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`. Sono inclusi esempi di output per ciascun comando.

2 Compilazione di un Programma C

2.1 Scrivere il programma C

Creare un file `programma.c`:

```c
#include <stdio.h>

int main() {
    printf("Ciao, mondo!\n");
    return 0;
}
```

2.2 Compilare con `gcc`

Eseguire il comando:

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

2.3 Eseguire il programma

Eseguire il programma compilato:

```
./programma
```

2.4 Output di esempio

```
Ciao, mondo!
```

3 Disassemblaggio di un Programma

3.1 Usare `objdump`

Per disassemblare il programma:

```
objdump -d programma
```

3.2 Output di esempio

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

3.3 Usare gdb

Avviare `gdb` e disassemblare la funzione `main`:

```
gdb programma
(gdb) disassemble main
```

3.4 Output di esempio

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

4 Analisi di un Programma

4.1 Analisi con gdb

Avviare `gdb`, impostare un breakpoint ed eseguire il programma:

```
gdb programma
(gdb) break main
(gdb) run
(gdb) next
(gdb) print $rax
(gdb) quit
```

4.2 Output di esempio

```
Breakpoint 1 at 0x401136
(gdb) run
Starting program: /path/to/programma

Breakpoint 1, 0x0000000000401136 in main ()
(gdb) next
Ciao, mondo!
(gdb) print $rax
$1 = 0
(gdb) quit
```

4.3 Analisi con strace

Tracciare le chiamate di sistema:

```
strace ./programma
```

4.4 Output di esempio

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

4.5 Analisi con `ltrace`

Tracciare le chiamate a funzioni di libreria:

```
ltrace ./programma
```

4.6 Output di esempio

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

4.7 Analisi con `valgrind`

Rilevare perdite di memoria:

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

4.8 Output di esempio

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

5 Analisi Avanzata

5.1 Analisi con `readelf`

Visualizzare informazioni sul file binario ELF:

```
readelf -h programma
readelf -S programma
readelf -s programma
```

5.2 Output di esempio

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

5.3 Analisi con `nm`

Visualizzare i simboli nel file binario:

```
nm programma
```

5.4 Output di esempio

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

5.5 Analisi con `ldd`

Visualizzare le librerie condivise:

```
ldd programma
```

5.6 Output di esempio

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