COMO COMPILAR UN KERNEL MANUALMENTE

Antes que nada cabe aclarar que existen muchas paginas que explican esto pero siempre me resulto difícil entenderlas, me daba la impresión de que me decían “escribí esto y hace lo otro” pero no me explicaban en que consistía exactamente lo que estaba haciendo. Por lo tanto si tenia errores no tenia idea de como seguir adelante, solamente repetir el proceso y también el error.

Así que empiezo explicando brevemente de que estamos hablando cuando nombramos al kernel y de su compilación.
Si buscamos la pagina www.kernel.org vamos a ver que nos ofrecen varios archivos comprimidos para bajar que consisten en las fuentes de las diferentes lineas del kernel (2.X 3.X y 4.X)
Esto se debe a que las versiones del kernel se van numerando con tres números y va cambiando el tercero, o sea 4.19.1 - 4.19.2 – 4.19.3 a así hasta que los cambios por importancia justifican un cambio de versión y se pasa al 4.20 .
Ese archivo que bajamos se denomina fuente del kernel, y seria el kernel escrito en lenguaje entendible por un humano (que sepa programar en C por supuesto) y sus archivos de configuración.

Cuando se habla de “compilar el kernel” estamos hablando de convertir esos archivos en lenguaje maquina, que es el lenguaje que “entienden” las computadoras.

Ahora bien, ¿que es el kernel?
Es el conjunto de drivers necesarios para que una computadora sea utilizable, hablamos de el driver de video, el de audio, el de red pero también hablamos de los drivers de todos los componentes internos tanto de la placa madre (motherboard) y del procesador para que el grupo constituido por motherboard, procesador y RAM funcionen y se comuniquen entre si.
Tenemos muchísimos componentes menos visibles que necesitan drivers. Los diferentes caches del procesador, los coprocesadores matemáticos, el FSB de la RAM, el North Bridge y South Bridge del mother y muchos mas.

Ahora empieza la parte confusa pero trato de abordarla por partes a ver si se entiende mejor.
Ubiquémonos en que el archivo que bajamos lo debemos descomprimir (esta en tar.xz). Asique o lo descomprimimos con cualquier programa grafico o en un terminal hacemos tar -Jxvf archivo.tar.xz.
Esto nos va a dar como resultado un directorio llamado linux-versionkernel (linux-4.19.2)
Ese directorio hay que copiarlo al directorio /*usr*/src y a partir de ahora vamos a situarnos siempre dentro de ese directorio , asi que hacemos cd /*usr/*src/linux-4.19.X/ y no vamos a salir de ahí hasta que terminemos.

Primero: ¿Como seleccionamos que drivers queremos en el kernel? Muy fácil. El kernel cuando compila va leyendo un archivo que se llama .config y que tiene marcadas las opciones.

Cada driver tiene tres opciones
( ) vació → este driver no esta soportado en el kernel .
(M) modulo → el driver esta en el kernel pero como modulo, va a ser cargado después del arranque
(*) integrado → el driver es parte del kernel en todo momento desde el arranque.

Como ejemplo si nuestro disco esta formateado bajo ext4 el soporte de ext4 debe estar INTEGRADO al kernel, porque si lo compilamos como modulo el disco no va a poder ser leído en el arranque.

