Red Hat Enterprise Linux 8

Desenvolvendo aplicações C e C++ no RHEL 8

Configurando uma estação de trabalho de desenvolvedores e desenvolvendo e depurando aplicações C e C++ no Red Hat Enterprise Linux 8

Last Updated: 2021-03-07

Red Hat Enterprise Linux 8 Desenvolvendo aplicações C e C++ no RHEL 8

Configurando uma estação de trabalho de desenvolvedores e desenvolvendo e depurando aplicações C e C++ no 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.

Nota Legal

Resumo

Este documento descreve as diferentes características e utilidades que fazem do Red Hat Enterprise Linux 8 uma plataforma empresarial ideal para o desenvolvimento de aplicativos.

Índice

TORNANDO O CÓDIGO ABERTO MAIS INCLUSIVO

A Red Hat tem o compromisso de substituir a linguagem problemática em nosso código, documentação e propriedades da web. Estamos começando com estes quatro termos: master, slave, blacklist e whitelist. Por causa da enormidade deste esforço, estas mudanças serão implementadas gradualmente ao longo de vários lançamentos futuros. Para mais detalhes, veja a mensagem de nosso CTO Chris Wright.

FORNECENDO FEEDBACK SOBRE A DOCUMENTAÇÃO DA RED HAT

Agradecemos sua contribuição em nossa documentação. Por favor, diga-nos como podemos melhorá-la. Para fazer isso:

- Para comentários simples sobre passagens específicas:
  1. Certifique-se de que você está visualizando a documentação no formato *Multi-page HTML*. Além disso, certifique-se de ver o botão **Feedback** no canto superior direito do documento.
  2. Use o cursor do mouse para destacar a parte do texto que você deseja comentar.
  3. Clique no pop-up **Add Feedback** que aparece abaixo do texto destacado.
  4. Siga as instruções apresentadas.
- Para enviar comentários mais complexos, crie um bilhete Bugzilla:
  1. Ir para o site da Bugzilla.
  2. Como Componente, use **Documentation**.
  3. Preencha o campo **Description** com sua sugestão de melhoria. Inclua um link para a(s) parte(s) relevante(s) da documentação.
  4. Clique em **Submit Bug**.

CAPÍTULO 1. CRIAÇÃO DE UMA ESTAÇÃO DE TRABALHO DE DESENVOLVIMENTO

O Red Hat Enterprise Linux 8 suporta o desenvolvimento de aplicações personalizadas. Para permitir que os desenvolvedores façam isso, o sistema deve ser configurado com as ferramentas e utilitários necessários. Este capítulo lista os casos de uso mais comuns para desenvolvimento e os itens a serem instalados.

1.1. PRÉ-REQUISITOS

- O sistema deve ser instalado, incluindo um ambiente gráfico, e subscrito.

1.2. HABILITAÇÃO DE DEPURAÇÃO E REPOSITÓRIOS DE FONTE

Uma instalação padrão do Red Hat Enterprise Linux não habilita os repositórios de depuração e fonte. Estes repositórios contêm informações necessárias para depurar os componentes do sistema e medir seu desempenho.

Procedimento

- Habilitar os canais do pacote de informações de origem e de depuração:

```
# 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
```

A parte **$(uname -i)** é automaticamente substituída por um valor correspondente para a arquitetura de seu sistema:

Nome da arquitetura	Valor
Intel e AMD de 64 bits	x86_64
ARM de 64 bits	aarch64
IBM POWER	ppc64le
IBM Z	s390x

1.3. CONFIGURAÇÃO PARA GERENCIAR VERSÕES DE APLICAÇÃO

O controle efetivo da versão é essencial para todos os projetos multi-desenvolvedores. O Red Hat Enterprise Linux é fornecido com Git, um sistema de controle de versão distribuído.

Procedimento

1. Instale o **git** pacote:

```
# yum instalar git
```

2. Opcional: Defina o nome completo e o endereço de e-mail associados ao seu compromisso Git:

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

Substitua *Full Name* e *email@example.com* por seu nome real e endereço de e-mail.

3. Opcional: Para alterar o editor de texto padrão iniciado por Git, defina o valor da opção de configuração **core.editor**:

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

Substituir *command* pelo comando a ser usado para iniciar o editor de texto selecionado.

Recursos adicionais

- Páginas de manual Linux para Git e tutoriais:

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

Note que muitos comandos Git têm suas próprias páginas de manual. Como exemplo, veja *git-commit(1)*.

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

1.4. CRIAÇÃO PARA DESENVOLVER APLICAÇÕES USANDO C E C++

O Red Hat Enterprise Linux inclui ferramentas para a criação de aplicações C e C.

Pré-requisitos

- Os repositórios de depuração e fonte devem ser habilitados.

Procedimento

1. Instale o **Development Tools** grupo de pacotes incluindo a GNU Compiler Collection (GCC), GNU Debugger (GDB), e outras ferramentas de desenvolvimento:

```
# yum group install "Ferramentas de Desenvolvimento"
```

2. Instalar a cadeia de ferramentas baseada na LLVM incluindo o compilador **clang** e o depurador **lldb**:

```
# yum instalar llvm-toolset
```

3. Opcional: Para dependências Fortran, instale o compilador Fortran GNU:

```
# yum instalar gcc-gfortran
```

1.5. INSTALAÇÃO PARA APLICAÇÕES DE DEPURAÇÃO

O Red Hat Enterprise Linux oferece múltiplas ferramentas de depuração e instrumentação para analisar e solucionar problemas de comportamento de aplicações internas.

Pré-requisitos

- Os repositórios de depuração e fonte devem ser habilitados.

Procedimento

1. Instalar as ferramentas úteis para a depuração:

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

2. Instale o **yum-utils** a fim de utilizar a ferramenta **debuginfo-install**:

```
# yum instalar yum-utils
```

3. Execute um roteiro auxiliar do SystemTap para configurar o ambiente.

```
# stap-prep-prep
```

Note que **stap-prep** instala pacotes relevantes para o kernel atualmente *running*, que podem não ser os mesmos que o(s) kernel(s) realmente instalado(s). Para garantir que o **stap-prep** instala o correto **kernel-debuginfo** e **kernel-headers** verifique novamente a versão atual do kernel usando o comando **uname -r** e reinicie seu sistema, se necessário.

4. Certifique-se de que as políticas **SELinux** permitam que as aplicações relevantes funcionem não apenas normalmente, mas também nas situações de depuração. Para mais informações, consulte Utilizando o SELinux.

Recursos adicionais

- Capítulo 7, *Habilitando a depuração com informações de depuração*

1.6. CONFIGURAÇÃO PARA MEDIR O DESEMPENHO DAS APLICAÇÕES

O Red Hat Enterprise Linux inclui várias aplicações que podem ajudar um desenvolvedor a identificar as causas da perda de desempenho da aplicação.

Pré-requisitos

- Os repositórios de depuração e fonte devem ser habilitados.

Procedimento

1. Instalar as ferramentas para a medição de desempenho:

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

2. Execute um roteiro auxiliar do SystemTap para configurar o ambiente.

```
# stap-prep-prep
```

Note que **stap-prep** instala pacotes relevantes para o kernel atualmente *running*, que podem não ser os mesmos que o(s) kernel(s) realmente instalado(s). Para garantir que o **stap-prep** instala o correto **kernel-debuginfo** e **kernel-headers** verifique novamente a versão atual do kernel usando o comando **uname -r** e reinicie seu sistema, se necessário.

3. Habilitar e iniciar o serviço de coletor do Co-Piloto de Desempenho (PCP):

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

PARTE I. CRIANDO APLICAÇÕES C OU C

A Red Hat oferece múltiplas ferramentas para a criação de aplicações usando as linguagens C e C. Esta parte do livro lista algumas das tarefas de desenvolvimento mais comuns.

CAPÍTULO 2. CÓDIGO DE CONSTRUÇÃO COM GCC

Este capítulo descreve situações em que o código fonte deve ser transformado em código executável.

2.1. RELAÇÃO ENTRE AS FORMAS DE CÓDIGO

Pré-requisitos

- Entendendo os conceitos de compilação e vinculação

Possíveis formas de código

Os idiomas C e C++ têm três formas de código:

- **Source code** escrito na língua C ou C }, presente como arquivos de texto simples.
  Os arquivos normalmente usam extensões como **.c**, **.cc**, **.cpp**, **.h**, **.hpp**, **.i**, **.inc**. Para uma lista completa das extensões suportadas e sua interpretação, consulte as páginas do manual do gcc:

  ```
  $ homem gcc
  ```

- **Object code**, criado por *compiling* o código fonte com um *compiler*. Esta é uma forma intermediária.
  Os arquivos de código objeto utilizam a extensão **.o**.

- **Executable code**, criado por *linking* código objeto com um *linker*.
  Os arquivos executáveis de aplicativos Linux não utilizam nenhuma extensão de nome de arquivo. Os arquivos executáveis de objetos compartilhados (biblioteca) usam a extensão de nome de arquivo **.so**.

NOTA

Também existem arquivos de biblioteca para ligação estática. Esta é uma variante do código do objeto que usa a extensão do nome do arquivo **.a**. A vinculação estática não é recomendada. Veja Seção 3.2, “Ligação estática e dinâmica” .

Manuseio de formulários de código no GCC

A produção de código executável a partir do código fonte é realizada em duas etapas, que requerem aplicações ou ferramentas diferentes. O GCC pode ser usado como um driver inteligente tanto para compiladores quanto para linkers. Isto permite utilizar um único comando **gcc** para qualquer uma das ações necessárias (compilação e link). GCC seleciona automaticamente as ações e sua seqüência:

1. Os arquivos fonte são compilados para objetos de arquivos.
2. Os arquivos objetos e bibliotecas estão ligados (incluindo as fontes compiladas anteriormente).

É possível executar o GCC para que ele execute somente a etapa 1, somente a etapa 2, ou ambas as etapas 1 e 2. Isto é determinado pelos tipos de entradas e tipo de saída(s) solicitada(s).

Como os projetos maiores requerem um sistema de construção que normalmente executa GCC separadamente para cada ação, é melhor considerar sempre a compilação e a ligação como duas ações distintas, mesmo que GCC possa executar ambas de uma só vez.

Recursos adicionais

- Seção 2.2, "Compilação de arquivos fonte para código objeto"
- Seção 2.6, "Código de ligação para criar arquivos executáveis"

2.2. COMPILAÇÃO DE ARQUIVOS FONTE PARA CÓDIGO OBJETO

Para criar arquivos de código objeto a partir de arquivos fonte e não um arquivo executável imediatamente, o GCC deve ser instruído a criar somente arquivos de código objeto como sua saída. Esta ação representa a operação básica do processo de construção para projetos maiores.

Pré-requisitos

- Arquivo(s) de código fonte C ou C
- GCC instalado no sistema

Procedimento

1. Mudança para o diretório contendo o(s) arquivo(s) de código fonte.
2. Execute **gcc** com a opção **-c**:

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

Os arquivos objeto são criados, com seus nomes refletindo os arquivos de código fonte originais: **source.c** resulta em **source.o**.

NOTA

Com o código fonte C, substitua o comando **gcc** por **g** para o manuseio conveniente das dependências da Biblioteca Padrão C.

Recursos adicionais

- Seção 2.5, "Opções para endurecimento de código com GCC"
- Seção 2.4, "Otimização do código com GCC"
- Seção 2.7, "Exemplo: Construindo um programa C com GCC"

2.3. HABILITANDO A DEPURAÇÃO DE APLICAÇÕES C E C++ COM GCC

Como a informação de depuração é grande, ela não é incluída em arquivos executáveis por padrão. Para permitir a depuração de suas aplicações C e C++ com ela, você deve instruir explicitamente o compilador a criá-la.

Para permitir a criação de informações de depuração com **GCC** ao compilar e vincular código, use a opção **-g**:

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

- Otimizações realizadas pelo compilador e pelo linker podem resultar em código executável que é difícil de relacionar com o código fonte original: variáveis podem ser otimizadas, loops desenrolados, operações fundidas com as operações ao redor, etc. Isto afeta negativamente a

depuração. Para melhorar a experiência de depuração, considere definir a otimização com a opção **-Og**. Entretanto, mudar o nível de otimização altera o código executável e pode mudar o comportamento real, incluindo a remoção de alguns bugs.

- Para incluir também definições macro nas informações de depuração, use a opção **-g3** ao invés de **-g**.
- A opção **-fcompare-debug** GCC testa o código compilado pelo GCC com informações de debug e sem informações de debug. O teste passa se os dois arquivos binários resultantes forem idênticos. Este teste garante que o código executável não seja afetado por nenhuma opção de depuração, o que garante ainda mais que não haja bugs ocultos no código de depuração. Note que o uso da opção **-fcompare-debug** aumenta significativamente o tempo de compilação. Consulte a página do manual GCC para obter detalhes sobre esta opção.

Recursos adicionais

- Capítulo 7, *Habilitando a depuração com informações de depuração*
- Usando a Coleção de Compiladores GNU (GCC)
- Depuração com GDB
- A página do manual do GCC:

```
$ homem gcc
```

2.4. OTIMIZAÇÃO DO CÓDIGO COM GCC

Um único programa pode ser transformado em mais de uma seqüência de instruções da máquina. Você pode alcançar um resultado mais otimizado se alocar mais recursos para analisar o código durante a compilação.

Com GCC, você pode definir o nível de otimização usando o **-O*level*** opção. Esta opção aceita um conjunto de valores no lugar do *level*.

Nível	Descrição
0	Otimizar para velocidade de compilação - sem otimização de código (padrão).
1, **2**, **3**	Otimizar para aumentar a velocidade de execução do código (quanto maior o número, maior a velocidade).
s	Otimizar para o tamanho do arquivo.
fast	O mesmo que um nível estabelecido em **3**, mais **fast** desconsidera o cumprimento de normas rigorosas para permitir otimizações adicionais.
g	Otimizar para a experiência de depuração.

Para a construção do release, use a opção de otimização **-O2**.

Durante o desenvolvimento, a opção **-Og** é útil para a depuração do programa ou biblioteca em algumas situações. Como alguns bugs se manifestam apenas com certos níveis de otimização, teste o programa ou biblioteca com o nível de otimização do lançamento.

A GCC oferece um grande número de opções para permitir otimizações individuais. Para mais informações, consulte os seguintes Recursos adicionais.

Recursos adicionais

- Usando a Coleção de Compiladores GNU
- Página do manual Linux para GCC:

```
$ homem gcc
```

2.5. OPÇÕES PARA ENDURECIMENTO DE CÓDIGO COM GCC

Quando o compilador transforma o código fonte em código objeto, ele pode adicionar várias verificações para evitar situações comumente exploradas e aumentar a segurança. Escolher o conjunto certo de opções de compilador pode ajudar a produzir programas e bibliotecas mais seguros, sem ter que alterar o código-fonte.

Opções de versão de lançamento

A seguinte lista de opções é o mínimo recomendado para desenvolvedores que tenham como alvo o Red Hat Enterprise Linux:

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

- Para os programas, adicione as opções de execução independente de cargos **-fPIE** e **-pie**.
- Para bibliotecas ligadas dinamicamente, a opção obrigatória **-fPIC** (Position Independent Code) aumenta indiretamente a segurança.

Opções de desenvolvimento

Utilize as seguintes opções para detectar falhas de segurança durante o desenvolvimento. Use estas opções em conjunto com as opções para a versão de lançamento:

```
$ gcc ... -Walloc-zero -Walloca-larga que -Wextra -Wformat-security -Wvlalarga que -Wvlalarga que -
Walloca-larga que ...
```

Recursos adicionais

- Guia de Codificação Defensiva
- Detecção de erros de memória usando GCC

2.6. CÓDIGO DE LIGAÇÃO PARA CRIAR ARQUIVOS EXECUTÁVEIS

A ligação é o passo final na construção de uma aplicação C ou C. Linking combina todos os arquivos objeto e bibliotecas em um arquivo executável.

Pré-requisitos

- Um ou mais arquivo(s) objeto(s)
- O GCC deve ser instalado no sistema

Procedimento

1. Mudança para o diretório contendo o(s) arquivo(s) de código objeto.
2. Rodar **gcc**:

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

Um arquivo executável chamado ***executable-file*** é criado a partir dos arquivos objeto fornecidos e das bibliotecas.

Para ligar bibliotecas adicionais, adicione as opções necessárias após a lista de arquivos objeto. Para mais informações, veja Capítulo 3, *Usando Bibliotecas com GCC*.

NOTA

Com o código fonte C, substitua o comando **gcc** por **g** para o manuseio conveniente das dependências da Biblioteca Padrão C.

Recursos adicionais

- Seção 2.7, "Exemplo: Construindo um programa C com GCC"
- Seção 3.2, "Ligação estática e dinâmica"

2.7. EXEMPLO: CONSTRUINDO UM PROGRAMA C COM GCC

Este exemplo mostra os passos exatos para construir um simples programa de amostra C.

Pré-requisitos

- Você deve entender como usar o GCC.

Procedimento

1. Crie um diretório **hello-c** e mude para ele:

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

2. Criar arquivo **hello.c** com o seguinte conteúdo:

```
#include <stdio.h>

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

3. Compilar o código com o GCC:

```
$ gcc -c olá.c
```

O arquivo objeto **hello.o** é criado.

4. Vincule um arquivo executável **helloworld** a partir do arquivo objeto:

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

5. Execute o arquivo executável resultante:

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

Recursos adicionais

- Seção 5.2, “Exemplo: Construindo um programa C usando um Makefile”

2.8. EXEMPLO: CONSTRUINDO UM PROGRAMA C COM GCC

Este exemplo mostra os passos exatos para construir uma amostra do programa C mínimo.

Pré-requisitos

- Você deve entender a diferença entre **gcc** e **g** .

Procedimento

1. Crie um diretório **hello-cpp** e mude para ele:

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

2. Criar arquivo **hello.cpp** com o seguinte conteúdo:

```
#include <iostream>

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

3. Compilar o código com **g** :

```
$ g -c hello.cpp
```

O arquivo objeto **hello.o** é criado.

4. Vincule um arquivo executável **helloworld** a partir do arquivo objeto:

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

5. Execute o arquivo executável resultante:

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

CAPÍTULO 3. USANDO BIBLIOTECAS COM GCC

Este capítulo descreve o uso de bibliotecas em código.

3.1. CONVENÇÕES DE NOMENCLATURA DA BIBLIOTECA

Uma convenção especial de nomes de arquivos é usada para bibliotecas: uma biblioteca conhecida como **foo** deve existir como arquivo **lib*foo*.so** ou **lib*foo*.a**. Esta convenção é automaticamente entendida pelas opções de entrada de ligação do GCC, mas não pelas opções de saída:

- Ao criar um link contra a biblioteca, a biblioteca pode ser especificada apenas por seu nome **foo** com a opção **-l** como **-l*foo***:

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

- Ao criar a biblioteca, o nome completo do arquivo **lib*foo*.so** ou **lib*foo*.a** deve ser especificado.

Recursos adicionais

- Seção 4.2, “O mecanismo soname”

3.2. LIGAÇÃO ESTÁTICA E DINÂMICA

Os desenvolvedores têm a opção de usar links estáticos ou dinâmicos ao construir aplicações com linguagens totalmente compiladas. Esta seção lista as diferenças, particularmente no contexto do uso das linguagens C e C++ no Red Hat Enterprise Linux. Para resumir, a Red Hat desestimula o uso de links estáticos em aplicações para o Red Hat Enterprise Linux.

Comparação de ligação estática e dinâmica

A ligação estática torna as bibliotecas parte do arquivo executável resultante. A vinculação dinâmica mantém estas bibliotecas como arquivos separados.

