MC202 – AULA DE LABORATÓRIO

Depuração com gdb

DEPURAÇÃO

É O PROCESSO DE ENCONTRAR, CORRIGIR OU MITIGAR OS EFEITOS DE CARACTERÍSTICAS INDESEJADAS NO CÓDIGO.

UTILIZAMOS O PROGRAMA `gdb` PARA REALIZAR ESTE PROCESSO COM O PADRÃO C-ANSI

POR QUE USAR O GDB?

- AJUDA A ENCONTRAR ERROS SEMÂNTICOS E COMPORTAMENTOS INDESEJADOS NO PROGRAMA
- ALTERNATIVA A UTILIZAR VÁRIOS `printf` PARA VERIFICAR O FLUXO E VALORES DE VARIÁVEIS DO PROGRAMA

GDB

- PARA SER HABILITADO, É NECESSÁRIO QUE O CÓDIGO A SER DEPURADO SEJA COMPILADO COM A FLAG -G
- PARA INICIAR O GDB:

- UTILIZE O COMANDO:

```
gdb arquivo.x
```

- OU DENTRO DO GDB:

```
(gdb) file arquivo.x
```

COMO O GDB FUNCIONA?

FUNCIONA COMO UM TERMINAL DE LINHA DE COMANDOS

- AUTOCOMPLETE USANDO A TECLA TAB
- HISTÓRICO DE COMANDOS COM ↑ E ↓
- USE HELP OU HELP PARA AJUDA

ASPECTOS BÁSICOS PARA TER EM MENTE

O GDB TEM TRÊS CONCEITOS QUE AJUDAM A DEPURAR SEU CÓDIGO:

- BREAKPOINTS
 - CRIA PONTOS DE PARADA EM LOCALIZAÇÕES ESPECÍFICAS DOS ARQUIVOS DE CÓDIGO
 - QUANDO O GDB ENCONTRA UM BREAKPOINT, ELE INTERROMPE A EXECUÇÃO DO PROGRAMA PARA QUE ELE POSSA SER ANALISADO

ASPECTOS BÁSICOS PARA TER EM MENTE

O GDB TEM TRÊS CONCEITOS QUE AJUDAM A DEPURAR SEU CÓDIGO:

- WATCHPOINTS
 - UM PONTO DE OBSERVAÇÃO EM ALGUMA VARIÁVEL DO PROGRAMA
 - SEMPRE QUE A VARIÁVEL É ATUALIZADA, A EXECUÇÃO É INTERROMPIDA E SEUS VALORES ANTERIORES SÃO MOSTRADOS

ASPECTOS BÁSICOS PARA TER EM MENTE

O GDB TEM TRÊS CONCEITOS QUE AJUDAM A DEPURAR SEU CÓDIGO:

- QUERYING
 - QUANDO A EXECUÇÃO ESTA INTERROMPIDA, PODEMOS EXECUTAR AÇÕES SOB O CÓDIGO NO ESCOPO ATUAL DE EXECUÇÃO
 - E POSSÍVEL IMPRIMIR CONTEÚDO DE VARIÁVEIS, E ATÉ MESMO FORÇAR A INVOCAÇÃO DE FUNÇÕES DO SEU CÓDIGO

ALGUNS COMANDOS DO GDB

(GDB) RUN: EXECUTA O PROGRAMA CARREGADO ANTERIORMENTE

(GDB) BREAK <ARQUIVO.C>:<LINHA> : CRIA UM BREAKPOINT NA <LINHA>DO <ARQUIVO.C>

(GDB) BREAK <NOME_FUNCAO>: CRIA UM BREAKPOINT NA DECLARAÇÃO DA FUNÇÃO <NOME_FUNCAO>

(GDB) WATCH <VARIABLE>: CRIA UM WATCHPOINT PARA A VARIÁVEL <VARIABLE>

ALGUNS COMANDOS DO GDB

(GDB) CONTINUE : RESUME A EXECUÇÃO DO PROGRAMA APÓS UMA INTERRUPÇÃO

(GDB) STEP : EXECUTA A PRÓXIMA LINHA DO PROGRAMA APÓS UMA INTERRUPÇÃO

(GDB) NEXT : EXECUTA O PRÓXIMO BLOCO DE COMANDOS APÓS UMA INTERRUPÇÃO

(GDB) PRINT <VARIABLE> : IMPRIME O CONTEÚDO DA VARIÁVEL <VARIABLE>

(GDB) CALL <EXPRESSAO> : FORÇA A EXECUÇÃO DE A PARTIR DOS SÍMBOLOS DEFINIDOS NO ESCOPO ATUAL

EXEMPLO!

