L'ambiente UNIX/Linux

Debug Multi-Process e Thread

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

Concorrenza e debug

- Effettuare il debug di programmi concorrenti è estremamente complesso
- Più complessa è l'applicazione concorrente più complesso è il debug
  - I deadlock sono i problemi più facili da individuare
  - Le race condition sono molto difficili da duplicare
- Occorre
  - Cercare di minimizzare le sorgenti di errore
  - Utilizzare strumenti opportuni

Fork e debug

- La maggior parte dei tool di debug non fornisce supporti particolari per il debug di programmi con istruzioni fork
  - Questo vale anche per GDB che è il debugger C/C++ più utilizzato
- Se un programma effettua una fork normalmente GDB
  - Continua il debug del processo originale (il genitore)
  - Ignora l'esecuzione dei figli

Fork e debug

- Spesso è però possibile effettuare il debug dei figli utilizzando degli stratagemmi
  - Le tecniche dipendono dall'IDE utilizzato per effettuare il debug
  - La maggiore flessibilità spesso si ottiene con il tool GDB non integrato in IDE specifici

GDB (con emacs)

- Si supponga un processo esegua una fork e si voglia effettuare il debug del processo padre e del figlio
- In GDB è possibile il seguente procedimento
  - All'interno del programma di cui effetture il debug, occorre
    - Inserire una **sleep** di attesa all'inizio dell'esecuzione dei processi padre e figlio
    - Eseguire il programma in background da linea di comando
    - Utilizzare il comando **ps** per conoscere i PID dei processi di cui effettuare il debug

GDB (con emacs)

- A livello di GDB occorre
  - Eseguire una o più istanze di GDB, una per ogni processo di cui effettuare il debug
    - Eseguire le istanze in modalità **super-user**
  - Fornire i comandi seguenti, in ciascuna istanza, prima del termina della sleep

```
attach <pid>
file <nome dell'eseguibile>
n
```

Per connettere il GDB al processo di cui effettuare il debug

Per caricare il file eseguibile, il codice e la relativa tabella dei simboli

Una serie di n (next) per superare la sleep e giungere alla prima istruzione che la segue

Esempio

Programma da debuggare

```
int main () {
  pid_t pid;
  pid = fork();
  if (pid == 0){
    sleep (20);
    fprintf(stdout, "Child : PIDreturned=%d ", pid);
    fprintf (stdout, "PID=%d; My parent PID=%d\n",
      getpid(), getppid());
  } else {
    sleep (20);
    fprintf(stdout, "Father: PIDreturned=%d ", pid);
    fprintf (stdout, "PID=%d; My parent PID=%d\n",
      getpid(), getppid());
  }
  return 0;
}
```

Occorre effettuare l'attach del processo entro questo tempo

Esempio

Comandi da shell

```
quer@quer-VB:~/CLionProjects$ make
gcc -Wall -g -o a e03-fork.c
quer@quer-VB:~/CLionProjects$ ./a &
[8] 33960
quer@quer-VB:~/CLionProjects$ ps
    PID TTY          TIME CMD
   2551 pts/2    00:00:00 bash
  33752 pts/2    00:00:02 emacs
  33960 pts/2    00:00:00 a
  33961 pts/2    00:00:00 a
  33962 pts/2    00:00:00 ps
quer@quer-VB:~/CLionProjects$
```

Esempio

Comandi da GDB

Debug figlio

Debug padre

emacs@quer-VB

File Edit Options Buffers Tools Gud Complete In/Out Signals Help

p p* Run Continue Next Line Step Li

```
  if (pid == 0){
    // Child
    //pause();
    sleep (20);
    fprintf(stdout, "Child : PIDreturned=%d    ", pid);
    fprintf (stdout, "PID=%d; My parent PID=%d\n",
      getpid(), getppid());
```

