gdb - Introdução

Adriano Cruz
adriano@nce.ufrj.br

Instituto de Matemática
Departamento de Ciência da Computação
UFRJ

3 de fevereiro de 2013

Section Summary

Bibliografia

1. http://sourceware.org/gdb/current/onlinedocs/gdb/

Objetivos

gdb

O objetivo deste capítulo é apresentar uma introdução à ferramenta de depuração de programas chamada gdb.

O que é?

- GNU debugger - gdb

O que é?

- GNU debugger - gdb
- Um depurador para diversas linguagens, incluindo C e C++.

O que é?

- GNU debugger - gdb
- Um depurador para diversas linguagens, incluindo C e C++.
- Um depurador permite que inspecionemos o que o programa está fazendo durante a sua execução.

O que é?

- GNU debugger - `gdb`
- Um depurador para diversas linguagens, incluindo `C` e `C++`.
- Um depurador permite que inspecionemos o que o programa está fazendo durante a sua execução.
- Erros, tais como, `segmentation faults`, podem ser mais fáceis de serem corrigidos com a ajuda do gdb.

O que é?

- Normally, you would compile a program like:

O que é?

- Normally, you would compile a program like:

```
gcc [flags] <source files> -o <output file>
```

O que é?

- Normally, you would compile a program like:

```
gcc [flags] <source files> -o <output file>
```

- Por exemplo:

O que é?

- Normally, you would compile a program like:

```
gcc [flags] <source files> -o <output file>
```

- Por exemplo:

```
gcc -Wall -Werror -ansi -pedantic-errors prog1.c -o prog1
```

O que é?

- Normally, you would compile a program like:

```
gcc [flags] <source files> -o <output file>
```

- Por exemplo:

```
gcc -Wall -Werror -ansi -pedantic-errors prog1.c -o prog1
```

- Para usar o gdb precisamos adicionar a opção -g para adicionar o suporte que gdb precisa.

O que é?

- Normally, you would compile a program like:

```
gcc [flags] <source files> -o <output file>
```

- Por exemplo:

```
gcc -Wall -Werror -ansi -pedantic-errors prog1.c -o prog1
```

- Para usar o gdb precisamos adicionar a opção -g para adicionar o suporte que gdb precisa.

```
gcc [other flags] -g <source files> -o <output file>
```

O que é?

- Normally, you would compile a program like:

```
gcc [flags] <source files> -o <output file>
```

- Por exemplo:

```
gcc -Wall -Werror -ansi -pedantic-errors prog1.c -o prog1
```

- Para usar o gdb precisamos adicionar a opção -g para adicionar o suporte que gdb precisa.

```
gcc [other flags] -g <source files> -o <output file>
```

- Por exemplo:

O que é?

- Normally, you would compile a program like:

```
gcc [flags] <source files> -o <output file>
```

- Por exemplo:

```
gcc -Wall -Werror -ansi -pedantic-errors prog1.c -o prog1
```

- Para usar o gdb precisamos adicionar a opção -g para adicionar o suporte que gdb precisa.

```
gcc [other flags] -g <source files> -o <output file>
```

- Por exemplo:

```
gcc -Wall -Werror -ansi -pedantic-errors -g prog1.c -o prog1
```

Como usar?

- Para rodar usar

Como usar?

- Para rodar usar

```
gdb ou gdb prog1
```

Como usar?

- Para rodar usar

```
gdb ou gdb prog1
```

- Irá aparecer um indicador (*prompt*) que como este:

Como usar?

- Para rodar usar

```
gdb ou gdb prog1
```

- Irá aparecer um indicador (*prompt*) que como este:

```
(gdb)
```

Como usar?

- Para rodar usar

```
gdb ou gdb prog1
```

- Irá aparecer um indicador (*prompt*) que como este:

```
(gdb)
```

- Se foi usada a primeira opção então é necessário agora dar o nome do programa:

Como usar?

- Para rodar usar

```
gdb ou gdb prog1
```

- Irá aparecer um indicador (*prompt*) que como este:

```
(gdb)
```

- Se foi usada a primeira opção então é necessário agora dar o nome do programa:

```
(gdb) file prog1
```

O shell

- gdb tem um shell interativo.

