# Depurando Aplicações C com o GDB

Mateus Boiani[1]

[1]Universidade do Estado de Santa Catarina
Departamento de Ciência da Computação

13 de outubro de 2016

# SUMÁRIO

# INTRODUÇÃO

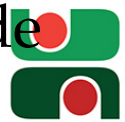
# INTRODUÇÃO

"Muitas vezes nos deparamos com situações inusitadas e aparentemente sem explicação enquanto programamos. E para descobrir o motivo desta situação utilizamos o famoso teste de mesa ou inserimos algumas impressões no meio do código afim de avaliar se uma determinada variável contém o valor preterido. Mas será que não existem métodos mais eficientes para identificar esses defeitos e explicar essas situações?"

UDESC

# O QUE É GDB?

- É um depurador :o
- Um depurador permite a visualização do que esta acontecendo dentro de um programa enquanto ele esta sendo executado.
- Permite quatro tipos de operações principais:
  - Iniciar o programa especificando qualquer coisa que possa afetar seu comportamento;
  - Fazer o programa parar em condições especificas;
  - Examinar o que aconteceu quando o programa foi interrompido;
  - Modificar informações no programa, de modo que seja possível experimentar os efeitos de uma correção;
- Funciona em diversas linguagens de programação além de C e C++;



UDESC

# COMO UTILIZÁ-LO?

Geralmente compilamos assim:

```
gcc [flags] <arquivos fonte> -o <saida>
```

Exemplo:

```
gcc -Wall main.c -o main
```

Para usar o gdb precisamos compilar usando **-g** para habilitar o suporte a debugg:

```
gcc [flags] -g <arquivos fonte> -o <saida>
```

Exemplo:

```
gcc -Wall -g main.c -o main
```

# INICIANDO

- Simplesmente digite “gdb” para entrar na interface. **(gdb)**
- É possível selecionar o executável que você deseja depurar de duas formas:
  1. **gdb** <**arquivo compilado**>
  2. Após entrar na interface digitar: **(gdb)** **file** <**arquivo compilado**>

UDESC

# EXECUTANDO O PROGRAMA

Para executar o programa basta executar:

```
(gdb) run
   ou
(gdb) r
```

Se o programa não tiver nenhum erro, ele vai executar normalmente como deveria. Caso contrário ele ira retornar alguma informação que possa ser útil para localizar o erro, como:

- Linha que o programa estava executando quando parou;
- Função que estava executando;
- Paramêtros que a função recebeu;
- Entre outros;

# DEPURANDO O PROGRAMA

Caso seu programa não esteja executando corretamente, com o gdb você pode investigar o problema de diversas formas, como:

- Executar o programa passo a passo;
- Utilizar pontos de parada;
- Utilizar pontos de parada condicional;
- Perseguindo Variáveis;
- Consultando e Modificando o conteúdo das variáveis em tempo de execução;
- Entre outros;

# UTILIZANDO PONTOS DE PARADA

Pontos de Parada ou *Breakpoints* podem ser usados para parar o programa no meio da sua execução em um determinado ponto. Esse ponto pode ser determinado pelo comando **break**. O comando pode ser combinado e utilizado de diversas formas, indicando que o programa pare em uma linha, em uma função ou quando uma condição for satisfeita.
Exemplo, parando em uma linha especifica:

```
(gdb) break arqfonte.c:<line>
```

# UTILIZANDO PONTOS DE PARADA

Exemplo, parando em uma função especifica:
Suponhamos que a função abaixo esteja definida:

```
1   int func(char *name, int *age);
```

Podemos parar sempre que esta função for chamada desta forma:

```
(gdb) break func
```

# DEPOIS DE PARAR EM UM PONTO DE PARADA

Depois de parar em um ponto de parada você pode fazer várias coisas, como:

- Continuar até o próximo ponto de parada;

```
(gdb) continue
```

- Pode executar a linha que está e parar na próxima linha de código;

```
(gdb) next
```

- Pode pular para a próxima linha, executá-la e parar;

```
(gdb) step
```

- Avaliar o valor das variáveis;

```
(gdb) print <var>
```

- Entre outros;

UDESC

# PONTOS DE PARADA CONDICIONAIS

Permite definir um ponto de parada e só parar quando uma condição específica ocorrer.
Exemplo:
Suponhamos o seguinte trecho de código:

```
1        ...
2        int a = 5, b = 6;
3        int n = a + b;
4        int *vet = (int *) malloc ( sizeof(int) * n);
5        ...
```

```
(gdb) break arqfonte.c:4 if n > 10
```

# CONSULTANDO VARIÁVEIS

É possível consultar o conteúdo de uma variável da seguinte forma:

```
(gdb) print <var>
```

Se a variável for um ponteiro é possível resolvê-lo antes de imprimir. De forma similar é possível acessar valor específicos de uma estrutura.

# CONSULTANDO VARIÁVEIS

Outra forma de ver o conteúdo de uma variável, por exemplo vetor é da seguinte forma, suponhamos o trecho de alocação abaixo:

```
1    ...
2    int *vec = (int *) malloc ( len * sizeof(int) );
3    ...
```

```
(gdb) print *vec@len
```

# CONSULTANDO VARIÁVEIS

Agora imagine que você tem o seguinte trecho de código:

```
1  ...
2  n = 5;
3  int *vec = (int *) malloc ( n * sizeof(int) );
4  ...
5  n = 32;
6  ...
7  for (int i = 0; i < n; i++ )
8      printf("%d ", vec[i]);
9  ...
```

Ao executar o programa você recebe o erro de *segmentation fault*. É possível analisar desta forma:

```
(gdb) print *vec@n
```

Caso n for maior que a capacidade do vetor alocado, possivelmente o gdb irá anunciar que a posição especificada não pode ser acessada.

# ALTERANDO O CONTEÚDO DE UMA VARIÁVEL

Para alterar o conteúdo de uma variável basta utilizar o comando:

```
(gdb) set variable var =  <value>
```

Por exemplo:

```
(gdb) set variable i =  10
```

# PERSEGUINDO VARIÁVEIS

Perseguindo Variáveis é um título de minha autoria para o que o gdb define como *Watchpoints*. *Watchpoints* permitem você acompanhar uma variável durante toda a execução do programa, parando a execução **sempre** que o conteúdo desta variável for modificado.
O comando é:

```
(gdb) watch var
```

# OUTRAS FUNCIONALIDADES BÁSICAS

Roda o programa até que a função seja finalizada.

```
(gdb) finish
```

Mostra as informações de todos *Breakpoints* declarados:

```
(gdb) info breakpoints
```

Para sair do gdb basta:

```
(gdb) quit
```

ou

```
(gdb) q
```

Outras informações, comandos e mais detalhes são encontrados na **documentação completa do gdb**.

UDESC

# CONCLUSÃO

" É uma ferramenta extremamente importante. Utilizada em larga escala por desenvolvedores por permitir a identificação e correção rápida em falhas de *software*."

OBRIGADO

Perguntas?!

UDESC