Un sistema operativo es un conjunto de programas y utilidades básicas que hacen que su computadora funcione. El centro de un sistema operativo es el núcleo (N. del T.: kernel). El núcleo es el programa más importante en la computadora, realiza todo el trabajo básico y le permite ejecutar otros programas.

**Teoría de elementos a utilizar**

**S.O. Debian**

Se usará esta distribución de Linux para usar su terminal y descargar las herramientas para compilar y crear los archivos que consolidarán el proyecto, esto por ser compatible con las mismas herramientas y varias arquitecturas de procesadores

Debian se caracteriza por:

* La disponibilidad en varias arquitecturas. La versión estable incluye soporte para 12 plataformas:
  + i386 – x86-32
  + amd64 – x86-64
  + alpha – DEC Alpha
  + sparc – Sun SPARC
  + arm – Arquitectura ARM
  + armel – Emulador de ARM Emulator
  + powerpc – Arquitectura PowerPC
  + ia64 – Arquitectura Intel Itanium (IA-64)
  + mips, mipsel – Arquitectura MIPS (big-endian y little-endian)
  + s390 – Arquitectura IBM ESA/390 y z/Architecture
  + m68k – Arquitectura Motorola 68k en Amiga, Atari, Mac, y varios sistemas embebidos VME
* Una amplia colección de software disponible. La actual versión 8.0 viene con más de ≈ 43.0004 paquetes y la versión 6.0 con casi ≈ 30.000.2
* Un grupo de herramientas para facilitar el proceso de instalación y actualización del software (APT, Aptitude, Dpkg, Synaptic, Dselect, etc.) Todas ellas obtienen información de donde descargar software desde /etc/apt/sources.list, que contiene los repositorios.
* No tiene marcado ningún entorno gráfico en especial, pudiéndose no instalar ninguno, o instalar GNOME, KDE, Xfce, LXDE, Enlightenment u otro.

http://cdimage.debian.org/debian-cd/8.6.0/i386/iso-cd/debian-8.6.0-i386-netinst.iso

**QEMU**

QEMU es un emulador de procesadores basado en la traducción dinámica de binarios (conversión del código binario de la arquitectura fuente en código entendible por la arquitectura huésped). QEMU también tiene capacidades de virtualización dentro de un sistema operativo, ya sea GNU/Linux, Windows, o cualquiera de los sistemas operativos admitidos; de hecho, es la forma más común de uso. Esta máquina virtual puede ejecutarse en cualquier tipo de Microprocesador o arquitectura (x86, x86-64, PowerPC, MIPS, SPARC, etc.). Está licenciado en parte con la LGPL y la GPL de GNU.

El objetivo principal es emular un sistema operativo dentro de otro sin tener que reparticionar el disco duro, empleando para su ubicación cualquier directorio dentro de éste.

El programa no dispone de GUI, solo se ejecuta la máquina por medio del comando establecido (en el caso de Linux)

*Características*

* Soporta emulación de IA-32 (x86) PC, AMD64 PC, MIPS R4000, Sun's SPARC sun4m, Sun's SPARC sun4u, ARM development boards (Integrator/CP y Versatile/PB), SH4 SHIX board, PowerPC (PReP y Power Macintosh), y arquitecturas ETRAX CRIS.
* Soporte para otras arquitecturas en host y sistemas emulados (ver página principal para una lista completa).
* Aumento de velocidad — algunas aplicaciones pueden correr a una velocidad cercana al tiempo real.
* Implementa el formato de imagen de disco Copy-On-Write. Se puede declarar una unidad virtual multi-gigabyte; la imagen de disco ocupará solamente el espacio actualmente utilizado.
* Implementa la superposición de imágenes. Se puede mantener el estado de una imagen del sistema huésped, y escribir cambios en un archivo de imagen separado. De esa forma, si por ejemplo el sistema huésped se colapsa, es sencillo volver a un estado anterior que haya sido guardado.
* Soporte para ejecutar binarios de Linux en otras arquitecturas.
* Es posible salvar y restaurar el estado de la máquina (por ejemplo programas en ejecución).
* Emulación de tarjetas de red virtuales.
* Soporte SMP.
* El sistema operativo huésped no necesita ser modificado o parcheado.
* Las utilidades de línea de comandos permiten un control total de QEMU sin tener que ejecutar X11.
* Control remoto de la máquina emulada a través del servidor VNC integrado.

apt-get install qemu

**NASM**

El Netwide Assembler o NASM, es un ensamblador libre para la plataforma Intel x86. Puede ser usado para escribir programas tanto de 16 bits como de 32 bits (IA-32). En el NASM, si se usan las bibliotecas correctas, los programas de 32 bits se pueden escribir de una manera tal para que sean portables entre cualquier sistema operativo x86 de 32 bits. El paquete también incluye un desensamblador, el NDISASM.

*Características*

El NASM puede generar varios formatos binarios en cualquier máquina, incluyendo COFF (y el ligeramente diferente formato Portable Executable usado por Microsoft Windows), el a.out, ELF, Mach-O, y el formato binario nativo Minix. El NASM incluso define su propio formato binario, RDOFF, que es usado actualmente solamente por el proyecto del sistema operativo RadiOS).

La variedad de formatos de la salida permite a uno portar los programas a virtualmente cualquier sistema operativo x86. Además, el NASM puede crear archivos binarios planos, usables para escribir boot loaders (cargadores de arranque), imágenes ROM, y varias facetas del desarrollo sistemas operativos. El NASM incluso puede correr en plataformas diferentes del x86, como SPARC y PowerPC, aunque no puede producir programas usables por esas máquinas.

