Red Hat Enterprise Linux 8

Desarrollo de aplicaciones C y C++ en RHEL 8

Configuración de una estación de trabajo para desarrolladores, y desarrollo y depuración de aplicaciones C y C++ en Red Hat Enterprise Linux 8

Last Updated: 2021-03-07

Red Hat Enterprise Linux 8 Desarrollo de aplicaciones C y C++ en RHEL 8

Configuración de una estación de trabajo para desarrolladores, y desarrollo y depuración de aplicaciones C y C++ en Red Hat Enterprise Linux 8

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

Legal Notice

Resumen

Este documento describe las diferentes características y utilidades que hacen de Red Hat Enterprise Linux 8 una plataforma empresarial ideal para el desarrollo de aplicaciones.

Table of Contents

HACER QUE EL CÓDIGO ABIERTO SEA MÁS INCLUSIVO

Red Hat se compromete a sustituir el lenguaje problemático en nuestro código, documentación y propiedades web. Estamos empezando con estos cuatro términos: maestro, esclavo, lista negra y lista blanca. Debido a la enormidad de este esfuerzo, estos cambios se implementarán gradualmente a lo largo de varias versiones próximas. Para más detalles, consulte el mensaje de nuestro CTO Chris Wright .

PROPORCIONAR COMENTARIOS SOBRE LA DOCUMENTACIÓN DE RED HAT

Agradecemos su opinión sobre nuestra documentación. Por favor, díganos cómo podemos mejorarla. Para ello:

- Para comentarios sencillos sobre pasajes concretos:
  1. Asegúrese de que está viendo la documentación en el formato *Multi-page HTML*. Además, asegúrese de ver el botón **Feedback** en la esquina superior derecha del documento.
  2. Utilice el cursor del ratón para resaltar la parte del texto que desea comentar.
  3. Haga clic en la ventana emergente **Add Feedback** que aparece debajo del texto resaltado.
  4. Siga las instrucciones mostradas.
- Para enviar comentarios más complejos, cree un ticket de Bugzilla:
  1. Vaya al sitio web de Bugzilla.
  2. Como componente, utilice **Documentation**.
  3. Rellene el campo **Description** con su sugerencia de mejora. Incluya un enlace a la(s) parte(s) pertinente(s) de la documentación.
  4. Haga clic en **Submit Bug**.

CAPÍTULO 1. CONFIGURACIÓN DE UNA ESTACIÓN DE TRABAJO DE DESARROLLO

Red Hat Enterprise Linux 8 soporta el desarrollo de aplicaciones personalizadas. Para permitir a los desarrolladores hacerlo, el sistema debe ser configurado con las herramientas y utilidades requeridas. Este capítulo enumera los casos de uso más comunes para el desarrollo y los elementos a instalar.

1.1. REQUISITOS PREVIOS

- El sistema debe estar instalado, incluyendo un entorno gráfico, y suscrito.

1.2. HABILITACIÓN DE LOS REPOSITORIOS DE DEPURACIÓN Y DE CÓDIGO FUENTE

Una instalación estándar de Red Hat Enterprise Linux no habilita los repositorios de depuración y de fuentes. Estos repositorios contienen la información necesaria para depurar los componentes del sistema y medir su rendimiento.

Procedimiento

- Habilitar los canales de paquetes de información de origen y de depuración:

```
# subscription-manager repos --enable rhel-8-for-$(uname -i)-baseos-debug-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-baseos-source-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-appstream-debug-rpms
# subscription-manager repos --enable rhel-8-for-$(uname -i)-appstream-source-rpms
```

La parte **$(uname -i)** se sustituye automáticamente por un valor correspondiente a la arquitectura de su sistema:

Nombre de la arquitectura	Valor
Intel y AMD de 64 bits	x86_64
ARM de 64 bits	aarch64
IBM POWER	ppc64le
IBM Z	s390x

1.3. CONFIGURACIÓN PARA GESTIONAR LAS VERSIONES DE LAS APLICACIONES

Un control de versiones eficaz es esencial para todos los proyectos de múltiples desarrolladores. Red Hat Enterprise Linux se entrega con Git, un sistema de control de versiones distribuido.

Procedimiento

1. Instale el **git** paquete:

```
# yum install git
```

2. Opcional: Establezca el nombre completo y la dirección de correo electrónico asociados a sus confirmaciones Git:

```
$ git config --global user.name "Full Name"
$ git config --global user.email "email@example.com"
```

Sustituye *Full Name* y *email@example.com* por tu nombre y dirección de correo electrónico reales.

3. Opcional: Para cambiar el editor de texto por defecto iniciado por Git, establezca el valor de la opción de configuración **core.editor**:

```
$ git config --global core.editor command
```

Sustituya *command* por el comando que se utilizará para iniciar el editor de texto seleccionado.

Recursos adicionales

- Páginas de manuales de Linux para Git y tutoriales:

```
$ man git
$ man gittutorial
$ man gittutorial-2
```

Tenga en cuenta que muchos comandos de Git tienen sus propias páginas de manual. Como ejemplo, véase *git-commit(1)*.

- *Git User's Manual*
- Pro Git
- Referencia

1.4. CONFIGURACIÓN PARA DESARROLLAR APLICACIONES CON C Y C++

Red Hat Enterprise Linux incluye herramientas para crear aplicaciones C y C++.

Requisitos previos

- Los repositorios de depuración y de fuentes deben estar habilitados.

Procedimiento

1. Instale el **Development Tools** grupo de paquetes que incluye la colección de compiladores de GNU (GCC), el depurador de GNU (GDB) y otras herramientas de desarrollo:

```
# yum group install \ "Herramientas de desarrollo"
```

2. Instale la cadena de herramientas basada en LLVM, incluyendo el compilador **clang** y el depurador **lldb**:

```
# yum install llvm-toolset
```

3. Opcional: Para las dependencias de Fortran, instale el compilador GNU Fortran:

```
# yum install gcc-gfortran
```

1.5. CONFIGURACIÓN PARA DEPURAR APLICACIONES

Red Hat Enterprise Linux ofrece múltiples herramientas de depuración e instrumentación para analizar y solucionar el comportamiento interno de las aplicaciones.

Requisitos previos

- Los repositorios de depuración y de fuentes deben estar habilitados.

Procedimiento

1. Instala las herramientas útiles para la depuración:

```
# yum install gdb valgrind systemtap ltrace strace
```

2. Instale el paquete **yum-utils** paquete para poder utilizar la herramienta **debuginfo-install**:

```
# yum install yum-utils
```

3. Ejecuta un script de ayuda de SystemTap para configurar el entorno.

```
# stap-prep
```

Tenga en cuenta que **stap-prep** instala paquetes relevantes para el núcleo actual *running*, que puede no ser el mismo que el núcleo o los núcleos realmente instalados. Para asegurarse de que **stap-prep** instala los paquetes correctos **kernel-debuginfo** y **kernel-headers** paquetes, vuelva a comprobar la versión actual del kernel utilizando el comando **uname -r** y reinicie su sistema si es necesario.

4. Asegúrese de que las políticas de **SELinux** permiten que las aplicaciones pertinentes se ejecuten no sólo normalmente, sino también en las situaciones de depuración. Para más información, consulte Uso de SELinux.

Recursos adicionales

- Capítulo 7, *Activación de la depuración con información de depuración*

1.6. CONFIGURACIÓN PARA MEDIR EL RENDIMIENTO DE LAS APLICACIONES

Red Hat Enterprise Linux incluye varias aplicaciones que pueden ayudar a un desarrollador a identificar las causas de la pérdida de rendimiento de la aplicación.

Requisitos previos

- Los repositorios de depuración y de fuentes deben estar habilitados.

Procedimiento

1. Instalar las herramientas para la medición del rendimiento:

```
# yum install perf papi pcp-zeroconf valgrind strace sysstat systemtap
```

2. Ejecuta un script de ayuda de SystemTap para configurar el entorno.

```
# stap-prep
```

   Tenga en cuenta que **stap-prep** instala paquetes relevantes para el núcleo actual *running*, que puede no ser el mismo que el núcleo o los núcleos realmente instalados. Para asegurarse de que **stap-prep** instala los paquetes correctos **kernel-debuginfo** y **kernel-headers** paquetes, vuelva a comprobar la versión actual del kernel utilizando el comando **uname -r** y reinicie su sistema si es necesario.

3. Habilite e inicie el servicio de recolección de Performance Co-Pilot (PCP):

```
# systemctl enable pmcd && systemctl start pmcd
```

PARTE I. CREACIÓN DE APLICACIONES EN C O C

Red Hat ofrece múltiples herramientas para crear aplicaciones utilizando los lenguajes C y C. Esta parte del libro enumera algunas de las tareas de desarrollo más comunes.

CAPÍTULO 2. CÓDIGO DE CONSTRUCCIÓN CON GCC

Este capítulo describe situaciones en las que el código fuente debe transformarse en código ejecutable.

2.1. RELACIÓN ENTRE LAS FORMAS DE CÓDIGO

Requisitos previos

- Comprender los conceptos de compilación y vinculación

Posibles formas de código

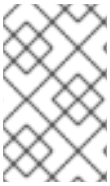
Los lenguajes C y C++ tienen tres formas de código:

- **Source code** escritos en el lenguaje C o C }, se presentan como archivos de texto sin formato. Los archivos suelen utilizar extensiones como **.c**, **.cc**, **.cpp**, **.h**, **.hpp**, **.i**, **.inc**. Para obtener una lista completa de las extensiones soportadas y su interpretación, consulte las páginas del manual de gcc:

```
$ man gcc
```

- **Object code**, creado por *compiling* el código fuente con un *compiler*. Se trata de una forma intermedia.
  Los archivos de código objeto utilizan la extensión **.o**.

- **Executable code**, creado por el código objeto *linking* con un *linker*.
  Los archivos ejecutables de aplicaciones de Linux no utilizan ninguna extensión de nombre de archivo. Los archivos ejecutables de objetos compartidos (bibliotecas) utilizan la extensión de nombre de archivo **.so**.



NOTA

También existen archivos de biblioteca para la vinculación estática. Se trata de una variante del código objeto que utiliza la extensión de nombre de archivo **.a**. No se recomienda la vinculación estática. Véase Sección 3.2, “Enlace estático y dinámico” .

Manejo de formas de código en GCC

La producción de código ejecutable a partir del código fuente se realiza en dos pasos, que requieren aplicaciones o herramientas diferentes. GCC puede utilizarse como controlador inteligente tanto para los compiladores como para los enlazadores. Esto le permite utilizar un único comando **gcc** para cualquiera de las acciones requeridas (compilar y enlazar). GCC selecciona automáticamente las acciones y su secuencia:

1. Los archivos fuente se compilan en archivos objeto.
2. Se enlazan los archivos de objetos y las bibliotecas (incluyendo las fuentes previamente compiladas).

Es posible ejecutar GCC para que realice sólo el paso 1, sólo el paso 2, o ambos pasos 1 y 2. Esto viene determinado por los tipos de entradas y el tipo de salida(s) solicitado(s).

Dado que los proyectos más grandes requieren un sistema de compilación que normalmente ejecuta GCC por separado para cada acción, es mejor considerar siempre la compilación y el enlazado como dos acciones distintas, aunque GCC pueda realizar ambas a la vez.

Recursos adicionales

- Sección 2.2, "Compilación de archivos fuente a código objeto"
- Sección 2.6, "Vinculación de código para crear archivos ejecutables"

2.2. COMPILACIÓN DE ARCHIVOS FUENTE A CÓDIGO OBJETO

Para crear archivos de código objeto a partir de archivos fuente y no un archivo ejecutable inmediatamente, se debe instruir a GCC para que cree sólo archivos de código objeto como su salida. Esta acción representa la operación básica del proceso de construcción para los proyectos más grandes.

Requisitos previos

- Archivo(s) de código fuente en C o C
- GCC instalado en el sistema

Procedimiento

1. Cambie al directorio que contiene los archivos de código fuente.
2. Ejecute **gcc** con la opción **-c**:

```
$ gcc -c source.c another_source.c
```

Se crean archivos de objetos, cuyos nombres reflejan los archivos de código fuente originales: **source.c** da como resultado **source.o**.

NOTA

Con el código fuente de C, sustituya el comando **gcc** por **g** para manejar convenientemente las dependencias de la biblioteca estándar de C.

Recursos adicionales

- Sección 2.5, "Opciones para endurecer el código con GCC"
- Sección 2.4, "Optimización del código con GCC"
- Sección 2.7, "Ejemplo: Construir un programa en C con GCC"

2.3. HABILITACIÓN DE LA DEPURACIÓN DE APLICACIONES C Y C++ CON GCC

Debido a que la información de depuración es grande, no se incluye por defecto en los archivos ejecutables. Para habilitar la depuración de tus aplicaciones C y C++ con ella, debes instruir explícitamente al compilador para que la cree.

Para permitir la creación de información de depuración con **GCC** al compilar y enlazar el código, utilice la opción **-g**:

```
$ gcc ... -g ...
```

- Las optimizaciones realizadas por el compilador y el enlazador pueden dar lugar a un código ejecutable difícil de relacionar con el código fuente original: las variables pueden ser optimizadas, los bucles desenrollados, las operaciones fusionadas con las circundantes, etc. Esto afecta negativamente a la depuración. Para mejorar la experiencia de depuración, considere la posibilidad de configurar la optimización con la opción **-Og**. Sin embargo, cambiar el nivel de optimización cambia el código ejecutable y puede cambiar el comportamiento real, incluyendo la eliminación de algunos errores.

- Para incluir también las definiciones de macros en la información de depuración, utilice la opción **-g3** en lugar de **-g**.

- La opción **-fcompare-debug** GCC comprueba el código compilado por GCC con información de depuración y sin información de depuración. La prueba pasa si los dos archivos binarios resultantes son idénticos. Esta prueba garantiza que el código ejecutable no se ve afectado por ninguna opción de depuración, lo que garantiza además que no hay errores ocultos en el código de depuración. Tenga en cuenta que el uso de la opción **-fcompare-debug** aumenta significativamente el tiempo de compilación. Consulte la página del manual de GCC para obtener detalles sobre esta opción.

Recursos adicionales

- Capítulo 7, *Activación de la depuración con información de depuración*
- Uso de la colección de compiladores de GNU (GCC)
- Depuración con GDB
- La página del manual de GCC:

  ```
  $ man gcc
  ```

2.4. OPTIMIZACIÓN DEL CÓDIGO CON GCC

Un mismo programa puede transformarse en más de una secuencia de instrucciones de máquina. Se puede conseguir un resultado más óptimo si se asignan más recursos al análisis del código durante la compilación.

Con GCC, puede establecer el nivel de optimización utilizando la opción **-O*level*** opción. Esta opción acepta un conjunto de valores en lugar del *level*.

Nivel	Descripción
0	Optimizar para la velocidad de compilación - sin optimización del código (por defecto).
1, **2**, **3**	Optimizar para aumentar la velocidad de ejecución del código (cuanto mayor sea el número, mayor será la velocidad).

Nivel	Descripción
s	Optimizar el tamaño del archivo.
fast	Lo mismo que una configuración de nivel **3**, además de que **fast** hace caso omiso del estricto cumplimiento de las normas para permitir optimizaciones adicionales.
g	Optimizar la experiencia de depuración.

Para las compilaciones de lanzamiento, utilice la opción de optimización **-O2**.

Durante el desarrollo, la opción **-Og** es útil para depurar el programa o la biblioteca en algunas situaciones. Dado que algunos errores se manifiestan sólo con determinados niveles de optimización, pruebe el programa o la biblioteca con el nivel de optimización de lanzamiento.

GCC ofrece un gran número de opciones para permitir optimizaciones individuales. Para más información, consulte los siguientes recursos adicionales.

Recursos adicionales

- Uso de la colección de compiladores de GNU
- Página del manual de Linux para GCC:

```
$ man gcc
```

2.5. OPCIONES PARA ENDURECER EL CÓDIGO CON GCC

Cuando el compilador transforma el código fuente en código objeto, puede añadir varias comprobaciones para evitar las situaciones más habituales y aumentar la seguridad. Elegir el conjunto adecuado de opciones del compilador puede ayudar a producir programas y bibliotecas más seguros, sin tener que cambiar el código fuente.

Opciones de la versión de lanzamiento

La siguiente lista de opciones es la mínima recomendada para los desarrolladores que tienen como objetivo Red Hat Enterprise Linux:

```
$ gcc ... -O2 -g -Wall -Wl,-z,now,-z,relro -fstack-protector-strong -fstack-clash-protection -
D_FORTIFY_SOURCE=2 ...
```

- Para los programas, añada las opciones **-fPIE** y **-pie** Position Independent Executable.
- En el caso de las bibliotecas enlazadas dinámicamente, la opción obligatoria **-fPIC** (Position Independent Code) aumenta indirectamente la seguridad.

Opciones de desarrollo

Utilice las siguientes opciones para detectar fallos de seguridad durante el desarrollo. Utilice estas opciones junto con las opciones de la versión de lanzamiento:

```
$ gcc ... -Walloc-cero -Walloca-larger-than -Wextra -Wformat-security -Wvla-larger-than ...
```

Recursos adicionales

- Guía de codificación defensiva
- Detección de errores de memoria con GCC

2.6. VINCULACIÓN DE CÓDIGO PARA CREAR ARCHIVOS EJECUTABLES

La vinculación es el último paso en la construcción de una aplicación C o C. La vinculación combina todos los archivos de objetos y bibliotecas en un archivo ejecutable.

Requisitos previos

- Uno o varios archivos de objetos
- GCC debe estar instalado en el sistema

Procedimiento

1. Cambie al directorio que contiene los archivos de código objeto.
2. Ejecutar **gcc**:

   ```
   $ gcc ... objfile.o another_object.o... -o executable-file
   ```

   Se crea un archivo ejecutable llamado ***executable-file*** se crea a partir de los archivos de objetos y bibliotecas suministrados.

   Para enlazar bibliotecas adicionales, añada las opciones necesarias después de la lista de archivos de objetos. Para más información, consulte Capítulo 3, *Uso de bibliotecas con GCC*.

NOTA

Con el código fuente de C, sustituya el comando **gcc** por **g** para manejar convenientemente las dependencias de la biblioteca estándar de C.

Recursos adicionales

- Sección 2.7, “Ejemplo: Construir un programa en C con GCC”
- Sección 3.2, “Enlace estático y dinámico”

2.7. EJEMPLO: CONSTRUIR UN PROGRAMA EN C CON GCC

Este ejemplo muestra los pasos exactos para construir un sencillo programa C de ejemplo.

Requisitos previos

- Debes entender cómo usar GCC.

Procedimiento

1. Cree un directorio **hello-c** y cambie a él:

```
$ mkdir hello-c
$ cd hello-c
```

2. Cree el archivo **hello.c** con el siguiente contenido:

```
#include <stdio.h>

int main() {
  printf("Hello, World!\n");
  return 0;
}
```

3. Compilar el código con GCC:

```
$ gcc -c hola.c
```

Se crea el archivo de objetos **hello.o**.

4. Enlaza un archivo ejecutable **helloworld** desde el archivo objeto:

```
$ gcc hello.o -o helloworld
```

5. Ejecute el archivo ejecutable resultante:

```
$ ./helloworld
Hello, World!
```

Recursos adicionales

- Sección 5.2, "Ejemplo: Construir un programa en C usando un Makefile"

2.8. EJEMPLO: CONSTRUIR UN PROGRAMA EN C CON GCC

Este ejemplo muestra los pasos exactos para construir un ejemplo de programa mínimo en C.

Requisitos previos

- Debe entender la diferencia entre **gcc** y **g** .

Procedimiento

1. Cree un directorio **hello-cpp** y cambie a él:

```
$ mkdir hello-cpp
$ cd hello-cpp
```

2. Cree el archivo **hello.cpp** con el siguiente contenido:

```
#include <iostream>
```

```
int main() {
  std::cout << "Hello, World!\n";
  return 0;
}
```

3. Compila el código con **g** :

```
$ g -c hola.cpp
```

Se crea el archivo de objetos **hello.o**.

4. Enlaza un archivo ejecutable **helloworld** desde el archivo objeto:

```
$ g hello.o -o helloworld
```

5. Ejecute el archivo ejecutable resultante:

```
$ ./helloworld
Hello, World!
```

CAPÍTULO 3. USO DE BIBLIOTECAS CON GCC

Este capítulo describe el uso de las bibliotecas en el código.

3.1. CONVENCIONES DE NOMENCLATURA DE LAS BIBLIOTECAS

Se utiliza una convención especial de nombres de archivos para las bibliotecas: se espera que una biblioteca conocida como **foo** exista como archivo **lib*foo*.so** o **lib*foo*.a**. Esta convención es entendida automáticamente por las opciones de entrada de enlace de GCC, pero no por las opciones de salida:

- Cuando se enlaza con la biblioteca, ésta sólo puede especificarse por su nombre **foo** con la opción **-l** como **-l*foo***:

```
$ gcc ... -lfoo...
```

- Al crear la biblioteca, el nombre completo del archivo **lib*foo*.so** o **lib*foo*.a** debe especificarse.

Recursos adicionales

- Sección 4.2, “El mecanismo de soname”

3.2. ENLACE ESTÁTICO Y DINÁMICO

Los desarrolladores tienen la opción de usar enlace estático o dinámico cuando construyen aplicaciones con lenguajes completamente compilados. Esta sección enumera las diferencias, particularmente en el contexto del uso de los lenguajes C y C++ en Red Hat Enterprise Linux. Para resumir, Red Hat desaconseja el uso del enlazado estático en aplicaciones para Red Hat Enterprise Linux.

Comparación de la vinculación estática y dinámica

La vinculación estática hace que las bibliotecas formen parte del archivo ejecutable resultante. La vinculación dinámica mantiene estas bibliotecas como archivos separados.

