Universidad Nacional Autónoma de México
Facultad de Ciencias

Administración de Sistemas UNIX/Linux
Semestre: 2025-2

Tarea

`Compilando el Kernel vol.3`

Profesor: Luis Enrique Serrano Gutiérrez
Ayudante: Erick Iram García Velasco

Equipo:
García Ponce José Camilo 319210536

Abril, 2024

1. Tabla comparativa (tamaño del kernel generado, compatibilidad, tiempo de compilación, estructura del archivo .config, etc) de las opciones:
    a. allnoconfig
    b. allmodconfig
    c. allyesconfig
    d. alldefconfig
    e. randconfig

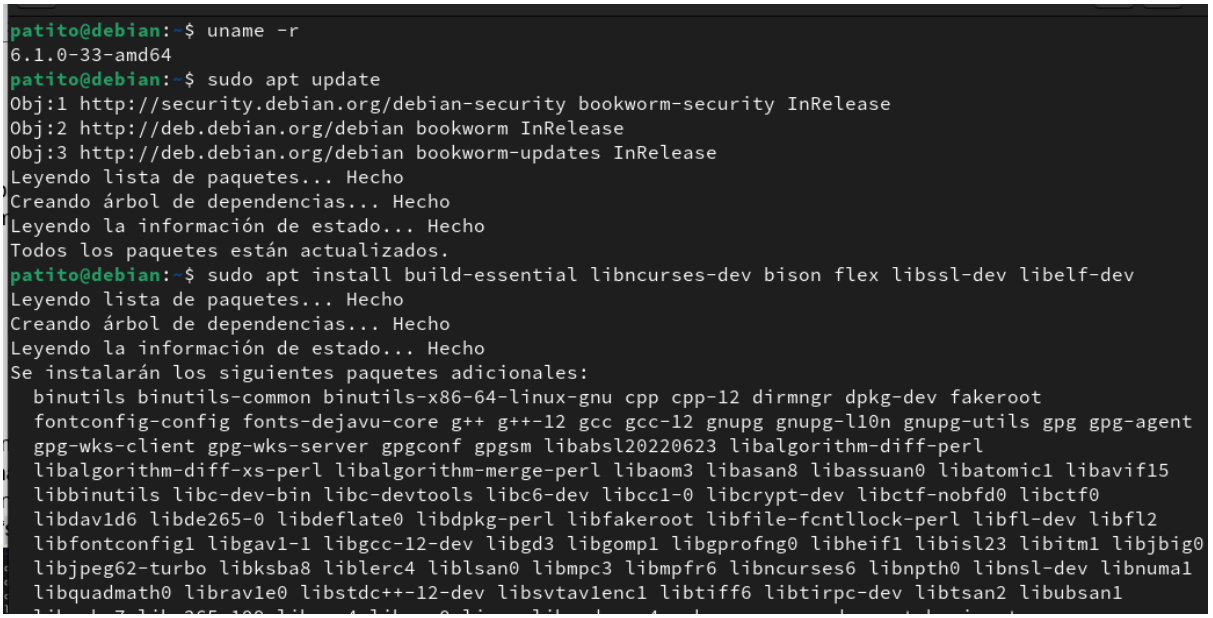
	Tamaño del kernel	Compatibilidad	Tiempo de compilación	Estructura del .config	Extra
allnoconfig	muy pequeño	muy baja (los sistemas pueden requerir más cosas de las que se marcan con allnoconfig)	muy rápido (ya que no tiene tantas cosas a compilar)	se genera un .config con todas las opciones posibles puestas en ‘n’ (que no estén habilitadas) o en algunos casos con ‘is not set’	generalmente los sistemas no logran arrancar con esta configuración, ya que faltan drivers básicos e init
allmodconfig	grande (pero usando varios módulos)	alto (aunque se requiere que el sistema soporte módulos)	lento (se deben compilar muchos módulos)	se genera un .config con todas las opciones posibles puestas en ‘m’ (que estén como módulos)	esta configuración es recomendable si se quiere experimentar ciertos drivers pero como módulos
allyesconfig	muy grande (es un kernel monolítico)	muy alto (aunque puede depender del hardware del equipo si es que no están comun)	muy lento (todo se debe compilar integrado)	se genera un .config con todas las opciones posibles puestas en ‘y’ (que estén habilitadas)	aunque el kernel es muy grande y tiene muchas opciones activada, si existen conflictos o errores puede no funcionar
alldefconfig	tamaño regular (depende de la versión del kernel y de la cantidad de opciones default que son ‘y’, ‘n’ y ‘m’)	alto (por lo general suele funcionar bien en equipos con hardware común)	regular (no es tan tardado pero puede tardar si la versión del kernel tiene muchas opciones con ‘y’ o ‘m’ en default)	se genera un .config con todas las opciones marcadas en su valor por default	es muy útil esta opción para usarla como punto de partida y luego agregar las opciones que se necesiten o se quieran
randconfig	el tamaño varía ya que	puede ser muy	el tamaño varía ya que	se genera un .config con	es util para gente que

	es al azar	compatible o nada depende del azar	es al azar	todas las opciones marcadas en un valor aleatorio	quiere probar la integridad de los kernels logrando detectar configuracion es extrañas que pueden causar fallos

