Introducción al kernel de linux

Mg. Ing. Gonzalo E. Sanchez
MSE - 2022

Implementación de Sistemas Operativos II

Intro kernel linux

- Generalidades
- Uso en sistemas embebidos
- Código fuente del kernel
- Esquema de versionado

Generalidades

Generalidades

- El kernel de linux es un componente de un sistema, no es todo el sistema.
- Ese sistema también requiere bibliotecas y aplicaciones para proveer funcionalidad a los usuarios.
- El kernel de linux fue creado en 1991 como hobby por un estudiante finlandés, Linus Torvalds
- Este kernel empezó a ser utilizado rápidamente por sistemas operativos gratuitos.

Generalidades

- Linus logró crear una comunidad de desarrolladores muy grande y dinámica alrededor de Linux.
- Hoy en dia, mas de mil personas contribuyen a cada release del kernel.
- Tanto compañías como individuos contribuyen
- Cualquiera de ustedes puede contribuir!

Linus Torvalds in 2014

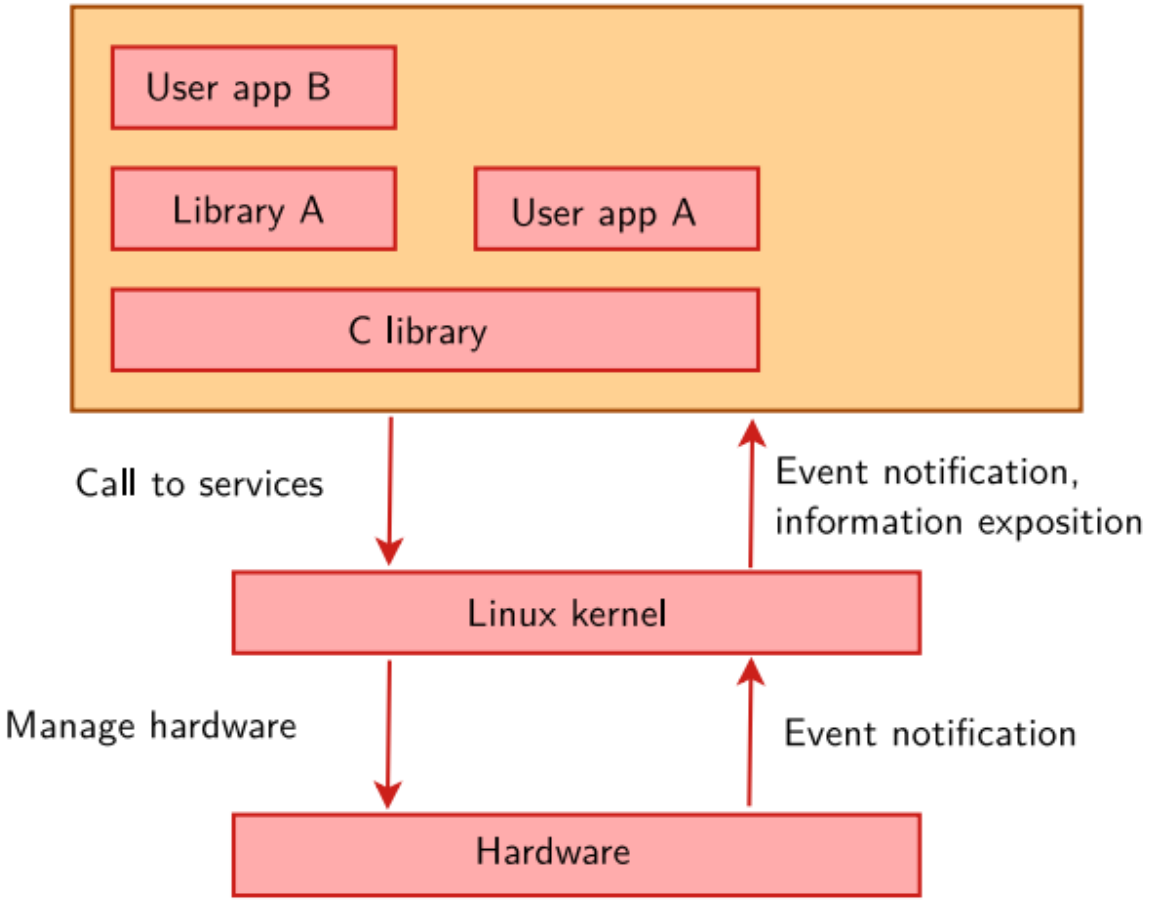
Generalidades

- Características principales del kernel:
  - Portabilidad y soporte de hardware amplio.
  - Escalabilidad: corre tanto en SBC como en servidores.
  - Seguridad revisada por muchos expertos. No se pueden esconder las fallas.
  - Gran y exhaustivo soporte para networking.
  - Estabilidad y confiabilidad.
  - Fácil de programar, mucha información en línea.

Generalidades

- Como cualquier otro OS, tiene la finalidad de gestionar el HW.
- Provee un set de APIs portables independientes de la arquitectura y el hardware.
- Permite a las aplicaciones en espacio usuario utilizar el hardware a través de estas APIs.
- Maneja los accesos concurrentes y la utilización de hardware hecho por distintas aplicaciones.
- Ejemplo: Distintas conexiones de red, una sola placa ethernet

Generalidades



Generalidades

- La interface principal entre el kernel y el espacio de usuario es un set de llamadas de sistema (System Calls).
- Este set se compone de unas 400 llamadas: Operaciones sobre archivos, networking, mapeo de memoria, etc.
- **Importante**: Este set se mantiene a través del tiempo.

Generalidades

- Desarrolladores solo pueden agregar system calls.
- Esta interface de system calls se envuelve por la biblioteca de C.
- **Importante**: Aplicaciones normalmente no efectúan system calls directamente.
- Utilizan la función correspondiente en la bibliote C para invocar el system call correspondiente.

