

Estudio de un Sistema Operativo

Diseño de Sistemas Operativos

Problemas de aula 1. Introducción

Juan Carlos Pérez & Sergio Sáez

Sergio Sáez ssaez@disca.upv.es

Juan Carlos Pérez jcperez@disca.upv.es

1.1 Introducción

Los sistemas del laboratorio están configurados con varias instalaciones separadas de Linux y Windows en diferentes particiones del disco. Cada instalación está prevista para una asignatura.

Al arrancar cada máquina, toma el control el cargador GRUB (GRand Unifier Bootloader) que reside en el MBR ("Master Boot Record") del disco de arranque y muestra varias opciones, correspondientes a las asignaturas que se dan en el laboratorio.

En el caso de nuestra asignatura, la etiqueta es **Linux Asignaturas GII-DSO**. El GRUB del MBR cederá entonces el control a otro GRUB que esta vez se halla en el sector de arranque de la partición dedicada a la asignatura (**/dev/sda7**). Este segundo GRUB puede dar varias opciones entre las que estará **Linux 2.6.32.72-generic-pae** y otras más en caso de que se hayan compilado núcleos durante las prácticas por parte de otros compañeros.

Recordad siempre apagar el equipo desde la interfaz gráfica o, desde fuera de ella usar **shutdown -h now**, **poweroff** o pulsar **Ctrl-Alt-Supr** antes de apagar. Las escrituras a disco pasan a través de las *buffer-cachés*, en RAM, y por tanto apagar repentinamente el sistema puede acarrear pérdida de datos y supondrá en cualquier caso que al arrancar la vez siguiente se tenga que llevar a cabo una comprobación de los sistemas de ficheros locales.

- La entrada al sistema requiere un nombre de usuario (*login*) y una palabra de paso (*password*).
- Cada usuario entra al sistema con un intérprete de órdenes (*shell*) determinado y en un directorio concreto (*home*). En vuestro caso, debéis entrar con vuestro usuario y palabra de paso de la UPV.
- El sistema "te invita" a escribir una orden mostrándote un *prompt* o símbolo del sistema. El aspecto habitual del *prompt* es similar a este:

```
anita@xpc01:~ $
```


Puede personalizarse al gusto de cada uno. Es buena costumbre que al usuario *root* se le muestre un símbolo distinto recordándole su poder omnímodo:

```
root@xpc01:/root #
```

Antes de continuar, familiaricémonos con el entorno en el que vamos a trabajar: las terminales virtuales y el intérprete de órdenes.

- En modo texto, linux tiene por omisión 6 consolas virtuales, accesibles pulsando **Ctrl-Alt-F1** a **Ctrl-Alt-F6**. Los entornos **F7** y **F8** sirven para alojar hasta dos sesiones gráficas de X-Windows.
- En modo gráfico, existen diversos programas que simulan una terminal de texto: **konsole**, **xterm**, **rxvt**, **eterm**, etc. Las ventajas de estas terminales son:
 - Que guardan una historia visible muy larga, accesible mediante barras de desplazamiento.
 - Que podemos ajustar colores y tipos de letra.
 - Que permiten copiar y pegar (copiar simplemente seleccionando texto y pegar con el botón central del ratón, lo cual es extensible a cualquier aplicación sobre X)
 - Que podemos tener muchas visibles a la vez.

(En el entorno KDE instalado en el laboratorio, las terminales se abren pulsando el

icono  o con la orden **konsole**)

- En cuanto al intérprete de órdenes, dispone de una historia de órdenes y capacidad de completar palabras. Teclas básicas del *bash* (*bourne again shell*):
 - Cursores para moverse en la línea (**izq.** y **der.**) e ir atrás y adelante en la historia de órdenes (**arriba** y **abajo**)
 - **Ctrl-r** para buscar hacia atrás en la historia.
 - **Ctrl-a** y **Ctrl-e** para ir al principio y al final de la línea.
 - **Ctrl-c** para abortar lo que llevemos escrito.
 - **TAB** para completar con posibles órdenes, ficheros o variables.

