Entorno de trabajo

Miquel Albert
Gerard Enrique

Universitat Oberta de Catalunya

Índice

Introducción

El entorno de trabajo que utilizaremos para desarrollar los problemas y las prácticas de programación será un PC basado en procesadores x86-64 (Intel64 o AMD64) sobre el cual se ejecutará un sistema operativo Linux de 64 bits, la versión Linux que proponemos es Linux Mint de 64 bits (basada en Ubuntu), pero se pueden utilizar otras versiones Linux de 64 bits.

Para la instalación de Linux Mint, como se explica más adelante, tendréis que descargar un archivo con una imagen ISO desde el sitio web de Linux Mint. El archivo ISO permite hacer una instalación del sistema en una máquina virtual, de VirtualBox por ejemplo, y también permite hacer la instalación de forma nativa.

Los lenguajes de programación que utilizaremos para escribir el código fuente de los problemas y de las prácticas de programación de la asignatura serán el lenguaje C para diseñar el programa principal y las operaciones de E/S y el lenguaje ensamblador x86-64 para implementar funciones concretas y ver cómo trabaja esta arquitectura a bajo nivel.

Herramientas propuestas

Editor: geany
Ensamblador: *yasm*
Enlazador: *gcc (internamente llama al enlazador ld)*
Compilador de C: *gcc*
Entorno a depuración: *kdbg* que utiliza *gdb*

Más adelante en este documento se explica cómo instalar y utilizar estas herramientas.

Proceso de desarrollo de un programa escrito en lenguaje ensamblador:

1. Edición del código fuente ensamblador (*geany*).
2. Ensamblaje del código fuente ensamblador y generación del código objeto (*yasm*).
3. Enlazado del código objeto y generación del código ejecutable (gcc).
4. Depuración del código ejecutable para la corrección de errores (*kdbg*).
5. Ejecución del programa.

Proceso de desarrollo de un programa escrito en lenguaje C:

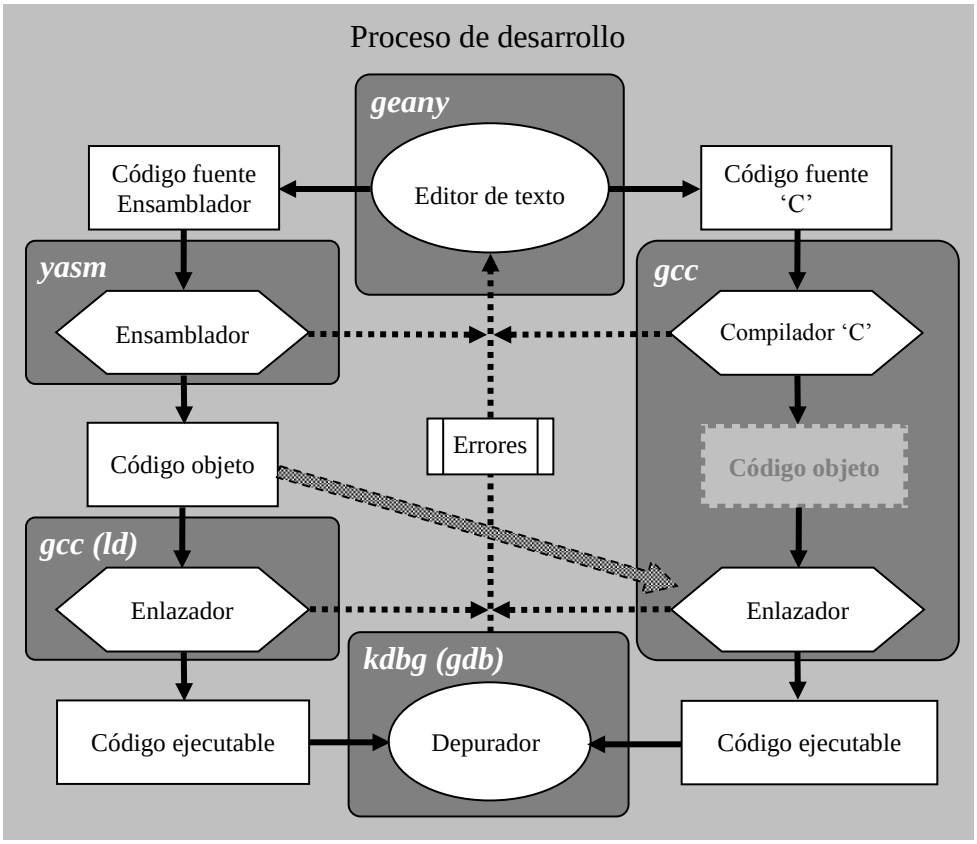
1. Edición del código fuente C (*geany*).
2. Compilación y enlazado del código fuente C y generación del código ejecutable (*gcc*).
3. Ejecución del programa.

Proceso de desarrollo de un programa escrito en lenguaje C que utiliza subrutinas hechas en ensamblador:

1. Edición del código fuente ensamblador, incluir la definición de las subrutinas necesarias cómo globales (*geany*).
2. Ensamblaje del código fuente ensamblador y generación del código objeto (*yasm*).
3. Edición del código fuente C, incluidas las definiciones y las llamadas a subrutinas en ensamblador (*geany*).
4. Compilación y enlazado del código fuente C con el código objeto generado del ensamblador y generación del código ejecutable (*gcc*).
5. Depuración del código ejecutable para la corrección de errores (*kdbg*).
6. Ejecución del programa.

Cuando en uno de estos procesos se detectan errores, hay que volver al inicio del proceso, modificar el código fuente para corregir los errores y repetir el proceso

cíclicamente hasta obtener un programa ejecutable libre de errores y que tenga la funcionalidad deseada.



Si se utiliza el archivo ISO para crear una máquina virtual dentro del entorno de virtualización VirtualBox, o se instala de forma nativa el sistema utilizando este archivo ISO no es necesario instalar las herramientas.

Si se quiere trabajar con otro sistema operativo Linux de 64bits, será necesario instalar todas las herramientas del entorno de trabajo.

1. Instalación de las herramientas

En este apartado se explica cómo instalar y configurar las herramientas del entorno de trabajo.

1.1. Instalación de VirtualBox

En primer lugar tenéis que comprobar si vuestro ordenador permite ejecutar una máquina virtual con un sistema operativo de 64 bits.
Para hacer la comprobación, en Windows, podéis ejecutar el programa "securable" que encontraréis en el siguiente enlace web:
http://www.grc.com/files/securable.exe
Al ejecutar este programa aparecen dos informaciones importantes: si el procesador es de 64 bits y si dispone de soporte para virtualización, las dos características son necesarias para poder ejecutar una máquina virtual con un sistema operativo de 64 bits dentro de VirtualBox.

En Linux, podéis comprobar si vuestro procesador es de 64 bits y si tiene soporte para virtualización, con el comando *lscpu*.
Abrid un terminal de Linux y ejecutad el comando:

```
$ lscpu
```

Si entre la información que aparece, se muestra la información siguiente:

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
...
Virtualization:        VT-x
```

Significa que sí tenéis soporte para virtualización. Si tenéis un procesador AMD, en la información sobre virtualización aparecerá AMD-V.

Si vuestro ordenador tiene un procesador de 64 bits pero no tiene soporte de virtualización por hardware, tendréis que instalar Linux de 64 bits en una partición del disco, o en un disco externo o memoria USB.
Si no sabéis como hacerlo contactad con el consultor de vuestra aula que os podrá ayudar.

1.1.1. Obtención e instalación del software de VirtualBox

Se puede obtener la última versión del software de VirtualBox a través de la página web siguiente:
`https://www.virtualbox.org/wiki/Downloads`

Encontrareis versiones para Windows, Mac (OS X) y Linux. En el caso de Windows y OS X hay una única versión, en el caso de Linux escoged la opción que se adapte mejor a vuestra distribución.

Una vez hayáis descargado el software ejecutadlo y seguid las indicaciones del mismo.

1.1.2. Instalación a través del gestor de paquetes (Linux)

En el caso de disponer de una distribución Linux, como sistema operativo anfitrión, basada en Debian tenéis la opción de añadir el repositorio de paquetes de VirtualBox a la herramienta de gestión de paquetes de Linux, para hacerlo seguid los pasos siguientes, desde un terminal de Linux:

Ejecutad el comando:

```
 $ wget -q http://download.virtualbox.org/virtualbox/debian/oracle_vbox.asc -O-
| sudo apt-key add -
```

Escribid todo el comando en una sola línea.

Editad el archivo `/etc/apt/sources.list` necesitáis permisos de superusuario para hacerlo, en el caso de Ubuntu y variantes podéis utilizar el comando sudo:

```
sudo gedit /etc/apt/sources.list
```

Añadid una de les líneas siguientes en función de vuestra distribución al archivo virtualbox.lst:

```
deb http://download.virtualbox.org/virtualbox/debian raring contrib
deb http://download.virtualbox.org/virtualbox/debian quantal contrib
deb http://download.virtualbox.org/virtualbox/debian precise contrib
deb http://download.virtualbox.org/virtualbox/debian oneiric contrib
deb http://download.virtualbox.org/virtualbox/debian natty contrib
deb http://download.virtualbox.org/virtualbox/debian maverick contrib non-free
deb http://download.virtualbox.org/virtualbox/debian lucid contrib non-free
deb http://download.virtualbox.org/virtualbox/debian karmic contrib non-free
deb http://download.virtualbox.org/virtualbox/debian hardy contrib non-free
deb http://download.virtualbox.org/virtualbox/debian wheezy contrib
deb http://download.virtualbox.org/virtualbox/debian squeeze contrib non-free
deb http://download.virtualbox.org/virtualbox/debian lenny contrib non-free
```

Por ejemplo, para Ubuntu 13.04 (raring), se puede hacer:

```
sudo sh -c 'echo "deb http://download.virtualbox.org/virtualbox/debian raring
contrib" >> /etc/apt/sources.list.d/virtualbox.list'
```

Escribid todo el comando en una sola línea.
El mismo comando funciona para Linux Mint 15.
Para instalar el software ejecutad los comandos siguientes

```
$ sudo apt-get update
$ sudo apt-get install virtualbox-4.2
```

1.2. Instalación del VirtualBox Extension Pack

Este paquete habilita algunas extensiones, cómo el controlador USB. Descargad el paquete desde la página de descargas:
https://www.virtualbox.org/wiki/Downloads

Se trata de la misma descarga para todas las plataformas (Windows, OS X y Linux)

Abrid el archivo descargado desde el explorador de archivos, directamente se detectará como un complemento de VirtualBox y se abrirá con este programa. Seguid las indicaciones del mismo.

En el caso de Linux, se os pedirá la contraseña del usuario para realizar tareas privilegiadas.

1.3. Añadir un usuario al grupo de VirtualBox (Linux)

En el caso de disponer de una distribución Linux, como sistema operativo anfitrión, es necesario añadir el usuario con el que trabajamos al grupo de usuarios vboxusers para tener acceso a los dispositivos USB desde dentro de las máquinas virtuales creadas con VirtualBox. Para hacerlo ejecutad el comando siguiente:

