

Guida alla Compilazione, Disassemblaggio e Analisi di un Programma C su Linux

Laboratorio di Reti e Sistemi Distributi @ UniME

February 26, 2025

1 Introduzione

Questa guida descrive i passaggi per compilare, disassemblare e analizzare un programma C su Linux utilizzando strumenti come `gcc`, `objdump`, `gdb`, `strace`, `ltrace`, `valgrind`, `readelf`, `nm` e `ldd`.

2 Compilazione di un Programma C

2.1 Scrivere il programma C

Creare un file `programma.c`:

```
1 #include <stdio.h>
2
3 int main() {
4     printf("Ciao, mondo!\n");
5     return 0;
6 }
```

2.2 Compilare con gcc

Eseguire il comando:

```
1 gcc -o programma programma.c
```

2.3 Eseguire il programma

Eseguire il programma compilato:

```
1 ./programma
```

2.4 Output di esempio

```
1 Ciao, mondo!
```

3 Disassemblaggio di un Programma

3.1 Usare objdump

Per disassemblare il programma:

```
1 objdump -d programma
```

3.2 Output di esempio

```
1 0000000000401136 <main>:
2   401136:      55                push    %rbp
3   401137:      48 89 e5          mov     %rsp,%rbp
4   40113a:      48 8d 3d c3 0e 00 00 lea     0xec3(%rip),%rdi
5   401141:      e8 ea fe ff ff    callq   401030 <puts@plt>
6   401146:      b8 00 00 00 00     mov     $0x0,%eax
7   40114b:      5d                pop     %rbp
8   40114c:      c3                retq
```

3.3 Usare gdb

Avviare gdb e disassemblare la funzione main:

```
1 gdb programma
2 (gdb) disassemble main
```

3.4 Output di esempio

```
1 Dump of assembler code for function main:
2   0x0000000000401136 <+0>:      push    %rbp
3   0x0000000000401137 <+1>:      mov     %rsp,%rbp
4   0x000000000040113a <+4>:      lea     0xec3(%rip),%rdi
5   0x0000000000401141 <+11>:     callq   0x401030 <puts@plt>
6   0x0000000000401146 <+16>:     mov     $0x0,%eax
7   0x000000000040114b <+21>:     pop     %rbp
8   0x000000000040114c <+22>:     retq
9 End of assembler dump.
```

4 Analisi di un Programma

4.1 Analisi con gdb

Avviare gdb, impostare un breakpoint ed eseguire il programma:

```
1 gdb programma
2 (gdb) break main
3 (gdb) run
4 (gdb) next
5 (gdb) print $rax
6 (gdb) quit
```

4.2 Output di esempio

```
1 Breakpoint 1 at 0x401136
2 (gdb) run
3 Starting program: /path/to/programma
4
5 Breakpoint 1, 0x0000000000401136 in main ()
6 (gdb) next
7 Ciao, mondo!
8 (gdb) print $rax
9 $1 = 0
10 (gdb) quit
```

4.3 Analisi con strace

Tracciare le chiamate di sistema:

```
1 strace ./programma
```

4.4 Output di esempio

```
1 execve("./programma", ["/programma"], 0x7ffd8a1b8e80 /* 54 vars */) = 0
2 brk(NULL) = 0x55a1a2b2e000
3 access("/etc/ld.so.preload", R_OK) = -1 ENOENT (File o directory non esistente)
4 openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
5 ...
```

4.5 Analisi con ltrace

Tracciare le chiamate a funzioni di libreria:

```
1 ltrace ./programma
```

4.6 Output di esempio

```
1 printf("Ciao, mondo!\n") = 13
```

4.7 Analisi con valgrind

Rilevare perdite di memoria:

```
1 valgrind --leak-check=full ./programma
```

4.8 Output di esempio

```
1 ==12345== HEAP SUMMARY:
2 ==12345==      in use at exit: 0 bytes in 0 blocks
3 ==12345==    total heap usage: 1 allocs, 1 frees, 1,024 bytes allocated
4 ==12345==
5 ==12345== All heap blocks were freed -- no leaks are possible
```

5 Analisi Avanzata

5.1 Analisi con readelf

Visualizzare informazioni sul file binario ELF:

```
1 readelf -h programma
2 readelf -S programma
3 readelf -s programma
```

5.2 Output di esempio

```
1 ELF Header:
2   Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
3   Class:                               ELF64
4   Data:                                  2's complement, little endian
5   Version:                             1 (current)
6   OS/ABI:                               UNIX - System V
7   ABI Version:                          0
8   Type:                                  EXEC (Executable file)
9   Machine:                              Advanced Micro Devices X86-64
10  Version:                              0x1
11  Entry point address:                   0x401030
12  Start of program headers:              64 (bytes into file)
13  Start of section headers:             13912 (bytes into file)
14  Flags:                                 0x0
15  Size of this header:                   64 (bytes)
16  Size of program headers:               56 (bytes)
17  Number of program headers:              11
18  Size of section headers:               64 (bytes)
19  Number of section headers:              30
20  Section header string table index:     29
```

5.3 Analisi con nm

Visualizzare i simboli nel file binario:

```
1 nm programma
```

5.4 Output di esempio

```
1 0000000000401136 T main
2 0000000000404030 R _IO_stdin_used
3                U puts@@GLIBC_2.2.5
```

5.5 Analisi con ldd

Visualizzare le librerie condivise:

```
1 ldd programma
```

5.6 Output di esempio

```
1 linux-vdso.so.1 (0x00007ffd8a1b8000)
2 libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f8c1a2b0000)
3 /lib64/ld-linux-x86-64.so.2 (0x00007f8c1a6b0000)
```