Compilación del kernel Linux

Explicación de práctica 1

Sistemas Operativos

Facultad de Informática
Universidad Nacional de La Plata

2025

Algunos objetivos de la práctica

- Comprender conceptos del kernel de Linux y de los kernels en general.
- Comprender los pasos de compilación del kernel de Linux.
- Poder personalizar y compilar un kernel.
- Tener un entorno listo para compilar el kernel de Linux para la siguiente práctica.

Agenda

Contenido:

- El kernel de Linux
- Herramientas / Compilación
- Compilación
- Apéndice
  - Debian way

El kernel de Linux

- Es software libre (excepto algunos blobs binarios).
- Podemos compilar versiones personalizadas.
- Escrito en:
  - Lenguaje C mayormente.
  - Assembler: Instrucciones especiales y bajo nivel.
  - Rust: A modo experimental para implementar módulos[1, 2].

1. https://docs.kernel.org/rust/
2. https://wusyong.github.io/posts/rust-kernel-module-01/

Herramientas necesarias

- GCC el compilador de C del proyecto GNU
- as y ld (assembler y linker de GNU)
- GNU Make: Organiza la compilación para poder ejecutarla con unos pocos comandos.
- diff/patch opcionalmente se pueden usar para crear o aplicar “parches” o actualizaciones al código.

Compilación

- Lenguaje C:
  - gcc -c hello.c -o hello.o # generar archivo objeto
  - gcc hello.o -o hello # linkear, genera ejecutable
- Assembler:
  - as -o hello.o hello.S # generar archivo objeto
  - ld -o hello hello.o # linkear, genera ejecutable
- Rust:
  - rustc --emit=obj -o hello.o hello.rs # generar archivo objeto
  - gcc hello.o -o hello -no-pie -lc # linkear, genera ejecutable
- Compilar múltiples archivos fuentes C y ASM:
  - Por cada fuente generar un programa objeto (con gcc, rust o as).
  - Linkear todos los programas en un solo ejecutable con ld o con gcc.

make al rescate

- GNU Make: Herramienta.
- Genera ejecutables u otros archivos a partir de archivos fuente.
- Si los archivos fuentes no cambiaron no hace nada.
- Fundamental para compilar código C pero se usa con otros fines.
- Ejemplos:
  - C a binarios ejecutables
  - LaTeX o Markdown a PDF
  - Assembler a binarios ejecutables
  - Paquete Python con varios fuentes a ejecutable en un solo archivo.

Fuente: https://www.gnu.org/software/make/manual/make.html

GNU Make

- Lee instrucciones de un archivo llamado Makefile (case sensitive)
- Makefile:
  - Rules que definen cómo generar un archivo (target)

```
target: prerequisites
    recipe
```

- Un target puede depender de prerequisitos (otros targets)
- Se pueden definir targets que no generen archivos, por ejemplo para mostrar una ayuda o invocar a múltiples targets con un comando.

```
all: target1 target2
target1: …
target2: …
.PHONY: all
```

GNU Make

- Variables

```
CFLAGS = -Wall -std=c11
SRC := $(wildcard src/*.c)

hello: $(SRC)
   gcc $(CFLAGS) $(SRC) -o hello
```

- Recursively expanded variable assignment / Simply expanded variable assignment.
- Existen reglas explícitas, wildcards y variables mágicas que permiten automatizar aún más.