La mayor parte de los comandos básicos de Emacs funcionan en el *bash*.

Un hábito absolutamente básico va a ser **leer Y ENTENDER los mensajes de error**.

Para fijar este concepto (de acuerdo, sólo es una excusa), veamos algunos mensajes de error procedentes de diversos UNIX desde los años 70. Comprueba cuáles son las respuestas en Linux

```
$ make love
Make: Don't know how to make love.
Stop.
```

```
$ whatis Windows
Windows: nothing
appropriate
```

```
$ find God  
find: God does not exist
```

```
$ sleep with me  
bad character
```

```
$ why did you get a divorce  
Too many arguments
```

```
$ \(-  
bash: (-: Command not found
```

Verás que la orden **why** no está instalada. Puedes instalarla usando:

```
$ sudo apt-get install why
```

La orden **sudo** te permite ejecutar lo que sigue como superusuario y la orden **apt-get install** sirve para instalar paquetes de la distribución Ubuntu

1.2 Las órdenes en UNIX

A excepción de unas pocas órdenes propias del *shell* (se pueden visualizar con la orden **help**), la inmensa mayoría de las órdenes en UNIX son programas ejecutables.

Los programas ejecutables se encuentran típicamente en los directorios **/bin**, **/usr/bin**, y alguno más.

Haz la siguiente prueba:

```
$ echo $PATH
```

Verás los directorios en los que el *shell* busca los ficheros ejecutables. Prueba:

```
ls /bin /usr/bin | wc -l
```

Y verás cuántos ejecutables tienes a tu alcance...

La filosofía UNIX es encapsular las acciones en programas pequeños y permitir la comunicación entre ellos para realizar tareas más complejas.

Fíjate bien que el directorio de trabajo (.), puede no estar en el **path del usuario.** Comprueba si es así en la instalación del laboratorio.

El formato típico de una orden o programa en UNIX es:

orden -opciones lista_argumentos

- **orden** es el nombre del programa.
- **-opciones** es una lista de letras precedidas por el guión, cada opción modifica el funcionamiento del programa en uno u otro sentido.
- **lista_argumentos** son los elementos (ficheros, procesos, etc.) sobre los que actuará la orden. Puede haber 0, 1 o más, separados por blancos (espacios, tabs, etc.). Recuerda que en UNIX siempre debe haber espacios de separación entre los componentes de la línea de órdenes. En MS-DOS, por ejemplo, podemos usar:

```
C: cd/dos
```

En UNIX eso significaría ejecutar el programa **cd/dos**, que probablemente no es lo que queremos. En su lugar, hemos de introducir un espacio entre el **cd** y su

argumento:

```
$ cd /dos
```

Otra forma de especificar las opciones es usando más de una letra, típicamente una o más palabras, siempre en ese caso precedidas por dos guiones en lugar de uno. Comprueba que todas las órdenes siguientes dan idéntico resultado:

```
$ ls -alh
$ ls -a -l -h
$ ls --all -l --human-readable
```

Otro ejemplo:

```
$ ls -l --color /etc /home
```

1.3 Ayuda en línea: orden **man**

UNIX tiene integrada ayuda en línea en forma de "páginas del manual". El manual está dividido en volúmenes o secciones. Éstos son:

Volumen	Contenido
1	Órdenes de usuario
2	Llamadas al sistema
3	Funciones de biblioteca
4	Ficheros especiales
5	Formatos de fichero
6	Juegos
7	Miscelánea
8	Administración y órdenes privilegiadas

Para obtener las páginas de manual de una orden se usa la orden **man [volumen] orden**. Por ejemplo, prueba **man man** 📖 para ver las páginas de la orden **man**. Muchas de las páginas del manual están traducidas al castellano, si quieres ver el manual en inglés, puedes cambiar al "locale" por omisión asignando a la variable de entorno **LC_ALL** el valor **C** mediante la orden:



```
$ export LC_ALL=C.
```