El enlace dinámico y el estático pueden compararse de varias maneras:

Uso de los recursos

La vinculación estática da lugar a archivos ejecutables más grandes que contienen más código. Este código adicional procedente de las bibliotecas no puede ser compartido por varios programas del sistema, lo que aumenta el uso del sistema de archivos y el uso de la memoria en tiempo de ejecución. Varios procesos que ejecuten el mismo programa enlazado estáticamente seguirán compartiendo el código.

Por otro lado, las aplicaciones estáticas necesitan menos reubicaciones en tiempo de ejecución, lo que reduce el tiempo de arranque, y requieren menos memoria privada de tamaño de conjunto residente (RSS). El código generado para la vinculación estática puede ser más eficiente que para la vinculación dinámica debido a la sobrecarga introducida por el código independiente de la posición (PIC).

Seguridad

Las bibliotecas enlazadas dinámicamente que proporcionan compatibilidad ABI pueden ser actualizadas sin cambiar los archivos ejecutables que dependen de estas bibliotecas. Esto es especialmente importante para las bibliotecas proporcionadas por Red Hat como parte de Red Hat Enterprise Linux, donde Red Hat proporciona actualizaciones de seguridad. La vinculación estática contra cualquiera de estas bibliotecas está fuertemente desaconsejada.

Compatibilidad

La vinculación estática parece proporcionar archivos ejecutables independientes de las versiones de las bibliotecas proporcionadas por el sistema operativo. Sin embargo, la mayoría de las bibliotecas dependen de otras. Con el enlazado estático, esta dependencia se vuelve inflexible y, como resultado, se pierde la compatibilidad hacia adelante y hacia atrás. La vinculación estática garantiza que sólo funciona en el sistema en el que se ha creado el archivo ejecutable.

AVISO

Las aplicaciones que enlazan estáticamente bibliotecas de la biblioteca GNU C (**glibc**) siguen necesitando que **glibc** esté presente en el sistema como biblioteca dinámica. Además, la variante de la biblioteca dinámica de **glibc** disponible en el tiempo de ejecución de la aplicación debe ser una versión idéntica en cuanto a bits a la presente mientras se enlaza la aplicación. Como resultado, se garantiza que la vinculación estática sólo funciona en el sistema en el que se construyó el archivo ejecutable.

Cobertura de apoyo

La mayoría de las bibliotecas estáticas proporcionadas por Red Hat están en el canal *CodeReady Linux Builder* y no son soportadas por Red Hat.

Funcionalidad

Algunas bibliotecas, especialmente la GNU C Library (**glibc**), ofrecen una funcionalidad reducida cuando se enlazan estáticamente.
Por ejemplo, cuando se vincula estáticamente, **glibc** no admite hilos ni ninguna forma de llamadas a la función **dlopen()** en el mismo programa.

Como resultado de las desventajas enumeradas, la vinculación estática debería evitarse a toda costa, especialmente para aplicaciones completas y las bibliotecas **glibc** y **libstdc** .

Casos de vinculación estática

La vinculación estática puede ser una opción razonable en algunos casos, por ejemplo:

- Utilizar una biblioteca que no está habilitada para el enlace dinámico.
- La vinculación totalmente estática puede ser necesaria para ejecutar el código en un entorno o contenedor vacío de **chroot**. Sin embargo, Red Hat no admite la vinculación estática mediante el paquete **glibc-static**.

Recursos adicionales

- Red Hat Enterprise Linux 8: Guía de compatibilidad de aplicaciones
- Descripción del repositorio CodeReady Linux Builder en el *Package manifest*

3.3. USO DE UNA BIBLIOTECA CON GCC

Una biblioteca es un paquete de código que puede reutilizarse en su programa. Una biblioteca en C o C está formada por dos partes:

- El código de la biblioteca
- Archivos de cabecera

Compilación de código que utiliza una biblioteca

Los archivos de cabecera describen la interfaz de la biblioteca: las funciones y variables disponibles en la biblioteca. La información de los archivos de cabecera es necesaria para compilar el código.

Normalmente, los archivos de cabecera de una biblioteca se colocan en un directorio diferente al del código de su aplicación. Para indicar a GCC dónde están los archivos de cabecera, utilice la opción **-I**:

```
$ gcc ... -Iinclude_path...
```

Sustituya *include_path* por la ruta real del directorio de archivos de cabecera.

La opción **-I** puede utilizarse varias veces para añadir varios directorios con archivos de cabecera. Al buscar un archivo de cabecera, estos directorios se buscan en el orden en que aparecen en las opciones de **-I**.

Vinculación de código que utiliza una biblioteca

Al enlazar el archivo ejecutable, deben estar disponibles tanto el código objeto de su aplicación como el código binario de la biblioteca. El código de las bibliotecas estáticas y dinámicas está presente en diferentes formas:

- Las bibliotecas estáticas están disponibles como ficheros de archivo. Contienen un grupo de archivos de objetos. El archivo tiene una extensión de nombre de archivo **.a**.
- Las bibliotecas dinámicas están disponibles como objetos compartidos. Son una forma de archivo ejecutable. Un objeto compartido tiene una extensión de nombre de archivo **.so**.

Para indicar a GCC dónde están los archivos o ficheros de objetos compartidos de una biblioteca, utilice la opción **-L**:

```
$ gcc ... -Llibrary_path -lfoo...
```

Sustituya *library_path* por la ruta real del directorio de la biblioteca.

La opción **-L** puede utilizarse varias veces para añadir varios directorios. Al buscar una biblioteca, estos directorios se buscan en el orden de sus opciones **-L**.

El orden de las opciones es importante: GCC no puede enlazar contra una biblioteca **foo** a menos que conozca el directorio con esta biblioteca. Por lo tanto, utilice las opciones **-L** para especificar los directorios de las bibliotecas antes de utilizar las opciones **-l** para enlazar con las bibliotecas.

Compilar y enlazar el código que utiliza una biblioteca en un solo paso

Cuando la situación permita compilar y enlazar el código en un solo comando **gcc**, utilice las opciones para las dos situaciones mencionadas anteriormente a la vez.

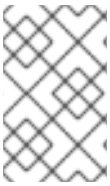
Recursos adicionales

- Uso de la colección de compiladores de GNU (GCC)

- Uso de la colección de compiladores de GNU (GCC)

3.4. USO DE UNA BIBLIOTECA ESTÁTICA CON GCC

Las bibliotecas estáticas están disponibles como archivos que contienen ficheros objeto. Después de enlazarlas, pasan a formar parte del archivo ejecutable resultante.

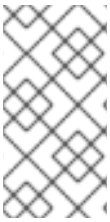


NOTA

Red Hat desaconseja el uso de enlaces estáticos por razones de seguridad. Consulte Sección 3.2, "Enlace estático y dinámico" . Utilice el enlazado estático sólo cuando sea necesario, especialmente contra las bibliotecas proporcionadas por Red Hat.

Requisitos previos

- GCC debe estar instalado en su sistema.
- Debe entender el enlace estático y dinámico.
- Usted tiene un conjunto de archivos fuente o de objetos que forman un programa válido, que requiere alguna biblioteca estática **foo** y ninguna otra biblioteca.
- La biblioteca **foo** está disponible como un archivo **libfoo.a**, y no se proporciona ningún archivo **libfoo.so** para la vinculación dinámica.



NOTA

La mayoría de las bibliotecas que forman parte de Red Hat Enterprise Linux están soportadas sólo para el enlazado dinámico. Los pasos siguientes sólo funcionan para las bibliotecas que están *not* habilitadas para el enlace dinámico. Vea Sección 3.2, "Enlace estático y dinámico".

Procedimiento

Para enlazar un programa a partir de los archivos fuente y objeto, añadiendo una biblioteca enlazada estáticamente **foo**, que se encuentra como un archivo **libfoo.a**:

1. Cambie al directorio que contiene su código.
2. Compilar los archivos fuente del programa con las cabeceras de la biblioteca **foo**:

   ```
   $ gcc ... -Iheader_path -c ...
   ```

   Sustituya *header_path* por una ruta a un directorio que contenga los archivos de cabecera de la biblioteca **foo**.
3. Enlaza el programa con la biblioteca **foo**:

   ```
   $ gcc ... -Llibrary_path -lfoo...
   ```

   Sustituya *library_path* por una ruta a un directorio que contenga el archivo **libfoo.a**.
4. Para ejecutar el programa más tarde, simplemente:

```
$ ./programa
```

ATENCIÓN

La opción de GCC **-static** relacionada con el enlazado estático prohíbe todo el enlazado dinámico. En su lugar, utilice las opciones **-Wl,-Bstatic** y **-Wl,-Bdynamic** para controlar el comportamiento del enlazador con mayor precisión. Véase Sección 3.6, “Uso de bibliotecas estáticas y dinámicas con GCC” .

3.5. USO DE UNA BIBLIOTECA DINÁMICA CON GCC

Las bibliotecas dinámicas están disponibles como archivos ejecutables independientes, necesarios tanto en tiempo de enlace como de ejecución. Son independientes del archivo ejecutable de la aplicación.

Requisitos previos

- GCC debe estar instalado en el sistema.
- Un conjunto de archivos fuente o de objetos que forman un programa válido, que requiere alguna biblioteca dinámica **foo** y ninguna otra biblioteca.
- La biblioteca **foo** debe estar disponible como archivo *libfoo.so*.

Enlazar un programa con una biblioteca dinámica

Para enlazar un programa con una biblioteca dinámica **foo**:

```
$ gcc ... -Llibrary_path -lfoo...
```

Cuando un programa se enlaza con una biblioteca dinámica, el programa resultante debe cargar siempre la biblioteca en tiempo de ejecución. Hay dos opciones para localizar la biblioteca:

- Utilizando un valor de **rpath** almacenado en el propio archivo ejecutable
- Uso de la variable **LD_LIBRARY_PATH** en tiempo de ejecución

Uso de un valor rpath almacenado en el archivo ejecutable

El **rpath** es un valor especial que se guarda como parte de un archivo ejecutable cuando se está enlazando. Más tarde, cuando el programa se cargue desde su archivo ejecutable, el enlazador en tiempo de ejecución utilizará el valor **rpath** para localizar los archivos de la biblioteca.

Al enlazar con **GCC**, para almacenar la ruta *library_path* como **rpath**:

```
$ gcc ... -Llibrary_path -lfoo -Wl,-rpath=library_path...
```

La ruta *library_path* debe apuntar a un directorio que contenga el archivo *libfoo.so*.

ATENCIÓN

No hay espacio después de la coma en la opción **-Wl,-rpath=**!

Para ejecutar el programa más tarde:

```
$ ./programa
```

Uso de la variable de entorno LD_LIBRARY_PATH

Si no se encuentra **rpath** en el archivo ejecutable del programa, el enlazador en tiempo de ejecución utilizará la variable de entorno **LD_LIBRARY_PATH**. El valor de esta variable debe cambiarse para cada programa. Este valor debe representar la ruta donde se encuentran los objetos de la biblioteca compartida.

Para ejecutar el programa sin **rpath** establecido, con las bibliotecas presentes en la ruta *library_path*:

```
$ export LD_LIBRARY_PATH=library_path:$LD_LIBRARY_PATH
$ ./program
```

La omisión del valor **rpath** ofrece flexibilidad, pero requiere la configuración de la variable **LD_LIBRARY_PATH** cada vez que se vaya a ejecutar el programa.

Colocación de la biblioteca en los directorios por defecto

La configuración del enlazador en tiempo de ejecución especifica una serie de directorios como ubicación por defecto de los archivos de las bibliotecas dinámicas. Para utilizar este comportamiento por defecto, copie su biblioteca en el directorio apropiado.

Una descripción completa del comportamiento del enlazador dinámico está fuera del alcance de este documento. Para más información, consulte los siguientes recursos:

- Páginas del manual de Linux para el enlazador dinámico:

  ```
  $ man ld.so
  ```

- Contenido del archivo de configuración **/etc/ld.so.conf**:

  ```
  $ cat /etc/ld.so.conf
  ```

- Informe de las bibliotecas reconocidas por el enlazador dinámico sin configuración adicional, que incluye los directorios:

  ```
  $ ldconfig -v
  ```

3.6. USO DE BIBLIOTECAS ESTÁTICAS Y DINÁMICAS CON GCC

A veces es necesario enlazar algunas bibliotecas estáticamente y otras dinámicamente. Esta situación conlleva algunos retos.

Requisitos previos

- Comprensión de la vinculación estática y dinámica

Introducción

gcc reconoce tanto las bibliotecas dinámicas como las estáticas. Cuando se encuentra la opción ***-lfoo*** **gcc** intentará primero localizar un objeto compartido (un archivo **.so**) que contenga una versión enlazada dinámicamente de la biblioteca **foo**, y luego buscará el archivo de almacenamiento (**.a**) que contenga una versión estática de la biblioteca. Así, las siguientes situaciones pueden resultar de esta búsqueda:

- Sólo se encuentra el objeto compartido, y **gcc** enlaza con él de forma dinámica.
- Sólo se encuentra el archivo, y **gcc** enlaza con él de forma estática.
- Se encuentran tanto el objeto compartido como el archivo, y por defecto, **gcc** selecciona la vinculación dinámica contra el objeto compartido.
- No se encuentra ni el objeto compartido ni el archivo, y la vinculación falla.

Debido a estas reglas, la mejor manera de seleccionar la versión estática o dinámica de una biblioteca para enlazarla es teniendo sólo la versión encontrada por **gcc**. Esto puede controlarse hasta cierto punto utilizando u omitiendo los directorios que contienen las versiones de las bibliotecas, al especificar las **-L*path*** opciones.

Además, dado que la vinculación dinámica es la predeterminada, la única situación en la que debe especificarse explícitamente la vinculación es cuando una biblioteca con ambas versiones presentes debe vincularse estáticamente. Hay dos resoluciones posibles:

- Especificación de las bibliotecas estáticas por ruta de archivo en lugar de la opción **-l**
- Uso de la opción **-Wl** para pasar opciones al enlazador

Especificación de las bibliotecas estáticas por archivo

Por lo general, se indica a **gcc** que enlace con la biblioteca **foo** con la opción **-l*foo*** opción. Sin embargo, es posible especificar la ruta completa del archivo **lib*foo*.a** que contiene la biblioteca:

```
$ gcc ... path/to/libfoo.a ...
```

Por la extensión del archivo **.a**, **gcc** entenderá que se trata de una biblioteca para enlazar con el programa. Sin embargo, especificar la ruta completa al archivo de la biblioteca es un método menos flexible.

Utilizando la opción -Wl

La opción **gcc -Wl** es una opción especial para pasar opciones al enlazador subyacente. La sintaxis de esta opción difiere de las otras opciones de **gcc**. La opción **-Wl** va seguida de una lista de opciones del enlazador separada por comas, mientras que otras opciones de **gcc** requieren una lista de opciones separada por espacios.

El enlazador **ld** utilizado por **gcc** ofrece las opciones **-Bstatic** y **-Bdynamic** para especificar si las bibliotecas que siguen a esta opción deben ser enlazadas estática o dinámicamente, respectivamente. Después de pasar **-Bstatic** y una biblioteca al enlazador, se debe restaurar manualmente el comportamiento de enlace dinámico por defecto para que las siguientes bibliotecas se enlacen dinámicamente con la opción **-Bdynamic**.

Para enlazar un programa, vincule la biblioteca **first** estáticamente (**libfirst.a**) y **second** dinámicamente (**libsecond.so**):

```
$ gcc ... -Wl,-Bstatic -l first -Wl,-Bdynamic -l second ...
```

NOTA

gcc puede configurarse para utilizar enlazadores distintos del predeterminado **ld**.

Recursos adicionales

- Uso de la colección de compiladores de GNU (GCC)
- Documentación de binutils 2.27

CAPÍTULO 4. CREACIÓN DE BIBLIOTECAS CON GCC

Este capítulo describe los pasos para crear bibliotecas y explica los conceptos necesarios que utiliza el sistema operativo Linux para las bibliotecas.

4.1. CONVENCIONES DE NOMENCLATURA DE LAS BIBLIOTECAS

Se utiliza una convención especial de nombres de archivos para las bibliotecas: se espera que una biblioteca conocida como **foo** exista como archivo **lib*foo*.so** o **lib*foo*.a**. Esta convención es entendida automáticamente por las opciones de entrada de enlace de GCC, pero no por las opciones de salida:

- Cuando se enlaza con la biblioteca, ésta sólo puede especificarse por su nombre **foo** con la opción **-l** como **-l*foo***:

```
$ gcc ... -lfoo...
```

- Al crear la biblioteca, el nombre completo del archivo **lib*foo*.so** o **lib*foo*.a** debe especificarse.

Recursos adicionales

- Sección 4.2, “El mecanismo de soname”

4.2. EL MECANISMO DE SONAME

Las bibliotecas cargadas dinámicamente (objetos compartidos) utilizan un mecanismo llamado *soname* para gestionar múltiples versiones compatibles de una biblioteca.

Requisitos previos

- Debe entender el enlace dinámico y las bibliotecas.
- Debes entender el concepto de compatibilidad ABI.
- Debes entender las convenciones de nomenclatura de las bibliotecas.
- Debes entender los enlaces simbólicos.

Introducción del problema

Una biblioteca cargada dinámicamente (objeto compartido) existe como un archivo ejecutable independiente. Esto permite actualizar la biblioteca sin actualizar las aplicaciones que dependen de ella. Sin embargo, este concepto plantea los siguientes problemas:

- Identificación de la versión actual de la biblioteca
- Necesidad de contar con varias versiones de la misma biblioteca
- Señalización de la compatibilidad ABI de cada una de las múltiples versiones

El mecanismo de soname

Para resolver esto, Linux utiliza un mecanismo llamado soname.

Una biblioteca **foo** versión *X.Y* es ABI-compatible con otras versiones con el mismo valor de *X* en un número de versión. Los cambios menores que preservan la compatibilidad aumentan el número *Y*. Los cambios mayores que rompen la compatibilidad aumentan el número *X*.

La versión real de la biblioteca **foo** *X.Y* existe como un archivo **libfoo.so.*x.y***. Dentro del archivo de la biblioteca, se registra un soname con el valor **libfoo.so.x** para señalar la compatibilidad.

Cuando se construyen las aplicaciones, el enlazador busca la biblioteca buscando el archivo **libfoo.so**. Debe existir un enlace simbólico con este nombre, que apunte al archivo real de la biblioteca. A continuación, el enlazador lee el nombre del archivo de la biblioteca y lo registra en el archivo ejecutable de la aplicación. Por último, el enlazador crea la aplicación que declara la dependencia de la biblioteca utilizando el soname, no un nombre o un nombre de archivo.

Cuando el enlazador dinámico en tiempo de ejecución enlaza una aplicación antes de ejecutarla, lee el soname del archivo ejecutable de la aplicación. Este soname es **libfoo.so.*x***. Debe existir un enlace simbólico con este nombre, que apunte al archivo real de la biblioteca. Esto permite cargar la biblioteca, independientemente del componente *Y* de una versión, porque el soname no cambia.

NOTA

El componente *Y* del número de versión no se limita a un solo número. Además, algunas bibliotecas codifican su versión en su nombre.

Lectura del nombre de un archivo

Para mostrar el nombre de un archivo de la biblioteca **somelibrary**:

```
$ objdump -p somelibrary | grep SONAME
```

Sustituya *somelibrary* por el nombre de archivo real de la biblioteca que desea examinar.

4.3. CREACIÓN DE BIBLIOTECAS DINÁMICAS CON GCC

Las bibliotecas enlazadas dinámicamente (objetos compartidos) permiten:

- conservación de recursos mediante la reutilización de códigos
- mayor seguridad al facilitar la actualización del código de la biblioteca

Siga estos pasos para construir e instalar una biblioteca dinámica desde el código fuente.

Requisitos previos

- Debes entender el mecanismo de soname.
- GCC debe estar instalado en el sistema.
- Debe tener el código fuente de una biblioteca.

Procedimiento

1. Cambie al directorio con las fuentes de la biblioteca.
2. Compile cada archivo fuente en un archivo objeto con la opción de código independiente de la posición **-fPIC**:

```
$ gcc ... -c -fPIC some_file.c...
```

Los archivos objeto tienen los mismos nombres de archivo que los del código fuente original, pero su extensión es **.o**.

3. Enlaza la biblioteca compartida desde los archivos de objetos:

```
$ gcc -shared -o libfoo.so.x.y -Wl,-soname,libfoo.so.x some_file.o ...
```

El número de versión mayor utilizado es X y el número de versión menor Y.

4. Copie el archivo ***libfoo.so.x.y*** en una ubicación apropiada, donde el enlazador dinámico del sistema pueda encontrarlo. En Red Hat Enterprise Linux, el directorio para las bibliotecas es **/usr/lib64**:

```
# cp libfoo.so.x.y /usr/lib64
```

Tenga en cuenta que necesita permisos de root para manipular los archivos de este directorio.

5. Crear la estructura de enlaces simbólicos para el mecanismo de soname:

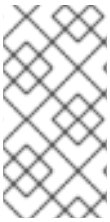
```
# ln -s libfoo.so.x.y libfoo.so.x
# ln -s libfoo.so.x libfoo.so
```

Recursos adicionales

- El proyecto de documentación de Linux

4.4. CREACIÓN DE BIBLIOTECAS ESTÁTICAS CON GCC Y AR

La creación de bibliotecas para la vinculación estática es posible mediante la conversión de los archivos de objetos en un tipo especial de archivo.



NOTA

Red Hat desaconseja el uso de la vinculación estática por razones de seguridad. Utilice la vinculación estática sólo cuando sea necesario, especialmente contra las bibliotecas proporcionadas por Red Hat. Consulte Sección 3.2, “Enlace estático y dinámico” para más detalles.

