📁 LPIC-2 / 🧠 Examen 201 - Kernel de Linux

201.2 Compilar un kernel

Teoría

La mayoría de las veces utilizarás los kernels precompilados proporcionados por tu distribución. Sin embargo, hay situaciones en las que puede ser necesario o deseable compilar un kernel personalizado:

- **Soporte de Hardware Específico:** Añadir soporte para hardware muy nuevo o inusual para el que los kernels de distribución no incluyen drivers o no los tienen habilitados.
- **Habilitar/Deshabilitar Características Específicas:** Incluir funcionalidades del kernel que no vienen activadas por defecto (ej: soporte para un sistema de archivos experimental) o, por el contrario, deshabilitar características para reducir el tamaño del kernel o minimizar la superficie de ataque.
- **Optimización de Rendimiento:** Compilar solo con los drivers y características necesarias para tu hardware específico, lo que puede resultar en un kernel ligeramente más pequeño y rápido.
- **Seguridad:** Deshabilitar características potencialmente inseguras o aplicar parches de seguridad que aún no están disponibles en los paquetes de la distribución.
- **Depuración:** Incluir herramientas de depuración del kernel.

Proceso General de Compilación del Kernel:

Compilar un kernel desde cero implica varios pasos secuenciales:

1. **Obtener el Código Fuente del Kernel:**
    - Descargar el código fuente oficial desde kernel.org.
    - Obtener el código fuente empaquetado por tu distribución. Esto es a menudo preferible, ya que puede incluir parches específicos de la distribución.
        - **Rama Debian/Ubuntu:** Paquete `linux-source-<version>`. Se instala típicamente en `/usr/src/`.
        - **Rama Red Hat/CentOS/Fedora:** Paquete `kernel-devel`. También se instala en `/usr/src/`.
    - Extraer el código fuente si viene en un archivo comprimido (ej: `tar xvfz linux-<version>.tar.gz`). La convención es extraerlo en `/usr/src/`.
2. **Configurar el Kernel:**
    - Este es el paso más crucial. Determina qué características, subsistemas y controladores se incluirán en el kernel compilado y si se compilarán *en* el kernel (`=y`) o como *módulos* cargables (`=m`). Los módulos se cargan bajo demanda y mantienen el tamaño del kernel principal más pequeño.

- **Empezar con una configuración existente:** La mejor práctica es empezar con la configuración del kernel que ya está funcionando en tu sistema.
    - Copia el archivo de configuración del kernel en ejecución a tu directorio de código fuente: `cp /boot/config-$(uname -r) .config` (en el directorio del código fuente).
    - Si ese archivo no existe, puedes intentar copiar `/proc/config.gz` (si está disponible) y descomprimirlo: `zcat /proc/config.gz > .config`.
- **Herramientas para Modificar la Configuración (`.config`):**
    - `make menuconfig`: La herramienta más común basada en ncurses (interfaz de texto con menús). Permite navegar por las opciones de configuración, seleccionarlas (espacio: `*` = compilado en, `M` = módulo, vacío = excluido) y guardarlas. Requiere el paquete `libncurses-dev` (Debian) o `ncurses-devel` (Red Hat).
    - `make oldconfig`: Utiliza un archivo `.config` existente y te pregunta solo sobre las nuevas opciones que no estaban en el archivo original (útil al actualizar el código fuente a una nueva versión).
    - `make xconfig`: Interfaz gráfica basada en Qt. Requiere librerías Qt devel.
    - `make gconfig`: Interfaz gráfica basada en GTK. Requiere librerías GTK devel.
    - `make config`: La herramienta más antigua, pregunta una por una por *todas* las opciones (¡muy tediosa!).
- Es importante deshabilitar las características que no necesitas y asegurarte de que los controladores para tu hardware esencial (controladores de disco, red) estén compilados *en* el kernel (`=y`) y no como módulos, de lo contrario, el sistema podría no arrancar.

3. **Compilar el Kernel y los Módulos:**

    - Este paso consume mucho tiempo y recursos (CPU, RAM, espacio en disco).
    - `make bzImage`: Compila la imagen principal del kernel.
    - `make modules`: Compila los módulos del kernel configurados como `=m`.
    - Puedes acelerar la compilación utilizando la opción `-j <número_jobs>` (ej: `make -j $(nproc) bzImage modules`). `$(nproc)` devuelve el número de núcleos de CPU.
    - Deberás instalar las dependencias de compilación necesarias (compilador GCC, `make`, `flex`, `bison`, `openssl-devel`/`libssl-dev`, etc. - los nombres de los paquetes varían ligeramente entre distribuciones).
4. **Instalar el Kernel y los Módulos:**

    - Este paso copia la imagen del kernel y los módulos compilados a las ubicaciones correctas del sistema. Requiere permisos de root.
    - `sudo make modules_install`: Instala los módulos compilados en `/lib/modules/<versión_kernel_compilado>/`.

- `sudo make install`: Copia la imagen del kernel (`vmlinuz-<versión>`) y otros archivos (`System.map-<versión>`, `config-<versión>`) a `/boot/`, genera un nuevo initramfs para el nuevo kernel y **actualiza la configuración del gestor de arranque** (GRUB o LILO) para incluir el nuevo kernel como una opción de arranque.
- **Método Recomendado (Construir Paquetes):** En un entorno de producción, es mucho más seguro y manejable construir paquetes `.deb` (Debian) o `.rpm` (Red Hat) a partir del código fuente del kernel y luego instalar esos paquetes con el gestor de paquetes (`dpkg -i` o `dnf install`). Esto permite desinstalar o gestionar versiones del kernel fácilmente.
    - **Rama Debian/Ubuntu:** Usa la herramienta `make-kpkg`. `make-kpkg clean`, `make-kpkg --initrd --append-to-version -custom_1.0 kernel_image kernel_headers`. Esto crea archivos `.deb` en el directorio padre.
    - **Rama Red Hat/CentOS/Fedora:** Usa herramientas `rpmbuild` y archivos `.spec` específicos. Este proceso es más complejo que `make-kpkg`.

5. **Actualizar el Gestor de Arranque (si `make install` no lo hizo correctamente):**
    - Asegurarse de que el gestor de arranque (GRUB2 es el más común) conoce el nuevo kernel y lo añade al menú de arranque. `make install` generalmente lo hace automáticamente. Si no, puede que necesites ejecutar `sudo update-grub` (Debian/Ubuntu) o `sudo grub2-mkconfig -o /boot/grub2/grub.cfg` (Red Hat/Fedora).

Riesgos y Precauciones:

- **Sistema Inarrancable:** Una configuración incorrecta o un error de compilación pueden impedir que el sistema arranque. **Siempre mantén tu kernel original funcionando** como opción de arranque.
- **Falta de Drivers:** Si olvidas incluir drivers esenciales (controlador de disco, tarjeta de red) o los compilas como módulos pero el initramfs no los incluye, el sistema podría no encontrar el sistema de archivos raíz o la red.
- **Inestabilidad:** Un kernel compilado incorrectamente puede causar fallos, cuelgues o comportamiento inesperado.

LPIC-2 se enfoca en entender el *proceso*, las *herramientas* y la *configuración*, más que en la compilación exitosa en sí misma. Practicar en una VM con snapshots es esencial.