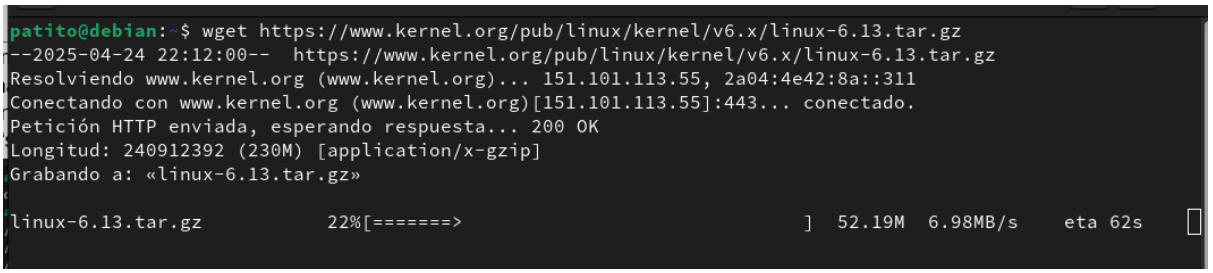
2. Compilación del kernel allnoconfig

Usando una máquina virtual con una instalación por default de Debian

Primero, instalamos lo necesario para compilar el kernel con los comandos: “sudo apt update” y “sudo apt install build-essential libncurses-dev bison flex libssl-dev libelf-dev”

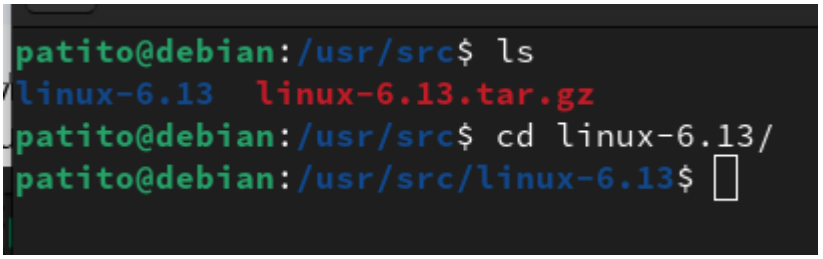


Descargamos el código del kernel (la versión 6.13) usando el comando “wget https://www.kernel.org/pub/linux/kernel/v6.x/linux-6.13.tar.gz”

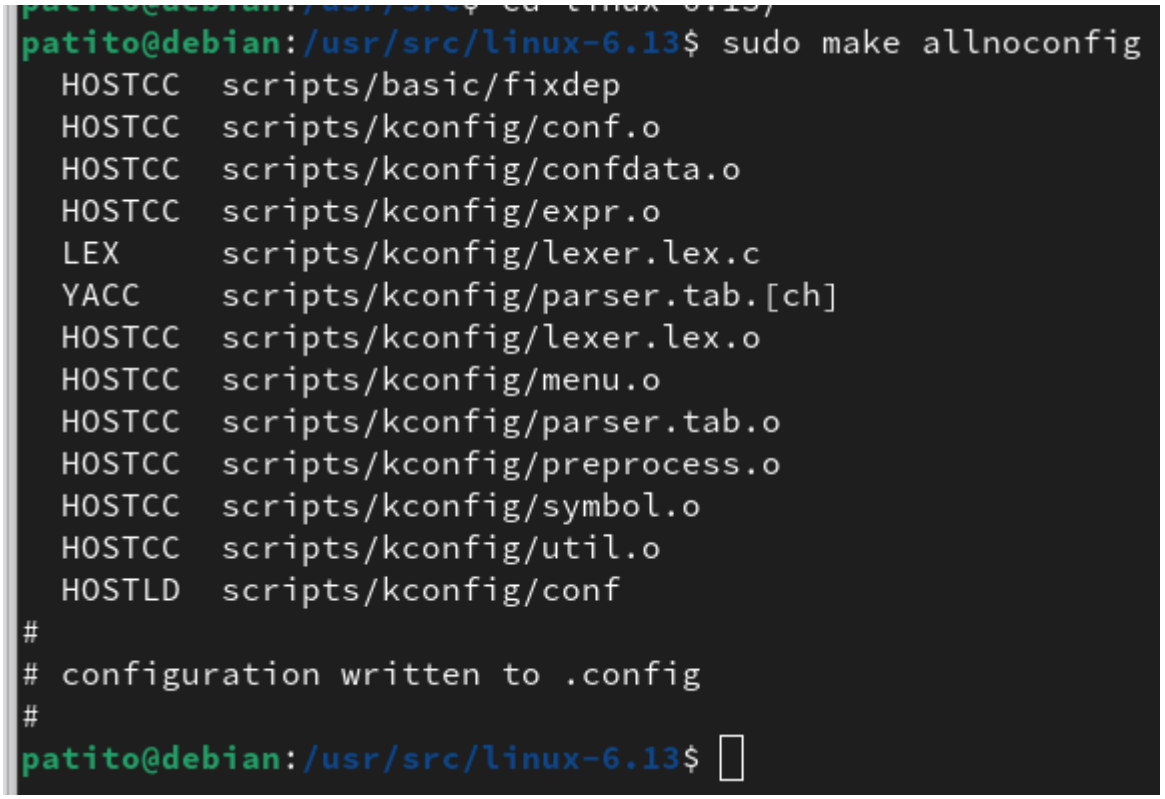


Movemos lo que descargamos a /usr/src con el comando “sudo mv linux-6.13.tar.gz /usr/src/”