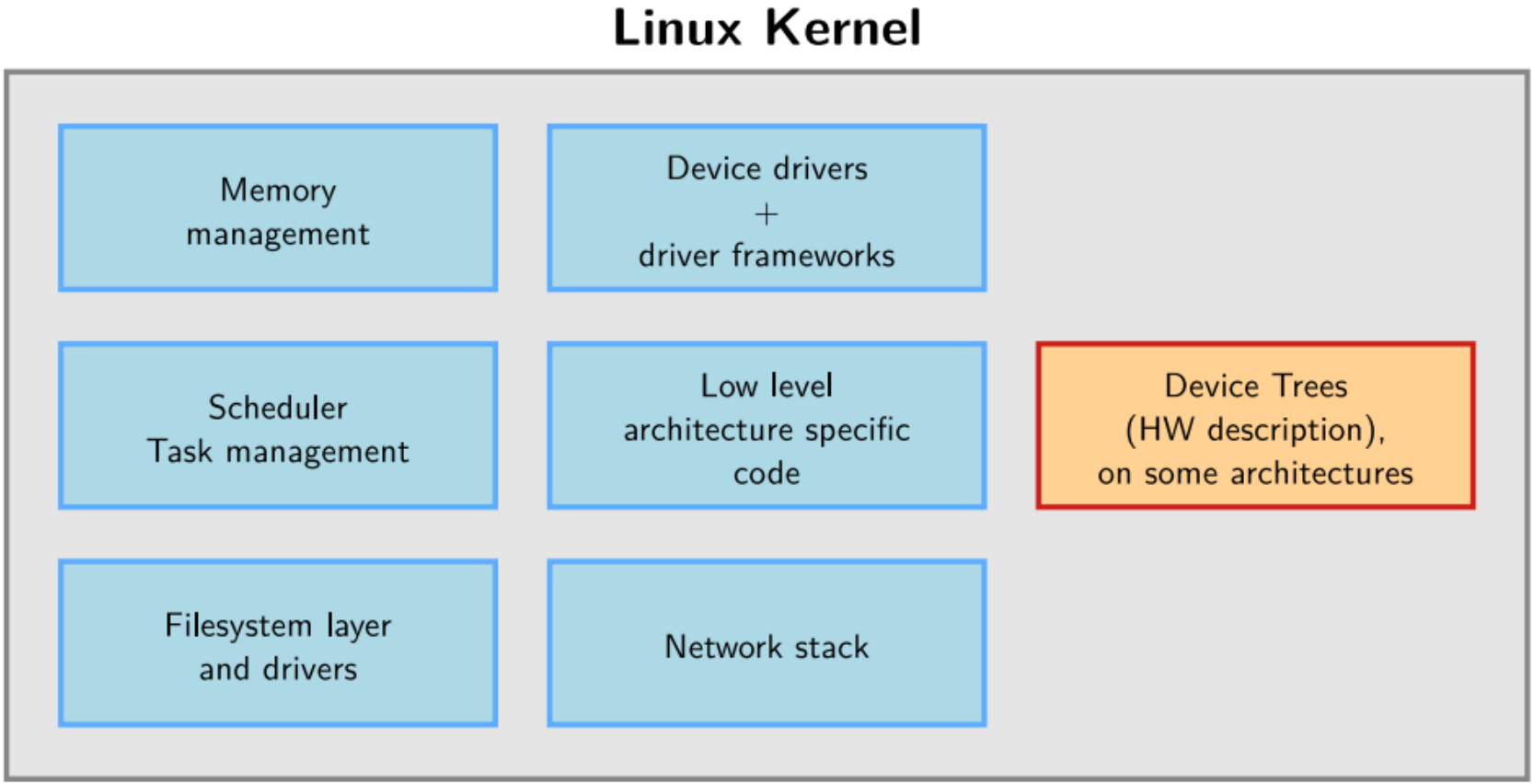
Generalidades

- Linux facilita información sobre el sistema y el kernel en el espacio usuario a través de pseudo filesystems.
- A veces se les llama virtual filesystems.
- Los pseudo filesystems permiten a las aplicaciones interactuar con archivos y directorios que no existen realmente.
- Se crean y se actualizan en tiempo real por el kernel.

Generalidades

- Los dos pseudo filesystems más importantes son ***proc*** y ***sysfs***:
- ***proc*** usualmente se monta en /proc.
  - Contiene información relacionada al OS (procesos, manejo de memoria, etc).
- ***sysfs*** usualmente se monta en /sys.
  - Representa el sistema como un conjunto de buses y dispositivos.
  - Da información acerca de estos dispositivos.

Generalidades



Generalidades

- El sistema de licencias para distribución del kernel es un tópico importante.
- Todos los archivos fuente de Linux son de software libre.
- Se distribuyen bajo la licencia GNU General Public License versión 2 (GPL v2 - ver link) .
- Esto implica que cuando se compra o recibe un dispositivo con linux, se debería recibir el código fuente del kernel.
- Si se produce un producto basado en linux, se debe liberar el código fuente con los mismos derechos, sin restricciones.

Generalidades

- Dentro del directorio **/arch** en los archivos fuente del kernel se encuentran las arquitecturas soportadas.
  - Arquitecturas de 32bits: Arm, microblaze, entre otros.
  - Arquitecturas de 64bits: Arm64, ia64, entre otros.
- Note que una arquitectura no mantenida que presenta problemas de compilación puede ser removida.
- El mínimo soportado es 32bits, y puede no tener MMU.
- En teoría, es posible correr linux en un Cortex-M4.

Uso en sistemas embebidos

Uso en Sistemas embebidos

- Las versiones oficiales del kernel de linux son llamadas versiones **mainline**.
- Estas están disponibles en la página https://kernel.org.
- Todos los kernels son publicado por el mismo Linus Torvalds.