Requisitos previos

- GCC y binutils deben estar instalados en el sistema.
- Debe entender el enlace estático y dinámico.
- Se dispone de archivo(s) de origen con funciones para compartir como biblioteca.

Procedimiento

1. Crear archivos de objetos intermedios con GCC.

```
$ gcc -c source_file.c...
```

Añada más archivos fuente si es necesario. Los archivos objeto resultantes comparten el nombre del archivo pero utilizan la extensión de nombre de archivo **.o**.

2. Convierta los ficheros objeto en una biblioteca estática (archivo) utilizando la herramienta **ar** del paquete **binutils**.

```
$ ar rcs libfoo.a source_file.o...
```

Se crea el archivo **libfoo.a**.

3. Utilice el comando **nm** para inspeccionar el archivo resultante:

```
$ nm libfoo.a
```

4. Copie el archivo de la biblioteca estática en el directorio correspondiente.

5. Al enlazar con la biblioteca, GCC reconocerá automáticamente, a partir de la extensión del nombre del archivo **.a**, que la biblioteca es un archivo para el enlace estático.

```
$ gcc ... -lfoo...
```

Recursos adicionales

- Página del manual de Linux para *ar(1)*:

```
$ man ar
```

CAPÍTULO 5. GESTIONAR MÁS CÓDIGO CON MAKE

La utilidad GNU make, comúnmente abreviada **make**, es una herramienta para controlar la generación de ejecutables a partir de archivos fuente. **make** determina automáticamente qué partes de un programa complejo han cambiado y necesitan ser recompiladas. **make** utiliza archivos de configuración llamados Makefiles para controlar la forma en que se construyen los programas.

5.1. GNU MAKE Y MAKEFILE RESUMEN

Para crear una forma utilizable (normalmente archivos ejecutables) a partir de los archivos fuente de un proyecto concreto, realice varios pasos necesarios. Registre las acciones y su secuencia para poder repetirlas más adelante.

Red Hat Enterprise Linux contiene GNU **make**, un sistema de construcción diseñado para este propósito.

Requisitos previos

- Comprender los conceptos de compilación y vinculación

GNU make

GNU **make** lee Makefiles que contienen las instrucciones que describen el proceso de construcción. Un Makefile contiene múltiples *rules* que describen una forma de satisfacer una determinada condición (*target*) con una acción específica (*recipe*). Las reglas pueden depender jerárquicamente de otra regla.

Ejecutar **make** sin ninguna opción hace que busque un Makefile en el directorio actual e intente alcanzar el objetivo por defecto. El nombre del archivo Makefile puede ser uno de los siguientes: **Makefile**, **makefile**, y **GNUmakefile**. El objetivo por defecto se determina a partir del contenido del Makefile.

Detalles del Makefile

Los Makefiles utilizan una sintaxis relativamente sencilla para definir *variables* y *rules*, que consiste en un *target* y un *recipe*. El objetivo especifica cuál es la salida si se ejecuta una regla. Las líneas con recetas deben comenzar con el carácter TAB.

Típicamente, un Makefile contiene reglas para compilar archivos fuente, una regla para enlazar los archivos objeto resultantes, y un objetivo que sirve como punto de entrada en la parte superior de la jerarquía.

Considere el siguiente **Makefile** para construir un programa en C que consiste en un solo archivo, **hello.c**.

```
all: hello

hello: hello.o
	gcc hello.o -o hello

hello.o: hello.c
	gcc -c hello.c -o hello.o
```

Este ejemplo muestra que para llegar al objetivo **all**, se necesita el archivo **hello**. Para obtener **hello**, se necesita **hello.o** (enlazado por **gcc**), que a su vez se crea a partir de **hello.c** (compilado por **gcc**).

El objetivo **all** es el objetivo por defecto porque es el primer objetivo que no comienza con un punto (.). Ejecutar **make** sin ningún argumento es entonces idéntico a ejecutar **make all**, cuando el directorio actual contiene este **Makefile**.

Típico makefile

Un Makefile más típico utiliza variables para la generalización de los pasos y añade un objetivo "clean" - eliminar todo excepto los archivos de origen.

```
CC=gcc
CFLAGS=-c -Wall
SOURCE=hello.c
OBJ=$(SOURCE:.c=.o)
EXE=hello

all: $(SOURCE) $(EXE)

$(EXE): $(OBJ)
	$(CC) $(OBJ) -o $@

%.o: %.c
	$(CC) $(CFLAGS) $< -o $@

clean:
	rm -rf $(OBJ) $(EXE)
```

Para añadir más archivos fuente a dicho Makefile sólo es necesario añadirlos a la línea donde se define la variable SOURCE.

Recursos adicionales

- GNU make: Introducción
- Capítulo 2, *Código de construcción con GCC*

5.2. EJEMPLO: CONSTRUIR UN PROGRAMA EN C USANDO UN MAKEFILE

Construya un programa C de ejemplo utilizando un Makefile siguiendo los pasos de este ejemplo.

Requisitos previos

- Debes entender los conceptos de Makefiles y **make**.

Procedimiento

1. Cree un directorio **hellomake** y cambie a este directorio:

```
$ mkdir hellomake
$ cd hellomake
```

2. Cree un archivo **hello.c** con el siguiente contenido:

```
#include <stdio.h>
```

```
int main(int argc, char *argv[]) {
  printf("Hello, World!\n");
  return 0;
}
```

3. Cree un archivo **Makefile** con el siguiente contenido:

```
CC=gcc
CFLAGS=-c -Wall
SOURCE=hello.c
OBJ=$(SOURCE:.c=.o)
EXE=hello

all: $(SOURCE) $(EXE)

$(EXE): $(OBJ)
	$(CC) $(OBJ) -o $@

%.o: %.c
	$(CC) $(CFLAGS) $< -o $@

clean:
	rm -rf $(OBJ) $(EXE)
```

ATENCIÓN

Las líneas de la receta del Makefile deben comenzar con el carácter de tabulación Al copiar el texto anterior de la documentación, el proceso de cortar y pegar puede pegar espacios en lugar de tabulaciones. Si esto ocurre, corrija el problema manualmente.

4. Ejecutar **make**:

```
$ make
gcc -c -Wall hello.c -o hello.o
gcc hello.o -o hello
```

Esto crea un archivo ejecutable **hello**.

5. Ejecute el archivo ejecutable **hello**:

```
$ ./hello
Hello, World!
```

6. Ejecute el objetivo de Makefile **clean** para eliminar los archivos creados:

```
$ make clean
rm -rf hello.o hello
```

Recursos adicionales

- Sección 2.7, “Ejemplo: Construir un programa en C con GCC”

- Sección 2.8, “Ejemplo: Construir un programa en C con GCC”

5.3. RECURSOS DE DOCUMENTACIÓN PARA MAKE

Para más información sobre **make**, consulte los recursos que se indican a continuación.

Documentación instalada

- Utilice las herramientas **man** y **info** para ver las páginas de los manuales y las páginas de información instaladas en su sistema:

```
$ man make
$ info make
```

Documentación en línea

- El manual de GNU Make alojado en la Fundación para el Software Libre
- Manual del usuario del conjunto de herramientas para desarrolladores de Red Hat

CAPÍTULO 6. CAMBIOS EN LA CADENA DE HERRAMIENTAS DESDE RHEL 7

The following sections list changes in toolchain since the release of the described components in Red Hat Enterprise Linux 7. See also Release notes for Red Hat Enterprise Linux 8.0 .

6.1. CAMBIOS EN GCC EN RHEL 8

En Red Hat Enterprise Linux 8, la cadena de herramientas GCC está basada en la serie de versiones GCC 8.2. Los cambios notables desde Red Hat Enterprise Linux 7 incluyen:

- Se han añadido numerosas optimizaciones generales, como el análisis de alias, las mejoras del vectorizador, el plegado de código idéntico, el análisis interprocedimental, el pase de optimización de la fusión de almacenes y otras.
- Se ha mejorado el desinfectante de direcciones.
- Se ha añadido el Sanitizador de fugas para la detección de fugas de memoria.
- Se ha añadido el Sanitizador de Comportamientos Indefinidos para la detección de comportamientos indefinidos.
- Ahora se puede producir información de depuración en el formato DWARF5. Esta capacidad es experimental.
- La herramienta de análisis de cobertura del código fuente GCOV se ha ampliado con varias mejoras.
- Se ha añadido soporte para la especificación OpenMP 4.5. Además, las características de descarga de la especificación OpenMP 4.0 son ahora compatibles con los compiladores de C, C , y Fortran.
- Se han añadido nuevas advertencias y diagnósticos mejorados para la detección estática de ciertos errores probables de programación.
- Las ubicaciones de las fuentes ahora se rastrean como rangos en lugar de puntos, lo que permite un diagnóstico mucho más rico. El compilador ofrece ahora pistas de "arreglo", sugiriendo posibles modificaciones del código. Se ha añadido un corrector ortográfico para ofrecer nombres alternativos y facilitar la detección de errores tipográficos.

Seguridad

GCC se ha ampliado para proporcionar herramientas que garanticen un endurecimiento adicional del código generado.

Para más detalles, consulte Sección 6.2, “Mejoras de seguridad en GCC en RHEL 8” .

Arquitectura y soporte del procesador

Las mejoras en la arquitectura y el soporte del procesador incluyen:

- Se han añadido múltiples opciones nuevas específicas para la arquitectura Intel AVX-512, varias de sus microarquitecturas y las extensiones de protección de software de Intel (SGX).

- La generación de código ahora puede dirigirse a las extensiones LSE de la arquitectura ARM de 64 bits, a las extensiones de punto flotante (FPE) de 16 bits de ARMv8.2-A y a las versiones de la arquitectura ARMv8.2-A, ARMv8.3-A y ARMv8.4-A.

- Se ha corregido la gestión de la opción **-march=native** en las arquitecturas ARM y ARM de 64 bits.

- Se ha añadido soporte para los procesadores z13 y z14 de la arquitectura IBM Z.

Lenguas y normas

Entre los cambios notables relacionados con las lenguas y las normas se encuentran:

- El estándar por defecto utilizado al compilar código en el lenguaje C ha cambiado a C17 con extensiones GNU.

- El estándar por defecto utilizado al compilar código en el lenguaje C ha cambiado a C 14 con extensiones GNU.

- La biblioteca de tiempo de ejecución de C es ahora compatible con los estándares C 11 y C 14.

- El compilador de C ahora implementa el estándar C 14 con muchas características nuevas, como plantillas de variables, agregados con inicializadores de miembros de datos no estáticos, el especificador extendido **constexpr**, funciones de desasignación de tamaño, lambdas genéricas, matrices de longitud variable, separadores de dígitos y otros.

- Se ha mejorado el soporte del lenguaje C estándar C11: Ahora están disponibles los atómicos ISO C11, las selecciones genéricas y el almacenamiento local de hilos.

- La nueva extensión **__auto_type** GNU C proporciona un subconjunto de la funcionalidad de la palabra clave C 11 **auto** en el lenguaje C.

- Los nombres de los tipos **_FloatN** y **_FloatNx** especificados por la norma ISO/IEC TS 18661-3:2015 son ahora reconocidos por el front end de C.

- El estándar por defecto utilizado al compilar código en el lenguaje C ha cambiado a C17 con extensiones GNU. Esto tiene el mismo efecto que utilizar la opción **--std=gnu17**. Anteriormente, el estándar por defecto era C89 con extensiones GNU.

- GCC puede ahora compilar experimentalmente código utilizando el estándar del lenguaje C 17 y ciertas características del estándar C 20.

- Pasar una clase vacía como argumento ahora no ocupa espacio en las arquitecturas Intel 64 y AMD64, como lo requiere la ABI de la plataforma. Pasar o devolver una clase con sólo constructores de copia y movimiento eliminados ahora utiliza la misma convención de llamada que una clase con un constructor de copia o movimiento no trivial.

- El valor devuelto por el operador C 11 **alignof** ha sido corregido para que coincida con el operador C **_Alignof** y devuelva la alineación mínima. Para encontrar la alineación preferida, utilice la extensión GNU **__alignof__**.

- La versión principal de la biblioteca **libgfortran** para el código del lenguaje Fortran se ha cambiado a la 5.

- Se ha eliminado el soporte para los lenguajes Ada (GNAT), GCC Go y Objective C/C. Utilice el conjunto de herramientas Go para el desarrollo de código Go.

Recursos adicionales

- Consulte también las Notas de la versión de Red Hat Enterprise Linux 8 .
- Uso de Go Toolset

6.2. MEJORAS DE SEGURIDAD EN GCC EN RHEL 8

This section decribes in detail the changes in GCC related to security and added since the release of Red Hat Enterprise Linux 7.0.

Nuevas advertencias

Se han añadido estas opciones de advertencia:

Opción	Muestra avisos para
-Wstringop-truncation	Llamadas a funciones de manipulación de cadenas acotadas como **strncat**, **strncpy**, y **stpncpy** que pueden truncar la cadena copiada o dejar el destino sin cambios.
-Wclass-memaccess	Objetos de tipos de clase no triviales manipulados de forma potencialmente insegura por funciones de memoria bruta como **memcpy** o **realloc**. La advertencia ayuda a detectar las llamadas que evitan los constructores definidos por el usuario o los operadores de asignación de copias, los punteros de tablas virtuales corruptas, los miembros de datos de tipos o referencias calificados como const, o los punteros de miembros. La advertencia también detecta las llamadas que podrían eludir los controles de acceso a los miembros de datos.
-Wmisleading-indentation	Lugares donde la sangría del código da una idea errónea de la estructura de bloques del código a un lector humano.
-Walloc-size-larger-than=*size*	Llamadas a funciones de asignación de memoria en las que la cantidad de memoria a asignar supera *size*. También funciona con funciones en las que la asignación se especifica multiplicando dos parámetros y con cualquier función decorada con el atributo **alloc_size**.
-Walloc-zero	Llamadas a funciones de asignación de memoria que intentan asignar una cantidad cero de memoria. También funciona con funciones en las que la asignación se especifica multiplicando dos parámetros y con cualquier función decorada con el atributo **alloc_size**.

Opción	Muestra avisos para
-Walloca	Todas las llamadas a la función **alloca**.
-Walloca-larger-than=*size*	Llamadas a la función **alloca** cuando la memoria solicitada es superior a *size*.
-Wvla-larger-than=*size*	Definiciones de matrices de longitud variable (VLA) que pueden superar el tamaño especificado o cuyo límite no se sabe si está suficientemente restringido.
-Wformat-overflow=*level*	Desbordamiento de búfer seguro y probable en las llamadas a la familia de funciones de salida formateada **sprintf**. Para más detalles y explicación del valor *level*, consulte la página del manual*gcc(1)*.
-Wformat-truncation=*level*	Truncamiento de la salida, tanto seguro como probable, en las llamadas a la familia de funciones de salida formateada **snprintf**. Para más detalles y explicación del valor *level*, consulte la página del manual *gcc(1)*.
-Wstringop-overflow=*type*	Desbordamiento del búfer en las llamadas a funciones de manejo de cadenas como **memcpy** y **strcpy**. Para más detalles y explicación del valor *level*, consulte la página del manual*gcc(1)*.

Advertencia de mejoras

Estas advertencias de GCC han sido mejoradas:

- La opción **-Warray-bounds** se ha mejorado para detectar más casos de índices de matrices y desplazamientos de punteros fuera de los límites. Por ejemplo, se detectan los índices negativos o excesivos en miembros de matrices flexibles y literales de cadena.
- La opción **-Wrestrict** introducida en GCC 7 ha sido mejorada para detectar muchos más casos de accesos solapados a objetos a través de argumentos restringidos a funciones estándar de manipulación de memoria y cadenas como **memcpy** y **strcpy**.
- La opción **-Wnonnull** se ha mejorado para detectar un conjunto más amplio de casos de paso de punteros nulos a funciones que esperan un argumento no nulo (decorado con el atributo **nonnull**).

Nuevo UndefinedBehaviorSanitizer

Se ha añadido un nuevo sanitizador en tiempo de ejecución para detectar comportamientos indefinidos llamado UndefinedBehaviorSanitizer. Cabe destacar las siguientes opciones:

Opción	Consulte
-fsanitize=float-divide-by-zero	Detecta la división por cero en coma flotante.

Opción	Consulte
-fsanitize=float-cast-overflow	Comprueba que el resultado de las conversiones de tipo punto flotante a entero no se desborda.
-fsanitize=bounds	Habilitar la instrumentación de los límites de la matriz y detectar los accesos fuera de los límites.
-fsanitize=alignment	Habilita la comprobación de la alineación y detecta varios objetos desalineados.
-fsanitize=object-size	Habilitar la comprobación del tamaño de los objetos y detectar varios accesos fuera de los límites.
-fsanitize=vptr	Permitir la comprobación de las llamadas a funciones miembro de C, los accesos a miembros y algunas conversiones entre punteros a clases base y derivadas. Además, detecta cuando los objetos referenciados no tienen el tipo dinámico correcto.
-fsanitize=bounds-strict	Habilitar la comprobación estricta de los límites de los arrays. Esto permite **-fsanitize=bounds** y la instrumentación de matrices flexibles similares a los miembros del array.
-fsanitize=signed-integer-overflow	Diagnosticar desbordamientos aritméticos incluso en operaciones aritméticas con vectores genéricos.
-fsanitize=builtin	Diagnostica en tiempo de ejecución los argumentos no válidos de los buildins prefijados en **__builtin_clz** o **__builtin_ctz**. Incluye comprobaciones de **-fsanitize=undefined**.
-fsanitize=pointer-overflow	Realiza pruebas baratas en tiempo de ejecución para la envoltura de punteros. Incluye comprobaciones de **-fsanitize=undefined**.

Nuevas opciones para AddressSanitizer

Estas opciones se han añadido a AddressSanitizer:

Opción	Consulte
-fsanitize=pointer-compare	Advierte sobre la comparación de punteros que apuntan a un objeto de memoria diferente.
-fsanitize=pointer-subtract	Advierte sobre la sustracción de punteros que apuntan a un objeto de memoria diferente.

Opción	Consulte
-fsanitize-address-use-after-scope	Sanear las variables cuya dirección se toma y se utiliza después de un ámbito donde se define la variable.

Otros desinfectantes e instrumentos

- Se ha añadido la opción **-fstack-clash-protection** para insertar sondas cuando el espacio de la pila se asigna de forma estática o dinámica para detectar de forma fiable los desbordamientos de la pila y así mitigar el vector de ataque que se basa en el salto de una página de guarda de la pila proporcionada por el sistema operativo.
- Se ha añadido una nueva opción **-fcf-protection=[full|branch|return|none]** para realizar la instrumentación del código y aumentar la seguridad del programa comprobando que las direcciones de destino de las instrucciones de transferencia de flujo de control (como la llamada indirecta a una función, el retorno de una función o el salto indirecto) son válidas.

Recursos adicionales

- Para más detalles y explicación de los valores suministrados a algunas de las opciones anteriores, consulte la página del manual *gcc(1)*:

```
$ man gcc
```

6.3. CAMBIOS QUE ROMPEN LA COMPATIBILIDAD EN GCC EN RHEL 8

C Cambio de ABI en std::string y std::list

La interfaz binaria de aplicación (ABI) de las clases **std::string** y **std::list** de la biblioteca **libstdc** cambió entre RHEL 7 (GCC 4.8) y RHEL 8 (GCC 8) para ajustarse al estándar C 11. La biblioteca **libstdc** soporta tanto la antigua como la nueva ABI, pero algunas otras bibliotecas del sistema C no lo hacen. Como consecuencia, las aplicaciones que enlazan dinámicamente con estas bibliotecas tendrán que ser reconstruidas. Esto afecta a todos los modos estándar de C, incluyendo C 98. También afecta a las aplicaciones creadas con los compiladores de Red Hat Developer Toolset para RHEL 7, que mantuvieron la antigua ABI para mantener la compatibilidad con las bibliotecas del sistema.

GCC ya no construye código Ada, Go y Objective C/C

Se ha eliminado del compilador GCC la capacidad de construir código en los lenguajes Ada (GNAT), GCC Go y Objective C/C.

Para construir código Go, utilice el conjunto de herramientas Go.

PARTE II. DEPURACIÓN DE APLICACIONES

La depuración de aplicaciones es un tema muy amplio. Esta parte proporciona a un desarrollador las técnicas más comunes para depurar en múltiples situaciones.

CAPÍTULO 7. ACTIVACIÓN DE LA DEPURACIÓN CON INFORMACIÓN DE DEPURACIÓN

Para depurar las aplicaciones y las bibliotecas, se necesita información de depuración. Las siguientes secciones describen cómo obtener esta información.

7.1. INFORMACIÓN DE DEPURACIÓN

Al depurar cualquier código ejecutable, hay dos tipos de información que permiten a las herramientas, y por extensión al programador, comprender el código binario:

- el texto del código fuente
- una descripción de la relación entre el texto del código fuente y el código binario

Esta información se llama información de depuración.

Red Hat Enterprise Linux utiliza el formato ELF para binarios ejecutables, bibliotecas compartidas o archivos **debuginfo**. Dentro de estos archivos ELF, el formato DWARF se utiliza para mantener la información de depuración.

Para mostrar la información DWARF almacenada en un archivo ELF, ejecute el comando **readelf -w *file*** comando.

ATENCIÓN

STABS es un formato más antiguo y menos capaz, utilizado ocasionalmente con UNIX. Red Hat desaconseja su uso. GCC y GDB proporcionan la producción y el consumo de STABS sólo en base al mejor esfuerzo. Algunas otras herramientas como Valgrind y **elfutils** no funcionan con STABS.