El NASM usa la tradicional sintaxis de Intel para el lenguaje ensamblador x86, mientras que otros ensambladores libres, como el ensamblador del GNU (GAS), utilizan la sintaxis de AT&T. También evita características como la generación automática de sobreescritura (override) de segmentos y la relacionada directiva ASSUME usada por el MASM y los ensambladores compatibles, pues estas pueden ser a menudo confusas -- los programadores deben seguir por sí mismos el contenido de los registros de segmento y la localización de variables a los que éstos se refieren.

*Linking*

El NASM produce principalmente archivos objeto, que por lo general no son ejecutables por sí mismos. La única excepción a esto son los binaries planos (.COM) que son inerentemente limitados en el uso moderno. Para traducir los archivos objeto a programas ejecutables, se debe usar un enlazador apropiado, por ejemplo, la utilidad "LINK" del Visual Studio de Windows, o el LD para sistemas similares a UNIX (como Gnu/Linux).

apt-get install nasm

**El compilador GCC**

GCC es un compilador integrado del proyecto GNU para C, C++, Objective C y Fortran; es capaz de recibir un programa fuente en cualquiera de estos lenguajes y generar un programa ejecutable binario en el lenguaje de la máquina donde ha de correr.

La sigla GCC significa "GNU Compiler Collection". Originalmente significaba "GNU C Compiler"; todavía se usa GCC para designar una compilación en C. G++ refiere a una compilación en C++.

gcc [ opción | archivo ] ...   
 g++ [ opción | archivo ] ...

apt-get install gcc

**grub-mkrescue**

Genera un GRUB bootable para cargar los archivos que se le especifican como parámetros  
Todos los argumentos que se señalan explícitamente como opciones en la instrucción grub-mkrescue se transmiten a “xorriso” en modo de emulación.  
  
Ya que el xorriso nativo instalado en las distribuciones Linux actuales (Debian) no es compatible para crear GRUB 2, lo cual se necesita para consolidar archivos se necesita instalar una versión más reciente  
apt-get install xorriso

**PASOS A SEGUIR**

1. Creación de carpeta (SO)
2. Crear archivo boot.asm que será el kernel, el grub lo mandará llamar y este a su vez mandará llamar al método kmain del archivo .C que dará instrucciones para ejecutarse

;;boot.asm

bits 32 ;nasm directive - 32 bit

section .text

;multiboot spec

align 4

dd 0x1BADB002 ;magic

dd 0x00 ;flags

dd - (0x1BADB002 + 0X00) ;checksum m+f+c tiene que ser cero

global start

extern kmain ;kmain se define en el archivo c

start:

cli

mov esp, stack\_space

call kmain

hlt

section .bss

stack\_space: resb 8192 ;8k para el stack

1. Crear archivo boot.c que tendrá el/los métodos que ejecutarán instrucciones

/\*

\* boot.c

\*/

void kmain(void) {

const char \*str = "Bienvenido al OS"; //Mensaje que se desplegará

char \*vidptr = (char\*)0xb8008; //la memoria de video inicia aqui

unsigned int i = 0;

unsigned int j = 0;

/\*Este ciclo limpia la pantalla

\*Hay 25 lineas cada una de 80 columnas, cada elemento toma 2 bytes\*/

while (j<80\*25\*2) {

/\*carácter en blanco\*/

vidptr[j]=' ';

vidptr[j+1]= 0x07;

j = j+2;

}

j = 0;

/\*Este ciclo escribe la cadena en la memoria de video\*/

while (str[j] != '\0'){

vidptr[i]=str[j];

vidptr[i+1]=0x07;

++j;

i=i+2;

}

return;

}

1. Abrir terminal y ejecutar código para compilar kernel  
   nasm –f elf32 boot.asm –o boot.o
2. Compilar archivo .C   
   gcc –m32 –c boot.c –o bootc.o
3. Crear archivo link.ld que tendrá las instrucciones que unirán los archivos del kernel y .C para crear un archivo objeto de salida. En este archivo se especifica la arquitectura de soporte  
   /\*

\* link.ld

\*/

OUTPUT\_FORMAT(elf32-i386)

ENTRY(start)

SECTIONS {

. = 0x100000;

.text : { \*(.text) }

.data : { \*(.data) }

.bss : { \*(.bss) }

}

1. Se compila el archivo link.ld estableciendo como parámetros que generará el archivo bajo arquitectura i386, tomará las instrucciones de compilación del archivo link.ld y unirá los archivos boot.o y bootc.o  
   ld –m elf\_i386 –T link.ld –o kernel boot.o bootc.o
2. Crear carpeta boot dentro de SO y meter en boot el archivo kernel generado en el paso anterior
3. Crear carpeta grub dentro de boot y dentro de grub crear el archivo grub.cfg, mismo que contendrá instrucciones para ejecutar el archivo “kernel” que contiene lo necesario para arrancar el sistema  
   menuentry "OS" {

multiboot /boot/kernel

boot

}

menuentry "Reiniciar" {

echo "Reiniciando..."

reboot

}

menuentry "Apagar"{

echo "Apagando..."

halt

}

1. Crear la carpeta “kernel” dentro de “SO” y la carpeta “boot” meterla en “kernel”
2. Abrir terminal en la carpeta SO para crear el iso bootable que reunirá la información del kernel y las directivas del GRUB. Se ejecuta el comando grub-mkrescue diciéndole que creará un objeto (-o) con nombre boot.iso y que sus archivos los encontrará en la carpeta “kernel”  
   grub-mkrescue –o boot.iso kernel
3. ¡Ya está listo! Ahora falta ejecutarlo. Se abre la terminal en la carpeta donde se generó el archivo boot.iso y se ejecuta:  
   qemu-system-i386 boot.iso  
   Esto abre la máquina virtual QEMU diciéndole que emule la arquitectura i386 cuando ejecute el sistema contenido en boot.iso