A ligação dinâmica e estática pode ser comparada de várias maneiras:

Uso de recursos

A ligação estática resulta em arquivos executáveis maiores, que contêm mais código. Este código adicional vindo de bibliotecas não pode ser compartilhado entre vários programas no sistema, aumentando o uso do sistema de arquivos e o uso de memória em tempo de execução. Múltiplos processos rodando o mesmo programa estaticamente vinculado ainda compartilharão o código. Por outro lado, as aplicações estáticas precisam de menos relocalizações de tempo de execução, levando a uma redução do tempo de inicialização, e requerem menos memória de tamanho de conjunto residente privado (RSS). O código gerado para a ligação estática pode ser mais eficiente do que para a ligação dinâmica devido à sobrecarga introduzida pelo código independente de posição (PIC).

Segurança

Bibliotecas dinamicamente vinculadas que proporcionam compatibilidade ABI podem ser atualizadas sem alterar os arquivos executáveis, dependendo destas bibliotecas. Isto é especialmente importante para as bibliotecas fornecidas pela Red Hat como parte do Red Hat Enterprise Linux, onde a Red Hat fornece atualizações de segurança. A ligação estática contra qualquer uma destas bibliotecas é fortemente desencorajada.

Compatibilidade

A ligação estática parece fornecer arquivos executáveis independentemente das versões das bibliotecas fornecidas pelo sistema operacional. Entretanto, a maioria das bibliotecas depende de outras bibliotecas. Com a ligação estática, esta dependência se torna inflexível e como resultado, tanto a compatibilidade para frente como para trás se perde. A ligação estática é garantida para funcionar somente no sistema onde o arquivo executável foi construído.

ATENÇÃO

As aplicações que ligam estaticamente bibliotecas da biblioteca C GNU (**glibc**) ainda requerem **glibc** para estar presentes no sistema como uma biblioteca dinâmica. Além disso, a variante de biblioteca dinâmica de **glibc** disponível no tempo de execução da aplicação deve ser um pouco idêntica àquela presente durante a ligação da aplicação. Como resultado, a ligação estática é garantida para funcionar somente no sistema onde o arquivo executável foi construído.

Cobertura de apoio

A maioria das bibliotecas estáticas fornecidas pela Red Hat estão no canal *CodeReady Linux Builder* e não são suportadas pela Red Hat.

Funcionalidade

Algumas bibliotecas, notadamente a Biblioteca C GNU (**glibc**), oferecem funcionalidade reduzida quando ligadas estaticamente.
Por exemplo, quando estaticamente vinculado, **glibc** não suporta threads e nenhuma forma de chamadas para a função **dlopen()** no mesmo programa.

Como resultado das desvantagens listadas, a ligação estática deve ser evitada a todo custo, particularmente para aplicações inteiras e para as bibliotecas de **glibc** e **libstdc** .

Estojos para ligação estática

A ligação estática pode ser uma escolha razoável em alguns casos, como por exemplo:

- Usando uma biblioteca que não está habilitada para a ligação dinâmica.
- A ligação totalmente estática pode ser necessária para a execução do código em um ambiente **chroot** vazio ou em um container. Entretanto, o link estático usando o pacote **glibc-static** não é suportado pela Red Hat.

Recursos adicionais

- Red Hat Enterprise Linux 8: Guia de Compatibilidade de Aplicações
- Descrição do repositório The CodeReady Linux Builder no *Package manifest*

3.3. USANDO UMA BIBLIOTECA COM GCC

Uma biblioteca é um pacote de código que pode ser reutilizado em seu programa. Uma biblioteca C ou C é composta de duas partes:

- O código da biblioteca

- Arquivos de cabeçalho

Compilação de código que utiliza uma biblioteca

Os arquivos de cabeçalho descrevem a interface da biblioteca: as funções e variáveis disponíveis na biblioteca. As informações dos arquivos de cabeçalho são necessárias para a compilação do código.

Tipicamente, os arquivos de cabeçalho de uma biblioteca serão colocados em um diretório diferente do código da sua aplicação. Para dizer ao GCC onde estão os arquivos de cabeçalho, use a opção **-I**:

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

Substitua *include_path* pelo caminho real para o diretório de arquivos de cabeçalho.

A opção **-I** pode ser usada várias vezes para adicionar vários diretórios com arquivos de cabeçalho. Ao procurar por um arquivo de cabeçalho, estes diretórios são pesquisados na ordem de aparecimento nas opções **-I**.

Código de ligação que utiliza uma biblioteca

Ao ligar o arquivo executável, tanto o código objeto de sua aplicação quanto o código binário da biblioteca devem estar disponíveis. O código para bibliotecas estáticas e dinâmicas está presente em diferentes formas:

- As bibliotecas estáticas estão disponíveis como arquivos. Elas contêm um grupo de arquivos objetos. O arquivo de arquivo tem uma extensão de nome de arquivo **.a**.
- Bibliotecas dinâmicas estão disponíveis como objetos compartilhados. Elas são uma forma de arquivo executável. Um objeto compartilhado tem uma extensão de nome de arquivo **.so**.

Para dizer ao GCC onde estão os arquivos ou arquivos objetos compartilhados de uma biblioteca, use a opção **-L**:

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

Substituir *library_path* pelo caminho real para o diretório da biblioteca.

A opção **-L** pode ser usada várias vezes para adicionar vários diretórios. Ao procurar uma biblioteca, estes diretórios são pesquisados na ordem de suas opções **-L**.

A ordem das opções é importante: O GCC não pode se conectar a uma biblioteca **foo** a menos que conheça o diretório com esta biblioteca. Portanto, use as opções **-L** para especificar os diretórios das bibliotecas antes de usar as opções **-l** para fazer um link contra bibliotecas.

Compilação e vinculação de código que utiliza uma biblioteca em uma única etapa

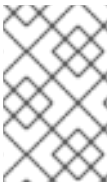
Quando a situação permitir que o código seja compilado e vinculado em um comando **gcc**, use as opções para ambas as situações mencionadas acima de uma só vez.

Recursos adicionais

- Usando a Coleção de Compiladores GNU (GCC)
- Usando a Coleção de Compiladores GNU (GCC)

3.4. USANDO UMA BIBLIOTECA ESTÁTICA COM GCC

As bibliotecas estáticas estão disponíveis como arquivos contendo arquivos objetos. Após a ligação, elas se tornam parte do arquivo executável resultante.



NOTA

A Red Hat desencoraja o uso de ligação estática por razões de segurança. Ver Seção 3.2, "Ligação estática e dinâmica". Use links estáticos somente quando necessário, especialmente contra bibliotecas fornecidas pela Red Hat.

Pré-requisitos

- O GCC deve ser instalado em seu sistema.
- Você deve compreender a ligação estática e dinâmica.
- Você tem um conjunto de arquivos fonte ou objeto formando um programa válido, exigindo alguma biblioteca estática **foo** e nenhuma outra biblioteca.
- A biblioteca **foo** está disponível como um arquivo **libfoo.a**, e nenhum arquivo **libfoo.so** é fornecido para links dinâmicos.

NOTA

A maioria das bibliotecas que fazem parte do Red Hat Enterprise Linux são suportadas apenas para links dinâmicos. Os passos abaixo só funcionam para as bibliotecas que são *not* habilitadas para links dinâmicos. Veja Seção 3.2, "Ligação estática e dinâmica" .

Procedimento

Para ligar um programa a partir de arquivos fonte e objeto, acrescentando uma biblioteca ligada estaticamente **foo**, que pode ser encontrada como um arquivo **libfoo.a**:

1. Mude para o diretório que contém seu código.
2. Compilar os arquivos fonte do programa com os cabeçalhos da biblioteca **foo**:

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

   Substitua *header_path* por um caminho para um diretório contendo os arquivos de cabeçalho para a biblioteca **foo**.

3. Ligue o programa com a biblioteca **foo**:

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

   Substitua *library_path* por um caminho para um diretório contendo o arquivo **libfoo.a**.

4. Para executar o programa mais tarde, simplesmente:

```
$ ./programa
```

CUIDADO

A opção **-static** GCC relacionada à ligação estática proíbe todas as ligações dinâmicas. Em vez disso, use as opções **-Wl,-Bstatic** e **-Wl,-Bdynamic** para controlar com mais precisão o comportamento do linker. Ver Seção 3.6, “Usando tanto bibliotecas estáticas como dinâmicas com GCC” .

3.5. USANDO UMA BIBLIOTECA DINÂMICA COM GCC

As bibliotecas dinâmicas estão disponíveis como arquivos executáveis independentes, necessários tanto no tempo de ligação como no tempo de execução. Elas permanecem independentes do arquivo executável de sua aplicação.

Pré-requisitos

- O GCC deve ser instalado no sistema.
- Um conjunto de arquivos fonte ou objeto formando um programa válido, exigindo alguma biblioteca dinâmica **foo** e nenhuma outra biblioteca.
- A biblioteca **foo** deve estar disponível como um arquivo *libfoo.so*.

Ligando um programa a uma biblioteca dinâmica

Para vincular um programa a uma biblioteca dinâmica **foo**:

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

Quando um programa é ligado contra uma biblioteca dinâmica, o programa resultante deve sempre carregar a biblioteca em tempo de execução. Há duas opções para a localização da biblioteca:

- Usando um valor **rpath** armazenado no próprio arquivo executável
- Usando a variável **LD_LIBRARY_PATH** em tempo de execução

Usando um valor armazenado no arquivo executável do rpath

O **rpath** é um valor especial salvo como parte de um arquivo executável quando ele está sendo vinculado. Mais tarde, quando o programa for carregado de seu arquivo executável, o linker de tempo de execução usará o valor **rpath** para localizar os arquivos da biblioteca.

Ao fazer o link com **GCC**, para armazenar o caminho *library_path* como **rpath**:

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

O caminho *library_path* deve apontar para um diretório contendo o arquivo *libfoo.so*.

CUIDADO

Não há espaço após a vírgula na opção **-Wl,-rpath=**!

Para executar o programa mais tarde:

```
$ ./programa
```

Usando a variável de ambiente LD_LIBRARY_PATH

Se não for encontrado **rpath** no arquivo executável do programa, o linker de tempo de execução usará a variável de ambiente **LD_LIBRARY_PATH**. O valor desta variável deve ser alterado para cada programa. Este valor deve representar o caminho onde os objetos da biblioteca compartilhada estão localizados.

Para executar o programa sem o conjunto **rpath**, com bibliotecas presentes no caminho *library_path*:

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

Deixar de fora o valor **rpath** oferece flexibilidade, mas requer a definição da variável **LD_LIBRARY_PATH** toda vez que o programa for executado.

Colocação da Biblioteca nos Diretórios Padrão

A configuração do linker de tempo de execução especifica uma série de diretórios como localização padrão dos arquivos dinâmicos da biblioteca. Para usar este comportamento padrão, copie sua biblioteca para o diretório apropriado.

Uma descrição completa do comportamento do linker dinâmico está fora do escopo deste documento. Para maiores informações, veja os seguintes recursos:

- Páginas de manual do Linux para o linker dinâmico:

  ```
  $ homem ld.so
  ```

- Conteúdo do arquivo de configuração /**etc/ld.so.conf**:

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

- Relatório das bibliotecas reconhecidas pelo linker dinâmico sem configuração adicional, que inclui os diretórios:

  ```
  $ ldconfig -v
  ```

3.6. USANDO TANTO BIBLIOTECAS ESTÁTICAS COMO DINÂMICAS COM GCC

Algumas vezes é necessário ligar algumas bibliotecas de forma estática e outras dinamicamente. Esta situação traz alguns desafios.

Pré-requisitos

- Entendendo a ligação estática e dinâmica

Introdução

gcc reconhece tanto as bibliotecas dinâmicas quanto as estáticas. Quando a **-l*foo*** se encontrar a opção **gcc** primeiro tentará localizar um objeto compartilhado (um arquivo **.so**) contendo uma versão dinamicamente ligada da biblioteca **foo**, e depois procurará o arquivo (**.a**) contendo uma versão estática da biblioteca. Assim, as seguintes situações podem resultar desta busca:

- Somente o objeto compartilhado é encontrado, e **gcc** se conecta dinamicamente com ele.

- Somente o arquivo é encontrado, e **gcc** links contra ele de forma estática.
- Tanto o objeto compartilhado quanto o arquivo são encontrados e, por padrão, **gcc** seleciona a ligação dinâmica contra o objeto compartilhado.
- Não é encontrado objeto compartilhado nem arquivo, e a ligação falha.

Devido a estas regras, a melhor maneira de selecionar a versão estática ou dinâmica de uma biblioteca para ligação é ter apenas essa versão encontrada por **gcc**. Isto pode ser controlado até certo ponto usando ou deixando de fora diretórios contendo as versões da biblioteca, ao especificar a **-L*path*** opções.

Além disso, como a ligação dinâmica é o padrão, a única situação em que a ligação deve ser explicitamente especificada é quando uma biblioteca com ambas as versões presentes deve ser ligada estaticamente. Há duas resoluções possíveis:

- Especificando as bibliotecas estáticas por caminho de arquivo em vez da opção **-l**
- Usando a opção **-Wl** para passar opções para o linker

Especificando as bibliotecas estáticas por arquivo

Normalmente, **gcc** é instruído a criar um link contra a biblioteca **foo** com o **-l*foo*** opção. Entretanto, é possível especificar o caminho completo para arquivar **lib*foo*.a** contendo, ao invés disso, a biblioteca:

```
$ gcc ... caminho/para/libfoo.a ...
```

A partir da extensão do arquivo **.a**, **gcc** entenderá que se trata de uma biblioteca para se conectar com o programa. No entanto, especificar o caminho completo para o arquivo da biblioteca é um método menos flexível.

Usando a opção -Wl

A opção **gcc -Wl** é uma opção especial para passar opções para o linker subjacente. A sintaxe desta opção difere das outras opções **gcc**. A opção **-Wl** é seguida por uma lista separada por vírgula de opções de linker, enquanto outras opções **gcc** requerem uma lista de opções separada por espaço.

O linker **ld** usado por **gcc** oferece as opções **-Bstatic** e **-Bdynamic** para especificar se as bibliotecas que seguem esta opção devem ser vinculadas de forma estática ou dinâmica, respectivamente. Após passar **-Bstatic** e uma biblioteca para o linker, o comportamento padrão de linkamento dinâmico deve ser restaurado manualmente para que as seguintes bibliotecas sejam vinculadas dinamicamente com a opção **-Bdynamic**.

Para vincular um programa, ligue a biblioteca **first** estaticamente (**libfirst.a**) e **second** dinamicamente (**libsecond.so**):

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

NOTA

gcc pode ser configurado para usar links que não o padrão **ld**.

Recursos adicionais

- Usando a Coleção de Compiladores GNU (GCC)

- Documentação para binúteis 2.27

CAPÍTULO 4. CRIANDO BIBLIOTECAS COM GCC

Este capítulo descreve os passos para a criação de bibliotecas e explica os conceitos necessários utilizados pelo sistema operacional Linux para bibliotecas.

4.1. CONVENÇÕES DE NOMENCLATURA DA BIBLIOTECA

Uma convenção especial de nomes de arquivos é usada para bibliotecas: uma biblioteca conhecida como **foo** deve existir como arquivo **lib*foo*.so** ou **lib*foo*.a**. Esta convenção é automaticamente entendida pelas opções de entrada de ligação do GCC, mas não pelas opções de saída:

- Ao criar um link contra a biblioteca, a biblioteca pode ser especificada apenas por seu nome **foo** com a opção **-l** como **-l*foo***:

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

- Ao criar a biblioteca, o nome completo do arquivo **lib*foo*.so** ou **lib*foo*.a** deve ser especificado.

Recursos adicionais

- Seção 4.2, “O mecanismo soname”

4.2. O MECANISMO SONAME

Bibliotecas carregadas dinamicamente (objetos compartilhados) usam um mecanismo chamado *soname* para gerenciar múltiplas versões compatíveis de uma biblioteca.

Pré-requisitos

- Você deve compreender a ligação dinâmica e as bibliotecas.
- Você deve compreender o conceito de compatibilidade ABI.
- Você deve compreender as convenções de nomenclatura das bibliotecas.
- Você deve compreender os vínculos simbólicos.

Introdução do problema

Uma biblioteca carregada dinamicamente (objeto compartilhado) existe como um arquivo executável independente. Isto torna possível atualizar a biblioteca sem atualizar as aplicações que dependem dela. Entretanto, surgem os seguintes problemas com este conceito:

- Identificação da versão atual da biblioteca
- Necessidade de múltiplas versões da mesma biblioteca presentes
- Sinalizando a compatibilidade ABI de cada uma das múltiplas versões

O mecanismo soname

Para resolver isto, o Linux usa um mecanismo chamado soname.

Uma versão da biblioteca **foo** *X.Y* é compatível com outras versões da ABI com o mesmo valor de *X* em um número de versão. Pequenas mudanças preservando a compatibilidade aumentam o número *Y*. Grandes mudanças que quebram a compatibilidade aumentam o número *X*.

A versão atual da biblioteca **foo** *X.Y* existe como um arquivo **libfoo.so.*x.y***. Dentro do arquivo da biblioteca, um soname é gravado com o valor **libfoo.so.x** para sinalizar a compatibilidade.

Quando as aplicações são construídas, o linker procura a biblioteca, procurando o arquivo **libfoo.so**. Um link simbólico com este nome deve existir, apontando para o arquivo real da biblioteca. O linker então lê o soname do arquivo da biblioteca e o grava no arquivo executável da aplicação. Finalmente, o linker cria o aplicativo que declara a dependência da biblioteca usando o soname, não um nome ou um nome de arquivo.

Quando o linker dinâmico de tempo de execução liga um aplicativo antes de ser executado, ele lê o soname do arquivo executável do aplicativo. Este soname é **libfoo.so.*x***. Um vínculo simbólico com este nome deve existir, apontando para o arquivo real da biblioteca. Isto permite carregar a biblioteca, independentemente do componente *Y* de uma versão, porque o soname não muda.

NOTA

O componente *Y* do número de versão não está limitado a um único número. Além disso, algumas bibliotecas codificam sua versão em seu nome.

Leitura do soname de um arquivo

Para exibir o soname de um arquivo da biblioteca **somelibrary**:

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

Substitua *somelibrary* pelo nome real do arquivo da biblioteca que você deseja examinar.

4.3. CRIANDO BIBLIOTECAS DINÂMICAS COM GCC

Bibliotecas dinamicamente ligadas (objetos compartilhados) permitem:

- conservação de recursos através da reutilização de códigos
- aumento da segurança, facilitando a atualização do código da biblioteca

Siga estes passos para construir e instalar uma biblioteca dinâmica a partir da fonte.

Pré-requisitos

- Você deve entender o mecanismo do soname.
- O GCC deve ser instalado no sistema.
- Você deve ter código fonte para uma biblioteca.

Procedimento

1. Mude para o diretório com as fontes da biblioteca.
2. Compilar cada arquivo fonte em um arquivo objeto com a opção de código independente de posição **-fPIC**:

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

Os arquivos objeto têm os mesmos nomes dos arquivos de código fonte originais, mas sua extensão é **.o**.

3. Vincular a biblioteca compartilhada a partir dos arquivos objeto:

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

O número da versão maior é X e o número da versão menor é Y.

4. Copie o ***libfoo.so.x.y*** para um local apropriado, onde o linker dinâmico do sistema possa encontrá-lo. No Red Hat Enterprise Linux, o diretório para bibliotecas é **/usr/lib64**:

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

Note que você precisa de permissões de root para manipular os arquivos neste diretório.

5. Criar a estrutura symlink para o mecanismo soname:

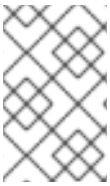
```
# ln -s libfoo.so.x.y libfoo.so.x
# ln -s libfoo.so.x libfoo.so
```

Recursos adicionais

- O Projeto de Documentação Linux

4.4. CRIAÇÃO DE BIBLIOTECAS ESTÁTICAS COM GCC E AR

A criação de bibliotecas para ligação estática é possível através da conversão de arquivos de objetos em um tipo especial de arquivo.



