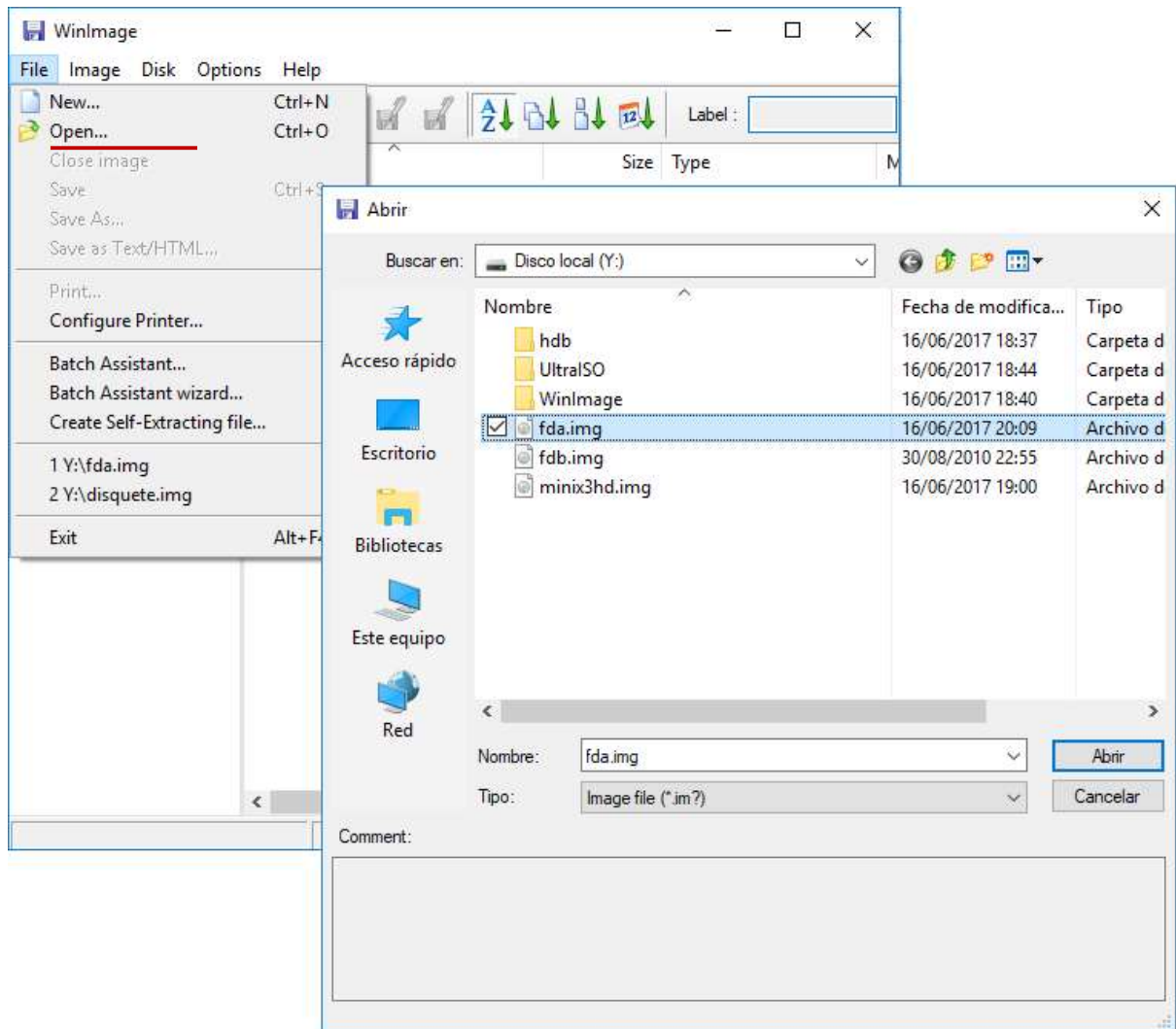
```

Vamos a copiar al disquete de la unidad A: el fichero de cuentas [/etc/passwd](#):

```
# mtools copy /etc/passwd a:
```

Para llevar a Windows el fichero copiado tenemos que quitar el disquete de la unidad A:, lo cual se lleva a cabo en el [monitor de qemu](#) con el comando: **eject floppy0**. Tras ejecutar ese comando veremos con **info block** que el disquete ha sido retirado de la unidad.

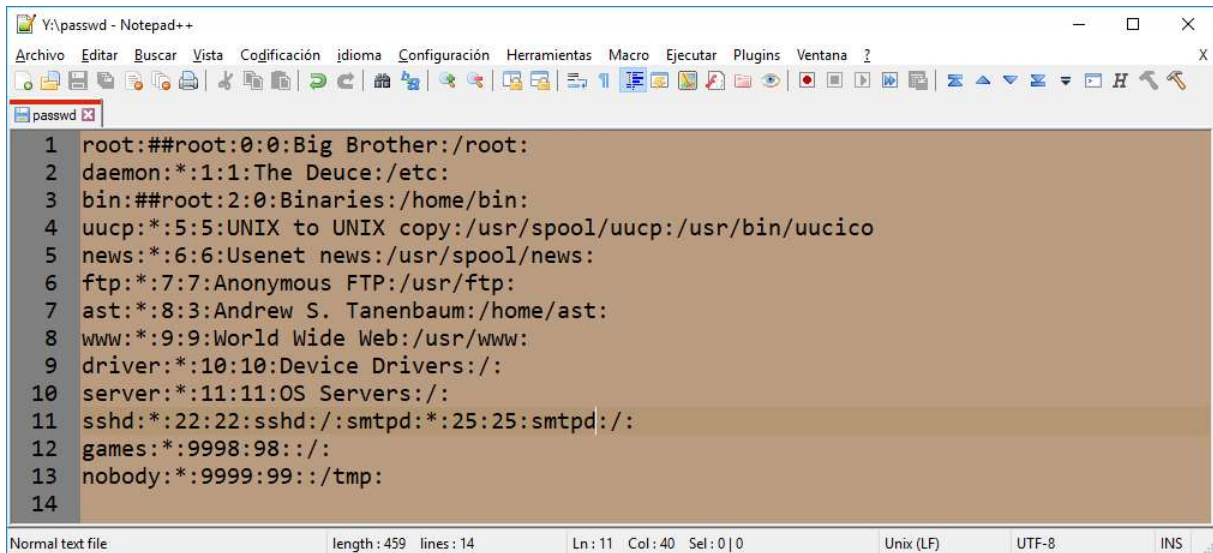
Ahora en Windows tenemos que ejecutar el programa [WinImage](#) (en X:\Utilidades) para abrir la imagen de disquete **Y:\fda.img**:



Una vez abierta la imagen **Y:fda.img** ya podemos acceder a los ficheros que contiene:



Tras extraer **passwd** podemos visualizar su contenido con el Notepad++.

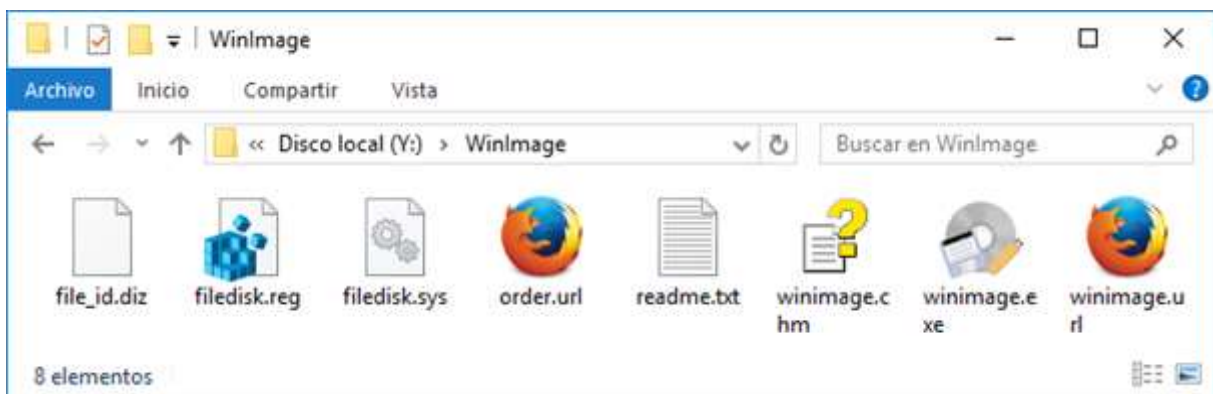
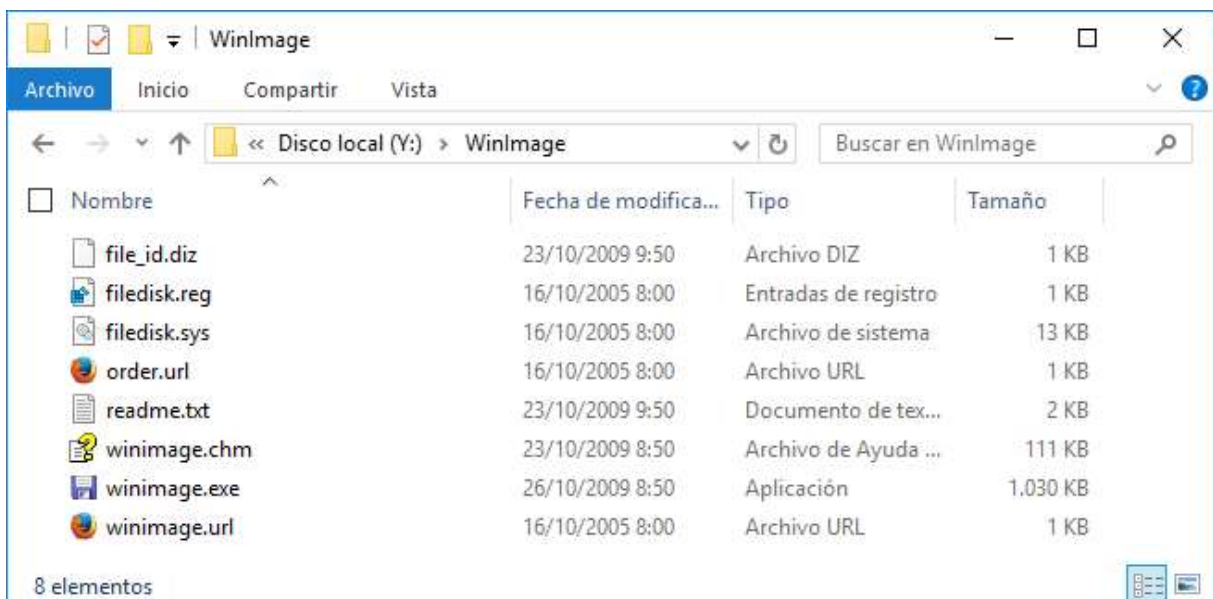


```

1 root:##root:0:0:Big Brother:/root:
2 daemon:*:1:1:The Deuce:/etc:
3 bin:##root:2:0:Binaries:/home/bin:
4 uucp:*:5:5:UNIX to UNIX copy:/usr/spool/uucp:/usr/bin/uucico
5 news:*:6:6:Usenet news:/usr/spool/news:
6 ftp:*:7:7:Anonymous FTP:/usr/ftp:
7 ast:*:8:3:Andrew S. Tanenbaum:/home/ast:
8 www:*:9:9:World Wide Web:/usr/www:
9 driver:*:10:10:Device Drivers:/:
10 server:*:11:11:OS Servers:/:
11 sshd:*:22:22:sshd:/:smtpd:*:25:25:smtpd:/:
12 games:*:9998:98:/:
13 nobody:*:9999:99:/tmp:
14