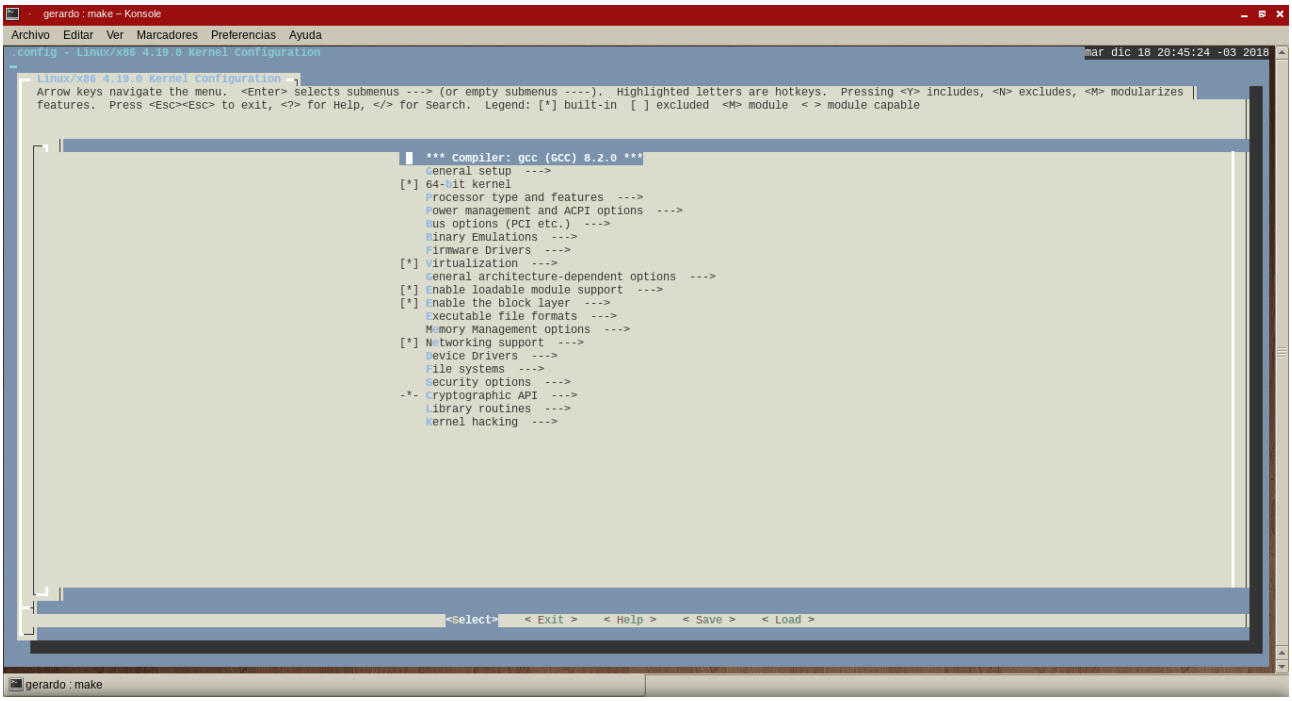
Ahora vamos de lleno a la configuración. Dijimos que el kernel cuando lo ponemos a compilar lo que hace es leer un archivo que se llama .config donde esta anotado que opción de la tres nombradas ( ) (M) o (*) seleccionamos para cada driver u opción.
Pero ese archivo si bien es perfectamente legible usa nombre clave y resulta muy difícil de editar a mano.
Para eso tenemos básicamente tres programas que vienen incluidos en la fuentes del kernel y que nos permiten movernos mas cómodamente por las opciones de compilación ademas de contar con un “help” que a grandes rasgos y en ingles nos aconseja que camino tomar si estamos indecisos con ese driver.

Entonces repasamos y ya nos ponemos a configurar.

Primero → bajamos las fuentes de www.kernel.org
Segundo → descomprimimos el archivo que bajamos para obtener un directorio linux-4.19 (ejemplo)
Tercero → abrimos un terminal y como root movemos la carpeta linux-4.18 a /*usr*/src/
Cuarto → no cambiamos a dicha carpeta con cd /*usr*/src/linux-4.19
Quinto → ya dentro de esa carpeta llamamos al programa elegido para empezar a configurar.

