PRÁCTICA 2:

Entrenamiento con el toolchain de GNU y QEMU para sistemas empotrados

David Martínez García

Alejandro Gea Belda

Tarea 1: Toolchain de GNU

1a)

man - Muestra el manual de usuario de cualquier comando que se pueda utilizar en el terminal. Se usa como: `man [COMANDO]`

size - Muestra el número de bytes necesarios por los segmentos de texto, datos y objetos de nuestro programa. Las dos últimas filas son la suma en decimal y hexadecimal.

```
$ size cat
   text    data     bss     dec     hex filename
  29150    1148     416   30714    77fa cat
```

strip - Descarta todos los símbolos de los archivos objetos. Esto puede ser utilizado para minimizar el tamaño de los archivos. Se usa como: `strip [OPCIONES] objfile [...]`

nm - Muestra información de los símbolos en los ficheros objeto. Se define como `nm [NOMBRE DEL ARCHIVO]`

ldd - Muestra por el terminal las librerías compartidas utilizadas por cada programa o por cada objeto. Se define como: `ldd [OPCIONES]... file...`

strings - Imprime las cadenas de caracteres que hay en los archivos dados. Sintaxis: `strings [OPCIONES] ...`

objdump - Muestra información sobre uno o más ficheros objeto. El tipo de información que muestra el comando depende de las opciones. Sintaxis: `objdump [OPCIONES] objfile ...`

readelf - Muestra información sobre uno o más archivos en formato elf. Sintaxis: `readelf [OPCIONES] objfile ...`

objcopy - Copia los contenidos de un fichero objeto a otro fichero destino. Sintaxis: `objcopy [options] infile [outfile]...`

as/gas - Lee un archivo e inicia el compilador (por defecto en lenguaje C). La salida por defecto de este comando es a.out. Sintaxis: `as [ARCHIVO][OPCIONES].`

wc - Muestra por pantalla los saltos de línea, las palabras y el contador de bytes de un archivo. Sintaxis: `wc [OPCIONES]... [ARCHIVO]...`

free - Muestra la cantidad de memoria usada y disponible en el sistema. Sintaxis: `free [OPCIONES]`

df - Muestra el uso del disco duro y otras informaciones como punto de montaje y sistema de ficheros. Se usa como: `df [PARÁMETROS]`

grep/egrep - Busca en uno o más archivos de entrada líneas que contengan una coincidencia con un patrón específico. De forma predeterminada, Grep genera las líneas coincidentes. `grep [OPCIONES] [EXPRESIÓN REGULAR] [ARCHIVO]`

less - Muestra página a página el contenido del fichero que se le pasa como argumento. El uso más habitual es: `less [NOMBRE ARCHIVO]`

more - Sirve para ver (pero no modificar) el contenido de un archivo o comando y visualizarlo por páginas. Sintaxis: `more [OPCIONES] [NOMBRE ARCHIVO]`

find - Realiza búsquedas utilizando parámetros, en una amplia variedad de condiciones, ya sea encontrar ficheros modificados por fecha, o bien por tamaño, permisos, usuarios, grupos, o tipos de archivo. Sintaxis: `find [RUTA] [PARÁMETROS] [VALORES]`

tail - Muestra de manera predeterminada las 10 últimas líneas (aunque esto se puede modificar) de un archivo de texto o de varios. Pasa usarlo: `tail [NOMBRE ARCHIVO]`

strace - Utilidad de línea de órdenes para comprobación de errores en el sistema operativo GNU/Linux. Permite vigilar las llamadas al sistema usadas por un determinado programa y todas las señales que éste recibe. Para ejecutarlo usaremos: `strace [COMANDO]`

ld - Combina un número de ficheros objeto y archivos, reubica sus datos y enlaza referencias de símbolos. Sinopsis: `ld [PARÁMETROS] OBJFILE…`

sudo - Permite a los usuarios ejecutar programas con los privilegios de seguridad de otro usuario de manera segura, convirtiéndose así temporalmente en el otro usuario durante la ejecución del programa. Sintaxis: `sudo [PARÁMETROS]`

