Introduction to GDB

Lezione 9

(taken from Owen HSU material)

Outline

- What's GDB ?
- Why GDB ?
- Basic GDB Commands
- Starting up GDB
- Examples

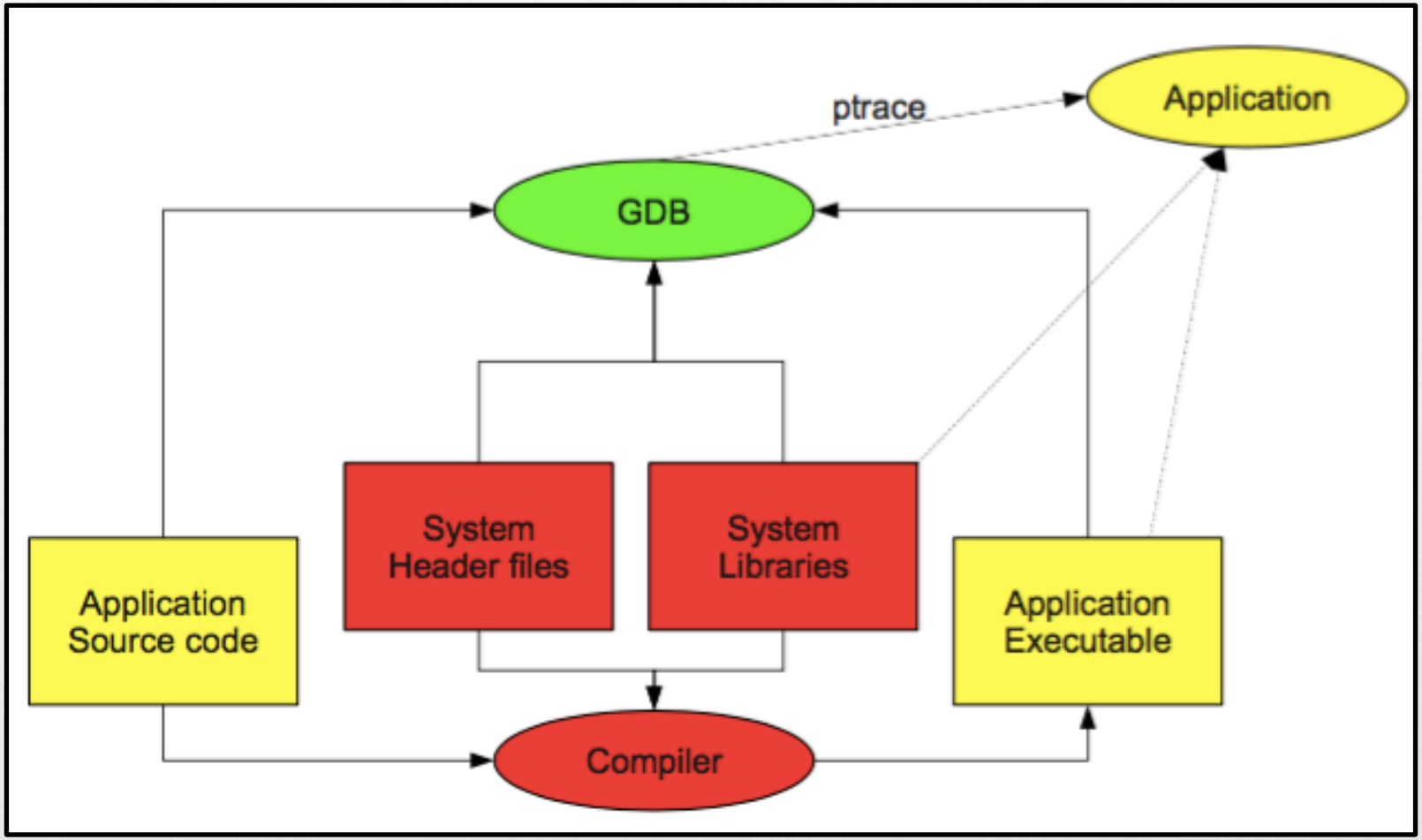
What's GDB ?

- GNU Debugger
- A text debugger for several language, including C/C++

Features

- An interactive shell
- Learn once, debug anywhere

Running a program under GDB



Where

`https://www.gnu.org/software/gdb/`

GDB: The GNU Project Debugger

[Bugs] [GDB Maintainers] [contributing] [current git] [documentation] [download] [home] [irc] [links] [mailing lists] [news] [schedule] [song] [wiki]

GDB: The GNU Project Debugger

What is GDB?