Recursos adicionales

- El estándar de depuración DWARF

7.2. HABILITACIÓN DE LA DEPURACIÓN DE APLICACIONES C Y C++ CON GCC

Debido a que la información de depuración es grande, no se incluye por defecto en los archivos ejecutables. Para habilitar la depuración de tus aplicaciones C y C++ con ella, debes instruir explícitamente al compilador para que la cree.

Para permitir la creación de información de depuración con **GCC** al compilar y enlazar el código, utilice la opción **-g**:

```
$ gcc ... -g ...
```

- Las optimizaciones realizadas por el compilador y el enlazador pueden dar lugar a un código ejecutable difícil de relacionar con el código fuente original: las variables pueden ser optimizadas, los bucles desenrollados, las operaciones fusionadas con las circundantes, etc. Esto afecta negativamente a la depuración. Para mejorar la experiencia de depuración, considere la posibilidad de configurar la optimización con la opción **-Og**. Sin embargo, cambiar el nivel de optimización cambia el código ejecutable y puede cambiar el comportamiento real, incluyendo la eliminación de algunos errores.

- Para incluir también las definiciones de macros en la información de depuración, utilice la opción **-g3** en lugar de **-g**.
- La opción **-fcompare-debug** GCC comprueba el código compilado por GCC con información de depuración y sin información de depuración. La prueba pasa si los dos archivos binarios resultantes son idénticos. Esta prueba garantiza que el código ejecutable no se ve afectado por ninguna opción de depuración, lo que garantiza además que no hay errores ocultos en el código de depuración. Tenga en cuenta que el uso de la opción **-fcompare-debug** aumenta significativamente el tiempo de compilación. Consulte la página del manual de GCC para obtener detalles sobre esta opción.

Recursos adicionales

- Capítulo 7, *Activación de la depuración con información de depuración*
- Uso de la colección de compiladores de GNU (GCC)
- Depuración con GDB
- La página del manual de GCC:

```
$ man gcc
```

7.3. PAQUETES DEBUGINFO Y DEBUGSOURCE

Los paquetes **debuginfo** y **debugsource** contienen información de depuración y código fuente de depuración para programas y bibliotecas. Para las aplicaciones y bibliotecas instaladas en paquetes de los repositorios de Red Hat Enterprise Linux, puede obtener los paquetes **debuginfo** y **debugsource** por separado desde un canal adicional.

Tipos de paquetes de información de depuración

Hay dos tipos de paquetes disponibles para la depuración:

Paquetes de depuración

Los paquetes **debuginfo** proporcionan la información de depuración necesaria para proporcionar nombres legibles para las características del código binario. Estos paquetes contienen archivos **.debug**, que contienen información de depuración DWARF. Estos archivos se instalan en el directorio **/usr/lib/debug**.

Paquetes Debugsource

Los paquetes **debugsource** contienen los archivos fuente utilizados para compilar el código binario. Con los respectivos paquetes **debuginfo** y **debugsource** instalados, depuradores como GDB o LLDB pueden relacionar la ejecución del código binario con el código fuente. Los archivos del código fuente se instalan en el directorio **/usr/src/debug**.

Diferencias con RHEL 7

En Red Hat Enterprise Linux 7, los paquetes **debuginfo** contenían ambos tipos de información. Red Hat Enterprise Linux 8 divide los datos del código fuente necesarios para la depuración de los paquetes **debuginfo** en paquetes separados **debugsource**.

Nombres de los paquetes

Un paquete **debuginfo** o **debugsource** proporciona información de depuración válida sólo para un paquete binario con el mismo nombre, versión, lanzamiento y arquitectura:

- Paquete binario ***packagename-version-release.architecture*.rpm**
- Paquete Debuginfo ***packagename*-debuginfo-*version-release.architecture*.rpm**
- Paquete Debugsource ***packagename*-debugsource-*version-release.architecture*.rpm**

Recursos adicionales

- Sección 7.1, "Información de depuración"
- Sección 1.2, "Habilitación de los repositorios de depuración y de código fuente"

7.4. OBTENCIÓN DE PAQUETES DEBUGINFO PARA UNA APLICACIÓN O LIBRERÍA USANDO GDB

La información de depuración es necesaria para depurar el código. Para el código que se instala desde un paquete, el depurador de GNU (GDB) reconoce automáticamente la información de depuración que falta, resuelve el nombre del paquete y proporciona consejos concretos sobre cómo obtener el paquete.

Requisitos previos

- La aplicación o la biblioteca que desea depurar debe estar instalada en el sistema.
- GDB y la herramienta **debuginfo-install** deben estar instalados en el sistema.
- Los canales que proporcionan los paquetes **debuginfo** y **debugsource** deben estar configurados y habilitados en el sistema.

Procedimiento

1. Inicie GDB conectado a la aplicación o biblioteca que desea depurar. GDB reconoce automáticamente la información de depuración que falta y sugiere un comando a ejecutar.

```
$ gdb -q /bin/ls
Reading symbols from /bin/ls...Reading symbols from .gnu_debugdata for /usr/bin/ls...(no
debugging symbols found)...done.
(no debugging symbols found)...done.
Missing separate debuginfos, use: dnf debuginfo-install coreutils-8.30-6.el8.x86_64
(gdb)
```

2. Salir de GDB: escribir **q** y confirmar con **Enter**.

```
(gdb) q
```

3. Ejecute el comando sugerido por GDB para instalar los paquetes necesarios de **debuginfo**:

```
# dnf debuginfo-install coreutils-8.30-6.el8.x86_64
```

   La herramienta de gestión de paquetes **dnf** ofrece un resumen de los cambios, pide confirmación y, una vez confirmada, descarga e instala todos los archivos necesarios.

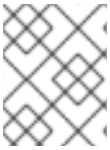
4. En caso de que GDB no pueda sugerir el paquete **debuginfo**, siga el procedimiento descrito en Sección 7.5, "Obtener manualmente los paquetes debuginfo de una aplicación o biblioteca" .

Recursos adicionales

- Manual del usuario de Red Hat Developer Toolset, sección Instalación de la información de depuración
- ¿Cómo puedo descargar o instalar paquetes debuginfo para sistemas RHEL?

7.5. OBTENER MANUALMENTE LOS PAQUETES DEBUGINFO DE UNA APLICACIÓN O BIBLIOTECA

Puede determinar manualmente qué paquetes de **debuginfo** necesita instalar localizando el archivo ejecutable y luego encontrando el paquete que lo instala.



NOTA

Red Hat recomienda que utilice GDB para determinar los paquetes a instalar. Utilice este procedimiento manual sólo si GDB no es capaz de sugerir el paquete a instalar.

Requisitos previos

- La aplicación o biblioteca debe estar instalada en el sistema.
- La aplicación o biblioteca se instaló desde un paquete.
- La herramienta **debuginfo-install** debe estar disponible en el sistema.
- Los canales que proporcionan los paquetes **debuginfo** deben estar configurados y habilitados en el sistema.

Procedimiento

1. Encuentra el archivo ejecutable de la aplicación o biblioteca.
   a. Utilice el comando **which** para encontrar el archivo de la aplicación.

   ```
   $ which less
   /usr/bin/less
   ```

   b. Utilice el comando **locate** para encontrar el archivo de la biblioteca.

   ```
   $ locate libz | grep so
   /usr/lib64/libz.so.1
   /usr/lib64/libz.so.1.2.11
   ```

   Si los motivos originales de la depuración incluyen mensajes de error, elija el resultado en el que la biblioteca tiene los mismos números adicionales en su nombre de archivo que los mencionados en los mensajes de error. En caso de duda, intente seguir el resto del procedimiento con el resultado en el que el nombre del archivo de la biblioteca no incluye números adicionales.

NOTA

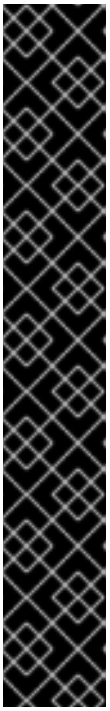
El comando **locate** es proporcionado por el paquete **mlocate**. Para instalarlo y habilitar su uso:

```
# yum install mlocate
# updatedb
```

2. Busca el nombre y la versión del paquete que proporcionó el archivo:

```
$ rpm -qf /usr/lib64/libz.so.1.2.7
zlib-1.2.11-10.el8.x86_64
```

La salida proporciona detalles del paquete instalado en el formato *name*:*epoch*-*version*.*release*.*architecture*.



IMPORTANTE

Si este paso no produce ningún resultado, no es posible determinar qué paquete proporcionó el archivo binario. Hay varios casos posibles:

- El archivo se instala desde un paquete que no es conocido por las herramientas de gestión de paquetes en su configuración *current*.
- El archivo se instala a partir de un paquete descargado localmente e instalado manualmente. En ese caso es imposible determinar automáticamente un paquete **debuginfo** adecuado.
- Sus herramientas de gestión de paquetes están mal configuradas.
- El archivo no se instala desde ningún paquete. En tal caso, no existe ningún paquete respectivo de **debuginfo**.

Dado que los pasos posteriores dependen de éste, debe resolver esta situación o abortar este procedimiento. Describir los pasos exactos de la solución de problemas está fuera del alcance de este procedimiento.

3. Instale los paquetes de **debuginfo** utilizando la utilidad **debuginfo-install**. En el comando, utilice el nombre del paquete y otros detalles que determinó durante el paso anterior:

```
# debuginfo-install zlib-1.2.11-10.el8.x86_64
```

Recursos adicionales

- Manual del usuario de Red Hat Developer Toolset, sección Instalación de la información de depuración
- ¿Cómo puedo descargar o instalar paquetes debuginfo para sistemas RHEL?

CAPÍTULO 8. INSPECCIÓN DEL ESTADO INTERNO DE LA APLICACIÓN CON GDB

Para encontrar por qué una aplicación no funciona correctamente, hay que controlar su ejecución y examinar su estado interno con un depurador. Esta sección describe cómo utilizar el depurador de GNU (GDB) para esta tarea.

8.1. DEPURADOR GNU (GDB)

Red Hat Enterprise Linux contiene el depurador GNU (GDB) que le permite investigar lo que ocurre dentro de un programa a través de una interfaz de usuario de línea de comandos.

Para obtener una interfaz gráfica de GDB, instale el entorno de desarrollo integrado Eclipse. Consulte Uso de Eclipse.

Capacidades de GDB

Una sola sesión de GDB puede depurar los siguientes tipos de programas:

- Programas multihilo y bifurcados
- Varios programas a la vez
- Programas en máquinas remotas o en contenedores con la utilidad **gdbserver** conectados a través de una conexión de red TCP/IP

Requisitos de depuración

Para depurar cualquier código ejecutable, GDB requiere información de depuración para ese código en particular:

- Para los programas desarrollados por usted, puede crear la información de depuración mientras construye el código.
- En el caso de los programas del sistema instalados a partir de paquetes, debe instalar sus paquetes debuginfo.

8.2. ADJUNTAR GDB A UN PROCESO

Para examinar un proceso, GDB debe ser *attached* al proceso.

Requisitos previos

- GDB debe estar instalado en el sistema

Iniciar un programa con GDB

Cuando el programa no se está ejecutando como un proceso, iniciarlo con GDB:

```
$ gdb program
```

Sustituya *program* por un nombre de archivo o una ruta de acceso al programa.

GDB se configura para iniciar la ejecución del programa. Puede configurar los puntos de interrupción y el entorno **gdb** antes de comenzar la ejecución del proceso con el comando **run**.

Adjuntar GDB a un proceso ya en marcha

Para adjuntar GDB a un programa que ya se está ejecutando como un proceso:

1. Encuentre el ID del proceso (*pid*) con el comando **ps**:

```
$ ps -C program -o pid h
 pid
```

   Sustituya *program* por un nombre de archivo o una ruta de acceso al programa.

2. Adjunte el BGF a este proceso:

```
$ gdb -p pid
```

   Sustituya *pid* por un número de identificación de proceso real de la salida de **ps**.

Adjuntar un GDB ya en marcha a un proceso ya en marcha

Para adjuntar un GDB ya en marcha a un programa ya en marcha:

1. Utilice el comando GDB **shell** para ejecutar el comando **ps** y encontrar el ID del proceso del programa (*pid*):

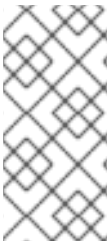
```
(gdb) shell ps -C program -o pid h
 pid
```

   Sustituya *program* por un nombre de archivo o una ruta de acceso al programa.

2. Utilice el comando **attach** para adjuntar GDB al programa:

```
(gdb) adjuntar pid
```

   Sustituya *pid* por un número de identificación de proceso real de la salida de **ps**.



NOTA

En algunos casos, es posible que GDB no pueda encontrar el archivo ejecutable correspondiente. Utilice el comando **file** para especificar la ruta:

```
(gdb) archivo path/to/program
```

Recursos adicionales

- Depuración con GDB
- Depuración con GDB

8.3. RECORRER EL CÓDIGO DEL PROGRAMA CON GDB

Una vez que el depurador **GDB** está conectado a un programa, puede utilizar una serie de comandos para controlar la ejecución del programa.

Requisitos previos

- Debe disponer de la información de depuración necesaria:
  - El programa es compilado y construido con información de depuración, o
  - Los paquetes debuginfo pertinentes están instalados
- GDB debe estar conectado al programa que se va a depurar

Comandos GDB para recorrer el código

r (correr)

Inicia la ejecución del programa. Si **run** se ejecuta con algún argumento, esos argumentos se pasan al ejecutable como si el programa se hubiera iniciado normalmente. Los usuarios normalmente emiten este comando después de establecer puntos de interrupción.

start

Inicia la ejecución del programa pero se detiene al principio de la función principal del programa. Si **start** se ejecuta con algún argumento, esos argumentos se pasan al ejecutable como si el programa se hubiera iniciado normalmente.

c (continuar)

Continuar la ejecución del programa desde el estado actual. La ejecución del programa continuará hasta que se cumpla una de las siguientes condiciones:

- Se alcanza un punto de interrupción.
- Se cumple una condición determinada.
- El programa recibe una señal.
- Se ha producido un error.
- El programa termina.

n (siguiente)

Continúa la ejecución del programa desde el estado actual, hasta que se alcanza la siguiente línea de código en el archivo fuente actual. La ejecución del programa continuará hasta que una de las siguientes situaciones se haga realidad:

- Se alcanza un punto de interrupción.
- Se cumple una condición determinada.
- El programa recibe una señal.
- Se ha producido un error.
- El programa termina.

s (paso)

El comando **step** también detiene la ejecución en cada línea secuencial de código en el archivo fuente actual. Sin embargo, si la ejecución se detiene actualmente en una línea de código fuente que contiene un **function call**, GDB detiene la ejecución después de introducir la llamada a la función (en lugar de ejecutarla).

until *location*

Continúa la ejecución hasta que se alcanza la ubicación del código especificado por la opción *location*.

fini (acabado)

Reanudar la ejecución del programa y detenerlo cuando la ejecución regrese de una función. La ejecución del programa continuará hasta que una de las siguientes situaciones se haga realidad:

- Se alcanza un punto de interrupción.
- Se cumple una condición determinada.
- El programa recibe una señal.
- Se ha producido un error.
- El programa termina.

q (abandonar)

Termina la ejecución y sale de GDB.

Recursos adicionales

- Sección 8.5, "Uso de los puntos de interrupción de GDB para detener la ejecución en lugares definidos del código"
- Depuración con GDB
- Depuración con GDB

8.4. MOSTRAR LOS VALORES INTERNOS DEL PROGRAMA CON GDB

Mostrar los valores de las variables internas de un programa es importante para entender lo que el programa está haciendo. GDB ofrece múltiples comandos que puede utilizar para inspeccionar las variables internas. Esta sección describe los más útiles de estos comandos:

p (imprimir)

Muestra el valor del argumento dado. Normalmente, el argumento es el nombre de una variable de cualquier complejidad, desde un simple valor único hasta una estructura. Un argumento también puede ser una expresión válida en el lenguaje actual, incluyendo el uso de variables de programa y funciones de biblioteca, o funciones definidas en el programa que se está probando.
Es posible ampliar GDB con *pretty-printer* scripts de Python o Guile para la visualización personalizada de estructuras de datos (como clases, structs) utilizando el comando **print**.

bt (backtrace)

Muestra la cadena de llamadas a funciones utilizadas para alcanzar el punto de ejecución actual, o la cadena de funciones utilizadas hasta que la ejecución fue terminada. Esto es útil para investigar errores graves (como los fallos de segmentación) con causas esquivas.
Añadiendo la opción **full** al comando **backtrace** también se muestran las variables locales.

Es posible ampliar GDB con *frame filter* scripts de Python para la visualización personalizada de los datos mostrados mediante los comandos **bt** y **info frame**. El término *frame* se refiere a los datos asociados a una única llamada a una función.

info

El comando **info** es un comando genérico para proporcionar información sobre varios elementos. Toma una opción que especifica el elemento a describir.

- El comando **info args** muestra las opciones de la llamada a la función que es el marco actualmente seleccionado.
- El comando **info locals** muestra las variables locales en el marco actualmente seleccionado.

Para obtener una lista de los posibles elementos, ejecute el comando **help info** en una sesión GDB:

```
(gdb) help info
```

l (lista)

Muestra la línea del código fuente donde se detuvo el programa. Este comando está disponible sólo cuando la ejecución del programa está detenida. Aunque no es estrictamente un comando para mostrar el estado interno, **list** ayuda al usuario a entender qué cambios en el estado interno ocurrirán en el siguiente paso de la ejecución del programa.

Recursos adicionales

- La API de GDB Python
- Depuración con GDB

8.5. USO DE LOS PUNTOS DE INTERRUPCIÓN DE GDB PARA DETENER LA EJECUCIÓN EN LUGARES DEFINIDOS DEL CÓDIGO

A menudo, sólo se investigan pequeñas porciones de código. Los puntos de interrupción son marcadores que indican a GDB que detenga la ejecución de un programa en un lugar determinado del código. Los puntos de interrupción se asocian más comúnmente con las líneas de código fuente. En ese caso, colocar un punto de interrupción requiere especificar el archivo fuente y el número de línea.

- A **place a breakpoint**:
  - Especifique el nombre del código fuente *file* y el *line* en ese archivo:

    ```
    (gdb) br file:line
    ```

  - Cuando *file* no está presente, se utiliza el nombre del archivo fuente en el punto actual de ejecución:

    ```
    (gdb) br line
    ```

  - Alternativamente, utilice el nombre de una función para poner el punto de interrupción en su inicio:

    ```
    (gdb) br function_name
    ```

- Un programa puede encontrar un error después de un cierto número de iteraciones de una tarea. Para especificar un **condition** adicional para detener la ejecución:

  ```
  (gdb) br file:line si condition
  ```

Sustituya *condition* por una condición en el lenguaje C o C. El significado de *file* y *line* es el mismo que el anterior.

- Para **inspect** el estado de todos los puntos de interrupción y de vigilancia:

```
(gdb) info br
```

- Para **remove** un punto de interrupción utilizando su *number* como se muestra en la salida de **info br**:

```
(gdb) borrar number
```

- Para **remove** un punto de interrupción en un lugar determinado:

```
(gdb) clear file:line
```

Recursos adicionales

- Depuración con GDB

8.6. USO DE LOS PUNTOS DE VIGILANCIA DE GDB PARA DETENER LA EJECUCIÓN EN CASO DE ACCESO A DATOS Y CAMBIOS

En muchos casos, es ventajoso dejar que el programa se ejecute hasta que ciertos datos cambien o se acceda a ellos. Esta sección enumera los casos de uso más comunes.

Requisitos previos

- Comprensión de **GDB**

Uso de puntos de control en GDB

Los puntos de vigilancia son marcadores que indican a **GDB** que detenga la ejecución de un programa. Los puntos de vigilancia están asociados a datos: para colocar un punto de vigilancia es necesario especificar una expresión que describa una variable, varias variables o una dirección de memoria.

- Para **place** un punto de vigilancia para datos **change** (escritura):

```
(gdb) ver expression
```

Sustituya *expression* por una expresión que describa lo que quiere ver. Para las variables, *expression* es igual al nombre de la variable.

- Para **place** un punto de vigilancia para los datos **access** (leer):

```
(gdb) rwatch expression
```

- Para **place** un punto de vigilancia para el acceso a los datos de **any** (tanto de lectura como de escritura):

```
(gdb) awatch expression
```

- Para **inspect** el estado de todos los puntos de control y de ruptura:

```
(gdb) info br
```

- Para **remove** un punto de vigilancia:

```
(gdb) borrar num
```

Sustituya la opción ***num*** por el número indicado por el comando **info br**.

Recursos adicionales

- Depuración con GDB

8.7. DEPURACIÓN DE PROGRAMAS BIFURCADOS O ROSCADOS CON GDB

Algunos programas utilizan bifurcaciones o hilos para conseguir una ejecución paralela del código. La depuración de múltiples rutas de ejecución simultáneas requiere consideraciones especiales.

Requisitos previos

- Debes entender los conceptos de bifurcación de procesos e hilos.

Depuración de programas bifurcados con GDB

La bifurcación es una situación en la que un programa (**parent**) crea una copia independiente de sí mismo (**child**). Utilice los siguientes ajustes y comandos para afectar a lo que hace GDB cuando se produce una bifurcación:

- El ajuste **follow-fork-mode** controla si GDB sigue al padre o al hijo después de la bifurcación.

  set follow-fork-mode parent

  Después de una bifurcación, depurar el proceso padre. Esta es la opción por defecto.

  set follow-fork-mode child

  Después de una bifurcación, depura el proceso hijo.

  show follow-fork-mode

  Muestra la configuración actual de **follow-fork-mode**.