vim|nano|joe - Se tratan de editores de texto de código abierto y libre disponible en la mayoría de los sistemas Unix. Sinopsis: `vim|nano|joe [OPCIONES] [NOMBRE ARCHIVO]`

md5sum - La función devuelve un valor que es prácticamente único para cada archivo, con la particularidad que una pequeña variación en el archivo provoca una salida totalmente distinta, lo que ayuda a detectar si el archivo sufrió alguna variación. En cuanto a la sintaxis añadiremos el comando md5sum al fichero deseado: `md5sum [NOMBRE ARCHIVO]`

file - Permite detectar el tipo y formato de un archivo. La invocación del comando **file** tiene el siguiente formato: `file [PARÁMETROS] ARCHIVO…`
Los parámetros posibles son por ejemplo: -d (Realiza las pruebas de sintaxis y de números mágicos del sistema. Esta es la opción por defecto), -h (Si el archivo a analizar es un enlace simbólico, lo identifica como tal), etc.

1b)

Ejemplo 1: Saber cuántas veces aparece una cierta palabra en un archivo de texto (sin tener en cuenta mayúsculas y minúsculas) y guardar el resultado en el archivo salida.txt

```
alu@UbuntuVirtualEPS2021: ~/Documentos/Sistemas Empotrados/P2
alu@UbuntuVirtualEPS2021:~/Documentos/Sistemas Empotrados/P2$ ls
alu@UbuntuVirtualEPS2021:~/Documentos/Sistemas Empotrados/P2$ touch entrada.txt
alu@UbuntuVirtualEPS2021:~/Documentos/Sistemas Empotrados/P2$ ls
entrada.txt
alu@UbuntuVirtualEPS2021:~/Documentos/Sistemas Empotrados/P2$ nano entrada.txt
alu@UbuntuVirtualEPS2021:~/Documentos/Sistemas Empotrados/P2$ touch salida.txt
alu@UbuntuVirtualEPS2021:~/Documentos/Sistemas Empotrados/P2$ ls
entrada.txt  salida.txt
alu@UbuntuVirtualEPS2021:~/Documentos/Sistemas Empotrados/P2$ grep -o -i Empotrados entrada.txt | wc -l > salida.txt
alu@UbuntuVirtualEPS2021:~/Documentos/Sistemas Empotrados/P2$ nano salida.txt
alu@UbuntuVirtualEPS2021:~/Documentos/Sistemas Empotrados/P2$
```

```
alu@UbuntuVirtualEPS2021: ~/Documentos/Sistemas Empotrados/P2
  GNU nano 4.8                                              entrada.txt
Ejemplo para la práctica 2
Palabra que hay que buscar: Empotrados.
Empotrados empotrados empotrados empotrados
Empotrados Empotrados empotrados
Empotrados Empotrados empotrados
Total: 11 veces.
```

```
alu@UbuntuVirtualEPS2021: ~/Documentos/Sistemas Empotrados/P2
  GNU nano 4.8                                               salida.txt
11
```

Ejemplo 2: Eliminar los carácteres especiales que tiene un archivo para así contar el número de palabras que se repite una palabra que se ha formado al eliminar los carácteres especiales.

1c)

Los archivos ELF tienen diferentes secciones que nos dan información del contenido del archivo. Algunas de estas secciones son:

- Text: ubicación y tamaño del código del programa
- Data: Ubicación y tamaño de las tablas y variables globales
- Rodata: Ubicación y tamaño de las variables tipo string
- Bss: Ubicación y tamaño de variables sin inicializar

Como ejemplo se ha creado un archivo de código en c que presenta un par de definiciones de variables y alguna operación aritmética sencilla.

