Under Atthack

15

Cover by Likas

UNDERATTHACK

IN QUESTO NUMERO

N.15

Prefazione

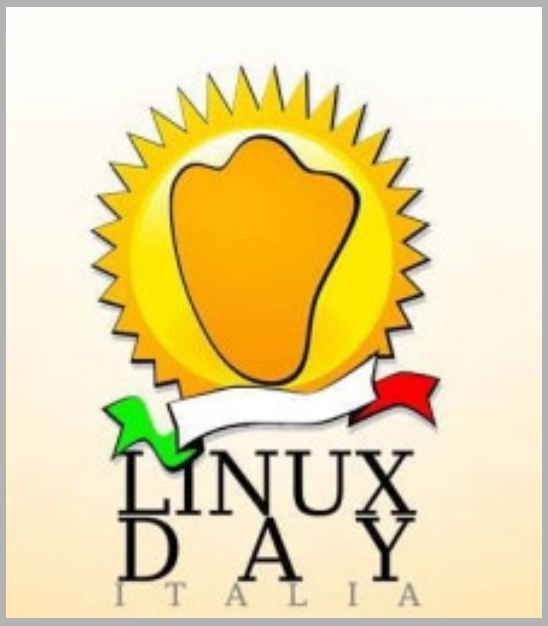
Buongiorno Onorevole, come va? Cosa fa questo weekend? Viene a cena da me sabato sera? Ci sono anche il direttore, il presidente, l'ingegnere e il commendatore; insomma se è libero ci farebbe molto piacere averLa con noi. Sa com'è, di questi tempi, con la situazione difficile in cui ci troviamo se non ci si stringe stretti non si sa mica dove si va a finire.
Ho deciso di scriverLe perché telefonarLe sta diventando un rischio, in questo pazzo paese forze occulte stanno tramando contro l'interesse di noi popolo sovrano. Ormai, come Lei saprà meglio di me, siamo oppressi da questi ignoranti che ci ronzano attorno come mosche chiedendo comportamenti 'eticamente e moralmente responsabili'; si preoccupano di più di chi entra a casa nostra piuttosto di chi esce per andare alle Cayman con i nostri soldi, che tra l'altro sono i loro!
Lo sa ultimamente su cosa ho riflettuto? Sulla differenza che credo esiste tra il poveraccio e il pezzente.
Mentre il poveraccio non ha nulla e non ha nulla da perdere, il pezzente è colui che vive come una formichina che si porta i quattro semini sotto terra convinta di avere uno status benestante che deve mantenere a qualunque costo.
È un po' la stessa differenza che si ha in guerra quando si punta il fucile contro un terrorista kamikaze o contro il civile inerme: il primo non ha nulla da perdere e ha già scelto di morire, quindi è assolutamente necessario premere il grilletto; il secondo invece pensa alla sua misera vita, i suoi miseri figli e il suo altrettanto misero futuro, così è disposto a fare qualunque cosa purché non si spari.
Quindi noi, che ne converrà siamo la parte responsabile del popolo, dobbiamo agire al meglio per guidare la democrazia, in una grave crisi il numero dei pezzenti aumenta perché di necessità siamo costretti a far pagare il plebeo per mantenere il nostro potere democratico, d'altra parte questi si trovano con il fucile puntato e possono essere convinti delle cose più assurde: se prenderanno lo stipendio più basso sarà un bene perchè terranno il loro lavoro, se pagheranno più tasse sarà un bene per i loro figli, se pagheranno più tasse i ricchi sarà un male sia per loro che per i loro figli.
D'altra parte la democrazia è un sistema che va curato giorno dopo giorno, bisogna costantemente stare attenti perchè la nostra libertà non venga sopraffatta dalla dittatura plebea. La plebe va spronata di continuo con le varie paure secondo del momento: il comunismo, il terrorismo, il default e chissà cos'altro escogiteremo in futuro!
La cosa importante del fucile democratico è che chi è sotto tiro non abbia mai il tempo di pensare a che ha il dito sul grilletto.
In ogni caso spero di approfondire l'argomento con Lei questo sabato. Con affetto

Floatman

P.S. Lasci la moglie a casa che c'è già il pieno. Ci siamo capiti...

Under_NEWS_AttHack

LINUXDAY



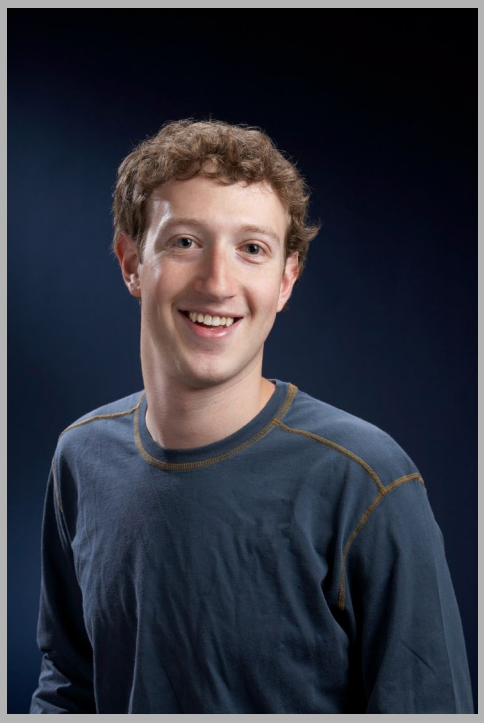
Un anno fa UnderAttHack lanciava un concorso ai propri lettori, e quest'anno lo ripetiamo:
Partecipate al LinuxDay?
Mandateci un articolo corredato di foto dove descrivete come avete passato il linux day nella vostra città, gli articoli più belli e meritevoli verranno pubblicati nel prossimo numero della ezine!

Anche quest'anno il linuxDay si svolge nel penultimo fine settimana di ottobre, il 22, dove sarete voi? Cosa vedrete? Personalmente ho deciso, data la mia grossa delusione per il linuxday di Palermo dell'anno scorso, di traferirmi un po' più al nord e con una piccola gitarella in solitaria sfiorerò la costa romagnola e andrò al LinuxDay di Modena... dove chi mi riconoscerà per primo avrà una bella sorpresa . A Palermo quest'anno avremo il nostro grafico Likas che si inerpicherà tra la gente scattando foto e facendo finta di fare una relazione!
Forza Likas e forza ragazzi aspettiamo una vostra collaborazione come fu per l'anno scorso!
LEGGERE DOPO IL 21 OTTOBRE
Buon linuxDay 2011
**

vikkio88

Qualcuno cambi il pannolino di

Circa quasi 6 anni fa nasceva facebook e la sua fama cresceva esponenzialmente.



Il giovinotto che lo creava da un idea non troppo personale (vedasi la storia dei Winkelvoss ben riassunta nel film da oscar "The Social network"), ma nemmeno tanto originale (vedasi: Badoo,MySpace,Twitter e altri miliardi di socialnetwork) , piano piano rubava al resto del web sempre più spazio fino a diventare da qualche anno:

Il Miliardario più giovane del mondo.

Mark Zuckerberg che tanto ha fatto spaventare i grandi del web (soprattutto google) ha incassato un successo dopo l'altro fino quasi a sentirsi invincibile e leader nel campo dei socialnetwork. Come lui, qualche anno prima il creatore di MySpace: Tom Anderson.

Purtroppo per lui da qualche anno tra molti iscritti hanno lasciato solo la loro triste pagina vuota...finchè, devatato si è dovuto piegare permettendo l'accesso a myspace tramite facebook, che suona un po' come: "Per accedere a SanSiro bisogna avere la tessera Juventus FC premier member".

Dopo tanti successi che si sono ripetuti negli anni si è raggiunta la cifra record di 400 milioni di iscritti (si vocifera) in tutto il mondo (tranne che in China), ma dopo anni di dominio incontrastato sembra che qualcosa si stia muovendo.

Qualcuno cambi il pannolino di Zuckerberg

Facebook ha svegliato il cane che dormiva (dormiva mica tanto poi).
Stiamo parlando di Google che di dormire non è ha voluto proprio sentir parlare, ma che tra i tanti tentativi che ha fatto di distruggere la supremazia sociale di facebook, tra i quali degni di nota sono: Buzz (simil twitter) e Wave (simil...boh) ora sembra aver azzeccato la giusta mossa: Copiare Diaspora e mettere un + davanti al proprio nome!

Nasceva qualche mese fa GugolPlas, prima per piccola nicchia di privilegiati, poi dopo una serie di inviti a catena si è allarganto fino a raggiungere nel giro di pochissimo tempo dimensioni che facebook aveva raggiunto in un anno. Peccato che la maggior parte della gente che si iscrive a google+ lo fa solo per curiosità e poi ritorna su facebook, che intanto incassando il colpo abbastanza bene si è sbrigato subito ad aggiornare la propria gestione della privacy copiando la copia di Diaspora!
Un copiacopia orrendo, e chi come me ha provato Diaspora sa di cosa parlo... D'altro canto gli sviluppatori del progetto OpenSource da qualche tempo hanno dichiarato che sono contenti di aver dato, in un modo o nell'altro una mano a cambiare una filosofia propria ai socialnetwork che vedeva la privacy dei contenuti gestibili solo tramite antiche arti di divinazione.