- El ajuste **set detach-on-fork** controla si el GDB mantiene el control del otro proceso (no seguido) o lo deja correr.

  set detach-on-fork on

  El proceso que no se sigue (según el valor de **follow-fork-mode**) se separa y se ejecuta de forma independiente. Este es el valor por defecto.

  set detach-on-fork off

  GDB mantiene el control de ambos procesos. El proceso que se sigue (dependiendo del valor de **follow-fork-mode**) se depura como siempre, mientras que el otro se suspende.

  show detach-on-fork

  Muestra la configuración actual de **detach-on-fork**.

Depuración de programas roscados con GDB

GDB tiene la capacidad de depurar hilos individuales, y de manipularlos y examinarlos

independientemente. Para hacer que GDB detenga sólo el hilo que se examina, utilice los comandos **set non-stop on** y **set target-async on**. Puede añadir estos comandos al archivo **.gdbinit**. Después de activar esa funcionalidad, GDB está listo para realizar la depuración de hilos.

GDB utiliza el concepto de *current thread*. Por defecto, los comandos se aplican sólo al hilo actual.

info threads

Muestra una lista de hilos con sus números **id** y **gid**, indicando el hilo actual.

thread *id*

Establece el hilo con el **id** especificado como el hilo actual.

thread apply *ids command*

Aplicar el comando ***command*** a todos los hilos listados por ***ids***. La opción ***ids*** es una lista de identificadores de hilos separada por espacios. Un valor especial **all** aplica el comando a todos los hilos.

break *location* thread *id* if *condition*

Establecer un punto de interrupción en un determinado ***location*** con un determinado ***condition*** sólo para el número de hilo ***id***.

watch *expression* thread *id*

Establece un punto de control definido por ***expression*** sólo para el número de hilo ***id***.

command&

Ejecuta el comando **command** y vuelve inmediatamente al prompt de gdb **(gdb)**, continuando cualquier ejecución de código en segundo plano.

interrupt

Detener la ejecución en el fondo.

Recursos adicionales

- Depuración con GDB
- Depuración con GDB

CAPÍTULO 9. GRABACIÓN DE LAS INTERACCIONES DE LA APLICACIÓN

El código ejecutable de las aplicaciones interactúa con el código del sistema operativo y las bibliotecas compartidas. La grabación de un registro de actividad de estas interacciones puede proporcionar suficiente información sobre el comportamiento de la aplicación sin necesidad de depurar el código real de la misma. Por otra parte, el análisis de las interacciones de una aplicación puede ayudar a determinar las condiciones en las que se manifiesta un fallo.

9.1. HERRAMIENTAS ÚTILES PARA REGISTRAR LAS INTERACCIONES DE LAS APLICACIONES

Red Hat Enterprise Linux ofrece múltiples herramientas para analizar las interacciones de una aplicación.

strace

La herramienta **strace** permite principalmente registrar las llamadas al sistema (funciones del núcleo) utilizadas por una aplicación.

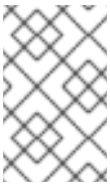
- La herramienta **strace** puede proporcionar una salida detallada sobre las llamadas, porque **strace** interpreta los parámetros y los resultados con el conocimiento del código del núcleo subyacente. Los números se convierten en los respectivos nombres de las constantes, las banderas combinadas a nivel de bits se expanden a la lista de banderas, los punteros a las matrices de caracteres se desreferencian para proporcionar la cadena real, y más. Puede faltar el soporte para las características más recientes del kernel.
- Puede filtrar las llamadas rastreadas para reducir la cantidad de datos capturados.
- El uso de **strace** no requiere ninguna configuración particular, excepto la configuración del filtro de registro.
- El rastreo del código de la aplicación con **strace** provoca una importante ralentización de la ejecución de la aplicación. Como resultado, **strace** no es adecuado para muchas implantaciones de producción. Como alternativa, considere el uso de **ltrace** o SystemTap.
- La versión de **strace** disponible en Red Hat Developer Toolset también puede realizar la manipulación de llamadas al sistema. Esta capacidad es útil para la depuración.

ltrace

La herramienta **ltrace** permite registrar las llamadas al espacio de usuario de una aplicación en objetos compartidos (bibliotecas dinámicas).

- La herramienta **ltrace** permite rastrear las llamadas a cualquier biblioteca.
- Puede filtrar las llamadas rastreadas para reducir la cantidad de datos capturados.
- El uso de **ltrace** no requiere ninguna configuración particular, excepto la configuración del filtro de registro.
- La herramienta **ltrace** es ligera y rápida, y ofrece una alternativa a **strace**: es posible rastrear las respectivas interfaces en bibliotecas como **glibc** con **ltrace** en lugar de rastrear las funciones del núcleo con **strace**.
- Dado que **ltrace** no maneja un conjunto conocido de llamadas como **strace**, no intenta explicar los valores pasados a las funciones de la biblioteca. La salida de **ltrace** sólo contiene

números y punteros en bruto. La interpretación de la salida de **ltrace** requiere consultar las declaraciones de interfaz reales de las bibliotecas presentes en la salida.



NOTA

En Red Hat Enterprise Linux 8.0, un problema conocido impide que **ltrace** rastree archivos ejecutables del sistema. Esta limitación no se aplica a los archivos ejecutables construidos por los usuarios.

SystemTap

SystemTap es una plataforma de instrumentación para sondear los procesos en ejecución y la actividad del kernel en el sistema Linux. SystemTap utiliza su propio lenguaje de scripting para programar manejadores de eventos personalizados.

- En comparación con el uso de **strace** y **ltrace**, la creación de scripts para el registro supone más trabajo en la fase de configuración inicial. Sin embargo, las capacidades de scripting amplían la utilidad de SystemTap más allá de la producción de registros.
- SystemTap funciona creando e insertando un módulo del kernel. El uso de SystemTap es eficiente y no crea una ralentización significativa del sistema o de la ejecución de aplicaciones por sí mismo.
- SystemTap viene con un conjunto de ejemplos de uso.

GDB

El depurador de GNU (GDB) está pensado principalmente para la depuración, no para el registro. Sin embargo, algunas de sus características lo hacen útil incluso en el escenario donde la interacción de una aplicación es la actividad principal de interés.

- Con GDB, es posible combinar convenientemente la captura de un evento de interacción con la depuración inmediata de la ruta de ejecución posterior.
- El BGF es más adecuado para analizar la respuesta a eventos infrecuentes o singulares, tras la identificación inicial de la situación problemática por parte de otras herramientas. El uso del BGF en cualquier escenario con eventos frecuentes resulta ineficaz o incluso imposible.

Recursos adicionales

- Guía para principiantes de Red Hat Enterprise Linux SystemTap
- Manual del usuario del conjunto de herramientas para desarrolladores de Red Hat

9.2. MONITORIZACIÓN DE LAS LLAMADAS AL SISTEMA DE UNA APLICACIÓN CON STRACE

La herramienta **strace** permite monitorizar las llamadas al sistema (kernel) realizadas por una aplicación.

Requisitos previos

- Debe tener instalado **strace** en el sistema.

Procedimiento

1. Identifique las llamadas del sistema a supervisar.

2. Inicie **strace** y adjúntelo al programa.

   - Si el programa que desea supervisar no se está ejecutando, inicie **strace** y especifique el *program*:

     ```
     $ strace -fvttTyy -s 256 -e trace=call program
     ```

   - Si el programa ya se está ejecutando, busque su id de proceso (*pid*) y adjunte **strace** a él:

     ```
     $ ps -C program
     (...)
     $ strace -fvttTyy -s 256 -e trace=call -ppid
     ```

   - Sustituya *call* por las llamadas al sistema que deben mostrarse. Puede utilizar la opción **-e trace=*call*** varias veces. Si se omite, **strace** mostrará todos los tipos de llamadas del sistema. Consulte la página del manual *strace(1)* para obtener más información.

   - Si no desea rastrear ningún proceso o hilo bifurcado, omita la opción **-f**.

3. La herramienta **strace** muestra las llamadas al sistema realizadas por la aplicación y sus detalles. En la mayoría de los casos, una aplicación y sus bibliotecas realizan un gran número de llamadas y la salida **strace** aparece inmediatamente, si no se establece un filtro para las llamadas al sistema.

4. La herramienta **strace** sale cuando el programa se cierra.
   Para terminar la monitorización antes de que el programa trazado salga, pulse **Ctrl C**.

   - Si **strace** inició el programa, el programa termina junto con **strace**.

   - Si se adjunta **strace** a un programa que ya se está ejecutando, el programa termina junto con **strace**.

5. Analizar la lista de llamadas al sistema realizadas por la aplicación.

   - Los problemas de acceso o disponibilidad de recursos aparecen en el registro como llamadas que devuelven errores.

   - Los valores pasados a las llamadas del sistema y los patrones de las secuencias de llamadas permiten conocer las causas del comportamiento de la aplicación.

   - Si la aplicación se bloquea, la información importante probablemente esté al final del registro.

   - La salida contiene mucha información innecesaria. Sin embargo, puede construir un filtro más preciso para las llamadas al sistema que le interesen y repetir el procedimiento.

NOTA

Es ventajoso tanto ver la salida como guardarla en un archivo. Para ello, utilice el comando **tee**:

```
$ strace ... |& tee your_log_file.log
```

Recursos adicionales

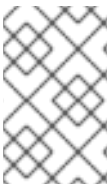
- La página del manual *strace(1)*:

```
$ man strace
```

- ¿Cómo puedo utilizar strace para rastrear las llamadas al sistema realizadas por un comando?
- Manual del usuario del conjunto de herramientas para desarrolladores de Red Hat

9.3. MONITORIZACIÓN DE LAS LLAMADAS A FUNCIONES DE LA BIBLIOTECA DE LA APLICACIÓN CON LTRACE

La herramienta **ltrace** permite supervisar las llamadas de una aplicación a las funciones disponibles en las bibliotecas (objetos compartidos).



NOTA

En Red Hat Enterprise Linux 8.0, un problema conocido impide que **ltrace** rastree archivos ejecutables del sistema. Esta limitación no se aplica a los archivos ejecutables construidos por los usuarios.

Requisitos previos

- Debe tener instalado **ltrace** en el sistema.

Procedimiento

1. Identifique las bibliotecas y funciones de interés, si es posible.
2. Inicie **ltrace** y adjúntelo al programa.
   - Si el programa que desea supervisar no se está ejecutando, inicie **ltrace** y especifique *program*:

```
$ ltrace -f -l library -e function program
```

   - Si el programa ya se está ejecutando, busque su id de proceso (*pid*) y adjunte **ltrace** a él:

```
$ ps -C program
(...)
$ ltrace -f -l library -e function program -ppid
```

   - Utilice las opciones **-e**, **-f** y **-l** para filtrar la salida:
     - Suministre los nombres de las funciones que se mostrarán como *function*. La opción **-e *function*** puede utilizarse varias veces. Si se omite, **ltrace** muestra las llamadas a todas las funciones.
     - En lugar de especificar funciones, puede especificar bibliotecas completas con la opción **-l *library*** opción. Esta opción se comporta de forma similar a la opción **-e *function*** opción.
     - Si no desea rastrear ningún proceso o hilo bifurcado, omita la opción **-f**.

Consulte la página del manual *ltrace*(1)_ para más información.

3. **ltrace** muestra las llamadas a la biblioteca realizadas por la aplicación.
   En la mayoría de los casos, una aplicación realiza un gran número de llamadas y la salida **ltrace** se muestra inmediatamente, si no se establece ningún filtro.

4. **ltrace** sale cuando el programa se cierra.
   Para terminar la monitorización antes de que el programa rastreado salga, pulse **ctrl C**.

   - Si **ltrace** inició el programa, el programa termina junto con **ltrace**.
   - Si se adjunta **ltrace** a un programa que ya se está ejecutando, el programa termina junto con **ltrace**.

5. Analizar la lista de llamadas a la biblioteca realizadas por la aplicación.

   - Si la aplicación se bloquea, la información importante probablemente esté al final del registro.
   - El resultado contiene mucha información innecesaria. Sin embargo, puede construir un filtro más preciso y repetir el procedimiento.

NOTA

Es ventajoso tanto ver la salida como guardarla en un archivo. Para ello, utilice el comando **tee**:

```
$ ltrace ... |& tee your_log_file.log
```

Recursos adicionales

- La página del manual *ltrace(1)*:

```
$ man ltrace
```

- Manual del usuario del conjunto de herramientas para desarrolladores de Red Hat

9.4. SUPERVISIÓN DE LAS LLAMADAS AL SISTEMA DE LA APLICACIÓN CON SYSTEMTAP

La herramienta SystemTap permite registrar manejadores de eventos personalizados para los eventos del núcleo. En comparación con la herramienta **strace**, es más difícil de usar pero más eficiente y permite una lógica de procesamiento más complicada. Un script de SystemTap llamado **strace.stp** se instala junto con SystemTap y proporciona una aproximación a la funcionalidad de **strace** utilizando SystemTap.

Requisitos previos

- SystemTap y los respectivos paquetes del kernel deben estar instalados en el sistema.

Procedimiento

1. Busque el ID del proceso (*pid*) del proceso que desea supervisar:

```
$ ps -aux
```

2. Ejecute SystemTap con el script **strace.stp**:

```
# stap /usr/share/systemtap/examples/process/strace.stp -x pid
```

El valor de *pid* es el identificador del proceso.

El script se compila en un módulo del kernel, que luego se carga. Esto introduce un ligero retraso entre la introducción del comando y la obtención de la salida.

3. Cuando el proceso realiza una llamada al sistema, el nombre de la llamada y sus parámetros se imprimen en el terminal.

4. El script sale cuando el proceso termina, o cuando se presiona **Ctrl C**.

9.5. USO DE GDB PARA INTERCEPTAR LAS LLAMADAS DEL SISTEMA DE LA APLICACIÓN

El depurador de GNU (GDB) le permite detener una ejecución en varias situaciones que surgen durante la ejecución del programa. Para detener la ejecución cuando el programa realiza una llamada al sistema, utilice un GDB *catchpoint*.

Requisitos previos

- Debes entender el uso de los puntos de interrupción de GDB.
- El GDB debe estar unido al programa.

Procedimiento

1. Establece el punto de captura:

```
(gdb) catch syscall syscall-name
```

El comando **catch syscall** establece un tipo especial de punto de interrupción que detiene la ejecución cuando el programa realiza una llamada al sistema.

La opción ***syscall-name*** especifica el nombre de la llamada. Puede especificar varios puntos de captura para varias llamadas al sistema. Si se omite la opción ***syscall-name*** hace que GDB se detenga en cualquier llamada al sistema.

2. Iniciar la ejecución del programa.

   - Si el programa no ha iniciado su ejecución, inícielo:

```
(gdb) r
```

   - Si la ejecución del programa se detiene, reanúdela:

```
(gdb) c
```

3. GDB detiene la ejecución después de que el programa realice cualquier llamada al sistema especificada.

Recursos adicionales

- Sección 8.4, "Mostrar los valores internos del programa con GDB"
- Sección 8.3, "Recorrer el código del programa con GDB"
- Depuración con GDB

9.6. USO DE GDB PARA INTERCEPTAR EL MANEJO DE SEÑALES POR PARTE DE LAS APLICACIONES

El depurador de GNU (GDB) le permite detener la ejecución en varias situaciones que surgen durante la ejecución del programa. Para detener la ejecución cuando el programa recibe una señal del sistema operativo, utilice un GDB *catchpoint*.

Requisitos previos

- Debes entender el uso de los puntos de interrupción de GDB.
- El GDB debe estar unido al programa.

Procedimiento

1. Establece el punto de captura:

   ```
   (gdb) atrapar señal signal-type
   ```

   El comando **catch signal** establece un tipo especial de punto de interrupción que detiene la ejecución cuando el programa recibe una señal. La opción ***signal-type*** especifica el tipo de señal. Utilice el valor especial **'all'** para capturar todas las señales.

2. Deja que el programa se ejecute.
   - Si el programa no ha iniciado su ejecución, inícielo:

     ```
     (gdb) r
     ```

   - Si la ejecución del programa se detiene, reanúdela:

     ```
     (gdb) c
     ```

3. GDB detiene la ejecución después de que el programa reciba cualquier señal especificada.

Recursos adicionales

- Sección 8.4, "Mostrar los valores internos del programa con GDB"
- Sección 8.3, "Recorrer el código del programa con GDB"
- Depuración con GDB

CAPÍTULO 10. DEPURACIÓN DE UNA APLICACIÓN BLOQUEADA

A veces, no es posible depurar una aplicación directamente. En estas situaciones, puedes recoger información sobre la aplicación en el momento de su finalización y analizarla después.

10.1. VERTEDEROS DE NÚCLEO: QUÉ SON Y CÓMO UTILIZARLOS

Un core dump es una copia de una parte de la memoria de la aplicación en el momento en que ésta dejó de funcionar, almacenada en formato ELF. Contiene todas las variables internas de la aplicación y la pila, lo que permite inspeccionar el estado final de la aplicación. Cuando se aumenta con el respectivo archivo ejecutable y la información de depuración, es posible analizar un archivo de volcado del núcleo con un depurador de manera similar a analizar un programa en ejecución.

El kernel del sistema operativo Linux puede registrar los volcados de núcleo automáticamente, si esta funcionalidad está habilitada. Alternativamente, puedes enviar una señal a cualquier aplicación en ejecución para que genere un volcado de núcleo independientemente de su estado real.

AVISO

Algunos límites pueden afectar a la capacidad de generar un volcado de núcleo. Para ver los límites actuales:

```
$ ulimit -a
```

10.2. GRABACIÓN DE LOS FALLOS DE LA APLICACIÓN CON LOS VOLCADOS DEL NÚCLEO

Para registrar las caídas de la aplicación, configure el guardado del volcado del núcleo y añada información sobre el sistema.

Procedimiento

1. Para habilitar los volcados de núcleo, asegúrese de que el archivo **/etc/systemd/system.conf** contiene las siguientes líneas:

   ```
   DumpCore=yes
   DefaultLimitCORE=infinity
   ```

   También puede añadir comentarios que describan si estos ajustes estaban presentes anteriormente, y cuáles eran los valores anteriores. Esto le permitirá revertir estos cambios más tarde, si es necesario. Los comentarios son líneas que comienzan con el carácter **#**.

   La modificación del archivo requiere un acceso de nivel de administrador.

2. Aplique la nueva configuración:

   ```
   # systemctl daemon-reexec
   ```

3. Eliminar los límites de los tamaños de volcado del núcleo:

```
# ulimit -c unlimited
```

Para invertir este cambio, ejecute el comando con el valor **0** en lugar de ***unlimited***.

4. Instale el paquete **sos** que proporciona la utilidad **sosreport** para recopilar información del sistema:

```
# yum install sos
```

5. Cuando una aplicación se bloquea, se genera un volcado del núcleo que es gestionado por **systemd-coredump**.

6. Crear un informe SOS para proporcionar información adicional sobre el sistema:

```
# sosreport
```

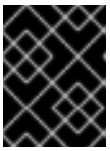
Esto crea un archivo **.tar** que contiene información sobre su sistema, como copias de los archivos de configuración.

7. Localice y exporte el volcado del núcleo:

```
$ coredumpctl list executable-name
$ coredumpctl dump executable-name > /path/to/file-for-export
```

Si la aplicación se estrelló varias veces, la salida del primer comando enumera más volcados de núcleo capturados. En ese caso, construya para el segundo comando una consulta más precisa utilizando la otra información. Consulte la página del manual *coredumpctl(1)* para más detalles.

8. Transfiera el volcado del núcleo y el informe SOS al ordenador donde se realizará la depuración. Transfiera también el archivo ejecutable, si lo conoce.



IMPORTANTE

Cuando no se conoce el archivo ejecutable, el análisis posterior del archivo central lo identifica.

9. Opcional: Elimine el volcado del núcleo y el informe SOS después de transferirlos, para liberar espacio en el disco.

Recursos adicionales

- Introducción a systemd en el documento *Configuring basic system settings*
- Cómo habilitar el volcado de archivos del núcleo cuando una aplicación se bloquea o se producen fallos de segmentación
- ¿Qué es un sosreport y cómo crear uno en Red Hat Enterprise Linux 4.6 y posteriores?

10.3. INSPECCIÓN DE LOS ESTADOS DE CAÍDA DE LA APLICACIÓN CON LOS VOLCADOS DEL NÚCLEO

Requisitos previos

- Debe tener un archivo de volcado del núcleo y un informe de sosreport del sistema en el que se produjo el fallo.
- GDB y elfutils deben estar instalados en su sistema.

Procedimiento

1. Para identificar el archivo ejecutable en el que se produjo el fallo, ejecute el comando **eu-unstrip** con el archivo de volcado del núcleo:

```
$ eu-unstrip -n --core=./core.9814
0x400000+0x207000 2818b2009547f780a5639c904cded443e564973e@0x400284
/usr/bin/sleep /usr/lib/debug/bin/sleep.debug [exe]
0x7fff26fff000+0x1000 1e2a683b7d877576970e4275d41a6aaec280795e@0x7fff26fff340 . -
linux-vdso.so.1
0x35e7e00000+0x3b6000
374add1ead31ccb449779bc7ee7877de3377e5ad@0x35e7e00280 /usr/lib64/libc-2.14.90.so
/usr/lib/debug/lib64/libc-2.14.90.so.debug libc.so.6
0x35e7a00000+0x224000
3ed9e61c2b7e707ce244816335776afa2ad0307d@0x35e7a001d8 /usr/lib64/ld-2.14.90.so
/usr/lib/debug/lib64/ld-2.14.90.so.debug ld-linux-x86-64.so.2
```

   La salida contiene detalles para cada módulo en una línea, separados por espacios. La información aparece en este orden:

   1. La dirección de memoria a la que se asignó el módulo
   2. El build-id del módulo y en qué lugar de la memoria se encuentra
   3. El nombre del archivo ejecutable del módulo, mostrado como **-** cuando es desconocido, o como **.** cuando el módulo no ha sido cargado desde un archivo
   4. La fuente de información de depuración, mostrada como un nombre de archivo cuando está disponible, como **.** cuando está contenida en el propio archivo ejecutable, o como **-** cuando no está presente en absoluto
   5. El nombre de la biblioteca compartida (*soname*) o **[exe]** para el módulo principal

   En este ejemplo, los detalles importantes son el nombre del archivo **/usr/bin/sleep** y el build-id **2818b2009547f780a5639c904cded443e564973e** en la línea que contiene el texto **[exe]**. Con esta información, puedes identificar el archivo ejecutable necesario para analizar el volcado del núcleo.