Para volver a ver el manual y los mensajes en castellano, asigna a la variable de entorno **LC_ALL** el valor **es_ES** (español de España) mediante la orden:

```
$ export LC_ALL=es_ES.utf8
```

Para ver qué "locales" hay disponibles, se puede usar la orden:

```
$ locale
```

Para averiguar si existe una orden que realice una función concreta que no sabemos cómo llevar a cabo, puede buscarse por contenido en una base de datos de descripciones de las órdenes usando **apropos**  o **man -k** (la k es de *keyword*). Para consultar las descripciones de una o más órdenes mediante su nombre, expresiones regulares o *wildcards*, puede usarse **whatis**  o bien **man -f**. Prueba:

```
$ whatis mkdir
$ whatis -w mk*
$ whatis -r mk.
$ apropos printer
```

Las páginas de manual de cada orden están en un fichero independiente que se llama **orden.1**, **orden.2**, etc., según el volumen del manual, y se almacenan en directorios como **/usr/share/man/man1**, **/usr/share/man/man2**, etc. Examina algunos directorios:

```
$ ls /usr/share/man
$ ls /usr/share/man/man1
```

Nota: Hoy en día las páginas están comprimidas para ahorrar espacio. De ahí la extensión **.gz**

Ejercicios:

- Obtén la página de manual de **la llamada al sistema mkdir**.
man 2 mkdir
Si no la encuentras es porque no están instaladas las páginas del manual para las herramientas de desarrollo. El paquete Ubuntu se llama **manpages-dev**. Instálalo.
- Averigua cuántas entradas en el manual tiene **mkdir**.
whatis mkdir

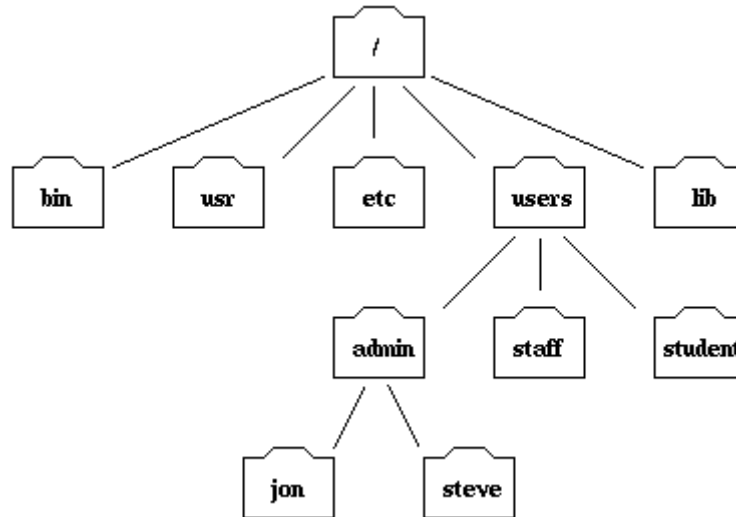
Otros recursos de ayuda:

- Cada vez más, la información más completa va apareciendo en formato "info". Un método más intuitivo de consultar la ayuda en línea en sistemas con el interfaz gráfico GNOME o KDE es mediante el navegador de ayuda de gnome, **gnome-help-browser** o kde, **khelppcenter**. Por ejemplo, el manual *info* de la orden **find** se consultará así:
\$ info find
o bien, con el navegador de gnome/kde
\$ gnome-help info:find
\$ khelppcenter info:find
- En **/usr/share/doc/packages/** está la documentación de todos los paquetes.
- En **/usr/share/doc/howto/** están los ficheros HOW-TO que son breves explicaciones sobre aspectos del sistema. En Insflug se traducen los "HOW-TOS" al castellano.

1.4 Manejo básico de directorios y ficheros

El sistema de ficheros de UNIX es jerárquico, con un directorio raíz **/** en el que encontramos ficheros y directorios. Dentro de estos últimos tenemos más ficheros y directorios, y así sucesivamente.