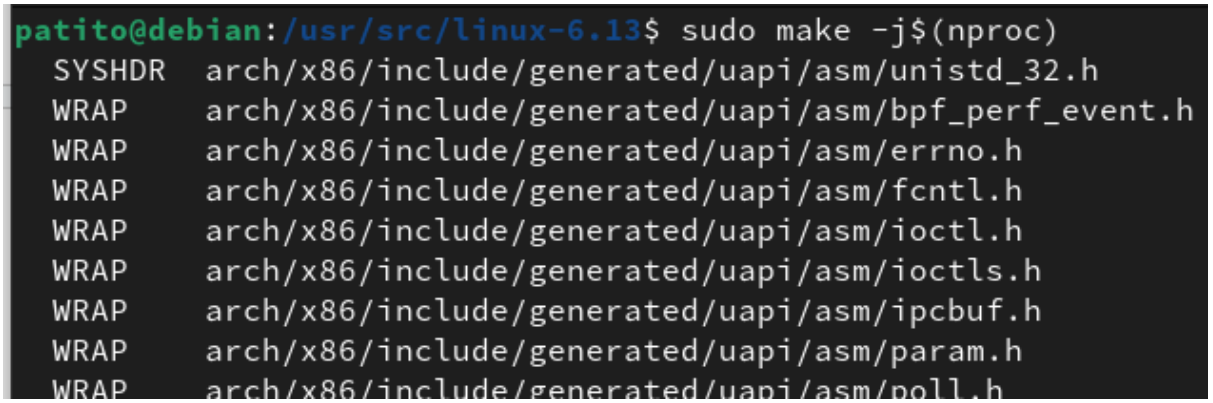
Después, extraemos lo que descargamos usando el comando: “sudo tar -xvf linux-6.13.tar.gz” y nos movemos a la nueva carpeta



Luego, usamos el comando “sudo make allnoconfig” para tener la nueva configuración



Posteriormente, compilamos el kernel y los módulos con el comando: “sudo make -j$(nproc)” (notemos que antes de ejecutar este comando ejecutamos el comando “sudo apt install bc”, como en las prácticas anteriores, y realizamos un snapshot)



```
  AS      arch/x86/boot/compressed/piggy.o
  LD      arch/x86/boot/compressed/vmlinux
  ZOFFSET arch/x86/boot/zoffset.h
  OBJCOPY arch/x86/boot/vmlinux.bin
  AS      arch/x86/boot/header.o
  LD      arch/x86/boot/setup.elf
  OBJCOPY arch/x86/boot/setup.bin
  BUILD   arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready  (#1)
patito@debian:/usr/src/linux-6.13$
```

Ahora tenemos que compilar los módulos con el comando “sudo make modules” y “sudo make modules_install”

```
patito@debian:/usr/src/linux-6.13$ sudo make modules
patito@debian:/usr/src/linux-6.13$ sudo make modules_install
  INSTALL /lib/modules/6.13.0/modules.builtin
  INSTALL /lib/modules/6.13.0/modules.builtin.modinfo
patito@debian:/usr/src/linux-6.13$
```

Instalamos el kernel con el comando “sudo make install”

```
patito@debian:/usr/src/linux-6.13$ sudo make install
  INSTALL /boot
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 6.13.0 /boot/vmlinuz-6.13.0
update-initramfs: Generating /boot/initrd.img-6.13.0
W: zstd compression (CONFIG_RD_ZSTD) not supported by kernel, using gzip
E: gzip compression (CONFIG_RD_GZIP) not supported by kernel
update-initramfs: failed for /boot/initrd.img-6.13.0 with 1.
run-parts: /etc/kernel/postinst.d/initramfs-tools exited with return code 1
make[1]: *** [arch/x86/Makefile:321: install] Error 1
make: *** [Makefile:251: __sub-make] Error 2
patito@debian:/usr/src/linux-6.13$
```

Actualizamos la configuración del grub con el comando ”sudo update-grub”

```
patito@debian:/usr/src/linux-6.13$ sudo update-grub
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-6.13.0
Found linux image: /boot/vmlinuz-6.1.0-33-amd64
Found initrd image: /boot/initrd.img-6.1.0-33-amd64
Found linux image: /boot/vmlinuz-6.1.0-22-amd64
Found initrd image: /boot/initrd.img-6.1.0-22-amd64
Warning: os-prober will not be executed to detect other bootable partitions.
Systems on them will not be added to the GRUB boot configuration.
Check GRUB_DISABLE_OS_PROBER documentation entry.
done
patito@debian:/usr/src/linux-6.13$
```

Después, reiniciamos el sistema con el comando ”sudo reboot”