O shell

- gdb tem um shell interativo.
- O shell pode lembrar-se de comandos anteriores (teclas de setas), automaticamente completar comandos com a tecla TAB etc

O shell

- gdb tem um shell interativo.
- O shell pode lembrar-se de comandos anteriores (teclas de setas), automaticamente completar comandos com a tecla TAB etc

Dica

Se precisar de informações acerca de um comando usar o comando `help` com ou sem um argumento.

O shell

- gdb tem um shell interativo.
- O shell pode lembrar-se de comandos anteriores (teclas de setas), automaticamente completar comandos com a tecla TAB etc

Dica

Se precisar de informações acerca de um comando usar o comando `help` com ou sem um argumento.

```
(gdb) help [command]
```

O shell

- gdb tem um shell interativo.
- O shell pode lembrar-se de comandos anteriores (teclas de setas), automaticamente completar comandos com a tecla TAB etc

Dica

Se precisar de informações acerca de um comando usar o comando `help` com ou sem um argumento.

```
(gdb) help [command]
```

```
(gdb) help next
```

Executando o programa

- Para rodar o programa use:

Executando o programa

- Para rodar o programa use:

```
(gdb) run
```

Executando o programa

- Para rodar o programa use:

```
(gdb) run
```

- Se o programa não tem nenhum erro ele irá executar normalmente.

Executando o programa

- Para rodar o programa use:

```
(gdb) run
```

- Se o programa não tem nenhum erro ele irá executar normalmente.
- Se o programa tem problemas o gdb irá imprimir informações sobre o erro, como por exemplo: a linha onde o erro ocorreu, e os parâmetros para a função que causou o erro:

Executando o programa

- Para rodar o programa use:

```
(gdb) run
```

- Se o programa não tem nenhum erro ele irá executar normalmente.
- Se o programa tem problemas o gdb irá imprimir informações sobre o erro, como por exemplo: a linha onde o erro ocorreu, e os parâmetros para a função que causou o erro:

```
Program received signal SIGSEGV, Segmentation fault.
0x00000000004005dc in soma (v=0x7ffff7fa2010, n=50000) at
seg.c:6
6 for (i = 0; i < 2*n; i++) s += *(v+i);
```

O programa culpado

```c
#include <stdio.h>
#include <stdlib.h>
int soma(int *v, int n) {
    int i, s = 0;
    for (i = 0; i < 2*n; i++) s += *(v+i);
    return s;
}
int main (void) {
    int n, i, s;
    int *v;

    scanf("%d", &n);
    v = (int *) malloc (n * sizeof(int));
    for (i = 0; i < n; i++) v[i] = i+1;
    s = soma(v, n);
    printf("%d\n", s);
    return 0;
}
```

Não funciona!

E se o programa rodou, mas não está funcionando?

Não funciona!

E se o programa rodou, mas não está funcionando?

Idéia Básica

Precisamos de um modo de rodar o programa passo a passo, observar variáveis, rodar até um determinado ponto etc

Breakpoints

- Breakpoints podem ser usados para parar o programa em um ponto determinado.

Breakpoints

- Breakpoints podem ser usados para parar o programa em um ponto determinado.
- Pontos podem ser números de linha, nomes de funções etc.

Breakpoints

- Breakpoints podem ser usados para parar o programa em um ponto determinado.
- Pontos podem ser números de linha, nomes de funções etc.
- A forma mais simples é o comando `break`.

Breakpoints

- Breakpoints podem ser usados para parar o programa em um ponto determinado.
- Pontos podem ser números de linha, nomes de funções etc.
- A forma mais simples é o comando `break`.
- Por exemplo,

Breakpoints

- Breakpoints podem ser usados para parar o programa em um ponto determinado.
- Pontos podem ser números de linha, nomes de funções etc.
- A forma mais simples é o comando `break`.
- Por exemplo,

```
(gdb) break prog.c:6
```

Breakpoints

- Breakpoints podem ser usados para parar o programa em um ponto determinado.
- Pontos podem ser números de linha, nomes de funções etc.
- A forma mais simples é o comando `break`.
- Por exemplo,

```
(gdb) break prog.c:6
```

- Este comando define um ponto de parada na linha 6 do arquivo `prog.c`

Breakpoints

- Breakpoints podem ser usados para parar o programa em um ponto determinado.
- Pontos podem ser números de linha, nomes de funções etc.
- A forma mais simples é o comando `break`.
- Por exemplo,

```
(gdb) break prog.c:6
```

- Este comando define um ponto de parada na linha 6 do arquivo `prog.c`
- Se o programa passar por este ponto ele irá parar e esperar por outro comando.