Il team di Diaspora si è inventato un sistema di gestione diverso, facendo diventare di proprietà degli utenti i contenuti, e la possibilit di gestire chi può vedere cosa.
Quindi da qualche tempo su Facebook tutti potranno fare vedere le fotine del culo dell'amante solo alla cerchia degli amici del calcetto!
E tutto con un effetto molto fiquo di cerchi che giravolteggiano! Che bello essere su un socialnetwork eh? Intanto chi vi parla tiene facebook dal 2008, e per lavoro o per università è diventato troppo insostituibile... viene quasi da piangere, sperando che il prossimo Zuckerberg di passaggio cambi tutto ancora una volta!

vikkio88

Post - Office

Ciao ragazzi, dopo un pausa più o meno meritata tornano ancora una volta le mail su questo numero. Ne abbiamo ricevute a valanga qui pubblichiamo le più interessanti/pubblicabili, la maggior parte di quelle che riceviamo riguardano soprattutto la possibilità di collaborare con la nostra ezine con qualche articolo od operativamente entrare nella nostra redazione! Questo ci rende molto felici e rispondo a tutti nello stesso modo:

Se volete partecipare con un articolo basta inviarlo via mail a: **underatthack@gmail.com**
Per partecipare attivamente alla redazione aiutandoci con il sito, con l'impaginazione ed entrare nella redazione basta iscriversi al forum: linux.forumattivo.eu partecipare attivamente alle sezioni aperte e pianpiano guadagnarsi un posto nella redazione!

Da:
sd dsd <iftrue@hotmail.it>

ciao ragazzi..
innanzitutto complimente i per la rivista,la leggo sempre ed e' molto interessante,riesce a prenderti molto durante la lettura,sia per gli argomenti sia per il modo di "fare".
Ho aperto un sito, volevo linkarvelo,dato che e' un sito per i programmatori, esperti e non.
www.informaticsfree.altervista.org
solo che finche' siamo i 5 a frequentarlo e' una risorsa inutile XD
io ve l'ho linkato,spero che vi interessi

Ciao e grazie dei complimenti, soprattutto per il termine: "modo di fare" che mi ha entusiasmato non poco!
Linkiamo sempre i siti dei nostri lettori appassionati nella sezione mail, perchè proprio grazie al passaparola prodotto dal banner che mettete in giro nei blog cresciamo sempre più! Grazie di leggerci e scusa per l'attesa

vikkio88

Da:
AnnoyancE <la.larva@live.com>

Salve a tutti voi.
Io sono uno di quelli che si è accorto che l'internet in Italia stà morendo sopratutto perchè non si parla più di hacking ma di stronzate del genere. Io vorrei creare una Community dove tutti noi ci possiamo ritrovare, perchè diciamoci la verità la maggior parte di noi si è persa e non sà più come rientrare in questo mondo. Io sarei uno di questi e vorrei rientrare ma purtroppo non esiste (a mio parere) un forum o web site dove potersi riunire, quindi io vorrei chiedervi..Mi potete aiutare a ricreare il mondo dove le mie conoscenze pur se scarse sono nate, vorrei che non morissero ma che crescessero con me, con la mia sete di conoscenza, vi prego aiutatemi.

AnnoyancE

Carissimo Annoyance,
A parte il fatto che credo sia poco rispettoso considerarci "noi" senza che ci conosciamo, avrei circa una cinquantina di appunti da farti sul tuo discorso, che oltre a non condividere credo sia formulato per frasi fatte e per luoghi comuni presi qua e la...
Cosa intendi per non hacking ma stronzate del genere?
Vuoi che fondiamo una community per cosa? Sarebbe opportuno che tu ridefinissi meglio le tue richieste, intanto se già ci hai individuato come persone con cui puoi parlare di questo perchè non ti registri al nostro forum e ne parliamo là?
Grazie della mail e scusa per l'attesa

vikkio88

Basic jQuery Tutorial

Nell'ultimo periodo ho preso la decisione di crearmi un CMS , e volevo ,e voglio tuttora, che in questo CMS non si debba ricaricare la pagina per inserire commenti o leggere gli articoli, per questo ho cominciato a studiacchiarmi jQuery e Ajax (in realtà lo facevo già prima asd).

Ora che è un po' che mi ci diverto, ho deciso di scrivere un tutorial. Non sarà un tutorial per esperti, dato che io non lo sono, lo scopo è quello di diffondere ciò che ho imparato col solo intento di facilitarne l'apprendimento ad altri.

Ho scelto di usare jQuery per il mio scopo perché è facile, e perché ci posso anche fare degli effetti fighi asd. Ma qual è il vantaggio ad usare jQuery, bè, il motto è *write less, do more* quindi mi sembra abbastanza evidente, di fatti con jQuery in due tre righe di codice si possono ottenere risultati sorprendenti. Un'altra caratteristica importante è la sua compatibilità con tutti i browser, ogni script che scriveremo, quasi sicuramente sarà cross-browser, senza dover battere la testa nemmeno su quella merda di IE :)

Prima di iniziare a parlare delle chiamate ajax e degli effetti però, dovete sorbirvi una parte di tutorial sull'uso basilare di jQuery.

jQuery Basics

"jQuery è una libreria di funzioni (un cosiddetto software framework) per le pagine web, codificata in javascript, che si propone come obiettivo quello di astrarre ad un livello più alto la programmazione lato client del comportamento di ogni singola pagina HTML."

Come non affidarsi a wikipedia per le definizioni? xD Comunque in parole povere jQuery serve per interagire con la pagina html. Per modificare attributi di alcuni elementi, per gestire gli eventi, per manipolare il DOM e tanto altro.

Download jQuery

Ogni volta che vorrete utilizzare jQuery in una vostra pagina html dovete includere il framework, in che modo? Esistono due possibilità:

-Vi scaricate l'ultima versione disponibile da http://docs.jquery.com/Downloading_jQuery e nel documento html la includete con:

```
<script type="text/Javascript" src="file_jquery.js"></script>
```

Dove **file_jquery** spero sappiate vada sostituito con il nome del file dove tenete jQuery.

In alternativa

-Linkate direttamete ad un file su un altro server:

```
<script type="text/Javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.1/jquery.min.js"></script>
```

Let's start

Come ho detto partiamo con le basi, iniziamo con un esempio. Abbiamo detto che jQuery serve per interagire con delle pagine html, il nostro primo script ci permetterà, cliccando su un link, di far diventare verde il suo background. Per prima cosa creiamo una pagina html semplicissima:

```
<html>
     <head>
             <script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jque-
ry/1.6.1/jquery.min.js" ></script>
             <script type="text/javascript" src="JQ.js"></script>
     </head>
     <body>
             <a href="#">Aggiungi background verde</a>
     </body>
</html>
```

Come vedete ho incluso jQuery con un collegamento esterno. Ora siamo pronti ad iniziare. Il nostro script va nella head, io per comodità nelle pagine html di esempio che scriverò lo includerò sempre da un file esterno chiamato **JQ.js.** Ecco il file che ci serve per il nostro esempio:

```
$(document).ready(function(){
     $("a").click(function(){
             $(this).css("background-color", "green");
     });
});
```

Le istruzioni che fanno si che lo sfondo del link diventi verde stanno nel mezzo a:

```
$(document).ready(function(){
     ...istruzioni...
});
```

Questo perché necessitiamo del DOM caricato per poter cambiare il background del link quando viene cliccato, e *$(document).ready()*; permette di vedere quando il **DOM** è stato **caricato**. Solo successivamente vengono eseguite le istruzioni racchiuse in *function(){}* passata come parametro alla funzione *.ready()* .
Analizziamo ora il succo di questo codice:

```
$("a").click(function(){
     $(this).css("background-color", "green");
});
```

L'istruzione *$("a")* è quello che viene definito un selettore, un selettore ci permette appunto di selezionare su quali elementi della pagina devono essere eseguite alcune operazioni. Nel nostro caso abbiamo selezionato *tutti gli* **a** del documento, quindi tutti i link (nella pagina html ne abbiamo solo uno ma se provate ad aggiungerne altri vedrete che cliccandoci sopra lo sfondo diventerà verde) ed a tutti essi abbiamo associato un qualcosa da fare nel caso in cui vengano cliccati. Infatti la funzione *.click()* è praticamente la stessa cosa di *onclick*, essa compie la funzione passata come argomento quando l'elemento associato al selettore viene cliccato. Andando ad esaminare la funzione passata come argomento a *click()* notiamo:

```
$(this).css("background-color", "green");
```

Il concetto è sempre il solito, si seleziona un elemento *$(this)*, e gli si applica una funzione, *.css()*. *$(this)* è uno speciale selettore che si riferisce al selettore a cui è applicata la funzione precedente (nel nostro caso, quindi, **this** si riferisce ad **a**).

La funzione *.css()* serve a cambiare un attributo di stile a tutti gli elementi selezionati, prende due parametri, uno è l'attributo da cambiare *(background-color)*, l'altro il valore da associare *(green)*.

Selettori

Abbiamo parlato di selettori, ma cerchiamo di approfondire la questione. Essi, come abbiamo detto, selezionano elementi del documento su cui vengono applicate determinate funzioni. Si possono selezionare degli elementi in vario modo:

Tag name

La selezione per il nome del tag è quella che abbiamo attuato nell'esempio precedente, ovvero:

```
$("nome_tag")
```

Dove **nome_tag** è il nome del tag da selezionare. E' bene ricordare che nel caso ci siano più elementi con lo stesso tag nel documento, questo li seleziona tutti, ad esempio, se provate ad aggiungere qualche altro link al file html precedente, e nel file **JQ.js** scrivete:

```
$(document).ready(function(){
    $("a").css("color", "lime");
});
```

Vedrete che tutti i link, appena il DOM è caricato, assumeranno il lime come colore del testo. Come fare a selezionare solo uno o determinati elementi?