Al reiniciar nos sale el error “Unable to mount root fs on “/dev/mapper/debian-vg-root” or unknown-block(0,0)”

```
Kernel panic - not syncing: VFS: Unable to mount root fs on "/dev/sda1" or unkno
wn-block(0,0)
CPU: 0 UID: 0 PID: 1 Comm: swapper Not tainted 6.13.0 #1
Hardware name: VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Plat
form, BIOS 6.00 11/12/2020
Call Trace:
 dump_stack_lvl+0x21/0x48
 dump_stack+0x12/0x18
 panic+0x264/0x274
 mount_root_generic+0x26e/0x2c4
 mount_root+0x17c/0x1f0
 prepare_namespace+0x38/0x264
 kernel_init_freeable+0x164/0x1ac
 ? rdinit_setup+0x30/0x30
 ? rest_init+0x74/0x74
 kernel_init+0x15/0x1bc
 ? schedule_tail+0x37/0x3c
 ret_from_fork+0x39/0x44
 ? rest_init+0x74/0x74
 ret_from_fork_asm+0x12/0x18
 entry_INT80_32+0xef/0xef
Kernel Offset: disabled
---[ end Kernel panic - not syncing: VFS: Unable to mount root fs on "/dev/sda1"
 or unknown-block(0,0) ]---
_
```

Entonces regresamos al snapshot que tenemos y ahora realizamos “sudo make menuconfig”

```
                    Linux/x86 6.13.0 Kernel Configuration
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted
letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc
to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module
capable

                General setup  --->
            [ ] 64-bit kernel
                Processor type and features  --->
            [ ] Mitigations for CPU vulnerabilities  ----
                Power management and ACPI options  --->
                Bus options (PCI etc.)  --->
                Binary Emulations  ----
            [ ] Virtualization  ----
                General architecture-dependent options  --->
            [ ] Enable loadable module support  ----
            -*- Enable the block layer  --->
                Executable file formats  --->
                Memory Management options  --->
            [ ] Networking support  ----
                Device Drivers  --->
                File systems  --->
                Security options  --->
            [ ] Cryptographic API  ----
                Library routines  --->
                Kernel hacking  --->
```

Ahora las configuraciones que agregaremos son las siguientes:
De Device Drivers habilitamos “SCSI device support” (solo dimos que si a esa opción y las demás opciones se marcaron por automático entonces las dejamos) y “SCSI disk support”, ya que por lo general VMware emula discos de tipo SCSI

```
[ ] RAID Transport Class
[*] SCSI device support
[*] legacy /proc/scsi/ support
    *** SCSI support type (disk, tape, CD-ROM) ***
[*] SCSI disk support
[ ] SCSI tape support
[ ] SCSI generic support
[*] /dev/bsg support (SG v4)
[ ] SCSI media changer support
[ ] Verbose SCSI error reporting (kernel size += 36K)
[ ] SCSI logging facility
[ ] Asynchronous SCSI scanning
    SCSI Transports  --->
[*] SCSI low-level drivers  --->
[ ] SCSI Device Handlers  ----
```

De File systems habilitamos “The Extended 4 (ext4) filesystem”, para que podamos soportar ese sistema de archivo

```
[ ] Validate filesystem parameter description
[ ] Second extended fs support (DEPRECATED)
[ ] The Extended 3 (ext3) filesystem
[*] The Extended 4 (ext4) filesystem
[*]   Use ext4 for ext2 file systems (NEW)
[ ]   Ext4 POSIX Access Control Lists (NEW)
[ ]   Ext4 Security Labels (NEW)
[ ]   Ext4 debugging support (NEW)
[ ] JBD2 (ext4) debugging support (NEW)
```

Y de General setup habilitamos “Initial RAM filesystem and RAM disk (initramfs/initrd) support” y todos los “Support initial ramdisk/ramfs compressed using …”, para poder cargar módulos y scripts esenciales desde initrd

```
[*] Initial RAM filesystem and RAM disk (initramfs/initrd) support
()    Initramfs source file(s)
[*]   Support initial ramdisk/ramfs compressed using gzip
[*]   Support initial ramdisk/ramfs compressed using bzip2
[*]   Support initial ramdisk/ramfs compressed using LZMA
[*]   Support initial ramdisk/ramfs compressed using XZ
[*]   Support initial ramdisk/ramfs compressed using LZO
[*]   Support initial ramdisk/ramfs compressed using LZ4
[*]   Support initial ramdisk/ramfs compressed using ZSTD
```

Con esta nueva configuración volvemos a hacer los pasos de “sudo make -j$(nproc)”

```
patito@debian:/usr/src/linux-6.13$ sudo make -j$(nproc)
  SYNC    include/config/auto.conf.cmd
  HOSTCC  scripts/kconfig/conf.o
  HOSTLD  scripts/kconfig/conf
  SYSHDR  arch/x86/include/generated/uapi/asm/unistd_32.h
  SYSHDR  arch/x86/include/generated/uapi/asm/unistd_64.h
  SYSHDR  arch/x86/include/generated/uapi/asm/unistd_x32.h
  SYSTBL  arch/x86/include/generated/asm/syscalls_32.h
  HOSTCC  arch/x86/tools/relocs_32.o
```