GNU Make

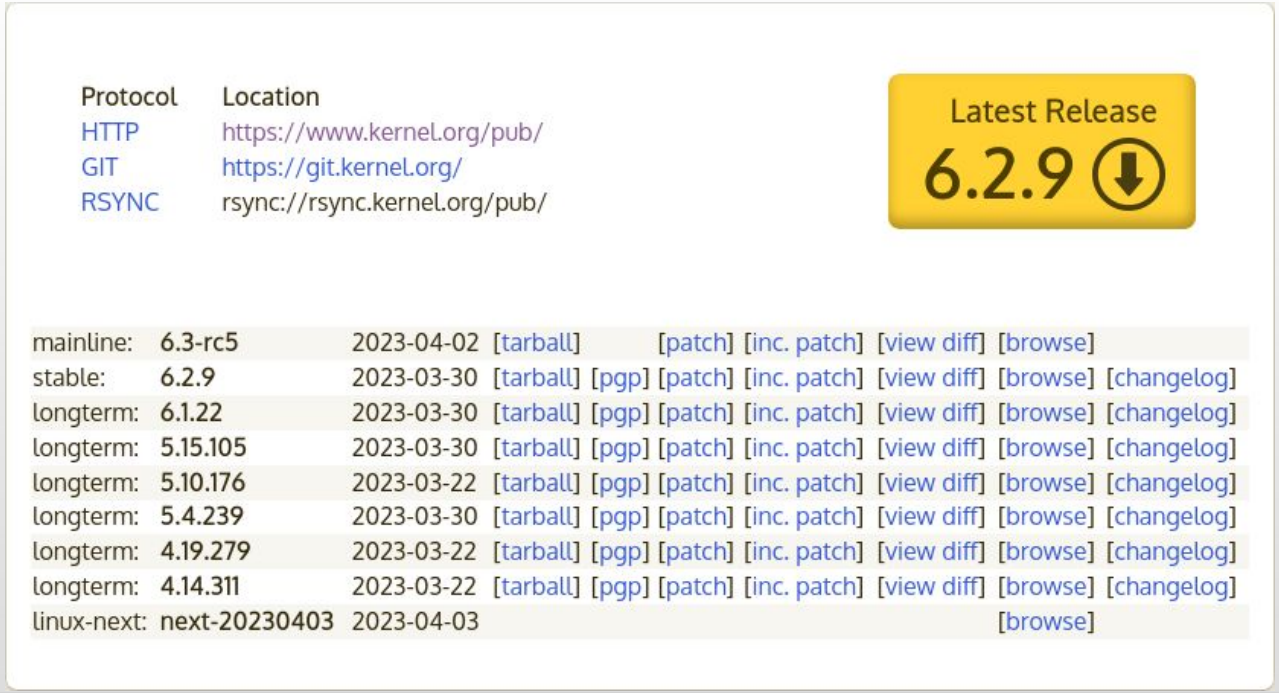
- Opciones:
  - make target # forma típica
  - make # invoca al primer target del Makefile (habitualmente “all”)
  - make -j 8 target # 8 jobs concurrentes
  - make -f other.makefile target # usa un archivo alternativo en lugar de Makefile
- Es buena práctica definir los targets:
  - all # poner al principio, debería construir el proyecto completo
  - help # mostrar ayuda de como usar el Makefile
  - clean # borra los binarios y archivos intermedios generados
  - Ninguno es un nombre de archivo así que marcarlos como phony:
    - .PHONY: all help clean

Proceso de compilación de Linux

1. Obtener el código fuente.
2. Preparar el árbol de archivos del código fuente.
3. Configurar el kernel
4. Construir el kernel a partir del código fuente e instalar los módulos.
5. Reubicar el kernel.
6. Creación del initramfs
7. Configurar y ejecutar el gestor de arranque (GRUB en general).
8. Reiniciar el sistema y probar el nuevo kernel.

Obtener el código fuente

http://www.kernel.org



Proceso de compilación de Linux

Para descargarlo desde la terminal:

```
$ cd /usr/src
$ sudo wget https://kernel.org/pub/linux/kernel/v6.x/linux-6.13.tar.xz
```

Opcional: Por cuestiones de seguridad, opcionalmente se puede descargar un archivo con una firma criptográfica del kernel descargado. Esta firma nos permite comprobar que el archivo que descargamos es exactamente el mismo que fue subido, y que no hubo modificaciones maliciosas de por medio.

```
$ gpg --locate-keys torvalds@kernel.org gregkh@kernel.org
$ wget https://kernel.org/pub/linux/kernel/v6.x/linux-6.13.tar.sign
$ xz -cd linux-6.13.tar.xz | gpg --verify linux-6.13.tar.sign -
gpg: Firmado el lun 08 ene 2024 02:47:49 -03
gpg:                usando RSA clave 647F28654894E3BD457199BE38DBBDC86092693E
gpg: Firma correcta de "Greg Kroah-Hartman <gregkh@kernel.org>" [desconocido]
gpg: ATENCIÓN: ¡Esta clave no está certificada por una firma de confianza!
gpg:          No hay indicios de que la firma pertenezca al propietario.
Huellas dactilares de la clave primaria: 647F 2865 4894 E3BD 4571  99BE 38DB BDC8
6092 693E
```