ID or Class

In questo caso vi viene in aiuto la possibilità di selezionare degli elementi per il loro ID o per la loro classe.
Ad esempio, scriviamo un'altra pagina html.

```
<html>
    <head>
        <script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jque-
ry/1.6.1/jquery.min.js"></script>
        <script type="text/javascript" src="JQ.js"></script>
    </head>
    <body>
        <a href="#">testo</a><br />
        <a href="#">Altro testo</a><br />
        <div id="agg"></div>
    </body>
</html>
```

Ed il nostro file JQ.js

```
$(document).ready(function(){
    $("a").click(function(){
        $("#agg").append($(this).text()+"<br />");
    });
});
```

In questo caso ogni volta che clicchiamo su un link del documento viene aggiunto il suo testo al div con id="agg". Come?

Abbiamo sempre i nostri link selezionati con $("a") e ad essi abbiamo sempre applicato il nostro evento click() e ad esso abbiamo passato la funzione:

```
function(){
     $("#agg").append($(this).text()+"<br />");
}
```

La quale seleziona il div con id="agg" tramite:

```
$("#agg")
```

Di fatti per selezionare un elemento con uno specifico id si fa:

```
$("#id_elemento")
```

Mentre per selezionare tutti gli elementi che appartengono ad una specifica classe:

```
$(".nome_classe")
```

Successivamente applichiamo al nostro div la funzione .append(), che aggiunge al testo interno all'elemento selezionato la stringa passata come argomento, che nel nostro caso è rappresentata da:

```
$(this).text()
```

che seleziona il testo dell'elemento $("a") cliccato.

```
+"<br />"
```

che manda a capo xD

Selezione CSS

Questo tipo di selezione segue le normali regole di selezione CSS, ad esempio, se volessimo che solo cliccando sui link che si trovano dentro al div con *id="agg"* si aggiunga testo ad esso, dovremo fare su una pagina html di questo tipo:

```
<html>
     <head>
            <script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jque-
ry/1.6.1/jquery.min.js"></script>
            <script type="text/javascript" src="JQ.js"></script>
     </head>
     <body>
            <a href="#">testo</a><br />
            <div id="agg">
                   <a href="#">Interno</a><br />
                   <a href="#">Interno 2</a><br />
            </div>
     </body>
</html>
```

Uno script a questo modo:

```
$(document).ready(function(){
     $("#agg a").click(function(){
            $("#agg").append($(this).text()+"<br />");
     });
});
```

Come vedete in **JQ.js** è cambiato solo il selettore dei link, adesso abbiamo:

```
$("#agg a")
```

Che seleziona tutti i link interni al div con id="agg", proprio come un normale CSS. La stessa cosa la possiamo fare con un'altra sintassi:

```
$("#agg").find("a")
```

Che praticamente dice *"trova tutti gli a dentro a #agg"*. Vista così la funzione find() non sembra utile, infatti si può semplicemente fare $("#agg a"), ma tenetela a mente perché vi tornerà comoda quando lavorerete sull'xml.

Una cosa da tenere a mente è che **.find()** ritorna un array di tutti gli elementi che 'matchano' con l'argomento passato a *find()*. Quindi eccovi due Tips estremamente utili (almeno dal mio punto di vista):

1. potete fare operazioni su ogni elemento dell'array che ritorna la find() usando la funzione *.each(function(){})*, che per ogni elemento dell'array esegue le istruzioni nella funzione passata come argomento:

```
$("#agg").find("a").each(function(){
//cosa fare su questo elemento: $(this).istruzioni
})
```

2. potete controllare se .find() ha trovato o meno qualche elemento che corrisponde a ciò che cercate, dato che .find() ritorna un array, infatti, basta un controllo come:

```
if($("#agg").find("a").length) {cosa fare se find trova qualcosa}
else{ cosa fare se non lo trova}
```

Selezione per attributo

Si possono selezionare determinati elementi filtrando gli attributi relativi a quel tag. Ad esempio, se nel nostro documento abbiamo vari link, e vogliamo che solo quelli che presentano l'attributo title siano di colore verde, possiamo, con jQuery, filtrare un tag per un suo attributo, cioè vedere se esso è presente nel tag o meno, nel nostro caso faremmo:

```
$("a[title]").css("color", "green");
```

Infatti per vedere se un attributo è presente o meno in un tag basta fare:

```
$("tag_name[attributo]")
```

Ma c'è di più, perché possiamo anche filtrare degli elementi per il valore di un attributo, per esempio, se voglio selezionare tutti i link con **title="prova"** basterà fare:

```
$("a[title='prova']"
```

La sintassi infatti è:

```
$("tag_name[attributo='valore']")
```

Ma non è finita qui perché, se per esempio volessimo selezionare tutti i link con attributo **title** che presenta al suo interno anche la stringa *"prova"* (quindi un title può essere "prova_uno", un altro "riprova"), si può dare un semplice:

```
$("a[title*='prova']")
```

Quindi basta mettere un asterisco * prima dell'uguale =

Selezionare i genitori

Esiste anche un modo di selezionare tutti gli elementi riferiti ad un tag che sono genitori di altri. Ad esempio, se noi volessimo selezionare tutti i div "genitori" (ovvero il div che lo racchiude) di un link basterà fare:

```
$("a").parents("div")
```

infatti la funzione *parents()* riferita ad un elemento accetta come parametro un selettore che seleziona l'elemento se è genitore del primo selezionato.

Eventi

Nel corso di questa guida avete notato che ho parlato di eventi applicabili a degli elementi, .click() è un evento che corrisponde a onclick; ciascuno degli eventi che è possibile scrivere come attributi di alcuni tag, come onmouseover, onclick ecc... corrisponde un evento jQuery.

Riprendiamo il primo esempio del tutorial, il cambio di colore di background di un link quando viene cliccato. Se volessimo cambiare colore ad esso solo quando il mouse è sopra, semplice, basta usare l'evento .hover():

```
$(document).ready(function(){
    $("a").hover(function(){
            $(this).css("background-color", "green");
    }, function(){
            $(this).css("background-color", "white");
    });
});
```

Come vedete .hover() accetta due parametri, una funziona che definisce cosa deve accadere quando il mouse è sopra l'elemento selezionato, ed un'altra funzione che definisce cosa deve accadere quando il mouse non è più sopra.

La logica che sta dietro gli eventi è sempre identica, quindi mi sembra inutile fare altri esempi, potete trovare una lista completa degli eventi disponibili a questo indirizzo: http://api.jquery.com/category/events/

Funzioni

Durante il tutorial abbiamo spesso usato delle funzioni su degli oggetti, css() è una funzione .append() è una funzione. Come per gli eventi, esiste un elenco completo delle funzioni sul sito ufficiale di jQuery.

Tuttavia voglio mostrarne alcune che ritengo utili.

attr()

La funzione att() può essere usata in vario modo, ma il suo scopo è sempre quello di aggiungere uno o più attribui all'elemento selezionato, esistono varie sintassi per fare ciò, quella che preferisco è:

```
$("selettore").attr({attributo: 'valore', attributo2: 'valore'});
```

Praticamente viene passato un array come argomento di *attr()* dove **attributo** è il nome dell'attributo e valore è il suo **valore.**

attr() può risultare molto utile anche per leggere degli attributi, la sintassi diventa:

```
$("selettore").attr('nome-attributo');
```

Che restituisce il valore dell'attributo passato come argomento.

removeAttr()

L'opposto di *attr()* infatti rimuove un attributo, la sua sintassi è:

```
$("selettore").removeAttr("attributo");
```

dove **attributo** è il nome dell'attributo da rimuovere.

hasClass() addClass() removeClass()

Queste tre funzioni hanno tutte la stessa sintassi per cui le metto insieme, servono, rispettivamente, a verificare se un elemento ha o meno una classe specifica *(hasClass())*, ad aggiungere una classe *(addClass())* o a rimuoverla *(removeClass())*.

Tutte e tre accettano come argomento il nome di una classe, la sintassi è la seguente:

```
$("selettore").hasClass("nome_classe");
$("selettore").addClass("nome_classe");
$("selettore").removeClass("nome_classe");
```

html() val() text()

Anche queste tre funzioni hanno una sintassi simile. Tutte e tre possono accettare un parametro o meno e, nel caso non venga passato:

html()	Restituisce l'html interno all'elemento selezionato;
val()	Restituisce il valore di "value" presente, ad esempio, in un'area d'inserimento testo.
text()	Restituisce il testo interno all'elemento selezionato.