- NOTA: puede ocurrir que una versión mainline no contenga soporte para cierto hardware.

Uso en Sistemas embebidos

- No tener soporte mainline ocurre aunque el código esté en desarrollo. Solamente no está aún listo para ser incorporado.
- Muchos fabricantes de chips y/o hardware mantienen sus propios kernels, dando soporte al hardware propio.
- Tener un núcleo fuera del mainline tiene dos problemas:
  - Solo mantenido por el fabricante, puede ser abandonado en cualquier momento.
  - Es propenso a tener un delta importante con el kernel mainline (menos estable).

Uso en Sistemas embebidos

- El codigo fuente del kernel esta disponible en https://kernel.org/pub/linux/kernel como tarballs.
- En nuestro caso, utilizaremos git.
  - Mucho más flexible
  - Permite trabajar a partir de una versión estable.
  - La creación de una rama propia para desarrollar es sumamente sencillo.
  - No es necesario aplicar patches a los tarballs, solamente se actualiza el repositorio.

Prioridades

HANDS ON

1. Clonar el repositorio de versiones estables según la guía de práctica.
2. Establecer la versión a ser utilizada.

Uso en Sistemas embebidos

- Para esta materia utilizaremos el kernel v5.10 (se puede utilizar el kernel v5.16).
- Bibliografía actual está basada en el kernel v4.9

Longterm release kernels

Version	Maintainer	Released	Projected EOL
5.10	Greg Kroah-Hartman & Sasha Levin	2020-12-13	Dec, 2026
5.4	Greg Kroah-Hartman & Sasha Levin	2019-11-24	Dec, 2025
4.19	Greg Kroah-Hartman & Sasha Levin	2018-10-22	Dec, 2024
4.14	Greg Kroah-Hartman & Sasha Levin	2017-11-12	Jan, 2024
4.9	Greg Kroah-Hartman & Sasha Levin	2016-12-11	Jan, 2023
4.4	Greg Kroah-Hartman & Sasha Levin	2016-01-10	Feb, 2022

Uso en Sistemas embebidos

- El kernel 5.4 posee:
  - 66.031 archivos (git ls-files | wc -l).
  - 27.679.764 líneas de código (git ls-files | xargs cat | wc -l).
  - 889.221.135 bytes (git ls-files | xargs cat | wc -c)
- Una vez compilado, el mínimo kernel de linux sin comprimir pesa entre 1 y 2 MB.
- Porque hay tanta diferencia de tamaño entre las fuentes y el kernel compilado?

Uso en Sistemas embebidos

- **Respuesta**: Miles de drivers, protocolos de networking, soporte de distintas arquitecturas, etc.
- El core es más bien pequeño.
- Algunos números aproximados:
  - drivers/: 57%
  - arch/: 16,3%
  - fs/: 5,5%
  - net/: 4,3%
  - kernel/: 1,2%

Uso en Sistemas embebidos

- El kernel está implementado en lenguaje C.
- Se utiliza assembler para algunas rutinas críticas y excepciones de CPU.
- No se utiliza C++ (http://vger.kernel.org/lkml/#s15-3).
- Todo el código se compila con gcc.
- Esto es porque se utilizan muchas extensiones propias de gcc.
- Por esto no es posible utilizar cualquier compilador ANSI C.

Uso en Sistemas embebidos

- **IMPORTANTE**: El kernel debe ser standalone. No puede depender de nada en el espacio usuario.
- Esto es por varias razones:
  - Arquitecturales: El espacio usuario se implementa sobre los servicios del kernel.
  - Técnicas: Durante el boot, el kernel está solo, aun antes de acceder al root filesystem.
- Esto implica que el kernel debe proveer sus propias implementaciones de bibliotecas.