```
$ sudo usermod -a -G vboxusers nombre_usuario
```

A continuación es necesario salir de la sesión y volver a entrar para activar los cambios.

1.4. Instalación de una máquina virtual

A continuación se explica la instalación de la máquina virtual de la asignatura utilizando el software de virtualización VirtualBox.

Para poder crear la máquina virtual, primero tenéis que de descargar el archivo ISO de Linux Mint.

Es necesario descargar una imagen ISO de 64 bits a través del enlace:
http://www.linuxmint.com/download.php

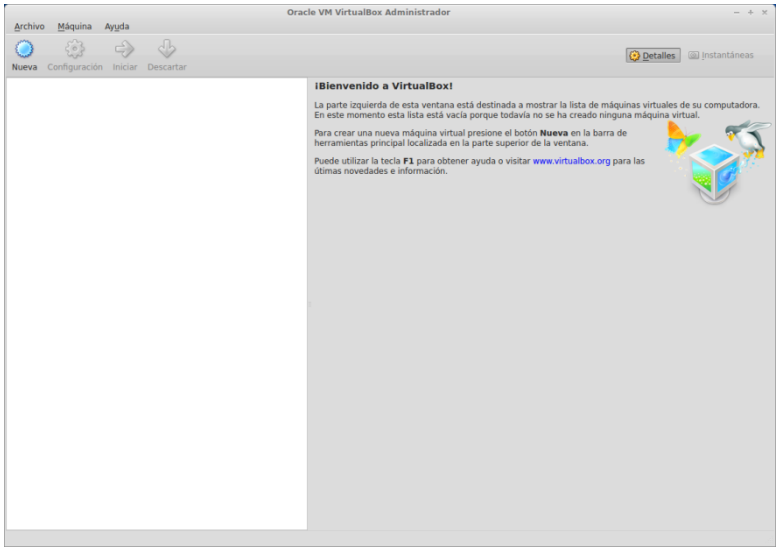
Linux Mint se distribuye en cuatro ediciones diferentes, según el entorno de escritorio: Cinnamon, MATE (basada en GNOME2), KDE y Xfce.

Podéis usar cualquiera de las cuatro ediciones, pero tened presente lo siguiente: las ediciones Cinnamon y KDE necesitan más recursos hardware y son escritorios pensados para trabajar con aceleración de gráficos 3D, las ediciones MATE y Xfce necesitan menos recursos y son más adecuadas para trabajar con ordenadores con menos recursos.

Descargad la versión que queráis, siempre que sea de 64 bits. Nosotros recomendamos utilizar la versión MATE de 64 bits.

En este documento se explicará la instalación a partir de la edición MATE de 64 bits. Una vez descargado el archivo ISO de la última versión, Linux Mint 15 MATE 64 bits, deberíamos tener el archivo siguiente: *linuxmint-15-mate-dvd-64bit.iso*

Abrid VirtualBox y escoged la opción *Nueva*.



En el asistente que aparece proporcionad un nombre a la máquina virtual y escoged el tipo de sistema operativo, Linux, y la versión, Ubuntu de 64 bits (ya que Linux Mint està basada en Ubuntu).

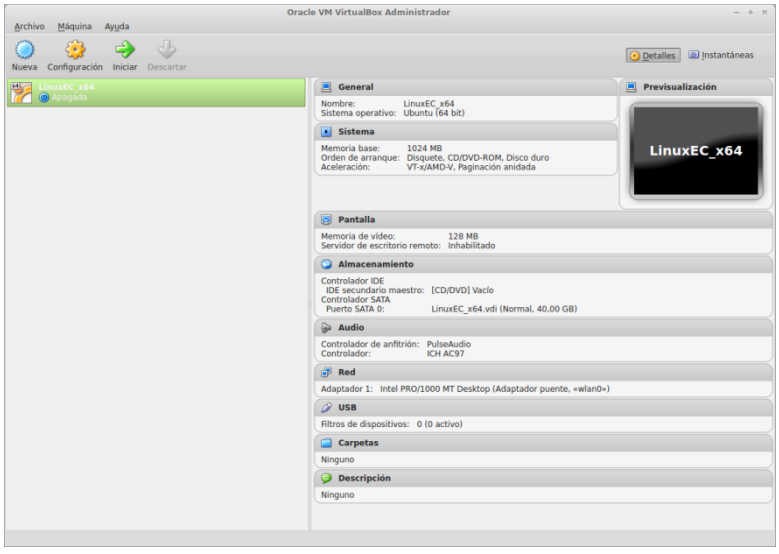
En la pantalla siguiente introducid la cantidad de memoria, se recomienda no sobrepasar el 50% de la memoria del ordenador, normalmente con 1024 MB o menos es suficiente.

En la pantalla siguiente, os pide añadir un disc duro virtual, cread un disco nuevo.

Escoged un disco virtual de tipo VDI (VirtualBox Disk Image) reservado dinámicamente,.

A continuación aparece una pantalla para escoger el tamaño del disco.
Normalmente con 8 o 10 GB es suficiente, pero si tenéis espacio suficiente en el disco podéis indicar un tamaño superior, por ejemplo 40GB. Este será el tamaño máximo, inicialmente no se ocupa todo el tamaño reservado, el archivo del disco duro virtual va creciendo a medida que es necesario

Aceptad los cambios y cread el disco virtual y la máquina virtual, al finalizar el proceso os tiene que aparecer una ventana como la siguiente:



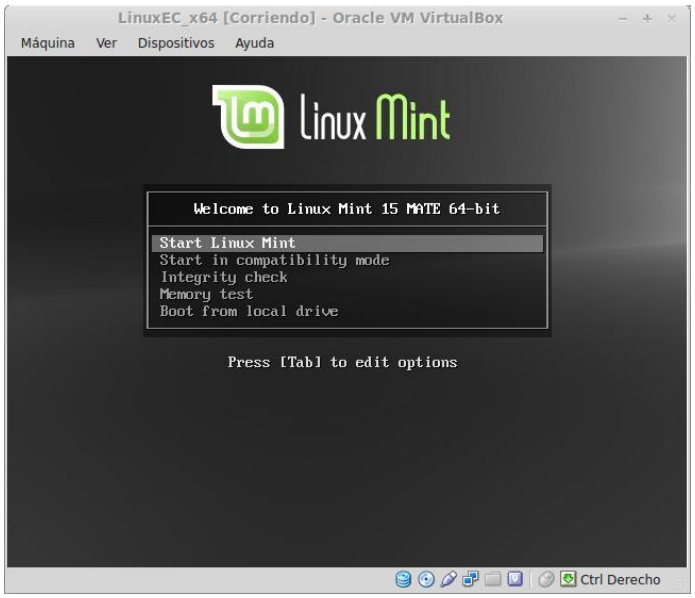
Para evitar problemas con la máquina virtual, os recomendamos cambiar el tamaño de la memoria de vídeo, indicando la máxima posible, 128 MB, pulsando sobre el apartado *Pantalla*.
También os recomendamos cambiar el tipo de adaptador de red, para hacerlo pulsad sobre *Red*, en la ventana que aparece pulsad sobre el desplegable que hay al lado de la etiqueta *Conectado a* y seleccionad *Adaptador puente* (bridge).

Iniciad la nueva máquina pulsando el botón *Iniciar*.
Probablemente aparecerá un mensaje indicando que la máquina virtual captura el control del teclado y que lo podéis liberar pulsando la tecla *Ctrl Derecha*, aceptad el mensaje, opcionalmente podéis marcar que no os vuelva avisar.

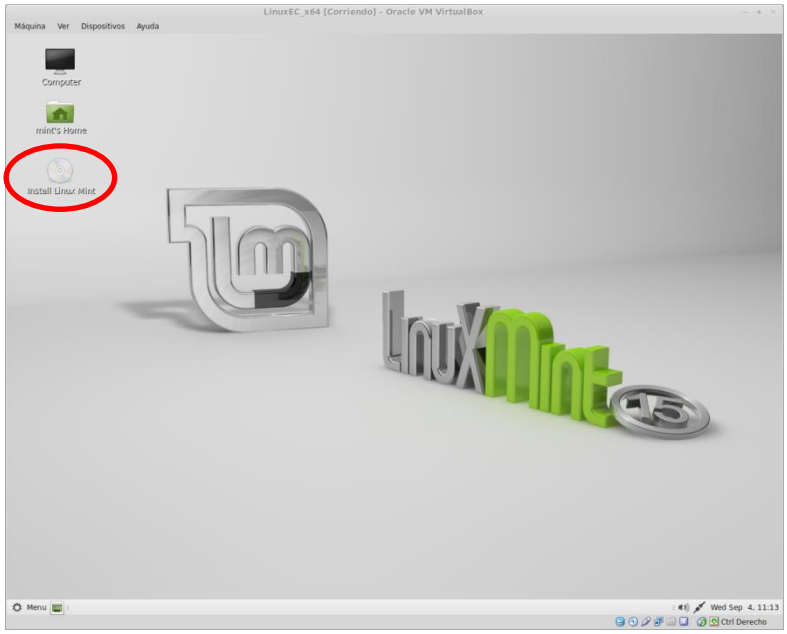
