Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni
Sede di Napoli

Debug con GDB

Gennaro Oliva
Laboratorio di sistemi operativi
01/06/2007

Debug

- Il debug di un programma è necessario quando un software non si comporta come ci aspetteremmo ovvero quando i risultati ottenuti nell'esecuzione sono differenti da quelli riportati nelle specifiche
- In genere questo accade per la presenza di piccoli frammenti di codice errati detti bug
- Il debug di un programma è la fase di ricerca dei bug che ne causano il malfunzionamento

Stato di un programma

- Lo stato di un programma è l'insieme dei valori di tutte le variabili
- Un bug causa un'alterazione dello stato del programma che determina un malfunzionamento
- Un debugger ci da la possibilità di eseguire le istruzioni di un programma una per volta ed analizzare lo stato del programma ad ogni passo

Manifestazione dei bug

- Cosa accade quando c'è un bug che non viene rilevato dal compilatore come errore di sintassi?

  1) l'esecuzione si interrompe inaspettatamente

  2) l'esecuzione non si conclude

  3) l'esecuzione termina fornendo risultati errati

Debug

- Il debug di un programma consta di tre fasi successive:
  - trovare le istruzioni che causano il bug
  - scoprire il motivo del bug
  - correggere il codice

- La prima fase è certamente la più difficile e le tecniche da utilizzare nella individuazione dei bug dipendono dalla sua manifestazione

Interruzione inaspettata

- Ispezioniamo lo stato del programma nel punto in cui si è verificata l'interruzione
- Scopriamo quali modifiche dello stato hanno determinato l'interruzione
- Monitoriamo l'esecuzione del programma per determinare in quale punto lo stato è cambiato in modo errato

Mancata conclusione

- Verifichiamo quali condizionali o istruzioni causano un loop nell'esecuzione
- Determiniamo se il loop è determinato da:
- Uno o più condizionali errati
- Un errato cambiamento stato
  - monitoriamo l'esecuzione del programma per determinare il punto in cui lo stato è cambiato in modo errato

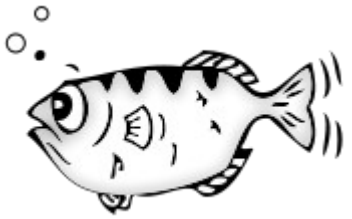
Esecuzione con risultato errato

- Verificando lo stato prima del risultato individuiamo le strutture dati che causano l'errore nel risultato
- Suddividiamo il programma in sezioni e verifichiamo se le singole sezioni di codice operano correttamente sulle strutture dati che causano l'errore e sulle parti dello stato da cui queste dipendono
- Individuiamo le sezioni di codice che modificano erroneamente lo stato del programma

GDB: The Gnu Project Debbuger

- GDB è il debugger ufficiale del progetto GNU
- Lo sviluppo di GDB è stato intrapreso nel 1986 da Richard Stallman
- GDB è software libero rilasciato nei termini della licenza GPL General Public License
- GDB supporta i linguaggi C, C++, Objective-C, Fortran, Java, Pascal, Assembler, Modula-2 e Ada

`http://sourceware.org/gdb/`



Dalla documentazone di gdb

- Lo scopo di un debugger quale GDB è di farti vedere cosa sta succedendo “all'interno” di un altro programma durante la sua esecuzione o quello che stava facendo il programma quando è crashato
- GDB può fare quattro cose per aiutarti a scovare un bug in azione:
  - eseguire il tuo programma specificando qualsiasi cosa che possa interferire con il suo comportamento
  - sospendere il tuo programma in specifiche condizioni
  - esaminare cosa è successo quando un programma si è interrotto
  - cambiare le cose nel tuo programma in modo che tu possa correggere gli effetti di un bug e scovarne un altro

Invocazione di GDB

- Per utilizzare il debugger GDB è necessario compilare i programmi con l'opzione –g:

```
[studente@lab]$ gcc -o test -g test.c
```

- L'opzione -g del compilatore gcc produce un file eseguibile che contiene informazioni di debug utilizzate da GDB
- Per eseguire il programma con GDB si utilizza la sintassi:

```
[studente@lab]$ gdb test

...

(gdb)
```

- Il prompt (gdb) indica che GDB è in attesa di comandi

Comandi base

- I comandi di base del gdb sono:
- run avvia l'esecuzione del programma
- help che fornisce informazioni sui comandi
- quit che termina l'esecuzione del debugger
- she esegue un comando in una shell
- Gli argomenti da passare al programma devono essere specificati quando si invoca l'esecuzione mediante run con la stessa sintassi che utilzza il programma
- Esempi:

```
(gdb) run -f data
(gdb) run < input
```

I comandi di stato

- I seguenti comandi consentono di esplorare lo stato di un programma in fase di esecuzione:
- where:visualizza la funzione in esecuzione in un determinato istante è utile nel caso di interruzione brusca del programma o nel caso di loop
- up,down:si muove attraverso lo stack delle procedure
- list:visualizza alcune linee del codice sorgente successive e precedenti all'istruzione corrente
- print:mostra il valore corrente di una variabile; può visualizzare qualsiasi tipo di dato: array, struct, oggetti, indirizzi di memoria, ...