GDB, the GNU Project debugger, allows you to see what is going on `inside' another program while it executes -- or what another program was doing at the moment it crashed.

GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act:

- *Start your program, specifying anything that might affect its behavior.*
- *Make your program stop on specified conditions.*
- *Examine what has happened, when your program has stopped.*
- *Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.*

The program being debugged can be written in Ada, C, C++, Objective-C, Pascal (and many other languages). Those programs might be executing on the same machine as GDB (native) or on another machine (remote). GDB can run on most popular UNIX and Microsoft Windows variants.

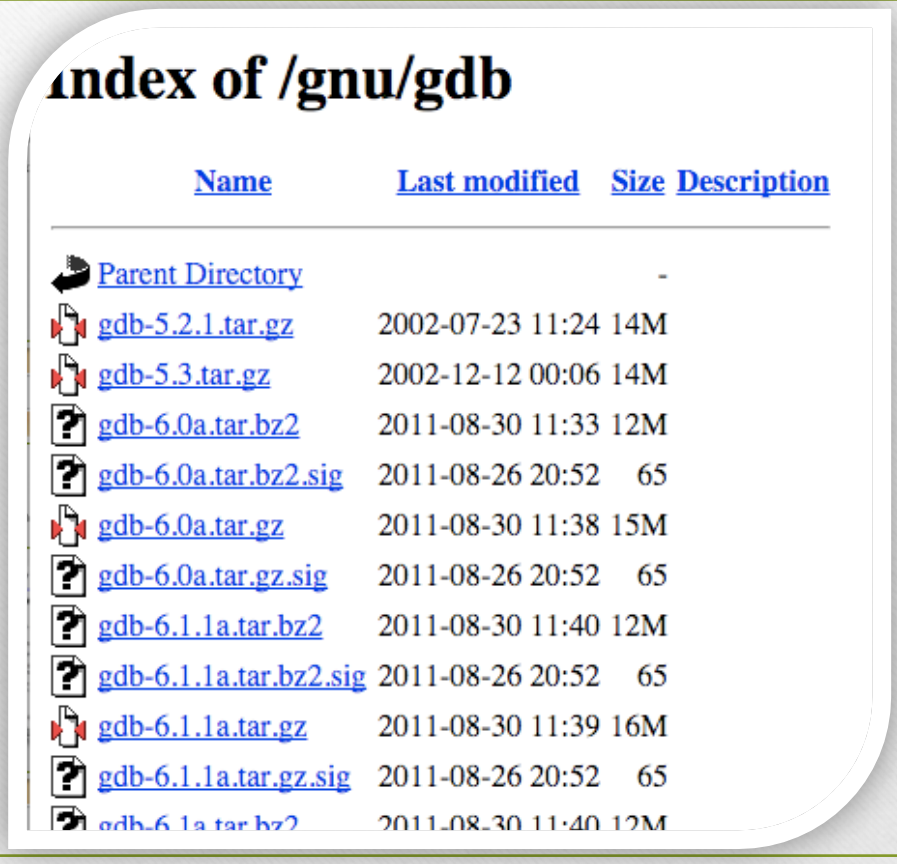
GDB version 7.10.1

Version 7.10.1 of GDB, the GNU Debugger, is now available for download. See the ANNOUNCEMENT for details including changes in this release.

...ata list (PROBLEMS) and documentation are also available.

Download

https://www.gnu.org/software/gdb/



ındex of /gnu/gdb

Name	Last modified	Size	Description
Parent Directory		-	
gdb-5.2.1.tar.gz	2002-07-23 11:24	14M	
gdb-5.3.tar.gz	2002-12-12 00:06	14M	
gdb-6.0a.tar.bz2	2011-08-30 11:33	12M	
gdb-6.0a.tar.bz2.sig	2011-08-26 20:52	65	
gdb-6.0a.tar.gz	2011-08-30 11:38	15M	
gdb-6.0a.tar.gz.sig	2011-08-26 20:52	65	
gdb-6.1.1a.tar.bz2	2011-08-30 11:40	12M	
gdb-6.1.1a.tar.bz2.sig	2011-08-26 20:52	65	
gdb-6.1.1a.tar.gz	2011-08-30 11:39	16M	
gdb-6.1.1a.tar.gz.sig	2011-08-26 20:52	65	
gdb-6.1a.tar.bz2	2011-08-30 11:40	12M	

Why GDB ?