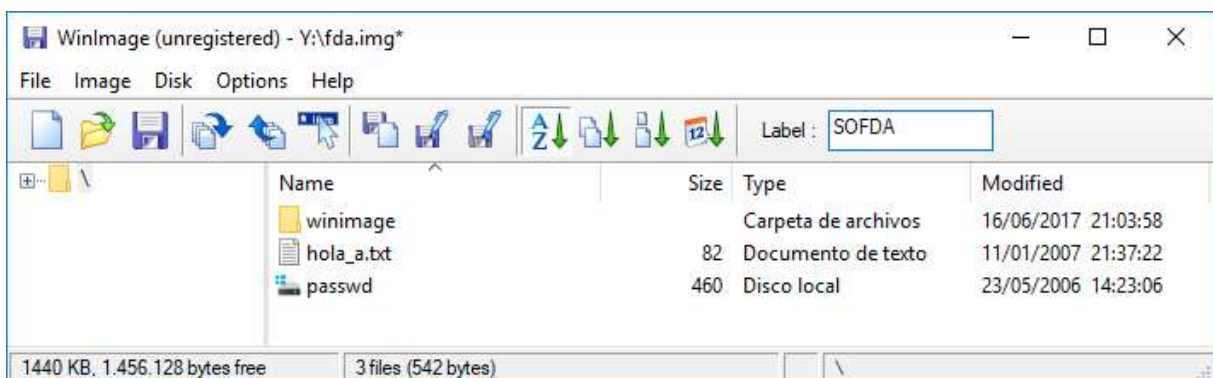
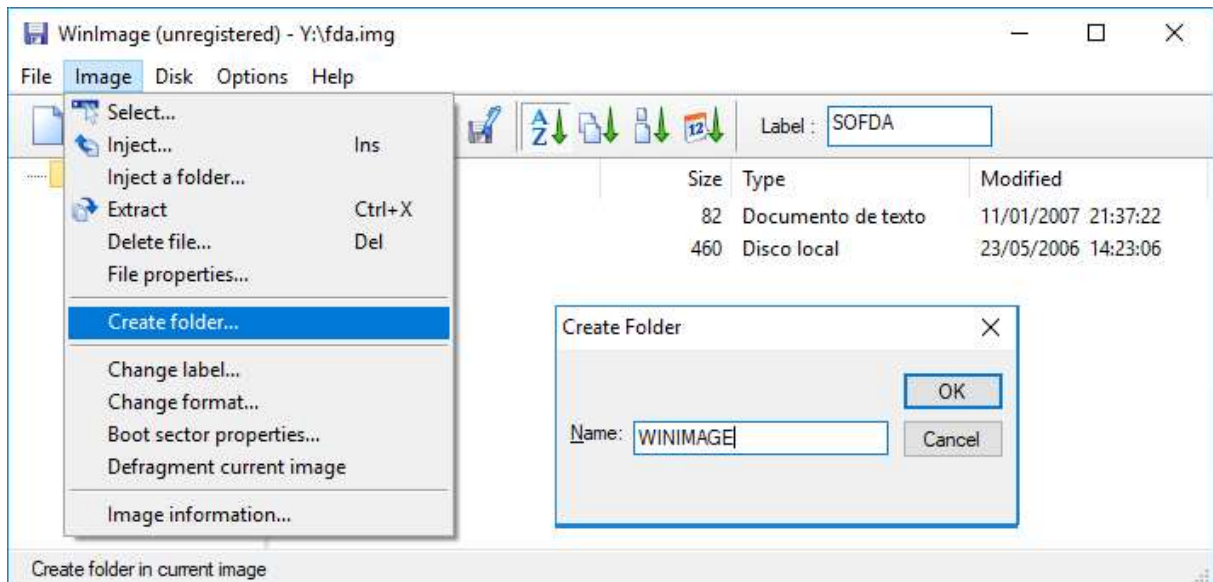
```

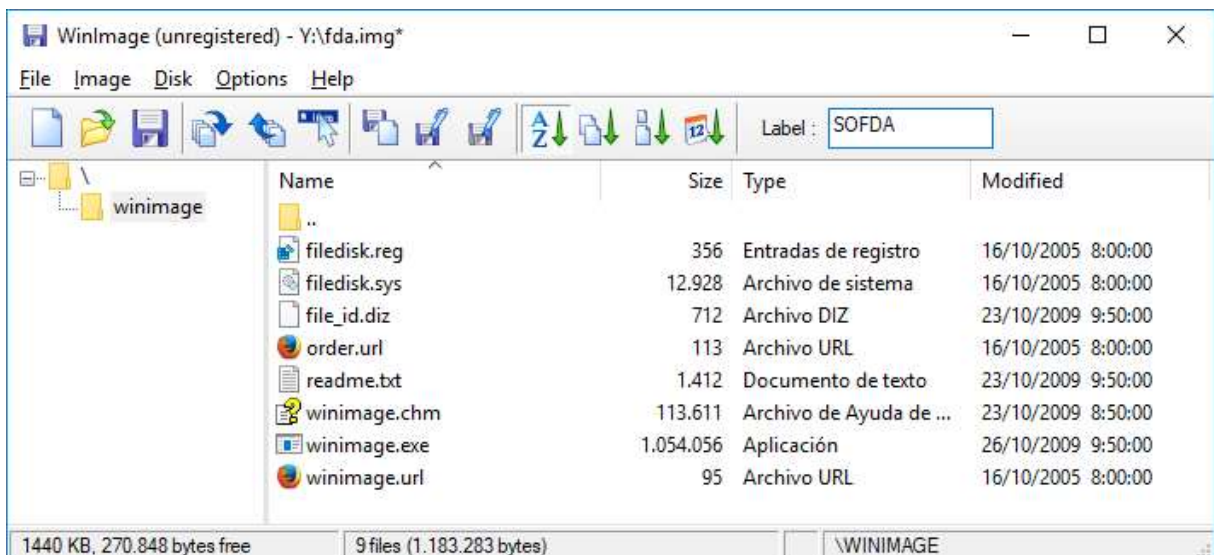
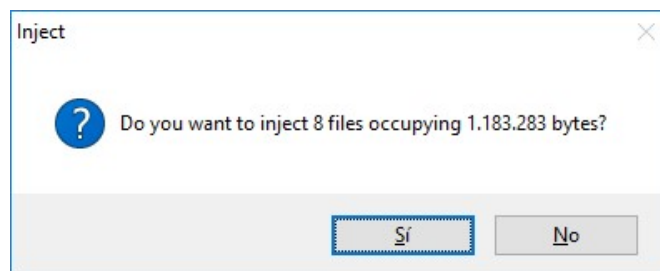
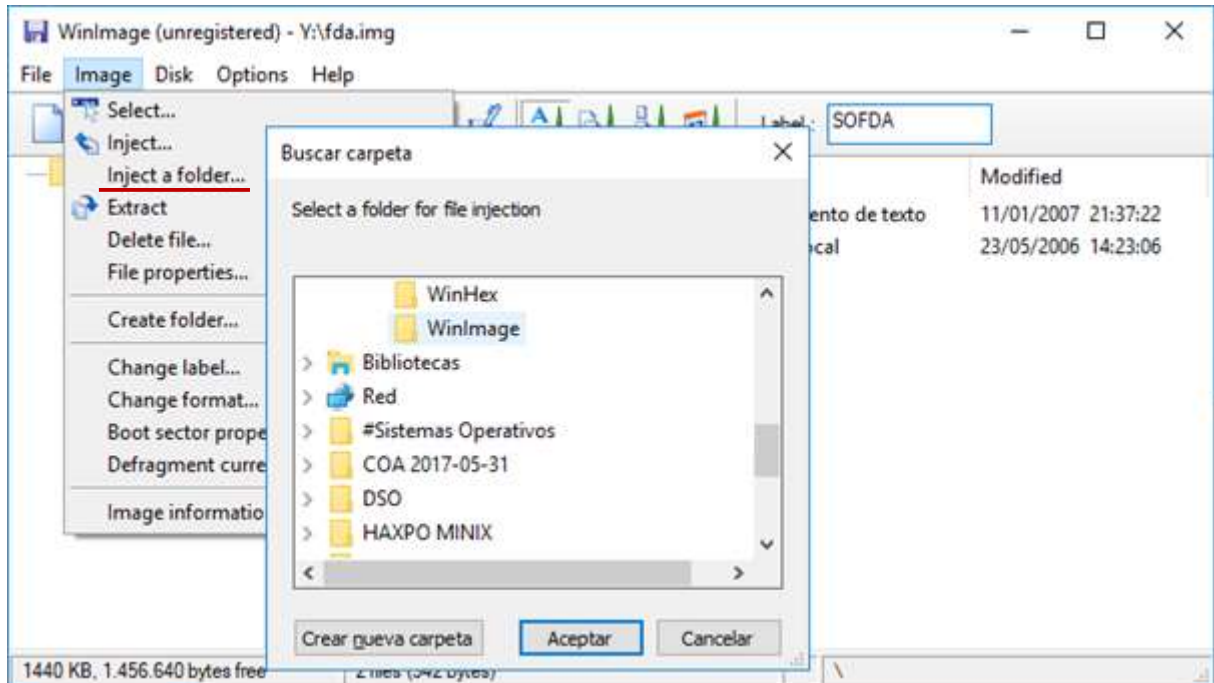
Hemos conseguido llevar ficheros de MINIX a Windows a través de una imagen de disquete. Deseamos ahora ser capaces de llevar ficheros de Windows a MINIX (recordemos que sin tener que reiniciar la máquina virtual que está en funcionamiento). Por ejemplo vamos a llevar todos los ficheros del directorio **Y:\WinImage** al directorio de MINIX: **/root/WinImage**.

Nombre	Fecha de modifica...	Tipo	Tamaño
file_id.diz	23/10/2009 9:50	Archivo DIZ	1 KB
filedisk.reg	16/10/2005 8:00	Entradas de registro	1 KB
filedisk.sys	16/10/2005 8:00	Archivo de sistema	13 KB
order.url	16/10/2005 8:00	Archivo URL	1 KB
readme.txt	23/10/2009 9:50	Documento de tex...	2 KB
winimage.chm	23/10/2009 8:50	Archivo de Ayuda ...	111 KB
winimage.exe	26/10/2009 8:50	Aplicación	1.030 KB
winimage.url	16/10/2005 8:00	Archivo URL	1 KB

Vemos que todos los ficheros caben en un solo disquete ($1 + 1 + 13 + 1 + 2 + 111 + 1030 + 1 = 1160 \text{ KB} < 1440 \text{ KB} = 1,44 \text{ MB}$), por lo que utilizamos WinImage para meter (opción **Inject a folder ...**) el directorio **WinImage** en la imagen de disquete **Y:\fda.img**. Antes de nada hay que crear en la imagen **Y:\fda.img** una nueva carpeta que llamaremos **WinImage**, e insertaremos dentro de ella todos los ficheros contenidos en **Y:\WinImage**:





Finalmente salvamos la imagen **Y:\fda.img** (opción **Save**) y la cerramos (opción **Close image**).

Ahora debemos introducir la imagen **Y:\fda.img** en la unidad de disquete **A:** de la máquina virtual qemu, para lo cual ejecutaremos el comando del monitor: **change floppy0 Y:\fda.img**.

```

QEMU (minix3)
Machine  View

sd0: [not inserted]
  Removable device: not locked, tray closed
(qemu) info block
ide0-hd0 (#block142): Y:\minix3hd.img (qcow)
  Cache mode:      writeback

floppy0: [not inserted]
  Removable device: not locked, tray closed

floppy1 (#block544): Y:\fdb.img (raw)
  Removable device: not locked, tray closed
  Cache mode:      writeback

ide0-hd1 (#block1170): json:{"driver": "raw", "file": {"fat-type": 0, "dir": "Y:\
\nhdb", "driver": "vfat", "floppy": false, "rw": true}} (raw)
  Cache mode:      writeback

ide1-cd0: [not inserted]
  Removable device: not locked, tray closed

sd0: [not inserted]
  Removable device: not locked, tray closed
(qemu) change floppy0 Y:\fda.img

```

Después debemos comprobar con un nuevo comando **info block** que la imagen se ha insertado correctamente en la unidad. Tras eso conmutamos a MINIX (**Ctrl + Alt + 1**) y ejecutamos los comandos:

```

# mtools dir a:
# mtools dir a:/WinImage
# mkdir /root/WinImage
# mtools copy a:/WinImage/* /root/WinImage
# ls -la /root/WinImage

```

Este ejemplo detallado debe ser suficiente para que el alumno competente sea capaz de transferir ficheros desde Windows a MINIX y viceversa sin necesidad de reiniciar MINIX.

[[mkfs](#) y [mount](#)]

Terminamos este apartado comentando cómo puede formatearse un disquete con el formato propio de MINIX. Por ejemplo vamos a formatear el disquete metido en la unidad B: (imagen **fdb.img**), vamos a montar en el directorio **/mnt** el sistema de ficheros que contiene **fdb.img** y vamos a copiar los ficheros de **/root/WinImage** al disquete formateado.