Jerarquía de directorios en UNIX:



- Un directorio contiene siempre al menos dos elementos. Concretamente dos directorios:
 - `.` (punto) que apunta al propio directorio, y
 - `..` (punto, punto) que apunta a su directorio padre. Los ficheros que empiezan por punto se llaman "ocultos" y la orden `ls` no los muestra a no ser que indiquemos la opción `-a` (de *all*). Probad:


```
$ ls -l
```

```
$ ls -la
```
- Para hacer referencia a un fichero utilizamos un *path* o *pathname* (que en castellano llamaremos ruta o, simplemente, nombre de fichero). Existen dos tipos:
 - nombres absolutos, que empiezan desde el directorio raíz. Un fichero queda identificado siempre con el mismo nombre absoluto:


```
$ ls -la /
```

```
$ ls -la /etc/passwd
```
 - nombres relativos, que empiezan desde el directorio actual. Dependiendo de en qué directorio nos encontremos, el fichero identificado puede variar. Podemos averiguar siempre el directorio actual con la orden `pwd`.


```
$ cd /
```

```
$ ls -la .
```

```
$ cd /etc
```

```
$ ls -la passwd
```

<code>/bin</code>	Ejecutables más básicos, vitales para el sistema.
<code>/boot</code>	Ficheros relativos al arranque y al núcleo.
<code>/dev</code>	Ficheros especiales de dispositivo (discos periféricos, etc.)
<code>/etc</code>	Ficheros de configuración del sistema.
<code>/home</code>	Directorios de los usuarios.
<code>/lib</code>	Bibliotecas básicas y módulos del núcleo.
<code>/mnt</code>	Directorio donde montar sistemas de ficheros temporalmente.

/proc	Pseudodirectorio de acceso a datos de núcleo y procesos.
/root	Directorio <i>home</i> del superusuario (root)
/sbin	Ejecutables básicos para administración del sistema.
/tmp	Para el almacenamiento de ficheros temporales.
/usr	Contiene todo lo demás (bibliotecas, ejecutables, datos, etc.)
/var	Datos variables, <i>spooling</i> , <i>logging</i> , <i>mail</i> , etc.

Para cambiar de directorio se usa la orden **cd**, por ejemplo:

cd	Nos llevan a nuestro directorio <i>home</i> , no importa donde estemos.
cd ~	(La vírgula representa abreviadamente a \$HOME en el <i>bash</i> y se obtiene en el teclado español con AltGr-4 . Un nombre popular para este símbolo es "greña".)
cd /	Nos lleva al directorio raíz.
cd ..	Nos lleva al directorio padre del actual.
cd .	No tiene efecto (nos lleva al directorio actual).
cd -	Nos lleva al último directorio en el que estuvimos antes de llegar al actual.

Ejercicios:

- ¿Es relativo o absoluto el *path* **/etc** ? Absoluto
- ¿Es relativo o absoluto el *path* **../etc/X11/X** ? Relativo
- ¿Es relativo o absoluto el *path* **/etc/X11/./passwd** ? Absoluto
- ¿Es relativo o absoluto el *path* **X11/X** ? Relativo
- Suponiendo que el directorio actual es **/home/alumno/dir_prueba**, ¿a qué directorios, expresados en forma absoluta, nos llevan las ordenes siguientes?
 - **\$ cd ../tmp/** a **/home/alumno/tmp**
 - **\$ cd /tmp** a **/tmp**
 - **\$ cd tmp** a **/home/alumno/dir_prueba/tmp**
 - **\$ cd ~/tmp** a **/home/alumno/tmp**







1.5 Órdenes que usaremos

Del gran número de órdenes disponibles en un sistema UNIX, vamos a entresacar las que más utilizaremos en estas prácticas.