Programmers make bug than debug

- `printf("===start debug===")`
- `printf("var: %d\n", var)`
- `printf("===end debug===" )`

GDB Helps Us to Find Out

- Watch or modify the variables in runtime
- Why programs fail or abort ?
- Current state of program
- Change the executing flow dynamically

Breakpoints

- **“break location”** will stop your program just before it executes any code associated with location.
- **“tbreak location”** enables a breakpoint only for a single stop
- **“condition bnum expression”** causes GDB to only stop at the breakpoint if the expression evalutes to non-zero.

Watchpoints (1/2)

- **"watch expression"** will stop your program whenever the value of expression changes.
  - GDB will use hardware support to implement watchpoints efficiently if possible; otherwise GDB will continue silently single-stepping until the value of expression has changed.
  - The whole expression is constantly re-evaluated; for example "watch p->x" will trigger both if the value of the "x" member of structure "p" currently points changes, **and** if "p" is reassigned to point to another structure (if that structure's "x" member holds a different value).

Watchpoints (2/2)

- **"watch expression"** will stop your program whenever the value of expression changes.
  - Once a variable refered to by expression goes out of scope, the watchpoint is disabled.
  - Use "watch -location expression" to instead evaluate expression only once, determine its current address, and stop your program only if the value at this address changes.

Continuing execution

- "**Continuing and stepping**"
  - "**continue**" resumes program execution.
  - "**step**" or "**next**" single-step to the next source line (stepping into/over function calls).
  - "**finish**" continues until the current function scope returns.
  - "**until**" continues until a location in the current function scope is reached (or it returns).
  - "**advance**" continues until a location is reached for the first time.

Continuing execution

- "**Skipping over functions and files**"
  - "**skip function**" steps over any invocation of function, even when using "step". (Useful for nested function calls.)
  - "**skip filename**" steps over all functions in the given file.

Inspecting program state

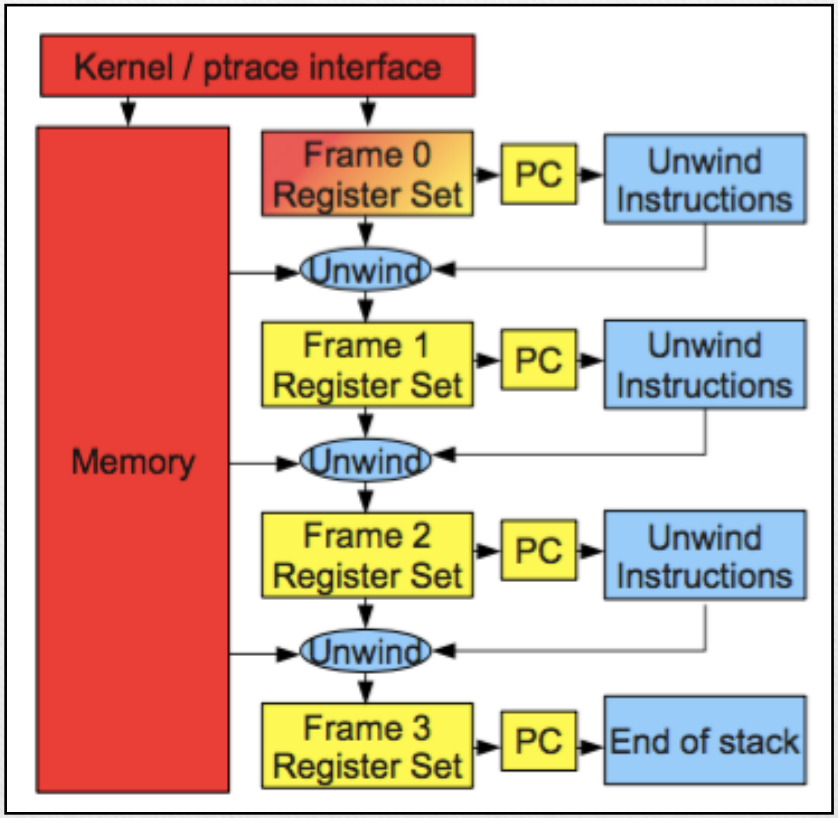
- "**Examining source files**"
  - "**list**" prints lines from a source file.
  - "**search [regexp]**" searches a source file.
  - "**directory**" specified directories to be searched for source files.