Preparar el árbol de archivos

Por convención el código fuente del kernel se guarda en /usr/src. Sin embargo, como dicho directorio generalmente no tiene permisos de escritura para usuarios no privilegiados, el archivo se puede descomprimir en un directorio donde tengamos permisos, como el $HOME del usuario actual.

```
$ mkdir $HOME/kernel
$ cd $HOME/kernel
$ tar xvf /usr/src/linux-6.13.tar.xz
```

Generalmente se crea un enlace simbólico llamado linux apuntando al directorio del código fuente que actualmente se está configurando

```
$ ln -s /usr/src/linux-6.13 /usr/src/linux
```

Configurar el kernel

El kernel Linux se configura mediante el archivo .config. Este archivo, que reside en el directorio de código fuente del kernel, contiene las instrucciones de qué es lo que el kernel debe compilar. También se puede ver la configuración de los kernels instalados en /boot:

```
$ cat /boot/config-$(uname -r) | tail -n5
CONFIG UCS2 STRING=y
CONFIG FONT SUPPORT=y
# CONFIG FONTS is not set
CONFIG FONT 8x8=y
CONFIG FONT 8x16=y
```

Existen tres interfaces que permiten generar este archivo:

- make config: modo texto y secuencial. Tedioso.
- make xconfig: interfaz gráfica utilizando un sistema de ventanas.No todos los sistemas tienen instalado X.
- make menuconfig: este modo utiliza ncurses, una librería que permite generar una interfaz con paneles desde la terminal. Generalmente el más utilizado.

Configurar el kernel

Las herramientas mencionadas permiten:

- Crear un archivo .config con las directivas de compilación
- Configurar un kernel desde cero es una tarea tediosa y propensa a errores (kernels que no arranquen). Estas herramientas automatizan el proceso por nosotros.
- Consejos
  - Lo ideal es ir manteniendo el .config para no tener que configurar todo de cero
  - Cada nueva versión, puede valerse de un .config anterior.
  - Por convención, es recomendable almacenar en el directorio /boot la imagen compilada del kernel junto con su .config.

Configurar el kernel (Módulos)

¿Qué es un módulo del Kernel?

- Un fragmento de código que puede cargarse/descargarse en el mapa de memoria del SO (Kernel) bajo demanda
- Permiten extender la funcionalidad del Kernel en “caliente” (sin necesidad de reiniciar el sistema)
- Todo su código se ejecuta en modo Kernel (privilegiado)
- Cualquier error en el módulo, puede colgar el SO
- Permiten que el kernel se desarrolle bajo un diseño más modular
- Los módulos disponibles se ubican en /lib/modules/version del kernel
- Con el comando lsmod es posible ver qué módulos están cargados
- Vamos a ampliar en las próximas prácticas

Configurar el kernel (Módulos)

¿El soporte lo damos como módulo o built-in?

- Si es built-in, el kernel crece. Mayor uso de memoria. Puede incrementar el tiempo de arranque.
- Si es built-in, es más eficiente su utilización. No hay que cargar un adicional en memoria, acceso directo.
- En el soporte como módulo:
  - Si se quiere dar soporte a algún dispositivo, se carga el módulo y no es necesario recompilar (el soporte debe estar en el kernel).
  - Si hay una modificación de un driver, solo se modifica el módulo y no todo el código del kernel.
  - Los módulos se cargan bajo demanda, con lo cual la utilización de memoria es menor.

Parcheando el kernel

- Patch es un mecanismo que permite aplicar actualizaciones sobre una versión base.
- Se basa en archivos diff (archivos de diferencia), que indican qué agregar y qué quitar.
- En kernel.org hay de 2 tipos:
  - No incrementales: Se aplican sobre la versión mainline anterior (X.0.0). Cómo el que usarán en la práctica.
  - Incrementales: Se aplican sobre la versión inmediatamente anterior.
- Permiten agregar funcionalidad (nuevos drivers, correcciones menores, etc.)
- A veces puede resultar más sencillo descargar el archivo de diferencia y aplicarlo en vez de descargar todo el código de la nueva versión.

```
$ cd linux; xzcat ../patch-6.13.7.xz | patch -p1
```

Tip

Parámetro útil: --dry-run

Configurar el kernel

En el trabajo práctico debemos dar el siguiente soporte adicional:

• Soporte para FS BTRFS: En la opción File Systems, tendremos que seleccionar Btrfs filesystem

• Soporte para dispositivos de Loopback: En la opción Device Drivers → Block Devices, tendremos que seleccionar Loopback device support.