Breakpoints

- Breakpoints podem ser usados para parar o programa em um ponto determinado.
- Pontos podem ser números de linha, nomes de funções etc.
- A forma mais simples é o comando `break`.
- Por exemplo,

```
(gdb) break prog.c:6
```

- Este comando define um ponto de parada na linha 6 do arquivo `prog.c`
- Se o programa passar por este ponto ele irá parar e esperar por outro comando.

Dica

Podem ser definidos tantos pontos de parada quanto forem necessários.

Mais breakpoints

- É possível usar o nome da função para definir o ponto de parada.

Mais breakpoints

- É possível usar o nome da função para definir o ponto de parada.
- Se o seu programa possui uma função **int** `soma(`**int** `*v,` **int** `n)` é possível usar o seguinte comando:

Mais breakpoints

- É possível usar o nome da função para definir o ponto de parada.
- Se o seu programa possui uma função **int** `soma(`**int** `*v,` **int** `n)` é possível usar o seguinte comando:

```
(gdb) break soma
```

Mais breakpoints

- É possível usar o nome da função para definir o ponto de parada.
- Se o seu programa possui uma função **int** `soma(`**int** `*v,` **int** `n)` é possível usar o seguinte comando:

```
(gdb) break soma
```

- Este comando define um ponto de parada no início da função `soma.c`

Mais breakpoints

- É possível usar o nome da função para definir o ponto de parada.
- Se o seu programa possui uma função **int** `soma(`**int** `*v,` **int** `n)` é possível usar o seguinte comando:

```
(gdb) break soma
```

- Este comando define um ponto de parada no início da função `soma.c`
- Se o programa passar por este ponto ele irá parar e esperar por outro comando.

Mais breakpoints

- É possível usar o nome da função para definir o ponto de parada.
- Se o seu programa possui uma função **int** `soma(`**int** `*v,` **int** `n)` é possível usar o seguinte comando:

```
(gdb) break soma
```

- Este comando define um ponto de parada no início da função `soma.c`
- Se o programa passar por este ponto ele irá parar e esperar por outro comando.

Dica

Podem ser definidos tantos pontos de parada quanto forem necessários.

E agora?

- Uma vez definido um breakpoint é possível usar o comando

E agora?

- Uma vez definido um breakpoint é possível usar o comando

```
run
```

E agora?

- Uma vez definido um breakpoint é possível usar o comando

```
run
```

- O programa irá reiniciar e parar no breakpoint se ele passar por lá.

E agora?

- Uma vez definido um breakpoint é possível usar o comando

```
run
```

- O programa irá reiniciar e parar no breakpoint se ele passar por lá.
- Também é possível continuar até o próximo breakpoint usando o comando

E agora?

- Uma vez definido um breakpoint é possível usar o comando

```
run
```

- O programa irá reiniciar e parar no breakpoint se ele passar por lá.
- Também é possível continuar até o próximo breakpoint usando o comando

```
continue
```

E agora?

- Uma vez definido um breakpoint é possível usar o comando

```
run
```

- O programa irá reiniciar e parar no breakpoint se ele passar por lá.
- Também é possível continuar até o próximo breakpoint usando o comando

```
continue
```

- Outra alternativa é executar o programa instrução a instrução usando o comando

E agora?

- Uma vez definido um breakpoint é possível usar o comando

```
run
```

- O programa irá reiniciar e parar no breakpoint se ele passar por lá.
- Também é possível continuar até o próximo breakpoint usando o comando

```
continue
```

- Outra alternativa é executar o programa instrução a instrução usando o comando

```
step
```

Mais passo a passo

- Parecido com `step`, há o comando

Mais passo a passo

- Parecido com `step`, há o comando

next

Mais passo a passo

- Parecido com `step`, há o comando

next

- Este comando também anda passo a passo, com a diferença que ele não entra na função. Ele a trata como uma única instrução.

Mais passo a passo

- Parecido com `step`, há o comando

next

- Este comando também anda passo a passo, com a diferença que ele não entra na função. Ele a trata como uma única instrução.