-(DOS)--- e03-fork.c 29% L19 (C/*l Abbrev)

```
Attaching to program: /home/quer/CLionProjects/a, proc
ptrace: No such process.
(gdb) attach 33961
Attaching to program: /home/quer/CLionProjects/a, proc
(gdb) `/home/quer/CLionProjects/a' has changed; re-rea
Reading symbols from /lib/x86_64-linux-gnu/libc.so.6..
Reading symbols from /usr/lib/debug/.build-id/18/78e6b
$e6d1dee.debug...
Reading symbols from /lib64/ld-linux-x86-64.so.2...
Reading symbols from /usr/lib/debug/.build-id/45/87364
$1c3a078.debug...
0x00007fd22d9ae1b4 in __GI___clock_nanosleep (clock_id
$@entry=0, flags=flags@entry=0, req=req@entry=0x7ffeb1e
$eb1e86c20) at ../sysdeps/unix/sysv/linux/clock_nanosle
78      ../sysdeps/unix/sysv/linux/clock_nanosleep.c:
(gdb) file a
Reading symbols from a...
(gdb) n
80      in ../sysdeps/unix/sysv/linux/clock_nanosleep.
(gdb)
117     in ../sysdeps/unix/sysv/linux/clock_nanosleep.
(gdb)
__GI___nanosleep (requested_time=requested_time@entry=
$remaining@entry=0x7ffeb1e86c20) at nanosleep.c:28
28      nanosleep.c: No such file or directory.
U:**-  *gud*          71% L97    (Debugger:run [end-st
```

emacs@quer-VB

File Edit Options Buffers Tools Gud Complete In/Out Signals Help

p p* Run Continue Next Line Step Line

```
  } else {
    // Father
    //pause();
    sleep (20);
    fprintf(stdout, "Father: PIDreturned=%d ", pid);
    fprintf (stdout, "PID=%d; My parent PID=%d\n",
      getpid(), getppid());
  }
```

-(DOS)--- e03-fork.c 64% L27 (C/*l Abbrev)

```
$@entry=0, flags=flags@entry=0, req=req@entry=0x7ffeb1e86c20, rem=rem@entry=0x7ff
$eb1e86c20) at ../sysdeps/unix/sysv/linux/clock_nanosleep.c:78
78      ../sysdeps/unix/sysv/linux/clock_nanosleep.c: No such file or directory.
(gdb) file a
Reading symbols from a...
(gdb) n
80      in ../sysdeps/unix/sysv/linux/clock_nanosleep.c
(gdb)
117     in ../sysdeps/unix/sysv/linux/clock_nanosleep.c
(gdb)
__GI___nanosleep (requested_time=requested_time@entry=0x7ffeb1e86c20, remaining=
$remaining@entry=0x7ffeb1e86c20) at nanosleep.c:28
28      nanosleep.c: No such file or directory.
(gdb)
__sleep (seconds=0) at ../sysdeps/posix/sleep.c:62
62      ../sysdeps/posix/sleep.c: No such file or directory.
(gdb)
64      in ../sysdeps/posix/sleep.c
(gdb)
```

U:**- *gud* Bot L334 (Debugger:run [end-stepping-range])

CLion

- In Clion è possibile
  - Accedere alla finestra GDB
  - Digitare i comandi

```
set follow-fork-mode child
set detach-on-fork off
```

The new process is debugged after a fork. The parent process runs unimpeded.

Both processes will be held under the control of gdb. One process (child or parent) is debugged as usual, while the other is held suspended.