Inspecting program state

- "**Source and machine code**"
  - "**info line linespec**" shows which addresses correspond to a source line
  - "**disassemble**" shows machine code.
  - Use "**set disassemble-next-line on**" to automatically disassemble the current source line whenever GDB stops.

Inspecting program state

- "**Examining the stack**"
  - GDB will use current register values and memory contents to re-construct the "call stack" - the series of function invocations that led to the current location.



HelloWorld Example

```
01  #include <stdio.h>
02  void func(char *pMem) {
03    printf("- func: %p\n\n", pMem);
04  }
03
05  const char *szHello = "Hello World";
06  int main(int argc, char *argv[])
07  {
08    printf("\n%s\n\n", szHello);
09
10    int i;
11    for (i=0; i<argc; i++) {
12      printf("argv[%d]\n", i);
13      printf("- main: %s\n", argv[i]);
14      func(argv[i]);
15    }
16
17    return 0;
18  }
```

Compile hello.c

display all messages

```
# gcc -Wall hello.c -o hello
# ./hello 123 abc

Hello World

argv[0]
- main: ./hello
- func: 0xbfbf099b

argv[1]
- main: 123
- func: 0xbfbf09a3

argv[2]
- main: abc
- func: 0xbfbf09a7
```

Starting up GDB

add debugging information to hello

```
# gcc -Wall -g hello.c -o hello
# gdb hello
GNU gdb Fedora (6.8-37.el5)
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute
it.
There is NO WARRANTY, to the extent permitted by law.  Type
"show copying"
and "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu"...
(gdb)
```

Basic GDB Commands

Command		Explanation
run	[args]	To run the program in gdb
start	[args]	Start (and automatically set a breakpoint at main)
break	[line/function]	Breakpoint on a line or function
break	[condition]	Breakpoint When the interrupt condition is established
continue		Continues running the program until the next breakpoint or error
step		Runs the next line of the program
next		Like step, but it does not step into functions
list	[line/function]	Code list
print	[exp]	Prints the current value of the expression
print	[var=val]	Prints the current value of the variable
backtrace		Displays the current stack Status
help	[subcommand]	help

```
This GDB was configured as "i386-redhat-linux-gnu"...
(gdb) break main
Breakpoint 1 at 0x80483b3: file hello.c, line 9.
(gdb) run 123 abc
Starting program: /root/gdb/hello

Breakpoint 1, main (argc=3, argv=0xbffe6a24) at hello.c:9
9           printf("\n%s\n\n", szHello);
(gdb) list
4         }
5
6         char *szHello = "Hello World";
7         int main(int argc, char *argv[])
8         {
9           printf("\n%s\n\n", szHello);
10
11          int i;
12          for (i=0; i<argc; i++) {
13            printf("argv[%d]\n", i);
(gdb) break 14
Breakpoint 2 at 0x80483e4: file hello.c, line 14.
(gdb) continue
Continuing.

Hello World

argv[0]

Breakpoint 2, main (argc=3, argv=0xbffe6a24) at hello.c:14
14            printf("- main: %s\n", argv[i]);
```

set breakpoint at main

run with @parm 123 abc

list code

set breakpoint at 14th line

continue to breakpoint

```
(gdb) next
- main: /root/gdb/hello
15          func(argv[i]);
(gdb) step
func (pMem=0xbffe797e "/root/gdb/hello") at hello.c:3
3         printf("- func: %x\n\n", pMem);
(gdb) backtrace
#0  func (pMem=0xbffe797e "/root/gdb/hello") at
#1  0x08048418 in main (argc=3, argv=0xbffe6a24) at hello.c:15
(gdb) list
1       #include <stdio.h>
2       void func(char *pMem) {
3         printf("- func: %x\n\n", pMem);
4       }
5
6       char *szHello = "Hello World";
7       int main(int argc, char *argv[])
8       {
9         printf("\n%s\n\n", szHello);
10
(gdb) print pMem
$1 = 0xbffe797e "/root/gdb/hello"
(gdb) continue
Continuing.
- func: 0xbffe797e

argv[1]

Breakpoint 2, main (argc=3, argv=0xbffe6a24) at hello.c:14
14          printf("- main: %s\n", argv[i]);
```

