

Programmazione assembly in linux – breve introduzione

Publicato da **Ritz** il 18/05/2000

Livello **base**

Introduzione

Siete stufo del solito Windoze asm? Bene, è venuto il momento di migrare a Linux e provare a vedere da lì cosa si può fare di bello con questo linguaggio.

NOTA: molto di questo materiale lo potete trovare sul Linux Assembly HOWTO (prelevabile sul nostro sito): questo testo ne rappresenta un breve riassunto e riadattamento in italiano con alcune aggiunte.

Iniziamo

Questi tute dovrebbero iniziare con frasi del tipo "Linux ormai è un sistema operativo che rappresenta una valida alternativa..." bla bla bla, ma queste cose non penso che vi interessino troppo, visto che se state leggendo questo testo significa che usate già Linux e vi interessa iniziare a programmare in asm pure da qui, o quantomeno siete interessati all'argomento.

Per "introduzione" all'asm coding under Linux non intendo una spiegazione dalle *basi* dell'asm, cose tipo architettura dei processori x86 e loro set di istruzioni devono già essere nel drive hdabrain di chi legge.

Similmente, non starò qui a spiegare i motivi per cui in Linux bisognerebbe programmare in asm quando esso è basato (+ di Win) su altri linguaggi, C in primis. Btw, come per ogni SO la programmazione in asm al 90% non è una questione di necessità (salvo casi particolari naturalmente), bensì una scelta che si fa per passione e curiosità personale, certo usare l'asm per fare *ogni* cosa sarebbe alquanto scomodo e lungo, anche se i risultati finali sarebbero eccellenti.

Fatta questa (pallosa lo so:)) premessa, iniziamo a vedere un po' di cosette per programmare in asm sotto tale sistema.

First of all, ci serve un compilatore (a meno di non andare avanti a scrivere opcode con BIEW:)), quelli più usati sono 2, il GAS e il NASM.

In GAS (GNU Assembler), scaricabile da <ftp://ftp.varesearch.com/pub/support/hjl/binutils/> (cmq è compreso nel GCC), è stato progettato inizialmente come compilatore 32 bit sotto unix, il che comporta una particolarità: esso utilizza la sintassi AT&T, e non quella Intel, e qui possono essere dolori per uno che ha già programmato in asm sotto Win32 o DOS. Infatti, una delle regole di tale sintassi è che il registro di destinazione in un'istruzione deve essere SEMPRE il secondo... se infatti noi volessimo copiare ebx in eax non dovremmo scrivere `mov eax, ebx`, bensì `mov ebx, eax`... e questo fatto potrebbe essere fonte di casini per i primi tempi. Altre caratteristiche di tale sintassi sono:

- Il nome dei registri è preceduto da "%" (es. `mov %ebx, %eax`) per poter includere simboli c nel codice senza far confusione.
- La lunghezza dell'operando va posposta all'istruzione che opera su di esso. Ad esempio, per copiare bx (word) in ax (word) si scriverà `movw %bx, %ax`, per il byte si usa la lettera b, per la dword la lettera l.
- Ogni operando immediato va posposto al simbolo "\$" (quello di Bill Gate\$), ad esempio `addl $5,%eax`.
- Non mettere un prefisso a un operando indica che è un indirizzo di memoria. Es. `movl $ciao,%eax` muove l'offset di ciao in eax, mentre `movl ciao,%eax` muove il contenuto di ciao (la dw da esso puntato) in eax.

Come vedete la cosa è abbastanza incasinata, e nonostante esista un prog in grado di convertire la sintassi Intel in AT&T (<ftp://x2ftp.oulu.fi/pub/msdos/programming/convert/ta2asv08.zip>) effettivamente per chi è già abituato a programmare in TASM o MASM sotto Win la sintassi AT&T risulta più complessa.

Infatti, il compilatore dove ci si sente più a proprio agio nel linux asm coding è il NASM, Netwide Assembler, che probabilmente è pure il + usato.

Lo si trova a <http://www.cryogen.com/Nasm/>, la versione più recente è la 0.98. Esso supporta vari formati di file, cmq a noi interessa la versione per linux, ovvero quella che genera gli ELF.

Esistono anche altri compilatori (es. AS86), cmq sono meno utilizzati dei quelli suddetti;).

Per quanto riguarda la programmazione tramite macro... sia il GAS che il NASM supportano il macroprocessing, per il primo esiste il GASP, che aggiunge proprio delle macro al linguaggio, e pure il NASM le supporta (fare riferimento ai docz scaricabili dal sito relativo).

Esistono poi altri tool o compilatori come AS86, CPP, M4, Tunes project, ecc... che però non ha senso analizzarne uno per uno, visto che potranno semmai essere utili in seguito.

Per il linker, invece, non ci sono problemi, in quanto potete usare sia ld che gcc, entrambi compresi in qualsiasi distribuzione di Linux.

Come introduzione penso che possa andar bene, se volete maggiori info leggete il Linux Assembly HOWTO.

I vari link che nella versione originale del tute sarebbero dovuti essere presenti qui sotto potrete trovarli nella sezione apposita del sito;).

Conclusioni

Spero che molti, leggendo questo breve essay, si siano interessati alla programmazione a basso livello sotto Linux, che penso sia molto interessante, forse proprio per il fatto che è un terreno ancora poco esplorato.

Per domande/richieste, proposte varie e così via scrivete al solito indirizzo ritz@freemail.it.

Byz all, Ritz

Attenzione

Le informazioni contenute all'interno di questo documento sono a puro scopo didattico. L'autore non incoraggia chi volesse utilizzarle per scopi illegali.