A continuación os aparecerá un asistente de instalación en el cual podéis especificar el archivo con la imagen para instalar el SO de la máquina virtual.
Pulsad el icono con forma de carpeta que aparece a la derecha, navegad por el sistema de archivos y escoged el archivo *linuxmint-15-mate-dvd-64bit.iso*

Pulsad *Iniciar* y se iniciará la instalación.
Si aparece un mensaje indicando que se soporta la integración del puntero o que la ventana está optimizada para trabajar con 32 bits de color, aceptad los mensajes, opcionalmente podéis marcar que no os vuelva a avisar.

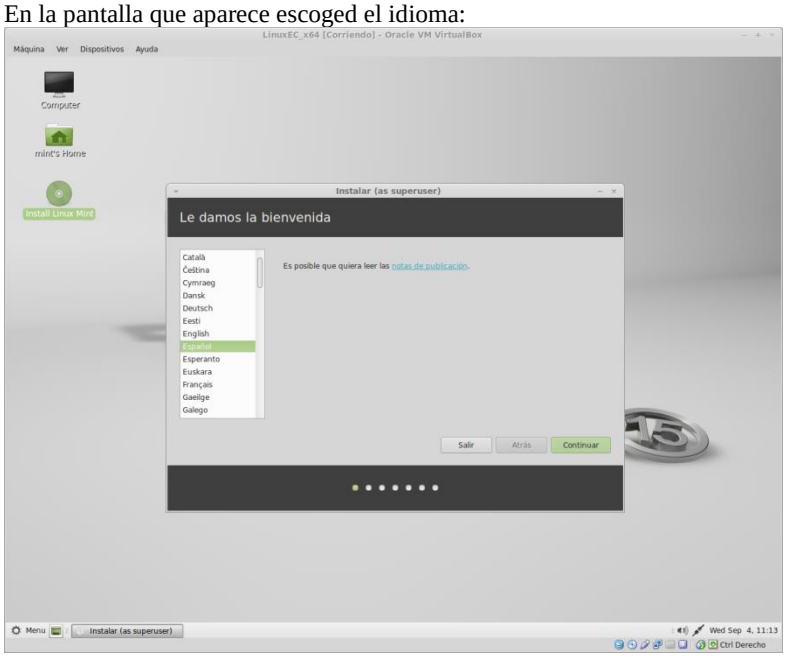
Aparece una pantalla con una cuenta atrás, esperad a que finalice y arranque el sistema automáticamente o pulsad Enter y en el menú que aparece escoged Start Linux Mint.



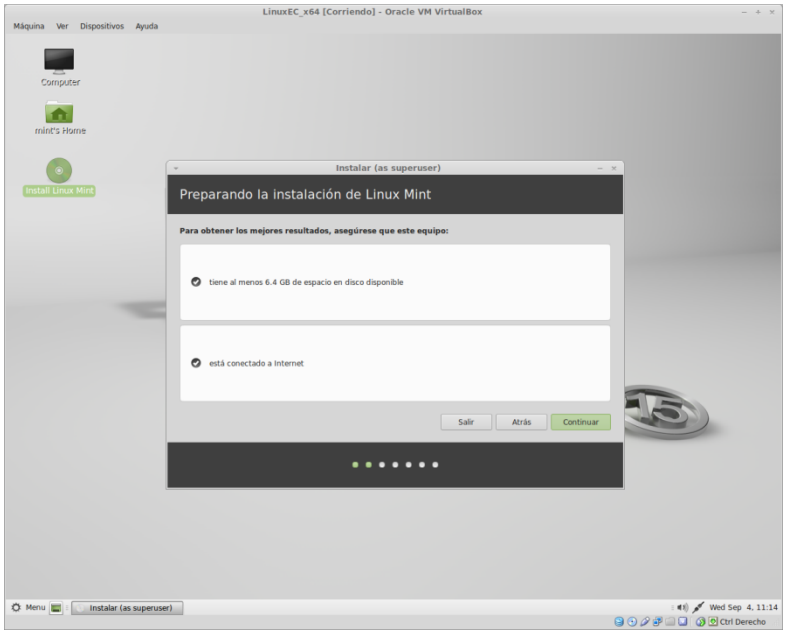
Una vez arrancado el sistema iniciad el programa de instalación, haciendo doble clic con el ratón sobre el icono del escritorio Install Linux Mint.



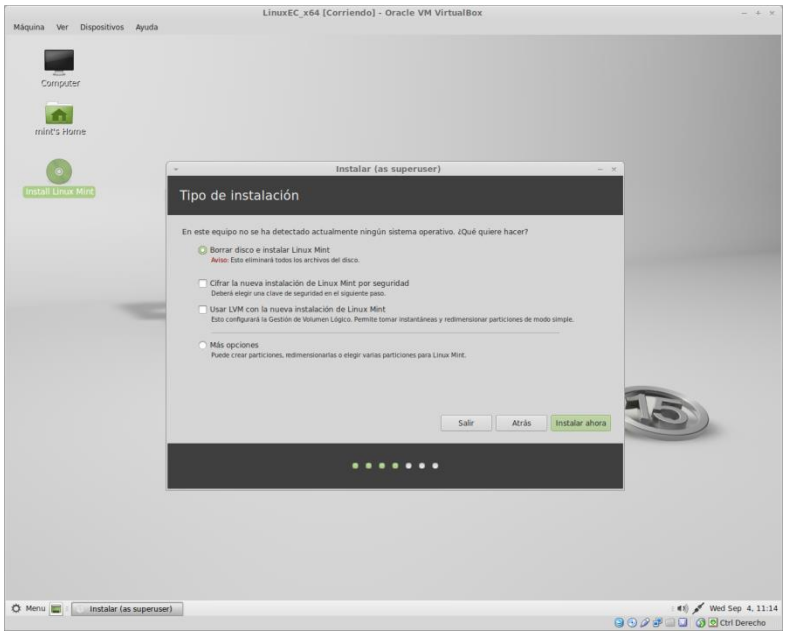
En la pantalla que aparece escoged el idioma:



Continuad con la instalación.

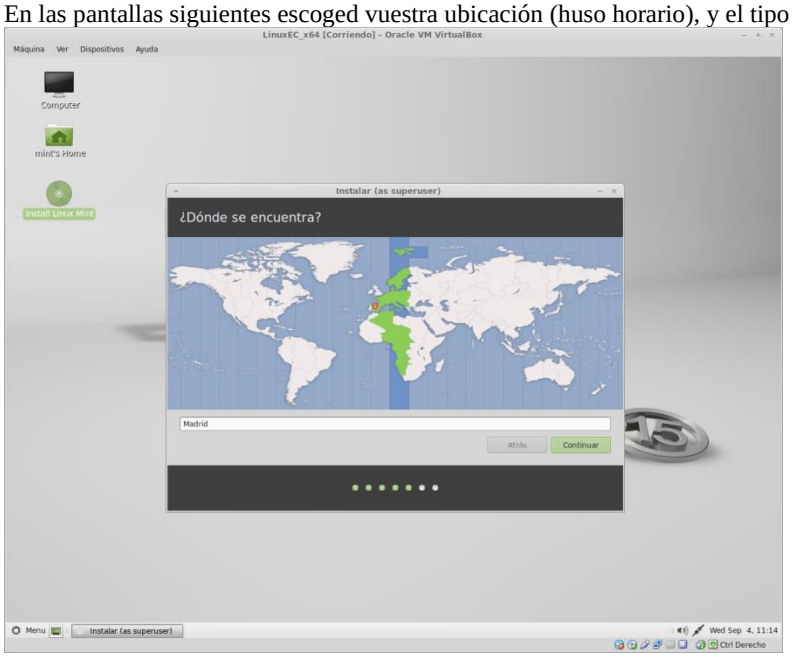


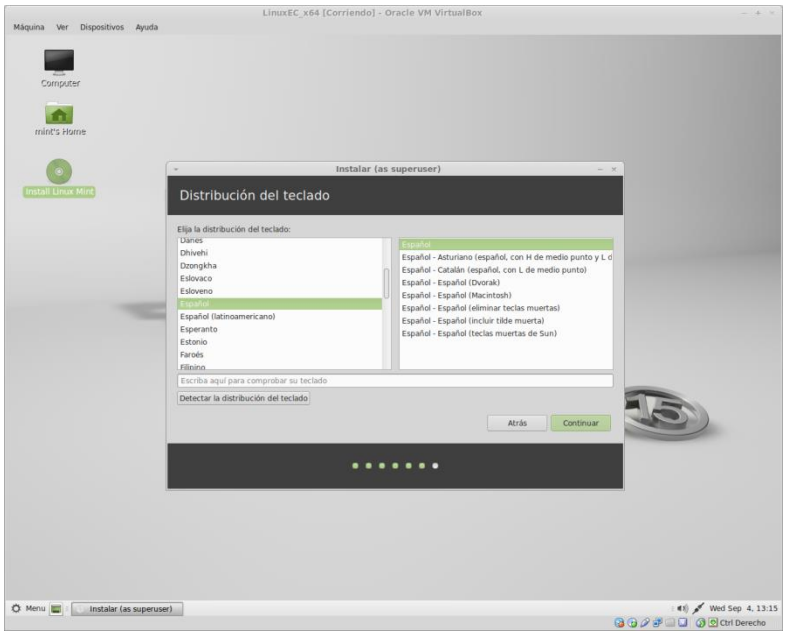
En la pantalla siguiente escoged la opción *Borrar el disco e Instalar Linux Mint*.



Pulsad el botón *Instalar ahora*.

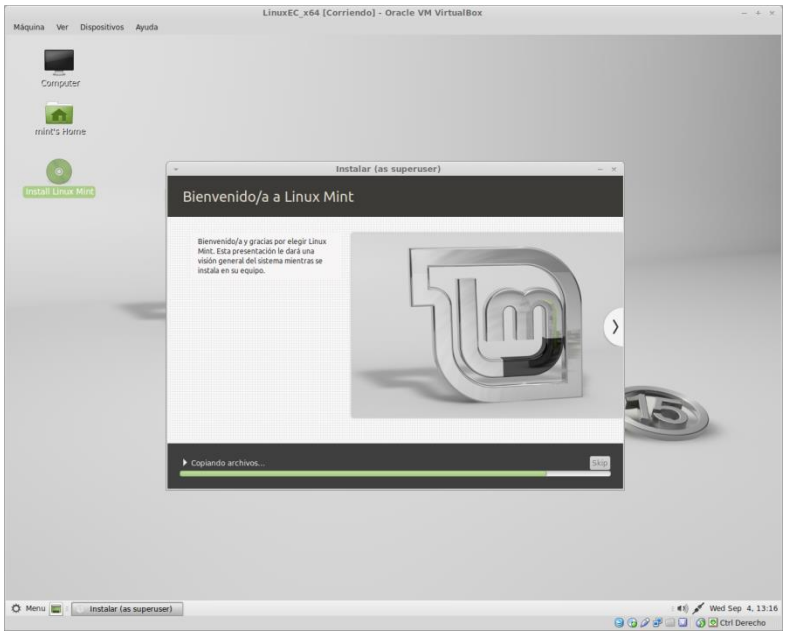
En las pantallas siguientes escoged vuestra ubicación (huso horario), y el tipo de teclado.



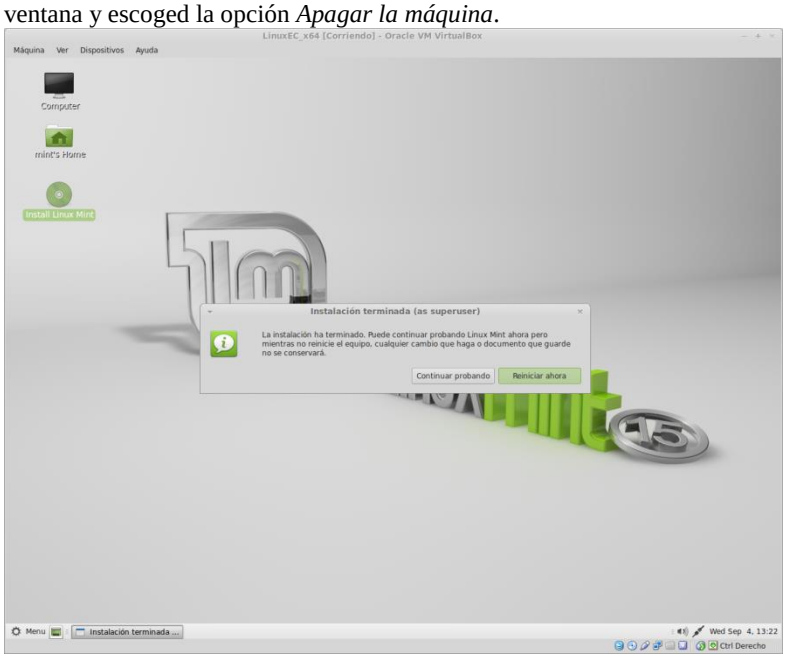


A continuación introducid vuestro nombre, un nombre para el ordenador, escoged un nombre de usuario y una contraseña. Podéis escoger la opción *Iniciar sesión automáticamente*, de esta forma no será necesario escribir el nombre de usuario y la contraseña cada vez que iniciéis la máquina virtual.

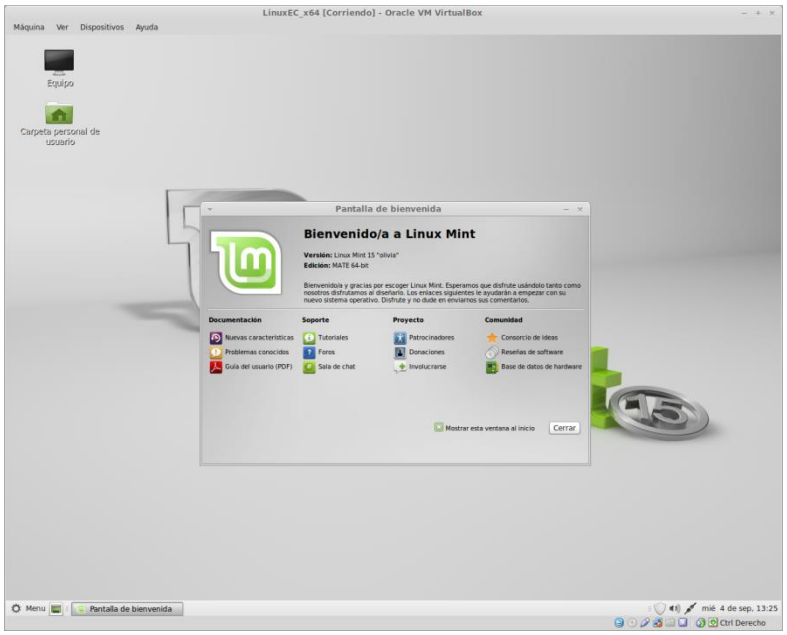
Pulsad *Continuar* y empezará la instalación, esperad hasta que finalice.



Finalmente os pedirá si queréis reiniciar el sistema, escoged *Reiniciar ahora.*
Si el sistema no se reinicia automáticamente, y se queda parado con la pantalla con un fondo negro pulsad ENTER sobre la pantalla de VirtualBox, o cerrad directamente la ventana y escoged la opción *Apagar la máquina*.



Después de reiniciar el sistema, debería de entrar automáticamente con el usuario creado y mostrar la pantalla inicial.

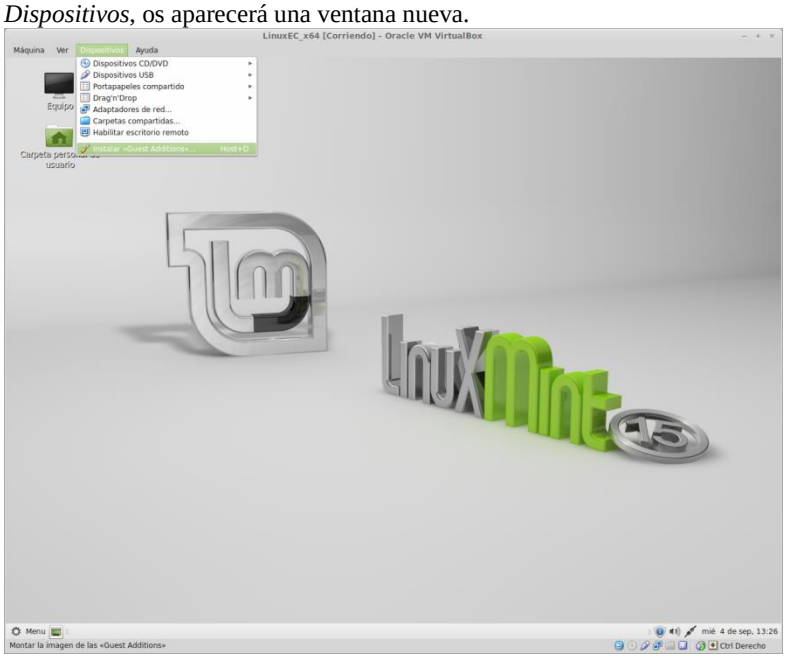


1.5. Instalar las Guest Additions de VirtualBox

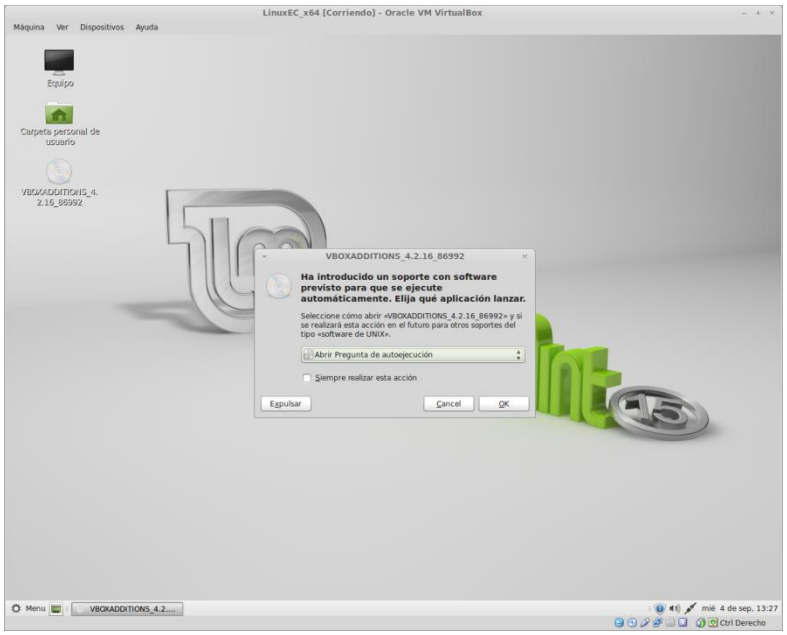
Es necesario instalar las Guest Additions para, entre otras cosas, permitir redimensionar la ventana de la máquina virtual o compartir carpetas entre la máquina anfitrión y la máquina virtual.

Las distribuciones de Linux Mint ya proporcionen una versión de las Guest Additions y en principio no será necesario realizar este paso.

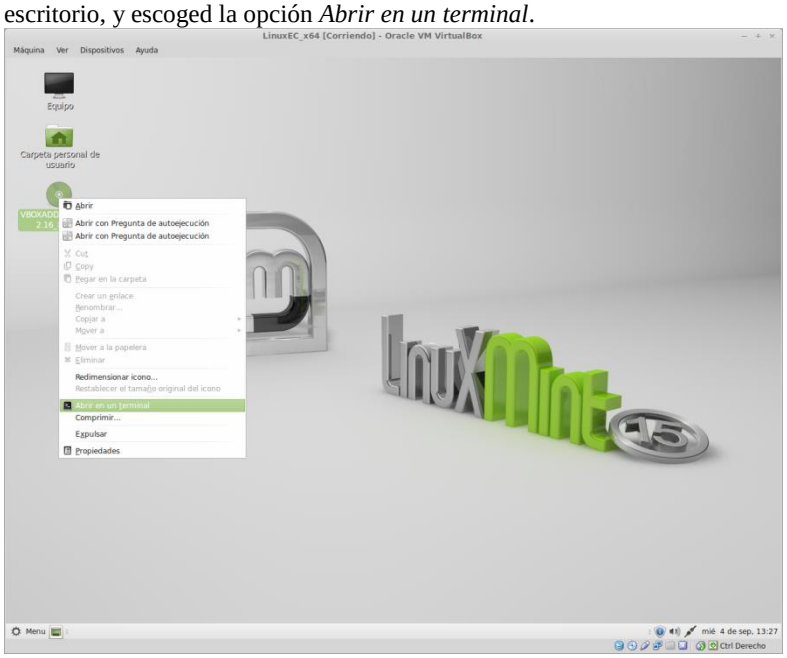
Para hacer esta instalación seleccionad la opción *Instalar Guest Additions* del menú *Dispositivos*, os aparecerá una ventana nueva.



Os aparecerá una ventana de ejecución automática. Pulsad el botón *Cancel*. Haremos la instalación desde una ventana de terminal.



Haced clic con el botón derecho del ratón sobre el icono del CD VBOXADDITIONS del escritorio, y escoged la opción *Abrir en un terminal*.



En la ventana de terminal que se abre, ejecutad el comando:

```
$ sudo ./VBoxLinuxAdditions.run
```

Os pedirá la contraseña del usuario para realizar tareas administrativas, introducidla y pulsad Enter.

Aparecerá un mensaje pidiendo confirmación, escribid ‘yes’ y pulsad Enter.

Cuando finalice la instalación podéis cerrar la ventana.

Podéis eliminar el disco virtual de las Guest Additions haciendo clic con el botón derecho del ratón sobre el icono del CD del escritorio y escogiendo la opción *Expulsar*.

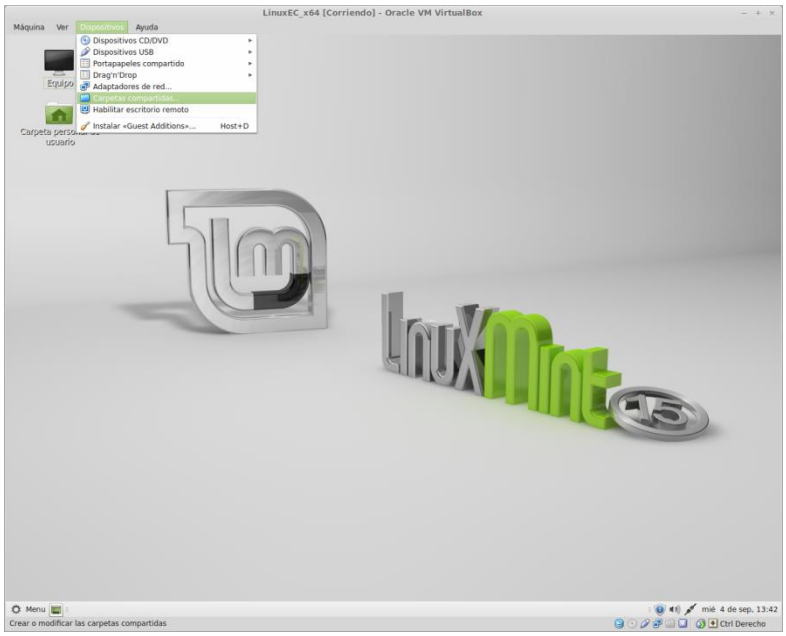
Finalmente reiniciad el sistema para activar los cambios, a través del menú de la parte inferior izquierda de Linux Mint con la opción Salir → Reiniciar.

1.6. Carpetas compartidas

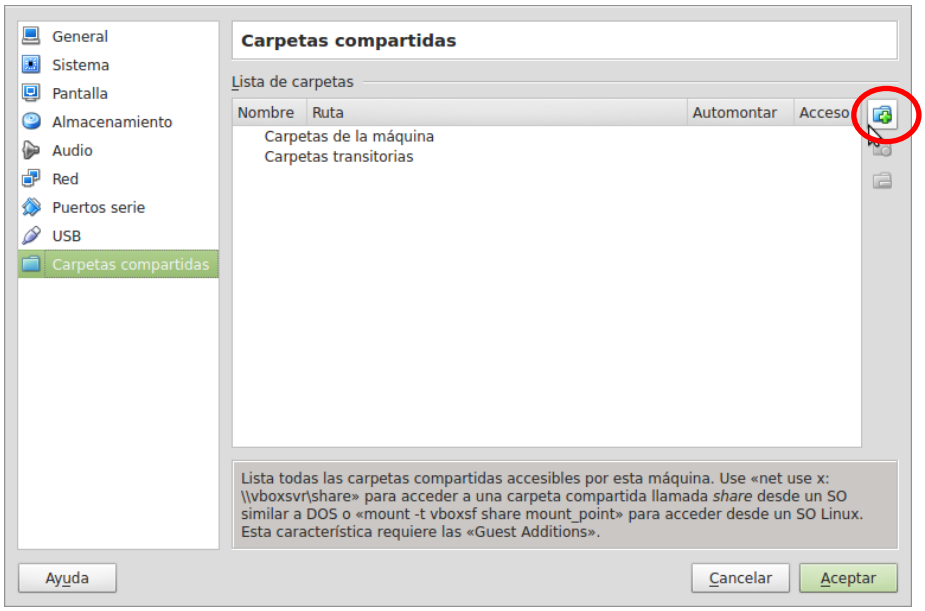
Podéis compartir una carpeta entre la máquina virtual y la máquina anfitriona, de forma que podréis acceder a los archivos de esta carpeta des de dentro y desde fuera de la máquina virtual.

Para crear una carpeta compartida seguid el proceso siguiente.

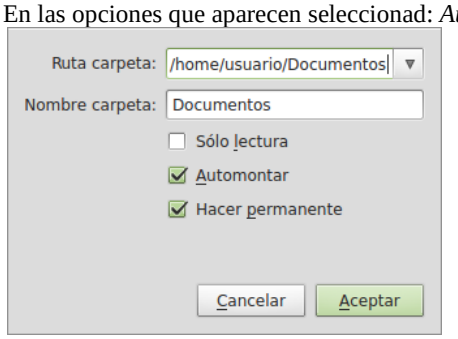
En la ventana de la máquina virtual abrid el menú *Dispositivos* de VirtualBox, escoged la opción *Carpetas compartidas*.



En la ventana que aparece pulsad el icono de la carpeta con un signo más de la parte derecha para crear una nueva carpeta compartida.



En la nueva ventana que aparece escoged la opción *Otro* en el desplegable *Ruta carpeta*, navegad por el sistema de archivos hasta la carpeta que queráis compartir.
En las opciones que aparecen seleccionad: *Automontar* y *Hacer permanente*.



Aceptad los cambios.

Es necesario añadir el usuario de la máquina virtual al grupo de Linux vboxsf, para hacerlo seguid los pasos siguientes:

1. Abrid un terminal (Menú → *Terminal*)
2. Ejecutad el comando siguiente:

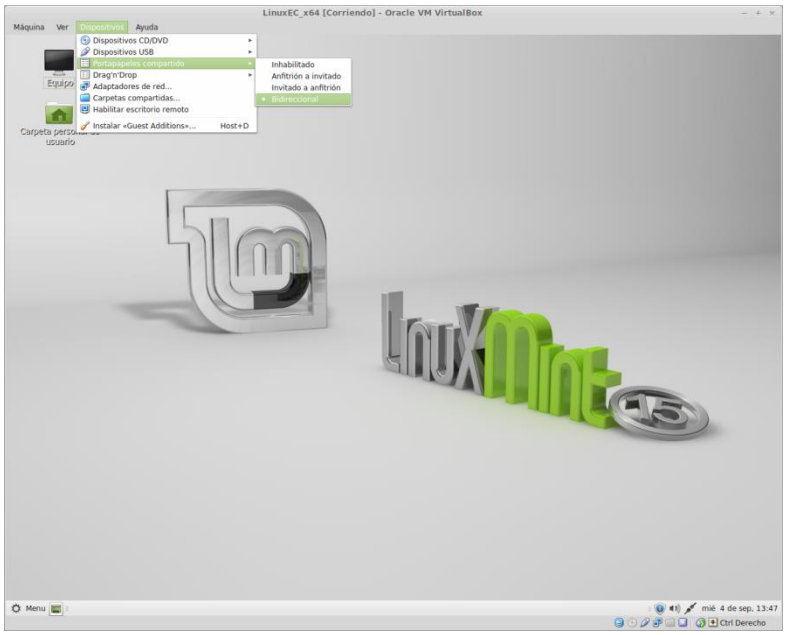
```
$ sudo usermod -a -G vboxsf usuario
```

Os pedirá que introduzcáis la contraseña del usuario.

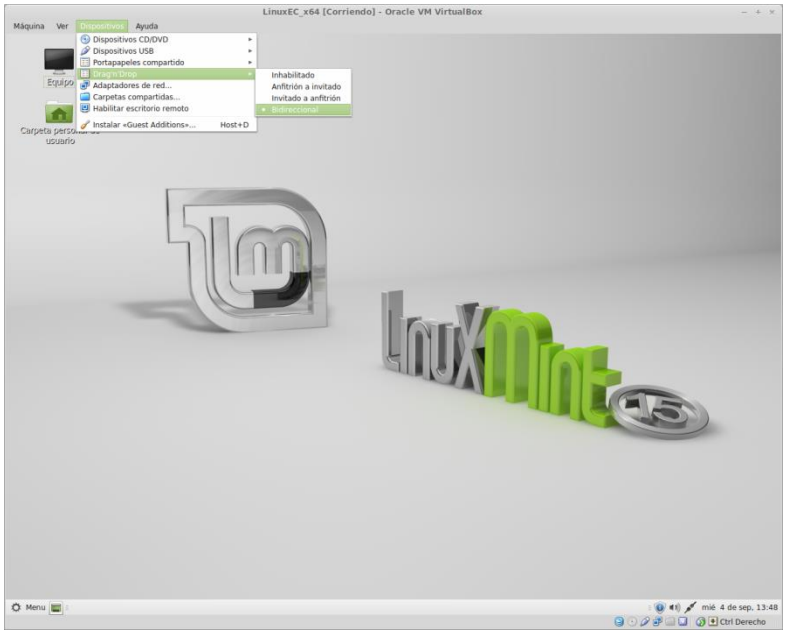
3. Reiniciad la máquina virtual. La nueva carpeta compartida debería aparecer montada en el directorio */media/sf_CarpetaCompartida*, en el caso del ejemplo anterior sería: */media/sf_Documentos*

1.7. Porta-papeles compartido y arrastrar archivos

Podéis compartir el porta-papeles entre la máquina virtual (cliente) y la máquina anfitriona, de forma que podemos copiar y enganchar contenido entre las dos máquinas.
Para activar la compartición del porta-papeles, acceded al menú *Dispositivos* → *Porta-papeles compartido* de VirtualBox, y escoged una de les opciones disponibles.



También podemos arrastrar archivos entre la máquina cliente y la máquina anfitriona, para activar el arrastre de archivos acceded al menú *Dispositivos → Drag'n'Drop* de VirtualBox, y escoged una de las opciones disponibles.



1.8. Acceder a un dispositivo USB

Si habéis instalado el paquete de VirtualBox, Oracle Extension Pack, podréis acceder a los dispositivos USB desde dentro de la máquina virtual.

Para hacerlo acceded al menú de VirtualBox *Dispositivos → Dispositivos USB*, y seleccionad el dispositivo deseado, se montará automáticamente y aparecerá en el escritorio de Linux.

1.9. Instalación de las herramientas

A continuación explicaremos una forma de instalar las herramientas en un entorno Linux Mint (també es válido para Ubuntu).

Para facilitar la tarea os proporcionamos un script que realiza la tarea de instalación por vosotros y es la forma que os recomendamos que uséis

Descargad el script dentro de la máquina virtual y a continuación ejecutadlo desde un terminal (Menú → *Terminal*).

Para ello primero cambiamos al directorio donde se encuentra el script, y a continuación lo ejecutamos. Suponiendo que tenemos el script en el escritorio del usuario ejecutamos los comandos siguientes:

```
$ cd /home/usuario/Escritorio
$ sh install_EC.sh
```

Os pedirá la contraseña de vuestro usuario para realizar tareas administrativas.

Os mostrará en el terminal los comandos que realiza esperad hasta que finalice y muestre nuevamente el símbolo del sistema. Todo el proceso puede tardar algunos minutos ya que se tienen que descargar algunos paquetes a través de Internet.

Script install_EC.sh

Encontraréis el script junto con este documento en el tablón de la asignatura.

Podéis descargarlo dentro de la máquina virtual accediendo con el navegador Firefox al campus virtual.

Si lo habéis descargado en vuestra máquina anfitriona lo podéis copiar dentro de VirtualBox utilizando Drag'n'Drop, arrastrando el archivo hacia la máquina virtual.

1.9.1. Instalación manual de las herramientas

No hay que realizar estos pasos si habéis usado el script, install_EC.sh.

Para instalar cualquier software es necesario tener privilegios de superusuario, por este motivo es necesario utilizar el comando *sudo* e introducir la contraseña del usuario cuando sea necesario.

Instalación y actualización del software manualmente

En primer lugar actualizaremos el software instalado en el sistema.

Abrid un terminal (Menú → *Terminal*) y ejecutad los comandos siguientes:

```
$ sudo apt-get -y update

$ sudo apt-get -y upgrade
```

Ejecutad cada comando en una sola línea.

A continuación instalaremos las herramientas necesarias, ejecutad los comandos siguientes:

```
$ sudo apt-get -y install geany yasm xterm kdbg

$ wget http://security.ubuntu.com/ubuntu/pool/universe/k/kdbg/
  kdbg 2.5.3-1 amd64.deb

$ sudo dpkg -i kdbg_2.5.3-1_amd64.deb

$ rm kdbg_2.5.3-1_amd64.deb
```

> Los paquetes gcc y gdb ya están instalados en Linux Mint, si utilizáis otra distribución puede ser necesario añadir los dos paquetes al primer comando.

Ejecutad cada comando en una sola línea.

Una vez instalados todos los paquetes podremos trabajar con ellos desde un terminal.

También podréis acceder a las herramientas que disponen de interfaz gráfica desde el menú de aplicaciones.

El editor debería aparecer en el menú

Menú → Aplicaciones → Programación → Geany

El entorno de depuración debería aparecer en el menú

Menú → Aplicaciones → Programación → KDbg

De todas formas es más práctico ejecutar el *kdbg* desde un terminal, ya que nos permite indicar el nombre del ejecutable que queremos depurar y localiza de forma automática el archivo con el código fuente.

Configurar kdbg manualmente

Es necesario configurar el entorno de depuración gdb/kdbg para que se muestre el código ensamblador generado para los programas escritos en C en notación Intel.

Para hacerlo es necesario realizar los pasos siguientes:

1. Abrid un terminal (*Menú → Terminal*) y ejecutad el comando siguiente para iniciar el editor de texto *geany* o iniciadlo desde el menú de aplicaciones (*Menú → Aplicaciones → Programación → Geany)*:

```
$ geany /home/usuario/.gdbinit
```

Sustituid “usuario” por vuestro nombre de usuario, fijaos que el nombre del archivo `.gdbinit` empieza con un punto.

2. Escribid una línia de texto con el contenido siguiente:

```
set disassembly-flavor intel
```

3. Grabad el archivo y cerrad el editor kate.
4. Iniciad kdbg, lo podéis hacer desde el terminal:

```
$ kdbg
```

Acceded al menú Preferencias → Opciones globales

En la línea: Como iniciar GDB, modificad el valor que aparece y poned:

`gdb --fullname` (se trata de eliminar el parámetro --nx)

5. La próxima vez que iniciéis kdbg el código ensamblador se mostrará en notación Intel.

Crear archivo de resaltado ensamblador para geany manualmente

Abrid un terminal (*Menú → Terminal*) y ejecutad el comando siguiente:

```
$ mkdir $HOME/.config/geany/filedefs –p
```

Este comando creará un directorio *filedefs* dentro del directorio de usuario.

Encontraréis un archivo con el nombre *filetypes.asm* en el tablón de la asignatura, junto a este mismo documento, descargadlo dentro de la máquina virtual para poder copiarlo.

Ejecutad el comando siguiente:

```
$ cp $HOME/Escritorio $HOME/.config/geany/filedefs/filetypes.asm
```

El comando copia el archivo *filetypes.asm* del *Escritorio* del usuario al directorio *filedefs* dentro del directorio del usuario, sustituid *Escritorio* por el directorio donde hayáis descargado el archivo *filetypes.asm*

2. Utilización de las herramientas

2.1. Editor de texto: geany

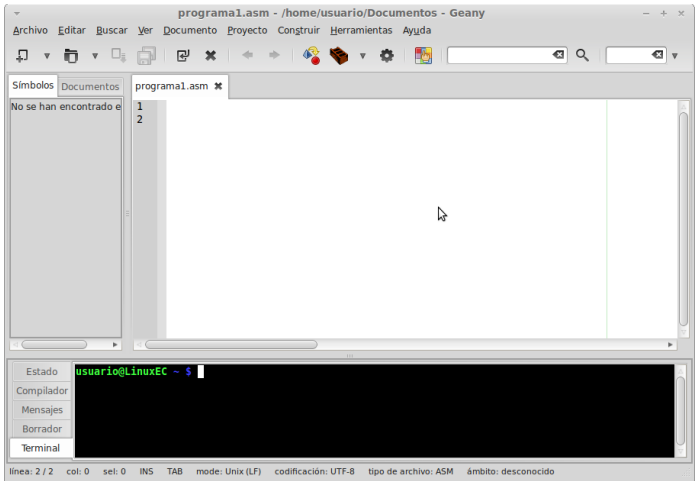
Para escribir código en lenguaje C o Ensamblador se puede utilizar cualquier editor de texto. Algunos editores disponen de resaltado de la sintaxis, característica que muestra los elementos del lenguaje utilizando diferentes colores para cada uno de ellos y otras funcionalidades que ayudan a programar. Existen diferentes editores con características parecidas. En este documento se describe la utilización del editor ***geany***, pero se pueden utilizar otros.

Para abrir el editor se puede acceder a través del menú, *Aplicaciones → Programación* o en modo línea desde un terminal ejecutando el comando *geany*, añadiendo opcionalmente como parámetro el nombre del archivo con el código fuente (archivo *.asm* o *.c*) que se quiere editar, si no existe lo crea.

```
usuario@Ubuntu:$ geany programa1.asm
```

Al ejecutar el editor se abre una ventana con la interfaz para trabajar con esta herramienta.

Geany integra el acceso a un terminal dentro de la misma aplicación; este terminal permite ejecutar comandos desde el editor mismo. Para abrir el terminal sólo se necesario pulsar el botón *Terminal* que se encuentra en la parte inferior izquierda de la ventana del editor.



Terminal Linux

En Linux tenemos que ejecutar muchos de los programas y herramientas des de un terminal, una línea de comandos.
En Linux Mint podéis acceder al terminal (o consola) a través del acceso directo del menú debajo del grupo Sistema: Terminal.
Y también desde el menú Aplicaciones → Herramientas del sistema → Terminal.
Si hacéis clic con el botón derecho sobre Terminal, os aparece un menú de opciones que os permite añadir un acceso directo al escritorio o al panel inferior.

2.2. Ensamblador: yasm

Un ensamblador es un programa que a partir de código fuente (código ensamblador) generar código objeto.

Hay diferentes ensambladores con sintaxis Intel para sistemas Linux. Como cada ensamblador utiliza una sintaxis diferente, se ha escogido uno en concreto para trabajar los ejercicios prácticos. En este documento se describe la utilización de Yasm.

Documentación *yasm*

Podéis encontrar información detallada de yasm en su web oficial: http://yasm.tortall.net/

Con el comando de línea de Linux *man yasm* podéis consultar rápidamente todos los parámetros y algunas cuestiones básicas sobre la sintaxis que se pueden utilizar con yasm..

Yasm es una nueva versión completa del ensamblador NASM (Netwide Assembler).

Actualmente soporta el juego de instrucciones de las arquitecturas x86 y x86-64, acepta las sintaxis NASM y GAS, genera código objeto en los formatos siguientes: ELF32, ELF64, Mach-O, RDOFF2, COFF, Win32, y Win64, y genera información de depuración en los formatos STABS, DWARF 2, y CodeView 8.

Algunas de las características de Yasm que hacen interesante su uso son:
- Multiplataforma: hay versiones de Yasm para Linux y Windows
- Sintaxis: utiliza la sintaxis de Intel, esta sintaxis es la más utilizada y podéis encontrar mucha documentación.

Es un programa que funciona en modo línea y que se tiene que ejecutar desde un terminal. Para ejecutarlo, se tiene que escribir el comando *yasm*, añadiendo los parámetros necesarios.

Los parámetros mínimos que necesitaremos para poder generar código objeto para un entorno Linux de 64 bits son los siguientes:

`nombre_archivo_fuente` Nombre del archivo con el código fuente; recomendamos utilizar la extensión `.asm` en estos archivos para facilitar la identificación.

`-f elf64` Parámetro para indicar el formato del código objeto que genera; en sistemas Linux de 64 bits el formato tiene que ser "elf64".

Opcionalmente se pueden especificar otros parámetros como por ejemplo:

`-g dwarf2` Parámetro para que genere información para la posterior depuración del programa; DWARF2 es un formato de información de depuración que está fuertemente asociado al formato de código objeto ELF.

`-o nombre_archivo_objeto` Parámetro para especificar el nombre del archivo de código objeto que se genera. Si no se pone este parámetro el archivo de código objeto generado se denomina igual que el archivo con el código fuente cambiando la extensión `.asm` por la extensión `.o` o añadiéndola si no tiene. Es recomienda utilizar esta extensión para facilitar la identificación.

```
$ yasm -f elf64 -g dwarf2 programa1.asm
```

Si en el proceso de ensamblaje no se producen errores, Yasm genera un archivo con el código objeto, si se producen errores, estos se muestran en pantalla con el número de

línea donde se ha detectado y una breve descripción, y no se genera el archivo con el código objeto. En el proceso de desarrollo se explica con un ejemplo como interpretar los errores.

2.3. Enlazador: gcc(ld)

El enlazador es un programa que a partir del código objeto generado por el ensamblador genera código ejecutable.

El enlazador del sistema es *ld*, pero nosotros utilizaremos el programa *gcc* que internamente es capaz de llamar a *ld*, no haremos llamadas explícitas a *ld; gcc* es un programa que funciona en modo línea y se tiene que ejecutar desde un terminal; para ejecutarlo hay que escribir el comando *gcc*, y añadir los parámetros necesarios.

Los parámetros mínimos que necesitaremos para poder generar código ejecutable son los siguientes:

`nombre_archivo_objeto` Nombre del archivo con el código objeto generado con el ensamblador; si no se ha especificado de manera diferente es el archivo con la extensión `.o`

Opcionalmente se pueden especificar otros parámetros como por ejemplo:

`-o nombre_archivo_ejecutable` Parámetro para especificar el nombre del archivo de código ejecutable que se genera. Si no se pone este parámetro, el archivo de código ejecutable generado se denomina `a.out`

```
$ gcc –o programa1 programa1.o
```

Si en el proceso de enlazado no se producen errores, se genera un archivo con el código ejecutable, si se producen errores estos se muestran en pantalla con una breve descripción, y no se genera el archivo con el código ejecutable, a pesar de que no es nada habitual que haya errores en este proceso.

2.4. Compilador de C: gcc

El compilador de C (*gcc*), además de compilar código C para generar el código objeto, llama al programa enlazador del sistema (*ld*) para generar el archivo ejecutable final a partir de los diferentes códigos objeto generados con el ensamblador y el código objeto obtenido a partir del código C.

gcc es un programa que funciona en modo línea y se tiene que ejecutar desde un terminal; para ejecutarlo se tiene que escribir el comando *gcc*, y añadir los parámetros necesarios.

Los parámetros mínimos que necesitaremos para la ejecución correcta del compilador de C son los siguientes:

`nombre_archivos_c` Lista de archivos de código C; los nombres de archivo con el código fuente C, tienen que tener la extensión `.c`

Opcionalmente se pueden especificar otros parámetros como por ejemplo:

`archivos_o`	Lista de archivos de código objeto; si no se ha especificado de manera diferente serán los archivos con la extensión `.o`
`-o archivo_salida`	Parámetro para especificar el nombre del archivo de código ejecutable que se genera.
`-g`	Parámetro para que genere información para la depuración posterior del programa.