Los tres programas posibles para configurar son los siguientes (esto se teclea en el terminal siempre dentro de la carpeta /*usr*/src/linux-4.19) :

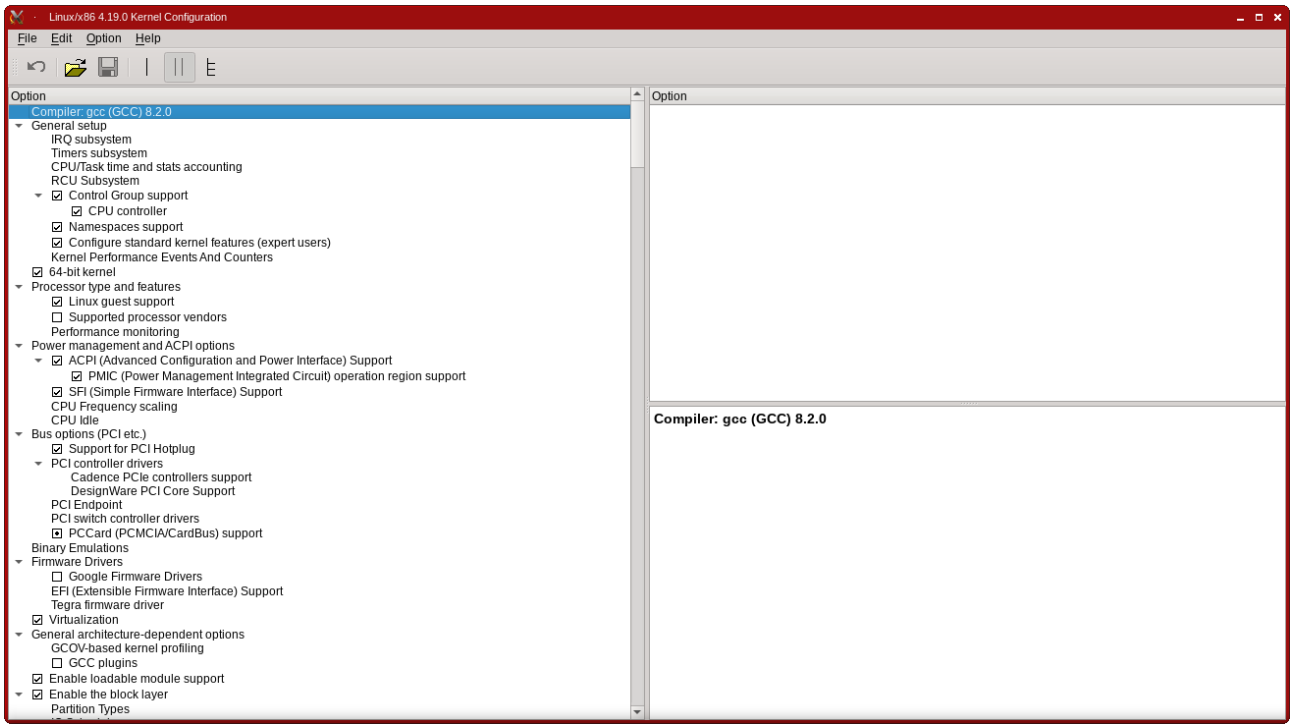
make menuconfig



Es un programa en modo texto y navegamos con los cursores , entramos en las secciones con Enter, salimos con Esc y cambiamos las opciones con la barra espaciadora.

