L' ENTRY POINT DI UN ESEGUIBILE E IL PROBLEMA DEL LINKING

1. INIZIO ESECUZIONE DI UN ESEGUIBILE

Quando io lancio l'eseguibile per farlo eseguire, il loader crea il processo (riempiendo tutte le necessarie tabelle di sistema) e carica in memoria l'immagine del processo,
poi passa il controllo al processo caricato e ne fa eseguire la prima istruzione macchina.
Nell'eseguibile, il formato **ELF** prevede una sezione header in cui e' contenuto un campo "**Entry point address**" il cui valore e' **l'indirizzo in cui si trovera' l'istruzione iniziale da eseguire dopo che il processo sara' stato caricato in memoria dal loader**.
Si può lanciare il comando *readelf -h nomefileseguibile* per vedere questo campo.

Nel codice assembly, **la prima istruzione macchina da eseguire è etichettata con la label _start**, che ne rappresenta l'indirizzo.
Quando il linker genera l'eseguibile, cerca l'istruzione etichettata con **_start** e ne pone l'indirizzo nel campo "**Entry point address**" della sezione header.
Se il linker non trova questa label, produce un errore e non genera l'eseguibile.

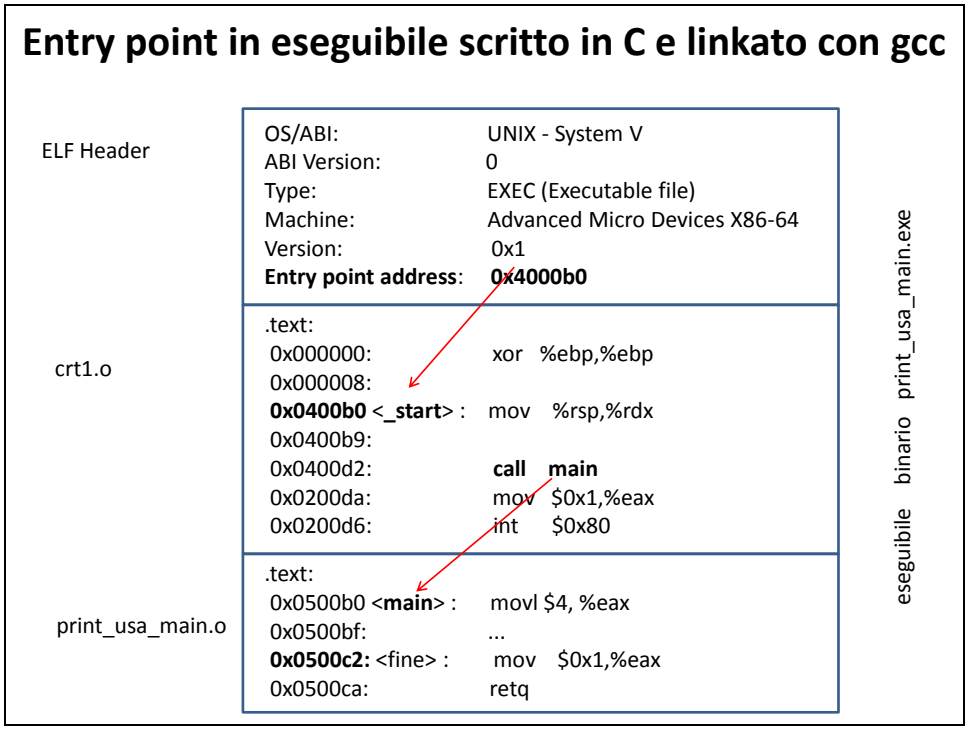
2. LINKING PER CREAZIONE di ESEGUIBILE scritto in LINGUAGGIO C CON gcc

- Quando io scrivo un programma codice in C, il punto di inizio del mio codice e' la funzione main.
- Ma **la prima istruzione che viene eseguita** su ordine del loader, dopo il caricamento in memoria del programma, **non e' quella del main**.

Infatti, prima di poter cominciare ad eseguire il codice del main, occorre svolgere alcune operazioni preliminari, atte a garantire il successivo corretto funzionamento della libc (la libreria standard del C), quali ad esempio impostare il vettore degli interrupt per il processo, inizializzare parte dello stack e altre ancora.
Queste operazioni non sono implementate all'interno del main ma vengono svolte da codice aggiunto autonomamente dal gcc quando effettua l'operazione di linking di moduli scritti in linguaggio C.
Questo codice aggiunto autonomamente dal gcc durante il linking fa parte delle librerie standard del C ed e' "stranamente" contenuto in dei moduli oggetto, ad esempio

/usr/lib/x86_64-linux-gnu/crt1.o
/usr/lib/gcc/x86_64-linux-gnu/5/crtbegin.o
/usr/lib/gcc/x86_64-linux-gnu/5/crtend.o

- E' dentro questo codice macchina, aggiunto dal gcc al momento del linking, che si trova l'istruzione macchina iniziale etichettata con _start.