```

# mkfs /dev/fd1                (formateamos el disquete de la unidad B:)
# mount /dev/fd1 /mnt          (montamos su sistema de ficheros en /mnt)
# cp /root/WinImage/* /mnt
# ls /mnt
# umount /dev/fd1              (desmontamos el sistema de ficheros)

```

Ejercicio: Quitar (con el monitor de qemu) el disquete **fdb.img** de la unidad B:, meterlo en la unidad A: (**/dev/fd0**), montarlo en un nuevo directorio **/root/nuevo** y comprobar que el contenido del directorio **/root/nuevo** pasa a ser todos los ficheros de **WinImage**. **[fin Ej.]**

2.5 Uso de un terminal a través de la línea de comunicación serie (TCP/IP)

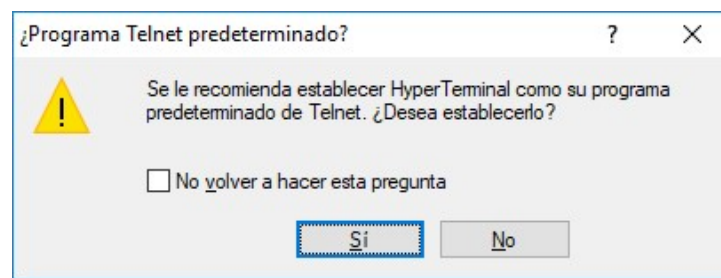
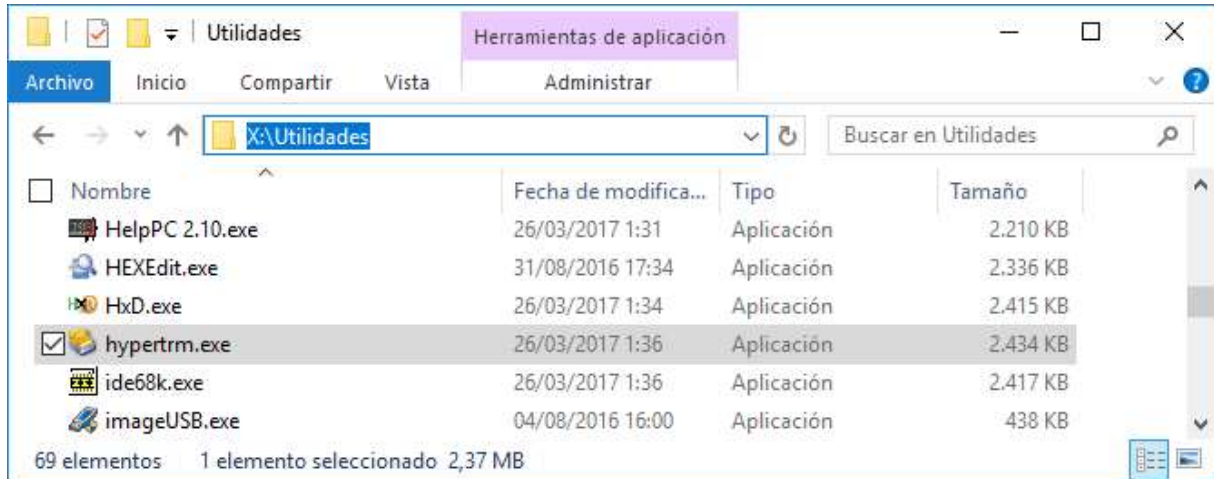
En este apartado vamos ver cómo podemos conectar un emulador de terminal a nuestra máquina virtual MINIX. La ventaja de tener un terminal tipo Windows conectado a una sesión de MINIX es que la sesión va a quedar registrada en un fichero de Windows que podremos consultar en caso de caída repentina de la máquina virtual, con el fin de intentar saber qué ha causado ese fallo.

Antes de nada hay que decir que MINIX es un sistema *multiprogramado multiusuario con tiempo compartido*, de manera que varios usuarios pueden estar utilizando el ordenador simultáneamente a través de terminales. Los comandos introducidos por cada usuario a través de su terminal son atendidos por un proceso intérprete de comandos independiente, de manera que ese proceso lee el comando desde el (teclado del) terminal y escribe las salidas del comando en (la pantalla de) el terminal. Incluso un único usuario puede abrir varias sesiones a la vez y conmutar entre ellas gracias al *multiscreen*, que consiste en conmutar manualmente de un terminal virtual a otro gracias a una combinación de teclas. Concretamente las teclas **Alt + F1**, **Alt + F2**, **Alt + F3** y **Alt + F4** permiten conmutar entre 4 terminales virtuales, pudiendo multiplexarse así el teclado y la pantalla físicos entre los cuatro terminales virtuales **/dev/ttyc0** (**/dev/console**), **/dev/ttyc1**, **/dev/ttyc2** y **/dev/ttyc3**. [**!!!** Alt + F4 cierra la ventana Windows]

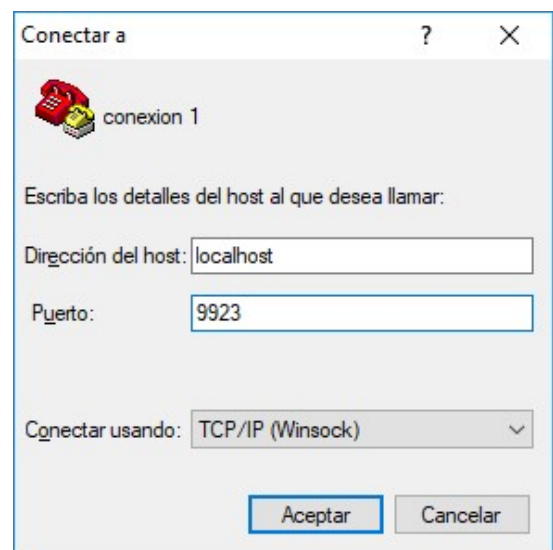
The image shows two overlapping terminal windows from a QEMU (minix3) environment. The top window, titled 'QEMU (minix3) - Press Ctrl+Alt+G to release grab', shows the initial boot sequence and a login prompt for 'root' at '10.0.0.1'. The bottom window, also titled 'QEMU (minix3) - Press Ctrl+Alt+G to release grab', shows the user 'ast' logging in at the same IP address. The user then enters the command 'passwd ast' to change their password. The output of 'passwd' shows the password was successfully changed. Finally, the user enters 'id', which returns 'uid=8(ast) gid=3(other)'. The prompt '\$' is visible at the end of the line.

En la pantalla anterior se ve como inicialmente el usuario **root** ha abierto una sesión desde el terminal **/dev/ttyc0** que corresponde a la consola principal **/dev/console**. El usuario **root** ha establecido la contraseña del usuario **ast** (Andrew Stuart Tanenbaum) haciendo uso del comando **passwd**. Luego el usuario ha conmutado al terminal **/dev/ttyc1** (presionando **Alt + F2**) y ha abierto una sesión como usuario **ast** (también podría haberla abierto como usuario **root**).

Dicho lo anterior vamos a conectar un terminal tipo Windows a MINIX. Para hacer la conexión tiene que estar en funcionamiento MINIX. Utilizaremos [HyperTerminal](#) como programa emulador de terminal, para lo cual, en Windows, pinchamos con el ratón sobre el icono del programa **X:\Utilidades\hypertrm.exe**.

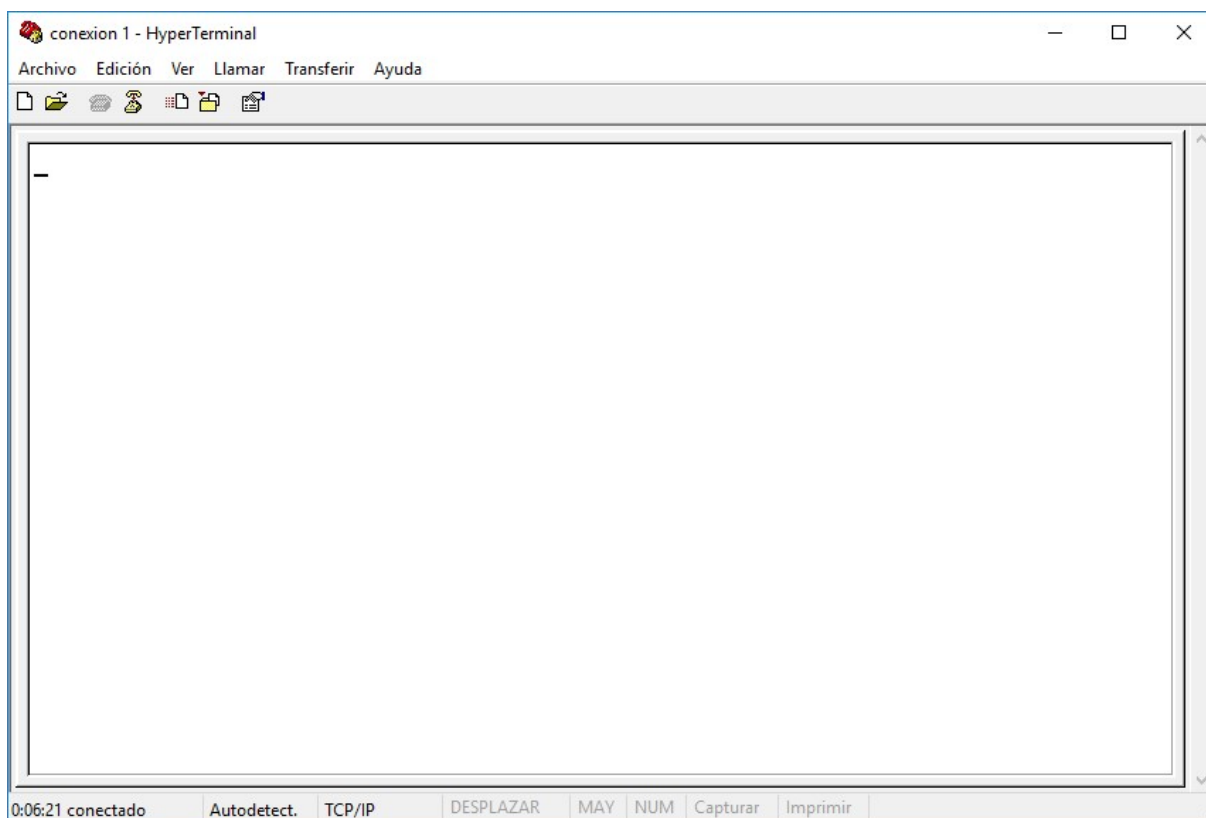


Respondemos que No.



La conexión se configura como de tipo TCP/IP en la dirección IP del equipo local del alumno (**localhost**, realmente **127.0.0.1**) y el puerto **9923**. La máquina virtual qemu está escuchando en dicha dirección IP **127.0.0.1** a través del puerto **9923** a la espera de que se establezca una conexión. En qemu dicha conexión está ligada (Y:\minix3.conf) al puerto serie virtual COM1, que en MINIX se referencia mediante el fichero especial de dispositivo **/dev/tty00**.

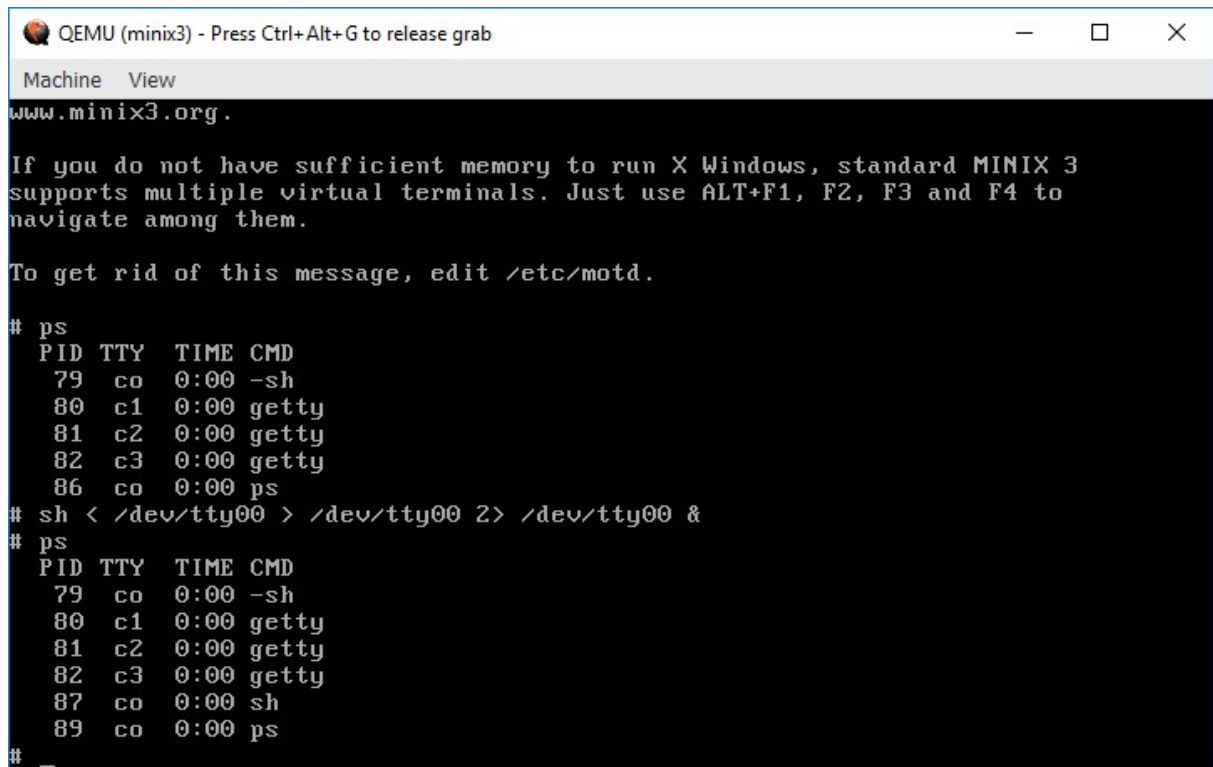
Tras establecerse la conexión se tiene la siguiente ventana de Windows, en cuya esquina inferior izquierda se indica el tiempo que ha transcurrido desde que se estableció la conexión.



Para poder ejecutar comandos de una sesión de MINIX podemos introducir el siguiente comando de redirección de la entrada estándar (descriptor de fichero **0**), salida estándar (descriptor de fichero **1**) y salida de error estándar (descriptor **2**) de un proceso ejecutando el intérprete de comandos (shell [sh](#)):

```
# sh < /dev/tty00 > /dev/tty00 2> /dev/tty00 &
```

En la siguiente pantalla vemos como el proceso con PID 79 es el interprete de comandos de la sesión abierta en la consola, los procesos con PID 80, 81 y 82 son procesos que esperan a que un usuario abra sesión en los terminales **/dev/ttyc1**, **/dev/ttyc2** y **/dev/ttyc3**, respectivamente. El proceso con PID 87 es un *proceso de fondo* hijo del proceso 79 que ejecuta el comando **sh** redirigido anterior, y finalmente el proceso con PID 89 es el proceso hijo del proceso 79 que ejecuta el comando **ps**.

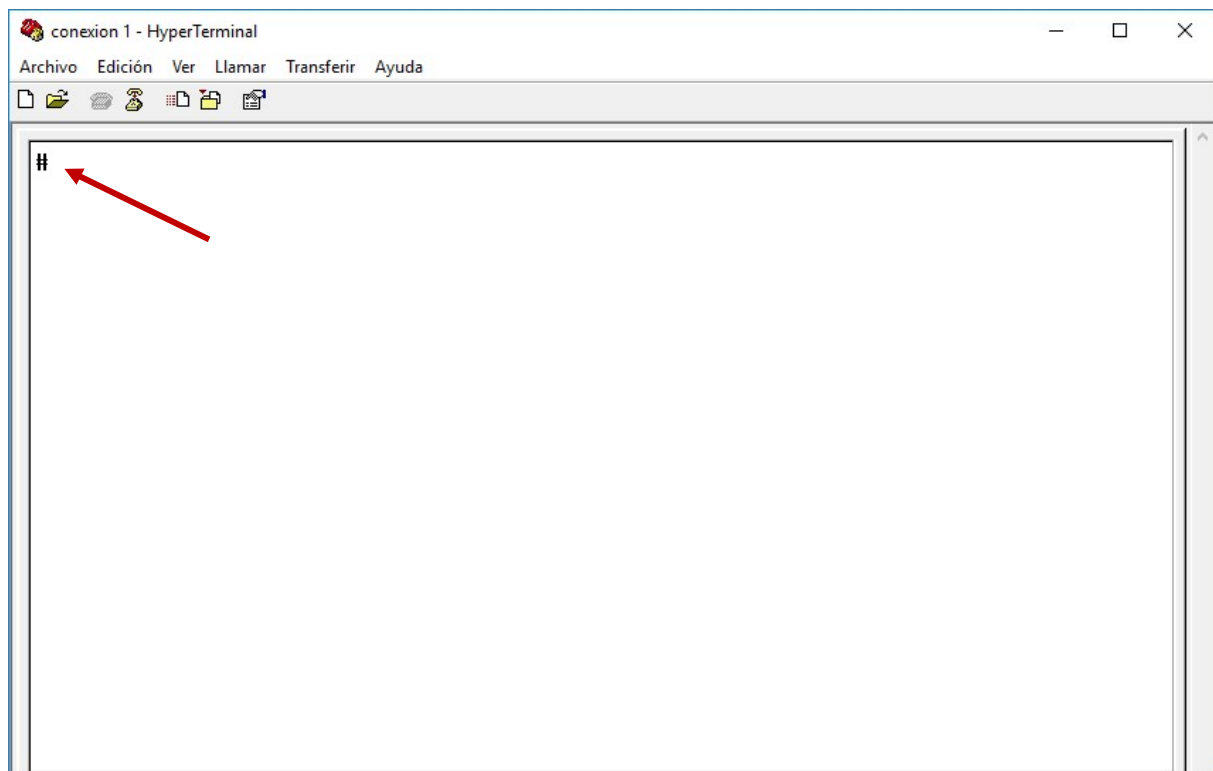


```
QEMU (minix3) - Press Ctrl+Alt+G to release grab
Machine View
www.minix3.org.

If you do not have sufficient memory to run X Windows, standard MINIX 3
supports multiple virtual terminals. Just use ALT+F1, F2, F3 and F4 to
navigate among them.

