Il debugger gdb in ambiente Linux: alcuni comandi

Massimo Torquati
torquati@di.unipi.it

Di seguito vengono descritti alcuni semplici comandi per il debugging di codice C e C++ usando il debugger GNU gdb. Per una descrizione completa e dettagliata di ogni comando, nonché di comandi più avanzati, fare riferimento al manuale utente scaricabile liberamente dal seguente indirizzo web:

http://sourceware.org/gdb/current/onlinedocs/gdb.pdf.gz

Per abilitare il debugging, la compilazione del codice sorgente deve avvenire con l'opzione ***-g***, inoltre non è consigliabile utilizzare per la compilazione le opzioni di ottimizzazione del compilatore. Ad esempio per compilare il sorgente *prova.c/prova.cpp* :

```
gcc -g prova.c -o prova
g++ -g -std=c++11 -Wall prova.cpp -o prova
```

- per lanciare il debugger sull'eseguibile *prova*:
  gdb *-**args** prova <eventuali argomenti di prova>*

- ***set args*** <lista argomenti> permette di settare gli argomenti nella linea di comando anche dopo il lancio dell'eseguibile con il comando precendente

- **b (break)** permette di settare un breakpoint ad una linea specifica del codice sorgente oppure ad una funzione, es:
  b 120 # inserisce un breakpoint alla linea 120 del file corrente
  b myfile.cpp:120 # inserisce un breakpoint alla linea 120 del file myfile.cpp
  b miaF # inserisce un breakpoint alla prima istruzione della funzione miaF

- **p (print)** stampa il contenuto di una variabile o di un indirizzo di memoria
  p argv[1] # stampa il primo argomento passato a linea di comando
  p myvar # stampa il contenuto di myvar
  p &myvar # stampa l'indirizzo di memoria di myvar
  p *(long *)0x614c20 # stampa il contenuto della locazione di memoria 0x614c20 castata a long
  p A[10]@10 # stampa a partire dalla locazione 10 dell'array A altre 10 entries
  ….

- **pt (ptype)** stampa il tipo di una variabile

- **l (list)** esegue il listing del codice
  l 10 stampa li listing centrato alla linea 10 del file corrente (stampa un pò di righe prima e dopo la linea 10)

- **n (next)** permette di avanzare all'istruzione successiva (non entra nelle funzioni/procedure)

- **s (step)** permette di avanzare all'istruzione successiva entrando nelle funzioni/procedure

- **bt (backtrace)** permette di visualizzare lo stack di tutte le chiamate con I relativi argomenti, lo stack è navigabile con I comandi **up** e **down**.

- **info threads** permette di visualizzare la lista dei threads (visualizza id, process id e frame)

- **thread 'ID'** permette di switchare il controllo del debugger sul thread con id 'ID'.

A caccia di bugs e di memory leaks con valgrind

Valgrind è uno strumento in grado di istrumentare il codice ed eseguire analisi dinamica (a tempo di esecuzione) di un programma tracciando tutti gli accessi in memoria e riportando all'utente un insieme di problematiche riscontrate durante l'esecuzione, tra le quali, i problemi relativi ad accessi non corretti a locazioni di memoria (detti anche *buffer overrun*), e memory leaks, cioè porzioni di memoria allocate che non sono più in uso dal programma ma che non sono state liberate. Valgrind può essere usato anche come strumento di profiling del codice.

Riferimento: http://valgrind.org/

E' importante compilare il codice con l'opzione **-g** e non è consigliabile utilizzare opzioni di ottimizzazione superiori a -O1.

- per lanciare valgrind sull'eseguibile prova:
  valgrind *prova <eventuali argomenti di prova>*

- per valutare presenza di memory leaks
  valgrind –leak-check=full *prova <eventuali argomenti di prova>*

Un breve tutorial di valgrind è reperibile al seguente link:

http://valgrind.org/docs/manual/quick-start.html