NOTA

A Red Hat desencoraja o uso de ligação estática por razões de segurança. Use links estáticos somente quando necessário, especialmente contra bibliotecas fornecidas pela Red Hat. Veja Seção 3.2, “Ligação estática e dinâmica” para mais detalhes.

Pré-requisitos

- GCC e binutils devem ser instalados no sistema.
- Você deve compreender a ligação estática e dinâmica.
- Arquivo(s) fonte(s) com funções a serem compartilhadas como uma biblioteca estão disponíveis.

Procedimento

1. Criar arquivos de objetos intermediários com GCC.

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

Anexar mais arquivos de origem, se necessário. Os arquivos objeto resultantes compartilham o nome do arquivo, mas use a extensão **.o**.

2. Transforme os arquivos objeto em uma biblioteca estática (arquivo) usando a ferramenta **ar** do pacote **binutils**.

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

O arquivo **libfoo.a** é criado.

3. Use o comando **nm** para inspecionar o arquivo resultante:

```
$nm libfoo.a
```

4. Copie o arquivo estático da biblioteca para o diretório apropriado.

5. Ao vincular-se contra a biblioteca, o GCC reconhecerá automaticamente a partir da extensão do nome do arquivo **.a** que a biblioteca é um arquivo para vínculo estático.

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

Recursos adicionais

- Página do manual Linux para *ar(1)*:

```
$ homem ar
```

CAPÍTULO 5. GERENCIANDO MAIS CÓDIGO COM MAKE

O utilitário GNU make, normalmente abreviado **make**, é uma ferramenta para controlar a geração de executáveis a partir de arquivos fonte. **make** determina automaticamente quais partes de um programa complexo foram alteradas e precisam ser recompiladas. **make** usa arquivos de configuração chamados Makefiles para controlar a forma como os programas são construídos.

5.1. VISÃO GERAL DO GNU MAKE E MAKEFILE

Para criar uma forma utilizável (geralmente arquivos executáveis) a partir dos arquivos de origem de um determinado projeto, execute várias etapas necessárias. Registre as ações e sua seqüência para poder repeti-las posteriormente.

O Red Hat Enterprise Linux contém o GNU **make**, um sistema de construção projetado para este fim.

Pré-requisitos

- Entendendo os conceitos de compilação e vinculação

GNU make

GNU **make** lê Makefiles que contém as instruções que descrevem o processo de construção. Um Makefile contém múltiplos *rules* que descrevem uma forma de satisfazer uma determinada condição (*target*) com uma ação específica (*recipe*). As regras podem depender hierarquicamente de outra regra.

Rodando **make** sem nenhuma opção faz com que ele procure por um Makefile no diretório atual e tente alcançar o alvo padrão. O nome real do arquivo Makefile pode ser um de **Makefile**, **makefile**, e **GNUmakefile**. O alvo padrão é determinado a partir do conteúdo do Makefile.

Detalhes do Makefile

Os makefiles utilizam uma sintaxe relativamente simples para definir *variables* e *rules*, que consiste de um *target* e um *recipe*. O alvo especifica qual é a saída se uma regra é executada. As linhas com receitas devem começar com o caracter TAB.

Normalmente, um Makefile contém regras para compilar arquivos fonte, uma regra para ligar os arquivos objeto resultantes, e um alvo que serve como ponto de entrada no topo da hierarquia.

Considere o seguinte **Makefile** para construir um programa em C que consiste em um único arquivo, **hello.c**.

```
all: hello

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

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

Este exemplo mostra que para atingir a meta **all**, é necessário o arquivo **hello**. Para obter **hello**, é necessário **hello.o** (vinculado por **gcc**), que por sua vez é criado a partir de **hello.c** (compilado por **gcc**).

O alvo **all** é o alvo padrão porque é o primeiro alvo que não começa com um período (.). Rodando **make** sem nenhum argumento é então idêntico a rodar **make all**, quando o diretório atual contém este **Makefile**.

Típico makefile

Um Makefile mais típico usa variáveis para generalização das etapas e adiciona um alvo "limpo" - remover tudo menos os arquivos fonte.

```
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)
```

A adição de mais arquivos fonte a esse Makefile requer apenas adicioná-los à linha onde a variável FONTE está definida.

Recursos adicionais

- Marca GNU: Introdução
- Capítulo 2, *Código de Construção com GCC*

5.2. EXEMPLO: CONSTRUINDO UM PROGRAMA C USANDO UM MAKEFILE

Construa uma amostra do programa C usando um Makefile seguindo as etapas deste exemplo.

Pré-requisitos

- Você deve entender os conceitos de Makefiles e **make**.

Procedimento

1. Crie um diretório **hellomake** e mude para este diretório:

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

2. Crie um arquivo **hello.c** com o seguinte conteúdo:

```
#include <stdio.h>

int main(int argc, char *argv[]) {
```

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

3. Crie um arquivo **Makefile** com o seguinte conteúdo:

```
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)
```

CUIDADO

As linhas de receitas Makefile devem começar com o caractere de tabulação! Ao copiar o texto acima da documentação, o processo de cortar e colar pode colar espaços em vez de abas. Se isso acontecer, corrija o problema manualmente.

4. Rodar **make**:

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

Isto cria um arquivo executável **hello**.

5. Execute o arquivo executável **hello**:

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

6. Execute o alvo Makefile **clean** para remover os arquivos criados:

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

Recursos adicionais

- Seção 2.7, "Exemplo: Construindo um programa C com GCC"
- Seção 2.8, "Exemplo: Construindo um programa C com GCC"

5.3. RECURSOS DE DOCUMENTAÇÃO PARA MAKE

Para maiores informações sobre **make**, veja os recursos listados abaixo.

Documentação instalada

- Use as ferramentas **man** e **info** para visualizar páginas de manual e páginas de informações instaladas em seu sistema:

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

Documentação on-line

- O Manual GNU Make , hospedado pela Free Software Foundation
- Guia do Usuário do Red Hat Developer Toolset

CAPÍTULO 6. MUDANÇAS NA CADEIA DE FERRAMENTAS DESDE A 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. MUDANÇAS NO GCC EM RHEL 8

No Red Hat Enterprise Linux 8, o conjunto de ferramentas GCC é baseado na série de lançamentos GCC 8.2. Mudanças notáveis desde o Red Hat Enterprise Linux 7 incluem:

- Numerosas otimizações gerais foram adicionadas, tais como análise de alias, melhorias de vetorizadores, dobramento de código idêntico, análise inter-processal, passe de otimização de fusão de lojas, e outras.
- O endereço Sanitizer foi melhorado.
- Foi adicionado o Saneante de vazamento para detecção de vazamentos de memória.
- Foi adicionado o Higienizador de Comportamento Indefinido para detecção de comportamento indefinido.
- As informações de depuração podem agora ser produzidas no formato DWARF5. Esta capacidade é experimental.
- A ferramenta de análise de cobertura de código fonte GCOV foi ampliada com várias melhorias.
- O suporte para a especificação OpenMP 4.5 foi adicionado. Além disso, as características de descarga da especificação OpenMP 4.0 são agora suportadas pelos compiladores C, C , e Fortran.
- Novos avisos e diagnósticos melhorados foram acrescentados para a detecção estática de certos erros de programação prováveis.
- Os locais de origem são agora rastreados como intervalos e não como pontos, o que permite diagnósticos muito mais ricos. O compilador agora oferece dicas "fix-it", sugerindo possíveis modificações de código. Um corretor ortográfico foi adicionado para oferecer nomes alternativos e facilitar a detecção de erros de digitação.

Segurança

O GCC foi ampliado para fornecer ferramentas para garantir o endurecimento adicional do código gerado.

Para mais detalhes, veja Seção 6.2, “Melhorias de segurança no GCC em RHEL 8” .

Arquitetura e suporte de processador

As melhorias na arquitetura e no suporte do processador incluem:

- Várias novas opções específicas de arquitetura para a arquitetura Intel AVX-512, várias de suas microarquiteturas e Extensões de Proteção de Software Intel (SGX) foram adicionadas.
- A geração de códigos pode agora visar as extensões LSE de arquitetura ARM de 64 bits, ARMv8.2-A Extensões de ponto flutuante de 16 bits (FPE) e ARMv8.2-A, ARMv8.3-A, e ARMv8.4-A versões de arquitetura.

- O manuseio da opção **-march=native** nas arquiteturas ARM e ARM de 64 bits foi corrigido.
- Foi adicionado suporte para os processadores z13 e z14 da arquitetura IBM Z.

Idiomas e normas

As mudanças notáveis relacionadas a idiomas e normas incluem:

- O padrão padrão usado na compilação de código na linguagem C mudou para C17 com extensões GNU.
- O padrão padrão usado na compilação de código na linguagem C mudou para C 14 com extensões GNU.
- A biblioteca de tempo de execução C agora suporta as normas C 11 e C 14.
- O compilador C agora implementa o padrão C 14 com muitas características novas, tais como modelos variáveis, agregados com inicializadores de dados não estáticos, o especificador estendido **constexpr**, funções de desalocação de tamanho, lambdas genéricas, matrizes de comprimento variável, separadores de dígitos e outros.
- O suporte ao padrão de linguagem C11 foi melhorado: Atomics ISO C11, seleções genéricas e armazenamento local de fios estão agora disponíveis.
- A nova extensão **__auto_type** GNU C fornece um subconjunto da funcionalidade da palavra-chave C 11 **auto** na linguagem C.
- Os nomes dos tipos **_FloatN** e **_FloatNx** especificados pela norma ISO/IEC TS 18661-3:2015 são agora reconhecidos pelo front end C.
- O padrão padrão usado na compilação de código na linguagem C mudou para C17 com extensões GNU. Isto tem o mesmo efeito que o uso da opção **--std=gnu17**. Anteriormente, o padrão era C89 com extensões GNU.
- A GCC pode agora compilar experimentalmente o código usando o padrão de linguagem C 17 e certas características do padrão C 20.
- Passar uma classe vazia como argumento agora não ocupa espaço nas arquiteturas Intel 64 e AMD64, como exigido pela plataforma ABI. Passar ou devolver uma classe com apenas uma cópia eliminada e mover construtores agora usa a mesma convenção de chamada que uma classe com uma cópia não trivial ou mover construtores.
- O valor retornado pelo operador C 11 **alignof** foi corrigido para combinar com o operador C **_Alignof** e retornar alinhamento mínimo. Para encontrar o alinhamento preferido, use a extensão GNU **__alignof__**.
- A versão principal da biblioteca **libgfortran** para o código do idioma Fortran foi alterada para 5.
- O suporte para os idiomas Ada (GNAT), GCC Go e Objective C/C foi removido. Use o conjunto de ferramentas Go para o desenvolvimento do código Go.

Recursos adicionais

- Veja também as Notas de Lançamento do Red Hat Enterprise Linux 8 .
- Usando o Go Toolset

6.2. MELHORIAS DE SEGURANÇA NO GCC EM 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.

Novos avisos

Estas opções de advertência foram acrescentadas:

Opção	Exibe avisos para
-Wstringop-truncation	Chamadas para funções de manipulação de cordas delimitadas tais como **strncat**, **strncpy** e **stpncpy** que podem truncar a corda copiada ou deixar o destino inalterado.
-Wclass-memaccess	Objetos de classes não triviais manipulados de maneiras potencialmente inseguras por funções de memória bruta, como **memcpy** ou **realloc**. O aviso ajuda a detectar chamadas que contornam construtores ou operadores de atribuição de cópia definidos pelo usuário, indicadores de tabela virtual corruptos, membros de dados de tipos ou referências constantes-qualificados, ou indicadores de membros. O aviso também detecta chamadas que contornariam os controles de acesso aos membros dos dados.
-Wmisleading-indentation	Lugares onde a recuo do código dá uma idéia enganosa da estrutura do bloco do código para um leitor humano.
-Walloc-size-larger-than=*size*	Chamadas para funções de alocação de memória onde a quantidade de memória a ser alocada ultrapassa *size*. Funciona também com funções onde a alocação é especificada multiplicando dois parâmetros e com qualquer função decorada com atributo **alloc_size**.
-Walloc-zero	Chamadas para funções de alocação de memória que tentam alocar uma quantidade zero de memória. Funciona também com funções onde a alocação é especificada multiplicando dois parâmetros e com qualquer função decorada com atributo **alloc_size**.
-Walloca	Todas as chamadas para a função **alloca**.
-Walloca-larger-than=*size*	Chamadas para a função **alloca** onde a memória solicitada é superior a *size*.

Opção	Exibe avisos para
-Wvla-larger-than=*size*	Definições de matrizes de comprimento variável (VLA) que podem ou exceder o tamanho especificado ou cujo limite não é conhecido para ser suficientemente limitado.
-Wformat-overflow=*level*	Tanto certo e provável excesso de buffer em chamadas para a família **sprintf** de funções de saída formatadas. Para mais detalhes e explicação do valor *level*, consulte a página do manual*gcc(1)*.
-Wformat-truncation=*level*	Truncagem de saída certa e provável nas chamadas para a família **snprintf** de funções de saída formatadas. Para mais detalhes e explicação do valor *level*, consulte a página do manual*gcc(1)*.
-Wstringop-overflow=*type*	O excesso de buffer em chamadas para funções de manuseio de strings, como **memcpy** e **strcpy**. Para mais detalhes e explicação do valor *level*, consulte a página do manual *gcc(1)*.

Melhorias de advertência

Estas advertências do GCC foram melhoradas:

- A opção **-Warray-bounds** foi melhorada para detectar mais instâncias de índices de matriz fora dos limites e offsets de ponteiro. Por exemplo, índices negativos ou excessivos em membros de matriz flexível e literais de cordas são detectados.
- A opção **-Wrestrict** introduzida no GCC 7 foi aprimorada para detectar muito mais casos de sobreposição de acessos a objetos através de argumentos restritos e qualificados para funções de memória padrão e manipulação de cordas, tais como **memcpy** e **strcpy**.
- A opção **-Wnonnull** foi aperfeiçoada para detectar um conjunto mais amplo de casos de passagem de indicações nulas para funções que esperam um argumento não-nulo (decorado com atributo **nonnull**).

Novo Anti-Comportamento Não Definido

Foi adicionado um novo higienizador de tempo de execução para detectar comportamento indefinido chamado UndefinedBehaviorSanitizer. As seguintes opções são dignas de nota:

Opção	Verifique
-fsanitize=float-divide-by-zero	Detectar divisão de ponto flutuante por zero.
-fsanitize=float-cast-overflow	Verifique se o resultado das conversões de ponto flutuante para inteiro não transborda.

Opção	Verifique
-fsanitize=bounds	Permitir a instrumentação de limites de matriz e detectar acessos fora dos limites.
-fsanitize=alignment	Permitir a verificação do alinhamento e detectar vários objetos desalinhados.
-fsanitize=object-size	Permitir a verificação do tamanho do objeto e detectar vários acessos fora dos limites.
-fsanitize=vptr	Permitir a verificação das chamadas de funções de membros C, acessos de membros e algumas conversões entre apontadores para classes base e derivadas. Além disso, detectar quando os objetos referenciados não têm o tipo dinâmico correto.
-fsanitize=bounds-strict	Permitir uma verificação rigorosa dos limites da matriz. Isto permite **-fsanitize=bounds** e a instrumentação de matrizes flexíveis como matrizes de membros.
-fsanitize=signed-integer-overflow	Diagnosticar transbordos aritméticos mesmo em operações aritméticas com vetores genéricos.
-fsanitize=builtin	Diagnosticar em tempo de execução argumentos inválidos para **__builtin_clz** ou **__builtin_ctz** prefixados construídos. Inclui cheques de **-fsanitize=undefined**.
-fsanitize=pointer-overflow	Realizar testes de tempo de execução baratos para embrulho de ponteiro. Inclui cheques do site **-fsanitize=undefined**.

Novas opções para o AddressSanitizer

Estas opções foram adicionadas ao AddressSanitizer:

Opção	Verifique
-fsanitize=pointer-compare	Advertir sobre a comparação de ponteiros que apontam para um objeto de memória diferente.
-fsanitize=pointer-subtract	Advertir sobre a subtração de ponteiros que apontam para um objeto de memória diferente.
-fsanitize-address-use-after-scope	Sanitizar variáveis cujo endereço é tomado e utilizado após um escopo onde a variável é definida.

Outros higienizadores e instrumentação

- A opção **-fstack-clash-protection** foi adicionada para inserir sondas quando o espaço da pilha é alocado de forma estática ou dinâmica para detectar de forma confiável os transbordamentos da pilha e assim mitigar o vetor de ataque que depende de saltar sobre uma página de proteção da pilha fornecida pelo sistema operacional.

- Uma nova opção **-fcf-protection=[full|branch|return|none]** foi adicionada para executar a instrumentação de código e aumentar a segurança do programa, verificando se os endereços-alvo das instruções de transferência de fluxo de controle (tais como chamada de função indireta, retorno de função, salto indireto) são válidos.

Recursos adicionais

- Para mais detalhes e explicação dos valores fornecidos a algumas das opções acima, consulte a página do manual *gcc(1)*:

```
$ homem gcc
```

6.3. MUDANÇAS DE COMPATIBILIDADE NO GCC EM RHEL 8

C ABI muda em std::string e std::list

A Interface Binária de Aplicação (ABI) das classes **std::string** e **std::list** da biblioteca **libstdc** mudou entre a RHEL 7 (GCC 4.8) e a RHEL 8 (GCC 8) para estar em conformidade com o padrão C 11. A biblioteca **libstdc** suporta tanto a antiga como a nova ABI, mas algumas outras bibliotecas do sistema C não suportam. Como conseqüência, as aplicações que se ligam dinamicamente a essas bibliotecas precisarão ser reconstruídas. Isto afeta todos os modos padrão C, incluindo o C 98. Também afeta as aplicações construídas com compiladores Red Hat Developer Toolset para RHEL 7, que mantiveram a antiga ABI para manter a compatibilidade com as bibliotecas do sistema.

O GCC não constrói mais código Ada, Go e Objective C/C

A capacidade de construir código no Ada (GNAT), GCC Go e Objective C/C languages foi removida do compilador GCC.

Para construir o código Go, use o Go Toolset em seu lugar.

PARTE II. APLICAÇÕES DE DEPURAÇÃO

As aplicações de depuração são um tópico muito amplo. Esta parte fornece a um desenvolvedor as técnicas mais comuns para depuração em múltiplas situações.

CAPÍTULO 7. HABILITANDO A DEPURAÇÃO COM INFORMAÇÕES DE DEPURAÇÃO

Para depurar aplicações e bibliotecas, a informação de depuração é necessária. As seções seguintes descrevem como obter essas informações.

7.1. INFORMAÇÕES SOBRE DEPURAÇÃO

Ao depurar qualquer código executável, dois tipos de informação permitem que as ferramentas, e por extensão o programador, compreendam o código binário:

- o texto do código fonte
- uma descrição de como o texto do código fonte se relaciona com o código binário

Tais informações são chamadas de informações de depuração.

O Red Hat Enterprise Linux usa o formato ELF para binários executáveis, bibliotecas compartilhadas, ou arquivos **debuginfo**. Dentro destes arquivos ELF, o formato DWARF é usado para armazenar as informações de depuração.

Para exibir informações do DWARF armazenadas em um arquivo ELF, execute o **readelf -w *file*** comando.

CUIDADO

O STABS é um formato mais antigo, menos capaz, ocasionalmente utilizado com UNIX. Seu uso é desencorajado pela Red Hat. GCC e GDB fornecem STABS produção e consumo somente na base do melhor esforço. Algumas outras ferramentas, como Valgrind e **elfutils** não funcionam com STABS.

Recursos adicionais

- O Padrão de Depuração DWARF

7.2. HABILITANDO A DEPURAÇÃO DE APLICAÇÕES C E C++ COM GCC

Como a informação de depuração é grande, ela não é incluída em arquivos executáveis por padrão. Para permitir a depuração de suas aplicações C e C++ com ela, você deve instruir explicitamente o compilador a criá-la.

Para permitir a criação de informações de depuração com **GCC** ao compilar e vincular código, use a opção **-g**:

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

- Otimizações realizadas pelo compilador e pelo linker podem resultar em código executável que é difícil de relacionar com o código fonte original: variáveis podem ser otimizadas, loops desenrolados, operações fundidas com as operações ao redor, etc. Isto afeta negativamente a depuração. Para melhorar a experiência de depuração, considere definir a otimização com a opção **-Og**. Entretanto, mudar o nível de otimização altera o código executável e pode mudar o comportamento real, incluindo a remoção de alguns bugs.
- Para incluir também definições macro nas informações de depuração, use a opção **-g3** ao invés de **-g**.

- A opção **-fcompare-debug** GCC testa o código compilado pelo GCC com informações de debug e sem informações de debug. O teste passa se os dois arquivos binários resultantes forem idênticos. Este teste garante que o código executável não seja afetado por nenhuma opção de depuração, o que garante ainda mais que não haja bugs ocultos no código de depuração. Note que o uso da opção **-fcompare-debug** aumenta significativamente o tempo de compilação. Consulte a página do manual GCC para obter detalhes sobre esta opção.

Recursos adicionais

- Capítulo 7, *Habilitando a depuração com informações de depuração*
- Usando a Coleção de Compiladores GNU (GCC)
- Depuração com GDB
- A página do manual do GCC:

```
$ homem gcc
```

7.3. PACOTES DE DEBUGINFO E DEBUGSOURCE

Os pacotes **debuginfo** e **debugsource** contêm informações de depuração e código fonte de depuração para programas e bibliotecas. Para aplicações e bibliotecas instaladas em pacotes dos repositórios do Red Hat Enterprise Linux, você pode obter pacotes separados **debuginfo** e **debugsource** de um canal adicional.

Tipos de pacotes de informações de depuração

Há dois tipos de pacotes disponíveis para depuração:

Pacotes de Debuginfo

Os pacotes **debuginfo** fornecem informações de depuração necessárias para fornecer nomes legíveis por pessoas para recursos de código binário. Estes pacotes contêm arquivos **.debug**, que contêm informações de depuração DWARF. Estes arquivos são instalados no diretório **/usr/lib/debug**.

Pacotes de fonte de depuração

Os pacotes **debugsource** contêm os arquivos fonte utilizados para a compilação do código binário. Com os respectivos pacotes **debuginfo** e **debugsource** instalados, depuradores como o GDB ou LLDB podem relacionar a execução do código binário com o código fonte. Os arquivos do código-fonte são instalados no diretório **/usr/src/debug**.

Diferenças em relação à RHEL 7

No Red Hat Enterprise Linux 7, os pacotes **debuginfo** continham ambos os tipos de informações. O Red Hat Enterprise Linux 8 divide os dados do código fonte necessários para a depuração dos pacotes **debuginfo** em pacotes separados **debugsource**.

Nomes dos pacotes

Um pacote **debuginfo** ou **debugsource** fornece informações de depuração válidas apenas para um pacote binário com o mesmo nome, versão, lançamento e arquitetura:

- Pacote binário ***packagename-version-release.architecture*.rpm**
- Pacote de Debuginfo ***packagename*-debuginfo-*version-release.architecture*.rpm**

- Pacote de fonte de depuração ***packagename*-debugsource-*version*-*release*.*architecture*.rpm**

Recursos adicionais

- Seção 7.1, "Informações sobre depuração"
- Seção 1.2, "Habilitação de depuração e repositórios de fonte"

7.4. OBTENDO PACOTES DE DEBUGINFO PARA UMA APLICAÇÃO OU BIBLIOTECA USANDO GDB

As informações de depuração são necessárias para depurar o código. Para o código que é instalado a partir de um pacote, o GNU Debugger (GDB) reconhece automaticamente as informações de depuração faltantes, resolve o nome do pacote e fornece conselhos concretos sobre como obter o pacote.

Pré-requisitos

- A aplicação ou biblioteca que você deseja depurar deve ser instalada no sistema.
- A GDB e a ferramenta **debuginfo-install** devem ser instaladas no sistema.
- Os canais que fornecem os pacotes **debuginfo** e **debugsource** devem ser configurados e habilitados no sistema.

Procedimento

1. Inicie a GDB anexada à aplicação ou biblioteca que você deseja depurar. A GDB reconhece automaticamente as informações de depuração em falta e sugere um comando para executar.

```
$ 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. Sair da GDB: digite **q** e confirme com **Enter**.

```
(gdb) q
```

3. Execute o comando sugerido pela GDB para instalar os pacotes necessários **debuginfo**:

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

A ferramenta de gerenciamento de pacotes **dnf** fornece um resumo das mudanças, pede confirmação e, uma vez confirmada, baixa e instala todos os arquivos necessários.

4. Caso a GDB não seja capaz de sugerir o pacote **debuginfo**, siga o procedimento descrito em Seção 7.5, "Obtenção manual de pacotes de debuginfo para uma aplicação ou biblioteca" .

Recursos adicionais

- Guia do Usuário do Red Hat Developer Toolset, seção Instalando Informações de Depuração
- Como posso baixar ou instalar pacotes de debuginfo para sistemas RHEL?

7.5. OBTENÇÃO MANUAL DE PACOTES DE DEBUGINFO PARA UMA APLICAÇÃO OU BIBLIOTECA

Você pode determinar manualmente quais pacotes **debuginfo** você precisa instalar, localizando o arquivo executável e depois encontrando o pacote que o instala.

NOTA

A Red Hat recomenda que você use a GDB para determinar os pacotes para instalação . Use este procedimento manual somente se a GDB não for capaz de sugerir o pacote a ser instalado.

Pré-requisitos

- A aplicação ou biblioteca deve ser instalada no sistema.
- A aplicação ou biblioteca foi instalada a partir de um pacote.
- A ferramenta **debuginfo-install** deve estar disponível no sistema.
- Os canais que fornecem os pacotes **debuginfo** devem ser configurados e habilitados no sistema.

Procedimento

1. Encontre o arquivo executável da aplicação ou biblioteca.
   a. Use o comando **which** para encontrar o arquivo de aplicação.

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

   b. Use o comando **locate** para encontrar o arquivo da biblioteca.

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

   Se os motivos originais para depuração incluírem mensagens de erro, escolha o resultado onde a biblioteca tem os mesmos números adicionais em seu nome de arquivo que aqueles mencionados nas mensagens de erro. Em caso de dúvida, tente seguir o restante do procedimento com o resultado onde o nome do arquivo da biblioteca não inclui números adicionais.

NOTA

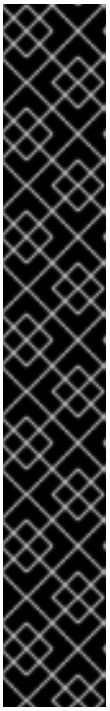
O comando **locate** é fornecido pelo pacote **mlocate**. Para instalá-lo e permitir seu uso:

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

2. Procure por um nome e uma versão do pacote que forneceu o arquivo:

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

A saída fornece detalhes para o pacote instalado no formato *name*:*epoch*-*version*.*release*.*architecture*.



IMPORTANTE

Se esta etapa não produzir nenhum resultado, não é possível determinar qual pacote forneceu o arquivo binário. Há vários casos possíveis:

- O arquivo é instalado a partir de um pacote que não é conhecido pelas ferramentas de gerenciamento de pacotes em sua configuração *current*.
- O arquivo é instalado a partir de um pacote baixado localmente e instalado manualmente. Determinar um pacote adequado **debuginfo** automaticamente é impossível nesse caso.
- Suas ferramentas de gerenciamento de pacotes estão mal configuradas.
- O arquivo não é instalado a partir de nenhum pacote. Em tal caso, não existe o respectivo pacote **debuginfo**.

Como os passos seguintes dependem desta, você deve resolver esta situação ou abortar este procedimento. A descrição exata das etapas de solução de problemas está além do escopo deste procedimento.

3. Instale os pacotes **debuginfo** usando o utilitário **debuginfo-install**. No comando, use o nome do pacote e outros detalhes que você determinou durante a etapa anterior:

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

Recursos adicionais

- Guia do Usuário do Red Hat Developer Toolset, seção Instalando Informações de Depuração
- Como posso baixar ou instalar pacotes de debuginfo para sistemas RHEL?

CAPÍTULO 8. APLICAÇÃO DE INSPEÇÃO DO ESTADO INTERNO COM A GDB

Para descobrir porque uma aplicação não funciona corretamente, controle sua execução e examine seu estado interno com um depurador. Esta seção descreve como usar o depurador GNU (GDB) para esta tarefa.

8.1. DEPURADOR GNU (GDB)

O Red Hat Enterprise Linux contém o depurador GNU (GDB) que lhe permite investigar o que está acontecendo dentro de um programa através de uma interface de usuário de linha de comando.

Para um front end gráfico para GDB, instale o ambiente de desenvolvimento integrado Eclipse. Veja Utilizando o Eclipse.

Capacidades da GDB

Uma única sessão de GDB pode depurar os seguintes tipos de programas:

- Programas multithreaded e de forquilha
- Programas múltiplos ao mesmo tempo
- Programas em máquinas remotas ou em containers com o utilitário **gdbserver** conectado através de uma conexão de rede TCP/IP

Requisitos para a depuração

Para depurar qualquer código executável, a GDB requer informações de depuração para esse código em particular:

- Para programas desenvolvidos por você, você pode criar as informações de depuração enquanto constrói o código.
- Para programas de sistema instalados a partir de pacotes, você deve instalar seus pacotes de debuginfo.

8.2. ANEXANDO A GDB A UM PROCESSO

A fim de examinar um processo, a GDB deve ser *attached* para o processo.

Pré-requisitos

- A GDB deve ser instalada no sistema

Iniciando um programa com a GDB

Quando o programa não estiver sendo executado como um processo, inicie-o com a GDB:

```
$ gdb program
```

Substituir *program* por um nome de arquivo ou caminho para o programa.

A GDB se prepara para iniciar a execução do programa. Você pode configurar pontos de parada e o ambiente **gdb** antes de iniciar a execução do processo com o comando **run**.

Anexar a GDB a um processo já em andamento

Para anexar a GDB a um programa já em execução como um processo:

1. Encontre o ID do processo (*pid*) com o comando **ps**:

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

   Substituir *program* por um nome de arquivo ou caminho para o programa.

2. Anexar a GDB a este processo:

```
$ gdb -p pid
```

   Substitua *pid* por um número de identificação de processo real da saída **ps**.

Anexar um GDB já em funcionamento a um processo já em funcionamento

Para anexar um GDB já em execução a um programa já em execução:

1. Use o comando **shell** GDB para executar o comando **ps** e encontrar o ID de processo do programa (*pid*):

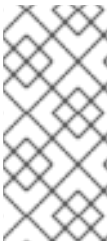
```
(gdb) shell ps -C program -o pid h
 pid
```

   Substituir *program* por um nome de arquivo ou caminho para o programa.

2. Use o comando **attach** para anexar o GDB ao programa:

```
(gdb) anexar pid
```

   Substituir *pid* por um número de identificação do processo real da saída **ps**.



NOTA

Em alguns casos, a GDB pode não ser capaz de encontrar o respectivo arquivo executável. Use o comando **file** para especificar o caminho:

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

Recursos adicionais

- Depuração com GDB
- Depuração com GDB

8.3. PASSANDO PELO CÓDIGO DO PROGRAMA COM A GDB

Uma vez que o depurador **GDB** esteja anexado a um programa, você pode usar uma série de comandos para controlar a execução do programa.

Pré-requisitos

- Você deve ter as informações de depuração necessárias disponíveis:
  - O programa é compilado e construído com informações de depuração, ou
  - Os pacotes de debuginfo relevantes são instalados
- A GDB deve ser anexada ao programa para ser depurada

Comandos GDB para passar através do código

r (executado)

Iniciar a execução do programa. Se **run** for executado com qualquer argumento, esses argumentos são passados ao executável como se o programa tivesse sido iniciado normalmente. Os usuários normalmente emitem este comando após definir os pontos de interrupção.

start

Iniciar a execução do programa, mas parar no início da função principal do programa. Se **start** for executado com qualquer argumento, esses argumentos são passados ao executável como se o programa tivesse sido iniciado normalmente.

c (continuar)

Continuar a execução do programa a partir do estado atual. A execução do programa continuará até que uma das seguintes situações se torne realidade:

- Um ponto de parada é atingido.
- Uma condição especificada é satisfeita.
- Um sinal é recebido pelo programa.
- Ocorre um erro.
- O programa é encerrado.

n (próximo)

Continuar a execução do programa a partir do estado atual, até que a próxima linha de código no arquivo fonte atual seja alcançada. A execução do programa continuará até que uma das seguintes situações se torne realidade:

- Um ponto de parada é atingido.
- Uma condição especificada é satisfeita.
- Um sinal é recebido pelo programa.
- Ocorre um erro.
- O programa é encerrado.

s (etapa)

O comando **step** também suspende a execução em cada linha seqüencial de código no arquivo fonte atual. Entretanto, se a execução for atualmente interrompida em uma linha de código fonte contendo um **function call**, o GDB interrompe a execução após entrar na chamada de função (em vez de executá-la).

until *location*

Continuar a execução até o local de código especificado pela opção *location* ser alcançado.

fini (acabamento)

Retomar a execução do programa e interrompê-la quando a execução retornar de uma função. A execução do programa continuará até que uma das seguintes ações se torne realidade:

- Um ponto de parada é atingido.
- Uma condição especificada é satisfeita.
- Um sinal é recebido pelo programa.
- Ocorre um erro.
- O programa é encerrado.

q (desistir)

Encerrar a execução e sair da GDB.

Recursos adicionais

- Seção 8.5, "Utilização de pontos de parada GDB para parar a execução em locais com código definido"
- Depuração com GDB
- Depuração com GDB

8.4. MOSTRANDO VALORES INTERNOS DO PROGRAMA COM A GDB

A exibição dos valores das variáveis internas de um programa é importante para a compreensão do que o programa está fazendo. A GDB oferece comandos múltiplos que você pode usar para inspecionar as variáveis internas. Esta seção descreve o mais útil destes comandos:

p (imprimir)

Mostrar o valor do argumento apresentado. Normalmente, o argumento é o nome de uma variável de qualquer complexidade, desde um simples valor único até uma estrutura. Um argumento também pode ser uma expressão válida na linguagem atual, incluindo o uso de variáveis de programa e funções de biblioteca, ou funções definidas no programa que está sendo testado.
É possível estender o GDB com *pretty-printer* Python ou Guile scripts para exibição personalizada de estruturas de dados (tais como classes, estruturas) usando o comando **print**.

bt (backtrace)

Exibir a cadeia de chamadas de funções usadas para alcançar o ponto de execução atual, ou a cadeia de funções usadas até que a execução fosse encerrada. Isto é útil para investigar erros graves (como falhas de segmentação) com causas elusivas.
Adicionando a opção **full** ao comando **backtrace** também exibe variáveis locais.

É possível estender o GDB com *frame filter* scripts Python para exibição personalizada dos dados exibidos usando os comandos **bt** e **info frame**. O termo *frame* refere-se aos dados associados a uma única chamada de função.

info

O comando **info** é um comando genérico para fornecer informações sobre vários itens. Ele toma uma opção especificando o item a ser descrito.

- O comando **info args** exibe opções da chamada de função que é o quadro atualmente selecionado.
- O comando **info locals** exibe variáveis locais no quadro selecionado atualmente.

Para obter uma lista dos itens possíveis, execute o comando **help info** em uma sessão da GDB:

```
(gdb) informações de ajuda
```

l (lista)

Mostrar a linha no código fonte onde o programa parou. Este comando está disponível somente quando a execução do programa é interrompida. Embora não seja estritamente um comando para mostrar o estado interno, **list** ajuda o usuário a entender que mudanças no estado interno acontecerão na próxima etapa da execução do programa.

Recursos adicionais

- O GDB Python API
- Depuração com GDB

8.5. UTILIZAÇÃO DE PONTOS DE PARADA GDB PARA PARAR A EXECUÇÃO EM LOCAIS COM CÓDIGO DEFINIDO

Muitas vezes, apenas pequenas porções de código são investigadas. Os pontos de parada são marcadores que dizem à GDB para parar a execução de um programa em um determinado lugar no código. Os pontos de parada são mais comumente associados a linhas de código fonte. Nesse caso, a colocação de um ponto de parada requer a especificação do arquivo fonte e do número da linha.

- Para **place a breakpoint**:
  - Especifique o nome do código fonte *file* e o *line* nesse arquivo:

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

  - Quando *file* não está presente, o nome do arquivo fonte no ponto de execução atual é usado:

    ```
    (gdb) br line
    ```

  - Alternativamente, use um nome de função para colocar o ponto de parada em seu início:

    ```
    (gdb) br function_name
    ```

- Um programa pode encontrar um erro após um certo número de iterações de uma tarefa. Para especificar um adicional **condition** para interromper a execução:

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

Substituir *condition* por uma condição no idioma C ou C. O significado de *file* e *line* é o mesmo que acima.

- Para **inspect**, o status de todos os pontos de parada e de vigia:

```
(gdb) info br
```

- Para **remove** um ponto de parada, utilizando seu *number* como mostrado na saída de **info br**:

```
(gdb) apagar number
```

- Para **remove** um ponto de parada em um determinado local:

```
(gdb) claro file:line
```

Recursos adicionais

- Depuração com GDB

8.6. UTILIZAÇÃO DE PONTOS DE VIGILÂNCIA GDB PARA INTERROMPER A EXECUÇÃO DE ACESSO AOS DADOS E MUDANÇAS

Em muitos casos, é vantajoso deixar o programa executar até que certos dados mudem ou sejam acessados. Esta seção lista os casos de uso mais comuns.

Pré-requisitos

- Compreensão **GDB**

Usando pontos de vigia na GDB

Os pontos de vigilância são marcadores que dizem a **GDB** para parar a execução de um programa. Pontos de vigilância estão associados a dados: a colocação de um ponto de vigilância requer a especificação de uma expressão que descreve uma variável, múltiplas variáveis, ou um endereço de memória.

- Para **place** um ponto de observação de dados **change** (escrever):

```
(gdb) relógio expression
```

Substitua *expression* por uma expressão que descreva o que você quer assistir. Para as variáveis, *expression* é igual ao nome da variável.

- Para **place** um ponto de observação de dados **access** (leia-se):

```
(gdb) rwatch expression
```

- Para **place** um ponto de vigilância para acesso aos dados **any** (tanto para leitura como para escrita):

```
(gdb) awatch expression
```

- Para **inspect** o status de todos os pontos de vigilância e pontos de parada:

```
(gdb) info br
```

- Para **remove** um ponto de vigilância:

```
(gdb) apagar num
```

Substitua o ***num*** com o número reportado pelo comando **info br**.

Recursos adicionais

- Depuração com GDB

8.7. DEPURAÇÃO DE FORQUILHAS OU PROGRAMAS ROSQUEADOS COM GDB

Alguns programas utilizam forquilhas ou roscas para conseguir a execução de códigos paralelos. A depuração de múltiplos caminhos de execução simultânea requer considerações especiais.

Pré-requisitos

- Você deve compreender os conceitos de bifurcação e rosca do processo.

Depuração de programas bifurcados com GDB

O forking é uma situação em que um programa (**parent**) cria uma cópia independente de si mesmo (**child**). Use as seguintes configurações e comandos para afetar o que a GDB faz quando um garfo ocorre:

- A configuração **follow-fork-mode** controla se a GDB segue os pais ou a criança após o garfo.

  set follow-fork-mode parent

  Depois de um garfo, depurar o processo pai. Este é o padrão.

  set follow-fork-mode child

  Depois de um garfo, depurar o processo da criança.

  show follow-fork-mode

  Exibir a configuração atual do **follow-fork-mode**.