execute to next statement

step into to next statement

print backtrace of all stack frames

print the value of pMem

```
(gdb) next
- main: 123
15          func(argv[i]);
(gdb) step
func (pMem=0xbffe798e "123") at hello.c:3
3         printf("- func: %x\n\n", pMem);
(gdb) print pMem
$2 = 0xbffe798e "123"
(gdb) print *pMem
$3 = 49 '1'
(gdb) continue
Continuing.
- func: 0xbffe798e
argv[2]

Breakpoint 2, main (argc=3, argv=0xbffe6a24) at hello.c:14
14          printf("- main: %s\n", argv[i]);
(gdb) next
- main: abc
15          func(argv[i]);
(gdb) next
- func: 0xbffe7992

12        for (i=0; i<argc; i++) {
(gdb) continue
Continuing.

Program exited normally.
```

print the value of *pMem

When the program is not wrong, GDB is sad ...

- Wake up, we have a lot of bug in the code
- And people do not often find bug in a simple way

For example, we have a bug into the code of a library

- `project`
  - `foo.c`
  - `bar.c`
  - `bar.h`

foo.c

```
01   #include "bar.h"
02   int foo = 3;
03   int main()
04   {
05      foo = 8;
06      bar(&foo);
07
08      return 0;
09   }
```

bar.c

```
01   #include <stdlib.h>
02   void bar(int  *val){
03      *val  =  11;
04      val  =  NULL;
05      *val  =  17;
06   }
```

bar.h

```
01   void bar(int*);
```

Mixed together to make it foobar

```
# gcc -Wall -g -fPIC -shared bar.c -o libbar.so
# gcc -Wall -g foo.c ./libbar.so -o foobar
# ./foobar
Segmentation fault

# gdb foobar
```

Info

Command	Explanation
info breakpoints	View current breakpoints
info watchpoints	Check the current watchpoint
info locals	See all current local variables
info registers	View current value register (the part)
info frame	View stack frames currently used
info stack	View program stack position
info proc	View program loaded itinerary (process)
info thread	inquire about existing threads
info source	Lists source files mentioned in loaded symbols
Info shared	View shared library information

```
(gdb) b main
Breakpoint 1 at 0x8048455: file foo.c, line 5.
(gdb) disp foo
(gdb) r
Starting program: /root/debug/a.out

Breakpoint 1, main () at foo.c:5
5           foo = 8;
1: foo = 3
(gdb) i sha
From        To          Syms Read   Shared Object Library
0x006fa7f0  0x0070fe7f  Yes         /lib/ld-linux.so.2
0x003442b0  0x003443f4  Yes         ./libbar.so
0x00732c80  0x0082db30  Yes         /lib/libc.so.6
(gdb) n
6           bar(&foo);
1: foo = 8
(gdb) s
bar (val=0x8049658) at bar.c:3
3           *val = 11;
1: foo = 8
(gdb) i s
#0  bar (val=0x8049658) at bar.c:3
#1  0x0804846b in main () at foo.c:6
```

display variable automatically

show loading shared library info

show stack info

```
(gdb) l
1       #include <stdlib.h>
2       void bar(int *val){
3         *val = 11;
4         val = NULL;
5         *val = 17;
6       }
(gdb) disp val
2: val = (int *) 0x8049658
(gdb) s
3         val = NULL;
2: val = (int *) 0x8049658
1: foo = 11
(gdb) s
4         *val = 17;
2: val = (int *) 0x0
1: foo = 11
(gdb) s

Program received signal SIGSEGV, Segmentation fault.
0x003443b2 in bar (val=0x0) at bar.c:4
4         *val = 17;
2: val = (int *) 0x0
1: foo = 11
```

Pointer is NULL

Use of NULL pointer: error !!

```
(gdb) b bar
Breakpoint 2 at 0x8f039f: file bar.c, line 3.
(gdb) i b
Num     Type           Disp Enb Address    Wh
1       breakpoint     keep y   0x08048455 in main at foo.c:5
        breakpoint already hit 1 time
2       breakpoint     keep y   0x008f039f in bar at bar.c:3
(gdb) d 1
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /root/gdb/foobar

Breakpoint 2, bar (val=0x8049658) at bar.c:3
3           *val = 11;
(gdb) s
4           val = NULL;
(gdb) shell vim bar.c
(gdb) shell gcc -Wall -g -fPIC -shared bar.c -o libbar.so
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /root/gdb/foobar

Breakpoint 2, bar (val=0x8049658) at bar.c:3
3           *val = 11;
(gdb) c
Continuing.

Program exited normally.
```

set breakpoint at bar( )

