Guia rápido - GDB e Valgrind

Rafael G. Cano

28 de março de 2012

1 GDB - GNU Debugger

Depuradores como GDB permitem que o programador veja detalhes da execução de um programa, auxiliando na correção de erros no código. Inicialmente, devem ser definidos alguns pontos do programa, denominados *breakpoints*, nos quais a execução será interrompida. Em seguida, o programa é executado, até que um dos breakpoints seja atingido. Nesse momento, é possível imprimir valores de variáveis ou avançar nas linhas do código, procurando por algum problema que comprometa a execução.

Suponha que o arquivo com o código de seu programa se chama `programa.c` e que o executável se chama `programa`. Suponha também que seu código define uma função `int f()`. Siga os seguintes passos para depurar o código com GDB:

1. Compile seu programa com a opção `-g`. Essa opção instrui o compilador a compilar o programa com informações adicionais para depuração, as quais são necessárias para uso do GDB.
   Use o seguinte comando: `gcc -g programa.c -o programa`
   Importante: não utilize a opção `-O3` junto com a opção `-g`. A compilação com otimização pode remover variáveis e até chamadas de função, comprometendo a depuração.

2. Execute o GDB com o comando: `gdb programa`

3. Defina breakpoints em seu código, nos pontos que deseja analisar com mais detalhes. Você pode definir um breakpoint em uma chamada de função ou em uma linha específica do código. Utilize os seguintes comandos:

   - Breakpoint na função main(): `b main`
   - Breakpoint na função f(): `b f`
   - Breakpoint na linha 32: `b programa.c:32`

   Obs: para deletar todos os breakpoints, use: `d`
   Para deletar um breakpoint específico, use o número do breakpoint desejado, por exemplo: `d 2`
   Para listar todos os breakpoints com seus números, use: `info b`

4. Execute o programa com o comando *run*: `r`
   Se seu programa recebe parâmetros pela linha de comando (por exemplo, um inteiro), forneça-os juntamente com o comando `r`, ou seja, utilize: `r 8`

5. Quando a execução for interrompida em um breakpoint, utilize os seguintes comandos:

   - Para imprimir o valor de uma variável *var*, use o comando *print*: `p var`
     Obs: utilize a mesma sintaxe que seria usada no programa em C para acessar o valor desejado, ou seja, se *var* é um inteiro, use `p var`, se é um apontador para um inteiro use `p *var`, se é um vetor use `p var[0]`, `p var[3]`, etc. Você também pode executar funções, por exemplo, com `p f()`
   - Para avançar para a próxima linha de código, use o comando *next*: `n`

- Para avançar várias linhas de código: `n 7`, `n 15`, etc.
- Se a linha atual contém uma chamada a uma subrotina (por exemplo *int v = f();*), use o comando *step*: `s`
  A diferença entre `n` e `s` é que `n` executa a linha sem interromper o programa na função *f()*, enquanto que `s` permite que a depuração continue dentro de *f()*.
- Para continuar a execução do programa , use o comando *continue*: `c`

6. Se seu programa for encerrado por um erro de execução (por exemplo, segfault), use o comando: `where`
   Esse comando mostra o ponto exato do programa onde o erro ocorreu.
7. Para sair do GDB, use o comando *quit*: `q`
8. Para mais informações sobre um comando, use: `help` *comando*

Para mais informações, veja: `http://www.unknownroad.com/rtfm/gdbtut/gdbtoc.html` e `http://www.gnu.org/software/gdb/documentation/`

2 Valgrind

Valgrind é útil principalmente para localizar erros relacionados aos acessos à memória. Em particular, ele pode ser usado para detectar acessos a posições inválidas de um vetor e "vazamentos de memória", ou seja, blocos de memória alocados com *malloc()* que não foram desalocados com *free()*. Considere que seu programa se chama `programa.c` e que o executável se chama `programa`. Suponha que é usado o seguinte código:

```
1:  int main(){
2:    int i;
3:    int *v = (int*)malloc(10*sizeof(int));
4:    for(i = 0; i<=10; i++)
5:      v[i] = i;
6:  }
```

O código acima tem dois problemas: a posição v[10] acessada na última iteração do laço não existe e o vetor não foi desalocado. O relatório do Valgrind aponta ambos os erros:

```
==4296== Invalid write of size 4
==4296==    at 0x80483F6: main (programa.c:5)
==4296==  Address 0x4021050 is 0 bytes after a block of size 40 alloc'd
==4296==    at 0x400677E: malloc (vg_replace_malloc.c:195)
==4296==    by 0x80483D8: main (programa.c:3)
==4296==
==4296==
==4296== HEAP SUMMARY:
==4296==     in use at exit: 40 bytes in 1 blocks
==4296==   total heap usage: 1 allocs, 0 frees, 40 bytes allocated
```

Para utilizar Valgrind, siga os seguintes passos:

1. Compile seu programa com a opção `-g`.
   Use o seguinte comando: `gcc -g programa.c -o programa`
   Importante: não utilize a opção `-O3` junto com a opção `-g`. A compilação com otimização pode remover variáveis e até chamadas de função, comprometendo a depuração.
2. Execute o comando: `valgrind ./programa`

Para mais informações, veja: `http://valgrind.org/docs/`