5 Debugging: utilizzo di `gdb`

Il *debugging* è il processo di ricerca e rimozione degli errori (bug) di un programma. In linguaggi ad alto livello, il primo strumento utilizzato è la stampa su terminale o su file di log, per individuare rapidamente i punti d'errore. Qui invece questo non è altrettanto facile, e dobbiamo affidarci ad un debugger completo, per l'appunto `gdb`.

Il principale scopo del debugger è far eseguire il programma *un passo alla volta*, e permetterci di osservare lo stato dei registri e della memoria a ciascuno di questi passi. Quali sono questi "passi" su cui soffermarsi lo decidiamo noi, definendo dei *breakpoints*: quando il programma giunge ad un brakepoint, il debugger mette in pausa l'esecuzione e lascia a noi il controllo. Possiamo allora, tramite specifici comandi, stampare informazioni sullo stato, proseguire un'istruzione alla volta, far continuare fino al prossimo breakpoint, etc.

5.1 Avvio di `gdb`

La sintassi più semplice è `gdb percorso_eseguibile`.
Lo script di debug nell'ambiente del corso fa dei passi in più, che semplificano l'utilizzo: definisce i comandi `rr` e `qq` (sezione successiva) ed esegue i comandi per mettere un breakpoint a `_main` ed avviare l'esecuzione del programma.

Attenzione a quello che `gdb` stampa all'avvio: se la terzultima riga è la seguente conviene fermarsi subito.

```
warning: Source file is more recent than executable.
```

Come dice l'errore, non abbiamo riassemblato il programma dopo le ultime modifiche.

5.2 Comandi per il controllo dell'esecuzione

Per guidare l'esecuzione del programma si usano i seguenti comandi[1]:

- **f**rame
  Mostra la posizione attuale del programma, ossia l'istruzione che sta per essere eseguita e la riga a cui corrisponde nel file sorgente.
- **b**rake *label*
  Inserisce un breakpoint alla posizione indicata da *label*. L'esecuzione del programma verrà quindi messa in pausa prima di eseguire l'istruzione associata a *label*.
- **c**ontinue
  Prosegue l'esecuzione fino al prossimo breakpoint.
- **s**tep
  Esegue una singola istruzione.
  Attenzione se questa è una `CALL`: il debugger si fermerà *dentro* il sottoprogramma chiamato.

[1] I caratteri evidenziati in **grassetto** sono il minimo necessario perché `gdb` riconosca il comando. Non c'è quindi bisogno di scriverli per intero.

- **fin**ish
  Prosegue l'esecuzione del sottoprogramma corrente (sia *f*) finché non termina (`RET`). L'esecuzione del programma verrà quindi messa in pausa prima di eseguire l'istruzione successiva all'istruzione `CALL f`.

- **n**ext
  Continua l'esecuzione fino alla successiva istruzione di questo file.
  Questo significa che si comporta come *step*, tranne quando l'istruzione da eseguire è una `CALL`. In tal caso, il sottoprogramma viene eseguito senza pause.

- **r**un
  Avvia l'esecuzione del programma. Se il programma è già in esecuzione, dopo aver chiesto conferma all'utente, riavvia il programma dall'inizio.

- **q**uit
  Termina il debugger. Se il programma è ancora in esecuzione, chiede prima conferma all'utente.

5.2.1 Comandi aggiuntivi

I seguenti comandi non fanno parte dei comandi standard di `gdb`. Sono invece comandi personalizzati definiti all'avvio dallo script di debug fornito nell'ambiente, aggiunti allo scopo di semplificare i casi d'uso più comuni.

- **rr**un
  Riavvia il programma, senza chiedere conferma.

- **qq**uit
  Termina il debugger, senza chiedere conferma.

5.3 Comandi per ispezionare lo stato del programma

Oltre a controllare il percorso seguito dal programma, è ovviamente utile controllare lo stato di registri e memoria prima e dopo l'esecuzione delle istruzioni di interesse. Per far questo useremo i seguenti comandi:

- **i**nfo **r**egister *registro*
  L'argomento *registro* è opzionale, e deve essere in minuscolo e senza caratteri preposti: *eax*, *bx*, *cl*.
  Se specificato, mostra il contenuto del registro, prima in esadecimale e poi in un formato che dipende dal tipo di registro:

  - decimale per registri accumulatori
  - label+offset per *eip*
  - lista dei flag a 1 per *eflags*

  Se non specificato, farà quanto sopra per tutti i registri.

- **x**/*NFU indirizzo*
  La `x` sta per *examine memory*, in questo caso non è un'abbreviazione e non si può usare una versione più lunga.
  Notiamo intanto che ci sono diversi argomenti: numero (*N*), formato (*F*), dimensione (*U*) e indirizzo. Sono tutti *opzionali*, perché questo è un comando *con memoria*. Verranno infatti ricordati gli ultimi argomenti ed utilizzati come valori di default, ad eccezione di *N* il cui default è sempre 1. Per evitare confusioni, si consiglia comunque di specificare sempre tutto.

  L'argomento *indirizzo* indica la locazione da cui *iniziare*. Questo può essere indicato:

  - in esadecimale: `x 0x56559066`

- tramite label preceduta da `&`: `x &buffer`
- tramite registro preceduto da `$`: `x $esi`

L'argomento *N* indica il numero di locazioni da accedere. Questo è un semplice numero decimale. Se negativo, le locazioni saranno mostrate andando all'indietro.

L'argomento *F* indica il formato[2] con cui interpretare il contenuto delle locazioni, e quindi come va stampato a schermo:

- `x` esadecimale
- `d` decimale
- `c` ASCII
- `t` binario

L'argomento *U* indica la dimensione[3] delle locazioni da accedere.

- `b` 1 byte
- `h` word, 2 byte
- `w` long, 4 byte

[2] Molti formati sono stati qui omessi perché, a noi, poco utili. Usare il comando `help x` per una lista esaustiva

[3] Le sigle sono diverse dal GAS perché diversa è la definizione di "word": h sta per halfword (per noi word), mentre w sta per word (per noi long).