Después hacemos “sudo make modules”, “sudo make modules_install” y “sudo make install”

```
patito@debian:/usr/src/linux-6.13$ sudo make modules
patito@debian:/usr/src/linux-6.13$ sudo make modules_install
  INSTALL /lib/modules/6.13.0/modules.builtin
  INSTALL /lib/modules/6.13.0/modules.builtin.modinfo
patito@debian:/usr/src/linux-6.13$ sudo make install
  INSTALL /boot
run-parts: executing /etc/kernel/postinst.d/initramfs-tools 6.13.0 /boot/vmlinuz-6.13.0
update-initramfs: Generating /boot/initrd.img-6.13.0
depmod: WARNING: could not open modules.order at /lib/modules/6.13.0: No such file or directory
depmod: WARNING: could not open modules.order at /var/tmp/mkinitramfs_y9usP3/lib/modules/6.13.0: No
such file or directory
run-parts: executing /etc/kernel/postinst.d/zz-update-grub 6.13.0 /boot/vmlinuz-6.13.0
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-6.13.0
Found initrd image: /boot/initrd.img-6.13.0
Found linux image: /boot/vmlinuz-6.13.0.old
Found initrd image: /boot/initrd.img-6.13.0
Found linux image: /boot/vmlinuz-6.1.0-33-amd64
Found initrd image: /boot/initrd.img-6.1.0-33-amd64
Found linux image: /boot/vmlinuz-6.1.0-22-amd64
Found initrd image: /boot/initrd.img-6.1.0-22-amd64
Warning: os-prober will not be executed to detect other bootable partitions.
Systems on them will not be added to the GRUB boot configuration.
Check GRUB_DISABLE_OS_PROBER documentation entry.
done
patito@debian:/usr/src/linux-6.13$
```

Actualizamos el grub con “sudo update-grub” y por último hacemos “sudo reboot”, para ver si ahora funciona

```
patito@debian:/usr/src/linux-6.13$ sudo update-grub
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-6.13.0
Found initrd image: /boot/initrd.img-6.13.0
Found linux image: /boot/vmlinuz-6.13.0.old
Found initrd image: /boot/initrd.img-6.13.0
Found linux image: /boot/vmlinuz-6.1.0-33-amd64
Found initrd image: /boot/initrd.img-6.1.0-33-amd64
Found linux image: /boot/vmlinuz-6.1.0-22-amd64
Found initrd image: /boot/initrd.img-6.1.0-22-amd64
Warning: os-prober will not be executed to detect other bootable partitions.
Systems on them will not be added to the GRUB boot configuration.
Check GRUB_DISABLE_OS_PROBER documentation entry.
done
patito@debian:/usr/src/linux-6.13$
```

Luego de reiniciar obtenemos el error de No working init found

```
Failed to execute /init (error -2)
Kernel panic - not syncing: No working init found.  Try passing init= option to
kernel. See Linux Documentation/admin-guide/init.rst for guidance.
CPU: 0 UID: 0 PID: 1 Comm: swapper Not tainted 6.13.0 #2
Hardware name: VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Plat
form, BIOS 6.00 11/12/2020
Call Trace:
 dump_stack_lvl+0x21/0x48
 dump_stack+0x12/0x18
 panic+0x264/0x274
 ? rest_init+0x74/0x74
 kernel_init+0x157/0x1bc
 ret_from_fork+0x39/0x44
 ? rest_init+0x74/0x74
 ret_from_fork_asm+0x12/0x18
 entry_INT80_32+0xef/0xef
Kernel Offset: disabled
---[ end Kernel panic - not syncing: No working init found.  Try passing init= o
ption to kernel. See Linux Documentation/admin-guide/init.rst for guidance. ]---
```

Por lo tanto reiniciamos la máquina virtual y vamos a agregar más opciones

Activaremos las opciones “CONFIG_MODULES=y”, esto para que el kernel soporte módulos cargables

```
CONFIG_RT_MUTEXES=y
CONFIG_MODULES=y
CONFIG_BLOCK=y
# CONFIG_BLOCK_LEGA
```

También habilitaremos “CONFIG_DEVTMPFS=y”, “CONFIG_DEVTMPFS_MOUNT=y” y “CONFIG_TMPFS=y”, para montar /dev automáticamente y para algunas cosas de /run

```
# CONFIG_UEVENT_HELPER
CONFIG_DEVTMPFS=y
# CONFIG_STANDALONE is
```

```
CONFIG_DEVTMPFS=y
CONFIG_DEVTMPFS_MOUNT=y
```

```
CONFIG_SYSFS=y
CONFIG_TMPFS=y
# CONFIG_HUGETLBFS is n
```

Luego volvemos a hacer “yes "n" | sudo make -j$(nproc)”, “sudo make modules”, “sudo make modules_install”, “sudo make install”, “sudo update-initramfs -c -k 6.13.0”, “sudo update-grub” y reiniciamos para ver si ahora funciona

```
patito@debian:/usr/src/linux-6.13$ yes "n" | sudo make -j$(nproc)
  SYNC    include/config/auto.conf.cmd
*
* Restart config...
*
*
* Generic Driver Options
*
Support for uevent helper (UEVENT_HELPER) [N/y/?] n
Maintain a devtmpfs filesystem to mount at /dev (DEVTMPFS) [Y/n/?] y
  Automount devtmpfs at /dev, after the kernel mounted the rootfs (DEVTMPFS_MOUNT) [Y/n/?] y
  Use nosuid,noexec mount options on devtmpfs (DEVTMPFS_SAFE) [N/y/?] (NEW) n
```