Dica

Digitando `step` ou `next` várias vezes pode ser cansativo. Se você digitar ENTER, gdb irá repetir o último comando digitado.

E as variáveis?

- Até agora somente vimos como parar um programa.

E as variáveis?

- Até agora somente vimos como parar um programa.
- E se for necessário ver valores de variáveis?

E as variáveis?

- Até agora somente vimos como parar um programa.
- E se for necessário ver valores de variáveis?
- O comando `print` imprime o valor da variável especificada.

E as variáveis?

- Até agora somente vimos como parar um programa.
- E se for necessário ver valores de variáveis?
- O comando `print` imprime o valor da variável especificada.

```
(gdb) print my var
(gdb) print/x my var
```

E as variáveis?

- Até agora somente vimos como parar um programa.
- E se for necessário ver valores de variáveis?
- O comando `print` imprime o valor da variável especificada.

```
(gdb) print my var
(gdb) print/x my var
```

- O comando `print/x` imprime o valor da variável especificada em hexa.

Pontos de observação

- Breakpoints interrompem o programa em uma linha ou função específica.

Pontos de observação

- Breakpoints interrompem o programa em uma linha ou função específica.
- watchpoints atuam em variáveis.

Pontos de observação

- Breakpoints interrompem o programa em uma linha ou função específica.
- watchpoints atuam em variáveis.
- Eles param o programa sempre que uma variável é modificada.

Pontos de observação

- Breakpoints interrompem o programa em uma linha ou função específica.
- watchpoints atuam em variáveis.
- Eles param o programa sempre que uma variável é modificada.
- Por exemplo, o comando:

Pontos de observação

- Breakpoints interrompem o programa em uma linha ou função específica.
- watchpoints atuam em variáveis.
- Eles param o programa sempre que uma variável é modificada.
- Por exemplo, o comando:

```
(gdb) watch var
```

Pontos de observação

- Breakpoints interrompem o programa em uma linha ou função específica.
- watchpoints atuam em variáveis.
- Eles param o programa sempre que uma variável é modificada.
- Por exemplo, o comando:

```
(gdb) watch var
```

- Após este comando, sempre que o valor da variável `var` é modificado, o programa irá imprimir o valor velho e o novo.

Pontos de observação

- Breakpoints interrompem o programa em uma linha ou função específica.
- watchpoints atuam em variáveis.
- Eles param o programa sempre que uma variável é modificada.
- Por exemplo, o comando:

```
(gdb) watch var
```

- Após este comando, sempre que o valor da variável `var` é modificado, o programa irá imprimir o valor velho e o novo.

Dica

gdb define qual é a variável a ser observada pelo seu escopo, relativamente a posição onde você está na hora da observação.

Outros comandos

backtrace - produz um trace da pilha das chamadas de função que levaram a um segmentation fault.

Outros comandos

backtrace - produz um trace da pilha das chamadas de função que levaram a um segmentation fault.

where - o mesmo que backtrace; funciona mesmo quando ainda estamos no meio do programa.

Outros comandos

backtrace - produz um trace da pilha das chamadas de função que levaram a um segmentation fault.

where - o mesmo que backtrace; funciona mesmo quando ainda estamos no meio do programa.

finish - roda até que a função atual termine.

Outros comandos

backtrace - produz um trace da pilha das chamadas de função que levaram a um segmentation fault.

where - o mesmo que backtrace; funciona mesmo quando ainda estamos no meio do programa.

finish - roda até que a função atual termine.

delete - apaga o ponto de parada especificado.

Outros comandos

backtrace - produz um trace da pilha das chamadas de função que levaram a um segmentation fault.

where - o mesmo que backtrace; funciona mesmo quando ainda estamos no meio do programa.

finish - roda até que a função atual termine.

delete - apaga o ponto de parada especificado.

info breakpoints - mostra informações sobre os breakpoints.

Outros comandos

backtrace - produz um trace da pilha das chamadas de função que levaram a um segmentation fault.

where - o mesmo que backtrace; funciona mesmo quando ainda estamos no meio do programa.

finish - roda até que a função atual termine.

delete - apaga o ponto de parada especificado.

info breakpoints - mostra informações sobre os breakpoints.

Dica

Olhar nas seções 5 e 9 do manual especificado no início para outros comandos úteis.

The End