Práctica 2 – Sistemas Empotrados

TAREA 1.a:

* César: 12 primeros comandos
* Lucas: 12 últimos

TAREA 1.c:

* César: 2 últimas
* Lucas: 3 primeras

TAREA 1.d: César

TAREA 1.e: Lucas

tarea 1

1a)

**man**: muestra ayuda sobre otros comandos, incluyendo una descripción de sus argumentos.

A screenshot of a computer

Description automatically generated

**size**: muestra el tamaño de un programa en decimal y hexadecimal, además del tamaño del texto en número de caracteres, los datos y las variables no usadas.

A close up of a text

Description automatically generated

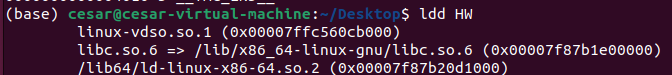
**strip**: elimina información de depuración del ejecutable.

**nm**: lista los símbolos de un archivo ejecutable.

A screenshot of a computer program

Description automatically generated

**ldd**: muestra dependencias compartidas de un programa.



**strings**: muestra las cadenas de texto legibles de un archivo ejecutable.

A computer screen shot of a computer

Description automatically generated

**objdump**: proporciona diversos tipos de información detallada sobre un archivo ejecutable.

A screen shot of a computer screen

Description automatically generated

Por ejemplo, con -a obtendríamos información acerca del título y el tipo de formato del archivo.

A black background with white text

Description automatically generated

**readelf**: muestra información acerca de la estructura de un archivo ejecutable.

A screen shot of a computer program

Description automatically generated

**objcopy**: crea un nuevo archivo ejecutable copiando el proporcionado como primer argumento.

A screenshot of a computer

Description automatically generated

**as**/**gas**: ensambla y genera el código objeto de un programa en ensamblador (.s).

A computer screen shot of white text

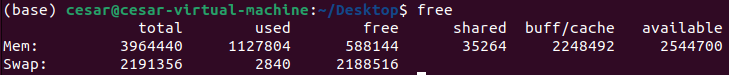
Description automatically generated

**wc**: muestra las líneas, palabras y caracteres, en ese orden, que contiene un archivo de texto.

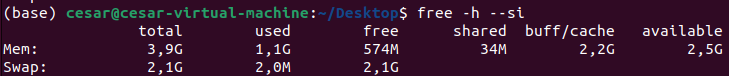
A screenshot of a computer

Description automatically generated

**free**: muestra la cantidad de memoria libre y ocupada del sistema, en kibibytes.



Usando los argumentos adicionales -h y --si, se obtiene una lectura más comprensible.

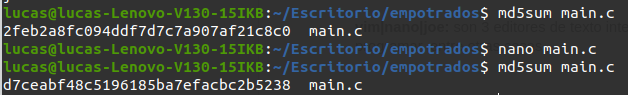


---------------------

**file:** te da información sobre un archivo como su tipo y contenido



**md5sum:** Proporciona una secuencia alfanumérica para un archivo que cambia con cada modificación que se le haga, de esta manera se pueden comprobar que las instalaciones han ido correctamente o si se ha modificado el contenido.



**vim|nano|joe:** son 3 editores de texto integrados en el entorno de la terminal, sirven para programar, entre otras cosas

* Vim: A black screen with text

  Description automatically generated
* Nano:A screenshot of a computer

  Description automatically generated
* Joe: A screenshot of a computer

  Description automatically generated

**sudo:** otorga privilegios de administrador al usuario para la ejecución de ese comando

**ld:** sirve para enlazar varios archivos objeto a un archivo ejecutable. De esta manera un programa puede, por ejemplo, ejecutar funciones que estén desarrolladas en otros archivos de código fuente

**strace:** Muestra las llamadas al sistema que realiza un programa, como las impresiones por pantalla o las reservas de memoria para las variables.

**Tail:** Muestra por pantalla las últimas líneas (por defecto 10) de un archivo de texto, también tiene la opción de ir mostrando sus actualizaciones



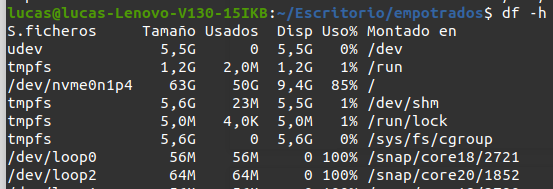
**find**

**more**

**less**

**grep/egrep**

**df:** Se utiliza para conocer información sobre el espacio libre en el sistema, la opción -h lo hace más cómodo de leer



1b)

Esta concatenación de comandos cuenta el número de veces que aparece la palabra indicada como argumento en el archivo de texto dado, ordenando cada aparición en una línea y contando después la cantidad de estas. El resultado lo imprime en un archivo de texto junto con una etiqueta.

En un primer momento se imprimían en líneas diferentes así que se ha utilizado un comando adicional que sustituye el salto de línea por un espacio en un nuevo fichero. Finalmente, se borra el primer fichero generado y se guarda únicamente el que interesa.

A screenshot of a computer program

Description automatically generated

Este segundo ejemplo consiste en mandar a un archivo .txt la información de otro archivo (también .txt en este caso). En lugar del método anterior, se ha utilizado *printf* para imprimir en el archivo de destino con el formato deseado. De esta manera resulta mucho más sencillo manejar los saltos de línea y no hace falta trabajar con archivos temporales.

A screenshot of a computer

Description automatically generated

1c)

**.bss:** La sección .bss no ocupa espacio en el archivo .ELF. En esta sección se encuentran las variables que se inicializan en tiempo de ejecución, de modo que al cargar el programa en memoria se les reserva espacio en la sección .bss y no en la sección de datos.

**.data:** La sección .data sí ocupa espacio en el archivo .ELF y almacena las variables inicializadas. Durante la ejecución del programa estos valores se copian al espacio de memoria que corresponda.

**.rodata:** La sección .rodata corresponde a valores constantes dentro del programa, que no serán modificados. Estos datos son invariables durante el tiempo de ejecución.

**.noinit**: almacena datos que no se inicializan automáticamente al cargar el programa en memoria. Esta sección es útil para datos que deben conservar su valor entre reinicios o que no necesitan inicialización inmediata.

**.text**: almacena el código ejecutable del programa, es decir, las instrucciones de máquina que el procesador ejecutará para realizar las operaciones del programa, determinando su lógica y comportamiento.

1d)

Estos dos conceptos, se refieren a áreas de la memoria utilizadas para almacenar datos en tiempo de ejecución, pero tienen distinta finalidad y características. Mientras que el stack está destinada a la gestión de datos relacionados con la ejecución de funciones, como las variables locales o direcciones de retorno, el heap es una estructura dinámica compartida por todos los subprocesos utilizada para almacenar datos en ejecución cuyo tamaño o duración no se conoce en tiempo de compilación.

En cuanto a manejo y eficiencia, el stack libera la memoria de forma auomática al salir del contexto de la función con una gestión rápida y eficiente. Por el contrario, en el heap la liberación de memoria debe explicitarse por el programador (aunque algunos lenguajes como C# constan de su propio recolector de basura) y el acceso a esta memoria es más lento.

1e)

El heap es la zona de memoria dedicada a almacenar datos en tiempo de ejecución. Es decir, es donde se reservan las zonas de memoria dinámica. Por este motivo no sería coherente que tuviera un tamaño definido ya que en tiempo de ejecución podría verse superado o desaprovechado en función de los requisitos del programa.

Al no tener un tamaño definido para el heap, los programas pueden solicitar y liberar memoria según lo necesiten y conseguir una gestión de la memoria más eficiente.