show all breakpoints

execute shell command without exit of gdb

Watchpoint

Command		Explanation
watch	[exp]	You can use a watchpoint to stop execution whenever the value of an expression changes
delete	[n]	Delete breakpoint
nexti		Execute one machine instruction, but if it is a function call, proceed until the function returns
stepi		Execute one machine instruction, then stop and return to the debugger
disassemble	[addr]	Disassembles a specified function or a function fragment

```
(gdb) wa foo
Hardware watchpoint 1: foo
(gdb) r
Starting program: /root/gdb/foobar
Hardware watchpoint 1: foo

Old value = 3
New value = 8
main () at foo.c:6
6           bar(&foo);
(gdb) si
0x08048466      6           bar(&foo);
(gdb) si
0x08048340 in bar@plt ()
(gdb) i s
#0  0x08048340 in bar@plt ()
#1  0x0804846b in main () at foo.c:6
(gdb) i f
Stack level 0, frame at 0xbfa76670:
 eip = 0x8048340 in bar@plt; saved eip 0x804846b
 called by frame at 0xbfa76680
 Arglist at 0xbfa76668, args:
 Locals at 0xbfa76668, Previous frame's sp is 0xbfa76670
 Saved registers:
  eip at 0xbfa7666c
```

set watchpoint to foo

step one instruction exactly

show stack frame info

```
(gdb) s
Single stepping until exit from function bar@plt,
which has no line number information.
0x00c6039c in bar () from ./libbar.so
(gdb) i s
#0  0x00c6039c in bar () from ./libbar.so
#1  0x0804846b in main () at foo.c:6
(gdb) i f
Stack level 0, frame at 0xbfa76670:
 eip = 0xc6039c in bar; saved eip 0x804846b
 called by frame at 0xbfa76680
 Arglist at 0xbfa76668, args:
 Locals at 0xbfa76668, Previous frame's sp is 0xbfa76670
 Saved registers:
  eip at 0xbfa7666c
(gdb) si
0x00c6039d in bar () from ./libbar.so
(gdb) si
0x00c6039f in bar () from ./libbar.so
(gdb) si
0x00c603a2 in bar () from ./libbar.so
(gdb) si
0x00c603a8 in bar () from ./libbar.so
(gdb) si
0x00c603af in bar () from ./libbar.so
(gdb) si
0x00c603b2 in bar () from ./libbar.so
```

Leaving the bar @ plt, enter ./libbar.so

Call bar () in main

```
(gdb) si

Program received signal SIGSEGV, Segmentation fault.
0x00c603b2 in bar () from ./libbar.so
(gdb) disas 0x00c603b2
Dump of assembler code for function bar:
0x00c6039c <bar+0>:     push   %ebp
0x00c6039d <bar+1>:     mov    %esp,%ebp
0x00c6039f <bar+3>:     mov    0x8(%ebp),%eax
0x00c603a2 <bar+6>:     movl   $0xb,(%eax)
0x00c603a8 <bar+12>:    movl   $0x0,0x8(%ebp)
0x00c603af <bar+19>:    mov    0x8(%ebp),%ea
0x00c603b2 <bar+22>:    movl   $0x11,(%eax)
0x00c603b8 <bar+28>:    pop    %ebp
0x00c603b9 <bar+29>:    ret
End of assembler dump.

(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /root/gdb/foobar
Hardware watchpoint 1: foo

Old value = 3
New value = 8
main () at foo.c:6
6           bar(&foo);
```

disassemble the section

Segmentation fault occurred when the 17 into eax

```
(gdb) s
Hardware watchpoint 1: foo

Old value = 8
New value = 11
0x005cb3a8 in bar () from ./libbar.so
(gdb) si
0x005cb3af in bar () from ./libbar.so
(gdb) si
0x005cb3b2 in bar () from ./libbar.so
(gdb) si

Program received signal SIGSEGV, Segmentation fault.
0x005cb3b2 in bar () from ./libbar.so
(gdb) disas 0x005cb3b2
Dump of assembler code for function bar:
0x005cb39c <bar+0>:     push   %ebp
0x005cb39d <bar+1>:     mov    %esp,%ebp
0x005cb39f <bar+3>:     mov    0x8(%ebp),%eax
0x005cb3a2 <bar+6>:     movl   $0xb,(%eax)
0x005cb3a8 <bar+12>:    movl   $0x0,0x8(%ebp)
0x005cb3af <bar+19>:    mov    0x8(%ebp),%eax
0x005cb3b2 <bar+22>:    movl   $0x11,(%eax)
0x005cb3b8 <bar+28>:    pop    %ebp
0x005cb3b9 <bar+29>:    ret
End of assembler dump.
```