L'operatore @, printf e display

- L'operatore @ consente di visualizzare un numero arbitrario di elementi di un array. Esempio:

```
(gdb) print Y[0]@5
```

- visualizza i primi 5 elementi dell'array Y
- Un'istruzione alternativa è printf che consente di visualizzare output formattati secondo la ben nota sintassi della funzione di libreria stdio

```
(gdb) printf "%d\t%d\n", X[0] , Y[0]
 5    0
```

- L'istruzione display consente di visualizzare costantemente il contenuto di una struttura dati

Breakpoint

- Un breakpoint è un punto in cui il programma deve essere bloccato in attesa di ulteriori direttive da parte dell'utente
- break è il comando per l'inserimento di breakpoint ed accetta come argomenti il nome di una procedura o un numero di linea. Esempi:

```
(gdb) break main
```

- inserisce un breakpoint alla prima istruzione

```
(gdb) break 16
```

- inserisce un breakpoint alla sedicesima riga di codice

Delete

- I breakpoint sono numerati con interi consecutivi
- delete è l'istruzione per eliminare i breakpoint
- Una volta giunti ad un breakpoint il debugger visualizza l'istruzione successiva e restituisce il prompt in attesa di istruzioni. Esempio:

```
Breakpoint 1, main (argc=1, argv=0xbfa5cf14)
  at scal.c:13

13          if ( argc != 3 )

(gdb)
```

Comandi per l'esecuzione

- next esegue l'istruzione successiva e qualora questa sia una funzione, esegue l'intera procedura
- step esegue l'istruzione successiva e qualora questa sia una funzione, si arresta alla prima istruzione nella procedura
- cont posegue l'esecuzione fino al prossimo breakpoint o alla fine del programma

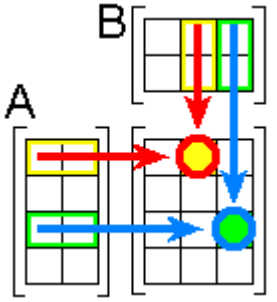
Watchpoint

- Un'alternativa ai breakpoint sono i watchpoint che interrompono l'esecuzione quando il programma accede ad una struttura dati (una variabile, un array, una struct, un oggetto, ...)
- Il vantaggio nell'uso di watchpoint sta nel non dover specificare l'istruzione in cui gdb deve fermarsi (come accade per il breakpoint)
- watch var interrompe l'esecuzione se il programma modifica var ovvero quando vi accede in scrittura
- rwatch var quando var è letta dal programma
- awatch var quando var è letta o scritta dal programma

matmat.c

- Il programma matmat.c esegue il prodotto tra due matrici

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots \\ a_{2,1} & a_{2,2} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} b_{1,1} & b_{1,2} & \dots \\ b_{2,1} & b_{2,2} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}$$



$$\mathbf{AB} = \begin{bmatrix} a_{1,1}\begin{bmatrix} b_{1,1} & b_{1,2} & \dots \end{bmatrix} + a_{1,2}\begin{bmatrix} b_{2,1} & b_{2,2} & \dots \end{bmatrix} + \cdots \\ a_{2,1}\begin{bmatrix} b_{1,1} & b_{1,2} & \dots \end{bmatrix} + a_{2,2}\begin{bmatrix} b_{2,1} & b_{2,2} & \dots \end{bmatrix} + \cdots \\ \vdots \end{bmatrix}$$

Immagini prese da http://www.wikipedia.org

GDB e fork

- GDB non ha supporto speciale per programmi che effettuano fork
- GDB di default continua a monitorare il processo padre mentre il figlio esegue l'esecuzione senza interruzioni
- Se si desidera monitorare il figlio è necessario modificare una variabile d'ambiente gdb chiamata follow-fork-mode
- Le variabili di ambiente gdb ne determinano il comportano e possono essere visualizzate mediante il comando show e modificate mediante il comando set

GDB e fork

- Se si intende effettuare il debug di entrambi i processi padre e figlio è possibile inserire una sleep subito dopo la fork nella sezione di codice del figlio e “attaccarsi” all'esecuzione del processo in una nuova sessione di gdb
- Per invocare gdb su un processo in esecuzione si specifica il PID del processo dopo il nome dell'eseguibile sulla linea di comando. Esempio:

```
[studente@lab]$ gdb programmaconfork 8854
attaching to programmaconfork process 8854
```

- Questa sintassi si può utilizzare su qualsiasi processo in esecuzione

GDB e thread

- GDB supporta il debug di programmi multi-thread
- GDB numera i thread mediante interi crescenti
- La creazione di un nuovo thread viene notificata all'utente
- info thread fornisce informazioni sui thread esistenti
- thread consente di cambiare il tread corrente
- thread apply consente di eseguire uno stesso comando su una lista di thread
- GDB consente di impostare un breakpoint per uno specifico thread

Breakpoint per thread

- Per impostare un break per uno specifico thread si utilizza la sintassi break line thread threadno. Esempio:

```
break 42 thread 2
```

effettua il break del tread 2 alla linea 42

- Se non si specifica nessun thread il breakpoint sarà valido per tutti
- Quando l'esecuzione di un programma multi-thread si arresta per qualsiasi motivo, tutti i thread si arrestano
- Quando l'esecuzione del programma riprende, tutti i thread riprendono anche se si usa solo step o next

Interfacce grafiche e Integrated Development Environment

- Esistono diverse interfacce grafiche per gdb:
  - xxgdb
  - ddd
  - KDbg
  - tgdb
- Molti ide hanno gdb integrato:
  - Kdevelop
  - Anjuta
  - Vim/Emacs
  - Dev C++ (windows)

Licenza

- Questa presentazione è fornita secondo i termini della licenza Creative Commons Attribuzione/Condividi allo stesso modo 2.5 secondo la quale tu sei libero:
  - di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare quest'opera
  - di modificare quest'opera
- Alle seguenti condizioni:
  - devi attribuire la paternità dell'opera nei modi indicati dall'autore o da chi ti ha dato l'opera in licenza.
  - se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con una licenza identica a questa.
- Il testo integrale della licenza è consultabile al seguente indirizzo: http://creativecommons.org/licenses/by-sa/2.5/it/legalcode