  - Il debugger effettuerà il debug del processo figlio invece del processo padre

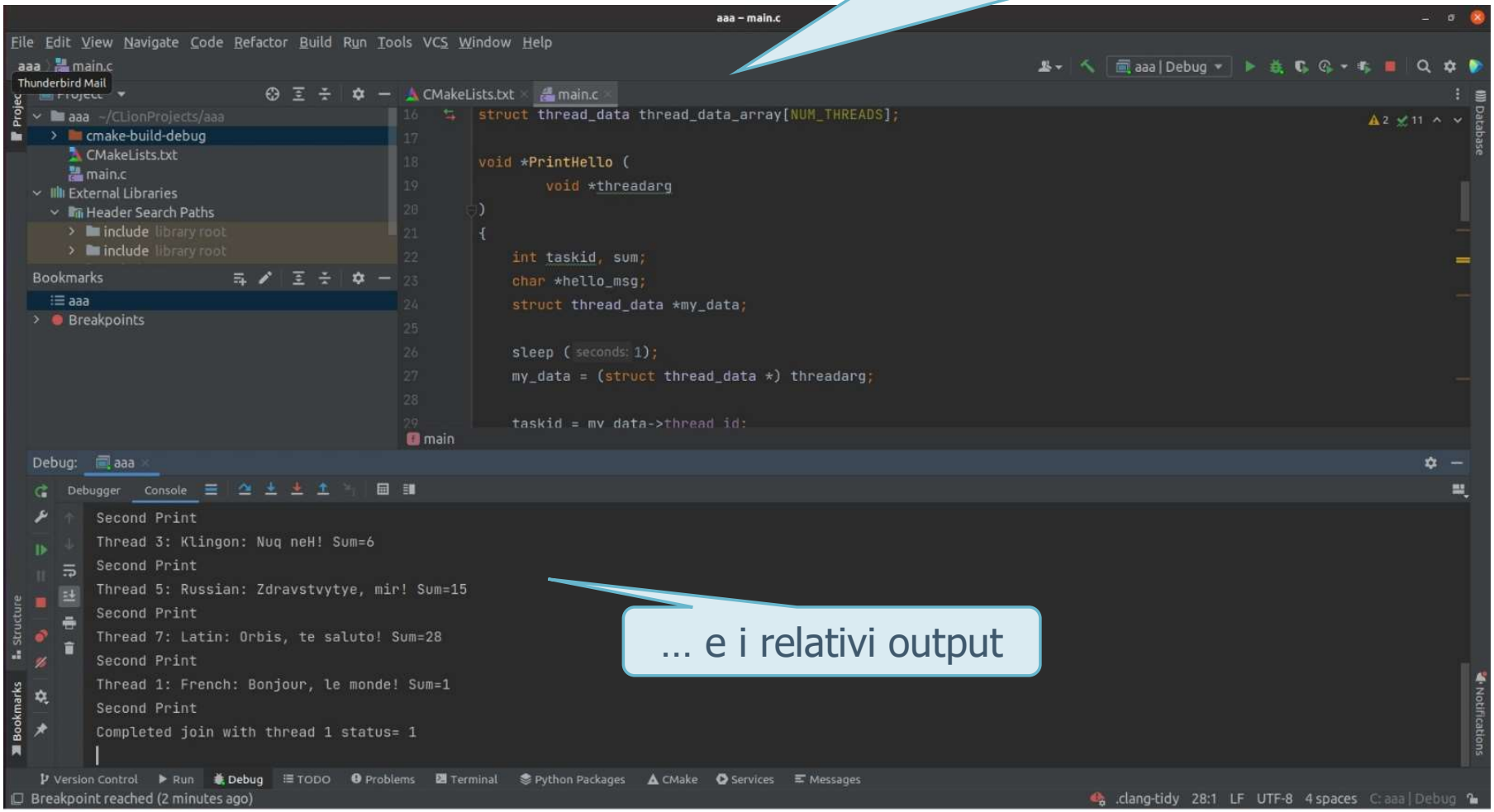
Thread e debug

- Il debug di programmi multi-thread è teoricamente più semplice
  - Inserendo un break-point nei vari thread il debugger si blocca automaticamente su di essi e quindi su ogni thread che si desidera controllare
- Sfortunatamente all'aumentare del numero di thread, il context switching rende l'operazione logicamente complessa
  - Alcuni IDE permettono di seguire i thread separatamente
    - Per esempio, in CLion è presente l'opzione **parallel stacks** in altri ambienti il comando **thread-step-over**

Debug di thread

Effetto debug in CLion

La presenza di un break point permette di seguire i vari thread ...



... e i relativi output

Debug di processi e thread

- Per maggiori informazioni effettuare delle ricerche WEB con parole chiave
  - Debug, debug multi process, debug multi-threaded, debug and fork, etc.