Uso en Sistemas embebidos

- Una de las metas del kernel es la portabilidad.
- Implica que todo el código fuera de arch/ debe ser portable.
- Para esto el kernel provee algunas macros y funciones:
  - Endianness
  - Acceso a memoria I/O
  - API para DMA
- En código de espacio kernel, no debe utilizarse punto flotante.
- No existe la seguridad que haya disponible una FPU.

Uso en Sistemas embebidos

- En espacio de kernel, no existe la protección de memoria.
- El kernel no trata de recuperarse de intentos de acceder direcciones de memoria inválidas.
- Solo muestra un mensaje *oops* por consola.
- El tamaño del stack es fijo, pudiendo ser 4 u 8 KB.
- Nunca se implementó un mecanismo para hacerlo crecer.

Código fuente del kernel

Código fuente del kernel

- Los siguientes directorios son importantes para ver la estructura del código fuente del kernel.
- arch/<ARCH>
  - Código específico para cada arquitectura.
  - **arch/<ARCH>/mach-<machine>** contiene código específico para el SoC.
  - **arch/<ARCH>/include/asm** contiene headers específicos de la arquitectura.
  - **arch/<ARCH>/boot/dts** contiene el código fuente del Device Tree para algunas arquitecturas.

Código fuente del kernel

- block/
  - Contiene el core de la capa block (dispositivos).
- certs/
  - Administración de certificados para ingresos con clave.
- COPYING
  - Condiciones de copia para linux (GPL).
- CREDITS
  - Principales contribuidores de linux.

Código fuente del kernel

- crypto/
  - Bibliotecas criptográficas.
- Documentation/
  - Documentación del kernel. Incluye prototipos y comentarios extraidos del código fuente.
- drivers/
  - Todos los drivers excepto los de sonido.
- fs/
  - filesystems como ser ext4, etc

Código fuente del kernel

- include/
  - Headers del kernel.
- include/uapi
  - Headers para las API de espacio usuario.
- kernel/
  - Core kernel de linux.
- Makefile
  - Makefile principal de linux.