Al reiniciar notamos que ahora nos sale un error que dice “Failed to execute /init (error -2) request_module: modprobe binfmt-464c cannot be processed, kmod busy with 50 threads for more than 5 seconds now”

```
Failed to execute /init (error -2)
request_module: modprobe binfmt-464c cannot be processed, kmod busy with 50 thre
ads for more than 5 seconds now
Kernel panic - not syncing: No working init found.  Try passing init= option to
kernel. See Linux Documentation/admin-guide/init.rst for guidance.
CPU: 0 UID: 0 PID: 1 Comm: swapper Not tainted 6.13.0 #3
Hardware name: VMware, Inc. VMware Virtual Platform/440BX Desktop Reference Plat
form, BIOS 6.00 11/12/2020
Call Trace:
 dump_stack_lvl+0x21/0x48
 dump_stack+0x12/0x18
 panic+0x264/0x274
 ? rest_init+0x74/0x74
 kernel_init+0x15f/0x1bc
 ret_from_fork+0x39/0x44
 ? rest_init+0x74/0x74
 ret_from_fork_asm+0x12/0x18
 entry_INT80_32+0xef/0xef
Kernel Offset: disabled
---[ end Kernel panic - not syncing: No working init found.  Try passing init= o
ption to kernel. See Linux Documentation/admin-guide/init.rst for guidance. ]---
```

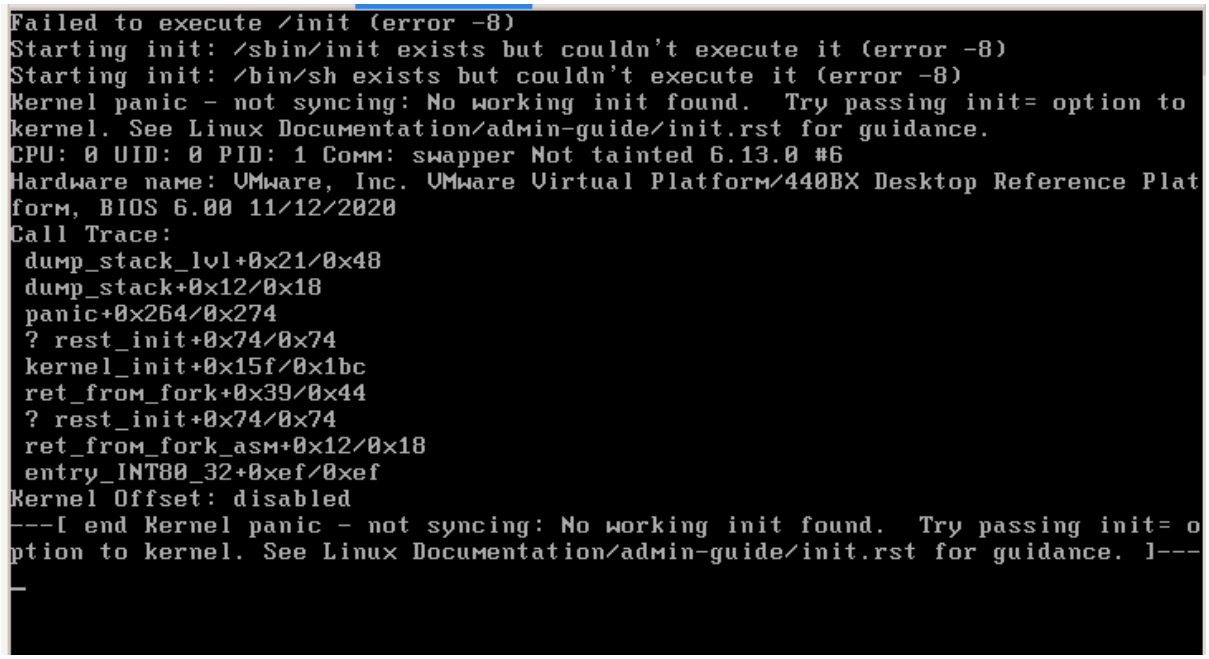
Entonces volvemos a prender la maquina virtual y agregamos otras opciones, “CONFIG_TMPFS=y” ”CONFIG_BINFMT_MISC=y”, “CONFIG_BINFMT_ELF=y”, “CONFIG_BINFMT_SCRIPT=y” y “CONFIG_CORE_DUMP_DEFAULT_ELF_HEADERS=y”, para el soporte de ejecutables binarios como binfmt_misc y para el soporte de tmpfs (sistema de archivos para archivos en memoria virtual)

```
CONFIG_SYSFS=y
CONFIG_TMPFS=y
```