- **ls**: Lista y muestra los atributos de ficheros y directorios. 📁
 - Ejemplos:


```
$ ls -lr /etc/i
```

- **more:** Visualiza página a página un fichero. Con **return** se baja una línea, con **espacio** una página, y con **q** se sale antes del final 
 - Ejemplos:
`$ more /etc/services`
 - Órdenes relacionadas:
 - **less:** Más "amigable". Van los cursores, etc.
 - **zless:** Para ver directamente ficheros comprimidos.
- **mkdir** Crea un directorio 
 - Ejemplos:
`$ mkdir /home/alumno/mi_dir`
`$ mkdir -p mi_dir/otro_dir`
- **rmdir:** Borra un directorio vacío 
 - Ejemplos:
`$ rmdir /home/alumno/mi_dir/otro_dir`
- **cp:** Copia ficheros y directorios. **Necesita al menos 2 parámetros...** 
 - Ejemplos:
`$ cp /etc/i* .`
`$ cp -r /etc/init.d .`
- **mv** Cambia de nombre o mueve a otro directorio un fichero. 
 - Ejemplos:
`$ mv k1 k99`
`$ mv /home/alumno/k1 /home/alumno/mi_dir`
- **rm:** Borra ficheros y directorios. 
 - Ejemplos:
`$ rm /home/alumno/k1`
`$ rm -r /home/alumno/mi_dir`
- **ln:** Crea un enlace a un fichero. El enlace puede ser *duro* o *simbólico*. 
 - Ejemplos:
`$ ln fich_existente nuevo_fich`
`$ ln -s fich_existente nuevo_fich`
- **diff:** Compara dos ficheros. 
 - Ejemplos:
`$ diff prog1.c prog2.c`
- **grep:** Busca patrones en ficheros 

- Ejemplos:
`$ grep www /etc/services`
- **locate:** Búsqueda básica de ficheros en todo el sistema. 
 - Ejemplos:
`$ locate profile`
`$ locate limit`
- **which:** Nos dice qué ficheros se ejecutarían si escribiéramos sus argumentos en el actual *shell* 
 - Ejemplos:
`$ which ls`
`$ which which`
- **gzip:** Comprime ficheros. 
 - Ejemplos:
`$ cp /usr/share/dict/spanish . ; gzip spanish`
 - Órdenes relacionadas:
 - **gunzip:** Descomprime lo que comprime gzip. 
 - **bzip2, bunzip2:** Un compresor normalmente más eficaz. 
- **tar:** Empaqueta y desempaqueta ficheros. 
 - Ejemplos:

Crea un fichero comprimido con el contenido del directorio `/home/alumno`
`$ tar -czvf copia.tgz /home/alumno`

Muestra el contenido del archivo comprimido `copia.tgz`
`$ tar -tzvf copia.tgz`

Extrae el contenido del archivo comprimido `copia.tgz` en el directorio actual
`$ tar -xzvf copia.tgz`

1.6 El compilador de C de GNU

- El "GNU CC" llamado comúnmente **gcc** es el conjunto de compilación de GNU. Compila programas escritos en "C", "C++", y "Objective C" a través de los 4 siguientes pasos:
 1. Preprocesado
 2. Compilación del código "C"

3. Ensamblado

4. Enlazado

- El proceso puede detenerse tras cualquiera de estos pasos y pueden examinarse o utilizarse los resultados intermedios.
- El compilador admite opciones para optimizar, introducir información de depuración, dar información de errores y avisos ("warnings"), compilar para varias arquitecturas y admite también numerosas extensiones al lenguaje "C" estándar (tanto en el "dialecto" de Kernighan and Ritchie, como en el ANSI c89 o c99. Ver la opción **-std**)

1.7 Compilación

- Recorreremos las diferentes características del compilador mediante un ejemplo sencillo:

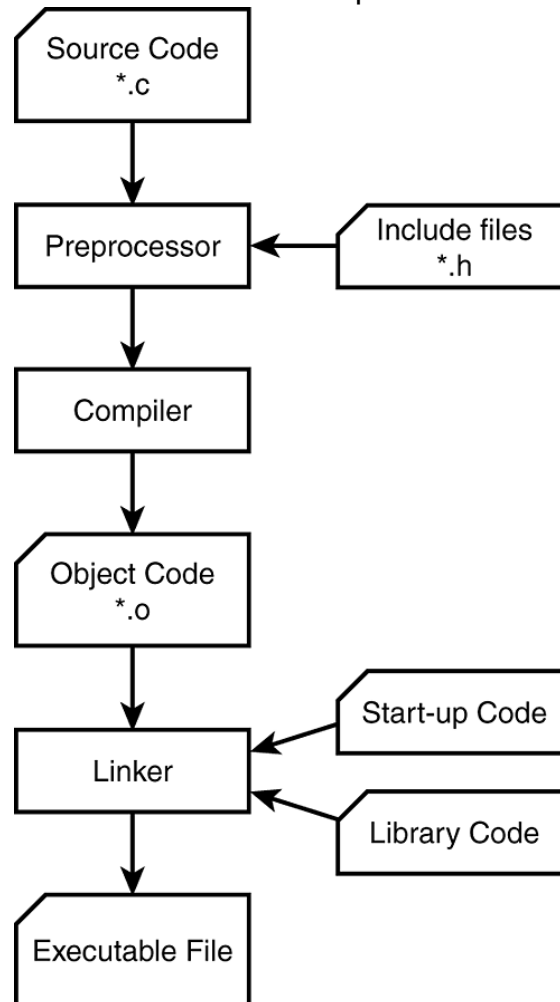
```
/*  
  hola.c - Programa "Hola mundo!" sencillo  
*/  
#include <stdio.h>  
int main(void)  
{  
    printf("Hola y todo ESO...!\n");  
    return 0;  
}
```

- Para editar, compilar y ejecutar este programa:

```
$ emacs hola.c  
$ gcc hola.c -o hola  
$ ./hola  
Hola y mis mejores DSOs!
```

- El proceso de compilación que se ha seguido al ejecutar esta simple orden consta, como se ha dicho antes, de varios pasos y ha requerido de diversas fuentes de información, tal y como se muestra en la figura:

Proceso de compilación



- Vamos a repasar estos pasos manualmente: En primer lugar le diremos al **gcc** que se detenga tras ejecutar el preprocesador usando la opción **-E**. La opción **-o** (de *output*) va siempre seguida del nombre de fichero que queremos generar. En este caso **hola.cpp**

\$ gcc -E hola.c -o hola.cpp

- Examina **hola.cpp** y comprobarás que se han incluido los contenidos de **stdio.h** junto con otros elementos introducidos por el preprocesador.
- La extensión escogida, **.cpp** es usada a menudo para código emitido por el preprocesador. Además, el compilador espera las siguientes extensiones de forma estándar:

Extension	Fichero
.c	Fichero fuente en "C"
.C, .cc	Fichero fuente en "C++"
.i	Fichero fuente en "C" preprocesado
.ii	Fichero fuente en "C++" preprocesado
.S, .s	Fichero fuente en ensamblador
.o	Código objeto compilado (relocalizable)
.a, .so	Código objeto de biblioteca (relocalizable)

- El paso siguiente es compilar **hola.cpp** y generar código objeto. Para indicarle al compilador que empiece a trabajar tras la fase de preprocesado usamos la opción **-x**

cpp-output y para indicarle que hay que detenerse tras la generación de código objeto usaremos la opción **-c**:

```
$ gcc -x cpp-output -c hola.cpp -o hola.o
```

- Finalmente generaremos el fichero ejecutable (que normalmente en UNIX no lleva extensión) simplemente indicándole al compilador que tome el fichero objeto y no se detenga antes de terminar. Si no se le da un nombre específico al ejecutable resultante mediante la opción **-o**, el compilador le da el nombre estándar **a.out**

```
$ gcc hola.o -o hola
```

- Normalmente los programas constan de más de un fichero fuente: el principal y otros que contienen funciones y variables que usará el principal u otros de entre ellos.
- Se suele crear un fichero de cabecera para cada fichero fuente que exporta funciones o variables. La extensión que se le da es normalmente **.h** (de *header*, "cabecera")
 - Fichero fuente **funciones.c**:

```
/*
funciones.c - Funciones auxiliares para quetal.c
*/

#include <stdio.h>

void saludar(void)
{
    printf("Qué tal? Cómo va ES0?\n");
}
```