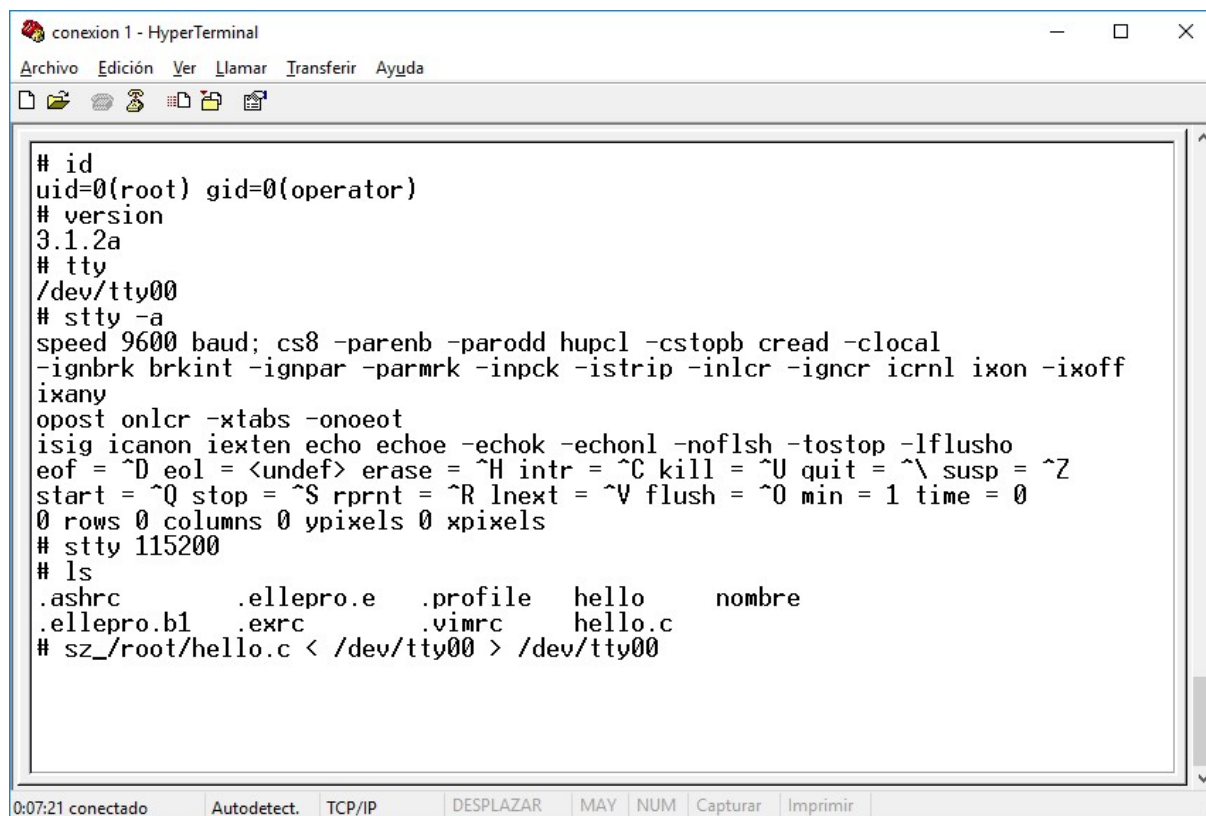
To get rid of this message, edit /etc/motd.

# ps
  PID TTY  TIME CMD
   79  co   0:00 -sh
   80  c1   0:00 getty
   81  c2   0:00 getty
   82  c3   0:00 getty
   86  co   0:00 ps
# sh < /dev/tty00 > /dev/tty00 2> /dev/tty00 &
# ps
  PID TTY  TIME CMD
   79  co   0:00 -sh
   80  c1   0:00 getty
   81  c2   0:00 getty
   82  c3   0:00 getty
   87  co   0:00 sh
   89  co   0:00 ps
#
```



```
conexion 1 - HyperTerminal
Archivo Edición Ver Llamar Transferir Ayuda
#
```

Se observa que aparece el *prompt* # ante el que ya podemos introducir comandos, que en nuestro caso son ejecutados por el proceso **sh** con PID 87.



```

conexion 1 - HyperTerminal
Archivo Edición Ver Llamar Transferir Ayuda
# id
uid=0(root) gid=0(operator)
# version
3.1.2a
# tty
/dev/tty00
# stty -a
speed 9600 baud; cs8 -parenb -parodd hupcl -cstopb cread -clocal
-ignbrk brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon -ixoff
ixany
opost onlcr -xtabs -onoeot
isig icanon iexten echo echoe -echok -echonl -noflsh -tostop -lflusho
eof = ^D eol = <undef> erase = ^H intr = ^C kill = ^U quit = ^\ susp = ^Z
start = ^Q stop = ^S rprnt = ^R lnext = ^V flush = ^O min = 1 time = 0
0 rows 0 columns 0 ypixels 0 xpixels
# stty 115200
# ls
.ashrc      .ellepro.e .profile   hello      nombre
.ellepro.bl .exrc      .vimrc    hello.c
# sz_/root/hello.c < /dev/tty00 > /dev/tty00
0:07:21 conectado  Autodetect. TCP/IP DESPLAZAR MAY NUM Capturar Imprimir

```

Concluimos que podemos ejecutar en MINIX comandos escritos desde el terminal que hemos conectado. Hay que tener algo de cuidado y evitar teclear Ctrl-C en el terminal cuando queramos abortar un comando. En su lugar es preferible ir a la consola de MINIX y matar el proceso que deseemos abortar utilizando los comandos: **ps** (para conocer el PID del proceso) y **kill** <PID> para matar el proceso identificado por su PID. [**kill -9** <PID> si se pone farruco]

Una ventaja adicional de **HyperTerminal** es que podemos transferir ficheros a Windows con el comando **sz**, por ejemplo: **sz /root/hello.c < /dev/tty00 > /dev/tty00**



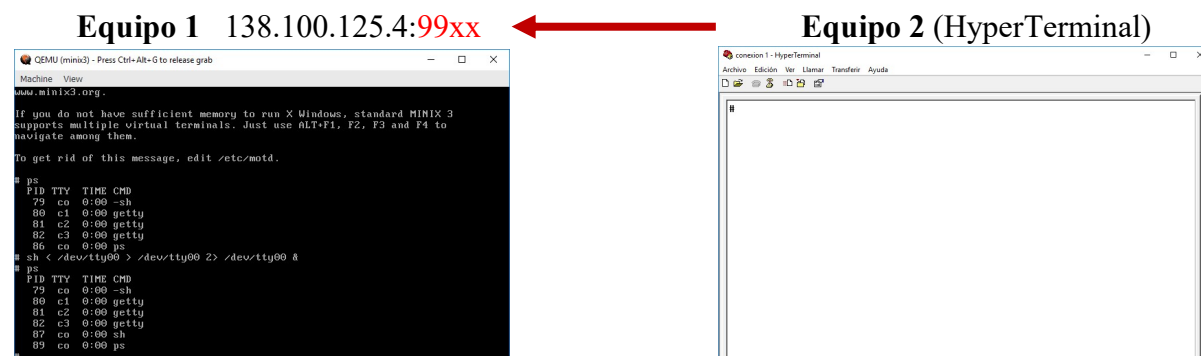
Ejercicio: Ponte de acuerdo con otro compañero en otro equipo para realizar este ejercicio.

Un alumno debe ejecutar primero en su ordenador la máquina virtual qemu **Minix3.exe** configurada (`minix3.conf`) para aceptar conexiones por su puerto serie a través de su IP (visualizarla con el comando de Windows `C:\> PING -4 %COMPUTERNAME%`). Por un mínimo de seguridad utilícese un puerto **99xx** que no conozca nadie más.

El otro alumno debe conectarse remotamente por TCP/IP con el **HyperTerminal** a la dirección IP (y puerto **99xx**) del ordenador donde se ejecuta Minix3.

Luego en el primer equipo, en Minix, ejecútase

```
# sh < /dev/tty00 > /dev/tty00 2> /dev/tty00 &
```



Una vez establecida la conexión compruébese que los dos alumnos pueden simultáneamente hacer uso de Minix, de manera que por ejemplo comandos introducidos como

```
cd / ; ls -lR
```

o

```
cd / ; du -a
```

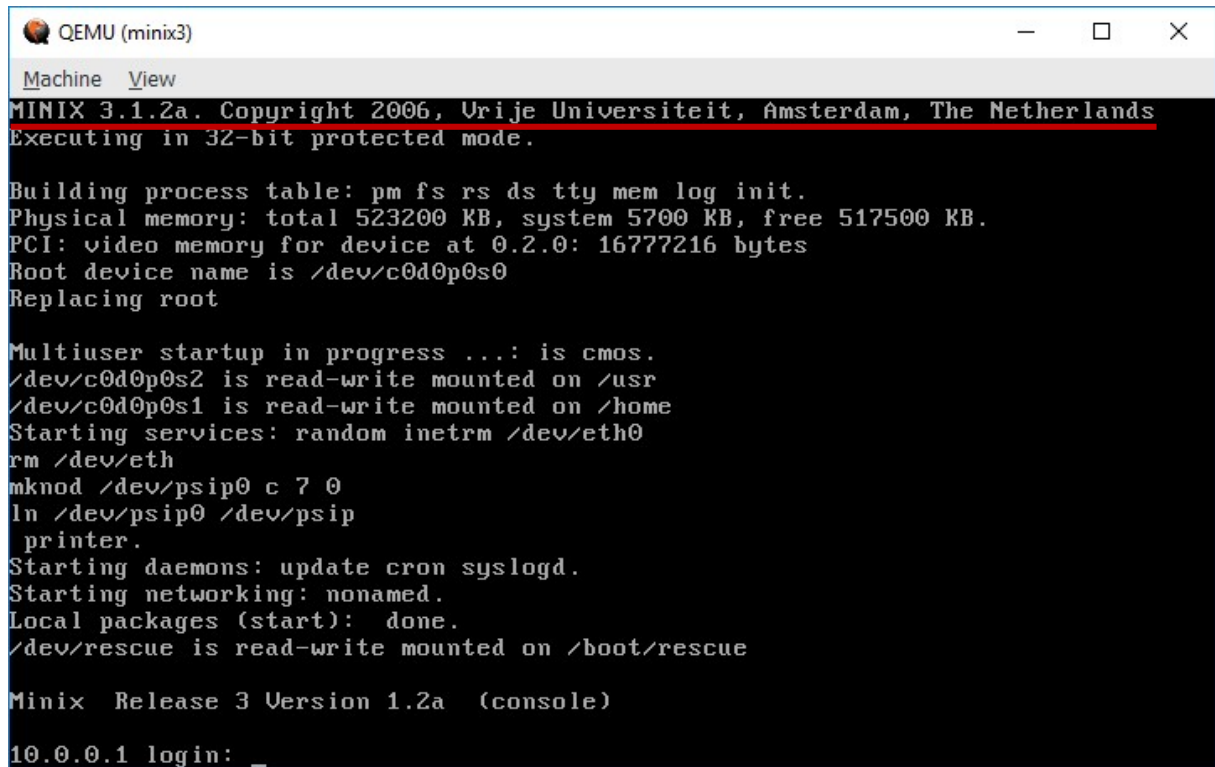
avanzan simultáneamente en la consola de Minix y en la sesión abierta con el HyperTerminal.

Pregunta: ¿Esta situación corresponde a un sistema en tiempo compartido (*time sharing*) o a un sistema de tiempo real (*real time*)?

[fin Ej.]

3 MODIFICACIÓN PUNTUAL DEL NÚCLEO DE MINIX

En este apartado vamos a llevar a cabo una modificación puntual del núcleo de MINIX consistente simplemente en hacer que el núcleo en vez de desplegar el mensaje que aparece en la primera línea de la pantalla siguiente, muestre una línea diferente con el nombre del alumno.



```
QEMU (minix3)
Machine View
MINIX 3.1.2a. Copyright 2006, Vrije Universiteit, Amsterdam, The Netherlands
Executing in 32-bit protected mode.

Building process table: pm fs rs ds tty mem log init.
Physical memory: total 523200 KB, system 5700 KB, free 517500 KB.
PCI: video memory for device at 0.2.0: 16777216 bytes
Root device name is /dev/c0d0p0s0
Replacing root

Multiuser startup in progress ...: is cmos.
/dev/c0d0p0s2 is read-write mounted on /usr
/dev/c0d0p0s1 is read-write mounted on /home
Starting services: random inetrm /dev/eth0
rm /dev/eth
mknod /dev/psip0 c 7 0
ln /dev/psip0 /dev/psip
printer.
Starting daemons: update cron syslogd.
Starting networking: nonamed.
Local packages (start): done.
/dev/rescue is read-write mounted on /boot/rescue