Solo dopo avere eseguito il codice aggiunto che inizia in _start il codice stesso passera' il controllo al main effettuando un salto al codice di inizio del main stesso.

Core OS C Runtime Objects (**CRT**)
The `crt1.o`, `crti.o`, and `crtn.o` objects comprise the core CRT (C RunTime) objects required to enable basic C programs to start and run. CRT objects are typically added by compiler drivers when building executables and shared objects.

`crt1.o` provides the `_start` symbol that the runtime linker, `ld.so.1`, jumps to in order to pass control to the executable, and is responsible for providing ABI mandated symbols and other process initialization, for calling `main()`, and ultimately, `exit()`. `crti.o` and `crtn.o` provide prologue and epilogue `.init` and `.fini` sections to encapsulate ELF init and fini code.

`crt1.o` is only used when building executables. `crti.o` and `crtn.o` are used by executables and shared objects.

These CRT objects are compatible with position independent (PIC), and position dependent (non-PIC) code, including both normal and position independent executables (PIE).

Nota bene: il gcc, quando deve eseguire il linking, in realta' lancia ld passandogli alcuni argomenti che indicano quale codice aggiungere e altre informazioni.
Ad esempio, in un sistema Linux su processore x64 o amd, se voglio linkare il mio modulo print_usa_main.o il gcc lancia il linker ld e gli passa i seguenti argomenti:

/usr/bin/ld -plugin /usr/lib/gcc/x86_64-linux-gnu/5/liblto_plugin.so -plugin-opt=/usr/lib/gcc/x86_64-linux-gnu/5/lto-wrapper -plugin-opt=-fresolution=/tmp/cc7yRp5X.res -plugin-opt=-pass-through=-lgcc -plugin-opt=-pass-through=-lgcc_s -plugin-opt=-pass-through=-lc -plugin-opt=-pass-through=-lgcc -plugin-opt=-pass-through=-lgcc_s --sysroot=/ --build-id --eh-frame-hdr -m elf_x86_64 --hash-style=gnu --as-needed **-dynamic-linker /lib64/ld-linux-x86-64.so.2** -z relro -o print_usa_main.exe **/usr/lib/gcc/x86_64-linux-gnu/5/../../../x86_64-linux-gnu/crt1.o /usr/lib/gcc/x86_64-linux-gnu/5/../../../x86_64-linux-gnu/crti.o /usr/lib/gcc/x86_64-linux-gnu/5/crtbegin.o** -L/usr/lib/gcc/x86_64-linux-gnu/5 -L/usr/lib/gcc/x86_64-linux-gnu/5/../../../x86_64-linux-gnu -L/usr/lib/gcc/x86_64-linux-gnu/5/../../../../lib -L/lib/x86_64-linux-gnu -L/lib/../lib -L/usr/lib/x86_64-linux-gnu -L/usr/lib/../lib -L/usr/lib/gcc/x86_64-linux-gnu/5/../../.. **print_usa_main.o** -lgcc --as-needed -lgcc_s --no-as-needed -lc -lgcc --as-needed -lgcc_s --no-as-needed **/usr/lib/gcc/x86_64-linux-gnu/5/crtend.o /usr/lib/gcc/x86_64-linux-gnu/5/../../../x86_64-linux-gnu/crtn.o**

Se guardate li' in mezzo, vedete elencati anche i moduli standard aggiunti dal gcc che vi ho citato prima.

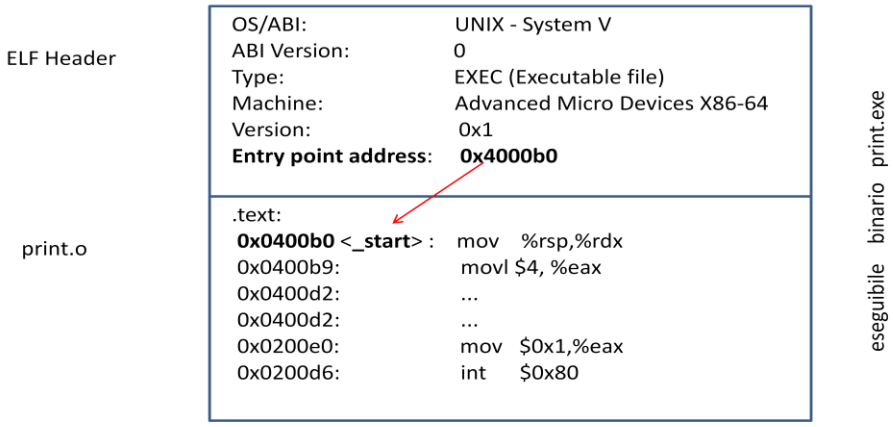
3. LINKING PER CREAZIONE di ESEGUIBILE scritto in ASSEMBLY CON ld

Se invece io scrivo in assembly il codice del mio modulo principale, sono io che devo indicare quale e' la istruzione iniziale etichettata con _start che deve essere eseguita per prima. Sono io che scrivo tutto cio' che deve essere fatto dall'inizio, **il linker non aggiunge niente.**
Nel mio codice assembly quindi **esiste la istruzione iniziale** ed e' indicata dalla etichetta/indirizzo **_start**.