- A configuração **set detach-on-fork** controla se a GDB mantém o controle do outro processo (não seguido) ou o deixa em funcionamento.

  set detach-on-fork on

  O processo que não é seguido (dependendo do valor de **follow-fork-mode**) é destacado e funciona de forma independente. Este é o padrão.

  set detach-on-fork off

  A GDB mantém o controle de ambos os processos. O processo que é seguido (dependendo do valor de **follow-fork-mode**) é depurado como de costume, enquanto o outro é suspenso.

  show detach-on-fork

  Exibir a configuração atual do **detach-on-fork**.

Depuração de programas roscados com GDB

A GDB tem a capacidade de depurar os fios individuais, e de manipulá-los e examiná-los

independentemente. Para fazer a GDB parar somente a linha que é examinada, use os comandos **set non-stop on** e **set target-async on**. Você pode adicionar estes comandos ao arquivo **.gdbinit**. Após essa funcionalidade ser ativada, a GDB está pronta para realizar a depuração de threads.

A GDB utiliza um conceito de *current thread*. Por padrão, os comandos se aplicam somente à linha atual.

info threads

Mostrar uma lista de tópicos com seus números **id** e **gid**, indicando o tópico atual.

thread *id*

Defina a linha com o especificado **id** como a linha atual.

thread apply *ids command*

Aplique o comando ***command*** a todos os tópicos listados por ***ids***. O ***ids*** é uma lista separada por espaço de ids de linha. Um valor especial **all** aplica o comando a todos os threads.

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

Estabelecer um ponto de parada em um certo ***location*** com um certo ***condition*** somente para o número do fio ***id***.

watch *expression* thread *id*

Estabeleça um ponto de vigia definido por ***expression*** somente para o número do fio ***id***.

command&

Executar o comando **command** e retornar imediatamente ao prompt do gdb **(gdb)**, continuando qualquer execução de código em segundo plano.

interrupt

Parar a execução em segundo plano.

Recursos adicionais

- Depuração com GDB
- Depuração com GDB

CAPÍTULO 9. INTERAÇÕES DA APLICAÇÃO DE GRAVAÇÃO

O código executável das aplicações interage com o código do sistema operacional e bibliotecas compartilhadas. O registro de um registro de atividades destas interações pode fornecer uma visão suficiente sobre o comportamento da aplicação sem depurar o código da aplicação real. Alternativamente, a análise das interações de uma aplicação pode ajudar a identificar as condições nas quais um bug se manifesta.

9.1. FERRAMENTAS ÚTEIS PARA O REGISTRO DE INTERAÇÕES DE APLICAÇÕES

O Red Hat Enterprise Linux oferece múltiplas ferramentas para analisar as interações de uma aplicação.

strace

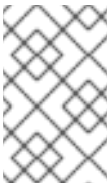
A ferramenta **strace** permite principalmente o registro de chamadas de sistema (funções do kernel) utilizadas por uma aplicação.

- A ferramenta **strace** pode fornecer uma saída detalhada sobre chamadas, porque **strace** interpreta parâmetros e resultados com conhecimento do código do kernel subjacente. Os números são transformados nos respectivos nomes constantes, bandeiras combinadas bitwise expandidas para lista de bandeiras, ponteiros para conjuntos de caracteres desreferenciados para fornecer a string real, e muito mais. Pode estar faltando suporte para características mais recentes do kernel.
- Você pode filtrar as chamadas rastreadas para reduzir a quantidade de dados capturados.
- O uso do **strace** não requer nenhuma configuração em particular, exceto para a instalação do filtro de log.
- O rastreamento do código da aplicação com **strace** resulta em uma desaceleração significativa da execução da aplicação. Como resultado, **strace** não é adequado para muitas implantações de produção. Como alternativa, considere o uso de **ltrace** ou SystemTap.
- A versão de **strace** disponível no Red Hat Developer Toolset também pode realizar a adulteração de chamadas do sistema. Esta capacidade é útil para a depuração.

ltrace

A ferramenta **ltrace** permite o registro das chamadas de espaço do usuário de uma aplicação em objetos compartilhados (bibliotecas dinâmicas).

- A ferramenta **ltrace** permite o rastreamento de chamadas para qualquer biblioteca.
- Você pode filtrar as chamadas rastreadas para reduzir a quantidade de dados capturados.
- O uso do **ltrace** não requer nenhuma configuração em particular, exceto para a instalação do filtro de log.
- A ferramenta **ltrace** é leve e rápida, oferecendo uma alternativa ao **strace**: é possível rastrear as respectivas interfaces em bibliotecas como **glibc** com **ltrace** em vez de rastrear as funções do kernel com **strace**.
- Como **ltrace** não lida com um conjunto conhecido de chamadas como **strace**, ele não tenta explicar os valores passados para as funções da biblioteca. A saída de **ltrace** contém apenas números brutos e ponteiros. A interpretação da saída **ltrace** requer a consulta das declarações de interface reais das bibliotecas presentes na saída.



NOTA

No Red Hat Enterprise Linux 8.0, um problema conhecido impede que **ltrace** rastreie arquivos executáveis do sistema. Esta limitação não se aplica a arquivos executáveis construídos pelos usuários.

SystemTap

SystemTap é uma plataforma de instrumentação para sondar processos em execução e atividades de kernel no sistema Linux. SystemTap usa sua própria linguagem de script para programação de manipuladores de eventos personalizados.

- Em comparação com o uso de **strace** e **ltrace**, o registro de dados significa mais trabalho na fase inicial de configuração. Entretanto, as capacidades de scripting estendem a utilidade do SystemTap além da simples produção de logs.
- SystemTap funciona através da criação e inserção de um módulo de núcleo. O uso do SystemTap é eficiente e não cria uma desaceleração significativa do sistema ou da execução da aplicação por si só.
- O SystemTap vem com um conjunto de exemplos de uso.

GDB

O GNU Debugger (GDB) destina-se principalmente à depuração, não ao registro. Entretanto, algumas de suas características o tornam útil mesmo no cenário em que a interação de um aplicativo é a principal atividade de interesse.

- Com a GDB, é possível combinar convenientemente a captura de um evento de interação com a depuração imediata do caminho de execução subseqüente.
- A GDB é mais adequada para analisar a resposta a eventos infrequentes ou singulares, após a identificação inicial da situação problemática por outras ferramentas. O uso da GDB em qualquer cenário com eventos frequentes torna-se ineficiente ou mesmo impossível.

Recursos adicionais

- Guia para Iniciantes do Red Hat Enterprise Linux SystemTap
- Guia do Usuário do Red Hat Developer Toolset

9.2. MONITORAR AS CHAMADAS DO SISTEMA DE UMA APLICAÇÃO COM STRACE

A ferramenta **strace** permite monitorar as chamadas do sistema (kernel) realizadas por uma aplicação.

Pré-requisitos

- Você deve ter **strace** instalado no sistema.

Procedimento

1. Identificar as chamadas do sistema a serem monitoradas.
2. Iniciar **strace** e anexá-lo ao programa.

- Se o programa que você deseja monitorar não estiver em execução, inicie **strace** e especifique o *program*:

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

- Se o programa já estiver em execução, encontrar sua identificação de processo (*pid*) e anexar **strace** a ele:

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

- Substituir *call* pelas chamadas do sistema a serem exibidas. Você pode usar o **-e trace=*call*** opção várias vezes. Se omitido, **strace** exibirá todos os tipos de chamadas do sistema. Consulte a página do manual *strace(1)* para mais informações.

- Se você não quiser rastrear nenhum processo bifurcado ou rosca, deixe de fora a opção **-f**.

3. A ferramenta **strace** exibe as chamadas do sistema feitas pela aplicação e seus detalhes.
   Na maioria dos casos, uma aplicação e suas bibliotecas fazem um grande número de chamadas e a saída de **strace** aparece imediatamente, se nenhum filtro para chamadas ao sistema for definido.

4. A ferramenta **strace** sai quando o programa sai.
   Para encerrar o monitoramento antes da saída do programa rastreado, pressione **Ctrl C**.

   - Se **strace** iniciou o programa, o programa termina junto com **strace**.

   - Se você anexou **strace** a um programa já em execução, o programa termina juntamente com **strace**.

5. Analisar a lista de chamadas de sistema feitas pela aplicação.

   - Os problemas de acesso ou disponibilidade de recursos estão presentes no registro como erros de retorno de chamadas.

   - Os valores passados para as chamadas ao sistema e os padrões de seqüências de chamadas fornecem uma visão das causas do comportamento da aplicação.

   - Se a aplicação falhar, as informações importantes estarão provavelmente no final do registro.

   - A saída contém muitas informações desnecessárias. Entretanto, você pode construir um filtro mais preciso para as chamadas de interesse do sistema e repetir o procedimento.

NOTA

É vantajoso tanto ver a saída quanto salvá-la em um arquivo. Use o comando **tee** para conseguir isso:

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

Recursos adicionais

- A página do manual *strace(1)*:

```
$ homem strace
```

- Como usar o strace para rastrear chamadas de sistema feitas por um comando?
- Guia do Usuário do Red Hat Developer Toolset

9.3. MONITORAMENTO DAS CHAMADAS DE FUNÇÃO DA BIBLIOTECA DA APLICAÇÃO COM LTRACE

A ferramenta **ltrace** permite monitorar as chamadas de uma aplicação para funções disponíveis em bibliotecas (objetos compartilhados).

NOTA

No Red Hat Enterprise Linux 8.0, um problema conhecido impede que **ltrace** rastreie arquivos executáveis do sistema. Esta limitação não se aplica a arquivos executáveis construídos pelos usuários.

Pré-requisitos

- Você deve ter **ltrace** instalado no sistema.

Procedimento

1. Identificar as bibliotecas e funções de interesse, se possível.
2. Iniciar **ltrace** e anexá-lo ao programa.
   - Se o programa que você deseja monitorar não estiver em execução, inicie **ltrace** e especifique *program*:

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

   - Se o programa já estiver em execução, encontrar sua identificação de processo (*pid*) e anexar **ltrace** a ele:

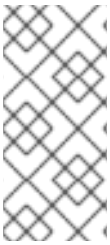
     ```
     $ ps -C program
     (...)
     $ ltrace -f -l library -e function program -ppid
     ```

   - Use as opções **-e**, **-f** e **-l** para filtrar a saída:
     - Forneça os nomes das funções a serem exibidas como *function*. O **-e *function*** pode ser usado várias vezes. Se omitido, **ltrace** exibe chamadas para todas as funções.
     - Em vez de especificar funções, você pode especificar bibliotecas inteiras com o **-l *library*** opção. Esta opção se comporta de forma semelhante à **-e *function*** opção.
     - Se você não quiser rastrear nenhum processo bifurcado ou rosca, deixe de fora a opção **-f**.

     Consulte a página *ltrace*(1)_ manual para mais informações.
3. **ltrace** exibe as chamadas da biblioteca feitas pela aplicação.

Na maioria dos casos, uma aplicação faz um grande número de chamadas e a saída **ltrace** aparece imediatamente, se nenhum filtro for configurado.

4. **ltrace** sai quando o programa sai.
   Para encerrar o monitoramento antes da saída do programa rastreado, pressione **ctrl C**.
   - Se **ltrace** iniciou o programa, o programa termina junto com **ltrace**.
   - Se você anexou **ltrace** a um programa já em execução, o programa termina junto com **ltrace**.
5. Analisar a lista de chamadas da biblioteca feitas pelo aplicativo.
   - Se a aplicação falhar, as informações importantes estarão provavelmente no final do registro.
   - A saída contém muitas informações desnecessárias. Entretanto, é possível construir um filtro mais preciso e repetir o procedimento.



NOTA

É vantajoso tanto ver a saída quanto salvá-la em um arquivo. Use o comando **tee** para conseguir isso:

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

Recursos adicionais

- A página do manual *ltrace(1)*:

```
$ homem ltrace
```

- Guia do Usuário do Red Hat Developer Toolset

9.4. MONITORAMENTO DE CHAMADAS DO SISTEMA DE APLICAÇÃO COM SYSTEMTAP

A ferramenta SystemTap permite o registro de manipuladores de eventos personalizados para eventos de kernel. Em comparação com a ferramenta **strace**, é mais difícil de usar, mas mais eficiente e permite uma lógica de processamento mais complicada. Um script SystemTap chamado **strace.stp** é instalado junto com o SystemTap e fornece uma aproximação da funcionalidade **strace** usando o SystemTap.

Pré-requisitos

- O SystemTap e os respectivos pacotes de kernel devem ser instalados no sistema.

Procedimento

1. Encontre o ID do processo (*pid*) do processo que você deseja monitorar:

```
$ ps -aux
```

2. Execute SystemTap com o script **strace.stp**:

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

O valor de *pid* é a identificação do processo.

O script é compilado em um módulo do kernel, que é então carregado. Isto introduz um pequeno atraso entre a entrada do comando e a obtenção da saída.

3. Quando o processo realiza uma chamada ao sistema, o nome da chamada e seus parâmetros são impressos no terminal.

4. O script sai quando o processo termina, ou quando você pressiona **Ctrl C**.

9.5. USANDO A GDB PARA INTERCEPTAR CHAMADAS DE SISTEMA DE APLICAÇÃO

O GNU Debugger (GDB) permite interromper uma execução em várias situações que surgem durante a execução do programa. Para interromper a execução quando o programa executa uma chamada ao sistema, use um GDB *catchpoint*.

Pré-requisitos

- Você deve compreender o uso de pontos de parada GDB.
- A GDB deve ser anexada ao programa.

Procedimento

1. Defina o ponto de captação:

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

O comando **catch syscall** define um tipo especial de ponto de parada que interrompe a execução quando o programa executa uma chamada ao sistema.

O ***syscall-name*** especifica o nome da chamada. Você pode especificar vários pontos de captura para várias chamadas de sistema. Deixando de fora o ***syscall-name*** faz com que a GDB pare em qualquer chamada ao sistema.

2. Iniciar a execução do programa.

   - Se o programa ainda não começou a ser executado, inicie-o:

```
(gdb) r
```

   - Se a execução do programa for interrompida, retomá-la:

```
(gdb) c
```

3. A GDB suspende a execução depois que o programa executa qualquer chamada de sistema especificada.

Recursos adicionais

- Seção 8.4, “Mostrando valores internos do programa com a GDB”

- Seção 8.3, “Passando pelo código do programa com a GDB”
- Depuração com GDB

9.6. USANDO A GDB PARA INTERCEPTAR O MANUSEIO DE SINAIS POR APLICAÇÕES

O GNU Debugger (GDB) permite interromper a execução em várias situações que surgem durante a execução do programa. Para interromper a execução quando o programa recebe um sinal do sistema operacional, use um GDB *catchpoint*.

Pré-requisitos

- Você deve compreender o uso de pontos de parada GDB.
- A GDB deve ser anexada ao programa.

Procedimento

1. Defina o ponto de captação:

   ```
   (gdb) sinal de captura signal-type
   ```

   O comando **catch signal** estabelece um tipo especial de ponto de parada que interrompe a execução quando um sinal é recebido pelo programa. O ***signal-type*** especifica o tipo de sinal. Use o valor especial **'all'** para capturar todos os sinais.

2. Deixe o programa funcionar.
   - Se o programa ainda não começou a ser executado, inicie-o:

     ```
     (gdb) r
     ```

   - Se a execução do programa for interrompida, retomá-la:

     ```
     (gdb) c
     ```

3. A GDB interrompe a execução depois que o programa recebe qualquer sinal especificado.

Recursos adicionais

- Seção 8.4, “Mostrando valores internos do programa com a GDB”
- Seção 8.3, “Passando pelo código do programa com a GDB”
- Depuração com GDB

CAPÍTULO 10. DEPURAÇÃO DE UMA APLICAÇÃO DE CRASHED

Às vezes, não é possível depurar uma aplicação diretamente. Nessas situações, é possível coletar informações sobre o pedido no momento de seu término e analisá-lo posteriormente.

10.1. LIXEIRAS: O QUE SÃO E COMO UTILIZÁ-LAS

Um despejo de núcleo é uma cópia de uma parte da memória da aplicação no momento em que a aplicação parou de funcionar, armazenada no formato ELF. Ele contém todas as variáveis internas e a pilha da aplicação, o que permite a inspeção do estado final da aplicação. Quando ampliado com o respectivo arquivo executável e informações de depuração, é possível analisar um arquivo de despejo central com um depurador de forma semelhante à análise de um programa em execução.

O kernel do sistema operacional Linux pode gravar os despejos do núcleo automaticamente, se esta funcionalidade estiver habilitada. Alternativamente, você pode enviar um sinal para qualquer aplicação em execução para gerar um despejo de núcleo, independentemente de seu estado real.

ATENÇÃO

Alguns limites podem afetar a capacidade de gerar uma lixeira de núcleo. Para ver os limites atuais:

```
$ ulimit -a
```

10.2. FALHAS NA APLICAÇÃO DE GRAVAÇÃO COM LIXEIRAS DE NÚCLEO

Para registrar falhas de aplicação, configurar o núcleo de economia de despejo e adicionar informações sobre o sistema.

Procedimento

1. Para ativar as lixeiras do núcleo, certifique-se de que o arquivo **/etc/systemd/system.conf** contenha as seguintes linhas:

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

   Você também pode adicionar comentários descrevendo se estas configurações estavam presentes anteriormente, e quais eram os valores anteriores. Isto permitirá que você reverta estas mudanças mais tarde, se necessário. Os comentários são linhas que começam com o caracter **#**.

   A alteração do arquivo requer acesso em nível de administrador.

2. Aplique a nova configuração:

```
# daemon-reexec systemctl
```

3. Remover os limites para os tamanhos de despejo do núcleo:

```
# ulimit -c ilimitado
```

Para reverter esta mudança, execute o comando com valor **0** em vez de ***unlimited***.

4. Instale o pacote **sos** que fornece o utilitário **sosreport** para a coleta de informações do sistema:

```
# yum instalar sos
```

5. Quando uma aplicação trava, um depósito central é gerado e tratado por **systemd-coredump**.

6. Criar um relatório SOS para fornecer informações adicionais sobre o sistema:

```
# sosreport
```

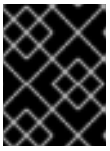
Isto cria um arquivo **.tar** contendo informações sobre seu sistema, tais como cópias de arquivos de configuração.

7. Localizar e exportar o despejo do núcleo:

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

Se a aplicação falhou várias vezes, a saída do primeiro comando lista mais lixeiras de núcleo capturado. Nesse caso, construir para o segundo comando uma consulta mais precisa usando as outras informações. Consulte a página do manual *coredumpctl(1)* para obter detalhes.

8. Transferir o despejo do núcleo e o relatório SOS para o computador onde a depuração será realizada. Transferir também o arquivo executável, se for conhecido.



IMPORTANTE

Quando o arquivo executável não é conhecido, a análise subseqüente do arquivo principal o identifica.

9. Opcional: Remover o despejo do núcleo e o relatório SOS depois de transferi-los, para liberar espaço em disco.

Recursos adicionais

- Introdução ao sistemad no documento *Configuring basic system settings*
- Como habilitar lixeiras de arquivos centrais quando uma aplicação trava ou falha de segmentação
- O que é um sosreport e como criar um no Red Hat Enterprise Linux 4.6 e posteriores?

10.3. INSPEÇÃO DE ESTADOS DE COLISÃO DE APLICAÇÕES COM LIXÕES DE NÚCLEO

Pré-requisitos

- Você deve ter um arquivo principal de despejo e um relatório do sistema onde ocorreu o acidente.
- GDB e elfutils devem ser instalados em seu sistema.