Minix Release 3 Version 1.2a (console)
10.0.0.1 login: _
```

Es decir el núcleo en el arranque (o tras pulsar **F7**) en vez de mostrar la línea:

MINIX 3.1.2a. Copyright 2006, Vrije Universiteit, Amsterdam, The Netherlands

Deberá mostrar la línea:

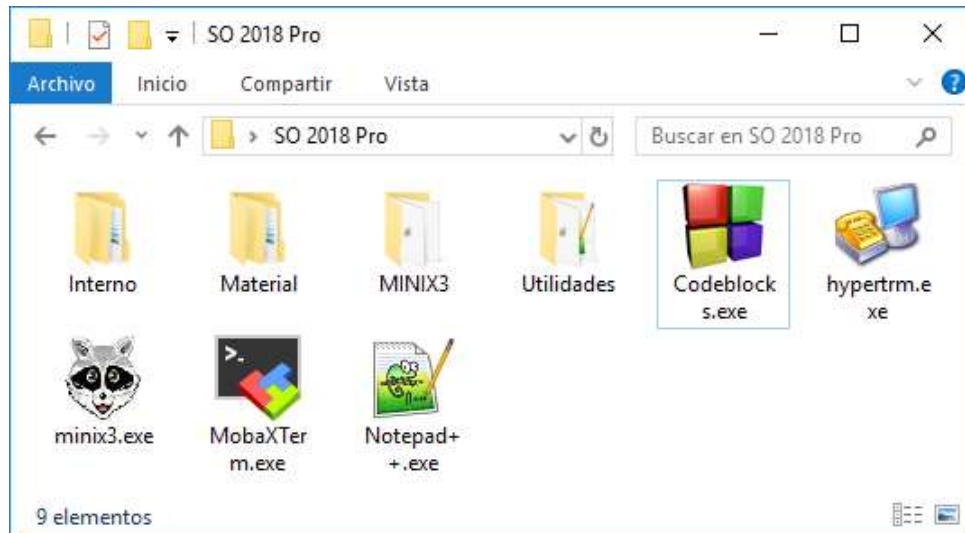
MINIX 3.1.2a. (C) 2020, Pedro Pablo Lopez Rodriguez, Madrid, Spain

pero poniendo en vez del nombre del profesor, el nombre del alumno.

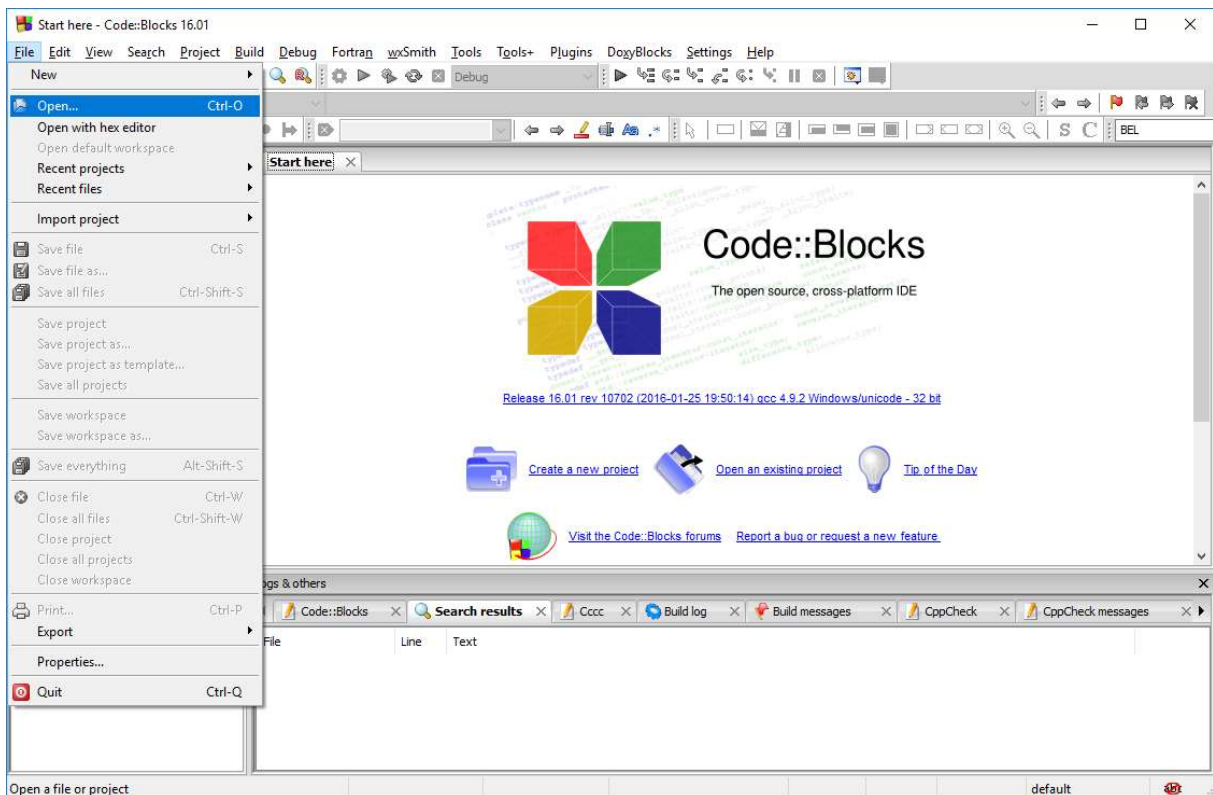
Lo primero que necesitamos es localizar los ficheros fuente de MINIX para modificarlos y recompilarlos. Dichos ficheros fuente se encuentran bajo el directorio **/usr/src** de MINIX. En particular, los ficheros correspondientes al núcleo (**kernel**), se encuentran en el directorio **/usr/src/kernel**.

No obstante ya que disponemos de una copia de dichos fuentes en el directorio **X:\MINIX3** de Windows, vamos a localizar en el directorio **X:\MINIX3\kernel** el fichero del núcleo que contiene la línea que queremos modificar.

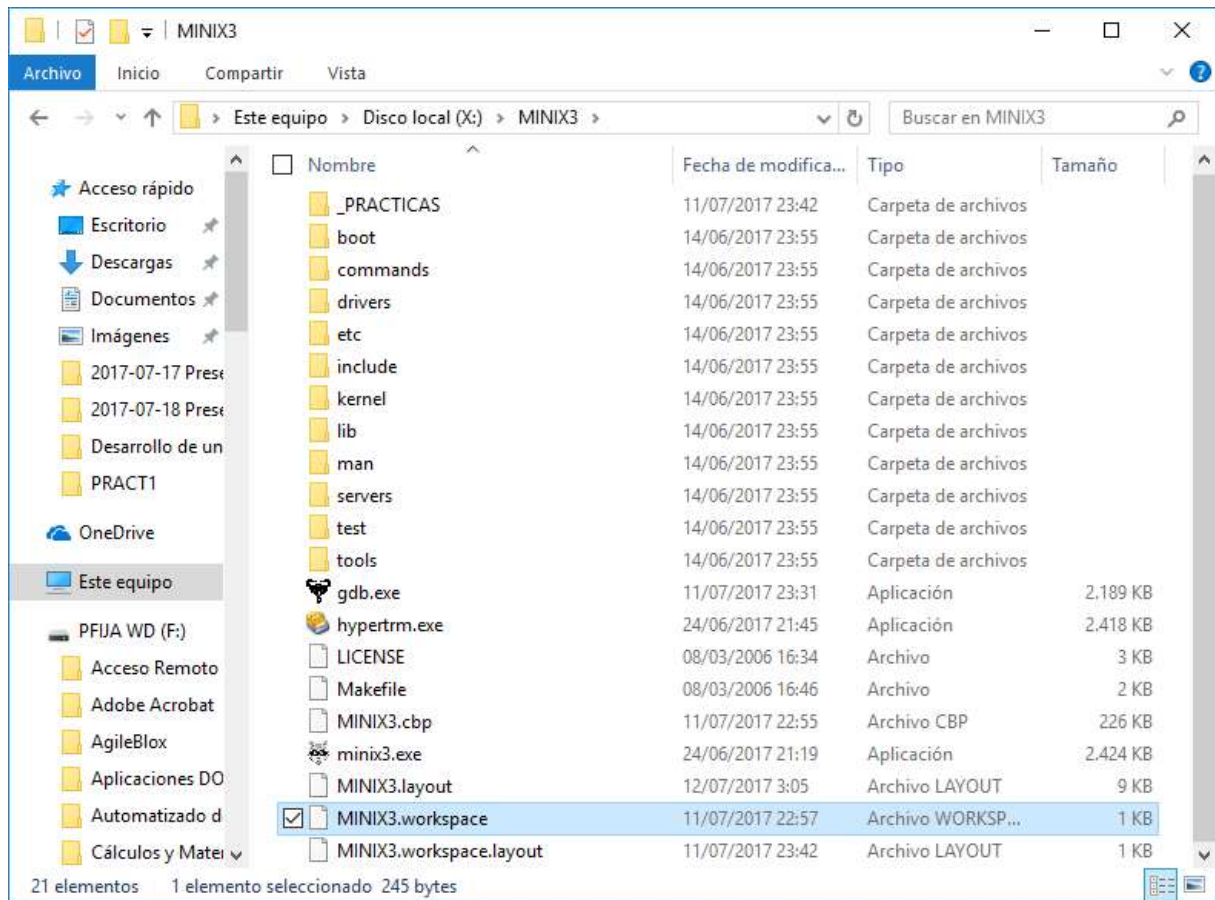
Para navegar por un proyecto tan voluminoso como MINIX (40 MB de programas) vamos a utilizar el IDE [Codeblocks](#). Pinchamos sobre el icono correspondiente presente en el directorio base del software de prácticas.

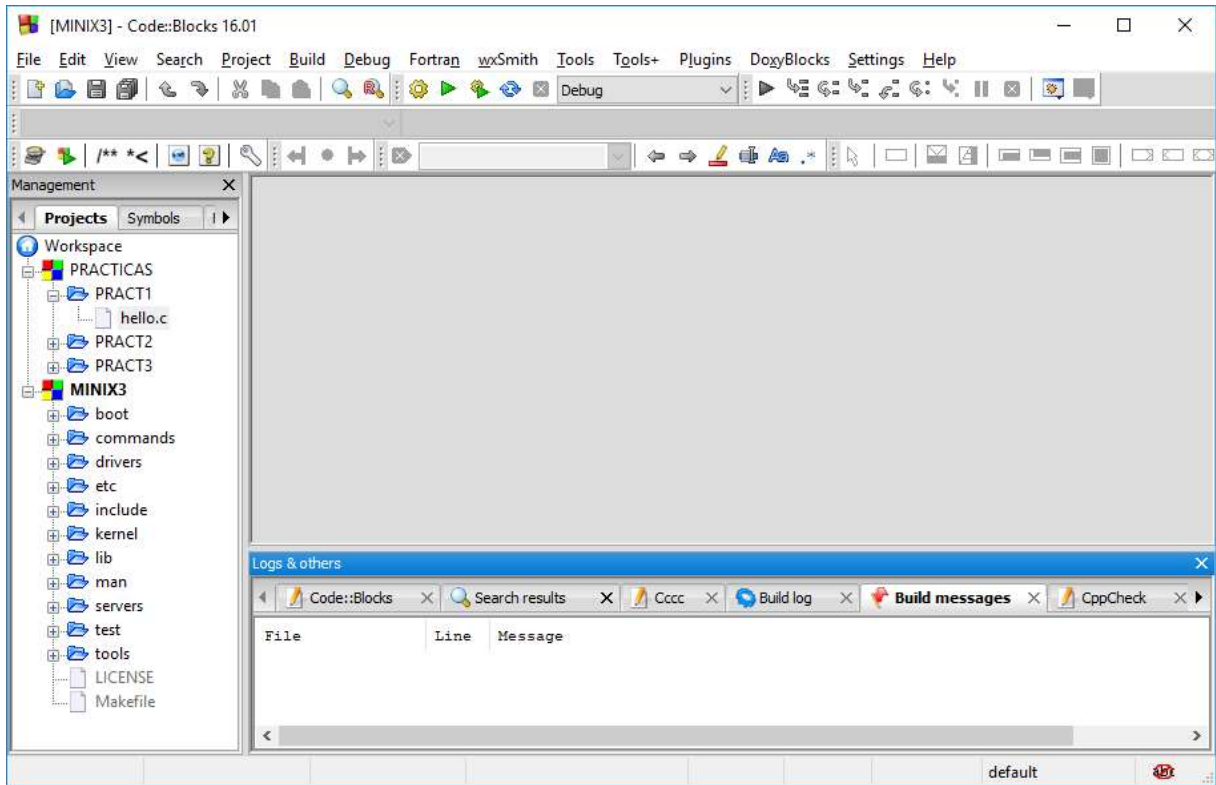


La pantalla inicial de Codeblocks es la siguiente:

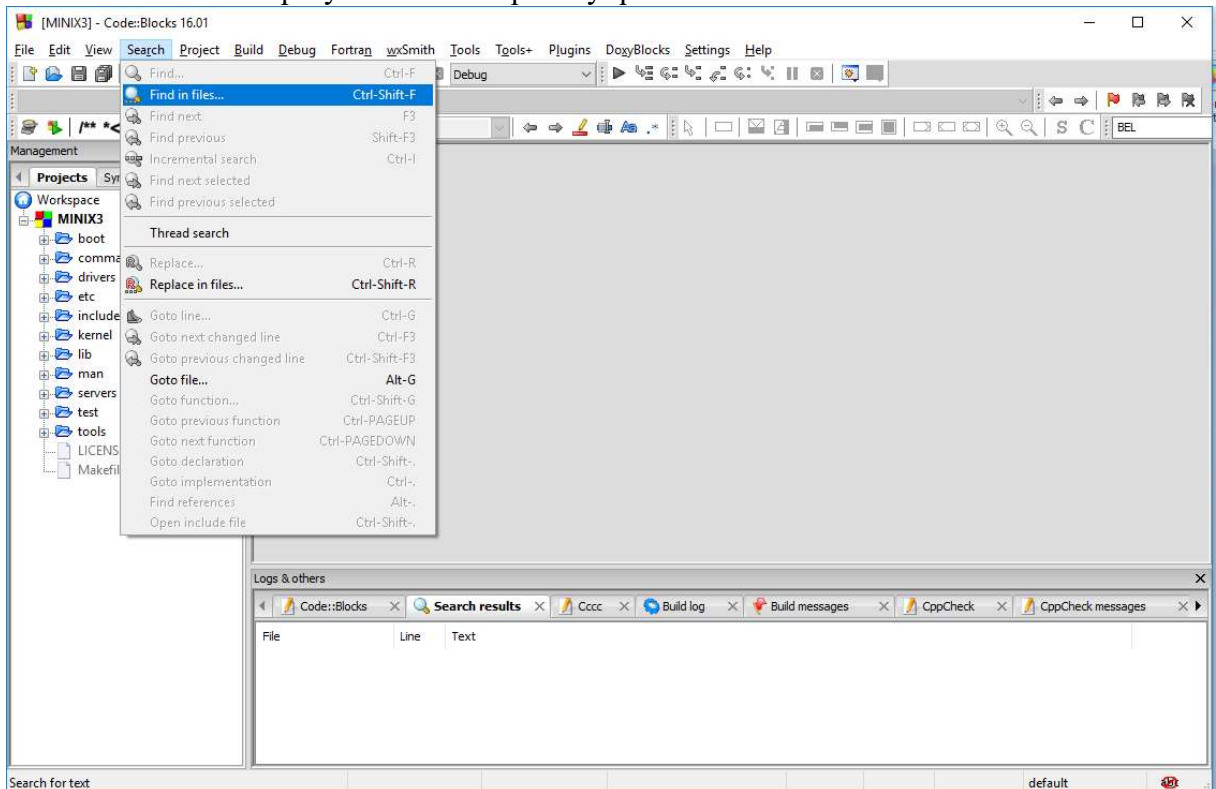


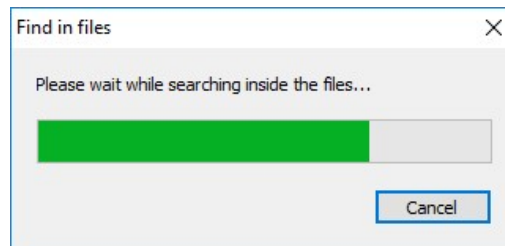
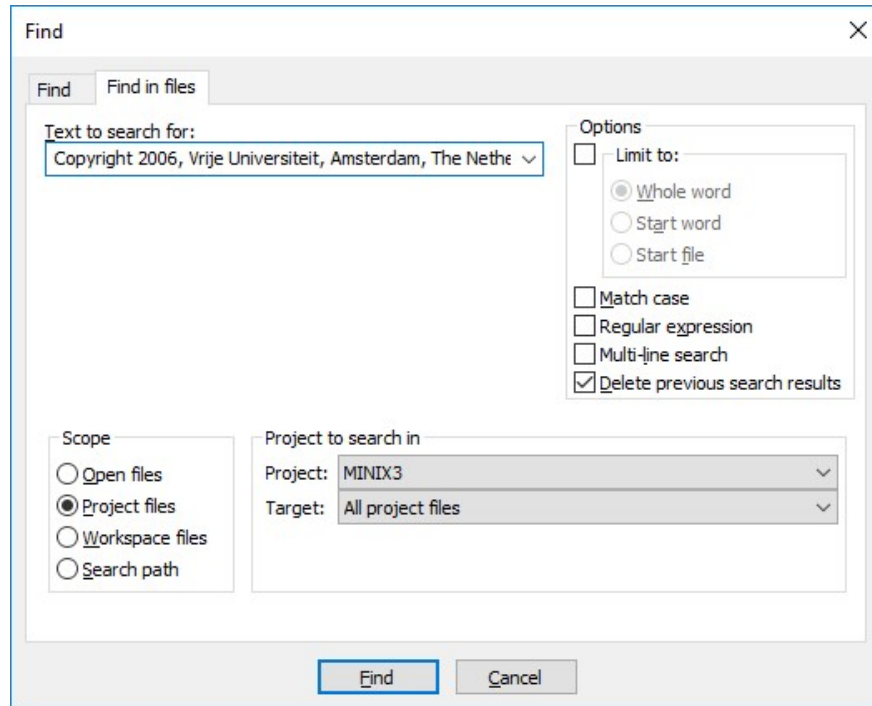
Abrimos el fichero **X:\MINIX3\MINIX3.workspace**

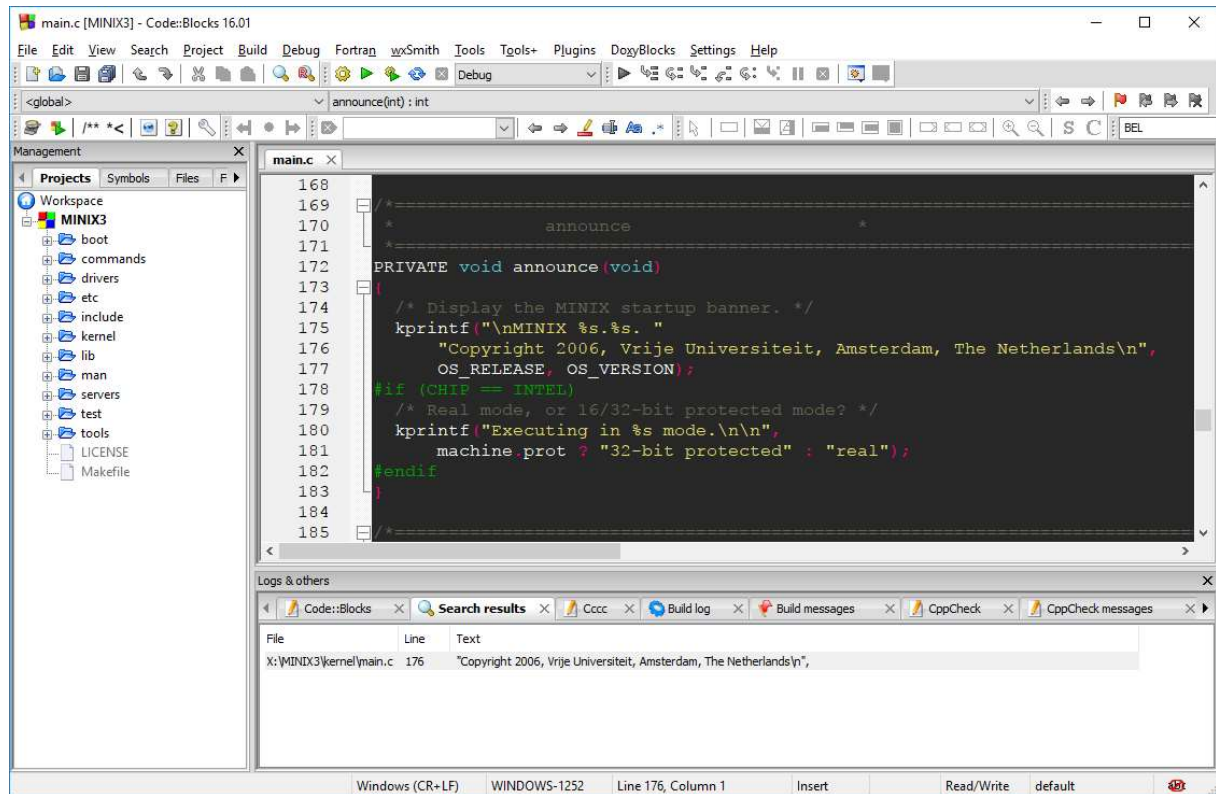




Buscamos en todo el proyecto la línea que hay que modificar:







- Editar una copia del fichero [main.c](#) en la que figure el nuevo mensaje (o editar directamente el fichero `/usr/src/kernel/main.c` de MINIX). Como puede apreciarse, la función que se utiliza para mostrar por pantalla los mensajes del kernel es [kprintf](#). No puede utilizarse la función [printf](#) ya que esta función de biblioteca implica llamadas al sistema (en concreto [write](#)) que sólo pueden realizarse desde un programa de usuario y nunca desde dentro del núcleo.
- Escribir a continuación la sentencia incluida en el código para mostrar el nuevo mensaje que debe aparecer durante el arranque del sistema:

- Si ha hecho la modificación en Windows, una vez modificado el fichero fuente deberá llevarlo a MINIX y copiarlo sobre el fichero `/usr/src/kernel/main.c` siguiendo para ello el procedimiento indicado en apartados anteriores.

- Tras actualizar en MINIX el fichero fuente `/usr/src/kernel/main.c`, tenemos que dar la orden de compilación. Para ello se recomienda primero ejecutar el comando `make` (a secas) en el directorio `/usr/src/kernel`, lo que genera si no hay errores y gracias a un fichero `Makefile` el fichero binario `kernel`. En segundo lugar hay que situarse en el directorio `/usr/src/tools`. En este directorio hay otro fichero `Makefile` que contiene además de las reglas para compilar, las necesarias para construir nuevas imágenes del sistema en el directorio `/boot/image`. Gracias al contenido de estos ficheros no hay que preocuparse de qué ficheros hay que recompilar y en qué orden, ni tampoco hay que preocuparse de dónde deben quedar los ficheros generados, ya que toda esa información está previamente especificada en forma de reglas en el fichero `Makefile`. Teclear: `make install`.
[Nota: La primera vez tarda bastante]

- Indicar qué saludo (*prompt*) muestra MINIX tras la recompilación (\$, #,*):

- Si la compilación ha ido bien, el *prompt* del sistema será “#”, en caso contrario será “*”.
- Las imágenes del núcleo, tanto la original como la que acabamos de generar, se guardan en el directorio `/boot/image`. Listar con `ls -l /boot/image` el contenido de dicho directorio, para comprobar que tenemos al menos dos imágenes. Cuando arrancamos MINIX, la opción **1** del menú de arranque elige para arrancar la imagen original, mientras que la opción **3** elige la última imagen generada.
- Si la recompilación ha ido bien, salir del sistema y reiniciar con la nueva imagen tecleando los comandos:
 - `# halt` // En la shell para detener el sistema MINIX (shutdown)
 - `d0p0s0> exit` // En el monitor para arrancar nuevamente MINIX

Si `exit` falla, teclear `off` (o apagar `qemu`) y volver a ejecutar `minix3.exe`.

En principio no hace falta teclear ninguna opción en el menú de arranque ya que por defecto se toma la opción **3** que es la que arranca la imagen recién modificada, pero a veces es mejor indicar la opción deseada para estar realmente seguros de la imagen que arrancamos. Si todo se ha hecho bien, deberá verse el mensaje introducido, un poco antes de la petición de *login* de Minix, o al presionarse F7. En caso contrario, arrancar con la imagen original –opción 1 del menú de arranque del monitor– y repetir todo el proceso.