Altrimenti le tre funzioni possono accettare una stringa come parametro:

html(stringa)	Rimpiazza l'html interno all'elemento selezionato con la stringa (che può essere codice html) passata come argomento della funzione;
val(stringa)	Rimpiazza il valore di "value" dell'elemento selezionato con la stringa passata come argomento della funzione;
text(stringa)	Rimpiazza il testo interno all'elemento selezionato con la stringa passata come argomento della funzione.

prepend()

Abbiamo già parlato della funzione *append()* che "appende" la stringa passata come argomento alla fine dell'elemento selezionato, bene, *prepend()* è l'opposto, perché questa funzione, con la sintassi:

```
prepend(stringa)
```

Serve ad inserire l'espressione **stringa** all'inizio del testo dell'elemento selezionato.

before() after()

Queste due funzioni prendono come parametro una **stringa** e la inseriscono, rispettivamente, prima o dopo l'elemento selezionato:

```
$("selettore").before(stringa);

$("selettore").after(stringa);
```

Potrebbero sembrare le stesse identiche funzioni di *prepend()* e *append()*, in realtà c'è una grossa differenza, *prepend()* e *append()* infatti mettono la stringa all'inizio o alla fine dell'elemento selezionato, mentre *before()* e *after()* inseriscono la stringa prima o dopo l'elemento, il seguente schema dovrebbe chiarire il concetto, le funzioni sono inserite nella posizione relativa ad un elemento in cui inserirebbero la stringa:

before() **<elemento>** prepend() **Testo elemento** append() **</elemento>** after()

replaceAll() repalceWith()

Essendo jQuery un framework per manipolare il DOM, non possono mancare le funzioni di replace. Ne esistono due *replaceAll()* e *replaceWith()*.
-La prima ha questa sintassi:

```
$('con cosa').replaceAll('tag_elementi');
```

che rimpiazza ogni elemento che corrisponde al tag passato come argomento di *replaceAll()* con **con cosa**.

Ad esempio, per rimpiazzare ogni link con una scritta in corsivo:

```
$('<em>scritta</em>').replaceAll('a');
```

-La seconda ha questa sintassi:

```
$('cosa').replaceWith('con cosa');
```

Adesso creiamo il nostro file JQ.js, al click del link con id='a', mostreremo il box, al click di quello con id='c' lo faremo scomparire.

```
$(document).ready(function(){
     $("#a").click(function(){$('#box').show('slow');});
     $("#c").click(function(){$('#box').hide('slow');});
});
```

E' tutto molto semplice, come vedete **.show()** e **.hide()** accettano come argomento una stringa che indica quanto deve durare l'effetto:

slow
normal
fast

Oppure potete specificarne la durata in millisecondi.

Se avete un box e volete che, cliccando su un link, esso appaia e scompaia alternativamente, basta usare la funzione **toggle()**, essa accetta come parametro la velocità dell'effetto proprio come le due funzioni precedenti. L'unica differenza è che viene applicata solo a uno dei due link, senza dover specificare altre funzioni, nel nostro caso il file JQ.js diventerebbe:

```
$(document).ready(function(){
     $("#a").click(function(){$('#box').toggle('slow');});
});
```

Funzioni che si usano alla stessa maniera delle tre appena viste (perciò non le spiego) , ma che creano effetti diversi sono:

Apparire	slideUp()	fadeIn()
Scomparire	slideDown()	fadeOut()
Toggle	slideToggle()	----------

Come detto abbiamo anche funzioni per personalizzare gli effetti grafici, spiego solo la principale che è **animate()**.

La sintassi della funzione animate() è simile a quelle viste in precedenza, infatti si applica ad un selettore nello stesso modo:

```
$(selettore).animate(parametri);
```

La funzione, completa di tutti i parametri, la possiamo dichiarare così:
animate(params, duration, easing, callback) //dichiarazione ripresa dal sito di jQuery xD

Andiamo a spiegare in linea teorica ogni parametro:

params	Una lista di attributi di stile (attributi css) che si vogliono animare su quell'elemento. Gli attributi che si possono animare non sono tutti, ma un piccolo gruppo, in generale si possono sintetizzare in attributi di: posizionamento (left, top...), margine, grandezza font, opacità elemento, grandezza bordi, altezza e larghezza elemento. Ad esempio non potete, di default, animare il background-color. Di default perché basta installare un semplice plugin per poterlo fare (in ogni caso non tratto questo argomento, basta googolare "plugin colore jquery" e trovate ciò che volete.) La lista di attributi deve essere racchiusa tra due parentesi graffe: {attributo: cosa_farci, attributo2: cosa_farci}
duration	La durata dell'effetto, che si esprime nello stesso modo delle funzioni viste in precedenza.
easing	Questo parametro servirebbe a definire che tipo di effetto vogliamo sull'easing, ma necessita di plugin, quindi non lo testiamo.
callback	Una funzione da richiamare alla fine dell'animazione.

Abbiamo parlato della lista di attributi passata come parametro params, e vi ho detto che è una lista di attributi composta da:

{attributo: cosa_farci, attributo2: cosa_farci}

Dove attributo, attributo2 ecc... Sono gli attributi css da animare, cosa_farci è l'animazione da applicarvi. Possiamo applicare vari tipi di animazioni ed in generale penso di poter distinguere due gruppi, uno di **"apparsa(si dice? boh)/ scomparsa"** e uno di **"incremento/decremento"**.

-apparsa/scomparsa.

Questo gruppo serve, naturalmente, a far apparire e scomparire l'elemento o gli elementi che corrispondono al selettore. I nostri attributi css dicono solo in che modo farli sparire o apparire, ad esempio, per far apparire un elemento nello stesso modo della funzione **slideDown()**, useremo *{height:"show"}*, di fatti se si usano gli attributi per far apparire o scomparire qualcosa, potranno assumere tre valori:

show	Mostra l'elemento animando quell'attributo.
hide	Nasconde l'elemento animando quell'attributo.
toggle	Fa il toggle.

Facciamo un esempio, riprendiamo la nostra pagina html:

```
<html>
     <head>
              <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.1/jquery.min.js"></script>
              <script type="text/javascript" src="JQ.js"></script>
              <style type="text/CSS">
              #box{
                       border: 1px solid black;
                       height: 500px;
                       width: 500px;
              }

              </style>
     </head>
     <body>
              <a href="#" id="a">Apri</a>
              <a href="#" id="c">Chiudi</a>
              <div id="box">

              </div>
     </body>
</html>
```

Ora, se vogliamo crearci un effetto figo, uno sliding orizzontale, basterà fare, nel nostro file JQ.js:

```
$(function(){
              $("#a").click(function(){$('#box').animate({width: 'show'}, 'slow')});
              $("#c").click(function(){$('#box').animate({width: 'hide'}, 'slow')});
});
```

Naturalmente, affinché si possano applicare le animazioni show hide o toggle, gli attributi su cui si applicano devono essere settati, altrimenti non vedreste un fico secco.

-incremento/decremento.

Le animazioni di incremento/decremento consistono nell'aumentare o diminuire, o semplicemente cambiare valore, agli attributi css, partiamo con un esempio:

html:

```
<html>
     <head>
              <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.1/jquery.min.js"></script>
              <script type="text/javascript" src='JQ.js'>
              </script>

              <style type="text/CSS">
              #box{
                       border: 1px solid black;
                       position: absolute;
                       left: 0;
                       top: 50;
                       width: 500px;
                       height: 100;
              }
              </style>
     </head>
     <body>
              <a href="#" id="a">Apri</a>
              <a href="#" id="c">Chiudi</a>
              <div id="box" class='black'>
              </div>
     </body>
</html>
```

JQ.js:

```
$(function(){
     $("#a").click(function(){$('#box').animate({top: '+=100', left: '+=100'}, 'slow')});
     $("#c").click(function(){$('#box').animate({top: '-=100', left: '-=100'}, 'slow')});
});
```

Questo codice non farà altro che aumentare o diminuire il valore di left e top di 100 pixel, a seconda che vanga cliccato #a o #c.
Quindi per aumentare o diminuire il valore di un attributo basta fare:

+=valore
-=valore

Ma possiamo anche semplicemente settare il valore dell'attributo a cui vogliamo giungere con un animazione, ad esempio:

```
left: '100'
```

Sposterà il box a 100px da sinistra. Adesso divertitevi a creare le vostre animazioni usando altri attributi css :)

Chiamate Ajax

Negli ultimi anni si è sviluppata una tecnologia chiamata Ajax, Asynchronous JavaScript and XML, che consente di avere pagine web dinamiche, che si aggiornano "pezzo per pezzo", infatti ajax invia richieste in background al server, che restituirà una risposta (in XML, JSON, o altro) con la quale si può aggiornare un pezzo di una pagina web senza la necessità di dover ricaricare l'intera pagina. Il vantaggio sta nel fatto che, aggiornando solo un pezzo di pagina, il caricamento sia più veloce. Uno degli svantaggi è che, usando javascript, uno script potrebbe non essere cross-browser. Fortunatamente jQuery implementa tre funzioni che, innanzitutto, semplificano molto una chiamata Ajax, e che, quasi sempre, funzionano correttamente con moltissimi dei browser in circolazione.
Queste funzioni sono **$.ajax(), $.post(), $.get().**

In questa mia mini-guida creeremo un guestbook (alla buona) in php, e faremo si che si possano inserire i commenti senza dover ricaricare la pagina.

Premessa
Non mi venite a dire che le pagine non hanno captcha, quindi si può spammare o cose così. E' solo un esempio di uso di ajax.

Usiamo un db mysql. Quindi creiamo una tabella chiamata 'guestbook' con un campo 'commento' (naturalmente varchar), in più mettiamo un campo id settandolo **PRIMARY KEY NOT NULL AUTO_INCREMENT**.