2. Obtenga el archivo ejecutable que se estrelló.
   - Si es posible, cópielo del sistema en el que se produjo el fallo. Utilice el nombre del archivo extraído del archivo del núcleo.
   - También puede utilizar un archivo ejecutable idéntico en su sistema. Cada archivo ejecutable construido en Red Hat Enterprise Linux contiene una nota con un valor único de build-id. Determine el build-id de los archivos ejecutables relevantes disponibles localmente:

```
$ eu-readelf -n executable_file
```

Utilice esta información para hacer coincidir el archivo ejecutable del sistema remoto con su copia local. El build-id del archivo local y el build-id que aparece en el volcado del núcleo deben coincidir.

- Por último, si la aplicación se instala desde un paquete RPM, puede obtener el archivo ejecutable del paquete. Utilice la salida de **sosreport** para encontrar la versión exacta del paquete requerido.

3. Obtenga las bibliotecas compartidas utilizadas por el archivo ejecutable. Utilice los mismos pasos que para el archivo ejecutable.

4. Si la aplicación se distribuye como un paquete, cargue el archivo ejecutable en GDB, para mostrar pistas sobre los paquetes de depuración que faltan. Para más detalles, consulte Sección 7.4, “Obtención de paquetes debuginfo para una aplicación o librería usando GDB” .

5. Para examinar el archivo del núcleo en detalle, cargue el archivo ejecutable y el archivo de volcado del núcleo con GDB:

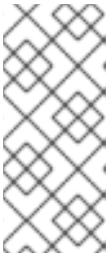
```
$ gdb -e executable_file -c core_file
```

Otros mensajes sobre los archivos que faltan y la información de depuración le ayudan a identificar lo que falta para la sesión de depuración. Vuelva al paso anterior si es necesario.

Si la información de depuración de la aplicación está disponible como un archivo en lugar de como un paquete, cargue este archivo en GDB con el comando **symbol-file**:

```
(gdb) archivo-símbolo program.debug
```

Sustituya *program.debug* por el nombre real del archivo.



NOTA

Puede que no sea necesario instalar la información de depuración para todos los archivos ejecutables contenidos en el volcado del núcleo. La mayoría de estos archivos ejecutables son bibliotecas utilizadas por el código de la aplicación. Es posible que estas bibliotecas no contribuyan directamente al problema que se está analizando, y no es necesario incluir información de depuración para ellas.

6. Utilice los comandos GDB para inspeccionar el estado de la aplicación en el momento en que se estrelló. Ver Capítulo 8, *Inspección del estado interno de la aplicación con GDB* .

NOTA

Cuando se analiza un archivo de núcleo, GDB no se adjunta a un proceso en ejecución. Los comandos para controlar la ejecución no tienen efecto.

Recursos adicionales

- Depuración con GDB
- Depuración con GDB
- Depuración con GDB

10.4. CREACIÓN Y ACCESO A UN VOLCADO DEL NÚCLEO CON COREDUMPCTL

La herramienta **coredumpctl** de **systemd** puede agilizar significativamente el trabajo con los volcados de núcleo en la máquina donde se produjo el fallo. Este procedimiento describe cómo capturar un volcado de núcleo de un proceso que no responde.

Requisitos previos

- El sistema debe estar configurado para utilizar **systemd-coredump** para el manejo de los volcados del núcleo. Para verificar que esto es cierto:

```
$ sysctl kernel.core_pattern
```

  La configuración es correcta si la salida comienza con lo siguiente:

```
kernel.core_pattern = |/usr/lib/systemd/systemd-coredump
```

Procedimiento

1. Encuentra el PID del proceso colgado, basado en una parte conocida del nombre del archivo ejecutable:

```
$ pgrep -a executable-name-fragment
```

   Este comando mostrará una línea de la forma

```
PID command-line
```

   Utilice el valor de *command-line* para verificar que el *PID* pertenece al proceso previsto.

   Por ejemplo:

```
$ pgrep -a bc
5459 bc
```

2. Envía una señal de aborto al proceso:

```
# kill -ABRT PID
```

3. Verifique que el núcleo ha sido capturado por **coredumpctl**:

```
$ coredumpctl list PID
```

   Por ejemplo:

```
$ coredumpctl list 5459
TIME                            PID   UID   GID SIG COREFILE  EXE
Thu 2019-11-07 15:14:46 CET    5459  1000  1000   6 present   /usr/bin/bc
```

4. Examine más a fondo o utilice el archivo del núcleo según sea necesario.

Puede especificar el volcado del núcleo por PID y otros valores. Consulte la página del manual *coredumpctl(1)* para obtener más detalles.

- Para mostrar los detalles del archivo del núcleo:

```
$ coredumpctl info PID
```

- Para cargar el archivo del núcleo en el depurador GDB:

```
$ coredumpctl debug PID
```

Dependiendo de la disponibilidad de la información de depuración, GDB sugerirá comandos a ejecutar, como por ejemplo

```
Faltan los debuginfos por separado, utilice: dnf debuginfo-install bc-1.07.1-5.el8.x86_64
```

Para más detalles sobre este proceso, consulte Sección 7.4, "Obtención de paquetes debuginfo para una aplicación o librería usando GDB".

- Para exportar el archivo de núcleo para su posterior procesamiento en otro lugar:

```
$ coredumpctl dump PID > /path/to/file_for_export
```

Sustituya */path/to/file_for_export* por el archivo en el que desea colocar el volcado del núcleo.

10.5. VOLCADO DE LA MEMORIA DEL PROCESO CON GCORE

El flujo de trabajo de depuración de volcado de núcleo permite el análisis del estado del programa fuera de línea. En algunos casos, puede utilizar este flujo de trabajo con un programa que todavía se está ejecutando, como cuando es difícil acceder al entorno con el proceso. Puedes utilizar el comando **gcore** para volcar la memoria de cualquier proceso mientras se está ejecutando.

Requisitos previos

- Debe entender qué son los vertederos de núcleo y cómo se crean.
- GDB debe estar instalado en el sistema.

Procedimiento

1. Averigüe la identificación del proceso (*pid*). Utilice herramientas como **ps**, **pgrep**, y **top**:

```
$ ps -C some-program
```

2. Vuelca la memoria de este proceso:

```
$ gcore -o filename pid
```

Esto crea un archivo ***filename*** y vuelca en él la memoria del proceso. Mientras se vuelca la memoria, se detiene la ejecución del proceso.

3. Una vez finalizado el volcado del núcleo, el proceso reanuda su ejecución normal.

4. Crear un informe SOS para proporcionar información adicional sobre el sistema:

```
# sosreport
```

   Esto crea un archivo tar que contiene información sobre su sistema, como copias de los archivos de configuración.

5. Transfiera el archivo ejecutable del programa, el volcado del núcleo y el informe SOS al ordenador donde se realizará la depuración.

6. Opcional: Elimine el volcado del núcleo y el informe SOS después de transferirlos, para liberar espacio en el disco.

Recursos adicionales

- ¿Cómo obtener un archivo de núcleo sin reiniciar una aplicación?

10.6. VOLCADO DE LA MEMORIA DEL PROCESO PROTEGIDO CON GDB

Se puede marcar la memoria de los procesos como no descargable. Esto puede ahorrar recursos y garantizar una seguridad adicional cuando la memoria del proceso contiene datos sensibles: por ejemplo, en aplicaciones bancarias o de contabilidad o en máquinas virtuales completas. Tanto los volcados del núcleo del kernel (**kdump**) como los volcados manuales del núcleo (**gcore**, GDB) no vuelcan la memoria marcada de esta manera.

En algunos casos, es necesario volcar todo el contenido de la memoria del proceso independientemente de estas protecciones. Este procedimiento muestra cómo hacerlo utilizando el depurador GDB.

Requisitos previos

- Debes entender lo que son los vertederos de núcleo.
- GDB debe estar instalado en el sistema.
- GDB debe estar ya unido al proceso con memoria protegida.

Procedimiento

1. Configurar el GDB para que ignore la configuración del archivo /**proc**/**PID**/**coredump_filter**:

```
(gdb) set use-coredump-filter off
```

2. Configurar GDB para que ignore la bandera de la página de memoria **VM_DONTDUMP**:

```
(gdb) set dump-excluded-mappings on
```

3. Vuelca la memoria:

```
(gdb) gcore core-file
```

   Sustituye *core-file* por el nombre del archivo donde quieres volcar la memoria.

Recursos adicionales

- Depuración con GDB - Cómo producir un archivo de núcleo de su programa

CAPÍTULO 11. CAMBIOS QUE ROMPEN LA COMPATIBILIDAD EN GDB

La versión de GDB provista en Red Hat Enterprise Linux 8 contiene un número de cambios que rompen la compatibilidad, especialmente para los casos donde la salida de GDB es leída directamente desde la terminal. Las siguientes secciones proporcionan más detalles sobre estos cambios.

No se recomienda analizar la salida de GDB. Prefiera las secuencias de comandos que utilizan la API de GDB de Python o la interfaz de máquina de GDB (MI).

GDBserver ahora arranca los inferiores con el shell

Para permitir la expansión y la sustitución de variables en los argumentos de la línea de comandos inferior, GDBserver ahora inicia el inferior en un shell, igual que GDB.

Para desactivar el uso del shell:

- Cuando utilice el comando **target extended-remote** GDB, desactive el shell con el comando **set startup-with-shell off**.
- Cuando utilice el comando **target remote** GDB, desactive el shell con la opción **--no-startup-with-shell** de GDBserver.

Ejemplo 11.1. Ejemplo de expansión del shell en las inferencias remotas de GDB

Este ejemplo muestra cómo la ejecución del comando **/bin/echo /*** a través de GDBserver difiere en las versiones 7 y 8 de Red Hat Enterprise Linux:

- En RHEL 7:

```
$ gdbserver --multi :1234
$ gdb -batch -ex 'target extended-remote :1234' -ex 'set remote exec-file /bin/echo' -ex
'file /bin/echo' -ex 'run /*'
/*
```

- En RHEL 8:

```
$ gdbserver --multi :1234
$ gdb -batch -ex 'target extended-remote :1234' -ex 'set remote exec-file /bin/echo' -ex
'file /bin/echo' -ex 'run /*'
/bin /boot (...) /tmp /usr /var
```

gcj soporte eliminado

Se ha eliminado el soporte para la depuración de programas Java compilados con el compilador GNU para Java (**gcj**).

Nueva sintaxis para los comandos de mantenimiento de volcado de símbolos

La sintaxis de los comandos de mantenimiento de volcado de símbolos ahora incluye opciones antes de los nombres de los archivos. Como resultado, los comandos que funcionaban con GDB en RHEL 7 no funcionan en RHEL 8.

Como ejemplo, el siguiente comando ya no almacena los símbolos en un archivo, sino que produce un mensaje de error:

```
(gdb) mantenimiento imprimir símbolos /tmp/out main.c
```

La nueva sintaxis para los comandos de mantenimiento de volcado de símbolos es:

```
maint print symbols [-pc address] [--] [filename]
maint print symbols [-objfile objfile] [-source source] [--] [filename]
maint print psymbols [-objfile objfile] [-pc address] [--] [filename]
maint print psymbols [-objfile objfile] [-source source] [--] [filename]
maint print msymbols [-objfile objfile] [--] [filename]
```

Los números de hilo ya no son globales

Anteriormente, GDB sólo utilizaba la numeración global de los hilos. La numeración se ha ampliado para mostrarse por inferior en la forma **inferior_num.thread_num**, como **2.1**. Como consecuencia, los números de hilos en la variable de conveniencia **$_thread** y en el atributo Python **InferiorThread.num** ya no son únicos entre inferiores.

GDB ahora almacena un segundo ID de hilo por hilo, llamado ID de hilo global, que es el nuevo equivalente a los números de hilo de las versiones anteriores. Para acceder al número de hilo global, utilice la variable de conveniencia **$_gthread** y el atributo de Python **InferiorThread.global_num**.

Por compatibilidad con el pasado, los ID de los hilos de la interfaz de la máquina (MI) siempre contienen los ID globales.

Ejemplo 11.2. Ejemplo de cambios en el número de hilos de GDB

En Red Hat Enterprise Linux 7:

```
# debuginfo-install coreutils
$ gdb -batch -ex 'file echo' -ex start -ex 'add-inferior' -ex 'inferior 2' -ex 'file echo' -ex start -ex 'info
threads' -ex 'pring $_thread' -ex 'inferior 1' -ex 'pring $_thread'
(...)
  Id   Target Id         Frame
* 2    process 203923 "echo" main (argc=1, argv=0x7fffffffdb88) at src/echo.c:109
  1    process 203914 "echo" main (argc=1, argv=0x7fffffffdb88) at src/echo.c:109
$1 = 2
(...)
$2 = 1
```

En Red Hat Enterprise Linux 8:

```
# dnf debuginfo-install coreutils
$ gdb -batch -ex 'file echo' -ex start -ex 'add-inferior' -ex 'inferior 2' -ex 'file echo' -ex start -ex 'info
threads' -ex 'pring $_thread' -ex 'inferior 1' -ex 'pring $_thread'
(...)
  Id   Target Id         Frame
  1.1  process 4106488 "echo" main (argc=1, argv=0x7fffffffce58) at ../src/echo.c:109
* 2.1  process 4106494 "echo" main (argc=1, argv=0x7fffffffce58) at ../src/echo.c:109
$1 = 1
(...)
$2 = 1
```

La memoria para los contenidos de valores puede ser limitada

Anteriormente, GDB no limitaba la cantidad de memoria asignada para el contenido de los valores.

Como consecuencia, la depuración de programas incorrectos podía hacer que GDB asignara demasiada memoria. Se ha añadido el ajuste **max-value-size** para permitir limitar la cantidad de memoria asignada. El valor por defecto de este límite es de 64 KiB. Como resultado, GDB en Red Hat Enterprise Linux 8 no mostrará valores demasiado grandes, sino que informará que el valor es demasiado grande.

Como ejemplo, la impresión de un valor definido como **char s[128*1024];** produce resultados diferentes:

- En Red Hat Enterprise Linux 7, **$1 = 'A' <repeats 131072 times>**
- En Red Hat Enterprise Linux 8, **value requires 131072 bytes, which is more than max-value-size**

Ya no se admite la versión Sun del formato stabs

Se ha eliminado el soporte para la versión de Sun del formato de archivo de depuración **stabs**. El formato **stabs** producido por GCC en RHEL con la opción **gcc -gstabs** sigue siendo soportado por GDB.

Cambios en el manejo de Sysroot

El comando **set sysroot *path*** especifica la raíz del sistema cuando se buscan los archivos necesarios para la depuración. Los nombres de directorios suministrados a este comando ahora pueden llevar como prefijo la cadena **target:** para que GDB lea las bibliotecas compartidas del sistema de destino (tanto local como remoto). El prefijo anteriormente disponible **remote:** se trata ahora como **target:**. Además, el valor de la raíz del sistema por defecto ha cambiado de una cadena vacía a **target:** por compatibilidad con versiones anteriores.

La raíz del sistema especificada se antepone al nombre del archivo del ejecutable principal, cuando GDB inicia procesos de forma remota, o cuando se adjunta a procesos ya en ejecución (tanto locales como remotos). Esto significa que para los procesos remotos, el valor por defecto **target:** hace que GDB siempre intente cargar la información de depuración desde el sistema remoto. Para evitar esto, ejecute el comando **set sysroot** antes del comando **target remote** para que los archivos de símbolos locales se encuentren antes que los remotos.

HISTSIZE ya no controla el tamaño del historial de comandos de GDB

Anteriormente, GDB utilizaba la variable de entorno **HISTSIZE** para determinar el tiempo que debía mantenerse el historial de comandos. GDB ha sido modificado para utilizar la variable de entorno **GDBHISTSIZE** en su lugar. Esta variable es específica sólo para GDB. Los valores posibles y sus efectos son:

- un número positivo - utilizar el historial de comandos de este tamaño,
- **-1** o una cadena vacía - guarda el historial de todos los comandos,
- valores no numéricos - se ignoran.

Limitación de finalización añadida

Ahora se puede limitar el número máximo de candidatos considerados durante la finalización mediante el comando **set max-completions**. Para mostrar el límite actual, ejecute el comando **show max-completions**. El valor por defecto es 200. Este límite evita que GDB genere listas de finalización excesivamente grandes y deje de responder.

Como ejemplo, la salida después de la entrada **p <tab><tab>** es:

- en RHEL 7 **Display all 29863 possibilities? (y or n)**
- en RHEL 8 **Display all 200 possibilities? (y or n)**

Eliminado el modo de compatibilidad HP-UX XDB

La opción **-xdb** para el modo de compatibilidad HP-UX XDB ha sido eliminada de GDB.

Manejo de señales para hilos

Anteriormente, GDB podía entregar una señal al hilo actual en lugar de al hilo para el que realmente se enviaba la señal. Este error ha sido corregido, y GDB ahora siempre pasa la señal al hilo correcto al reanudar la ejecución.

Además, el comando **signal** ahora siempre entrega correctamente la señal solicitada al hilo actual. Si el programa se detiene por una señal y el usuario cambia de hilo, GDB pide confirmación.

Modos de puntos de interrupción siempre insertados y desactivados y auto fusionados

Se ha modificado el valor **breakpoint always-inserted**. Se ha eliminado el valor **auto** y el comportamiento correspondiente. El valor por defecto es ahora **off**. Además, el valor **off** ahora hace que GDB no elimine los puntos de interrupción del objetivo hasta que todos los hilos se detengan.

los comandos de remotebaud ya no son compatibles

Los comandos **set remotebaud** y **show remotebaud** ya no son compatibles. Utilice en su lugar los comandos **set serial baud** y **show serial baud**.

PARTE III. HERRAMIENTAS ADICIONALES PARA EL DESARROLLO

Además de las herramientas relacionadas con el desarrollo que están disponibles como parte del sistema operativo, los desarrolladores pueden instalar conjuntos de herramientas adicionales en Red Hat Enterprise Linux. Estos conjuntos de herramientas pueden contener herramientas para diferentes lenguajes, cadenas de herramientas alternativas o versiones alternativas de herramientas del sistema.

CAPÍTULO 12. USO DEL CONJUNTO DE HERRAMIENTAS GCC

12.1. QUÉ ES EL CONJUNTO DE HERRAMIENTAS GCC

Red Hat Enterprise Linux 8 presenta GCC Toolset, un flujo de aplicaciones que contiene versiones más actualizadas de herramientas de desarrollo y análisis de rendimiento. GCC Toolset es similar a Red Hat Developer Toolset para RHEL 7.

GCC Toolset está disponible como un flujo de aplicaciones en forma de una colección de software en el repositorio **AppStream**. GCC Toolset está totalmente soportado bajo los Acuerdos de Nivel de Suscripción de Red Hat Enterprise Linux, es funcionalmente completo y está destinado al uso en producción. Las aplicaciones y bibliotecas proporcionadas por GCC Toolset no reemplazan las versiones del sistema de Red Hat Enterprise Linux, no las anulan y no se convierten automáticamente en opciones predeterminadas o preferidas. Utilizando un marco de trabajo llamado colecciones de software, un conjunto adicional de herramientas para desarrolladores se instala en el directorio **/opt/** y es habilitado explícitamente por el usuario bajo demanda utilizando la utilidad **scl**. A menos que se indique lo contrario para herramientas o características específicas, GCC Toolset está disponible para todas las arquitecturas soportadas por Red Hat Enterprise Linux.

12.2. INSTALACIÓN DEL CONJUNTO DE HERRAMIENTAS GCC

Al instalar el conjunto de herramientas GCC en un sistema se instalan las herramientas principales y todas las dependencias necesarias. Tenga en cuenta que algunas partes del conjunto de herramientas no se instalan por defecto y deben instalarse por separado.

Procedimiento

- Para instalar la versión de GCC Toolset *N*:

```
# yum install gcc-toolset-N
```

12.3. INSTALACIÓN DE PAQUETES INDIVIDUALES DE GCC TOOLSET

Para instalar sólo ciertas herramientas del conjunto de herramientas de GCC en lugar de todo el conjunto de herramientas, liste los paquetes disponibles e instale los seleccionados con la herramienta de gestión de paquetes **yum**. Este procedimiento es útil también para los paquetes que no se instalan por defecto con el conjunto de herramientas.

Procedimiento

1. Lista de los paquetes disponibles en la versión de GCC Toolset *N*:

```
$ yum list available gcc-toolset-N-\N-
```

2. Para instalar cualquiera de estos paquetes:

```
# yum install package_name
```

Sustituya *package_name* por una lista separada por espacios de los paquetes a instalar. Por ejemplo, para instalar los paquetes **gcc-toolset-9-gdb-gdbserver** y **gcc-toolset-9-gdb-doc**:

```
# yum install gcc-toolset-9-gdb-gdbserver gcc-toolset-9-gdb-doc
```

12.4. DESINSTALACIÓN DE GCC TOOLSET

Para eliminar GCC Toolset de su sistema, desinstálelo mediante la herramienta de gestión de paquetes **yum**.