Se scrivo codice assembly, allora per linkare il mio codice macchina devo usare il linker **ld** (usato anche dal gcc stesso) che costruisce l'eseguibile usando il mio codice macchina ma non aggiunge altro codice in autonomia.
Ad esempio, per linkare il mio modulo assembly print.o lancero' il linker ld con questi parametri:

ld -o print.exe print.o

Entry point in eseguibile scritto in assembly e linkato con ld



4. PROBLEMA nel LINKING con gcc PER CREAZIONE di ESEGUIBILE scritto in ASSEMBLY

Se invece io scrivo in assembly il codice del mio modulo principale, indicando l'istruzione iniziale etichettata con _start e poi linko il mio modulo oggetto chiamando il gcc senza parametri speciali, **il gcc** normalmente **aggiunge dei moduli** aggiuntivi **che contengono** essi stessi **una istruzione iniziale** etichettata con **_start**.

- In tal modo, **durante il linking saranno presenti due istruzioni etichettate con _start ed il linker non sa quale usare come istruzione iniziale da far eseguire**.
  Percio' il linker provoca un errore e non genera l'eseguibile.
  L'errore viene indicato cosi': "multiple definition of `_start"

Inoltre, **nel codice aggiunto dal gcc c'e' un salto alla istruzione del main, ma nel mio codice assembly non ho messo una label main, perche' sono partito da _start.**

- Quindi **il linker non riesce a trovare una etichetta main e percio' non riesce a risolvere il simbolo main che manca**.
  Il linker quindi provoca un altro errore e non genera l'eseguibile.
  L'errore viene indicato cosi': "In function `_start': undefined reference to `main'"

4. SOLUZIONI del PROBLEMA nel LINKING con gcc PER CREAZIONE di ESEGUIBILE scritto in ASSEMBLY

Esistono due modi diversi per risolvere il problema che impedirebbe di linkare con gcc dei moduli scritti in assembly.

4.1 MODO 1 - LINKING con gcc, SENZA LIBRERIE STANDARD, di ESEGUIBILE scritto in ASSEMBLY

Ordino al gcc di fare il linking SENZA USARE LE SUE LIBRERIE STANDARD, che tanto non mi servono perche' io uso solo istruzioni assembly.
Per fare cio' AGGIUNGO NEL COMANDO DI LINKING ordinato al gcc il flag **-nostdlib** che ordina di non usare le librerie standard del C.

```
gcc -nostdlib -o print.exe print.o
```

In tal modo il gcc non aggiunge i moduli aggiuntivi iniziali e quindi non aggiunge l'istruzione etichettata con _start e nemmeno aggiunge il salto verso l'etichetta main.

Risolto.
Nasce il problema che non posso usare le librerie standard del C all'interno del mio codice assembly.

4.2 MODO 2 - LINKING con gcc, CON LIBRERIE STANDARD, di ESEGUIBILE scritto in ASSEMBLY

Modifico il mio codice assembly NON FACENDOGLI FARE la parte iniziale di esecuzione.
Cioè faccio partire il mio codice assembly non da _start bensi' da main, come se fosse un programma scritto in C.

In pratica, nel mio codice assembly principale, al posto delle due righe di codice seguenti:

```
.globl _start
_start:
```

devo sostituire le due righe seguenti:

```
.globl main
main:
```

Poi assemblero' il sorgente assembly con l'assemblatore as o col gcc

```
as -o print.o --gstabs print.s  # assemblaggio diretto con as
  oppure con
gcc -c -g -o print.o print.s    # assemblaggio tramite gcc che chiama as
```

A questo punto posso usare il gcc per linkare normalmente senza dover chiedere di escludere le librerie di default.

```
gcc -o print.exe print.o
```

Infatti, in questo modo, il gcc linker aggiunge il suo codice macchina (che parte da _start e che poi salta al main) e trova nel modulo oggetto originato dall'assembly il simbolo main che indica da dove far partire il codice utente.

In questo modo, nel mio eseguibile mi ritrovo anche le librerie standard e quindi, volendo, potrei richiamare da assembly anche le funzioni C presenti nelle librerie standard, purche' sia io da assembly a preparare correttamente lo stack per la chiamata a funzione C, come farebbe il compilatore, e analogamente a recuperare il risultato e ripulire lo stack dopo la fine della chiamata a funzione.