Per i più svogliati:

```
CREATE TABLE IF NOT EXISTS guestbook (
   id int(30) PRIMARY KEY NOT NULL AUTO_INCREMENT,
  commento varchar(1000) DEFAULT NULL,
);
```

Quindi creiamo un file .php come se dovessimo creare il guestbook normalmente, senza jQuery, io l'ho fatto così:

```
<?php
$connection=mysql_connect("localhost", "*", "*");
mysql_select_db("*", $connection);

if(isset($_POST["commento"])) mysql_query("INSERT INTO
guestbook(commento)values('".mysql_real_escape_string(htmlentities($_
POST['commento'])."')", $connection);

$res=mysql_query("SELECT * FROM guestbook", $connection);

?>
<html>
<head>
<title>Prova Guestbook</title>
</head>
<body>
	<div id='comments'>
	<?php
		while($row=mysql_fetch_row($res)){
			echo '<p>'.$row[1].'</p>';
		}
		mysql_close($connection);
	?>
	</div>
	<form action='guestbook.php' method='post'>
		<textarea name='commento'></textarea>
		<input type='submit' value='invia' />
	</form>
</body>
```

Questo dovrebbe essere abbastanza chiaro no? :)

Potete comunque trovare questo esempio a:
http://pylinx.altervista.org/ajax_gst/guestbook_p.php

Adesso viene il momento di fare queste operazioni implementando ajax. Ora, io inserirò direttamente il codice intero della pagina e andrò a spiegarlo successivamente. Quello che vi dico, intanto, è che Ajax riceve risposte XML JSON ecc... Quindi è importante creare due pagine php, una che è che andrà ad interfacciarsi direttamente con l'utente, dove ci saranno tutti i commenti inseriti, il form per l'inserimento e dove faremo anche la chiamata ajax, che però andrà a fare una richiesta ad un'altra pagina php. Essa restituirà una risposta, per questo esempio, in xml, che noi rielaboreremo così da inserire quest'ultimo commento senza dover ricaricare la pagina.

La pagina principale:

```
<?php
$connection=mysql_connect("*", "*", "*");
mysql_select_db("*", $connection);

$res=mysql_query("SELECT * FROM guestbook", $connection);

?>
<html>
<head>
<title>Prova Guestbook</title>
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.6.1/jquery.min.js"></script>

<script type='text/Javascript'>
function addComment(){
      commento=$("textarea[name='commento']").val();
      $.ajax({
            url: 'inserisci.php',
            type: 'POST',
            dataType: 'xml',
            data: {commento: commento},
            success: function(xml){
                  $('#comments').append("<p>"+$(xml).find('last').text()+"<p>");

            }
      });
}
</script>
</head>
<body>
      <div id='comments'>
      <?php
            while($row=mysql_fetch_row($res)){
                  echo '<p>'.$row[1].'</p>';
            }
            mysql_close($connection);
      ?>
      </div>
      <form onsubmit='addComment(); return false;'>
            <textarea name='commento'></textarea>
            <input type='submit' value='invia' />
      </form>
</body>
```

La pagina per l'inserimento del commento che fornisce la risposta xml:

```
<?php
$connection=mysql_connect("localhost", "*", "*");
mysql_select_db("*", $connection);
if(isset($_POST["commento"])) mysql_query("INSERT INTO guestbook(commento)values('".mysql_
real_escape_string(htmlentities($_POST['commento'])."')", $connection);
$last=mysql_query("SELECT * FROM guestbook WHERE id=(SELECT MAX(id) FROM guestbook)", $con-
nection);
$last=mysql_fetch_row($last);
header('Content-Type: xml');
echo "<?xml version='1.0' encoding='UTF-8' ?>
      <c>
            <last>
                  ".$last[1]."
            </last>
      </c>";
mysql_close($connection);
?>
```

Potete trovare il guestbook all'indirizzo:

http://pylinx.altervista.org/ajax_gst/guestbook.php

La pagina che fornisce la risposta xml è inserisci.php (ma è inutile che vi dia il link).

Adesso tento di spiegare :)
Come vedete abbiamo la pagina guestbook.php che inizializza la connessione al db e scrive in output, nel div con **id='comments'** , ogni commento inserito finora. Dopo questo div abbiamo un form che contiene la textarea per l'inserimento del commento e il bottone per inviarlo. Ora, di default quando facciamo il **submit** di un form esso rimanda alla pagina specificata in action='pagina' , se action però non è presente, o non rimada a nessuna pagina, viene ricaricata la pagina in cui si trova il form. Noi vogliamo che quando il form viene inviato, invece di caricare una nuova pagina, esso esegua una funzione che contiene la nostra chiamata ajax, senza ricaricare la pagina corrente. Perciò dobbiamo settare il form in questo modo:

```
<form onsubmit='funzione(); return false;'>

…

</form>
```

Con il solo parametro **onsubmit** specificato, che rimanda alla nostra funzione e ritornerà false, dato che non vogliamo che carichi la pagina, quindi quel return false ci torna estremamente utile :)

Adesso l'unica cosa da fare è creare la funzione che esegue la chiamata ajax. Nel mio file l'ho chiamata **addComment()** ed è definita nella head. Sostituite il nome della funzione in **onsubmit** con quello della funzione in cui facciamo le nostre operazioni.
Andiamo adesso ad esaminare la nostra funzione:

```
function addComment(){
      commento=$("textarea[name='commento']").val();
      $.ajax({
            url: 'inserisci.php',
            type: 'POST',
            dataType: 'xml',
            data: {commento: commento},
            success: function(xml){
                  $('#comments').append("<p>"+$(xml).find('last').text()+"<p>");

            }
      });
}
```

Essendo una richiesta ad una pagina che richiede dei parametri (si possono fare anche chiamate ajax che non lo richiedano), dobbiamo avere tutte le variabili che ci servono, nel nostro caso abbiamo bisogno solo del commento dell'utente, in quanto lo script php inserisci.php lo inserirà nella tabella 'guestbook', per cui, la prima cosa da fare nella nostra funzione, è ottenere il valore della textarea quando il form viene inviato. Ricordate tutta la prima parte di tutorial su jQuery? E' il momento di usarla:

```
commento=$("textarea[name='commento']").val();
```

Con questa istruzione selezioniamo la textarea con **name='commento'** e ne prendiamo il valore (value) con la funzione *.val()*.
Adesso possiamo fare la nostra chiamata ajax, con la funzione $.ajax(). La funzione $.ajax() accetta un numero enorme di parametri (che non spiegherò), ma quelli principali si.

La funzione ha questa sintassi:

```
$.ajax({
     parametro: valore,
     parametro2: valore,
...
});
```

Vediamo un po' di questi parametri:

url:	E' l'unico parametro obbligatorio, esso vuole come valore una stringa in cui viene specificato, appunto, l'url della pagina a cui fare la richiesta ajax. Nel nostro caso **inserisci.php**
type:	E' il tipo di richiesta da fare, quindi 'POST' o 'GET' (o altre non supportate da tutti i browser). Se non lo specificate di default viene settato 'GET'.
dataType:	Il tipo di dato in cui sarà elaborata la risposta del server, nel nostro caso 'xml', ma può anche essere: html, json, script.
data:	Questo è molto importante, sono tutte le variabili da passare alla pagina a cui si fa la richiesta. Può essere: -**una stringa** così formata *"variabile=valore&variabile2=valore"* ovvero come in una richiesta 'GET' in un normale form inviato. -**un array** di nomi di variabili con i rispettivi valori (io lo preferisco così) così composto: *{variabile: valore, variabile2: valore}* Entrambi i metodi funzionano sia con 'POST' che con 'GET'.
success:	Che specifica cosa fare nel caso di successo della richiesta. Vuole come valore una funzione.
cache:	E' di default uguale a true, in caso venga impostato uguale a false, si forzerà il browser affinché non inserisca tra la cache la pagina a cui è inviata la richiesta.

Andiamo adesso ad analizzare la chiamata che facciamo nel nostro caso:

```
$.ajax({
            url: 'inserisci.php',
            type: 'POST',
            dataType: 'xml',
            data: {commento: commento},
            success: function(xml){
                  $('#comments').append("<p>"+$(xml).find('last').text()+"<p>");

            }
      });
```

Come vedete abbiamo passato come *url* la pagina *"inserisci.php"*, abbiamo impostato *type: "POST"*, e come *dataType* abbiamo messo *"xml"*, in quanto riceveremo una risposta in quel formato dal server, come data abbiamo passato un array (anche se ce n'è una sola) di variabili, dove abbiamo settato che la variabile *commento*: uguale a *commento* (la variabile creata prima della chiamata che contiene il valore della textarea).

Al momento dell'invio della richiesta viene controllato se essa ha avuto successo o meno, e abbiamo passato una *funzione a* **success**: che, in caso di successo, appende al *div* con *id='comments'* il testo dell'ultimo commento inserito, che ci viene inviato dal server in **xml**.