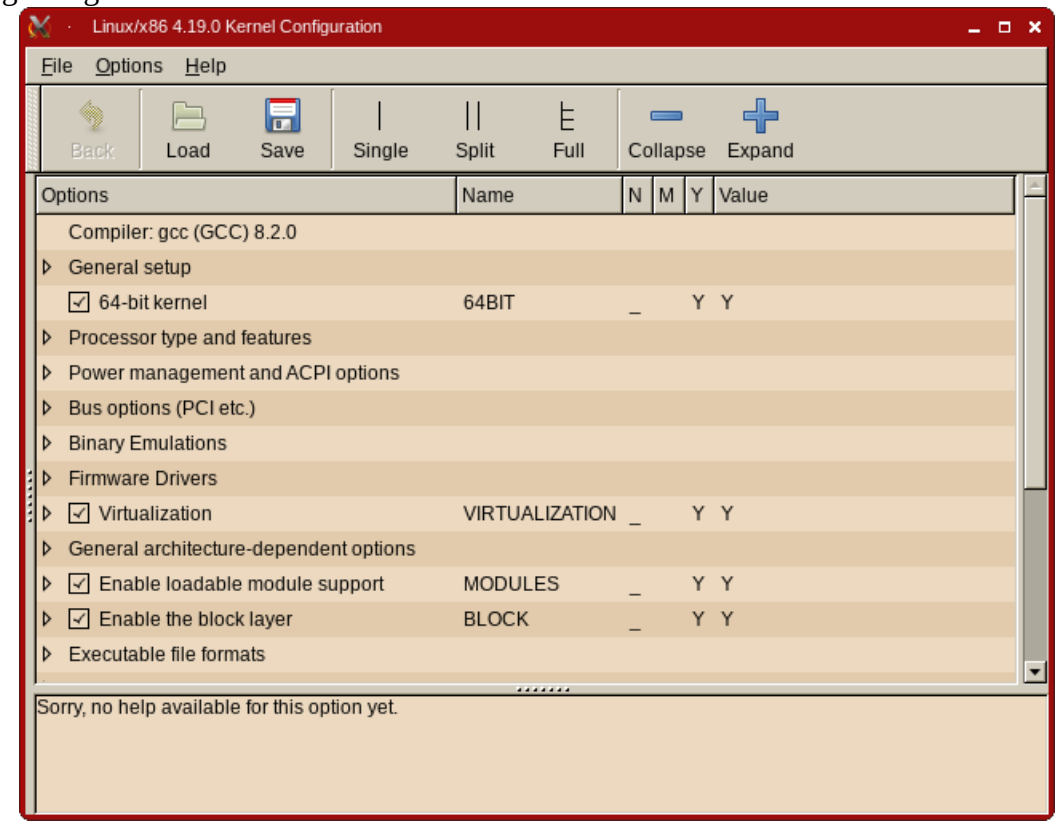
El segundo :

make xconfig



Este ya es en modo grafico y tenemos mouse y clic para hacer todo mas fácil.

Y el tercero:
make gconfig



Cambien en modo grafico y con mouse.

Nota : no se olviden que estamos en terminal modo root pero en entorno usuario, dependiendo de la distro puede ser que aparezca un aviso de Could not open xserver o algo asi.
Si les pasa eso se arregla en la terminal modo usuario haciendo xhost +x , así habilitamos a todos los usuarios a tener aplicaciones graficas en nuestro entorno.

A partir de acá todo es cuestión de seleccionar los dirvers que se ajusten a nuestra maquina o todos los que queramos, teniendo en cuenta que cuantos mas drivers metamos vamos a tener un kernel mas grande.

De todas maneras hay dos formas principales de configurar el kernel. Podemos incluir todos los drivers que existen en el kernel (kernel monolítico) o incluir lo fundamental para el arranque y el resto como módulos (kernel modular). O una tercera seria hacer un kernel mínimo, con los drivers exactos para nuestro hardware y nada mas. Piensen que el kernel trae drivers para cientos de placas de red y de sonido y nosotros tenemos una o dos.
Ni piensen que voy a entrar en la discusión de que tipo de kernel conviene mas porque esa discusión ya debe llevar 25 años y todavía no hay acuerdo.

Entonces acá es cuando tenemos que empezar a conocer nuestro hard para configurar el kernel de tal forma que lo soporte y obviamente cada uno de ustedes va a compilar su kernel a sus necesidades.

Pero si le paso tres opciones que son aconsejables incluir mas allá de obviamente los sistemas de archivos, si tienen windows incluyan soporte para ntfs y fat32 y por supuesto que soporte el sistema principal.

Estas opciones son dentro de General Setup

Kernel Compression Mode → elegir Lz4 (es mas rápido que los otros algoritmos de compresión)

Preemption Model → elegir Preemptible Kernel Low Latency) El kernel de baja latencia reacciona mas rápido.