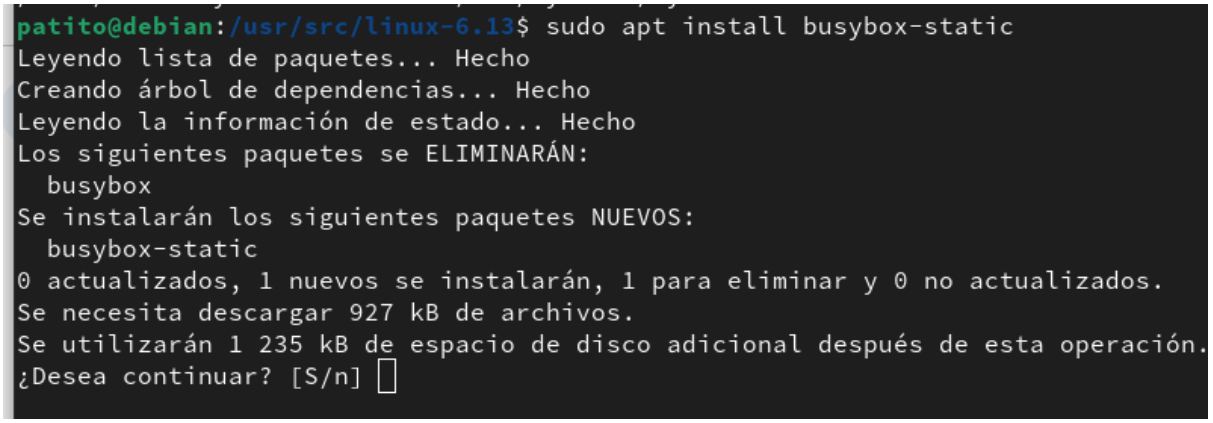
```
#
CONFIG_BINFMT_ELF=y
CONFIG_ELFCORE=y
CONFIG_CORE_DUMP_DEFAULT_ELF_HEADERS=y
CONFIG_BINFMT_SCRIPT=y
CONFIG_BINFMT_MISC=y
CONFIG_COREDUMP=y
# end of Executable file formats
```

Y volvemos a compilar y todo ese proceso para ver si ahora funciona, pero volvió a fallar y ahora con este error “Failed to execute /init (error -8) Starting init: /sbin/init exists but couldn't

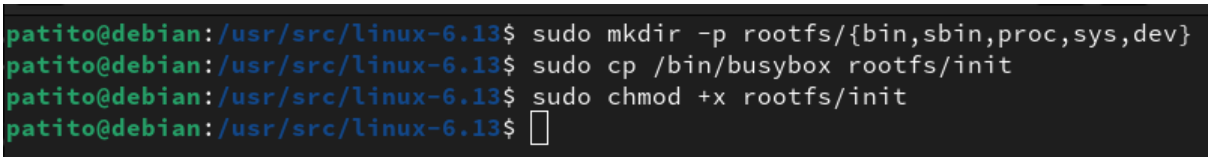
execute it (error -8) Starting /bin/sh exists but couldn't execute it (error -8) Kernel panic - not syncing: No working init found”



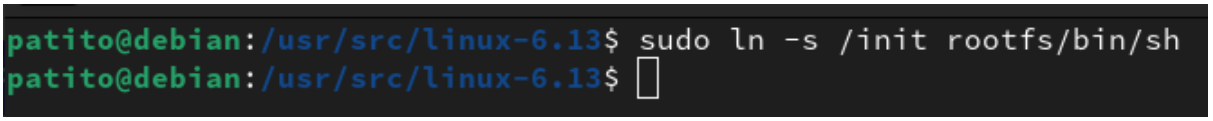
Como no volvió a funcionar entonces vamos a intentar usar un init estático con BusyBox, para esto primero instalamos BusyBox estático con “sudo apt install busybox-static”



Luego creamos un directoria raíz simple con los comandos “sudo mkdir -p rootfs/{bin,sbin,proc,sys,dev}”, “sudo cp /bin/busybox rootfs/init” y “sudo chmod +x rootfs/init”



Despues usamos “sudo ln -s /init rootfs/bin/sh” para agregar enlaces de comandos comunes



Posteriormente empaquetamos con “cd rootfs” y ”sudo sh -c "find . | cpio -o -H newc | gzip > ../initramfs-busybox.cpio.gz”

```
patito@debian:/usr/src/linux-6.13$ cd rootfs/
patito@debian:/usr/src/linux-6.13/rootfs$ sudo sh -c "find . | cpio -o -H newc |
 gzip > ../initramfs-busybox.cpio.gz"
3874 bloques
patito@debian:/usr/src/linux-6.13/rootfs$
```

Después intentamos probar el kernel con QEMU por lo tanto lo instalamos primero

```
patito@debian:/usr/src/linux-6.13$ sudo apt install qemu-system-x86
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes adicionales:
```

Y para probar el kernel usamos el comando “qemu-system-x86_64 -kernel arch/x86/boot/bzImage -initrd initramfs-busybox.cpio.gz -nographic -append "console=ttyS0 earlyprintk=serial,ttyS0,115200”

```
Freeing unused kernel image (initmem) memory: 300K
Write protecting kernel text and read-only data: 3548k
Run /init as init process
Failed to execute /init (error -8)
Run /sbin/init as init process
Run /etc/init as init process
Run /bin/init as init process
Run /bin/sh as init process
Starting init: /bin/sh exists but couldn't execute it (error -8)
Kernel panic - not syncing: No working init found.  Try passing init= option to.
CPU: 0 UID: 0 PID: 1 Comm: swapper Not tainted 6.13.0 #6
Hardware name: QEMU Standard PC (i440FX + PIIX, 1996), BIOS 1.16.2-debian-1.16.4
Call Trace:
 dump_stack_lvl+0x21/0x48
 dump_stack+0x12/0x18
 panic+0x264/0x274
 ? rest_init+0x74/0x74
 kernel_init+0x15f/0x1bc
 ret_from_fork+0x39/0x44
 ? rest_init+0x74/0x74
 ret_from_fork_asm+0x12/0x18
 entry_INT80_32+0xef/0xef
Kernel Offset: disabled
---[ end Kernel panic - not syncing: No working init found.  Try passing init= -
```