Esse será o código usado para o exemplo

Lembrando que ele deve ser compilado com a flag `-g` para que o GDB funcione corretamente.

```
#include <stdio.h>

int funcao1(int a)
{
    int b;
    b = 7;
    b = a + b;
    return b / 2;
}

int funcao2(void)
{
    int i;
    printf("Informe um inteiro:\n");
    scanf("%d", &i);
    i += 5;
    return i;
}

int main(void)
{
    int a = 0, b = 0, i = 0;

    a = funcao1(2);
    b = funcao2();
    i = funcao1(b);

    printf("%d, %d, %d\n", a, b, i);
    return 0;
}
```

EXEMPLO!

Para executar o gdb basta
chamar no terminal
`$gdb NOME_DO_EXECUTAVEL`

nesse caso:

`$gdb exemplo`

A interface do GDB é muito
semelhante a de um terminal.

```
pirate@yogurt ~/p/m/p/m/gdb_class> gdb exemplo
GNU gdb (Debian 8.1-4) 8.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from exemplo...(no debugging symbols found)...done.
(gdb) █
```

EXEMPLO!

run: Uma vez aberto o GDB, o comando run (r) executa o programa desde o seu início.

```
(gdb) r
Starting program: /home/pirate/projects/master/ped/mc202/gdb_class/exemplo
Informe um inteiro:
5
4, 10, 8
[Inferior 1 (process 4460) exited normally]
(gdb) □
```

EXEMPLO!

break: para parar o programa em um ponto específico usamos o comando break (b). Nesse exemplo ele recebe como argumento o nome da função onde ele deve parar (main). Uma vez que ele entra no main, a execução para e o controle é passado para o usuário.

```
(gdb) break main
Breakpoint 1 at 0x5555555551b5
(gdb) r
Starting program: /home/pirate/projects/master/ped/mc202/gdb_class/exemplo

Breakpoint 1, 0x00005555555551b5 in main ()
(gdb) □
```

EXEMPLO!

continue (c): O comando continue devolve a execução para o programa. Ele é diferente de "run" pois o programa continua do ponto onde ele havia parado.

```
(gdb) break main
Breakpoint 1 at 0x5555555551b5
(gdb) r
Starting program: /home/pirate/projects/master/ped/mc202/gdb_class/exemplo

Breakpoint 1, 0x00005555555551b5 in main ()
(gdb) c
Continuing.
Informe um inteiro:
2
4, 7, 7
[Inferior 1 (process 5515) exited normally]
(gdb) □
```

EXEMPLO!

`info break (i b)`: Esse comando mostra os breakpoints existentes, cada um é identificado por um valor inteiro. Esse valor é importante, pois para nós removermos um breakpoint devemos usar o comando `delete (d)` seguido do inteiro que identifica ele.

```
(gdb) info break
Num      Type          Disp Enb Address          What
1        breakpoint    keep y   0x0000555555551b5 <main+4>
          breakpoint already hit 1 time
(gdb) de
define   delete    demangle detach
(gdb) de
define   delete    demangle detach
(gdb) delete 1
(gdb) info break
No breakpoints or watchpoints.
(gdb) □
```

EXEMPLO!

```
(gdb) b funcao1
Breakpoint 2 at 0x5555555555159
(gdb) r
Starting program: /home/pirate/projects/master/ped/mc202/gdb_class/exemplo

Breakpoint 2, 0x00005555555555159 in funcao1 ()
(gdb) c
Continuing.
Informe um inteiro:
6

Breakpoint 2, 0x00005555555555159 in funcao1 ()
(gdb) c
Continuing.
4, 11, 9
[Inferior 1 (process 6893) exited normally]
(gdb) □
```

EXEMPLO!