```

QEMU (minix3 [qcow]) - Press Ctrl+Alt+G to release grab
Machine  View
0668000  066a000    5968    572   63280    4096  log
067b000  067d000    7056   2412   1356     768  init

MINIX 3.1.2a. Copyright 2020, Pedro Pablo Lopez Rodriguez, Madrid, Spain
Executing in 32-bit protected mode.

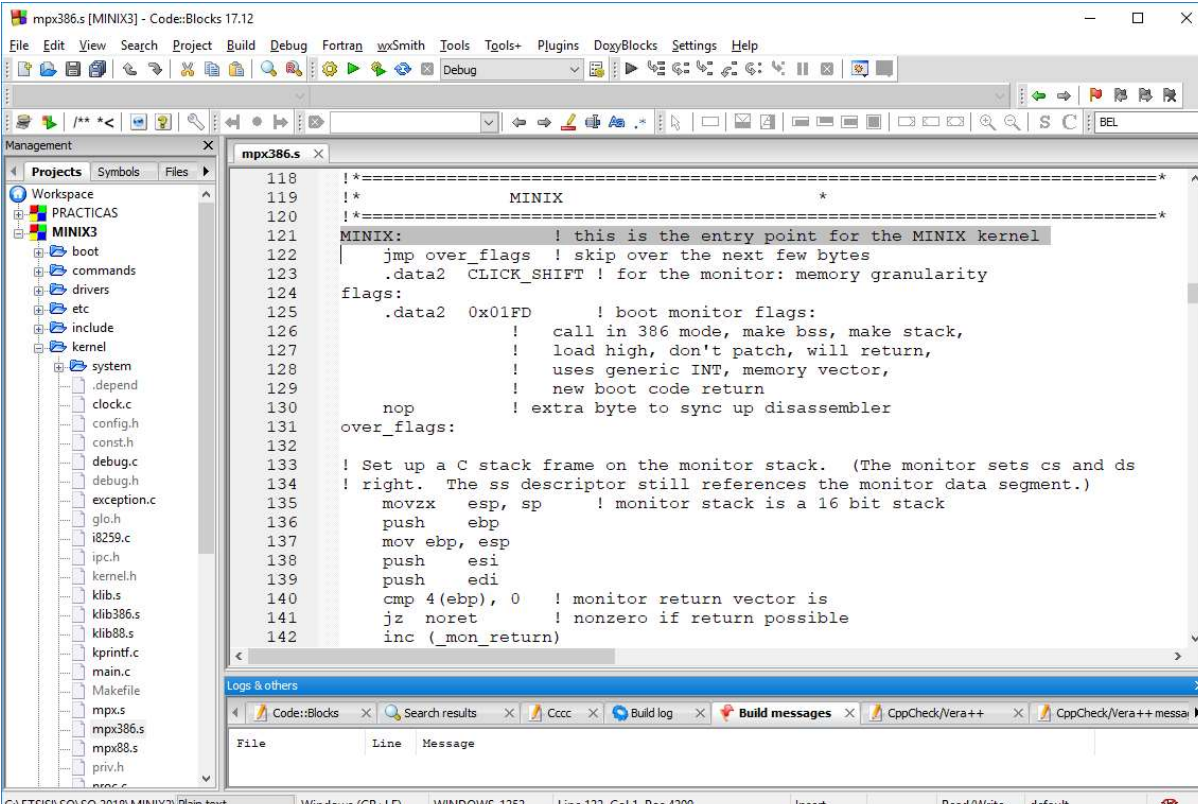
Building process table: pm fs rs ds tty mem log init.
Physical memory: total 32160 KB, system 5700 KB, free 26460 KB.
PCI: video memory for device at 0.2.0: 16777216 bytes
Root device name is /dev/c0d0p0s0
Replacing root

Multiuser startup in progress ...: is cmos.
/dev/c0d0p0s2 is read-write mounted on /usr
/dev/c0d0p0s1 is read-write mounted on /home
Starting services: random inet printer.
Starting daemons: update cron syslogd.
Starting networking: nonamed.
Local packages (start): done.
/dev/rescue is read-write mounted on /boot/rescue

Minix Release 3 Version 1.2a (console)
10.0.0.1 login: _

```

Ahora ilustraremos la modificación puntual de una parte de Minix escrita en ensamblador estableciendo el registro hardware de depuración **DR0** de la máquina con el número de matrícula del alumno, por ejemplo 0987. Lo que queremos es escribir ese valor (como si fuera el número hexadecimal 0x00000987) en el registro DR0 que MINIX no utiliza para nada. Un sitio apropiado para colocar esas instrucciones sería el punto de entrada del núcleo de MINIX, que corresponde a la función **MINIX** presente en el fichero [/usr/src/kernel/mpx386.s](#):



```

118  !*-----*
119  !*                MINIX                *
120  !*-----*
121  MINIX:                ! this is the entry point for the MINIX kernel
122  | jmp over_flags ! skip over the next few bytes
123  |.data2 CLICK_SHIFT ! for the monitor: memory granularity
124  flags:
125  |.data2 0x01FD ! boot monitor flags:
126  | ! call in 386 mode, make bss, make stack,
127  | ! load high, don't patch, will return,
128  | ! uses generic INT, memory vector,
129  | ! new boot code return
130  | nop ! extra byte to sync up disassembler
131  over_flags:
132
133  ! Set up a C stack frame on the monitor stack. (The monitor sets cs and ds
134  ! right. The ss descriptor still references the monitor data segment.)
135  movzx esp, sp ! monitor stack is a 16 bit stack
136  push ebp
137  mov ebp, esp
138  push esi
139  push edi
140  cmp 4(ebp), 0 ! monitor return vector is
141  jz noret ! nonzero if return possible
142  inc (_mon_return)

```

Las instrucciones de ensamblador necesarias serían:

```
mov eax,0x00000987    ! lleva ese valor al registro acumulador EAX
mov dr0,eax           ! lleva el contenido de EAX a DR0
```

pudiendo ubicarse esas instrucciones inmediatamente después de la etiqueta **over_flags**.

Modifique con el editor **mined** el fichero **/usr/src/mpx386.s** añadiendo esas dos instrucciones, **pero poniendo el número de matrícula del alumno** en vez de 0x00000987. Recompile MINIX como se hizo anteriormente (**make install**). Reinicie MINIX con el comando **halt** y **exit**. Si todo se ha hecho bien, tras reiniciar, ejecutar **Ctrl + Alt + 2** para ir al monitor de qemu, y en el monitor de qemu ejecutar el comando **info registers**. Debe aparecer en DR0 el número de matrícula del alumno:

```

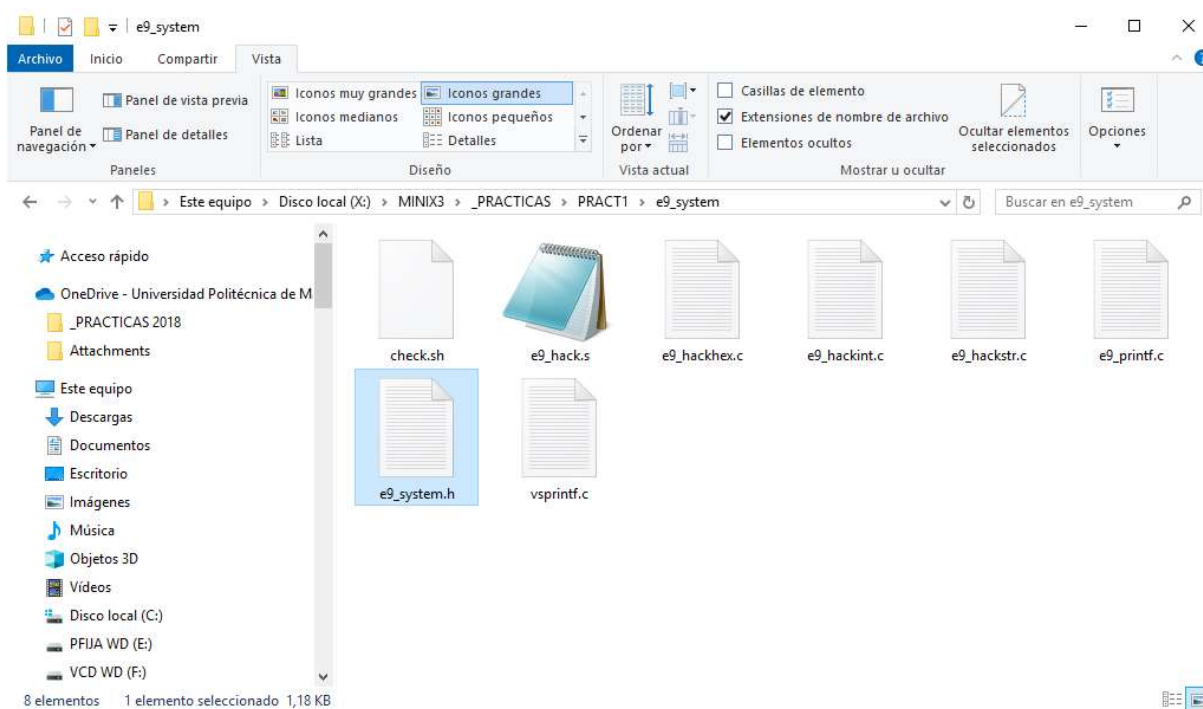
QEMU (minix3 [qcow])
Machine View
ES =001B 00007000 0000ba93 00409300 DPL=0 DS [-WA]
CS =0030 00001000 00005e7f 00409a00 DPL=0 CS32 [-R-]
SS =001B 00007000 0000ba93 00409300 DPL=0 DS [-WA]
DS =001B 00007000 0000ba93 00409300 DPL=0 DS [-WA]
FS =000d 00007000 0000bfff 0040b300 DPL=1 DS [-WA]
GS =000d 00007000 0000bfff 0040b300 DPL=1 DS [-WA]
LDT=0078 0000a11c 00000027 00408200 DPL=0 LDT
TR =0040 00009c9c 00000067 00408900 DPL=0 TSS32-aui
GDT= 00009d04 000003b7
IDT= 00008148 000003bf
CR0=00000011 CR2=00000000 CR3=00000000 CR4=00000000
DR0=00000987 DR1=0baca1a0 DR2=00000000 DR3=00000000
DR6=ffff0ff0 DR7=00000400
EFER=0000000000000000
FCW=037f FSW=0000 [ST=0] FTW=00 MXCSR=00001f80
FPR0=0000000000000000 0000 FPR1=0000000000000000 0000
FPR2=0000000000000000 0000 FPR3=0000000000000000 0000
FPR4=0000000000000000 0000 FPR5=0000000000000000 0000
FPR6=0000000000000000 0000 FPR7=0000000000000000 0000
XMM00=00000000000000000000000000000000 XMM01=00000000000000000000000000000000
XMM02=00000000000000000000000000000000 XMM03=00000000000000000000000000000000
XMM04=00000000000000000000000000000000 XMM05=00000000000000000000000000000000
XMM06=00000000000000000000000000000000 XMM07=00000000000000000000000000000000
(qemu) info registers

```

4 INCORPORACIÓN DE UNA BIBLIOTECA AL NÚCLEO DE MINIX

En el apartado anterior tan solo se han realizado modificaciones triviales del núcleo de MINIX. Ahora vamos a mostrar cómo se añade al núcleo una nueva biblioteca que nos permita escribir por el [puerto 0xE9](#) (dirección 0x00E9 del espacio direccionamiento de E/S del procesador x86). En la máquina virtual qemu puede conectarse dicho puerto a un terminal como el hyperterminal visto en el [apartado 2.5](#). Esta posibilidad nos será de utilidad cuando queramos depurar el núcleo de MINIX ya que podremos visualizar por el terminal trazas de la ejecución sin que dichas trazas interfieran con los mensajes propios del núcleo a través de la pantalla.

La biblioteca que vamos a incorporar al núcleo se encuentra en el subdirectorio **e9_system** del directorio base de la práctica 1:



El fichero de cabeceras de la biblioteca es **e9_system.h**. Como se ve en la figura de la página siguiente, la biblioteca **e9_system** ofrece cinco funciones:

`e9_hack`, `e9_hackstr`, `e9_hackhex`, `e9_hackint` y `e9_printf`

Las cuatro primeras funciones permiten enviar por el puerto 0xE9 un carácter, un string, un número formateado en hexadecimal y un número formateado en decimal, respectivamente. La última función (`e9_printf`) es análoga a la función [kprintf](#) (o [printf](#) de `stdio.h`) pero escribe por el puerto 0xE9 en vez de por la pantalla (o por la salida estándar en el caso de `printf`).

```

1  /* ----- */
2  /*                               e9_system.h                               */
3  /* ----- */
4  /*                               definicion modulo/unidad/biblioteca       */
5  /* ----- */
6  /* ----- */
7  /* [Qemu-devel] [PATCH] Port E9 hack, for debugging purposes:          */
8  /* ----- */
9  /* https://lists.gnu.org/archive/html/qemu-devel/2005-01/msg00169.html */
10 /* ----- */
11
12 #ifndef H_E9_SYSTEM
13 #define H_E9_SYSTEM
14
15 void e9_hack ( char car );          /* escribe car en el puerto 0xE9 */
16
17 void e9_hackstr ( char * str );    /* escribe str por el puerto 0xE9 */
18
19 void e9_hackint ( int n );         /* escribe n por el puerto 0xE9 */
20
21 void e9_hackhex ( unsigned n, unsigned ancho ); /* escribe n en hexad. */
22
23 int e9_printf ( const char * format, ... ); /* printf por 0xE9 */
24
25 #endif /* H_E9_SYSTEM */
26

```

La implementación de las cinco funciones anteriores se hace en cinco ficheros diferentes: **e9_hack.s** (fichero en ensamblador), **e9_hackstr.c**, **e9_hackhex.c**, **e9_hackint.c** y **e9_printf.c**.

```

1  ! ----- !
2  !                               e9_hack.s                               !
3  ! ----- !
4  !                               implementacion modulo/unidad/biblioteca     !
5  ! ----- !
6
7  .sect .text; .sect .rom; .sect .data; .sect .bss
8  .extern _e9_hack
9  .sect .text
10
11 _e9_hack:
12
13     push ebp
14     mov  ebp,esp
15     movb al,8(ebp)
16
17     outb 0xE9 ! escribe el byte contenido en el registro AL en el puerto 0xE9
18             ! AL (8 bits de menor peso del registro acumulador AX o EAX)
19
20     leave
21     ret
22

```

La programación de las funciones `e9_hack`, `e9_hackstr`, `e9_hackhex` y `e9_hackint` es muy sencilla. Por el contrario la programación de la función `e9_printf` resulta mas complicada y se apoya en la implementación de la función `vsprintf` contenida en el fichero `vsprintf.c` obtenido adaptando un fichero de la [versión 0,11 de Linux](#) (año 1991). Ni Tanenbaum ni yo nos hacemos responsables de las feas palabrotas incluidas por Linus Torvalds en dicho fichero.

Llegados aquí está clara la tarea que debe realizar el alumno:

Tras el punto del programa `main.c` desde donde se escribía por pantalla la línea:

```
MINIX 3.1.2a. (C) 2020, Pedro Pablo Lopez Rodriguez, Madrid, Spain
```

(pero poniendo en vez del nombre del profesor, el nombre del alumno) el alumno debe añadir una llamada a la función `e9_printf` de la biblioteca `e9_system` que escriba ese mismo mensaje pero por el puerto `0xE9` en vez de por la pantalla como hace la función `kprintf`.

Los pasos a seguir son los siguientes:

- 1) En Windows llevar el directorio `e9_system` (con todo su contenido) al directorio `Y:\hdb`
- 2) Arrancar MINIX 3 sobre la máquina virtual `qemu`.
- 3) En MINIX copiar el directorio `e9_system` del directorio de Windows `Y:\hdb` al directorio `/root/pract1/e9_system`, para lo cual se necesitarán los comandos:

```
# cd /root/pract1
# mtools copy /dev/c0d1p0:/e9_system .
```

- 4) Comprobar que los ficheros de la biblioteca se han copiado correctamente, haciendo uso del script `check.sh`:

```
# cd /root/pract1/e9_system
# sh check.sh
# ls
# rm -f *.o
```

- 5) Copiar al fichero de fuentes del núcleo de MINIX todos los ficheros necesarios:

```
# cp e9_system.h /usr/src/kernel
# cp e9_hack.s e9_hackstr.c e9_hackhex.c e9_hackint.c /usr/src/kernel
# cp e9_printf.c vsprintf.c /usr/src/kernel
```

Alternativamente y de manera mucho mas escueta:

```
# cp *.[hsc] /usr/src/kernel
```

- 6) Ir a `/usr/src/kernel` y modificar con el editor `minied` el fichero **Makefile**, añadiendo como ficheros objeto necesarios para la obtención del fichero ejecutable del núcleo (kernel) todos los ficheros correspondientes a los ficheros copiados:

```
OBJS = start.o protect.o klib.o table.o kprintf.o main.o proc.o \
      i8259.o exception.o system.o clock.o utility.o debug.o \
      e9_hack.o e9_hackstr.o e9_hackhex.o e9_hackint.o \
      e9_printf.o vsprintf.o
```

- 7) Abrir con `minied` el fichero **main.c**, incluir (directiva `#include`) en él la nueva biblioteca **e9_system.h** y añadir la línea en la que se llame a la función de la biblioteca **e9_printf**, de manera que se escriba por el puerto `0xE9` la línea:

```
MINIX 3.1.2a. (C) 2020, Pedro Pablo Lopez Rodriguez, Madrid, Spain
```

(pero poniendo en vez del nombre del profesor, el nombre del alumno).

- 8) Recompila el núcleo de minix (`cd ../tools ; make install`) y cruce los dedos para que no se haya equivocado en ninguno de los pasos y la compilación y generación de la imagen del núcleo sean correctas.
- 9) Cierre MINIX (`# halt`) y la máquina virtual (`d0p0s0> off`).
- 10) En Windows abra el fichero de configuración de la máquina virtual `Y:\minix3.conf` y configure el puerto `0xE9` (consola de depuración) para que se asocie con el puerto `tcp/ip` `9923` de la dirección local `127.0.0.1`.

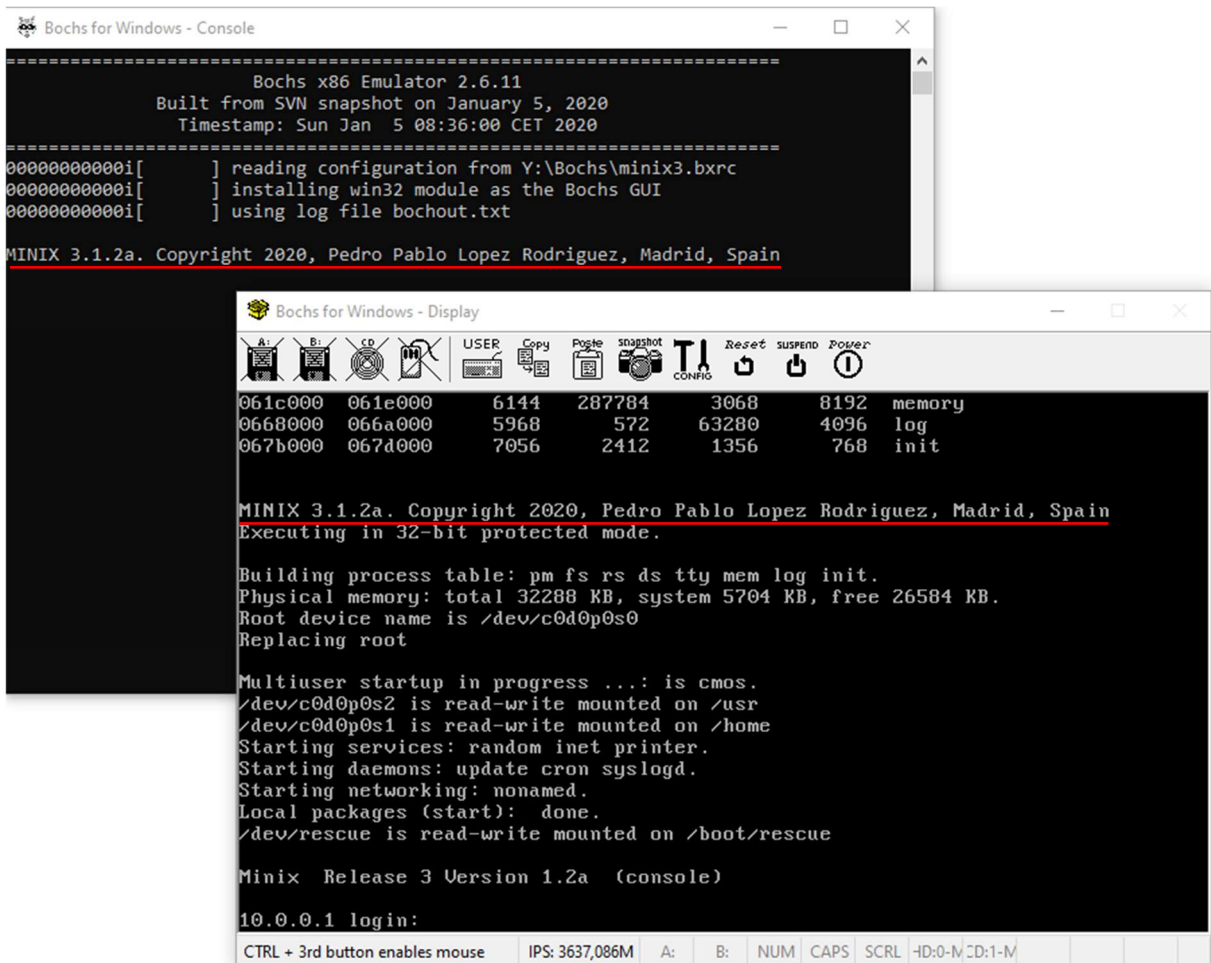
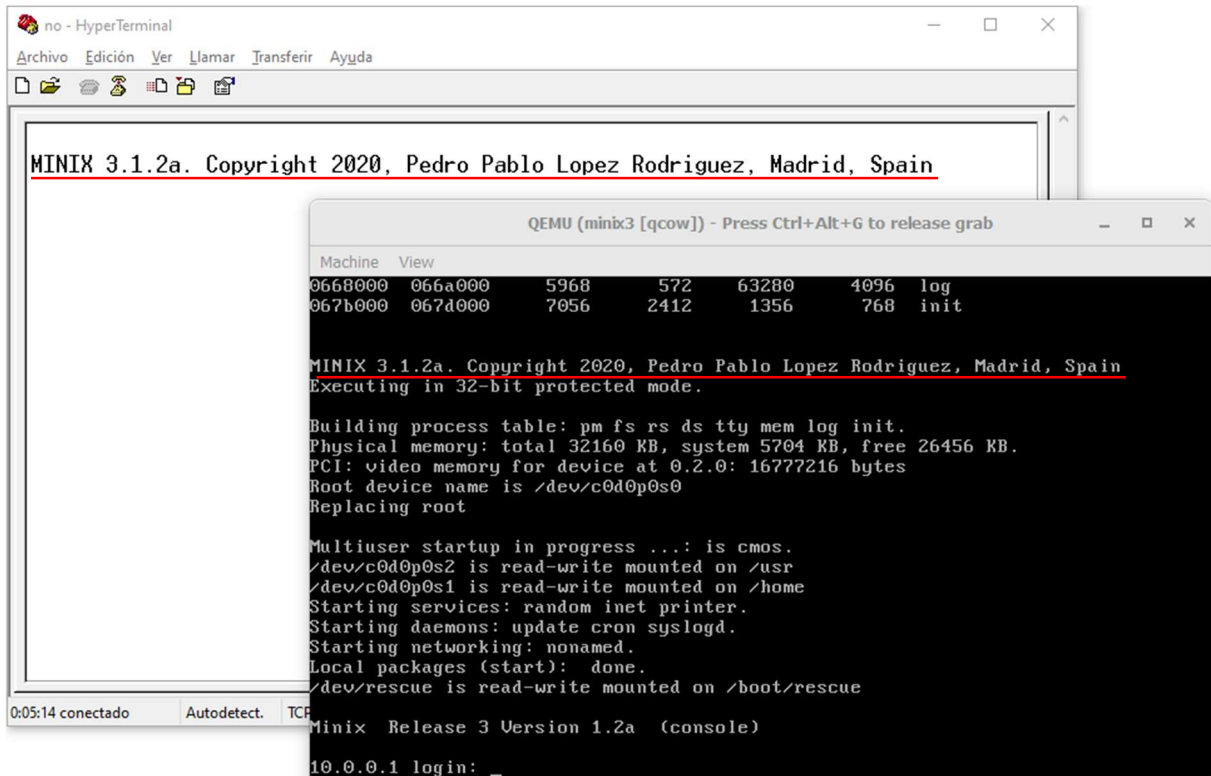
```
-debugcon tcp:127.0.0.1:9923,server # (consola de depuracion)
```

Comente (`#`) la línea correspondiente al puerto serie que estaba utilizando antes `127.0.0.1:9923`.

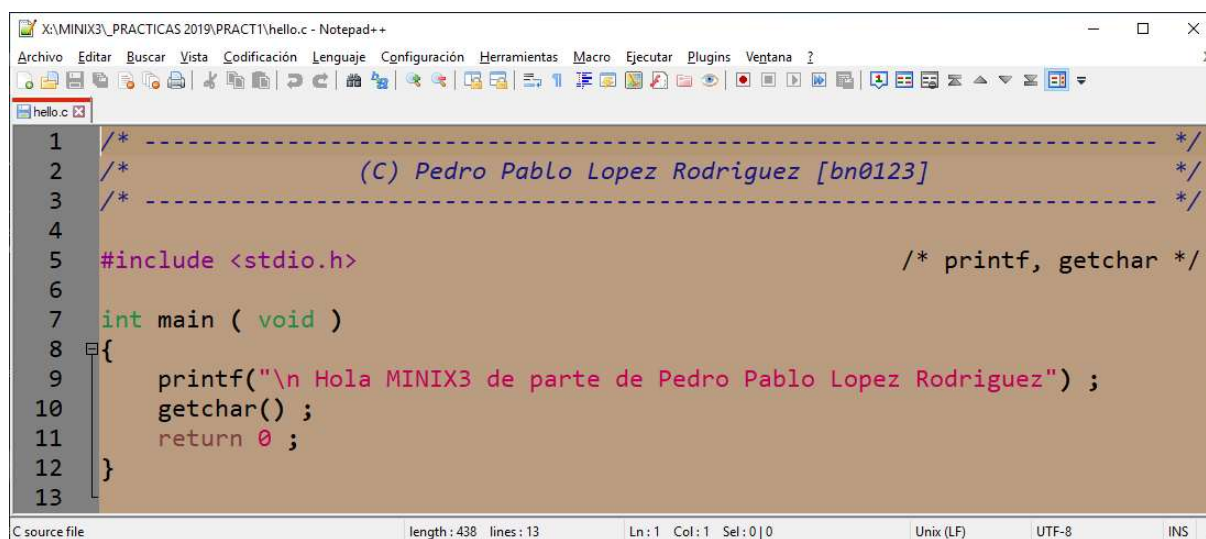
- 11) Arranque MINIX sobre la máquina virtual `qemu`, la cual quedará a la espera de que se conecte un cliente a través de `127.0.0.1:9923`.
- 12) Conecte el Hyperterminal a la dirección `127.0.0.1:9923`, y ... voilà (siguiente página). Pruebe también dentro de Minix a pulsar la tecla `F7`.

Los pasos 10), 11) y 12) pueden simplificarse mucho haciendo uso de la máquina virtual `Bochs`:

- 10) En Windows pinchamos con el ratón sobre el icono `X:\MINIX3\qcow2raw.cmd` para convertir la imagen de disco duro `Y:\minix3hd.qcow` (`qcow`) a `Y:\minix3hd.img` (`raw`).
- 11) Lanzamos la máquina virtual MINIX **sobre `Bochs`** pinchando con el ratón sobre el icono `X:\MINIX3\minix3 bochs (raw).exe`, y ... voilà (siguiente página, abajo).



Como punto final, el alumno puede tratar de modificar el programa (de usuario) **hello.c** original:



```
1  /* ----- */
2  /*           (C) Pedro Pablo Lopez Rodriguez [bn0123]           */
3  /* ----- */
4
5  #include <stdio.h>                                           /* printf, getchar */
6
7  int main ( void )
8  {
9      printf("\n Hola MINIX3 de parte de Pedro Pablo Lopez Rodriguez" ) ;
10     getchar() ;
11     return 0 ;
12 }
13
```

para que escriba ese mensaje (además de en la salida estándar mediante printf) en el puerto 0xE9 haciendo uso de la biblioteca **e9_system.h**. Tal y como están los ficheros en /root/pract1 bastaría hacer:

- 1) Mover /root/pract1/hello.c al subdirectorio **e9_system** (# mv hello.c e9_system) y modificar /root/pract1/e9_system/hello.c incluyendo la biblioteca e9_system.h y añadiendo una segunda línea idéntica a la del printf, pero con la función e9_printf.
- 2) Compilar el nuevo fichero hello.c con el comando:

```
# cc hello.c e9_hack.s e9_hackstr.c e9_printf.c vsprintf.c -o hello
```

El alumno tras compilar con éxito deberá ejecutar en el intérprete de comandos el fichero ejecutable (hello) obtenido.

```
# ./hello
```

- Explique al profesor en el siguiente recuadro qué sucede en ese momento y a qué se debe (agrande el recuadro si necesita extenderse en la explicación):

5 ANEXOS

5.1 Manual del editor “elvis”

Para crear un archivo nuevo (sea **nombres**) con elvis teclear: **elvis**

nombres

Posteriormente podremos modificarlo abriéndolo de la misma forma:

elvis nombres

Algunos comandos útiles:

i	Pasar a modo inserción
o	Pasa a modo inserción abriendo una línea nueva debajo de la actual
a	Pasa a modo inserción añadiendo a la derecha de la letra actual
<Esc>	Salir del modo inserción
k	Mover cursor hacia arriba
j	Mover cursor hacia abajo
l	Mover cursor hacia la derecha
h	Mover cursor hacia la izquierda
r	Reemplaza un caracter
R	Reemplaza caracteres hasta pulsar <Esc>
d< >	Borra un caracter. < > significa espacio
dw	Borra una palabra
dd	Borra una línea
3dd	Borra tres líneas
.	Repite la acción anterior
u	Deshace la acción anterior
/cadena	Buscar la cadena de caracteres indicada
n	Repetir la búsqueda anterior
Y	Copia una línea al buffer
3Y	Copia tres líneas al buffer
p	Pega el contenido del buffer
:q!	Salir sin salvar
:w	Guardar los cambios realizados hasta el momento
ZZ	Salir salvando

5.2 Manual del editor “mined”

Command: `mined` - MINIX editor **Syntax:** `mined [file]`
Flags: (none)
Examples: `mined /user/ast/book.3` # Edit an existing file
`mined` # Call editor to create a new file
`ls -l | mined` # Use *mined* as a pager to inspect
listings

Mined is a simple screen editor. At any instant, a window of 24 lines is visible on the screen. The current position in the file is shown by the cursor. Ordinary characters typed in are inserted at the cursor. Control characters and keys on the numeric keypad (at the right-hand side of the keyboard) are used to move the cursor and perform other functions.

Commands exist to move forward and backward a word, and delete words either in front of the cursor or behind it. A word in this context is a sequence of characters delimited on both ends by white space (space, tab, line feed, start of file, or end of file). The commands for deleting characters and words also work on line feeds, making it possible to join two consecutive lines by deleting the line feed between them.

The editor maintains one save buffer (not displayed). Commands are present to move text from the file to the buffer, from the buffer to the file, and to write the buffer onto a new file. If the edited text cannot be written out due to a full disk, it may still be possible to copy the whole text to the save buffer and then write it to a different file on a different disk with CTRL-Q. It may also be possible to escape from the editor with CTRL-S and remove some files.

Some of the commands prompt for arguments (file names, search patterns, etc.). All commands that might result in loss of the file being edited prompt to ask for confirmation.

A key (command or ordinary character) can be repeated *n* times by typing *ESC n key* where *ESC* is the 'escape' key.

Forward and backward searching requires a regular expression as the search pattern. Regular expressions follow the same rules as in the UNIX editor, *ed*. These rules can be stated as:

Any displayable character matches itself.

. (period) matches any character except line feed.

^ (circumflex) matches the start of the line.

\$ (dollar sign) matches the end of the line.

\c matches the character *c* (including period, circumflex, etc).

[*string*] matches any of the characters in the string.

[^*string*] matches any of the characters except those in the string.

[*x-y*] matches any characters between *x* and *y* (e.g., [a-z]).

*pattern** matches any number of occurrences of *pattern*.

Some examples of regular expressions are:

The boy	matches the string 'The boy'
^\$	matches any empty line.
^.\$	matches any line containing exactly 1 character
^A.*\.\$	matches any line starting with an A, ending with a period.
^[A-Z]*\$	matches any line containing only capital letters (or empty).
[A-Z0-9]	matches any line containing a capital letter or a digit.
.*X\$	matches any line ending in 'X'
A.*B	matches any line containing an 'A' and then a 'B'

Control characters cannot be entered into a file simply by typing them because all of them are editor commands. To enter a control character, depress the ALT key, and then while holding it down, hit the ESC key. Release both ALT and ESC and type the control character. Control characters are displayed in reverse video.

The *mined* commands are as follows:

CURSOR MOTION

arrows	Move the cursor in the indicated direction
CTRL-A	Move cursor to start of current line
CTRL-Z	Move cursor to end of current line
CTRL-^	Move cursor to top of screen
CTRL-<u>_</u>	Move cursor to end of screen
CTRL-F	Move cursor forward to start of next word
CTRL-B	Move cursor backward to start of previous word

SCREEN MOTION

Home key	Move to first character of the file
End key	Move to last character of the file
PgUp key	Scroll window up 23 lines (closer to start of the file)
PgDn key	Scroll window down 23 lines (closer to end of the file)
CTRL-U	Scroll window up 1 line
CTRL-D	Scroll window down 1 line

MODIFYING TEXT

Del key	Delete the character under the cursor
Backspace	Delete the character to left of the cursor
CTRL-N	Delete the next word
CTRL-P	Delete the previous word
CTRL-T	Delete tail of line (all characters from cursor to end of line)
CTRL-O	Open up the line (insert line feed and back up)
CTRL-G	Get and insert a file at the cursor position

BUFFER OPERATIONS

CTRL-@	Set mark at current position for use with CTRL-C and CTRL-K
CTRL-C	Copy the text between the mark and the cursor into the buffer
CTRL-K	Delete text between mark and cursor; also copy it to the buffer
CTRL-Y	Yank contents of the buffer out and insert it at the cursor
CTRL-Q	Write the contents of the buffer onto a file

MISCELLANEOUS

Numeric +	Search forward (prompts for regular expression)
numeric -	Search backward (prompts for regular expression)
numeric 5	Display the file status
CTRL-]	Go to specific line
CTRL-R	Global replace <i>pattern</i> with <i>string</i> (from cursor to end)
CTRL-L	Line replace <i>pattern</i> with <i>string</i>
CTRL-W	Write the edited file back to the disk
CTRL-X	Exit the editor
CTRL-S	Fork off a shell (use CTRL-D to get back to the editor)
CTRL-\	Abort whatever the editor was doing and wait for command
CTRL-E	Erase screen and redraw it
CTRL-V	Visit (edit) a new file

Author

Mined was designed by Andy Tanenbaum and written by Michiel Huisjes.