Procedimento

1. Para identificar o arquivo executável onde ocorreu a falha, execute o comando **eu-unstrip** com o arquivo dump do 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
```

   A saída contém detalhes para cada módulo em uma linha, separados por espaços. As informações estão listadas nesta ordem:

   1. O endereço de memória onde o módulo foi mapeado
   2. A parte interna do módulo e onde na memória foi encontrado
   3. O nome do arquivo executável do módulo, exibido como **-** quando desconhecido, ou como **.** quando o módulo não tiver sido carregado de um arquivo
   4. A fonte de informações de depuração, exibida como um nome de arquivo quando disponível, como **.** quando contido no próprio arquivo executável, ou como **-** quando não presente em absoluto
   5. O nome da biblioteca compartilhada (*soname*) ou **[exe]** para o módulo principal

   Neste exemplo, os detalhes importantes são o nome do arquivo **/usr/bin/sleep** e o build-id **2818b2009547f780a5639c904cded443e564973e** na linha que contém o texto **[exe]**. Com estas informações, é possível identificar o arquivo executável necessário para a análise do despejo do núcleo.

2. Obtenha o arquivo executável que falhou.
   - Se possível, copie-a do sistema onde ocorreu o acidente. Use o nome do arquivo extraído do arquivo principal.
   - Você também pode usar um arquivo executável idêntico em seu sistema. Cada arquivo executável construído no Red Hat Enterprise Linux contém uma nota com um valor de build-id único. Determine o build-id dos arquivos executáveis relevantes disponíveis localmente:

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

Use estas informações para comparar o arquivo executável no sistema remoto com sua cópia local. O build-id do arquivo local e o build-id listado no lixão central devem coincidir.

- Finalmente, se o aplicativo for instalado a partir de um pacote RPM, você pode obter o arquivo executável a partir do pacote. Use a saída **sosreport** para encontrar a versão exata do pacote necessário.

3. Obtenha as bibliotecas compartilhadas utilizadas pelo arquivo executável. Use os mesmos passos que para o arquivo executável.

4. Se o aplicativo for distribuído como um pacote, carregue o arquivo executável no GDB, para exibir dicas de pacotes de debuginfo em falta. Para mais detalhes, veja Seção 7.4, "Obtendo pacotes de debuginfo para uma aplicação ou biblioteca usando GDB".

5. Para examinar o arquivo principal em detalhes, carregue o arquivo executável e o arquivo principal de despejo com a GDB:

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

Outras mensagens sobre arquivos ausentes e informações de depuração ajudam a identificar o que está faltando para a sessão de depuração. Retornar à etapa anterior, se necessário.

Se a informação de depuração da aplicação estiver disponível como um arquivo em vez de um pacote, carregue este arquivo no GDB com o comando **symbol-file**:

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

Substituir *program.debug* pelo nome do arquivo real.

NOTA

Talvez não seja necessário instalar as informações de depuração para todos os arquivos executáveis contidos no lixão principal. A maioria destes arquivos executáveis são bibliotecas utilizadas pelo código da aplicação. Estas bibliotecas podem não contribuir diretamente para o problema que você está analisando, e você não precisa incluir informações de debugging para eles.

6. Use os comandos da GDB para inspecionar o estado da aplicação no momento em que ela caiu. Ver Capítulo 8, *Aplicação de inspeção do Estado Interno com a GDB* .

NOTA

Ao analisar um arquivo principal, a GDB não é anexada a um processo em execução. Os comandos para controlar a execução não têm efeito.

Recursos adicionais

- Depuração com GDB
- Depuração com GDB
- Depuração com GDB

10.4. CRIAÇÃO E ACESSO A UM DEPÓSITO CENTRAL DE LIXO COM COREDUMPCTL

A ferramenta **coredumpctl** de **systemd** pode agilizar significativamente o trabalho com lixões no núcleo da máquina onde ocorreu o acidente. Este procedimento delineia como capturar um despejo de núcleo de um processo não responsivo.

Pré-requisitos

- O sistema deve ser configurado para usar **systemd-coredump** para o manuseio do núcleo de despejo. Para verificar isto é verdade:

```
$ sysctl kernel.core_pattern
```

  A configuração é correta se a saída começar com o seguinte:

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

Procedimento

1. Encontre o PID do processo pendurado, baseado em uma parte conhecida do nome do arquivo executável:

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

   Este comando irá emitir uma linha na forma

```
PID command-line
```

   Use o valor *command-line* para verificar se o *PID* pertence ao processo pretendido.

   Por exemplo:

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

2. Enviar um sinal de aborto para o processo:

```
# matar -ABRT PID
```

3. Verifique se o núcleo foi capturado por **coredumpctl**:

```
Lista de coredumpctl de dólares PID
```

   Por exemplo:

```
$ 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. Examinar ou usar o arquivo principal conforme necessário.

Você pode especificar o despejo do núcleo pelo PID e outros valores. Consulte a página do manual *coredumpctl(1)* para mais detalhes.

- Para mostrar detalhes do arquivo principal:

```
$ coredumpctl info PID
```

- Para carregar o arquivo central no depurador GDB:

```
$ coredumpctl debug PID
```

Dependendo da disponibilidade de informações de depuração, a GDB sugerirá comandos para execução, como por exemplo:

```
Falta de debuginfos separados, uso: dnf debuginfo-instalar bc-1.07.1-5.el8.x86_64
```

Para mais detalhes sobre este processo, ver Seção 7.4, "Obtendo pacotes de debuginfo para uma aplicação ou biblioteca usando GDB".

- Exportar o arquivo principal para processamento posterior em outro lugar:

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

Substitua */path/to/file_for_export* pelo arquivo onde você quer colocar o despejo do núcleo.

10.5. MEMÓRIA DE PROCESSO DE DESPEJO COM GCORE

O fluxo de trabalho da depuração do núcleo permite a análise do estado do programa offline. Em alguns casos, é possível utilizar este fluxo de trabalho com um programa que ainda está em execução, como por exemplo quando é difícil acessar o ambiente com o processo. Você pode usar o comando **gcore** para descarregar a memória de qualquer processo enquanto ele ainda está em execução.

Pré-requisitos

- É preciso entender o que são lixões nucleares e como eles são criados.
- A GDB deve ser instalada no sistema.

Procedimento

1. Descubra a identificação do processo (*pid*). Use ferramentas como **ps**, **pgrep**, e **top**:

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

2. Despeje a memória deste processo:

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

Isto cria um arquivo ***filename*** e despeja nele a memória do processo. Enquanto a memória está sendo despejada, a execução do processo é interrompida.

3. Após a conclusão do despejo do núcleo, o processo retoma a execução normal.

4. Criar um relatório SOS para fornecer informações adicionais sobre o sistema:

```
# sosreport
```

   Isto cria um arquivo de alcatrão contendo informações sobre seu sistema, tais como cópias de arquivos de configuração.

5. Transferir o arquivo executável do programa, a lixeira do núcleo e o relatório SOS para o computador onde será realizada a depuração.

6. Opcional: Remover o despejo do núcleo e o relatório SOS depois de transferi-los, para liberar espaço em disco.

Recursos adicionais

- Como obter um arquivo principal sem reiniciar um aplicativo?

10.6. MEMÓRIA DE PROCESSO PROTEGIDA CONTRA DUMPING COM GDB

Você pode marcar a memória dos processos como não devendo ser descartada. Isto pode economizar recursos e garantir segurança adicional quando a memória do processo contém dados sensíveis: por exemplo, em aplicações bancárias ou contábeis ou em máquinas virtuais inteiras. Tanto os despejos do núcleo do kernel (**kdump**) como os despejos manuais do núcleo (**gcore**, GDB) não despejam a memória marcada desta forma.

Em alguns casos, você deve despejar todo o conteúdo da memória do processo, independentemente destas proteções. Este procedimento mostra como fazer isto usando o depurador GDB.

Pré-requisitos

- É preciso entender o que são lixões centrais.
- A GDB deve ser instalada no sistema.
- A GDB já deve estar presa ao processo com memória protegida.

Procedimento

1. Defina a GDB para ignorar as configurações no arquivo **/proc/PID/coredump_filter**:

```
(gdb) definir o filtro de uso-corpo-corpo
```

2. Defina a GDB para ignorar a bandeira da página de memória **VM_DONTDUMP**:

```
(gdb) colocar os mapeamentos excluídos do lixão em
```

3. Despeje a memória:

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

   Substitua *core-file* pelo nome do arquivo onde você quer despejar a memória.

Recursos adicionais

- Depuração com a GDB - Como produzir um arquivo principal a partir de seu programa

CAPÍTULO 11. MUDANÇAS NA GDB QUEBRAS DE COMPATIBILIDADE

A versão do GDB fornecida no Red Hat Enterprise Linux 8 contém uma série de mudanças que quebram a compatibilidade, especialmente para casos em que a saída do GDB é lida diretamente do terminal. As seções seguintes fornecem mais detalhes sobre estas mudanças.

Não se recomenda a saída da GDB para análise. Preferir scripts usando o Python GDB API ou a Interface de Máquina GDB (MI).

O GDBserver agora começa inferiors com shell

Para permitir expansão e substituição de variáveis em argumentos de linha de comando inferior, o GDBserver agora inicia o inferior em uma concha, o mesmo que o GDB.

Para desativar o uso da casca:

- Ao usar o comando **target extended-remote** GDB, desabilite o shell com o comando **set startup-with-shell off**.
- Ao utilizar o comando **target remote** GDB, desabilite o shell com a opção **--no-startup-with-shell** do GDBserver.

Exemplo 11.1. Exemplo de expansão de conchas em inferiors remotos de GDB

Este exemplo mostra como a execução do comando **/bin/echo /*** através do GDBserver difere nas versões 7 e 8 do Red Hat Enterprise Linux:

- Sobre a 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 /*'
/*
```

- Sobre a 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 suporte removido

O suporte para depuração de programas Java compilados com o Compilador GNU para Java (**gcj**) foi removido.

Nova sintaxe para comandos de manutenção de dumping de símbolos

A sintaxe dos comandos de manutenção de dumping de símbolos agora inclui opções antes dos nomes dos arquivos. Como resultado, comandos que funcionavam com GDB no RHEL 7 não funcionam no RHEL 8.

Como exemplo, o seguinte comando não mais armazena símbolos em um arquivo, mas produz uma mensagem de erro:

```
(gdb) símbolos de impressão de manutenção /tmp/out main.c
```

A nova sintaxe para os comandos de manutenção do símbolo dumping é:

```
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]
```

Os números de rosca não são mais globais

Anteriormente, a GDB utilizava apenas a numeração global de linhas. A numeração foi estendida para ser exibida por inferior no formulário **inferior_num.thread_num**, tal como **2.1**. Como conseqüência, a numeração de segmentos na variável de conveniência **$_thread** e no atributo **InferiorThread.num** Python não são mais únicos entre os inferiores.

O GDB agora armazena um segundo ID de linha por linha, chamado de ID global de linha, que é o novo equivalente dos números de linha nos lançamentos anteriores. Para acessar o número global de thread, use a variável de conveniência **$_gthread** e o atributo **InferiorThread.global_num** Python.

Para compatibilidade com versões anteriores, as identificações de linha da Interface de Máquina (MI) sempre contêm as identificações globais.

Exemplo 11.2. Exemplo de mudanças no número de fios da GDB

No 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
```

No 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
```

A memória para conteúdos de valor pode ser limitada

Anteriormente, a GDB não limitava a quantidade de memória alocada para o conteúdo de valores. Como

conseqüência, a depuração de programas incorretos poderia fazer com que a GDB alocasse demasiada memória. A configuração **max-value-size** foi adicionada para permitir limitar a quantidade de memória alocada. O valor padrão deste limite é 64 KiB. Como resultado, o GDB no Red Hat Enterprise Linux 8 não exibirá valores muito grandes, mas informará que o valor é muito grande em seu lugar.

Como exemplo, a impressão de um valor definido como **char s[128*1024];** produz resultados diferentes:

- No Red Hat Enterprise Linux 7, **$1 = 'A' <repeats 131072 times>**
- No Red Hat Enterprise Linux 8, **value requires 131072 bytes, which is more than max-value-size**

A versão Sun do formato de facadas não é mais suportada

O suporte para a versão Sun do formato de arquivo de debug **stabs** foi removido. O formato **stabs** produzido pela GCC em RHEL com a opção **gcc -gstabs** ainda é suportado pela GDB.

Mudanças no manuseio do sistema

O **set sysroot *path*** especifica a raiz do sistema ao procurar por arquivos necessários para a depuração. Os nomes de diretórios fornecidos a este comando podem agora ser prefixados com a string **target:** para fazer o GDB ler as bibliotecas compartilhadas do sistema alvo (tanto local quanto remoto). O prefixo anteriormente disponível **remote:** é agora tratado como **target:**. Além disso, o valor padrão da raiz do sistema mudou de uma string vazia para **target:** para compatibilidade retroativa.

A raiz do sistema especificado é prefixada ao nome do arquivo do executável principal, quando o GDB inicia processos remotamente, ou quando ele se liga a processos já em execução (tanto locais quanto remotos). Isto significa que para processos remotos, o valor padrão **target:** faz com que a GDB sempre tente carregar as informações de depuração a partir do sistema remoto. Para evitar isto, execute o comando **set sysroot** antes do comando **target remote** para que os arquivos de símbolos locais sejam encontrados antes dos remotos.

HISTSIZE não controla mais o tamanho do histórico de comando da GDB

Anteriormente, a GDB utilizava a variável de ambiente **HISTSIZE** para determinar quanto tempo o histórico de comando deveria ser mantido. A GDB foi alterada para usar a variável de ambiente **GDBHISTSIZE** em seu lugar. Esta variável é específica apenas para a GDB. Os valores possíveis e seus efeitos são:

- um número positivo - use um histórico de comando deste tamanho,
- **-1** ou uma corda vazia - manter o histórico de todos os comandos,
- valores não-numéricos - ignorados.

Limitação de conclusão adicionada

O número máximo de candidatos considerados durante a conclusão pode agora ser limitado usando o comando **set max-completions**. Para mostrar o limite atual, execute o comando **show max-completions**. O valor padrão é 200. Este limite evita que a GDB gere listas de conclusão excessivamente grandes e se torne insensível.

Como exemplo, a saída após a entrada **p <tab><tab>** é:

- na RHEL 7 **Display all 29863 possibilities? (y or n)**
- no RHEL 8 **Display all 200 possibilities? (y or n)**

Modo de compatibilidade HP-UX XDB removido

A opção **-xdb** para o modo de compatibilidade HP-UX XDB foi removida da GDB.

Manuseio de sinais para roscas

Anteriormente, a GDB podia fornecer um sinal para a linha atual em vez da linha para a qual o sinal era realmente enviado. Este erro foi corrigido e agora a GDB sempre passa o sinal para a linha correta ao retomar a execução.

Além disso, o comando **signal** agora sempre entrega corretamente o sinal solicitado para a linha atual. Se o programa for interrompido por um sinal e o usuário trocar de linha, a GDB pede confirmação.

Modos de ponto de parada sempre fora de linha e auto-moldados

A configuração **breakpoint always-inserted** foi alterada. O valor **auto** e o comportamento correspondente foi removido. O valor padrão agora é **off**. Além disso, o valor **off** agora faz com que a GDB não remova os pontos de parada do alvo até que todos os threads parem.

comandos remotebaud não são mais suportados

Os comandos **set remotebaud** e **show remotebaud** não são mais suportados. Use os comandos **set serial baud** e **show serial baud** em seu lugar.

PARTE III. CONJUNTOS DE FERRAMENTAS ADICIONAIS PARA O DESENVOLVIMENTO

Além das ferramentas relacionadas ao desenvolvimento que estão disponíveis como parte do sistema operacional, os desenvolvedores podem instalar conjuntos de ferramentas adicionais no Red Hat Enterprise Linux. Estes conjuntos de ferramentas podem conter ferramentas para diferentes idiomas, cadeias de ferramentas alternativas, ou versões alternativas de ferramentas de sistema.

CAPÍTULO 12. USANDO O CONJUNTO DE FERRAMENTAS GCC

12.1. O QUE É GCC TOOLSET

O Red Hat Enterprise Linux 8 apresenta o GCC Toolset, um conjunto de ferramentas GCC, um fluxo de aplicações contendo versões mais atualizadas de ferramentas de desenvolvimento e análise de desempenho. O GCC Toolset é similar ao Red Hat Developer Toolset for RHEL 7.

O GCC Toolset está disponível como um Application Stream na forma de uma coleção de software no repositório **AppStream**. O GCC Toolset é totalmente suportado sob os Acordos de Nível de Assinatura do Red Hat Enterprise Linux, é funcionalmente completo, e destina-se ao uso em produção. As aplicações e bibliotecas fornecidas pelo GCC Toolset não substituem as versões do sistema Red Hat Enterprise Linux, não as substituem e não se tornam automaticamente escolhas default ou preferidas. Usando uma estrutura chamada coleções de software, um conjunto adicional de ferramentas de desenvolvimento é instalado no diretório /**opt**/ e é explicitamente habilitado pelo usuário sob demanda usando o utilitário **scl**. A menos que seja observado o contrário para ferramentas ou recursos específicos, o GCC Toolset está disponível para todas as arquiteturas suportadas pelo Red Hat Enterprise Linux.

12.2. INSTALANDO O CONJUNTO DE FERRAMENTAS GCC

A instalação do GCC Toolset em um sistema instala as principais ferramentas e todas as dependências necessárias. Note que algumas partes do conjunto de ferramentas não são instaladas por padrão e devem ser instaladas separadamente.

Procedimento

- Para instalar o GCC Toolset versão *N*:

```
# yum instalar gcc-toolset-N
```

12.3. INSTALAÇÃO DE PACOTES INDIVIDUAIS DO GCC TOOLSET

Para instalar somente certas ferramentas do GCC Toolset em vez de todo o conjunto de ferramentas, liste os pacotes disponíveis e instale os pacotes selecionados com a ferramenta de gerenciamento de pacotes **yum**. Este procedimento é útil também para pacotes que não são instalados por padrão com o conjunto de ferramentas.

Procedimento

1. Liste os pacotes disponíveis em GCC Toolset versão *N*:

```
Lista de yum disponível gcc-toolset-N-\*
```

2. Para instalar qualquer um destes pacotes:

```
# instalação do yum package_name
```

Substituir *package_name* por uma lista de pacotes a serem instalados, separados por espaço. Por exemplo, para instalar os pacotes **gcc-toolset-9-gdb-gdbserver** e **gcc-toolset-9-gdb-doc**:

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

12.4. DESINSTALANDO O CONJUNTO DE FERRAMENTAS GCC

Para remover o GCC Toolset de seu sistema, desinstale-o usando a ferramenta de gerenciamento de pacotes **yum**.

Procedimento

- Para desinstalar o GCC Toolset versão *N*:

```
# yum remove gcc-toolset-N{\i1}
```

12.5. EXECUTANDO UMA FERRAMENTA DO GCC TOOLSET

Para executar uma ferramenta do GCC Toolset, use o utilitário **scl**.

Procedimento

- Para executar uma ferramenta do GCC Toolset versão *N*:

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

12.6. EXECUTANDO UMA SESSÃO DE SHELL COM O GCC TOOLSET

GCC Toolset permite executar uma sessão shell onde as versões de ferramentas GCC Toolset são usadas em vez das versões de sistema destas ferramentas, sem usar explicitamente o comando **scl**. Isto é útil quando você precisa iniciar as ferramentas de forma interativa muitas vezes, como ao configurar ou testar uma configuração de desenvolvimento.

Procedimento

- Para executar uma sessão shell onde as versões de ferramentas da versão GCC Toolset *N* substituem as versões de sistema destas ferramentas:

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

12.7. INFORMAÇÕES RELACIONADAS

- Guia do Usuário do Red Hat Developer Toolset

CAPÍTULO 13. CONJUNTO DE FERRAMENTAS GCC 9

Este capítulo fornece informações específicas para o GCC Toolset versão 9 e as ferramentas contidas nesta versão.

13.1. FERRAMENTAS E VERSÕES FORNECIDAS PELO GCC TOOLSET 9

O GCC Toolset 9 fornece as seguintes ferramentas e versões:

Tabela 13.1. Versões de ferramentas no GCC Toolset 9

Nome	Versão	Descrição

Nome	Versão	Descrição
GCC	9.2.1	Um conjunto de compiladores para C, C, e F

Nome	Versão	Descrição
GDB	8.3	Um depurador de linha de comando para programas escritos em C,

Nome	Versão	Descrição
		n.
Valgrind	3.15.0	Uma estrutura de instrumentação e uma série de ferramentas p

Nome	Versão	Descrição
		operfil das aplicações afim de detectar erros de memória, identifica

Nome	Versão	Descrição
		r p r o b l g n a s d e g e r e n c i a m e n t o d e m e m ó r i a e r e l a t a r q u a l q u e r u s o d e a r g u m e

Nome	Versão	Descrição
		próprios sem chamadas ao sistema.
SystemTap	4.1	Uma ferramenta de rastreamento

Nome	Versão	Descrição
		e sondagem para monitorar as atividades de todo o sistema sem a necessidad

Nome	Versão	Descrição
		entar, recompilar, instalar e reiniciar.
Dyninst	10.1.0	Uma biblioteca para instrument

Nome	Versão	Descrição
		arcomexecutáveisdeespaçodousuáriodurantesuaexecu

Nome	Versão	Descrição
binutils	2.32	Contém um vinculador, montador e outras ferramentas para manipulação de arquivos binários e outras utilidades para inspecionar e

Nome	Versão	Descrição
		arquivos de objetos binários.
elfutils	0.176	Uma coleção de ferramentas binária

Nome	Versão	Descrição
		utilidades para inspecionar e manipular arquivos ELF.
dwz	0.12	Uma ferram

Nome	Versão	Descrição
		otimizar as informações de depuração DWARF contidas nas bibliot

Nome	Versão	Descrição
		ecas compartilhadas ELF e nos executáveis ELF por tamanho.

Nome	Versão	Descrição
fazer	4.2.1	Uma ferramenta de automação para o controle de dependência.

strace Nome	5. Versão	Descrição
		Uma ferramenta de depuração para monitorar as chamadas de sistema que u

Nome	Versão	Descrição
		mpr ogr ama usa e sinaliza que ele recebe.
ltrace	0.7.91	Uma ferramenta de depuração para

Nome	Versão	Descrição
		adas para bibliotecas dinâmicas que um programa faz. Elas também

Nome	Versão	Descrição
		Para rastrear as chamadas de sistema executadas por programas.

Nome	Versão	Descrição
annobin	9.08	Uma ferramenta de verificação de segurança de construção.

13.2. COMPATIBILIDADE C NO CONJUNTO DE FERRAMENTAS GCC 9

IMPORTANTE

As informações de compatibilidade aqui apresentadas aplicam-se somente ao GCC do GCC Toolset 9.

O compilador GCC no GCC Toolset pode usar as seguintes normas C:

C 14

Esta é a configuração padrão de linguagem **default** para o GCC Toolset 9, com extensões GNU, equivalente a usar explicitamente a opção **-std=gnu 14**.
O uso da versão em idioma C 14 é suportado quando todos os objetos C compilados com a respectiva bandeira foram construídos usando a versão 6 ou posterior do GCC.

C 11

Este padrão de linguagem está disponível no GCC Toolset 9.
O uso da versão em idioma C 11 é suportado quando todos os objetos C compilados com a respectiva bandeira foram construídos usando a versão 5 ou posterior do GCC.

C 98

Este padrão de linguagem está disponível no GCC Toolset 9. Binários, bibliotecas compartilhadas e objetos construídos usando este padrão podem ser livremente misturados independentemente de serem construídos com GCC do GCC Toolset, Red Hat Developer Toolset, e RHEL 5, 6, 7 e 8.

C 17, C 2a

Estes padrões linguísticos estão disponíveis no GCC Toolset 9 apenas como uma capacidade experimental, instável e sem suporte. Além disso, a compatibilidade de objetos, arquivos binários e bibliotecas construídas usando estes padrões não pode ser garantida.

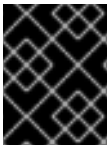
Todos os padrões de linguagem estão disponíveis tanto na variante compatível com o padrão quanto com as extensões GNU.

Ao misturar objetos construídos com o GCC Toolset com aqueles construídos com o conjunto de ferramentas RHEL (particularmente os arquivos **.o** ou **.a**), o conjunto de ferramentas GCC Toolset deve ser usado para qualquer ligação. Isto assegura que quaisquer novos recursos de biblioteca fornecidos apenas pelo conjunto de ferramentas GCC sejam resolvidos no momento do link.

13.3. ESPECIFICIDADES DO GCC NO CONJUNTO DE FERRAMENTAS GCC 9

Ligação estática de bibliotecas

Algumas características mais recentes da biblioteca estão estaticamente ligadas em aplicações construídas com o GCC Toolset para suportar a execução em múltiplas versões do Red Hat Enterprise Linux. Isto cria um risco adicional de segurança menor como errata padrão do Red Hat Enterprise Linux não altera este código. Se surgir a necessidade de os desenvolvedores reconstruírem suas aplicações devido a este risco, a Red Hat comunicará isto usando uma errata de segurança.



IMPORTANTE

Devido a este risco adicional de segurança, os desenvolvedores são fortemente aconselhados a não ligar estaticamente toda a sua aplicação pelas mesmas razões.

Especificar bibliotecas após arquivos objeto ao fazer a ligação

No GCC Toolset, as bibliotecas são ligadas usando scripts de linker que podem especificar alguns símbolos através de arquivos estáticos. Isto é necessário para garantir a compatibilidade com múltiplas versões do Red Hat Enterprise Linux. Entretanto, os scripts de linker usam os nomes dos respectivos arquivos objetos compartilhados. Como conseqüência, o linker usa regras de manuseio de símbolos diferentes das esperadas, e não reconhece símbolos requeridos pelos arquivos objeto quando a opção adicionando a biblioteca é especificada antes das opções especificando os arquivos objeto:

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

A utilização de uma biblioteca do GCC Toolset desta forma resulta na mensagem de erro do linker **undefined reference to symbol**. Para evitar este problema, siga a prática padrão de linker e especifique a opção adicionando a biblioteca após as opções especificando os arquivos objeto:

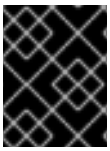
```
$ scl enable gcc-toolset-9 'gcc objfile.o -lsomelib'
```

Note que esta recomendação também se aplica ao usar a versão básica do Red Hat Enterprise Linux **GCC**.

13.4. ESPECIFICIDADES DOS BINUTILS NO GCC TOOLSET 9

Ligação estática de bibliotecas

Algumas características mais recentes da biblioteca estão estaticamente ligadas em aplicações construídas com o GCC Toolset para suportar a execução em múltiplas versões do Red Hat Enterprise Linux. Isto cria um risco adicional de segurança menor como errata padrão do Red Hat Enterprise Linux não altera este código. Se surgir a necessidade de os desenvolvedores reconstruírem suas aplicações devido a este risco, a Red Hat comunicará isto usando uma errata de segurança.



IMPORTANTE

Devido a este risco adicional de segurança, os desenvolvedores são fortemente aconselhados a não ligar estaticamente toda a sua aplicação pelas mesmas razões.

Especificar bibliotecas após arquivos objeto ao fazer a ligação

No GCC Toolset, as bibliotecas são ligadas usando scripts de linker que podem especificar alguns símbolos através de arquivos estáticos. Isto é necessário para garantir a compatibilidade com múltiplas versões do Red Hat Enterprise Linux. Entretanto, os scripts de linker usam os nomes dos respectivos arquivos objetos compartilhados. Como conseqüência, o linker usa regras de manuseio de símbolos diferentes das esperadas, e não reconhece símbolos requeridos pelos arquivos objeto quando a opção adicionando a biblioteca é especificada antes das opções especificando os arquivos objeto:

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

A utilização de uma biblioteca do GCC Toolset desta forma resulta na mensagem de erro do linker **undefined reference to symbol**. Para evitar este problema, siga a prática padrão de linker, e especifique a opção adicionando a biblioteca após as opções especificando os arquivos objeto:

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

Note que esta recomendação também se aplica ao usar a versão básica do Red Hat Enterprise Linux **binutils**.

CAPÍTULO 14. CONJUNTO DE FERRAMENTAS GCC 10

Este capítulo fornece informações específicas para o GCC Toolset versão 10 e as ferramentas contidas nesta versão.

14.1. FERRAMENTAS E VERSÕES FORNECIDAS PELO GCC TOOLSET 10

O GCC Toolset 10 fornece as seguintes ferramentas e versões:

Tabela 14.1. Versões de ferramentas no GCC Toolset 10

Nome	Versão	Descrição

Nome	Versão	Descrição
GCC	10.2.1	Um conjunto de compiladores suportáteis com suporte para C, C, e F

Nome	Versão	Descrição
GDB	9.2	Um depurador de linha de comando para programas escritos em C,

Nome	Versão	Descrição
		n.
Valgrind	3.16.0	Uma estrutura de instrumentação e uma série de ferramentas p

Nome	Versão	Descrição
		operfil das aplicações afim de detectar erros de memória, identifica

Nome	Versão	Descrição
		r problemas de gerenciamento de memória e relatar qualquer uso de argume

Nome	Versão	Descrição
		nt OS ri ó pri os em ch am ad as ao si st em a.
SystemTap	4.3	Um a fe rr am en ta de ra st re am en to

Nome	Versão	Descrição
		Compara monitora as atividades de todo o sistema a cada necessidad

Nome	Versão	Descrição
		entar, recompilar, instalar e reiniciar.
Dyninst	10.1.0	Uma biblioteca para instrument

Nome	Versão	Descrição
		Para arcomexecutáveisdeespaçodousuáriodurantesuaexecu

Nome	Versão	Descrição
binutils	2.30	Uma coleção de ferramentas binárias e outras utilidades para inspecionar e

Nome	Versão	Descrição
		arquivos de objetos binários.
elfutils	0.180	Uma coleção de ferramentas binária

Nome	Versão	Descrição
		utilidades para inspecionar e manipular os arquivos ELF.
dwz	0.12	Uma ferram

Nome	Versão	Descrição
		otimizar as informações de depuração DWARF contidas nas bibliot

Nome	Versão	Descrição
		e c a s ç h p a r t i l h a d a s E L F e n o s e x e c u t á v e i s E L F p o r t a m a n h o.

Nome	Versão	Descrição
fazer	4.2.1	Uma ferramenta de automação para controle de dependência.

strace Nome	5. Versão	Descrição
		Uma ferramenta de depuração para monitorar as chamadas de sistema usadas

Nome	Versão	Descrição
		mpr ogra ma usa esinaliza que elerecebe.
ltrace	0.7.91	Uma ferramenta de depuração para

Nome	Versão	Descrição
		madas para bibliotecas dinâmicas que um programa faz. Elas também

Nome	Versão	Descrição
		Depuração de chamadas de sistema executadas por programas.

Nome	Versão	Descrição
annobin	9.29	Uma ferramenta de verificação de segurança de construção.

14.2. COMPATIBILIDADE C NO CONJUNTO DE FERRAMENTAS GCC 10

IMPORTANTE

As informações de compatibilidade aqui apresentadas aplicam-se somente ao GCC do GCC Toolset 10.

O compilador GCC no GCC Toolset pode usar as seguintes normas C:

C 14

Esta é a configuração padrão de linguagem **default** para o GCC Toolset 10, com extensões GNU, equivalente a usar explicitamente a opção **-std=gnu 14**.
O uso da versão em idioma C 14 é suportado quando todos os objetos C compilados com a respectiva bandeira foram construídos usando a versão 6 ou posterior do GCC.

C 11

Este padrão de linguagem está disponível no GCC Toolset 10.
O uso da versão em idioma C 11 é suportado quando todos os objetos C compilados com a respectiva bandeira foram construídos usando a versão 5 ou posterior do GCC.

C 98

Este padrão de linguagem está disponível no GCC Toolset 10. Binários, bibliotecas compartilhadas e objetos construídos usando este padrão podem ser livremente misturados independentemente de serem construídos com GCC do GCC Toolset, Red Hat Developer Toolset, e RHEL 5, 6, 7 e 8.

C 17

Este padrão de linguagem está disponível no GCC Toolset 10.

C 20

Este padrão de linguagem está disponível no GCC Toolset 10 apenas como uma capacidade experimental, instável e não suportada. Além disso, a compatibilidade de objetos, arquivos binários e bibliotecas construídas usando este padrão não pode ser garantida.

Todos os padrões de linguagem estão disponíveis tanto na variante compatível com o padrão quanto com as extensões GNU.

Ao misturar objetos construídos com o GCC Toolset com aqueles construídos com o conjunto de ferramentas RHEL (particularmente os arquivos **.o** ou **.a**), o conjunto de ferramentas GCC Toolset deve ser usado para qualquer ligação. Isto assegura que quaisquer novos recursos de biblioteca fornecidos apenas pelo conjunto de ferramentas GCC sejam resolvidos no momento do link.

14.3. ESPECIFICIDADES DO GCC NO CONJUNTO DE FERRAMENTAS GCC 10

Ligação estática de bibliotecas

Algumas características mais recentes da biblioteca estão estaticamente ligadas em aplicações construídas com o GCC Toolset para suportar a execução em múltiplas versões do Red Hat Enterprise Linux. Isto cria um risco adicional de segurança menor como errata padrão do Red Hat Enterprise Linux não altera este código. Se surgir a necessidade de os desenvolvedores reconstruírem suas aplicações devido a este risco, a Red Hat comunicará isto usando uma errata de segurança.

IMPORTANTE

Devido a este risco adicional de segurança, os desenvolvedores são fortemente aconselhados a não ligar estaticamente toda a sua aplicação pelas mesmas razões.

Especificar bibliotecas após arquivos objeto ao fazer a ligação

No GCC Toolset, as bibliotecas são ligadas usando scripts de linker que podem especificar alguns símbolos através de arquivos estáticos. Isto é necessário para garantir a compatibilidade com múltiplas versões do Red Hat Enterprise Linux. Entretanto, os scripts de linker usam os nomes dos respectivos arquivos objetos compartilhados. Como conseqüência, o linker usa regras de manuseio de símbolos diferentes das esperadas, e não reconhece símbolos requeridos pelos arquivos objeto quando a opção adicionando a biblioteca é especificada antes das opções especificando os arquivos objeto:

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

A utilização de uma biblioteca do GCC Toolset desta forma resulta na mensagem de erro do linker **undefined reference to symbol**. Para evitar este problema, siga a prática padrão de linker e especifique a opção adicionando a biblioteca após as opções especificando os arquivos objeto:

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

Note que esta recomendação também se aplica ao usar a versão básica do Red Hat Enterprise Linux GCC.

14.4. ESPECIFICIDADES DOS BINUTILS NO GCC TOOLSET 10

Ligação estática de bibliotecas

Algumas características mais recentes da biblioteca estão estaticamente ligadas em aplicações construídas com o GCC Toolset para suportar a execução em múltiplas versões do Red Hat Enterprise Linux. Isto cria um risco adicional de segurança menor como errata padrão do Red Hat Enterprise Linux não altera este código. Se surgir a necessidade de os desenvolvedores reconstruírem suas aplicações devido a este risco, a Red Hat comunicará isto usando uma errata de segurança.

IMPORTANTE

Devido a este risco adicional de segurança, os desenvolvedores são fortemente aconselhados a não ligar estaticamente toda a sua aplicação pelas mesmas razões.

Especificar bibliotecas após arquivos objeto ao fazer a ligação

No GCC Toolset, as bibliotecas são ligadas usando scripts de linker que podem especificar alguns símbolos através de arquivos estáticos. Isto é necessário para garantir a compatibilidade com múltiplas versões do Red Hat Enterprise Linux. Entretanto, os scripts de linker usam os nomes dos respectivos arquivos objetos compartilhados. Como conseqüência, o linker usa regras de manuseio de símbolos diferentes das esperadas, e não reconhece símbolos requeridos pelos arquivos objeto quando a opção adicionando a biblioteca é especificada antes das opções especificando os arquivos objeto:

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

A utilização de uma biblioteca do GCC Toolset desta forma resulta na mensagem de erro do linker **undefined reference to symbol**. Para evitar este problema, siga a prática padrão de linker, e especifique a opção adicionando a biblioteca após as opções especificando os arquivos objeto:

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

Note que esta recomendação também se aplica ao usar a versão básica do Red Hat Enterprise Linux **binutils**.

CAPÍTULO 15. USANDO AS IMAGENS DOS RECIPIENTES DO CONJUNTO DE FERRAMENTAS GCC

Os componentes do GCC Toolset 10 estão disponíveis nas duas imagens do contêiner:

- Conjunto de ferramentas GCC 10 Toolchain
- GCC Toolset 10 Perftools

As imagens do conjunto de ferramentas GCC são baseadas na imagem base **rhel8** e estão disponíveis para todas as arquiteturas suportadas pela RHEL 8:

- Arquiteturas AMD e Intel de 64 bits
- A arquitetura ARM de 64 bits
- IBM Power Systems, Little Endian
- IBM Z

15.1. CONTEÚDO DAS IMAGENS DOS RECIPIENTES DO CONJUNTO DE FERRAMENTAS GCC

As versões de ferramentas fornecidas nas imagens do GCC Toolset 10 correspondem às versões de componentes do GCC Toolset 10.

O conjunto de ferramentas GCC 10 Conteúdo da cadeia de ferramentas

A imagem **rhel8/gcc-toolset-10-toolchain** fornece o compilador GCC, o depurador GDB, e outras ferramentas relacionadas ao desenvolvimento. A imagem do recipiente consiste nos seguintes componentes:

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

O conteúdo do GCC Toolset 10 Perftools

A imagem **rhel8/gcc-toolset-10-perftools** fornece uma série de ferramentas para depuração, monitoramento de desempenho e análise posterior das aplicações. A imagem do recipiente consiste nos seguintes componentes:

Componente	Pacote
Valgrind	gcc-toolset-10-valgrind

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

Recursos adicionais

- Para usar os componentes do GCC Toolset na RHEL 7, use o Red Hat Developer Toolset que fornece ferramentas de desenvolvimento similares para os usuários da RHEL 7
- Instruções sobre o uso das imagens do conjunto de ferramentas do Red Hat Developer Toolset no RHEL 7

15.2. ACESSO E FUNCIONAMENTO DAS IMAGENS DOS CONTÊINERES DO GCC TOOLSET

A seção seguinte descreve como acessar e executar as imagens dos containers do GCC Toolset.

Pré-requisitos

- Podman é instalado.

Procedimento

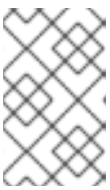
1. Acesse o Red Hat Container Registry usando suas credenciais do Portal do Cliente:

```
$ podman login registry.redhat.io
Username: username
Password: ********
```

2. Puxe uma imagem de contêiner que você precisa, executando um comando relevante como raiz:

```
# podman pull registry.redhat.io/rhel8/gcc-toolset-10-toolchain
```

```
# podman pull registry.redhat.io/rhel8/gcc-toolset-10-perftools
```



NOTA

Nas versões RHEL 8.1 e posteriores, você pode configurar seu sistema para trabalhar com containers como um usuário não-root. Para obter detalhes, veja Running containers como root ou rootless.

3. Opcional: Verifique se o puxão foi bem sucedido executando um comando que lista todas as imagens de contêineres em seu sistema local:

```
# imagens do podman
```

4. Execute um contêiner lançando uma concha de bash dentro de um contêiner:

```
# podman run -it image_name /bin/bash
```

A opção **-i** cria uma sessão interativa; sem esta opção, a casca se abre e sai instantaneamente.

A opção **-t** abre uma sessão terminal; sem esta opção você não pode digitar nada na concha.

Recursos adicionais

- Construção, execução e gerenciamento de recipientes Linux no RHEL 8
- Um artigo no blog Red Hat
- Inscrições no Registro de Contêineres Red Hat

15.3. EXEMPLO: USANDO A IMAGEM DO CONJUNTO DE FERRAMENTAS GCC 10 TOOLCHAIN CONTAINER

Este exemplo mostra como puxar e começar a usar a imagem do GCC Toolset 10 Toolchain container.

Pré-requisitos

- Podman é instalado.

Procedimento

1. Acesse o Red Hat Container Registry usando suas credenciais do Portal do Cliente:

```
$ podman login registry.redhat.io
Username: username
Password: ********
```

2. Puxe a imagem do recipiente como raiz:

```
# podman pull registry.redhat.io/rhel8/gcc-toolset-10-toolchain
```

3. Lançar a imagem do recipiente com uma concha interativa como raiz:

```
# podman run -it registry.redhat.io/rhel8/gcc-toolset-10-toolchain /bin/bash
```

4. Execute as ferramentas do GCC Toolset como esperado. Por exemplo, para verificar a versão do compilador **gcc**, execute:

```
bash-4.4$ gcc -v
...
gcc version 10.2.1 20200804 (Red Hat 10.2.1-2) (GCC)
```

5. Para listar todas as embalagens fornecidas no contêiner, execute:

```
bash-4,4$ rpm -qa
```

CAPÍTULO 16. CONJUNTOS DE FERRAMENTAS DE COMPILAÇÃO

A RHEL 8.0 fornece os seguintes conjuntos de ferramentas de compilação como Fluxos de Aplicação:

- LLVM Toolset 9.0.1, que fornece a estrutura de infra-estrutura do compilador LLVM, o compilador Clang para os idiomas C e C, o depurador LLDB, e ferramentas relacionadas para análise de código. Veja o guia Using LLVM Toolset (Usando o conjunto de ferramentas LLVM).
- Rust Toolset 1.41, que fornece o compilador da linguagem de programação Rust **rustc**, o **cargo** build tool and dependency manager, o plugin **cargo-vendor**, e as bibliotecas necessárias. Veja o guia Using Rust Toolset (Usando o Rust Toolset).
- Go Toolset 1.13, que fornece as ferramentas da linguagem de programação Go e bibliotecas. Go é conhecida alternativamente como **golang**. Veja o guia Using Go Toolset.

CAPÍTULO 17. O PROJETO ANNOBIN

O projeto Annobin é uma implementação do projeto de especificação da marca d'água. O projeto de especificação da marca d'água pretende adicionar marcadores a objetos de formato executável e vinculável (ELF) para determinar suas propriedades. O projeto Annobin consiste no plugin **annobin** e no programa **annockeck**.

O plugin **annobin** escaneia a linha de comando da Coleção Compiladora GNU (GCC), o estado de compilação e o processo de compilação, e gera as notas ELF. As notas ELF registram como o binário foi construído e fornecem informações para o programa **annocheck** para realizar verificações de endurecimento de segurança.

O verificador de endurecimento de segurança faz parte do programa **annocheck** e é ativado por padrão. Ele verifica os arquivos binários para determinar se o programa foi construído com as opções necessárias de endurecimento de segurança e compilado corretamente. **annocheck** é capaz de verificar recursivamente diretórios, arquivos e pacotes RPM para arquivos objetos ELF.

NOTA

Os arquivos devem estar no formato ELF. **annocheck** não lida com nenhum outro tipo de arquivo binário.

A seção seguinte descreve como fazê-lo:

- Use o plugin **annobin**
- Use o programa **annocheck**
- Remover notas redundantes **annobin**

17.1. USANDO O PLUGIN ANNOBIN

A seção seguinte descreve como fazê-lo:

- Habilite o plugin **annobin**
- Passe as opções para o plugin **annobin**

17.1.1. Habilitando o plugin do anobin

A seção seguinte descreve como habilitar o plug-in **annobin** via **gcc** e via **clang**.

Procedimento

- Para habilitar o plugin **annobin** com **gcc**, use:

```
$ gcc -fplugin=annobin
```

  - Se **gcc** não encontrar o plugin **annobin**, use:

```
$ gcc -iplugindir=/path/to/directory/containing/annobin/
```

    Substitua */path/to/directory/containing/annobin/* pelo caminho absoluto para o diretório que contém **annobin**.

- Para encontrar o diretório que contém o plugin **annobin**, use:

```
$ gcc --print-file-name=plugin
```

- Para habilitar o plugin **annobin** com **clang**, use:

```
$ clang -fplugin=/path/to/directory/containing/annobin/
```

Substitua */path/to/directory/containing/annobin/* pelo caminho absoluto para o diretório que contém **annobin**.

17.1.2. Passando opções para o plugin do anobin

A seção seguinte descreve como passar opções para o plugin **annobin** via **gcc** e via **clang**.

Procedimento

- Para passar opções para o plugin **annobin** com **gcc**, use:

```
$ gcc -fplugin=annobin -fplugin-arg-annobin-option file-name
```

Substituir *option* pelos argumentos da linha de comando **annobin** e substituir *file-name* pelo nome do arquivo.

Exemplo

- Para exibir detalhes adicionais sobre o que está fazendo **annobin**, use:

```
$ gcc -fplugin=annobin -fplugin-arg-annobin-verbose file-name
```

Substituir *file-name* pelo nome do arquivo.

- Para passar opções para o plugin **annobin** com **clang**, use:

```
$ clang -fplugin=/path/to/directory/containing/annobin/ -Xclang -plugin-arg-annobin -Xclang option file-name
```

Substituir *option* pelos argumentos da linha de comando **annobin** e substituir */path/to/directory/containing/annobin/* pelo caminho absoluto para o diretório que contém **annobin**.

Exemplo

- Para exibir detalhes adicionais sobre o que está fazendo **annobin**, use:

```
$ clang -fplugin=/usr/lib64/clang/10/lib/annobin.so -Xclang -plugin-arg-annobin -Xclang verbose file-name
```

Substituir *file-name* pelo nome do arquivo.

17.2. USANDO O PROGRAMA ANNOCHECK

A seção seguinte descreve como usar **annocheck** para examinar:

- Arquivos
- Diretórios
- Pacotes de RPM
- **annocheck** ferramentas extras

NOTA

annocheck escaneia recursivamente diretórios, arquivos e pacotes de RPM para arquivos objetos ELF. Os arquivos têm que estar no formato ELF. **annocheck** não lida com nenhum outro tipo de arquivo binário.

17.2.1. Usando o anocheck para examinar os arquivos

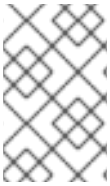
A seção seguinte descreve como examinar os arquivos ELF usando **annocheck**.

Procedimento

- Para examinar um arquivo, use:

```
$ annocheck file-name
```

Substituir *file-name* pelo nome de um arquivo.



NOTA

Os arquivos devem estar no formato ELF. **annocheck** não lida com nenhum outro tipo de arquivo binário. **annocheck** processa bibliotecas estáticas que contêm arquivos objetos ELF.

Informações adicionais

- Para mais informações sobre **annocheck** e possíveis opções de linha de comando, consulte a página de manual **annocheck**.

17.2.2. Usando o Annocheck para examinar diretórios

A seção seguinte descreve como examinar os arquivos ELF em um diretório usando **annocheck**.

Procedimento

- Para escanear um diretório, use:

```
$ annocheck directory-name
```

Substitua *directory-name* pelo nome de um diretório. **annocheck** examina automaticamente o conteúdo do diretório, seus subdiretórios e quaisquer arquivos e pacotes RPM dentro do diretório.

NOTA

annocheck procura apenas por arquivos ELF. Outros tipos de arquivos são ignorados.

Informações adicionais

- Para mais informações sobre **annocheck** e possíveis opções de linha de comando, consulte a página de manual **annocheck**.

17.2.3. Usando o annocheck para examinar pacotes de RPM

A seção seguinte descreve como examinar os arquivos ELF em um pacote RPM usando **annocheck**.

Procedimento

- Para escanear um pacote RPM, use:

```
$ annocheck rpm-package-name
```

  Substitua *rpm-package-name* pelo nome de um pacote RPM. **annocheck** escaneia recursivamente todos os arquivos ELF dentro do pacote RPM.

NOTA

annocheck procura apenas por arquivos ELF. Outros tipos de arquivos são ignorados.

- Para escanear um pacote RPM com informações de depuração RPM fornecidas, use:

```
$ annocheck rpm-package-name --debug-rpm debuginfo-rpm
```

  Substituir *rpm-package-name* pelo nome de um pacote RPM, e *debuginfo-rpm* pelo nome de um RPM debug info associado ao RPM binário.

Informações adicionais

- Para mais informações sobre **annocheck** e possíveis opções de linha de comando, consulte a página de manual **annocheck**.

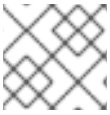
17.2.4. Usando ferramentas extras do Annocheck

annocheck inclui múltiplas ferramentas para examinar arquivos binários. Você pode habilitar estas ferramentas com as opções de linha de comando.

A seção seguinte descreve como habilitar o:

- ferramenta**built-by**
- ferramenta**notes**
- ferramenta**section-size**

Você pode habilitar várias ferramentas ao mesmo tempo.



NOTA

O verificador de endurecimento é ativado por padrão.

17.2.4.1. Habilitando a ferramenta built-by

Você pode usar a ferramenta **annocheck built-by** para encontrar o nome do compilador que construiu o arquivo binário.

Procedimento

- Para ativar a ferramenta **built-by**, use:

```
$ annocheck --enable-built-by
```

Informações adicionais

- Para mais informações sobre a ferramenta **built-by**, consulte a opção de linha de comando **--help**.

17.2.4.2. Habilitando a ferramenta notes

Você pode usar a ferramenta **annocheck notes** para exibir as notas armazenadas dentro de um arquivo binário criado pelo plugin **annobin**.

Procedimento

- Para ativar a ferramenta **notes**, use:

```
$ annocheck --enable-notes
```

  As notas são exibidas em uma seqüência ordenada pelo intervalo de endereços.

Informações adicionais

- Para mais informações sobre a ferramenta **notes**, consulte a opção de linha de comando **--help**.

17.2.4.3. Habilitando a ferramenta section-size

Você pode usar a ferramenta **annocheck section-size** para exibir o tamanho das seções nomeadas.

Procedimento

- Para ativar a ferramenta **section-size**, use:

```
$ annocheck --section-size=name
```

  Substitua *name* pelo nome da seção nomeada. A saída é restrita a seções específicas. Um resultado cumulativo é produzido no final.

Informações adicionais

- Para mais informações sobre a ferramenta **section-size**, consulte a opção de linha de comando **--help**.

17.2.4.4. Noções básicas sobre checagem de endurecimento

O verificador de endurecimento é ativado por padrão. Você pode desabilitar o verificador de endurecimento com a opção de linha de comando **--disable-hardened**.

17.2.4.4.1. Opções de verificadores de endurecimento

O programa **annocheck** verifica as seguintes opções:

- A ligação preguiçosa é desativada usando a opção **-z now** linker.
- O programa não tem uma pilha em uma região executável de memória.
- As relocalizações para a tabela GOT estão definidas para leitura apenas.
- Nenhum segmento do programa tem todos os três bits de permissão de leitura, escrita e execução definidos.
- Não há realocações contra códigos executáveis.
- As informações do runpath para localização de bibliotecas compartilhadas em tempo de execução incluem apenas diretórios enraizados em /usr.
- O programa foi compilado com notas de **annobin** habilitadas.
- O programa foi compilado com a opção **-fstack-protector-strong** ativada.
- O programa foi compilado com **-D_FORTIFY_SOURCE=2**.
- O programa foi compilado com **-D_GLIBCXX_ASSERTIONS**.
- O programa foi compilado com **-fexceptions** habilitado.
- O programa foi compilado com **-fstack-clash-protection** habilitado.
- O programa foi compilado em **-O2** ou superior.
- O programa não tem nenhuma relocalização realizada em um programa gravável.
- Os executáveis dinâmicos têm um segmento dinâmico.
- Bibliotecas compartilhadas foram compiladas com **-fPIC** ou **-fPIE**.
- Os executáveis dinâmicos foram compilados com **-fPIE** e vinculados com **-pie**.
- Se disponível, foi utilizada a opção **-fcf-protection=full**.
- Se disponível, foi utilizada a opção **-mbranch-protection**.
- Se disponível, foi utilizada a opção **-mstackrealign**.

17.2.4.4.2. Desabilitando o verificador de endurecimento

A seção seguinte descreve como desativar o verificador de endurecimento.

Procedimento

- Para escanear as notas em um arquivo sem o verificador de endurecimento, use:

```
$ annocheck --enable-notes --disable-hardened file-name
```

  Substituir *file-name* pelo nome de um arquivo.

17.3. REMOÇÃO DE NOTAS ANOBINAS REDUNDANTES

O uso do site **annobin** aumenta o tamanho dos binários. Para reduzir o tamanho dos binários compilados com **annobin**, você pode remover notas redundantes **annobin**. Para remover as notas redundantes **annobin** utilize o programa **objcopy**, que faz parte do pacote **binutils**.

Procedimento

- Para remover as notas redundantes **annobin**, use:

```
$ objcopy --merge-notes file-name
```

  Substituir *file-name* pelo nome do arquivo.

PARTE IV. TÓPICOS COMPLEMENTARES

CAPÍTULO 18. MUDANÇAS DE COMPATIBILIDADE EM COMPILADORES E FERRAMENTAS DE DESENVOLVIMENTO

librtkaio removido

Com esta atualização, a biblioteca **librtkaio** foi removida. Esta biblioteca forneceu acesso de E/S assíncrona em tempo real de alto desempenho para alguns arquivos, que foi baseada no suporte de E/S assíncrona do kernel Linux (KAIO).

Como resultado da remoção:

- As aplicações usando o método **LD_PRELOAD** para carregar **librtkaio** exibem um aviso sobre uma biblioteca ausente, carregam a biblioteca **librt** em seu lugar e funcionam corretamente.
- Aplicações que utilizam o método **LD_LIBRARY_PATH** para carregar **librtkaio** carregam a biblioteca **librt** e funcionam corretamente, sem qualquer aviso.
- Aplicações usando a chamada do sistema **dlopen()** para acessar **librtkaio** diretamente carregar a biblioteca **librt**.

Os usuários do site **librtkaio** têm as seguintes opções:

- Usar o mecanismo de emergência descrito acima, sem nenhuma alteração em suas aplicações.
- Alterem o código de suas aplicações para usar a biblioteca **librt**, que oferece uma API compatível com o POSIX.
- Alterem o código de suas aplicações para usar a biblioteca **libaio**, que oferece uma API compatível.

Tanto **librt** quanto **libaio** podem fornecer características e desempenho comparáveis sob condições específicas.

Note que o pacote **libaio** tem o nível 2 de compatibilidade da Red Hat, enquanto **librtk** e o removido **librtkaio** nível 1.

Para mais detalhes, veja https://fedoraproject.org/wiki/Changes/GLIBC223_librtkaio_removal

Interfaces Sun RPC e NIS removidas de glibc

A biblioteca **glibc** não fornece mais interfaces Sun RPC e NIS para novas aplicações. Estas interfaces agora estão disponíveis apenas para a execução de aplicações legadas. Os desenvolvedores devem mudar suas aplicações para usar a biblioteca **libtirpc** em vez da Sun RPC e **libnsl2** em vez da NIS. As aplicações podem se beneficiar do suporte a IPv6 nas bibliotecas de substituição.

As bibliotecas nosegneg de 32 bits Xen foram removidas

Anteriormente, os pacotes **glibc** i686 continham um build alternativo **glibc**, que evitava o uso do registro do segmento de descritores de linha com offsets negativos (**nosegneg**). Esta construção alternativa só foi utilizada na versão 32-bit do hypervisor do Projeto Xen sem suporte de virtualização de hardware, como uma otimização para reduzir o custo de paravirtualização total. Estas construções alternativas não são mais utilizadas e foram removidas.

make novo operador != causa uma interpretação diferente de certas sintaxes de makefile existentes

O operador de atribuição de shell **!=** foi adicionado ao GNU **make** como uma alternativa à função **$(shell ...)** para aumentar a compatibilidade com os makefiles BSD. Como conseqüência, variáveis com nome terminando em ponto de exclamação e imediatamente seguidas por atribuição como

variable!=value são agora interpretadas como a atribuição de shell. Para restaurar o comportamento anterior, adicionar um espaço após o ponto de exclamação, tal como **variable! =value**.

Para mais detalhes e diferenças entre o operador e a função, consulte o manual do GNU **make**.

Biblioteca Valgrind para suporte de depuração MPI removida

A biblioteca de embalagens **libmpiwrap.so** para **Valgrind**, fornecida pelo pacote **valgrind-openmpi**, foi removida. Esta biblioteca habilitou **Valgrind** para depurar programas usando a Interface de Passagem de Mensagens (MPI). Esta biblioteca era específica para a versão de implementação Open MPI nas versões anteriores do Red Hat Enterprise Linux.

Os usuários de **libmpiwrap.so** são encorajados a construir sua própria versão a partir de fontes a montante específicas para sua implementação e versão de MPI. Forneça estas bibliotecas personalizadas para **Valgrind** usando a técnica **LD_PRELOAD**.

Cabeçalhos de desenvolvimento e bibliotecas estáticas removidas de **valgrind-devel**

Anteriormente, o sub-pacote **valgrind-devel** era usado para incluir arquivos de desenvolvimento para o desenvolvimento de ferramentas de valgrind personalizadas. Esta atualização remove estes arquivos porque eles não têm uma API garantida, têm que ser ligados estaticamente e não têm suporte. O pacote **valgrind-devel** ainda contém os arquivos de desenvolvimento para programas e arquivos de cabeçalho com reconhecimento de valor, como **valgrind.h**, **callgrind.h**, **drd.h**, **helgrind.h**, e **memcheck.h**, que são estáveis e bem suportados.

CAPÍTULO 19. OPÇÕES PARA EXECUTAR UMA APLICAÇÃO RHEL 6 OU 7 NO RHEL 8

Para executar uma aplicação Red Hat Enterprise Linux 6 ou 7 no Red Hat Enterprise Linux 8, um espectro de opções está disponível. Um administrador de sistemas precisa de orientação detalhada do desenvolvedor da aplicação. A lista a seguir descreve as opções, considerações e recursos fornecidos pela Red Hat.

Executar a aplicação em uma máquina virtual com um sistema operacional convidado RHEL correspondente

Os custos de recursos são elevados para esta opção, mas o ambiente está próximo das exigências da aplicação, e esta abordagem não requer muitas considerações adicionais. Esta é a opção atualmente recomendada.

Executar a aplicação em um recipiente com base na respectiva versão RHEL

Os custos de recursos são mais baixos do que nos casos anteriores, enquanto os requisitos de configuração são mais rigorosos. Para detalhes sobre a relação entre hosts de contêineres e espaços de usuários convidados, veja a Matriz de Compatibilidade de Contêineres do Red Hat Enterprise Linux.

Execute o aplicativo nativamente no RHEL 8

Esta opção oferece os menores custos de recursos, mas também os requisitos mais rigorosos. O desenvolvedor da aplicação deve determinar uma configuração correta do sistema RHEL 8. Os recursos a seguir podem ajudar o desenvolvedor nesta tarefa:

- Red Hat Enterprise Linux 8: Guia de Compatibilidade de Aplicações
- Red Hat Enterprise Linux 7: Guia de Compatibilidade de Aplicações
- Notas de lançamento do Red Hat Enterprise Linux 8.0
- Considerações ao adotar a RHEL 8

Observe que esta lista não é um conjunto completo de recursos necessários para determinar a compatibilidade da aplicação. Estes são apenas pontos de partida com listas de alterações incompatíveis conhecidas e políticas da Red Hat relacionadas à compatibilidade.

Além disso, o *What is Kernel Application Binary Interface (kABI)?* O artigo Knowledge Centered Support contém informações relevantes para o núcleo e a compatibilidade.