No exemplo anterior o breakpoint foi colocado na "funcao1". Perceba que ambas as vezes que o programa chama a funcao1 ele é interrompido e o controle é passado para o usuário, que deve usar "continue" para continuar sua execução

```
(gdb) b funcao1
Breakpoint 2 at 0x55555555159
(gdb) r
(gdb) r
Starting program: /home/pirate/projects/master/ped/mc202/gdb_class/exemplo

Breakpoint 2, 0x000055555555159 in funcao1 ()
(gdb) c
Continuing.
Informe um inteiro:
6

Breakpoint 2, 0x000055555555159 in funcao1 ()
(gdb) c
Continuing.
4, 11, 9
[Inferior 1 (process 6893) exited normally]
(gdb) □
```

EXEMPLO!

Exemplo deletando o breakpoint da "funcao1"

```
(gdb) i b
Num      Type           Disp Enb Address              What
2        breakpoint      keep y  0x000055555555159 <funcao1+4>
          breakpoint already hit 2 times
(gdb) d 2
(gdb) 
```

EXEMPLO!

Nesse exemplo usamos um argumento diferente para o break. Ao invés de dar o nome de uma função, usamos `b NOME_DO_ARQUIVO:LINHA_DO_ARQUIVO` isso faz o gdb parar exatamente antes de executar a linha especificada.

```
(gdb) b exemplo.c:24
Breakpoint 1 at 0x11ce: file exemplo.c, line 24.
(gdb) r
Starting program: /home/pirate/projects/master/ped/mc202/gdb_class/exemplo

Breakpoint 1, main () at exemplo.c:24
24      a = funcao1(2);
(gdb) □
```

EXEMPLO!

```
(gdb) b exemplo.c:24
Breakpoint 1 at 0x111ce: file exemplo.c, line 24.
(gdb) r
Starting program: /home/pirate/projects/master/ped/mc202/gdb_class/exemplo

Breakpoint 1, main () at exemplo.c:24
24      a = funcao1(2);
(gdb) print a
$1 = 0
(gdb) next
25      b = funcao2();
(gdb) print a
$2 = 4
(gdb) n
Informe um inteiro:
2
26      i = funcao1(b);
(gdb) n
28      printf("%d, %d, %d\n", a, b, i);
(gdb) n
4, 7, 7
29      return 0;
(gdb) n
30  }
(gdb) n
__libc_start_main (main=0x555555551b1 <main>, argc=1, argv=0x7fffffff188, init=<optimized out>, fini=<optimized out>,
    rtld_fini=<optimized out>, stack_end=0x7fffffff178) at ../csu/libc-start.c:344
344  ../csu/libc-start.c: No such file or directory.
(gdb) □
```

EXEMPLO!

next (n): Na imagem anterior exemplificamos o uso do comando next. Ele faz com que apenas a próxima linha de código seja executada e o controle volte para o desenvolvedor. Perceba que ele sempre executa a linha inteira, ou seja: ele não entra dentro de funções.

```
(gdb) b exemplo.c:24
Breakpoint 1 at 0x11ce: file exemplo.c, line 24.
(gdb) r
Starting program: /home/pirate/projects/master/ped/mc202/gdb_class/exemplo
Breakpoint 1, main () at exemplo.c:24
24      a = funcao1(2);
(gdb) print a
$1 = 0
(gdb) next
25      b = funcao2();
(gdb) print a
$2 = 4
(gdb) n
Informe um inteiro:
2      i = funcao1(b);
(gdb) n
28      printf("%d, %d, %d\n", a, b, i);
(gdb) n
4, 7, 7
29      return 0;
(gdb) n
30  }
(gdb) n
__libc_start_main (main=0x555555551b1 <main>, argc=1, argv=0x7fffffff188, init=<optimized out>, fini=<optimized out>,
rtld_fini=<optimized out>, stack_end=0x7fffffff178) at ../csu/libc-start.c:344
344  ../csu/libc-start.c: No such file or directory.
(gdb) □
```