Andiamo adesso ad analizzare ciò che combina il server, nella pagina inserisci.php:

```
<?php
$connection=mysql_connect("localhost", "*", "*");
mysql_select_db("*", $connection);
if(isset($_POST["commento"])) mysql_query("INSERT INTO guestbook(commento)values('".mysql_
real_escape_string(htmlentities($_POST['commento'])."')", $connection);
$last=mysql_query("SELECT * FROM guestbook WHERE id=(SELECT MAX(id) FROM guestbook)", $con-
nection);
$last=mysql_fetch_row($last);
header('Content-Type: xml');
echo "<?xml version='1.0' encoding='UTF-8' ?>
      <c>
            <last>
                  ".$last[1]."
            </last>
      </c>";
mysql_close($connection);
?>
```

Come vedete prima viene effettuata la connessione al db, successivamente viene controllato che sia settata la variabile $_POST['commento'], che viene inserito nella tabella guestbook.

Successivamente viene fatta una select:

```
$last=mysql_query("SELECT * FROM guestbook WHERE id=(SELECT MAX(id) FROM guestbook)", $con-
nection);
```

Che seleziona tutto dal campo con l'id più alto, di fatti l'ultimo commento è quello con id più alto, dato che questo campo è **auto_increment.**

Successivamente passiamo un header:

```
header('Content-Type: xml');
```

Che specifica che la risposta è in xml.

Infine stampiamo in output la risposta xml:

```
echo "<?xml version='1.0' encoding='UTF-8' ?>
      <c>
            <last>
                  ".$last[1]."
            </last>
      </c>";
```

Come detto questo era solo un esempio, potevate tranquillamente settare il *dataType* nella chiamata ajax a *'html',* farvi dare una risposta in html dal server, ed inserire direttamente quella nel div id='comments'. Adesso non vi resta che provare :)

Con l'html potevate fare così.
La funzione addComment() vi diventa:

```
function addComment(){
      commento=$("textarea[name='commento']").val();
      $.ajax({
            url: 'inserisci.php',
            type: 'POST',
            dataType: 'html',
            data: {commento: commento},
            success: function(html){
                  $('#comments').append(html);
            }
      });
}
```

e cambiare, nel file inserisci.php, solo l'header e la risposta:

```
header('Content-Type: text/html');
echo "<p>".$last[1]."</p>";
```

Per fare le chiamate ajax in jQuery,come ho detto, esistono altre due funzioni che semplificano ancora di più il lavoro, ma è superfluo parlarne, dato che oramai sapete farle con $.ajax(), quindi ripeterei le solite cose (dato che i parametri sono meno che in $.ajax() e quelli che ci sono sono gli stessi), se volete dargli uno sguardo potete trovarle in inglese nella documentazione ufficiale:

http://api.jquery.com/jQuery.post/

http://api.jquery.com/jQuery.get/

Conclusioni

Spero di avervi incuriosito con questo tutorial e che adesso andrete a crearvi i vostri script che implementano jQuery. Boh, che devo dire poi? Spero di scrivere ancora per questa enzine :)

PyLinX

Bilancio di esercizio

Introduzione

Se mi chiedessero per quale motivo singolo e principale preferisco utilizzare GNU/Linux come sistema operativo, credo risponderei con la comodità di avere una bella serie di interpreti pronti all'uso dalla prima installazione.
I linguaggi di scripting sono normalmente molto intuitivi e sicuramente semplici da modificare anche per chi non conosce il linguaggio in questione; la contropartita è data dalla minore velocità di esecuzione.

Un metodo tipico di testare la velocità di un linguaggio è quello di misurare il tempo richiesto per l'esecuzione di loop abbastanza lunghi, tali da consentire la visualizzazione di una differenza nei tempi.
Nei due esempi seguenti andremo a testare un loop che calcoli e scriva il medesimo output con i quadrati dei numeri da 0 a 1.000; si mostrerà la differenza tra Bash e C, cioè i due linguaggi (uno di scripting e uno compilato) più diffusi in ambiente GNU/Linux.

```
#!/bin/bash

#
# Nome: squares.sh
# Scrive i valori dei quadrati tra 0 e 1000 con Bash
#

for (( i=0 ; i<=1000 ; i++ )); do
  echo -e "$i\t--> $(( $i * $i ))"
done
```

...altra versione più nobile:

```
#include <stdio.h>

/*
 * Nome: squares
 * Scrive i valori dei quadrati tra 0 e 1000 con C
 */

int main () {
  int i;
  for(i=0; i<=1000; i++)
    printf("%d\t--> %d\n", i, i*i);
  return 0;
}
```

È chiaro a tutti che il time del secondo programma avrà valori ben diversi dal primo script:

```
floatman@debian:~/UAH$ time ./squares.sh
0     --> 0
1     --> 1
.....
999   --> 998001
1000  --> 1000000

real 0m0.323s
user 0m0.120s
sys  0m0.040s
floatman@debian:~/UAH$ time ./squares
0     --> 0
1     --> 1
.....
999   --> 998001
1000  --> 1000000

real 0m0.009s
user 0m0.000s
sys  0m0.006s
```

Stato patrimoniale

Altra cosa chiara è che non tutti i linguaggi hanno uguale velocità di esecuzione, ad esempio Python è considerato il tipico linguaggio "lento", mentre Perl nell'immaginario collettivo è identificato come un tipico linguaggio ad esecuzione rapida:

```
#!/usr/bin/python

# Quadrati tra 0 e 1000 con Python:

for i in range(0, 1001):
  print i, "\t--> ", i*i

# EOF #

#!/usr/bin/perl

# Quadrati tra 0 e 1000 con Perl:

use strict;

for ( my $i = 0; $i <= 1000; $i++ ) {
  print "$i\t--> " . $i*$i . "\n"
}

# EOF #

floatman@debian:~/UAH$ time ./squares.py
0     -->  0
1     -->  1
.....
999   -->  998001
1000  -->  1000000
```

```
real 0m0.187s
user 0m0.048s
sys  0m0.044s
floatman@debian:~/UAH$ time ./squares.pl
0     --> 0
1     --> 1
.....
999   --> 998001
1000  --> 1000000

real 0m0.017s
user 0m0.004s
sys  0m0.008s
```

Come volevasi dimostrare Perl è estremamente rapido e viene infatti utilizzato anche in script piuttosto complessi dove la velocità di esecuzione inizia a diventare un parametro rilevante per l'usabilità del programma; oltretutto l'interprete è universalmente disponibile quindi il binomio lo renderebbe il linguaggio perfetto.

Fino a questo punto credo di aver detto cose ben conosciute; i tipici pilastri della razionalità che diventano assiomi fondamentali per chi lavora in quell'ambiente informatico che come sanno gli addetti ai lavori è molto meno aperto all'innovazione di quanto si crede.
Mentre girovagavo per la rete in cerca di un qualche appiglio per uscire da un CGI che mi aveva portato a rasentare il collasso nervoso, come fulmine a ciel sereno mi imbattevo in una cosa che avrebbe sconvolto completamente le poche sicurezze che avevo.
Il primo risultato della mia scoperta è stata la soluzione del mio problema CGIodale (=caffè + sigaretta + "ciao a domani!" immediati); il secondo risultato, molto più piacevole, lo state leggendo adesso...

Conto economico

Sebbene sia spesso snobbato, esiste un altro interprete disponibile in quasi tutte le distribuzioni che sebbene sembri il Calimero dei linguaggi di scripting raggiunge performance comparabili e superiori al suo ben più lodato parente...

```
#!/usr/bin/awk -f

# Quadrati tra 0 e 1000 con Awk:

BEGIN {

i=0
for (i=0; i <= 1000; i++) {
  printf i "\t--> " i*i "\n";
}

}

# EOF #

floatman@debian:~/UAH$ time ./squares.awk
0    --> 0
1    --> 1
.....
999  --> 998001
1000 --> 1000000

real 0m0.016s
user 0m0.002s
sys  0m0.005s
```

```
real 0m0.187s
user 0m0.048s
sys  0m0.044s
floatman@debian:~/UAH$ time ./squares.pl
0     --> 0
1     --> 1
.....
999   --> 998001
1000  --> 1000000

real 0m0.017s
user 0m0.004s
sys  0m0.008s
```

Come si è appena visto Awk è in grado di produrre ottimi risultati, oltre al fatto di essere il più delle volte disponibile in tutte le macchine almeno nelle sue moderne forme Nawk, Mawk o Gawk.
Una sintassi abbastanza semplice, un'ottima gestione di vari tipi di dati e delle espressioni regolari, una rapidità di esecuzione tra le migliori presenti sul mercato.
Insomma è una piattaforma molto potente che meriterebbe miglior apprezzamento, anche confrontandolo con i classici tool utilizzati normalmente.

Nell'esempio successivo dall'output del comando 'ls -R' nella home utente si andrà a sostituire ogni occorrenza della lettera 'a' con la lettera 'k'; operazione che d'istinto verrebbe eseguita in questo modo:

```
floatman@debian:~$ time ls -R | sed 's|a|k|g'
.....
wine-bin-unstkble_1.3.25-0.1_i386.deb
wine-unstkble_1.3.25-0.1_i386.deb

real 0m1.435s
user 0m0.272s
sys  0m0.232s
```

in realtà il metodo migliore è ancora il passaggio dei dati ad Awk:

```
floatman@debian:~$ time ls -R | awk '{gsub("a","k")}; 1'
.....
wine-bin-unstkble_1.3.25-0.1_i386.deb
wine-unstkble_1.3.25-0.1_i386.deb

real 0m1.245s
user 0m0.209s
sys  0m0.181s
```

Per precisione è doveroso dire che la funzione 'gsub' è presente in gawk, nawk e mawk. A questo punto in ogni caso credo sia evidente la necessità di un uso maggiore di questo linguaggio nella realizzazione di script.