Pero nos volvió a salir el mismo error, por lo que no logramos avanzar más, debido al paro no pudimos conseguir tanto ayuda o guia para realizar la tarea, entonces la idea que nos ocurre es hacer un diff de nuestro .config actual y del .config generado por oldconfig y agregar todas las opciones que nos falta para lograr que el kernel pueda funcionar
Primero copiamos el .config actual con “sudo cp .config config_original”

```
patito@debian:/usr/src/linux-6.13$ sudo cp .config config_original
patito@debian:/usr/src/linux-6.13$
```

Después creamos el nuevo .config con “yes "" | sudo make oldconfig'”

```
patito@debian:/usr/src/linux-6.13$ yes "" | sudo make oldconfig
.config:887:warning: symbol value '0' invalid for BASE_SMALL
.config:6364:warning: symbol value 'm' invalid for FB_BACKLIGHT
.config:8020:warning: symbol value 'm' invalid for VFIO_VIRQFD
.config:9250:warning: symbol value 'm' invalid for ANDROID_BINDER_IPC
.config:9410:warning: symbol value 'm' invalid for FSCACHE
*
* Restart config...
*
*
* General setup
*
Compile also drivers which will not load (COMPILE_TEST) [N/y/?] n
Compile the kernel with warnings as errors (WERROR) [N/y/?] n
Local version - append to kernel release (LOCALVERSION) []
```

Luego copiamos esta configuración con “sudo cp .config config_nueva”

```
patito@debian:/usr/src/linux-6.13$ sudo cp .config config_nueva
patito@debian:/usr/src/linux-6.13$
```

Posteriormente hacer el diff de las dos configuraciones para ver que opciones nos faltan con “sudo diff -u config_original config_nueva > ~/config_diferencia.patch”, el diferencial se adjunta por separado

```
patito@debian:/usr/src/linux-6.13$ sudo diff -u config_original config_nueva > ~
/config_diferencia.patch
patito@debian:/usr/src/linux-6.13$ cd
patito@debian:~$ ls
config_diferencia.patch
patito@debian:~$
```

Ahora modificamos el .config que creamos con allnoconfig para que se vea igual que el creado con defconfig y compilamos el kernel con la nueva configuración esperando que ahora si sirve el kernel, realizamos todo el proceso de compilar el kernel, cargar módulos, actualizar el grub y eso, y reiniciamos la máquina virtual

```
patito@debian:~$ uname -r
6.13.0
patito@debian:~$
```

Al final el kernel logró funcionar

También se intento usar un init personalizado usando BusyBox estático pero no funcionó del todo

```
BusyBox v1.35.0 (Debian 1:1.35.0-4+b3) built-in shell (ash)
Enter 'help' for a list of built-in commands.

(initramfs) uname -r
6.13.0
(initramfs)
```

Esta tarea estuvo muy pesada y difícil

3. Preguntas
    a. ¿Qué es Kconfig?
    Es el sistema de configuración que usa el kernel de Linux, es utilizado para que podemos elegir cuales características y opciones queremos que incluya nuestro kernel antes de compilarlo. Usa archivos para poder organizar la jerarquía de las opciones disponibles. Además de ser usado en el kernel de Linux también lo usan Zephyr y coreboot. Los usuarios podemos interactuar con Kconfig mediante interfaces con menús, como make menuconfig.

    b. ¿Cuál es la diferencia entre make defconfig y make alldefconfig?
    Aunque ambos sirven para generar un .config, se diferencian en cómo lo hacen.
    defconfig genera el .config usando una configuración predeterminada la cual varía dependiendo de la arquitectura del sistema (como x86, ARM, PowerPC, etc).
    alldefconfig genera el .config poniendo todas las opciones en su valor por default.

    c. ¿Cómo afecta allnoconfig la compatibilidad con hardware?
    Como todas las opciones no son habilitadas en el .config que genera allnoconfig entonces tendremos que faltaran algunas muy importantes como drivers, sin los cuales el kernel y el sistema no se van a poder comunicar con los dispositivos de hardware y por lo tanto causando que el sistema no funcione correctamente o no arranque el equipo con el hardware que tiene. Aparte de drivers puede pasar que al kernel le falte características necesarias para poder usar los dispositivos de hardware, causando que no funcione correctamente.

    d. ¿Por qué randconfig puede generar un kernel inestable?
    Como randconfig genera un .config donde a las opciones les pone un valor aleatorio entonces no tenemos la certeza de que todas las opciones necesarias para que funcione el sistema estén actividad, por ejemplo puede faltar el sistema de archivos que usa el equipo o no tener cosas relacionadas a drivers importantes o al init. Además como las opciones se seleccionan aleatoriamente es posible que se seleccionen algunas opciones que no sean tan compatibles o no funcionen tan bien con otras opciones seleccionadas, lo cual también puede causar problemas mientras el sistema esté funcionando.