Esquema de versionado

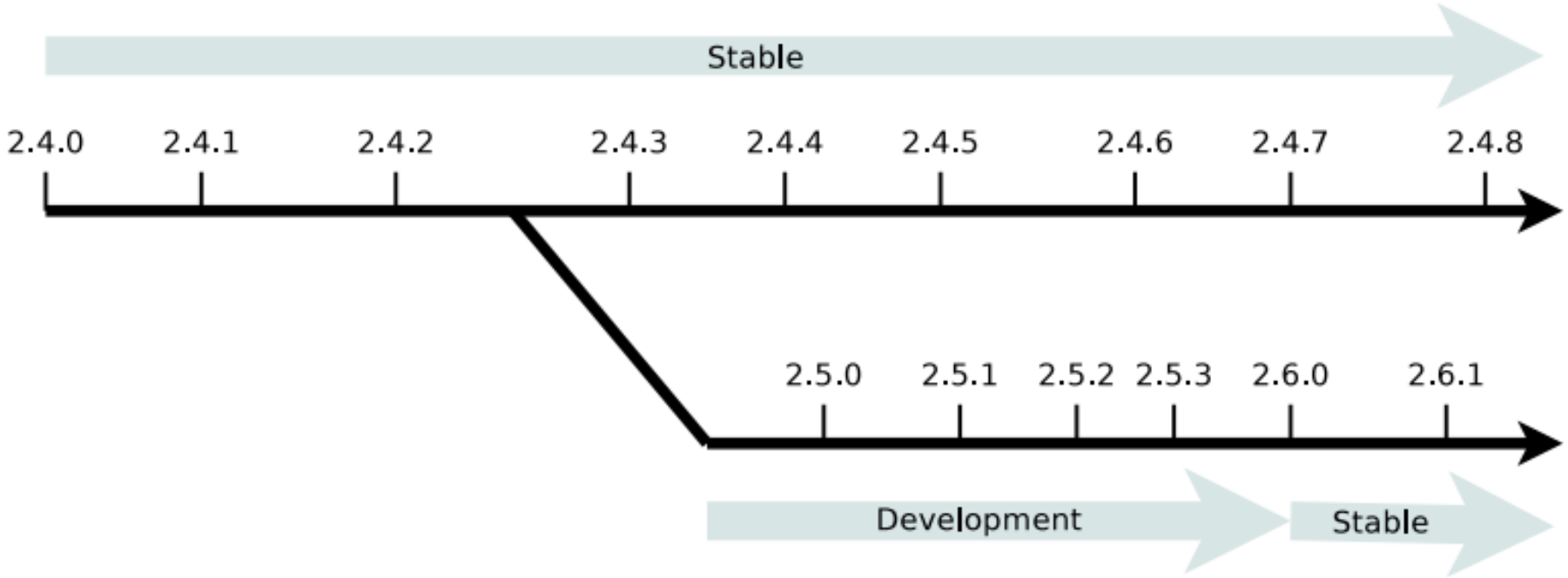
Esquema de versionado

- Hasta la versión 2.6 solamente había una rama estable mayor cada 2 o 3 años.
- Estaban identificadas por un número par en el medio.
- **EJEMPLO**: 1.0.x, 2.0.x, 2.2.x, 2.6.x

Esquema de versionado

- Existía una rama development para integrar nuevas funcionalidades y cambios mayores.
- Estas versiones se identificaban por un número impar en el medio.
- **EJEMPLO**: 2.1.x, 2.3.x, 2.5.x
- Luego de cierto tiempo, la versión de desarrollo se volvía la base de la rama estable.
- También existían minor releases de vez en cuando (2.2.23, 2.5.12, etc)

Esquema de versionado



Esquema de versionado

- A partir del kernel 2.6.0, los desarrolladores pudieron introducir muchas nuevas funcionalidades.
- Todas ellas fueron ingresadas a un ritmo estable, sin tener que hacer cambios disruptivos en los subsistemas existentes.
- Desde entonces, no ha habido la necesidad de crear una rama nueva de desarrollo.
- Esto usualmente rompía muchas compatibilidades con la rama estable