Nello scripting da shell una delle cose che crea maggiori rallentamenti è l'utilizzo dei pipe (" | ") per l'elaborazione seriale dei dati. Un esempio di comando lento potrebbe essere dato prendendo gli script di esempio iniziale e sostituendo il nome 'script' con 'test' nell'output:

```
floatman@debian:~/UAH$ ls | grep script | sed 's|script|test|g'
test.awk
test.pl
test.py
test.sh
```

Su script complessi sta allo sviluppatore il miglioramento del rendimento in fase di creazione o correzione del programma, considerando il fatto che a volte la scelta è obbligata.
Un esempio tipico di quanto detto è il conteggio delle linee, normalmente attuato con 'head' e 'tail':

```
floatman@debian:~/UAH$ cat numerate.txt
prima
seconda
terza
quarta
quinta
sesta
settima
ottava
nona
decima
undicesima
dodicesima
tredicesima
quattordicesima
quindicesima
sedicesima
diciassettesima
diciottesima
diciannovesima
ventesima

#!/bin/bash

# richiesta.sh
# legge il numero di riga dell'input usando head e tail

richiesta=$1

output=$(head -n $1 numerate.txt | tail -n 1)

echo "testo della riga n.$richiesta: $output"

# ----- EOF ----- #
```

Di seguito proponiamo uno stesso script su shell che trasferisce ad Awk la gestione dell'output:

```
#!/bin/bash

# richiesta2.sh
# legge il numero di riga dell'input usando awk

richiesta="$1"

output=$(awk 'NR=="'"$richiesta"'"' numerate.txt)

echo "testo della riga n.$richiesta: $output"

# ----- EOF ----- #
```

Sebbene questo non vuole essere un manuale su Awk ma un invito ad un suo maggior utilizzo è necessaria qualche precisazione per chi non conosce il linguaggio.

L'unica operazione che esegue l'interprete è la lettura diretta della riga richiesta:

```
awk 'NR=="'"$richiesta"'"' numerate.txt
```

- 'NR' identifica esattamente il numero di riga (Number of Record), parametro gestibile tramite operatore di relazione nella stessa forma C-based della shell; in altre parole 'NR<=6' emula 'head -n 6' così come 'NR>=6' emula 'tail -n 6'.

- Cosa più importante è la gestione del trasferimento di variabili tra la shell e l'interprete Awk, attuato nella forma (non è l'unica) "'"$variabile"'" (apice doppio + singolo + doppio).

Anche se è chiaro il fatto che nel secondo script si presenta una singola operazione il volume di lavoro è minimo; riuscirà Awk a recuperare tempo?

```
floatman@debian:~/UAH$ time ./richiesta.sh 4
testo della riga n.4: quarta

real 0m0.009s
user 0m0.014s
sys  0m0.000s
floatman@debian:~/UAH$ time ./richiesta2.sh 4
testo della riga n.4: quarta

real 0m0.007s
user 0m0.012s
sys  0m0.000s
```

Per quanto non sembri, la differenza è enorme.

Nota integrativa

Oltre alle ben conosciute funzioni per il trattamento testi Awk presenta buone doti anche nel trattamento di dati numerici, campo in cui Bash fallisce miseramente (escluso passaggio da 'bc').

Un esempio carino potrebbe essere l'individuazione della media di una colonna di numeri.
Creiamo un file di testo a due colonne (media.txt) di questa forma:

```
124   285
296   836
936   284
681   513
468   954
374   127
```

media della prima colonna...

```
floatman@debian:~/UAH$ awk '{sum=sum+$1} END {print sum/NR}' media.txt
479.833
```

...e media della seconda colonna:

```
floatman@debian:~/UAH$ awk '{sum=sum+$2} END {print sum/NR}' media.txt
499.833
```

$1 e $2 identificano le due colonne, 'NR' è ancora il numero di riga visto in precedenza.

La questione però non si ferma a questo punto; ad esempio il mio interprete Mawk su Debian permette calcoli arrotondati fino alla quinta cifra decimale con una buona serie di funzioni matematiche decisamente utili:

```
#!/usr/bin/awk -f

# math.awk
# script dimostrazione di alcune funzioni
# matematiche gestite da Awk usando come
# esempio i numeri ‘5’ e ‘3’ oltre ad una
# approssimazione di P-Greco (PI) al nono
# decimale

BEGIN {
num1=5;
num2=3;
PI=3.141592654

print “Primo numero: 5”;
print “Secondo numero: 3”;

print “Somma: “, num1+num2;
print “Sottrazione: “, num1-num2;
print “Prodotto: “, num1*num2;
print “Divisione: “, num1/num2;

print “Rad.quadrata del quoziente: “, sqrt(num1/num2);
print “Log. (ln) del prodotto: “, log(num1*num2);
print “Esponenziale del log: “, exp(log(num1*num2));
print “..log(14) con arrotondamento (=2.639057):”, log(14);

print “Seno di 0: “, sin(0);
print “Coseno di 0: “, cos(0);

print “Area del triangolo b=5,h=3: “, num1*num2/2;
print “Circonferenza di raggio 5: “, 2*PI*num1;
print “Area del cerchio di raggio 3: “, PI*num2*num2;

exit;
}
```

l’output sarà il seguente

```
floatman@debian:~/UAH$ ./math.awk
Primo numero: 5
Secondo numero: 3
Somma:  8
Sottrazione:  2
Prodotto:  15
Divisione:  1.66667
Rad.quadrata del quoziente:  1.29099
Log. (ln) del prodotto:  2.70805
Esponenziale del log:  15
..log(14) con arrotondamento (=2.639057): 2.63906
Seno di 0:  0
Coseno di 0:  1
Area del triangolo b=5,h=3:  7.5
Circonferenza di raggio 5:  31.4159
Area del cerchio di raggio 3:  28.2743
```

Una capacità di calcolo non indifferente, soprattutto considerando il fatto che parliamo della forma semplice dell'interprete, senza l'ausilio di librerie o moduli vari di tipo più o meno esotico.
Ulteriori informazioni e script sono presenti in rete in gran volume, oltre alla consultazione del corposo manuale.
Molto interessante un documento su gnu.org che sarebbe troppo lungo da trattare: "TCP/IP internetworking with gawk".

Conclusione

Si potrebbe semplicemente dire che abbiamo dato alcuni esempi che mostrano quanto Awk possa essere uno strumento molto utile, la conclusione sarebbe però ovvia e non metterebbe in risalto la vera essenza generale di questo documento.
Ciò che è scritto in questo articolo è in verità la tipica scoperta dell'acqua calda; abbiamo studiato e sfruttato qualcosa che esiste in tutte le macchine anche se il suo uso resta al più relegato alla lettura delle colonne di dati nei comuni script su shell.
Questo fatto dovrebbe farci riflettere su quanto poco si riesce a spingere al massimo il proprio sistema studiandone ogni sua parte ed elaborando le funzioni più utili in modo da semplificare il nostro lavoro; in parole povere sarebbe bene strizzare al meglio le arance prima di comprare la cassetta nuova.
Forse la stessa rapidità dell'innovazione (non solo in campo informatico) ci fa perdere di vista quella stessa efficienza che il progresso dovrebbe aumentare, ciò che è più nuovo e più complesso appare per principio più utile e più adatto a qualunque esigenza.
Molte volte la soluzione migliore è talmente semplice e talmente sotto il nostro naso che quasi ci rifiutiamo di vederla o testarla, troppo scontata per essere la risposta, troppo ovvia per essere una valida soluzione.
Evidentemente il concetto caro all'hacking del non reinventare la ruota non è poi così intuitivo né così semplice da adottare.
Non è sufficiente la sola esperienza per risolvere al meglio i problemi, è necessaria l'intuizione (un comportamento innato) così come la capacità di mettere sempre in gioco il complesso delle proprie credenze.
Non si finisce veramente mai di imparare...

floatman

Parallelizzare con OpenMP

Sfruttiamo la potenza dei processori multicore

Introduzione

La legge di Moore ci ha ormai abituati ad hardware sempre più performante, e i chipmaker hanno fatto, e continuano a fare, a gara per poter vendere processori sempre più potenti. Fino all'avvento dei processori dual core, si pensava di poter aumentare la potenza di un pc aumentandone il clock, ma il raggiungimento dei limiti strutturali dei chip spostarono l'attenzione di Intel e AMD sull'unione di più processori, creando processori dual core, quad core... e ora è difficile non avere sulle nostre macchine dei processori multicore!

E, dato che sono sempre più diffusi, perchè non sfruttarli in applicazioni reali?

Molte applicazioni che utilizzano calcolo multicore sono relative a simulazioni fisiche o chimiche, come le simulazioni di aero/fluidodinamica, oppure elaborazioni di flussi audio/video, motori di ricerca, data mining, protein folding, intelligenza artificiale, computer vision... e tutte queste applicazioni devono spesso rispondere nel minor tempo possibile, talvolta anche in real-time. Tutto questo calcolo deve essere in qualche maniera velocizzato, anche sfruttando le caratteristiche dei nuovi processori multicore che, ho detto prima, sono molto diffusi.