Si compilamos el programa y escribimos por terminal el comando `readelf -a [NOMBRE FICHERO OBJETO]` obtenemos por terminal toda la información contenida en el fichero .elf. Al principio tenemos información general del fichero como su versión o su tamaño.

```
Encabezado ELF:
  Mágico:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Clase:                             ELF64
  Datos:                             complemento a 2, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  Versión ABI:                       0
  Tipo:                              DYN (Fichero objeto compartido)
  Máquina:                           Advanced Micro Devices X86-64
  Versión:                           0x1
  Dirección del punto de entrada:               0x10a0
  Inicio de encabezados de programa:          64 (bytes en el fichero)
  Inicio de encabezados de sección:         14808 (bytes en el fichero)
  Opciones:                          0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```

Debajo de esta información encontramos todos los encabezados del archivo junto con su tamaño y su dirección en el disco. Algunos de los encabezados que nos interesan son:

```
Encabezados de Sección:
  [Nr] Nombre              Tipo             Dirección         Despl
       Tamaño              TamEnt           Opts   Enl   Info  Alin
  [ 0]                     NULL             0000000000000000  00000000
       0000000000000000    0000000000000000           0     0     0
  [16] .text               PROGBITS         00000000000010a0  000010a0
       0000000000000225    0000000000000000  AX       0     0     16
  [18] .rodata             PROGBITS         0000000000002000  00002000
       0000000000000015    0000000000000000   A       0     0     4
  [25] .data               PROGBITS         0000000000004000  00003000
       0000000000000010    0000000000000000  WA       0     0     8
  [26] .bss                NOBITS           0000000000004010  00003010
       0000000000000008    0000000000000000  WA       0     0     1
```

Otra manera de mirar el tamaño de los diferentes encabezados del archivo es utilizando el comando `size [NOMBRE FICHERO OBJETO]`

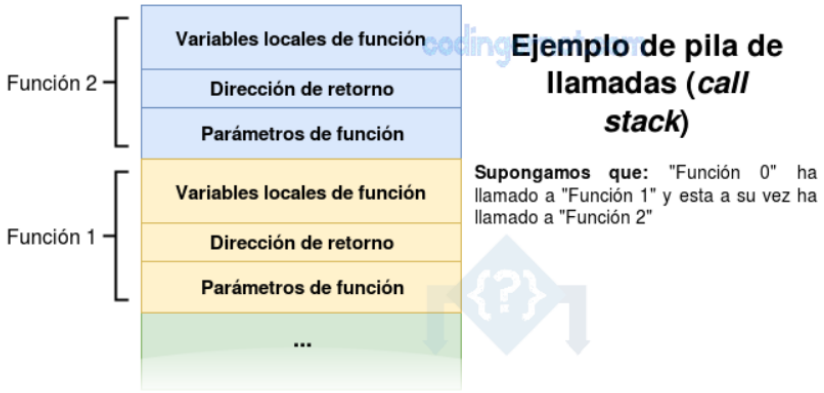
```
alumno@VDIUbuntuEPS2022:~/Escritorio/Empotrados/Practica1$ size ejecutable
   text    data     bss     dec     hex filename
   1944     616       8    2568     a08 ejecutable
```

En este caso tenemos que el programa creado tiene 1944 bits de datos, 616 bits de variables globales, 8 bits de variables sin inicializar y su tamaño total es de 2560 bits.

1d)

La pila de llamadas (stack) es una estructura de datos que se dedica a almacenar las subrutinas en una ejecución de un programa. Por ejemplo, al hacer una llamada a una función, se reserva un bloque de la pila encargado de guardar las variables y otros datos utilizados en la ejecución de la función. Una vez finalizada la función, ese bloque de memoria se libera para estar disponible más adelante si fuera necesario.

La pila tiene una estructura donde el último elemento puesto es el primero que se resuelve



Por otro lado, el almacenamiento libre (heap) permite almacenar memoria dinámica durante la ejecución. En este espacio es donde se van almacenando y borrando las variables dinámicas que se utilicen en nuestro código. Esta estructura de datos no sigue una metodología predefinida como la pila, así que es complicado mantener el control de los bloques de memoria reservados.

1e)

El heap no tiene un tamaño definido porque es el encargado de almacenar la memoria dinámica. Por ejemplo, si en un código creamos un vector de enteros que va creciendo con el tiempo, el tamaño del heap también crecerá.

CUADERNOS DE BITÁCORA

- 25/09/2023 -

David Martínez y Alejandro Gea: Se ha repartido el trabajo para empezar a buscar los términos correspondientes a la primera parte de la práctica (apartado 1a)

- 02/10/2023 -

Para memorias: latex overlift