EXEMPLO!

step (s): O comando step é muito semelhante ao next, mas ao contrário do anterior ele entra dentro da função que vai ser chamada.

```
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/pirate/projects/master/ped/mc202/gdb_class/exemplo

Breakpoint 1, main () at exemplo.c:24
24      a = funcao1(2);
(gdb) s
funcao1 (a=2) at exemplo.c:6
6      b = 7;
(gdb) □
```

EXEMPLO!

backtrace (bt): Esse comando permite o desenvolvedor de ver quais as funções que já foram chamadas. Nesse exemplo bt é chamado dentro da funcao1, podemos ver que o topo da pilha (posição 0) é a funcao1, e o "frame" seguinte (a função que chamou a funcao1) é a main (posição 1).

```
Breakpoint 1, main () at exemplo.c:24
24      a = funcao1(2);
(gdb) s
funcao1 (a=2) at exemplo.c:6
6      b = 7;
(gdb) backtrace
#0  funcao1 (a=2) at exemplo.c:6
#1  0x00005555555551d8 in main () at exemplo.c:24
(gdb) □
```

EXEMPLO!

```
Breakpoint 1, main () at exemplo.c:24
24      a = funcao1(2);
(gdb) s
funcao1 (a=2) at exemplo.c:6
6      b = 7;
(gdb) backtrace
#0  funcao1 (a=2) at exemplo.c:6
#1  0x000055555555551d8 in main () at exemplo.c:24
(gdb) p a
$3 = 2
(gdb) frame 1
#1  0x000055555555551d8 in main () at exemplo.c:24
24      a = funcao1(2);
(gdb) p a
$4 = 0
(gdb) frame 0
#0  funcao1 (a=2) at exemplo.c:6
6      b = 7;
(gdb) □
```

EXEMPLO!

Na imagem anterior podemos ver exemplo de vários comandos úteis.

`print (p)`: recebe como parâmetro uma variável e imprime o valor dela.

`frame (f)`: Faz o gdb analisar o escopo do frame passado por argumento.

```
Breakpoint 1, main () at exemplo.c:24
24      a = funcao1(2);
(gdb) s
funcao1 (a=2) at exemplo.c:6
6      b = 7;
(gdb) backtrace
#0 funcao1 (a=2) at exemplo.c:6
#1 0x000055555555551d8 in main () at exemplo.c:24
(gdb) p a
$3 = 2
(gdb) frame 1
#1 0x000055555555551d8 in main () at exemplo.c:24
24      a = funcao1(2);
(gdb) p a
$4 = 0
(gdb) frame 0
#0 funcao1 (a=2) at exemplo.c:6
6      b = 7;
(gdb) □
```

EXEMPLO!

Nesse exemplo podemos ver que inicialmente estávamos no frame do topo (posição 0, funcao1). Ambas as funções funcao1 e main possuem a variável "a". Quando pedimos para imprimir "a", ele imprime o valor de "a" na funcao1. Depois trocamos para o frame 1 (função main). Dessa vez, quando pedimos para imprimir "a", ele imprime o valor referente a variável da função main.

```
Breakpoint 1, main () at exemplo.c:24
24      a = funcao1(2);
(gdb) s
funcao1 (a=2) at exemplo.c:6
6      b = 7;
(gdb) backtrace
#0  funcao1 (a=2) at exemplo.c:6
#1  0x000055555555551d8 in main () at exemplo.c:24
(gdb) p a
$3 = 2
(gdb) frame 1
#1  0x000055555555551d8 in main () at exemplo.c:24
24      a = funcao1(2);
(gdb) p a
$4 = 0
(gdb) frame 0
#0  funcao1 (a=2) at exemplo.c:6
6      b = 7;
(gdb) □
```