When foo changes will automatically display,

Even when change is also displayed in the library

```
(gdb) shell objdump -d libbar.so|less
(gdb) shell vim libbar.so

:%!xxd

...
0000390: d283 c404 5b5d c38b 1c24 c390 5589 e58b  ....[]...$..U...
00003a0: 4508 c700 0b00 0000 c745 0800 0000 008b  E........E......
00003b0: 4508 c700 1100 0000 5dc3 9090 9090 9090  E.......].......
...

...
0000390: d283 c404 5b5d c38b 1c24 c390 5589 e58b  ....[]...$..U...
00003a0: 4508 c700 0b00 0000 9090 9090 9090 908b  E........E......
00003b0: 4508 c700 1100 0000 5dc3 9090 9090 9090  E.......].......
...
:%!xxd -r
:wq!

(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /root/foo/foobar

Program exited normally.
```

Had to manually modify the content libbar.so

GDB Only this?

GDB interactive learning method

linklist.c

```
01  typedef struct node node;
02  struct node {
03    int data;
04    node *next;
05  };
06
07  int main(void)
08  {
09    node *p, *q, *r;
10    p->next = q;
11
12    return 0;
13  }
```

GDB interactive learning method

```
# gcc -Wall -g linklist.c -o linklist
  linklist.c: In function 'main':
  linklist.c:9: warning: unused variable 'r'
  # ./linklist
  Segmentation fault
```

Ask GDB seniors

```
(gdb) b main
Breakpoint 1 at 0x8048365: file linklist.c, line 10.
(gdb) r
Starting program: /root/gdb/linklist

Breakpoint 1, main () at linklist.c:10
10          p->next = q;
(gdb) c
Continuing.

Program received signal SIGSEGV, Segmentation fault.
0x0804836b in main () at linklist.c:10
10          p->next = q;
```

The whole world of men will commit wrong ...

Esercizi

- Ho due tabelle. Nella prima ho un elenco di alunni con le rispettive matricole; nella seconda ho lo stesso elenco di alunni ma con voti e materie. Voglio avere una sola tabella che comprende elenco di alunni, matricole, voti e materie.

Prima tabella (tabella1.txt)

```
Luigi    45695
Nicola   45696
Eugenio  45697
Luisa    45698
Gino     45699
Emma     45700
Rosy     45701
Nino     45702
Marco    45703
Ennio    45704
Paola    45705
```

Seconda tabella(tabella2.txt)

```
Luigi    27   matematica
Nicola   28   fisica
Eugenio  18   fisica
Luisa    21   matematica
Gino     23   matematica
Emma     24   fisica
Rosy     23   matematica
Nino     29   matematica
Marco    18   fisica
Ennio    25   fisica
Paola    30   fisica
```

Esercizi

```
#!/usr/bin/awk -f

BEGIN {
          print "\n\t.............START.....................\n"
          FORMAT="\t%-12s%-12s%-8s%s\n"
          printf FORMAT,"ALUNNI","MATRICOLE","VOTI","MATERIE"
   }
    {
           if (FILENAME == "tabella1.txt") {
                matricole[$1] = $2
           }
           if (FILENAME == "tabella2.txt") {
               printf FORMAT, $1,matricole[$1],$2,$3
           }
    }
END {
print "\n\t.................END.................\n"
}
```

Esercizi

- Ho un file di testo in cui ricorrono parole separate da spazi (ogni riga ha un numero indipendente di parole).
- Vogliamo visualizzare le parole doppie (e relativo numero di riga): quelle che ricorrono due volte di seguito.
- Bisogna considerare anche come parole doppie che ricorrono come ultima parola di una riga e prima parola della riga successiva

casa dolce **casa**
casa dolce dolce

casa at line 2
dolce at line 2

Esercizi

```
NF > 0  {                                  NB: all'inizio lastword vale ""
    if ($1 == lastword)                    confronta $1 con ult. parola riga preced.
       printf "double %s at line %d\n", $1, NR
    for (i=2; i<=NF; i++)
       if ($i == $(i-1))
          printf "double %s at line %d\n", $i, NR
    lastword = $NF                         aggiorna lastword
}
```