```
$ gcc –g –o programa2 programa1.o programa2.c
```

Si en el proceso de compilación no se producen errores, se genera un archivo con el código ejecutable; si se producen errores, se muestran en pantalla con el número de línea donde se ha detectado y una breve descripción, y no se genera el archivo con el código ejecutable. En el proceso de desarrollo se explica con un ejemplo como interpretar los errores.

Si no se especifica ningún archivo de código fuente C, sólo se especifican archivos con código objeto, se genera el ejecutable a partir de estos archivos. De este modo podemos generar código ejecutable a partir de archivos objeto obtenidos con el ensamblador, proceso equivalente a la ejecución del enlazador (*ld*) explicado anteriormente.

```
$ gcc –g –o programa1 programa1.o
```

2.5. Depurador : kdbg (gdb)

Un depurador es el programa necesario para seguir la ejecución de un programa a paso y ver cómo evolucionan los valores de registros y variables; es una herramienta básica para poder detectar y corregir errores en nuestros programas. Tener un buen dominio de esta herramienta puede reducir drásticamente el tiempo de desarrollo de un programa.

El depurador *gdb* es un programa que funciona en modo línea y se tiene que ejecutar desde un terminal. Se inicia mostrando un indicador de órdenes (*prompt*) para introducir las órdenes que nos permiten hacer el seguimiento de la ejecución del código a paso, pero hacer la depuración de este modo resulta bastante complejo.

Para facilitar este proceso de depuración de los programas escritos en C y ensamblador utilizaremos una interfaz gráfica para el *gdb*: *KDbg*.

Para ejecutar esta interfaz se puede hacer desde el menú del sistema *Aplicaciones* → *Programación* → *KDbg*, o en modo línea desde un terminal ejecutando la orden *kdbg*, añadiendo como parámetro el nombre del archivo ejecutable que se quiere depurar.

```
$ kdbg programa1
```

Al ejecutar el depurador, se abrirá una ventana con la interfaz para trabajar con esta herramienta.

Si no se indica el nombre del programa que se quiere depurar habrá que abrir el archivo ejecutable desde la interfaz del programa.

Para abrir el archivo ejecutable se tiene que utilizar el botón *Ejecutable* de la barra de herramientas o la opción de menú *Archivo → Ejecutable*, navegar por los archivos hasta encontrar el correcto, y seleccionar el archivo ejecutable (*programa1*) y nunca el archivo con el código fuente (*programa1.asm*).

Hay que tener presente que el depurador trabaja sobre el código ejecutable del programa y, si detectamos un error en el código mientras hacemos el seguimiento paso a paso, no lo podremos modificar desde este entorno; tendremos que volver al editor del código fuente, hacer las modificaciones necesarias, guardar los cambios, generar un archivo ejecutable nuevo, cargar nuevamente el archivo ejecutable en el depurador y volver a probar si el problema se ha solucionado.

En el proceso de desarrollo se explica con un ejemplo como utilizar esta herramienta.

2.6. Ejecución

El archivo con el código ejecutable generado por el *gcc* (o por el *ld*) se puede ejecutar desde un terminal, simplemente especificando el nombre del archivo, añadiendo delante "./" para indicar que se encuentra en el directorio actual.

```
$ ./programa1
```

Caminos en Linux

En Linux los caminos (path) por defecto en que el sistema operativo busca un programa para ejecutarlo, no incluyen nunca el directorio actual; por este motivo para ejecutar un programa hay que poner ./ delante del nombre del programa.

3. Proceso de desarrollo en ensamblador

Para comprobar que el entorno de programación funciona correctamente se seguirán las diferentes fases del proceso de desarrollo a partir de un código fuente dado en ensamblador. Ahora no os preocupáis de analizar el código, más adelante ya trabajaremos las directivas, las instrucciones y la declaración de datos en profundidad.

3.1. Edición del código fuente

Hay que acceder al editor *geany* a partir de la opción de menú correspondiente (*Aplicaciones → Accesorios → geany*).

También se puede ejecutar desde un terminal (*Aplicaciones → Accesorios → Terminal*)

```
$ geany hola.asm
```

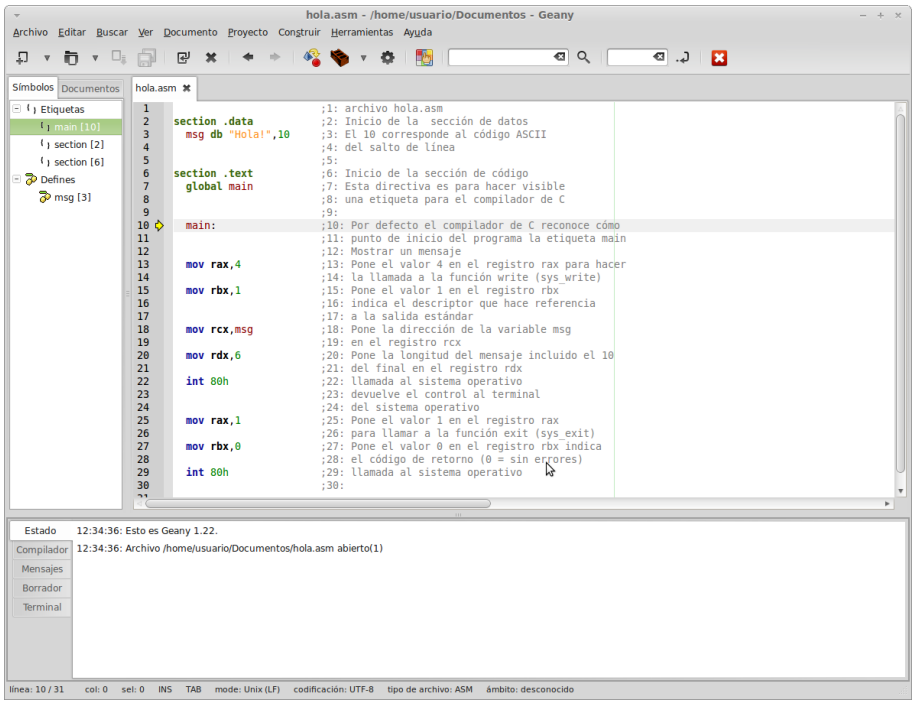
En la ventana del editor se tiene que escribir el código fuente, de 30 líneas, siguiente:

```
                    ;1: archivo hola.asm
section .data       ;2: Inicio de la  sección de datos
  msg db "Hola!",10 ;3: El 10 corresponde al código ASCII
                    ;4: del salto de línea
                    ;5:
section .text       ;6: Inicio de la sección de código
  global main       ;7: Esta directiva es para hacer visible
                    ;8: una etiqueta para el compilador de C
                    ;9:
  main:             ;10: Por defecto el compilador de C reconoce cómo
                    ;11: punto de inicio del programa la etiqueta main
                    ;12: Mostrar un mensaje
  mov rax,4         ;13: Pone el valor 4 en el registro rax para hacer
                    ;14: la llamada a la función write (sys write)
  mov rbx,1         ;15: Pone el valor 1 en el registro rbx
                    ;16: indica el descriptor que hace referencia
                    ;17: a la salida estándar
  mov rcx,msg       ;18: Pone la dirección de la variable msg
                    ;19: en el registro rcx
  mov rdx,6         ;20: Pone la longitud del mensaje incluido el 10
                    ;21: del final en el registro rdx
  int 80h           ;22: llamada al sistema operativo
                    ;23: devuelve el control al terminal
                    ;24: del sistema operativo
  mov rax,1         ;25: Pone el valor 1 en el registro rax
                    ;26: para llamar a la función exit (sys exit)
  mov rbx,0         ;27: Pone el valor 0 en el registro rbx indica
                    ;28: el código de retorno (0 = sin errores)
  int 80h           ;29: llamada al sistema operativo
                    ;30:
```

Código fuente ejemplos

Podéis encontrar todos los códigos fuente de los ejemplos de este documento en el archivo zip que contiene este mismo documento.

Una vez escrito el código, se recomienda almacenar el archivo indicando un nombre con la extensión .asm; *hola.asm* por ejemplo.



Geany reconoce algunos elementos del lenguaje ensamblador, instrucciones, registros, definición de variables y etiquetas entre otros.
Debajo de la pestaña *Símbolos* os mostrará las etiquetas y definiciones que ha encontrado en el código fuente, permitiendo acceder directamente a ellas.

3.2. Ensamblaje del código fuente

Para ensamblar el código hay que ejecutar desde el terminal (podéis utilizar el terminal del editor Geany) el comando siguiente:

```
$ yasm -f elf64 -g dwarf2 hola.asm
```

Si no se ha detectado ningún error el ensamblador no mostrará ningún mensaje, y se habrá generado un archivo con el nombre *hola.o en* el mismo directorio donde está el archivo con el código fuente (*.asm*).

Si se han detectado errores, se mostrarán en pantalla junto con el número de línea donde se encuentra el error; hay que modificar el código para corregir los errores indicados, guardar el código modificado y volver a ensamblar el código.

Haciendo los cambios siguientes en el código aparecerán 4 errores al ensamblarlo.

```
mov rcx,msg1 ;18: -> modificamos el nombre de la etiqueta
mov rdx,     ;20: -> eliminamos un operando
inta 80h     ;22: -> modificamos el código de operación
```

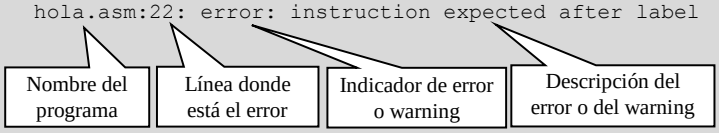
```
$ yasm -f elf64 -g dwarf2 hola.asm

hola.asm:18: error: undefined symbol `msg1' (first use)
hola.asm:18: error:  (Each undefined symbol is reported only once.)
hola.asm:20: error: expected operando, vaso end of line
```

Comandos desde geany

El editor Geany, permite definir accesos directos a los comandos de ensamblaje, enlazado, etc.
Podéis utilizar la opción de menú *Construir → Establecer comandos de construcción* para personalizar los accesos a las herramientas.

```
hola.asm:22: error: instruction expected after label
```



Con esta información tenemos que saber detectar cual es la causa del error o del *warning* y como corregirlo, a pesar de que la información que da el programa ensamblador *yasm* no sea muy exhaustiva.

3.3. Enlazado del código objeto y generación del ejecutable

La etapa siguiente consiste en generar el ejecutable utilizando el compilador *gcc*; para hacerlo hay que ejecutar desde un terminal (se puede utilizar el terminal integrado con el editor Geany) el comando siguiente:

```
$ gcc –o hola hola.o
```

Si el proceso se ejecuta correctamente se generará un archivo nuevo, en el mismo directorio, con el nombre indicado por el parámetro –o; si no se pone este parámetro, el archivo ejecutable generado se denomina `a.out`.

Esta etapa también se puede hacer utilizando el enlazador *ld*, pero recomendamos hacerlo con *el gcc* porque de este modo unificamos el proceso por cuando tengamos que trabajar con C o combinar el código ensamblador con código C.

3.4. Ejecución del programa

Para ejecutar el programa, hay que escribir desde el terminal (se puede utilizar el terminal integrado con el editor *Geany*) el nombre del ejecutable e indicar el directorio actual delante:

```
$ ./hola
```

El programa se tendría que ejecutar correctamente y mostrar el mensaje "`Hola!`"

