Debian MIPS

Nicola Basilico, Jacopo Essenziale
Dipartimento di Informatica
Università degli Studi di Milano

Emulazione di una macchina Debian MIPS

Installazione dell'emulatore

Installazione di Qemu tramite `apt-get`:

```
sudo apt-get install -y qemu qemu-kvm libvirt-bin
```

Informazioni più dettagliate, incluse le istruzioni per più sistemi operativi, sono disponibili a questo link.

Configurazione e avvio

Scaricare l'archivio zip a questo link e scompattarlo in una directory locale. L'archivio contiene:

- un'immagine del kernel `vmlinux-3.2.0-4-4kc-malta`;
- un'immagine per il disco virtuale `debian_wheezy_mips_standard.qcow2`.

Avviare la macchina virtuale:

```
qemu-system-mips
    -M malta
    -kernel vmlinux-3.16.0-4-4kc-malta
    -m 512
    -hda debian_wheezy_mips_standard.qcow2
    -net nic
    -net user,hostfwd=tcp::20011-:22
    -append "root=/dev/sda1 console=tty0"
```

Tramite il comando riportato sopra andremo ad utilizzare il PC System Emulator di Qemu per MIPS (`qemu-system-mips`) per ottenere l'emulazione di una macchina con queste caratteristiche:

- processore con ISA MIPS32 (tra le più recenti), nello specifico il processore è il MIPS32 4Kc; si veda la lista completa processori MIPS, mentre con il comando `qemu-system-mips -cpu '?'` si veda la lista di tutti i processori MIPS emulabili; sul sistema emulato possiamo inoltre ottenere dettagli sulla CPU con il comando `cat /proc/cpuinfo`;
- scheda madre MIPS Malta;
- 512 megabyte di RAM;
- sistema operativo Debian 8 Jessie (Linux `debian-mips`, versione del kernel `3.16.51-2`, release `3.16.0-4-4kc-malta`);
- porta 22 della macchina emulata mappata sulla porta 20011 della macchina host, accesso `ssh` con nome utente e password "user".

Una volta avviata la macchina virtuale, possiamo aprire una shell via `ssh`:

```
ssh user@localhost -p 20011 # password: user
```

Da C ad Assembly

Si consideri il programma C riportato nel file *helloworld.c*.

```c
// helloworld.c
#include <stdio.h>
int main()
{
    printf("Hello World!\n");
    return 0;
}
```

Compiliamo il file usando `gcc` per ottenere un eseguibile:

```
gcc helloworld.c -o helloworld
```

Lanciamo l'eseguibile:

```
./helloworld # Hello World!
```

Per produrre l'eseguibile, il compilatore ha svolto (in ordine):

1) pre-processing: rimozione commenti, espansione di header e macro;
2) compilazione vera e propria: produce linguaggio assembly;
3) assemblaggio: produce linguaggio macchina;
4) linking: risoluzione dipendenze, produce un file eseguibile.

Possiamo richiedere a `gcc` di fermarsi dopo lo step di compilazione (2) e di produrre un file con il codice assembly ottenuto (che possiamo ispezionare):

```
gcc -S helloworld.c -o helloworld.s
```

Svolgiamo manualmente l'assemblaggio (3) usando `as`, l'assembler GNU:

```
as helloworld.s -o helloworld.o
```

Il file oggetto ottenuto (`helloworld.o`) contiene le istruzioni in linguaggio macchina, ma non è ancora eseguibile:

```
chmod +x helloworld.o
./helloworld # cannot execute binary file: Exec format error
```

Dobbiamo fare il linking usando `ld`, il linker GNU:

```
ld hello.o
    -o hello
    -dynamic-linker /lib/ld.so.1 /usr/lib/mips-linux-gnu/crt*.o
    -lc
```

- con l'opzione `-dynamic-linker` richiediamo a `ld` di utilizzare la libreria `ld.so.1` per eseguire il linking delle librerie a run-time (linking dinamico);
- le librerie `/usr/lib/mips-linux-gnu/crt*.o` contengono le routine di startup per il programma (e in particolare le definizioni per le label `__start` e `__init`) che chiameranno il `main` del nostro programma e a cui restituiremo il controllo al termine;
- l'opzione `-lc` richiede il linking di `libc`, la libreria standard del C.

Il file `helloworld` è ora un *eseguibile*:

```
./helloworld # Hello World!
```

Da Assembly all'eseguibile

Scriviamo un programma assembly che stampi la stringa "Hello world!" e generiamo un file eseguibile da lanciare sulla macchina virtuale. Dobbiamo considerare due aspetti:

- `ld` si richiede una entry label di default con nome `__start`, quindi usiamo questa label al posto di `main` (usando `main` avremmo un warning) oppure, se usiamo `main`, dobbiamo ricordarci di linkare `ld.so.1`;
- le syscall non saranno quelle simulate da SPIM ma, ovviamente, le reali syscall esposte dal kernel Linux, dobbiamo capire come usarle.

Utilizzo delle syscall

Supponiamo di avere una stringa da stampare sullo standard output, necessitiamo quindi di una syscall simile a quella che in SPIM/MARS è `print_string`. Per capire come utilizzare *direttamente* le syscall del kernel Linux dobbiamo incrociare diverse informazioni.