Compilación del kernel

```
$ make
```

El comando make busca el archivo Makefile, interpreta sus directivas y compila el kernel. Este proceso puede durar mucho tiempo dependiendo del procesador que tengamos.

Verificar que este proceso no arroje errores al concluir.

Tip

```
$ make -jX # X es el número de threads
```

Compilación de módulos

Compilando

```
$ make modules
```

El comando make modules compila todos los módulos necesarios para satisfacer las opciones que hayan sido seleccionadas como módulo.

Tip
Generalmente la tarea anterior se encuentra incluida en la compilación del kernel con el comando make

Instalación del kernel

Recién en este paso es necesario convertirse en root.
Al terminar el proceso de compilación, la imagen del kernel quedará ubicada en directorio-del-código/arch/arquitectura/boot/.
El próximo paso, entonces, es instalar el kernel y otros archivos en el directorio /boot.
Ejemplo con arquitectura i386

```
$ cd $HOME/kernel/linux-6.13
$ sudo su
# cp arch/i386/boot/bzImage /boot/vmlinuz-6.13.7
# cp System.map /boot/System.map-6.13.7
# cp .config /boot/config-6.13.7
```

Por supuesto, también existe una regla en el Makefile que realiza esto mismo automáticamente.
`$ sudo make install`
System.map: http://rlworkman.net/system.map/

Instalar los módulos

Los módulos compilados deben residir en el directorio /lib/modules/version-del-kernel.
Al igual que en el paso anterior, el archivo Makefile tiene una regla para instalar los módulos.

```
$ sudo make modules install
```

El parámetro modules install es una regla del Makefile que ubica los módulos del kernel recién compilado en el directorio correspondiente.

Creación del initramfs

- Un initramfs es un sistema de archivos temporal que se monta durante el arranque del sistema.
- Contiene ejecutables, drivers y módulos necesarios para lograr iniciar el sistema.
- Luego del proceso de arranque el disco se desmonta.

```
# mkinitramfs -o /boot/initrd.img-6.8 6.8
```

Configuración del gestor de arranque

- En el trabajo práctico utilizaremos la versión 2 de GRUB.
- Luego de instalar el kernel, para que el gestor de arranque lo reconozca simplemente deberemos ejecutar, como usuario privilegiado, el siguiente comando:

```
# update-grub2
```

Finalización del proceso

- Verificar que el kernel esté instalado en /boot:

```
$ test -f /boot/vmlinuz-6.13.7 && echo ‘‘Kernel instalado’’
```

- Verificar que los módulos estén instalados en /lib/modules/6.13.7

```
$ test -d /lib/modules/6.13.7-arquitectura && echo ‘‘Modulos
aparentemente instalados’’
```

- Verificar que el gestor de arranque haya indexado el kernel. Si el gestor de arranque utilizado es GRUB 2, entonces el siguiente comando debería generar un par de líneas de salida.

```
$ cat /boot/grub/grub.cfg | grep --color 6.13.7
```

- Paso final: ¡reiniciar y probar!

Agenda

Contenido:

- Fundamentos
- Historia
- Versionado
- Compilación
- Apéndice
  - Debian way

Debian way

Instalar un kernel de forma empaquetada ofrece muchas ventajas:
- Integración con el sistema operativo
- Bugfixes
- Actualizaciones de seguridad
- LTS (Long Term Support)

Debian designa una versión de Linux a la cual dar soporte durante todo el ciclo de vida de su versión estable, actualmente Debian 12 bookworm. Esto implica portar actualizaciones críticas y bugfixes. La versión de Linux que está en la actual Debian estable es la 6.1.

```
# apt-get install linux-source-6.1
$ tar xjf /usr/src/linux-source-6.1.tar.bz2
$ cd linux-source-6.1
$ make menuconfig
$scripts/config --disable DEBUG INFO
$ make clean
$ make deb-pkg
# dpkg -i ../linux-image-6.1.*.deb
```

Referencias

Linus Torvalds
Linux 3.0-rc1 https://lkml.org/lkml/2011/5/29/204, 2011
Andrew Tanembaum
Minix http://www.minix3.org/
Linus Torvalds
Linux History https://www.cs.cmu.edu/~awb/linux.history.html, 1992
Linus Torvalds
Release notes for Linux v0.12
http://ftp.funet.fi/pub/linux/historical/kernel/old-versions/RELNOTES-0.12, 1992
Debian Linux Kernel Handbook
http://kernel-handbook.alioth.debian.org/

¿Preguntas?