```
$ ./hola
Hola!
$
```

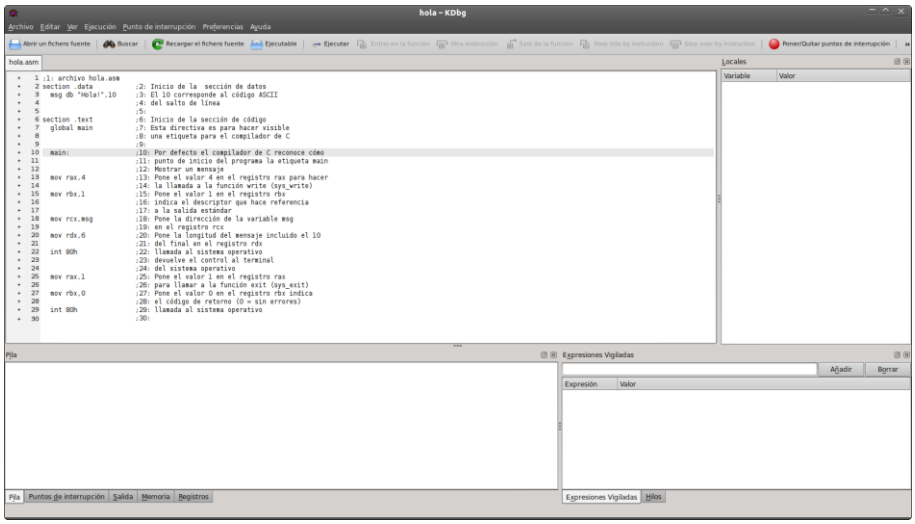
3.5. Depuración del programa con KDbg (gdb)

A continuación se explica la utilización de la entorno a depuración KDbg.

Para ejecutarlo hay que ejecutar desde el terminal (se puede utilizar el terminal integrado con el editor Geany) el comando siguiente:

```
usuario@Ubuntu:$ kdbg hola
```

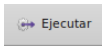
Si no se indica el nombre del programa que se quiere depurar, habrá que abrir el ejecutable desde la interfaz del programa KDbg. Se tiene que utilizar la opción de menú *Archivo → Ejecutable*, navegar por los archivos hasta encontrar el correcto, y seleccionar el archivo ejecutable (*hola*) y no el archivo con el código fuente (*hola.asm*).



Se describen a continuación las operaciones más frecuentes que se suelen realizar con el entorno KDbg.

1) Ejecución del programa

Para ejecutar el programa tenemos las opciones siguientes:

- Pulsar el botón 

- Escoger la opción de menú *Ejecución → Ejecutar*
- Pulsar la tecla F5.

El código se ejecuta de golpe hasta el primer punto de ruptura o interrupción (*breakpoint)*; si no hemos definido ningún punto de interrupción se ejecutará todo el código, la salida del programa por pantalla se hará en la ventana de ejecución del entorno (*Datos de salida*).

Si no se ha definido ningún punto de interrupción, se realiza una ejecución completa del programa, y la salida por pantalla será la misma que si hubiéramos ejecutado el programa directamente desde el terminal.

Para interrumpir la ejecución hay que escoger la opción de menú *Ejecución → Matar*; para reiniciar la ejecución desde el principio *Ejecución → Reiniciar*.

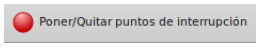
Ejecutad el programa y comprobad que en la ventana de ejecución se muestra el mensaje:

```
Hola!
```

2) Depurar el programa

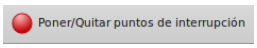
Para poder ejecutar el código paso a paso, primero hay que definir un punto de interrupción en la instrucción a partir de la cual queremos seguir la ejecución paso a paso. Si queremos depurar el código desde el principio, tendremos que poner un punto de ruptura en la primera instrucción. Cuando se inicia la ejecución del programa (*Ejecutar*), ésta queda detenida en el primer punto de interrupción que encuentra, y podemos seguir la ejecución en paso a partir de aquel punto.

Para definir un punto de interrupción disponemos de las opciones siguientes:

- Pulsar con el ratón al principio de la instrucción (delante del signo +) donde se quiere definir el punto de ruptura, la línea quedará marcada con un punto de color rojo.
- Situar el cursor en la línea donde se quiere introducir el punto de ruptura y pulsar el botón 

o pulsar la tecla F9

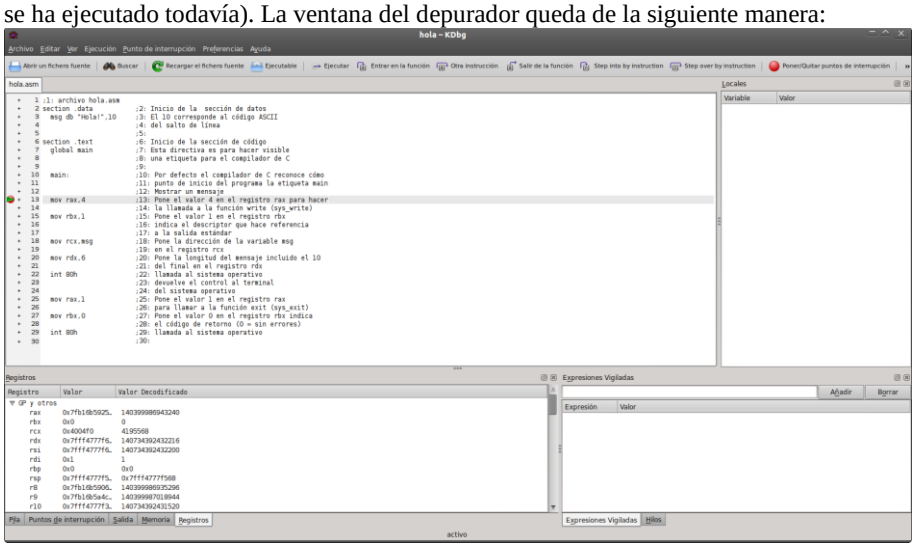
Para borrar un punto de ruptura definido tenemos las opciones siguientes:

- Volver a pulsar con el ratón a primeros de la instrucción donde se encuentra el punto de ruptura, desaparecerá el punto de color rojo.
- Situar el cursor en la línea donde es el punto de ruptura y pulsar el botón 

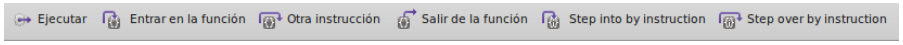
o pulsar la tecla F9

Situaos en la línea 13 (`mov rax,4)`, primera instrucción del programa, y definid un punto de interrupción.

Ejecutad el programa (*Ejecutar*). La ejecución se detendrá en la primera instrucción, instrucción donde hemos puesto el punto de interrupción. Aparece un triángulo verde en el principio de la línea que indica que es la siguiente instrucción que será ejecutada (no se ha ejecutado todavía). La ventana del depurador queda de la siguiente manera:



KDbg dispone de una barra de herramientas desde donde se pueden realizar las operaciones más habituales de depuración, también se puede acceder a estas operaciones a través de teclas de función y a través del menú *Ejecución*.



Ejecutar: ejecución hasta el punto de interrupción siguiente; F5 o Ejecución → *Ejecutar*
Entrar en la función: ejecución paso a paso, entrando dentro de las funciones o subrutinas; F8 o Ejecución → *Entrar en la función*
Otra instrucción: ejecución paso a paso, sin entrar dentro de las funciones o subrutinas; F10 o Ejecución → *Otra instrucción*
Salir de la función: salir de una función o subrutina; F6 o Ejecución → *Salir de la función*
Step into by instruction: ejecución instrucción a instrucción, entrando dentro de las funciones o subrutinas ensamblador; Mayúsculas+F8 o Ejecución → *Step into by instruction*.
Step over by instruction: ejecución instrucción a instrucción, sin entrar dentro de las funciones o subrutinas ensamblador; Mayúsculas+F10 o Ejecución → *Step over by instruction*.

Si estamos depurando código ensamblador, las operaciones de ejecutar paso a paso (Entrar en la función) y ejecutar instrucción a instrucción (Step into by instruction) harán lol mismo, pero si depuramos código C el comportamiento será diferente.

Ejecutad el programa instrucción por instrucción hasta que finalice.

3) Ejecución hasta el cursor

Podemos ejecutar un conjunto de instrucciones de golpe, desde la instrucción donde está detenida la ejecución (instrucción marcada con un triángulo verde) y hasta la instrucción marcada por la posición del cursor pulsando la tecla F7.

Esta opción es muy práctica porque permite ir fácilmente a un punto concreto del código para seguir desde allí la ejecución paso a paso.

4) Continuar la ejecución

Si el programa se encuentra parado en un punto de ruptura, podemos continuar la ejecución de este hasta el punto de ruptura siguiente volviendo a pulsar la opción *Ejecutar*.

5) Ver variables y registros

Podemos visualizar el valor de variables definidas en nuestro programa, otras expresiones o registros en la parte *Expresiones vigiladas* (en la parte inferior derecha de la pantalla principal). Para visualizar el contenido de una expresión hay que escribir el nombre en el cuadro de texto y pulsar el botón *Añadir*.
Podemos utilizar el botón *Borrar* para borrar una expresión.

Podemos escribir diferentes tipos de expresiones para su visualización:

- **nombre_variable**: se mostrará el contenido de la variable
- **&variable**: se mostrará la dirección de la variable y se podrá ver también su contenido

- **$registro**: se mostrará el contenido del registro, por ejemplo $rax, $rbp o $r9. También podemos ver una parte de un registro, por ejemplo: $al, $ah, $ax, $eax, $r9l, $r9w, $r9d.

  Para ver una parte de uno de los registros entre r8 y r15, hay que utilizar los sufijos: l (byte menos significativo), w (word: los dos bytes menos significativos), d (double word: los 4 bytes menos significativos). Para los registros rax, rbx, rcx y rdx podemos utilizar: l (byte menos significativo), h (segundo byte menos significativo); para los registros rsi, rdi, rbp podemos utilizar l para ver el byte menos significativo; para los registros rax, rbx, rcx, rdx, rsi, rdi, rbp, podemos ver los 2 bytes menos significativos con ax, bx, cx, dx, si, di, y bp, o podemos ver los 4 bytes menos significativos con eax, ebx, ecx, edx, esi, edi, y ebp.

  Hay que indicar los nombres de los registros en minúsculas y empezando siempre con $.

Se puede forzar la manera como se muestran los datos indicando un tipo de dato delante de la expresión, utilizando char (1 byte), short (2 bytes), int (4 bytes) o long (8 bytes)

- **(char) variable**: se mostrará el valor de la variable como un solo carácter, 1 byte
- **(char[8]) variable**: se mostrará el valor de 8 bytes a partir de la dirección de la variable como caracteres.
- **(int[2]) variable**: se mostrará el valor de 2 enteros de 32 bits.
- **(char[8]) $rax**: se mostrará el valor de los 8 bytes del registro RAX: char[0] corresponderá al registro AL, char[1] corresponderá al registro AH.
- **$ah: s**e mostrará el valor del segundo byte menos significativo del registro RAX
- **$si:** se mostrará el valor de los 2 bytes menos significativos del registro RSI.
- **$r8d: s**e mostrará el valor de los 4 bytes menos significativos del registro R8.

Se puede forzar que los valores se muestren en un determinado formato añadiendo un prefijo delante de la expresión: **/t** binario, **/d** decimal con signo, **/u** decimal sin signo, **/x** hexadecimal, **/c** carácter:

- **/c var**
- **/d var**
- **/u var**
- **/t $rax**
- **/x $rax**

Podemos combinar los tipos de datos con los prefijos de un cierto formato:

- **/u (char[8]) var:** se mostrará la variable como 8 bytes, cada byte se verá como un números decimal sin signo.
- **/u (char[8]) $rax:** igual que en el caso anterior, pero mostrando los 8 bytes del registro RAX.
- **/x (char[8]) $rax:** igual que en el caso anterior, pero mostrando los 8 bytes del registro RAX en hexadecimal.

Si tenemos una variable de tipo apuntador (una dirección de memoria) podemos ver su contenido con el operador *

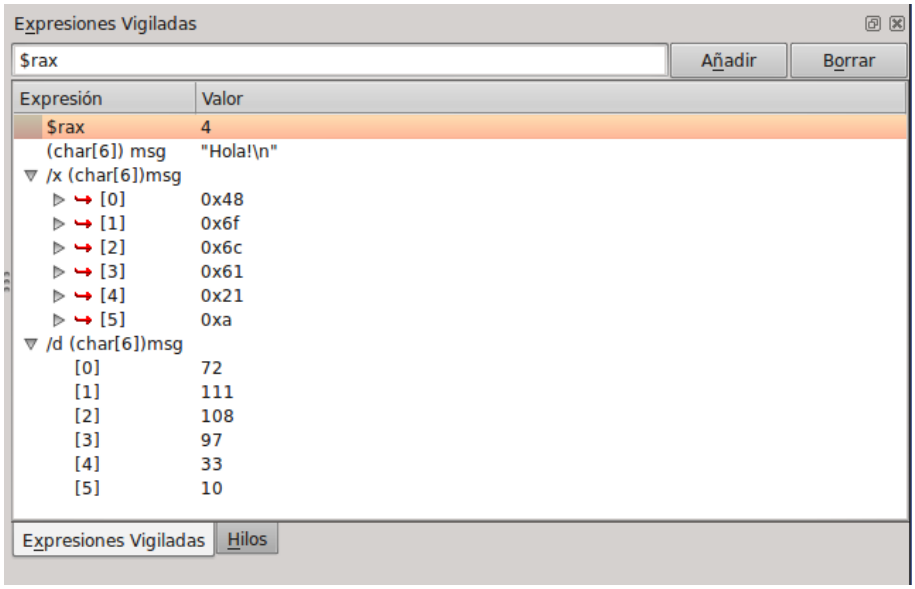
- **(* apuntador)**

Si tenemos definido un vector, podemos ver su contenido poniendo directamente el nombre del vector (ya que el nombre ya representa un apuntador al primer elemento):

- **/d vector:** se mostraran los elementos del vector como valores decimales.
- **/x vector:** se mostraran los elementos del vector como valores hexadecimales.

Podemos ver un elemento concreto de un vector indicando su posición entre [ ], podemos indicar la posición como un valor constante o utilizando un registro o variable:

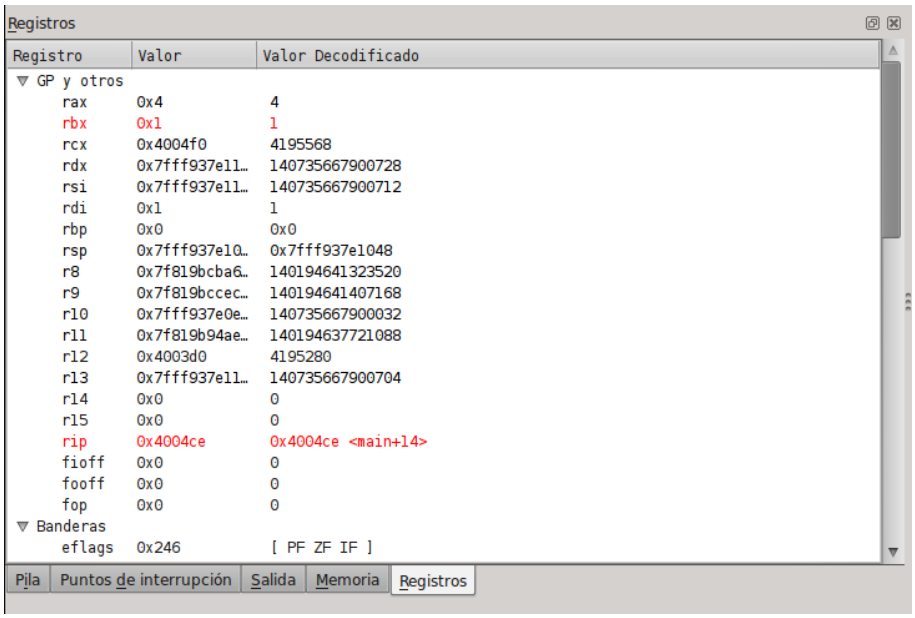
- **/d vector[0]:** se mostrará el primer elemento del vector como un valor decimal.
- **/x vector[$rax]:** se mostrará el elemento del vector indicado por el registro RAX como un valor hexadecimal.
- **/x vector[indice]:** se mostrará el elemento del vector indicado por la variable **indice** como un valor hexadecimal.



Cuando se depura un programa escrito en C, en la parte superior de la derecha de la pantalla se pueden ver automáticamente las variables locales definidas y el valor que tienen.

6) Ver banco de registros

Podemos ver los valores de los registros seleccionando la pestaña *Registros* en la parte inferior de la pantalla.

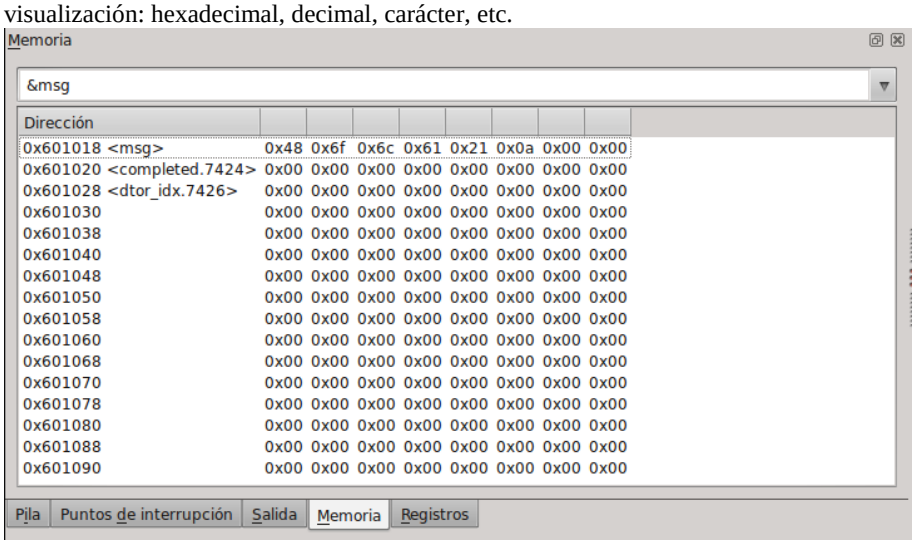


Se pueden ver los bits activos del registro EFLAGS pulsando con el botón derecho del ratón encima del nombre de los registros y seleccionando la opción GDB Default.

7) Ver contenido de memoria

Pulsando la pestaña *Memoria* de la parte inferior de la pantalla se puede ver el contenido de la memoria a partir de una dirección especificada, se puede utilizar el nombre de una variable o etiqueta poniendo delante el símbolo &, por ejemplo: *&msg*.

Si pulsamos con el botón derecho del ratón en esta ventana, nos aparece una lista de opciones de visualización del contenido de la memoria, se recomienda escoger la opción *Bytes*, para ver el contenido byte a byte. También podemos escoger el formato de visualización: hexadecimal, decimal, carácter, etc.



4. Proceso de desarrollo en C y ensamblador

Ahora veremos cómo utilizar este entorno de programación para desarrollar programas a partir de código fuente C y también como incorporar llamadas a código fuente ensamblador. Ahora no os preocupáis de analizar el código, más adelante ya trabajaremos las directivas, las instrucciones y la declaración de datos en profundidad.

En este proceso, primero crearemos el código fuente ensamblador y lo dejaremos a punto para poderlo llamar después desde el programa en C que haremos a continuación.

4.1. Edición del código fuente ensamblador

Hay que acceder al editor Geany a partir de la opción de menú correspondiente (*Aplicaciones → Utilidades → Geany*) o escribiendo el comando *geany* desde un terminal (*Aplicaciones → Accesorios → Terminal*)