Per prima cosa ispezioniamo la lista di tutte le syscall offerte dal sistema operativo. Lo facciamo lanciando questo comando sul sistema emulato:

```
man syscalls
```

Dalla lunga lista ci accorgiamo che esiste una syscall `write` che, nelle man pages ha 2 sezioni come indicato dal "(2)" dopo il nome della funzione. Esistono due `write`. Una è un comando della bash che implementa una utility per spedire messaggi ad altri utenti (si può invocare da terminale). La seconda è invece una syscall del sistema operativo. Dal manuale di `man` vediamo che le descrizioni inerenti alle syscall stanno nella sezione 2 della man page:

```
man man
#NAME
#       man - an interface to the on-line reference manuals
# [...]
# 1   Executable programs or shell commands
# 2   System calls (functions provided by the kernel)
# [...]
```

Per ottenere quindi la man page della *syscall* `write` (e non dell'*utility* `write`) dobbiamo richiamare la seconda sezione:

```
man write # manuale dell'utility write, non ci interessa
#NAME
#     write - send a message to another user
# [...]

man 2 write # manuale della syscall write, è quello che ci serve
#NAME
#       write - write to a file descriptor
```

Dalla man page ci accorgiamo che la signature della syscall è:

```
ssize_t write(int fd, const void *buf, size_t count);
```

Con la signature ricaviamo il tipo e l'ordine dei parametri che, nel codice assembly, tratteremo seguendo le convenzioni di chiamata a procedura del MIPS. Nello specifico:

- il primo parametro, da passare in `$a0`, è un intero che rappresenta il file descriptor; nei sistemi UNIX lo standard output è indicato con il valore `1`;
- il secondo parametro, da passare in `$a1`, è un puntatore al buffer che contiene i bytes da stampare; sarà quindi il base address della stringa che vogliamo stampare;
- il terzo parametro, da passare in `$a2`, è il numero di byte da stampare e cioè il numero di caratteri della nostra stringa (in C, `size_t` è un intero unsigned su non meno di 16 bit).

Per risalire al codice della syscall ispezioniamo (sempre sul sistema emulato) il file `/usr/include/mips-linux-gnu/asm/unistd.h`: la `write` ha codice 4004, valore che dovremo quindi caricare nel registro `$v0`.

```
    .data
hello:  .asciiz "\nHello, World!\n"
    .text
    .globl __start
__start:
    li $v0, 4004 # codice syscall write
    li $a0, 1 # standard output
    la $a1, hello # base address stringa
    li $a2, 15 # lunghezza stringa in bytes
    syscall
    li $v0, 4001 # exit
    syscall
```

Assemblaggio e linking

Una volta scritto il sorgente assembly (si veda il file `helloworld.asm`), lo assembliamo e lo linkiamo. In questo caso non stiamo usando nessuna libreria esterna e quindi non abbiamo dipendenze da segnalare al linker. Il linker non ha dipendenze esterne da risolvere, ma comunque ristruttura il file oggetto in un formato che il sistema operativo riconosce come eseguibile.

```
as helloworld.asm -o helloworld.o
ld helloworld.o -o helloworld
./helloworld # Hello World!
```

Utilizzo di libc

Nei sistemi UNIX l'uso diretto delle syscall non è l'unico modo per utilizzare le funzioni del kernel. La libreria standard del C `libc` fornisce una serie di funzioni per svolgere task comuni come la gestione della memoria e l'input/output. Costituiscono un livello di astrazione sopra le syscall del kernel.

Hello world con printf

Scriviamo un programma assembly che stampi "Hello world" usando `printf` (fornita da `libc`). Come primo step dobbiamo installare la documentazione di `libc` per avere a disposizione le man pages delle varie funzioni.

```
sudo apt-get install -y manpages-dev manpages-posix-dev
```

Dal manuale di `man` vediamo che le man pages delle funzioni di libreria stanno nella sezione 3:

```
man man
# [...]
# 1   Executable programs or shell commands
# 2   System calls (functions provided by the kernel)
# 3   Library calls (functions within program libraries)
# [...]
```

Visualizziamo la man page di `printf`:

```
man 3 printf
```

da cui vediamo che la signature della funzione è:

```
int printf(const char *format, ...);
```

Il primo parametro, che verrà passato in `$a0`, è la stringa di formato seguita dalla lista degli argomenti da passare nei registri `a1`, `a2`, etc. (come specificato dalla convezione di passaggio parametri in MIPS).

```
# helloworld.asm
    .data
hello:  .asciiz "Hello world with printf!\n"
    .text
    .globl main
main:
    subu   $sp, $sp, 4
    sw     $ra, 0($sp)
    la     $a0, hello # primo parametro: stringa di controllo
    jal    printf
    lw     $ra, 0($sp)
    addiu  $sp, $sp, 4
    jr     $ra
```

Nel codice asssembly riportato sopra notiamo i seguenti aspetti:

- utilizziamo la label `main` anzichè `__start`; quindi in fase di linking dovremo ricordarci di includere le librerie `crt*.o` che si occuperanno di assemblare, insieme al nostro programma, le procedure di init che invocheranno `main`;
- per lo stesso motivo del punto precedente, il programma termina con `jr $ra` anzichè con una syscall `exit`; il `main` è quindi un callee non foglia (per via di `jal printf`), deve quindi salvare il registro `$ra` sullo stack per preservarlo attraverso la chiamata a funzione;
- la label `printf` non è definita in questo sorgente; in fase di linking specificheremo di includere `libc`, il linker andrà quindi a risolvere la label con l'indirizzo a cui sta `printf`.

Assemblaggio:

```
as time.asm -o time.o
```

Linking:

```
ld time
    -o time.o
    -dynamic-linker /lib/ld.so.1 /usr/lib/mips-linux-gnu/crt*.o
    -lc
```

Esecuzione:

```
./time # Hello world with printf!
```

Esercizio

Si scriva un programma che chieda all'utente di inserire un carattere e stampi il tempo di risposta in secondi (definito come il tempo che intercorre tra il momento in cui viene fatta la richiesta all'utente e il momento in cui il carattere è inserito). Si faccia uso della syscall `time` e delle funzioni `printf` e `getchar` dalla libreria standard del C.