Initial RAM Filesystem and RAM disk support → dejar un asterisco para incluir, en un momento explico. (1)

Compiler Optimizacion → elegir Performance Cambien es mas rápido

Ahora suponemos que ya seleccionamos todo lo que necesitamos y en cualquier caso salimos y elegimos la opción SAVE, de acuerdo, pero ¿que estamos haciendo con SAVE? Estamos generando el famoso archivo .config del cual se va a valer el compilador para saber que incluye y que no. Este archivo es muy importante porque si mañana bajamos un kernel nuevo, lo descomprimimos y lo copiamos a /*usr*/src/ y en lugar de hace todo esto le copiamos adentro de la carpeta el .config que ya tenemos y , en lugar de make menucofig, hacemos make oldconfig y ya tenemos todas las mismas opciones que venimos usando seleccionadas mucho mas rápido.

Entonces ya terminamos de seleccionar todo y salvamos, así que tenemos nuestro .config. Y ahora viene la compilación propiamente dicha.

Siempre en la misma terminal y la misma carpeta ejecutamos tres comando enlazados.
De esta forma :

make && make modules && make modules_install

make → compila todo lo que incluimos en el kernel
make modules → compila lo que elegimos compilar como módulos
make modules_install → instala los módulos, crea una carpeta /lib/modules/versión de kernel. Cada kernel que tengamos (podemos tener varios) va a buscar los módulos a su carpeta de módulos .

El comando make al compilar lo que hace es crear una imagen del kernel y un mapa del sistema.
La imagen esta en (siempre dentro de nuestra carpeta) en *arch/x86_64/boot/ y se llama bzImage.*
Y el mapa esta en el mismo directorio donde estamos y se llama System.map

Estos dos archivos son los que nos van a permitir bootear el kernel nuevo.
¿Como? Vamos al directorio /boot de nuestro sistema raíz y creamos una carpeta que podamos identificar con el nuevo kernel.

Ej : mkdir /*boot/Kernel419/*
cp ./System.map /boot/Kernel419/
cp arch/x86_64/boot/bzImage /boot/Kernel419/

Ahora lo único que falta es incluir el nuevo kernel en nuestro arranque. Yo uso lilo pero calculo que la mayoria usa Grub así que bajan el Grub-customizer y agregan la entrada nueva.

NUNCA BORREN LA ENTRADA VIEJA, si llega a fallar el kernel no tienen como entrar al sistema y eso nos pasa a todos todos los días.

(1) Initial RAM disk support. Vamos a suponer que compilamos el kernel , nos que dos perfecto, ni un error, todo lindisimo. Pero notamos que hay un modulo que necesitamos tenerlo disponible desde el arranque porque cuando el sistema termino de arrancar es tarde y el dispositivo ya fallo o no esta disponible, necesitaríamos alguna herramienta que nos permita de alguna forma incluir ese modulo al arranque . Esa herramienta es el Initial RAM Disk. Que no es ni mas ni menos que un disco virtual en la RAM que contiene los módulos que necesitamos (o todos) y los pone al servicio del kernel desde le principio.

Este disco se puede crear con :

```
mkinitramfs -o ~/tmp/initramfs-$(uname -r)
```

Esto crea el archivo initramfs-versiondekernel que también debemos copiar al directorio /boot y configurar la entrada de Grub para que lo use.

Bien, he tratado de dejar un poco mas claro de que se trata a la compilación del kernel, como todo en Linux se aprende a golpes y estropeando sistemas, así que mi recomendación es que tengan a mano un pendrive con SuperGrub y tomen practica es recuperar arranques, así prueban tranquilos. Seguro que esto tiene errores y estaría muy bien que me los señalen. No trate de dar una clase, no estoy a la altura, solamente trate de compartir algunos conocimientos por si les son útiles a alguien.

Gerardo Braica