```
$ geany hola_asm.asm
```

En la ventana del editor se tiene que escribir el código fuente, de 25 líneas, siguiente:

```
                    ;1: archivo hola_asm.asm
section .data       ;2: Inicio de la  sección de datos
  msg db "Hola!",10 ;3: El 10 corresponde al código ASCII
                    ;4: del salto de línea
                    ;5:
section .text       ;6: Inicio de la sección de código
  global printHola  ;7: Esta directiva es para hacer visible
                    ;8: una etiqueta para el compilador de C
                    ;9:
  printHola:        ;10: Nombre de la subrutina de ensamblador
                    ;11: que llamaremos desde el programa en C
                    ;12: Mostrar un mensaje
  mov rax,4         ;13: Pone el valor 4 en el registro rax para hacer
                    ;14: la llamada a la función write (sys write)
  mov rbx,1         ;15: Pone el valor 1 en el registro rbx
                    ;16: para indicar el descriptor que hace
                    ;17: referencia a la salida estándar
  mov rcx,msg       ;18: Pone la dirección de la variable msg
                    ;19: en el registro rcx
  mov rdx,6         ;20: Pone la longitud del mensaje incluido el 10
                    ;21: del final en el registro rdx
  int 80h           ;22: llamada al sistema operativo
                    ;23:
  ret               ;24: retorno de llamada a subrutina.
                    ;25:
```

Este código es muy parecido al código *hola.asm* utilizado en el punto anterior. Las modificaciones que se han hecho son cambiar la etiqueta *main* por la etiqueta *printHola*, añadir la instrucción *ret* para volver el control al programa que llame a esta subrutina, y eliminar las instrucciones para volver al sistema operativo.

Una vez escrito el código, guardadlo indicando un nombre con la extensión *.asm*, por ejemplo: *hola_asm.asm*

4.2. Ensamblaje del código fuente ensamblador

Para ensamblar el código hay que ejecutar desde el terminal (podéis utilizar el terminal del editor Geany) el comando siguiente:

```
$ yasm –f elf64 –g dwarf2 hola_asm.asm
```

Si no se ha detectado ningún error, el ensamblador no mostrará ningún mensaje, y se habrá generado un archivo con el nombre *hola_asm.o en* el mismo directorio donde se encuentra el código fuente (*.asm*).

Si se han detectado errores, se mostrarán en pantalla junto con el número de línea donde se encuentra el error; hay que modificar el código para corregir los errores indicados, guardar el código modificado y volver a ensamblar el código.

4.3. Edición del código fuente C

Hay que acceder al editor Geany a partir de la opción de menú correspondiente (*Aplicaciones → Utilidades → Geany*) o escribiendo el comando *geany* desde un terminal (*Aplicaciones → Accesorios → Terminal*)

```
$ geany hola c.c
```

En la ventana del editor se tiene que escribir el código fuente, siguiente:

```
#include <stdio.h>

extern void printHola();

int main() {
   printHola();
   printf("Hola!!!\n");
}
```

Una vezescrito el código, guardadlo indicando un nombre con la extensión *.c*, por ejemplo: *hola_c.c*

4.4. Compilación del código fuente C, ensamblaje con del código objeto ensamblador y generación del ejecutable

La siguiente etapa consiste en generar el ejecutable utilizando el compilador *gcc*; para hacerlo hay que ejecutar desde un terminal (se puede utilizar el terminal integrado con el editor Geany) el comando siguiente:

```
$ gcc –o hola c –g hola asm.o hola c.c
```

Si el proceso se ejecuta correctamente, se generará un nuevo archivo, en el mismo directorio con el nombre indicado por el parámetro –o; si no se pone este parámetro, el archivo ejecutable generado se denominará `a.out`.

4.5. Ejecución del programa

Para ejecutar el programa, hay que escribir desde el terminal (se puede utilizar el terminal integrado con el editor Geany) el nombre del ejecutable, recordando indicar el directorio actual delante:

```
$ ./hola_c
```

El programa se tendría que ejecutar correctamente y mostrar los dos mensajes "`Hola!`" y "`Hola!!!`"

```
$ ./hola c
Hola!
Hola!!!
$
```

4.6. Depuración del programa con KDbg

A continuación se utilizará el depurador; hay que ejecutar desde el terminal (se puede utilizar el terminal integrado con el editor Geany) el comando siguiente:

```
$ kdbg hola_c
```

Una vez abierto el depurador:

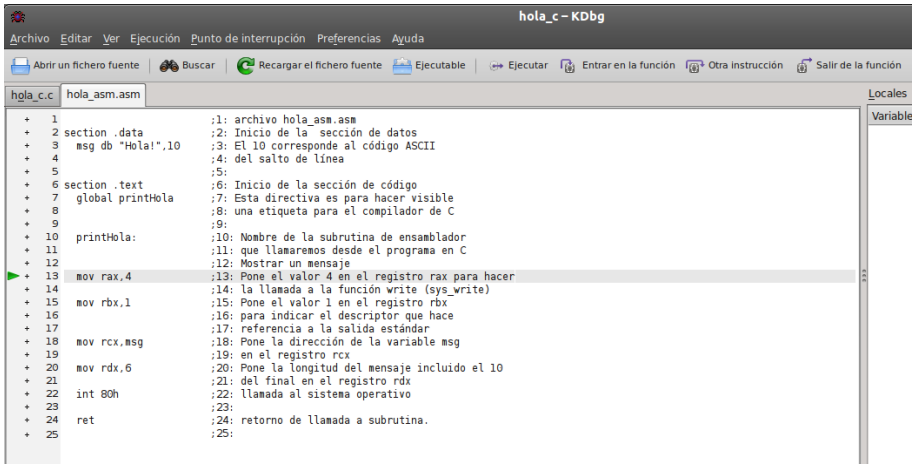
1) Definid un punto de interrupción en la línea con la instrucción `printHola()`

2) Situaos en la línea con la instrucción `printHola()` y haced clic con el ratón sobre el signo + del principio de la línea; os aparecerá el código ensamblador para aquella línea de código C.

3) Iniciad la ejecución del programa (*Ejecutar*) y ejecutad instrucción por instrucción, entrando dentro de las funciones:

Step into by instruction: Mayúsculas+F8 o Ejecución → *Step into by instruction*.

Cuando entréis dentro de la subrutina printHola veréis el código fuente del programa ensamblador: hola_asm.asm (en una nueva pestaña).



Podéis finalizar la ejecución instrucción a instrucción o volver al código C con el comando *Ejecución* → *Salir de la función* (F6).