EXEMPLO!

```
Breakpoint 1, main () at exemplo.c:24
24      a = funcao1(2);
(gdb) s
funcao1 (a=2) at exemplo.c:6
6      b = 7;
(gdb) backtrace
#0  funcao1 (a=2) at exemplo.c:6
#1  0x000055555555551d8 in main () at exemplo.c:24
(gdb) p a
$3 = 2
(gdb) frame 1
#1  0x000055555555551d8 in main () at exemplo.c:24
24      a = funcao1(2);
(gdb) p a
$4 = 0
(gdb) frame 0
#0  funcao1 (a=2) at exemplo.c:6
6      b = 7;
(gdb) call funcao1(3)
$5 = 5
(gdb) call funcao2(a)
Informe um inteiro:
3
$6 = 8
(gdb) □
```

EXEMPLO!

call: Esse comando permite o usuário chamar funções existentes dentro do programa mesmo que elas não façam parte do fluxo correto.

Esse comando pode ser muito útil para debugar pois o desenvolvedor pode passar os parâmetros desejados, e essas funções ainda interagem com a entrada e saída do programa.

```
Breakpoint 1, main () at exemplo.c:24
24      a = funcao1(2);
(gdb) s
funcao1 (a=2) at exemplo.c:6
6      b = 7;
(gdb) backtrace
#0  funcao1 (a=2) at exemplo.c:6
#1  0x000055555555551d8 in main () at exemplo.c:24
(gdb) p a
$3 = 2
(gdb) frame 1
#1  0x000055555555551d8 in main () at exemplo.c:24
24      a = funcao1(2);
(gdb) p a
$4 = 0
(gdb) frame 0
#0  funcao1 (a=2) at exemplo.c:6
6      b = 7;
(gdb) call funcao1(3)
$5 = 5
(gdb) call funcao2(a)
Informe um inteiro:
3
$6 = 8
(gdb) □
```

EXEMPLO!

call:

Isso quer dizer que o desenvolvedor pode criar funções feitas para serem chamadas apenas dentro do gdb que tem a finalidade de imprimir dados mais complexos durante o processo de debug.

```
Breakpoint 1, main () at exemplo.c:24
24      a = funcao1(2);
(gdb) s
funcao1 (a=2) at exemplo.c:6
6      b = 7;
(gdb) backtrace
#0 funcao1 (a=2) at exemplo.c:6
#1 0x000055555555551d8 in main () at exemplo.c:24
(gdb) p a
$3 = 2
(gdb) frame 1
#1 0x000055555555551d8 in main () at exemplo.c:24
24      a = funcao1(2);
(gdb) p a
$4 = 0
(gdb) frame 0
#0 funcao1 (a=2) at exemplo.c:6
6      b = 7;
(gdb) call funcao1(3)
$5 = 5
(gdb) call funcao2(a)
Informe um inteiro:
3
$6 = 8
(gdb) □
```

EXEMPLO!

watch:

O comando watch recebe como parâmetro uma variável. Ele funciona de forma semelhante a um breakpoint, mas ao invés de para quando a execução chega em um ponto, ele para quando a execução troca o valor da variável especificada.

```
Breakpoint 1, main () at exemplo.c:24
24      a = funcao1(2);
(gdb) s
funcao1 (a=2) at exemplo.c:6
6      b = 7;
(gdb) backtrace
#0  funcao1 (a=2) at exemplo.c:6
#1  0x000055555555551d8 in main () at exemplo.c:24
(gdb) p a
$3 = 2
(gdb) frame 1
#1  0x000055555555551d8 in main () at exemplo.c:24
24      a = funcao1(2);
(gdb) p a
$4 = 0
(gdb) frame 0
#0  funcao1 (a=2) at exemplo.c:6
6      b = 7;
(gdb) call funcao1(3)
$5 = 5
(gdb) call funcao2(a)
Informe um inteiro:
3
$6 = 8
(gdb) frame 1
#1  0x000055555555551d8 in main () at exemplo.c:24
24      a = funcao1(2);
(gdb) watch b
Hardware watchpoint 2: b
(gdb) c
Continuing.
Informe um inteiro:
8

Hardware watchpoint 2: b

Old value = 0
New value = 13
main () at exemplo.c:26
26      i = funcao1(b);
(gdb) □
```

EXEMPLO!

quit:

Comando usado para sair do gdb =)

DÚVIDAS?