- Fichero de cabecera **funciones.h**:

```
/*
funciones.h - Fichero de cabecera de funciones.c
*/

void saludar(void);
```

- Fichero con el programa principal **quetal.c**:

```
/*
quetal.c - Programa "Hola mundo" modificado...
*/

#include <stdio.h>

#include "funciones.h"
int main(void)
{
    printf("Hola!\n");
    saludar();
}
```

```
    return 0;  
}
```

- Compilación: Lo habitual es compilar cada fichero fuente excepto el que contiene el programa principal, creando así ficheros objeto `.o` (o bibliotecas) y luego compilar el programa principal, enlazando los objetos y las bibliotecas en un último paso.

```
$ gcc -c funciones.c  
$ gcc quetal.c funciones.o -o quetal
```

alternativamente

```
$ gcc -c funciones.c  
$ gcc -c quetal.c  
$ gcc quetal.o funciones.o -o quetal  
  
$ gcc quetal.c funciones.c -o quetal
```

- A veces los ficheros de cabecera están en directorios distintos al actual. Para que el compilador los encuentre se usa la opción `-I`. Por omisión, el compilador consulta siempre los directorios `/usr/local/include`, y `/usr/include`, entre otros.

Crea un directorio `cabeceras`, mueve ahí `funciones.h` y compila de nuevo el programa `quetal`

- Además de los ficheros objeto que se quieren enlazar con el programa que se está compilando, se pueden especificar también bibliotecas (*libraries*) con las que enlazar. Para ello se usa la opción `-l` seguida por el nombre de la biblioteca sin "lib" al principio y sin la extensión. Por ejemplo, para usar la biblioteca `libm.so`, se pondrá en la línea de compilación `-lm`

Añade una línea en el programa `quetal.c` para que se imprima el seno de PI medios (constante `M_PI_2`) después del saludo. (La cabecera de la función `sin()` está definida en `math.h`)

Prueba a compilar sin la biblioteca matemática `libm.so` y examina qué error obtienes.

¡OJO! Las nuevas versiones del GCC incluyen la mayoría de las funciones matemáticas clásicas integradas ("compiler built-ins"). Para evitar que el GCC use la versión integrada de la función seno (y provocar que nos dé el error que esperamos), hay que usar la opción `gcc -fno-builtins`

- Pueden existir dos versiones de cada biblioteca con idéntica funcionalidad:
 - La que se enlaza de forma estática (v.g. `libm.a`), y se incluye en el propio programa ejecutable del usuario.
 - La de enlace dinámico (v.g. `libm.so`) que se incorpora al proceso de usuario sólo cuando es necesario, y su código es compartido por todos los procesos que la

utilizan.

Por omisión el compilador utiliza la versión dinámica de las bibliotecas.

- Para averiguar de qué bibliotecas dinámicas depende un programa se puede utilizar la orden **ldd**. Probad con **ldd quetal**.

Debería mostrar una lista de todas las bibliotecas dinámicas de las que dependen el programa y su localización. Hay que tener en cuenta que podría haber más de una versión de la misma biblioteca dinámica en el sistema (una versión genérica, una optimizada para un procesador específico, etc.)

- Para compilar utilizando las versiones estáticas de cada biblioteca se utiliza la opción **-static** del compilador. Ojo! el orden de los parámetros en este caso es muy importante. Los ficheros que se van a compilar deben ir antes de la opción **-static** y de las posibles bibliotecas (v.g. **-lm**)

```
$ gcc quetal.c funciones.c -static -o quetal-s
```

Comprobad la diferencia de tamaño de los ficheros ejecutables. Utilizad la orden **strip** para eliminar los símbolos de los ficheros ejecutables y comprobad como se reduce todavía más su tamaño.