Procedimiento

- Para desinstalar la versión de GCC Toolset *N*:

```
# yum remove gcc-toolset-M\N - yum remove gcc-toolset-
```

12.5. EJECUTAR UNA HERRAMIENTA DE GCC TOOLSET

Para ejecutar una herramienta de GCC Toolset, utilice la utilidad **scl**.

Procedimiento

- Para ejecutar una herramienta de la versión de GCC Toolset *N*:

```
$ scl enable gcc-toolset-N tool
```

12.6. EJECUCIÓN DE UNA SESIÓN DE SHELL CON GCC TOOLSET

GCC Toolset permite ejecutar una sesión de shell en la que se utilizan las versiones de las herramientas de GCC Toolset en lugar de las versiones del sistema de estas herramientas, sin utilizar explícitamente el comando **scl**. Esto es útil cuando se necesita iniciar interactivamente las herramientas muchas veces, como cuando se configura o prueba una configuración de desarrollo.

Procedimiento

- Para ejecutar una sesión de shell en la que las versiones de las herramientas del conjunto de herramientas de GCC *N* anulan las versiones del sistema de estas herramientas:

```
$ scl enable gcc-toolset-N bash
```

12.7. INFORMACIÓN RELACIONADA

- Manual del usuario del conjunto de herramientas para desarrolladores de Red Hat

CAPÍTULO 13. CONJUNTO DE HERRAMIENTAS GCC 9

Este capítulo proporciona información específica sobre la versión 9 de GCC Toolset y las herramientas que contiene esta versión.

13.1. HERRAMIENTAS Y VERSIONES PROPORCIONADAS POR GCC TOOLSET 9

GCC Toolset 9 proporciona las siguientes herramientas y versiones:

Tabla 13.1. Versiones de herramientas en GCC Toolset 9

Nombre	Versión	Descripción

Nombre	Versión	Descripción
GCC	9.2.1	Un conjunto de compiladores soportátiles con soporte para a C, C, y

Nombre	Versión	Descripción
GDB	8.3	Un depurador de línea de comandos para programas escritos en

Nombre	Versión	Descripción
		an.
Valgrind	3.15.0	Un marco de instrumentación y una serie de herramientas para

Nombre	Versión	Descripción
		aplicaciones con el fin de detectar errores de memoria, identificar

Nombre	Versión	Descripción
		proporciona una gestión de memoria eficiente de cualquier uso de argumentos in

Nombre	Versión	Descripción
		en las llamadas al sistema.
SystemTap	4.1	Una herramienta de rastreo y sondeo para

Nombre	Versión	Descripción
		rsuputilvis ar las actividades de todo el sistema sin necesidad de instrumentar,

Nombre	Versión	Descripción
		rcomprobar, instalar y reiniciar.
Dyninst	10.1.0	Una biblioteca para instrumentar y trabajar con lo

Nombre	Versión	Descripción
		s d e l e s p a c i o d e u s u a r i o d u r a n t e s u e j e c u c i ó n.
binutils	2.32	U n a c o l e c c i ó n d e

Nombre	Versión	Descripción
		hetrans binarias y otras utilidades para inspeccionar y manipular archivo

Nombre	Versión	Descripción
		s objetos y binarios.
elfutils	0.176	Una colección de herramientas binarias y otras utilidades espa

Nombre	Versión	Descripción
		rinsperymanipulararchivosELF.
dwz	0.12	Unaherramientaparaoptimizarlain

Nombre	Versión	Descripción
		fuerte dependuraciónDWARFcontenidaenlasbibliotecascompartidasELF

Nombre	Versión	Descripción
		y enlaces a tablas ELF por su tamaño.
hacer	4.2.1	Una herramienta de automatización d

Nombre	Versión	Descripción
		ción con seguimiento de dependencias.
strace	5.1	Una herramienta de dep

Nombre	Versión	Descripción
		una acontrolar las llamadas al sistema que utiliza un programa y las señal

Nombre	Versión	Descripción
		e.
ltrace	0.7.91	Una herramienta de depuración para mostrar las llamadas a bibliotecas dinámicas.

Nombre	Versión	Descripción
		tecnicas que realiza un programa. También puede supervisar las llama

Nombre	Versión	Descripción
		ejecutadas por los programas.

Nombre	Versión	Descripción
annobin	9.08	Una herramienta de comprobación de las seguridad de la construc

Nombre	Versión	Descripción

13.2. COMPATIBILIDAD CON C EN GCC TOOLSET 9

IMPORTANTE

La información de compatibilidad que se presenta aquí se aplica únicamente al GCC del GCC Toolset 9.

El compilador de GCC en GCC Toolset puede utilizar los siguientes estándares de C:

C 14

Esta es la configuración estándar del lenguaje **default** para el conjunto de herramientas GCC 9, con extensiones GNU, equivalente a utilizar explícitamente la opción **-std=gnu 14**.
El uso de la versión del lenguaje C 14 es compatible cuando todos los objetos C compilados con la bandera respectiva han sido construidos utilizando la versión 6 de GCC o posterior.

C 11

Este lenguaje estándar está disponible en GCC Toolset 9.
El uso de la versión del lenguaje C 11 es compatible cuando todos los objetos C compilados con la bandera respectiva han sido construidos utilizando la versión 5 de GCC o posterior.

C 98

Este estándar de lenguaje está disponible en GCC Toolset 9. Los binarios, las bibliotecas compartidas y los objetos construidos con este estándar pueden mezclarse libremente independientemente de que se hayan construido con GCC desde GCC Toolset, Red Hat Developer Toolset y RHEL 5, 6, 7 y 8.

C 17, C 2a

Estos estándares de lenguaje están disponibles en GCC Toolset 9 sólo como una capacidad experimental, inestable y no soportada. Además, no se puede garantizar la compatibilidad de los objetos, archivos binarios y bibliotecas creados con estos estándares.

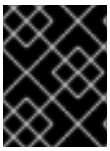
Todos los estándares del lenguaje están disponibles tanto en su variante estándar como con extensiones GNU.

Cuando se mezclan objetos construidos con GCC Toolset con los construidos con el toolchain de RHEL (particularmente **.o** o **.a** archivos), se debe utilizar el toolchain de GCC Toolset para cualquier enlace. Esto asegura que cualquier característica más nueva de la biblioteca proporcionada sólo por GCC Toolset se resuelve en el momento del enlace.

13.3. PARTICULARIDADES DE GCC EN GCC TOOLSET 9

Vinculación estática de las bibliotecas

Ciertas características más recientes de las bibliotecas están enlazadas estáticamente en las aplicaciones construidas con GCC Toolset para soportar la ejecución en múltiples versiones de Red Hat Enterprise Linux. Esto crea un riesgo de seguridad menor adicional ya que las erratas estándar de Red Hat Enterprise Linux no cambian este código. Si surge la necesidad de que los desarrolladores reconstruyan sus aplicaciones debido a este riesgo, Red Hat lo comunicará mediante una errata de seguridad.



IMPORTANTE

Debido a este riesgo de seguridad adicional, se recomienda encarecidamente a los desarrolladores que no enlacen estáticamente toda su aplicación por las mismas razones.

Especificar las bibliotecas después de los archivos de objetos al enlazar

En el conjunto de herramientas GCC, las bibliotecas se enlazan utilizando scripts de enlace que pueden especificar algunos símbolos a través de archivos estáticos. Esto es necesario para asegurar la compatibilidad con múltiples versiones de Red Hat Enterprise Linux. Sin embargo, los scripts del enlazador utilizan los nombres de los respectivos archivos de objetos compartidos. Como consecuencia, el enlazador utiliza reglas de manejo de símbolos diferentes a las esperadas y no reconoce los símbolos requeridos por los archivos de objetos cuando la opción que añade la biblioteca se especifica antes de las opciones que especifican los archivos de objetos:

```
$ scl enable gcc-toolset-9 'gcc -lsomelib objfile.o'
```

El uso de una biblioteca de GCC Toolset de esta manera resulta en el mensaje de error del enlazador **undefined reference to symbol**. Para evitar este problema, siga la práctica de enlazado estándar y especifique la opción de añadir la biblioteca después de las opciones que especifican los archivos objeto:

```
$ scl enable gcc-toolset-9 'gcc objfile.o -lsomelib'
```

Tenga en cuenta que esta recomendación también se aplica cuando se utiliza la versión básica de Red Hat Enterprise Linux de **GCC**.

13.4. PARTICULARIDADES DE BINUTILS EN GCC TOOLSET 9

Vinculación estática de las bibliotecas

Ciertas características más recientes de las bibliotecas están enlazadas estáticamente en las aplicaciones construidas con GCC Toolset para soportar la ejecución en múltiples versiones de Red Hat Enterprise Linux. Esto crea un riesgo de seguridad menor adicional ya que las erratas estándar de Red Hat Enterprise Linux no cambian este código. Si surge la necesidad de que los desarrolladores reconstruyan sus aplicaciones debido a este riesgo, Red Hat lo comunicará mediante una errata de seguridad.

IMPORTANTE

Debido a este riesgo de seguridad adicional, se recomienda encarecidamente a los desarrolladores que no enlacen estáticamente toda su aplicación por las mismas razones.

Especificar las bibliotecas después de los archivos de objetos al enlazar

En el conjunto de herramientas GCC, las bibliotecas se enlazan utilizando scripts de enlace que pueden especificar algunos símbolos a través de archivos estáticos. Esto es necesario para asegurar la compatibilidad con múltiples versiones de Red Hat Enterprise Linux. Sin embargo, los scripts del enlazador utilizan los nombres de los respectivos archivos de objetos compartidos. Como consecuencia, el enlazador utiliza reglas de manejo de símbolos diferentes a las esperadas y no reconoce los símbolos requeridos por los archivos de objetos cuando la opción que añade la biblioteca se especifica antes de las opciones que especifican los archivos de objetos:

```
$ scl enable gcc-toolset-9 'ld -lsomelib objfile.o'
```

El uso de una biblioteca de GCC Toolset de esta manera resulta en el mensaje de error del enlazador **undefined reference to symbol**. Para evitar este problema, siga la práctica de enlazado estándar y especifique la opción de añadir la biblioteca después de las opciones que especifican los archivos de objetos:

```
$ scl enable gcc-toolset-9 'ld objfile.o -lsomelib'
```

Tenga en cuenta que esta recomendación también se aplica cuando se utiliza la versión básica de Red Hat Enterprise Linux de **binutils**.

CAPÍTULO 14. CONJUNTO DE HERRAMIENTAS GCC 10

Este capítulo proporciona información específica sobre la versión 10 de GCC Toolset y las herramientas que contiene esta versión.

14.1. HERRAMIENTAS Y VERSIONES PROPORCIONADAS POR GCC TOOLSET 10

GCC Toolset 10 proporciona las siguientes herramientas y versiones:

Tabla 14.1. Versiones de herramientas en GCC Toolset 10

Nombre	Versión	Descripción

Nombre	Versión	Descripción
GCC	10.2.1	Un conjunto de compiladores soportátiles con soporte para C, C, y

Nombre	Versión	Descripción
GDB	9.2	Un depurador de línea de comandos para aprogramas escritos en

Nombre	Versión	Descripción
		an.
Valgrind	3.16.0	Un marco de instrumentación y una serie de herramientas para

Nombre	Versión	Descripción
		aplicaciones con el fin de detectar errores de memoria, identificar

Nombre	Versión	Descripción
		proporciona una gestión de memoria eficiente para cualquier uso de argumentos in

Nombre	Versión	Descripción
		en las llamadas al sistema.
SystemTap	4.3	Una herramienta de rastreo y sondeo para

Nombre	Versión	Descripción
		rusuplovisar las actividades de todo el sistema sin necesidad de instrumentar,

Nombre	Versión	Descripción
		rcompilar, instalar y reiniciar.
Dyninst	10.1.0	Una biblioteca para instrumentar y trabajar con lo

Nombre	Versión	Descripción
		s d e l e s p a c i o d e u s u a r i o d u r a n t e s u e j e c u c i ó n.
binutils	2.35	U n a c o l e c c i ó n d e

Nombre	Versión	Descripción
		hebitrabinarias y otras utilidades para inspeccionar y manipular archivo

Nombre	Versión	Descripción
		s u b j e t o s y b i n a r i o s.
elfutils	0.180	U n a c o l e c c i ó n d e h e r r a m i e n t a s b i n a r i a s y o t r a s u t i l i d a d e s p a

Nombre	Versión	Descripción
		rmsprocesionarymanipulararchivosELF.
dwz	0.12	Unaherramientaparaoptimizarlain

Nombre	Versión	Descripción
		formación de depuración DWARF contenida en las bibliotecas compartidas ELF

Nombre	Versión	Descripción y enlaces a tablas ELF por su tamaño.
hacer	4.2.1	Una herramienta de automatización d

Nombre	Versión	Descripción
		ción con seguimiento de dependencias.
strace	5.7	Una herramienta de dep

Nombre	Versión	Descripción
		a controlar las llamadas al sistema que utiliza un programa y las señal

Nombre	Versión	Descripción
		e.
ltrace	0.7.91	Una herramienta de depuración para mostrar las llamadas a bibliotecas dinámicas.

Nombre	Versión	Descripción
		tecnicas que realiza un programa. También puede supervisar las llama

Nombre	Versión	Descripción
		ejecutadas por los programas.

Nombre	Versión	Descripción
annobin	9.29	Una herramienta de comprobación de las seguridad de la construc

Nombre	Versión	Descripción

14.2. COMPATIBILIDAD CON C EN GCC TOOLSET 10

IMPORTANTE

La información de compatibilidad que se presenta aquí se aplica sólo al GCC del GCC Toolset 10.

El compilador de GCC en GCC Toolset puede utilizar los siguientes estándares de C:

C 14

Esta es la configuración estándar del lenguaje **default** para el conjunto de herramientas GCC 10, con extensiones GNU, equivalente a utilizar explícitamente la opción **-std=gnu 14**.
El uso de la versión del lenguaje C 14 es compatible cuando todos los objetos C compilados con la bandera respectiva han sido construidos utilizando la versión 6 de GCC o posterior.

C 11

Este lenguaje estándar está disponible en GCC Toolset 10.
El uso de la versión del lenguaje C 11 es compatible cuando todos los objetos C compilados con la bandera respectiva han sido construidos utilizando la versión 5 de GCC o posterior.

C 98

Este estándar de lenguaje está disponible en GCC Toolset 10. Los binarios, las bibliotecas compartidas y los objetos construidos con este estándar pueden mezclarse libremente independientemente de que se hayan construido con GCC desde GCC Toolset, Red Hat Developer Toolset y RHEL 5, 6, 7 y 8.

C 17

Este lenguaje estándar está disponible en GCC Toolset 10.

C 20

Este estándar de lenguaje está disponible en GCC Toolset 10 sólo como una capacidad experimental, inestable y no soportada. Además, no se puede garantizar la compatibilidad de los objetos, archivos binarios y bibliotecas creados con este estándar.

Todos los estándares del lenguaje están disponibles tanto en su variante estándar como con extensiones GNU.

Cuando se mezclan objetos construidos con GCC Toolset con los construidos con el toolchain de RHEL (particularmente **.o** o **.a** archivos), se debe utilizar el toolchain de GCC Toolset para cualquier enlace. Esto asegura que cualquier característica más nueva de la biblioteca proporcionada sólo por GCC Toolset se resuelve en el momento del enlace.

14.3. ESPECÍFICOS DE GCC EN GCC TOOLSET 10

Vinculación estática de las bibliotecas

Ciertas características más recientes de las bibliotecas están enlazadas estáticamente en las aplicaciones construidas con GCC Toolset para soportar la ejecución en múltiples versiones de Red Hat Enterprise Linux. Esto crea un riesgo de seguridad menor adicional ya que las erratas estándar de Red

Hat Enterprise Linux no cambian este código. Si surge la necesidad de que los desarrolladores reconstruyan sus aplicaciones debido a este riesgo, Red Hat lo comunicará mediante una errata de seguridad.

IMPORTANTE

Debido a este riesgo de seguridad adicional, se recomienda encarecidamente a los desarrolladores que no enlacen estáticamente toda su aplicación por las mismas razones.

Especificar las bibliotecas después de los archivos de objetos al enlazar

En el conjunto de herramientas GCC, las bibliotecas se enlazan utilizando scripts de enlace que pueden especificar algunos símbolos a través de archivos estáticos. Esto es necesario para asegurar la compatibilidad con múltiples versiones de Red Hat Enterprise Linux. Sin embargo, los scripts del enlazador utilizan los nombres de los respectivos archivos de objetos compartidos. Como consecuencia, el enlazador utiliza reglas de manejo de símbolos diferentes a las esperadas y no reconoce los símbolos requeridos por los archivos de objetos cuando la opción que añade la biblioteca se especifica antes de las opciones que especifican los archivos de objetos:

```
$ scl enable gcc-toolset-10 'gcc -lsomelib objfile.o'
```

El uso de una biblioteca de GCC Toolset de esta manera resulta en el mensaje de error del enlazador **undefined reference to symbol**. Para evitar este problema, siga la práctica de enlazado estándar y especifique la opción de añadir la biblioteca después de las opciones que especifican los archivos objeto:

```
$ scl enable gcc-toolset-10 'gcc objfile.o -lsomelib'
```

Tenga en cuenta que esta recomendación también se aplica cuando se utiliza la versión básica de Red Hat Enterprise Linux de **GCC**.

14.4. PARTICULARIDADES DE BINUTILS EN GCC TOOLSET 10

Vinculación estática de las bibliotecas

Ciertas características más recientes de las bibliotecas están enlazadas estáticamente en las aplicaciones construidas con GCC Toolset para soportar la ejecución en múltiples versiones de Red Hat Enterprise Linux. Esto crea un riesgo de seguridad menor adicional ya que las erratas estándar de Red Hat Enterprise Linux no cambian este código. Si surge la necesidad de que los desarrolladores reconstruyan sus aplicaciones debido a este riesgo, Red Hat lo comunicará mediante una errata de seguridad.

IMPORTANTE

Debido a este riesgo de seguridad adicional, se recomienda encarecidamente a los desarrolladores que no enlacen estáticamente toda su aplicación por las mismas razones.

Especificar las bibliotecas después de los archivos de objetos al enlazar

En el conjunto de herramientas GCC, las bibliotecas se enlazan utilizando scripts de enlace que pueden especificar algunos símbolos a través de archivos estáticos. Esto es necesario para asegurar la compatibilidad con múltiples versiones de Red Hat Enterprise Linux. Sin embargo, los scripts del enlazador utilizan los nombres de los respectivos archivos de objetos compartidos. Como consecuencia,

el enlazador utiliza reglas de manejo de símbolos diferentes a las esperadas y no reconoce los símbolos requeridos por los archivos de objetos cuando la opción que añade la biblioteca se especifica antes de las opciones que especifican los archivos de objetos:

```
$ scl enable gcc-toolset-10 'ld -lsomelib objfile.o'
```

El uso de una biblioteca de GCC Toolset de esta manera resulta en el mensaje de error del enlazador **undefined reference to symbol**. Para evitar este problema, siga la práctica de enlazado estándar y especifique la opción de añadir la biblioteca después de las opciones que especifican los archivos de objetos:

```
$ scl enable gcc-toolset-10 'ld objfile.o -lsomelib'
```

Tenga en cuenta que esta recomendación también se aplica cuando se utiliza la versión básica de Red Hat Enterprise Linux de **binutils**.

CAPÍTULO 15. USO DE LAS IMÁGENES CONTENEDORAS DE GCC TOOLSET

Los componentes de GCC Toolset 10 están disponibles en las dos imágenes de contenedores:

- Cadena de herramientas GCC 10
- GCC Toolset 10 Perftools

Las imágenes del contenedor GCC Toolset se basan en la imagen base **rhel8** y están disponibles para todas las arquitecturas soportadas por RHEL 8:

- Arquitecturas de 64 bits de AMD e Intel
- La arquitectura ARM de 64 bits
- IBM Power Systems, Little Endian
- IBM Z

15.1. CONTENIDO DE LAS IMÁGENES DEL CONTENEDOR DE GCC TOOLSET

Las versiones de las herramientas proporcionadas en las imágenes del contenedor de GCC Toolset 10 coinciden con las versiones de los componentes de GCC Toolset 10 .

Contenido de la cadena de herramientas GCC Toolset 10

La imagen **rhel8/gcc-toolset-10-toolchain** proporciona el compilador GCC, el depurador GDB y otras herramientas relacionadas con el desarrollo. La imagen contenedora consta de los siguientes componentes:

Componente	Paquete
gcc	gcc-toolset-10-gcc
g	gcc-toolset-10-gcc-c
gfortran	gcc-toolset-10-gcc-gfortran
gdb	gcc-toolset-10-gdb

El contenido de GCC Toolset 10 Perftools

La imagen **rhel8/gcc-toolset-10-perftools** proporciona una serie de herramientas para la depuración, la supervisión del rendimiento y el análisis posterior de las aplicaciones. La imagen del contenedor consta de los siguientes componentes:

Componente	Paquete
Valgrind	gcc-toolset-10-valgrind

Componente	Paquete
SystemTap	gcc-toolset-10-systemtap
Dyninst	gcc-toolset-10-dyninst
elfutils	gcc-toolset-10-elfutils

Recursos adicionales

- Para utilizar los componentes de GCC Toolset en RHEL 7, utilice Red Hat Developer Toolset que proporciona herramientas de desarrollo similares para los usuarios de RHEL 7
- Instrucciones sobre el uso de las imágenes de contenedor de Red Hat Developer Toolset en RHEL 7

15.2. ACCESO Y EJECUCIÓN DE LAS IMÁGENES CONTENEDORAS DE GCC TOOLSET

La siguiente sección describe cómo acceder y ejecutar las imágenes de contenedor de GCC Toolset.

Requisitos previos

- Podman está instalado.