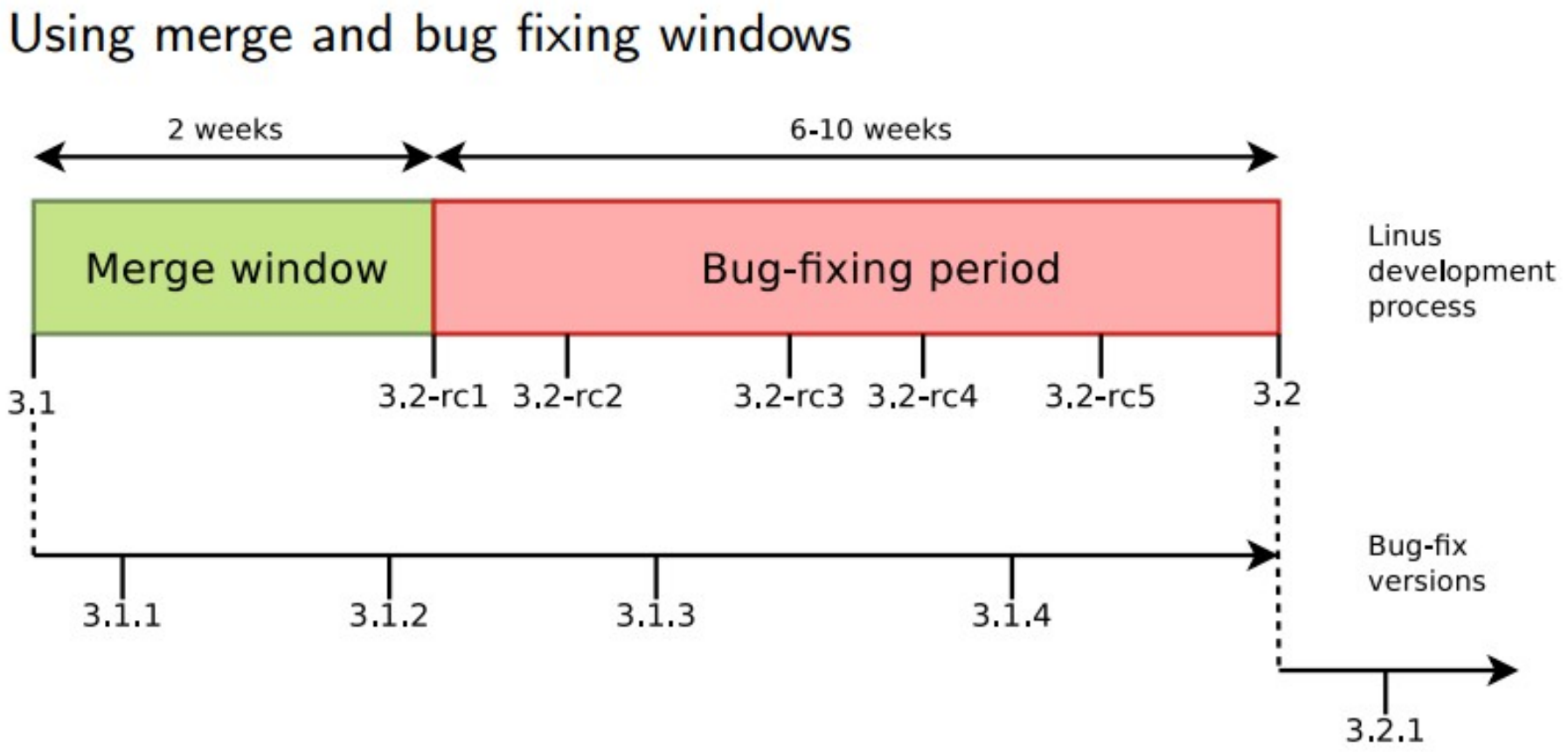
Esquema de versionado

- Gracias a no tener la necesidad de una nueva rama develop, más funcionalidades son puestas a disposición del usuario.
- Además, se hace a un ritmo mayor que lo que sucedía con anterioridad.
- Desde el 2003 hasta el 2011 las versiones recibían el nombre de 2.6.x
- Linux 3.0 fue lanzado en julio de 2011.
- Esto en realidad es solo un cambio de esquema de numeración.

Esquema de versionado

- Las versiones oficiales del kernel se llaman ahora 3.x
- **EJEMPLO**: 3.0, 3.1, 3.2, etc.
- Versiones ya estabilizadas son nombradas 3.x.y
- **EJEMPLO**: 3.0.2, 3.4.3, etc.
- Únicamente se remueve un dígito comparado con el esquema de numeración anterior.

Esquema de versionado



Esquema de versionado

- Luego del release de una versión 3.x (por ejemplo) se abre una ventana de merge.
- En esta ventana se agregan las adiciones más importantes.
- La ventana de merge se cierra por el release de la versión de testeo 3.(x+1)-rc1
- El periodo de bug fixing se abre por 6 a 10 semanas.
- En intervalos regulares durante este periodo se lanzan nuevas versiones test 3.(x+1)-rcY.

Esquema de versionado

- Al considerarse que el núcleo es suficientemente estable, el kernel 3.(x+1) es lanzado y el proceso comienza de nuevo.

- Este proceso también tiene problemas: los fixes de bugs y de seguridad solo son lanzados para versiones recientes estables.

- Algunos usuarios necesitan un kernel reciente, pero que sea LTS para tener actualizaciones de seguridad.

Esquema de versionado

- Es un punto importante a tener en cuenta al momento de seleccionar el kernel para el sistema embebido.
- En el frontpage de kernel.org se muestran las versiones actuales y cuanto tiempo van a ser soportadas.
- Esto se conoce como EOL (end of life).

Gracias.