Procedimiento

1. Acceda al Red Hat Container Registry utilizando sus credenciales del Portal del Cliente:

```
$ podman login registry.redhat.io
Username: username
Password: ********
```

2. Extraiga una imagen de contenedor que necesite ejecutando un comando relevante como root:

```
# podman pull registry.redhat.io/rhel8/gcc-toolset-10-toolchain
```

```
# podman pull registry.redhat.io/rhel8/gcc-toolset-10-perftools
```

NOTA

En RHEL 8.1 y versiones posteriores, puede configurar su sistema para trabajar con contenedores como usuario no root. Para más detalles, consulte Ejecutar contenedores como root o sin root.

3. Opcional: Compruebe que la extracción se ha realizado con éxito ejecutando un comando que liste todas las imágenes de contenedores en su sistema local:

```
# imágenes de podman
```

4. Ejecutar un contenedor lanzando un shell bash dentro de un contenedor:

```
# podman run -it image_name /bin/bash
```

La opción **-i** crea una sesión interactiva; sin esta opción el shell se abre y sale instantáneamente.

La opción **-t** abre una sesión de terminal; sin esta opción no se puede escribir nada en el shell.

Recursos adicionales

- Creación, ejecución y gestión de contenedores Linux en RHEL 8
- Un artículo del blog de Red Hat
- Entradas en el Registro de Contenedores de Red Hat

15.3. EJEMPLO: USO DE LA IMAGEN CONTENEDORA DE GCC TOOLSET 10 TOOLCHAIN

Este ejemplo muestra cómo extraer y empezar a utilizar la imagen contenedora del GCC Toolset 10 Toolchain.

Requisitos previos

- Podman está instalado.

Procedimiento

1. Acceda al Red Hat Container Registry utilizando sus credenciales del Portal del Cliente:

```
$ podman login registry.redhat.io
Username: username
Password: ********
```

2. Extrae la imagen del contenedor como root:

```
# podman pull registry.redhat.io/rhel8/gcc-toolset-10-toolchain
```

3. Inicie la imagen del contenedor con un shell interactivo como root:

```
# podman run -it registry.redhat.io/rhel8/gcc-toolset-10-toolchain /bin/bash
```

4. Ejecute las herramientas del conjunto de herramientas GCC como se espera. Por ejemplo, para verificar la versión del compilador **gcc**, ejecute:

```
bash-4.4$ gcc -v
...
gcc version 10.2.1 20200804 (Red Hat 10.2.1-2) (GCC)
```

5. Para listar todos los paquetes proporcionados en el contenedor, ejecute

```
bash-4.4$ rpm -qa
```

CAPÍTULO 16. CONJUNTOS DE HERRAMIENTAS DE COMPILACIÓN

RHEL 8.0 proporciona los siguientes conjuntos de herramientas de compilación como flujos de aplicaciones:

- LLVM Toolset 9.0.1, que proporciona el marco de infraestructura del compilador LLVM, el compilador Clang para los lenguajes C y C, el depurador LLDB y herramientas relacionadas para el análisis de código. Consulte la guía de uso del conjunto de herramientas LLVM.
- Rust Toolset 1.41, que proporciona el compilador del lenguaje de programación Rust **rustc**, la herramienta de construcción y el gestor de dependencias **cargo**, el plugin **cargo-vendor** y las bibliotecas necesarias. Consulte la guía Using Rust Tools et.
- Go Toolset 1.13, que proporciona las herramientas y bibliotecas del lenguaje de programación Go. Go se conoce también como **golang**. Consulte la guía de uso de Go Toolset.

CAPÍTULO 17. EL PROYECTO ANNOBIN

El proyecto Annobin es una implementación del proyecto de especificación Watermark. El proyecto de especificación Watermark pretende añadir marcadores a los objetos de formato ejecutable y enlazable (ELF) para determinar sus propiedades. El proyecto Annobin consiste en el plugin **annobin** y el programa **annockeck**.

El plugin **annobin** analiza la línea de comandos de la Colección de Compiladores de GNU (GCC), el estado de compilación y el proceso de compilación, y genera las notas ELF. Las notas ELF registran cómo se construyó el binario y proporcionan información para que el programa **annocheck** realice comprobaciones de seguridad.

El comprobador de seguridad forma parte del programa **annocheck** y está activado por defecto. Comprueba los archivos binarios para determinar si el programa se construyó con las opciones de endurecimiento de seguridad necesarias y se compiló correctamente. **annocheck** es capaz de escanear recursivamente directorios, archivos y paquetes RPM en busca de archivos objeto ELF.

NOTA

Los archivos deben estar en formato ELF. **annocheck** no maneja ningún otro tipo de archivo binario.

La siguiente sección describe cómo:

- Utilice el plugin **annobin**
- Utilice el programa **annocheck**
- Eliminar las notas redundantes de **annobin**

17.1. USO DEL PLUGIN ANNOBIN

La siguiente sección describe cómo:

- Activar el plugin **annobin**
- Pasar opciones al plugin **annobin**

17.1.1. Activación del plugin annobin

La siguiente sección describe cómo activar el plugin **annobin** a través de **gcc** y a través de **clang**.

Procedimiento

- Para activar el plugin **annobin** con **gcc**, utilice:

```
$ gcc -fplugin=annobin
```

  - Si **gcc** no encuentra el plugin **annobin**, utilice:

```
$ gcc -iplugindir=/path/to/directory/containing/annobin/
```

    Sustituya */path/to/directory/containing/annobin/* por la ruta absoluta del directorio que contiene **annobin**.

- Para encontrar el directorio que contiene el plugin **annobin**, utilice:

```
$ gcc --print-file-name=plugin
```

- Para activar el plugin **annobin** con **clang**, utilice:

```
$ clang -fplugin=/path/to/directory/containing/annobin/
```

Sustituya */path/to/directory/containing/annobin/* por la ruta absoluta del directorio que contiene **annobin**.

17.1.2. Pasar opciones al plugin annobin

La siguiente sección describe cómo pasar opciones al plugin **annobin** a través de **gcc** y a través de **clang**.

Procedimiento

- Para pasar opciones al plugin **annobin** con **gcc**, utilice:

```
$ gcc -fplugin=annobin -fplugin-arg-annobin-option file-name
```

Sustituya *option* por los argumentos de la línea de comandos **annobin** y sustituya *file-name* por el nombre del archivo.

Ejemplo

- Para mostrar detalles adicionales sobre lo que **annobin** está haciendo, utilice:

```
$ gcc -fplugin=annobin -fplugin-arg-annobin-verbose file-name
```

Sustituya *file-name* por el nombre del archivo.

- Para pasar opciones al plugin **annobin** con **clang**, utilice:

```
$ clang -fplugin=/path/to/directory/containing/annobin/ -Xclang -plugin-arg-annobin -Xclang option file-name
```

Sustituya *option* por los argumentos de la línea de comandos **annobin** y sustituya */path/to/directory/containing/annobin/* por la ruta absoluta del directorio que contiene **annobin**.

Ejemplo

- Para mostrar detalles adicionales sobre lo que **annobin** está haciendo, utilice:

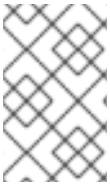
```
$ clang -fplugin=/usr/lib64/clang/10/lib/annobin.so -Xclang -plugin-arg-annobin -Xclang verbose file-name
```

Sustituya *file-name* por el nombre del archivo.

17.2. USO DEL PROGRAMA ANNOCHECK

La siguiente sección describe cómo utilizar **annocheck** para examinar:

- Archivos
- Directorios
- Paquetes RPM
- **annocheck** herramientas adicionales



NOTA

annocheck explora recursivamente directorios, archivos y paquetes RPM en busca de archivos objeto ELF. Los archivos tienen que estar en el formato ELF. **annocheck** no maneja ningún otro tipo de archivo binario.

17.2.1. Uso de annocheck para examinar archivos

La siguiente sección describe cómo examinar los archivos ELF utilizando **annocheck**.

Procedimiento

- Para examinar un archivo, utilice:

```
$ annocheck file-name
```

Sustituya *file-name* por el nombre de un archivo.

NOTA

Los archivos deben estar en formato ELF. **annocheck** no maneja ningún otro tipo de archivo binario. **annocheck** procesa bibliotecas estáticas que contienen archivos de objetos ELF.

Información adicional

- Para obtener más información sobre **annocheck** y las posibles opciones de la línea de comandos, consulte la página de manual **annocheck**.

17.2.2. Uso de annocheck para examinar directorios

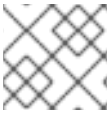
La siguiente sección describe cómo examinar los archivos ELF en un directorio utilizando **annocheck**.

Procedimiento

- Para escanear un directorio, utilice:

```
$ annocheck directory-name
```

Sustituya *directory-name* por el nombre de un directorio. **annocheck** examina automáticamente el contenido del directorio, sus subdirectorios y los archivos y paquetes RPM que se encuentren en él.



NOTA

annocheck sólo busca archivos ELF. Otros tipos de archivos son ignorados.

Información adicional

- Para obtener más información sobre **annocheck** y las posibles opciones de la línea de comandos, consulte la página de manual **annocheck**.

17.2.3. Uso de annocheck para examinar los paquetes RPM

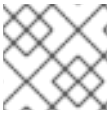
La siguiente sección describe cómo examinar los archivos ELF en un paquete RPM utilizando **annocheck**.

Procedimiento

- Para escanear un paquete RPM, utilice:

```
$ annocheck rpm-package-name
```

Sustituya *rpm-package-name* por el nombre de un paquete RPM. **annocheck** explora recursivamente todos los archivos ELF dentro del paquete RPM.



NOTA

annocheck sólo busca archivos ELF. Otros tipos de archivos son ignorados.

- Para escanear un paquete RPM con la información de depuración proporcionada RPM, utilice:

```
$ annocheck rpm-package-name --debug-rpm debuginfo-rpm
```

Sustituya *rpm-package-name* por el nombre de un paquete RPM, y *debuginfo-rpm* por el nombre de un RPM de información de depuración asociado al RPM binario.

Información adicional

- Para obtener más información sobre **annocheck** y las posibles opciones de la línea de comandos, consulte la página de manual **annocheck**.

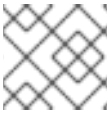
17.2.4. Uso de las herramientas adicionales de annocheck

annocheck incluye múltiples herramientas para examinar archivos binarios. Puede activar estas herramientas con las opciones de la línea de comandos.

La siguiente sección describe cómo habilitar el:

- **built-by** herramienta
- **notes** herramienta
- **section-size** herramienta

Puedes habilitar varias herramientas al mismo tiempo.



NOTA

El comprobador de endurecimiento está activado por defecto.

17.2.4.1. Activación de la herramienta built-by

Puede utilizar la herramienta **annocheck built-by** para encontrar el nombre del compilador que construyó el archivo binario.

Procedimiento

- Para activar la herramienta **built-by**, utilice:

```
$ annocheck --enable-built-by
```

Información adicional

- Para obtener más información sobre la herramienta **built-by**, consulte la opción de línea de comandos **--help**.

17.2.4.2. Activación de la herramienta notes

Puede utilizar la herramienta **annocheck notes** para mostrar las notas almacenadas en un archivo binario creado por el plugin **annobin**.

Procedimiento

- Para activar la herramienta **notes**, utilice:

```
$ annocheck --enable-notes
```

  Las notas se muestran en una secuencia ordenada por el rango de direcciones.

Información adicional

- Para obtener más información sobre la herramienta **notes**, consulte la opción de línea de comandos **--help**.

17.2.4.3. Activación de la herramienta section-size

Puede utilizar la herramienta **annocheck section-size** para mostrar el tamaño de las secciones nombradas.

Procedimiento

- Para activar la herramienta **section-size**, utilice:

```
$ annocheck --section-size=name
```

  Sustituya *name* por el nombre de la sección nombrada. La salida se restringe a secciones específicas. Al final se produce un resultado acumulativo.

Información adicional

- Para obtener más información sobre la herramienta **section-size**, consulte la opción de línea de comandos **--help**.

17.2.4.4. Conceptos básicos del verificador de endurecimiento

El comprobador de endurecimiento está activado por defecto. Puede desactivar el verificador de endurecimiento con la opción de línea de comandos **--disable-hardened**.

17.2.4.4.1. Opciones de comprobación del endurecimiento

El programa **annocheck** comprueba las siguientes opciones:

- Lazy binding se desactiva con la opción del enlazador **-z now**.
- El programa no tiene una pila en una región de memoria ejecutable.
- Las reubicaciones de la tabla GOT se establecen como de sólo lectura.
- Ningún segmento de programa tiene establecidos los tres bits de permiso de lectura, escritura y ejecución.
- No hay reubicaciones contra el código ejecutable.
- La información de la ruta de ejecución para localizar las bibliotecas compartidas en tiempo de ejecución incluye sólo los directorios con raíz en /usr.
- El programa fue compilado con **annobin** notas habilitadas.
- El programa fue compilado con la opción **-fstack-protector-strong** activada.
- El programa fue compilado con **-D_FORTIFY_SOURCE=2**.
- El programa fue compilado con **-D_GLIBCXX_ASSERTIONS**.
- El programa fue compilado con **-fexceptions** activado.
- El programa fue compilado con **-fstack-clash-protection** activado.
- El programa fue compilado en **-O2** o superior.
- El programa no tiene ninguna reubicación en una escritura.
- Los ejecutables dinámicos tienen un segmento dinámico.
- Las bibliotecas compartidas se compilaron con **-fPIC** o **-fPIE**.
- Los ejecutables dinámicos se compilaron con **-fPIE** y se enlazaron con **-pie**.
- Si está disponible, se utilizó la opción **-fcf-protection=full**.
- Si está disponible, se utilizó la opción **-mbranch-protection**.
- Si está disponible, se utilizó la opción **-mstackrealign**.

17.2.4.4.2. Desactivar el comprobador de endurecimiento

En la siguiente sección se describe cómo desactivar el comprobador de seguridad.

Procedimiento

- Para escanear las notas de un archivo sin el comprobador de endurecimiento, utilice:

```
$ annocheck --enable-notes --disable-hardened file-name
```

  Sustituya *file-name* por el nombre de un archivo.

17.3. ELIMINACIÓN DE NOTAS DE ANNOBIN REDUNDANTES

El uso de **annobin** aumenta el tamaño de los binarios. Para reducir el tamaño de los binarios compilados con **annobin** puede eliminar las notas redundantes de **annobin**. Para eliminar las notas redundantes de **annobin** utilice el programa **objcopy**, que forma parte del paquete **binutils**.

Procedimiento

- Para eliminar las notas redundantes de **annobin**, utilice:

```
$ objcopy --merge-notes file-name
```

  Sustituya *file-name* por el nombre del archivo.

PARTE IV. TEMAS COMPLEMENTARIOS

CAPÍTULO 18. CAMBIOS QUE ROMPEN LA COMPATIBILIDAD EN LOS COMPILADORES Y HERRAMIENTAS DE DESARROLLO

librtkaio eliminado

Con esta actualización, se ha eliminado la biblioteca **librtkaio**. Esta biblioteca proporcionaba un acceso de E/S asíncrono de alto rendimiento en tiempo real para algunos archivos, que se basaba en el soporte de E/S asíncrona del núcleo de Linux (KAIO).

Como resultado de la eliminación:

- Las aplicaciones que utilizan el método **LD_PRELOAD** para cargar **librtkaio** muestran una advertencia sobre una biblioteca que falta, cargan la biblioteca **librt** en su lugar y se ejecutan correctamente.
- Las aplicaciones que utilizan el método **LD_LIBRARY_PATH** para cargar **librtkaio** cargan la biblioteca **librt** en su lugar y se ejecutan correctamente, sin ninguna advertencia.
- Las aplicaciones que utilizan la llamada al sistema **dlopen()** para acceder a **librtkaio** cargan directamente la biblioteca **librt**.

Los usuarios de **librtkaio** tienen las siguientes opciones:

- Utilizar el mecanismo de reserva descrito anteriormente, sin ningún cambio en sus aplicaciones.
- Cambiar el código de sus aplicaciones para utilizar la biblioteca **librt**, que ofrece una API compatible con POSIX.
- Cambiar el código de sus aplicaciones para utilizar la biblioteca **libaio**, que ofrece una API compatible.

Tanto **librt** como **libaio** pueden ofrecer características y prestaciones comparables en condiciones específicas.

Tenga en cuenta que el paquete **libaio** tiene un nivel de compatibilidad de Red Hat de 2, mientras que **librtk** y el eliminado **librtkaio** tienen un nivel 1.

Para más detalles, consulte https://fedoraproject.org/wiki/Changes/GLIBC223_librtkaio_removal

Eliminación de las interfaces RPC y NIS de Sun glibc

La biblioteca **glibc** ya no proporciona las interfaces Sun RPC y NIS para las nuevas aplicaciones. Estas interfaces están ahora disponibles sólo para ejecutar aplicaciones heredadas. Los desarrolladores deben cambiar sus aplicaciones para utilizar la biblioteca **libtirpc** en lugar de Sun RPC y **libnsl2** en lugar de NIS. Las aplicaciones pueden beneficiarse del soporte de IPv6 en las bibliotecas de reemplazo.

Se han eliminado las bibliotecas nosegneg para Xen de 32 bits

Anteriormente, los paquetes **glibc** i686 contenían una compilación alternativa **glibc**, que evitaba el uso del registro del segmento descriptor de hilos con desplazamientos negativos (**nosegneg**). Esta compilación alternativa sólo se utilizaba en la versión de 32 bits del hipervisor del proyecto Xen sin soporte de virtualización por hardware, como una optimización para reducir el coste de la paravirtualización completa. Estas construcciones alternativas ya no se utilizan y han sido eliminadas.

make nuevo operador != provoca una interpretación diferente de cierta sintaxis de makefile existente

El operador de asignación del shell **!=** se ha añadido a GNU **make** como alternativa a la función **$(shell ...)** para aumentar la compatibilidad con los makefiles de BSD. Como consecuencia, las variables con nombres que terminan en signo de exclamación e inmediatamente seguidas de una asignación, como **variable!=value**, se interpretan ahora como una asignación del shell. Para restaurar el comportamiento anterior, añada un espacio después del signo de exclamación, como **variable! =value**.

Para más detalles y diferencias entre el operador y la función, consulte el manual de GNU **make**.

Se ha eliminado el soporte de la biblioteca Valgrind para la depuración de MPI

Se ha eliminado la biblioteca envolvente **libmpiwrap.so** para **Valgrind** proporcionada por el paquete **valgrind-openmpi**. Esta biblioteca permitía a **Valgrind** depurar programas que utilizaban la interfaz de paso de mensajes (MPI). Esta biblioteca era específica de la versión de implementación de Open MPI en versiones anteriores de Red Hat Enterprise Linux.

Se anima a los usuarios de **libmpiwrap.so** a que construyan su propia versión a partir de fuentes upstream específicas para su implementación y versión de MPI. Suministre estas bibliotecas personalizadas a **Valgrind** utilizando la técnica **LD_PRELOAD**.

Se han eliminado las cabeceras de desarrollo y las bibliotecas estáticas de **valgrind-devel**

Anteriormente, el subpaquete **valgrind-devel** solía incluir archivos de desarrollo para desarrollar herramientas valgrind personalizadas. Esta actualización elimina estos archivos porque no tienen una API garantizada, tienen que ser enlazados estáticamente y no están soportados. El paquete **valgrind-devel** aún contiene los archivos de desarrollo para programas compatibles con valgrind y archivos de cabecera como **valgrind.h**, **callgrind.h**, **drd.h**, **helgrind.h**, y **memcheck.h**, que son estables y están bien soportados.

CAPÍTULO 19. OPCIONES PARA EJECUTAR UNA APLICACIÓN RHEL 6 O 7 EN RHEL 8

Para ejecutar una aplicación de Red Hat Enterprise Linux 6 o 7 en Red Hat Enterprise Linux 8, hay un espectro de opciones disponibles. Un administrador de sistemas necesita una guía detallada del desarrollador de la aplicación. La siguiente lista resume las opciones, consideraciones y recursos proporcionados por Red Hat.

Ejecutar la aplicación en una máquina virtual con un sistema operativo invitado de la versión RHEL correspondiente

El coste de los recursos es elevado para esta opción, pero el entorno se ajusta a los requisitos de la aplicación y este enfoque no requiere muchas consideraciones adicionales. Esta es la opción recomendada actualmente.

Ejecutar la aplicación en un contenedor basado en la versión respectiva de RHEL

Los costes de recursos son menores que en los casos anteriores, mientras que los requisitos de configuración son más estrictos. Para más detalles sobre la relación entre los hosts de contenedores y los espacios de usuario invitados, consulte la Matriz de compatibilidad de contenedores de Red Hat Enterprise Linux.

Ejecutar la aplicación de forma nativa en RHEL 8

Esta opción ofrece el menor coste de recursos, pero también los requisitos más estrictos. El desarrollador de la aplicación debe determinar una configuración correcta del sistema RHEL 8. Los siguientes recursos pueden ayudar al desarrollador en esta tarea:

- Red Hat Enterprise Linux 8: Guía de compatibilidad de aplicaciones
- Red Hat Enterprise Linux 7: Guía de compatibilidad de aplicaciones
- Notas de la versión de Red Hat Enterprise Linux 8.0
- Consideraciones para adoptar RHEL 8

Tenga en cuenta que esta lista no es un conjunto completo de recursos necesarios para determinar la compatibilidad de las aplicaciones. Son sólo puntos de partida con listas de cambios incompatibles conocidos y políticas de Red Hat relacionadas con la compatibilidad.

Además, el artículo *What is Kernel Application Binary Interface (kABI)?* Artículo de soporte centrado en el conocimiento contiene información relevante para el núcleo y la compatibilidad.