OpenMP

OpenMP (Open Multi-Processing) è lo standard *de-facto* per la programmazione parallela a memoria condivisa su CPU, consiste in una serie di direttive che permettono di sincronizzare thread, ottenendo quindi un'alta velocità, senza avere a che fare direttamente con gli stessi: la maggior parte del lavoro di sincronizzazione è gestita autonomamente.

Disponibile sulla maggior parte dei compilatori, tra cui GCC/G++ e Visual Studio, è un prodotto molto utilizzato, soprattutto in ambito scientifico. GCC è stato uno dei primi compilatori, dalla versione 4.3.1, ad implementare le specifiche 3.0, rilasciate nel maggio 2008. Le specifiche di OpenMP sono state estese anche al Fortran, linguaggio ancora molto usato in ambito HPC.

A running example

Vi propongo un esempio: un piccolo programma in C++ che sfrutta Dijkstra per avere tutte le distanze tra tutti i punti di un grafo orientato e pesato.

```
#include <stdio.h>
#include <stdlib.h>
#include <climits>

#define NODES 1001
#define PATH "graph_file.txt"
int matrix[NODES][NODES];
bool visited[NODES][NODES];
int fullpaths[NODES][NODES];

void dijkstra(int startnode)
{
   bool isGraphUpdated=true;
   int j, m, i;
   do
   {
     //cerchiamo percorsi più brevi.
     for(m=INT_MAX, i=0; i<NODES; i++)
         if(!visited[startnode][i] && fullpaths[startnode][i]<=m)
            m=fullpaths[startnode][j=i];

      visited[startnode][j]=true;
      if(m==INT_MAX) isGraphUpdated=false;  //stop

      if (isGraphUpdated)
      {
            //aggiorniamo il grafo
            for(i=0; i<NODES; i++)
            {
                   if(matrix[j][i] && matrix[j][i]!=INT_MAX
            && fullpaths[startnode][i]>fullpaths[startnode][j]+matrix[j][i])
                   {
                   fullpaths[startnode][i]=fullpaths[startnode][j]+matrix[j][i];
                   }
            }
        }
    }while(isGraphUpdated);
}
```

La funzione "dijkstra" non fa nient'altro che (ovviamente) applicare l'algoritmo di Dijkstra sul grafo a partire dal nodo "startnode", ottenendo tutte le distanze dei percorsi possibili da startnode.

```
{
     /* assumiamo che le matrici fullpaths e matrix siano già
               inizializzate, esattamente con i soliti valori, che corrispondono
               al grafo di partenza. la matrice visited è inizializzata
               con tutti gli elementi a falso */
     for(int i=0; i<NODES; i++)
            {
                dijkstra(i);
     }
     return 0;
}
```

Questo è il main, che richiama N volte la funzione "dijkstra", che ha complessità computazionale N^2. Ora il nostro codice è seriale, ovvero verrà eseguito da un solo processore.

Ora provate ad aggiungere questa riga, prima del ciclo for.

```
//codice precedente
//carichiamo l'array, non è importante come
#pragma omp parallel for
for(int i=0; i<NODES; i++)
    //codice successivo
```

Compilate con l'opzione -fopenmp e lanciate. Ora il calcolo è splittato su più thread, e sarà sicuramente molto più veloce!
Su un grafo di 1000 nodi, si è avuto uno speedup del 371%, passando da un tempo di 17.481 secondi (sequenziale) a 4.716 secondi (parallelo, 4 thread attivi). E' possibile, inserendo una sola riga di codice, avere questo speedup? Sì, potere di OpenMP :D
In questo esempio non abbiamo problemi di sincronizzazione in scrittura sulla matrice *fullpaths*, quindi è complice anche il fatto che il codice è stato strutturato per funzionare in parallelo, ma è OpenMP a fare tutto il lavoro :D

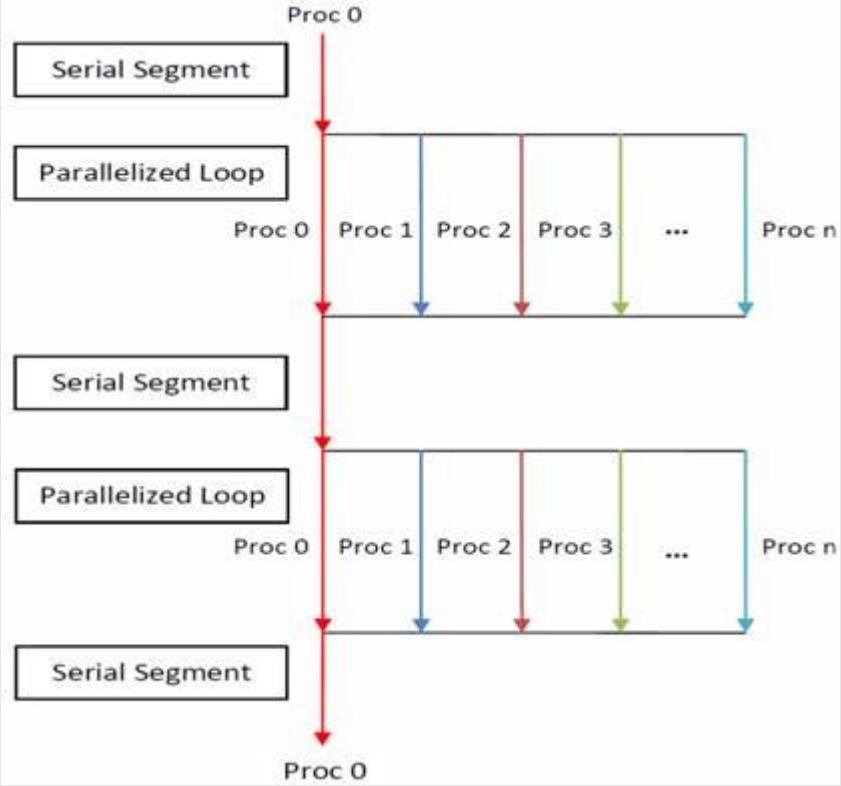
Come funziona?

Il modello di esecuzione, come descritto dalle specifiche di OpenMP, si basa sul modello fork-join.
Quando l'eseguibile parte, esiste un solo thread, chiamato "thread iniziale", che lavora sequenzialmente. Quando trova una direttiva "parallel" crea un gruppo di thread (fork) e diventa il capo del nuovo gruppo ("team master"). Il team master assegna ad ogni thread il blocco di codice compreso nella direttiva "parallel", ed ognuno comincia ad eseguirlo, fino a quando non ha terminato l'esecuzione (join). Il team master è l'unico che rimane attivo, e continua l'esecuzione in modo sequenziale. Per esempio:

```
#include <omp.h>
#include <iostream>
using namespace std;
int main()
{
    omp_set_num_threads(4);
    #pragma omp parallel //questa è una direttiva "parallel"
    {
       cout<<"ciao, il mio id è:\t"<<omp_get_thread_num()<<endl;
    }
    //solo il master è sopravvissuto!
    cout<<"io sono l'unico rimasto! "<<omp_get_thread_num()<<endl;
    return 0;
}
```

in questo esempio vengono creati 3 nuovi thread (4 compreso il thread master, che è anche il thread iniziale), che stampano in maniera concorrente la scritta "ciao, il mio id è:", seguito dall'identificativo del thread, quindi comparirà a schermo l'unica scritta "io sono l'unico rimasto! 0".

Penso che un'immagine sia più chiara di tutte le parole che ho utilizzato.



Nel caso ci siano delle direttive "parallel" annidate ogni thread del team creato nel livello superiore diventa il team master di un'ulteriore team, che esegue il blocco annidato in maniera parallela.
La differenza tra i due esempi finora presentati, è che nel secondo i thread eseguono **solo una volta** il codice assegnato, mentre nel primo caso i thread **si dividevano le iterazioni**: ci sono 1000 iterazioni? ogni thread si assume il compito di fare 250, in concorrenza con gli altri thread del solito team: questo tipo di direttive si chiama **"worksharing construct"**, che sono studiati appositamente per distribuire tra i thread l'esecuzione del blocco di codice compreso nel ciclo for.

Ora dobbiamo sincronizzare...

Esistono diverse direttive e varie funzioni che ci permettono di sincronizzare i vari thread nel nostro codice OpenMP, qui elenco le più interessanti:

- direttiva **master**: specifica un blocco di codice eseguito solo dal team master, gli altri thread continuano ignorandolo.

```
#pragma omp master
{
    /*codice*/
}
```

- direttiva ***barrier***: direttiva che blocca l'esecuzione dei thread che hanno "incontrato" la barriera -avendo concluso il lavoro precedente- fino a quando **tutti** i thread non l'hanno raggiunta. I thread che hanno terminato l'esecuzione precedentemente "idlano", ovvero rimangono in attesa senza fare calcoli.

```
#pragma omp barrier
```

- direttiva **critical**: crea una zona critica per l'esecuzione del codice indicat

```
#pragma omp critical
{
     /*codice*/
}
```

questa particolare direttiva permette di creare una zona critica, ovvero una parte di codice eseguita da un solo thread alla volta, mentre gli altri attendono il loro turno. Normalmente il codice delimitato da questa direttiva legge o scrive variabili globali, come dei buffer, in cui altri thread possono accedere modificando il contenuto.

- direttiva **atomic**: assicura che l'operazione delimitata dalla direttiva non sia interrotta o disturbata da altri thread, gestisce autonomamente la concorrenza.

```
#pragma omp atomic
  x binop=expr;
```

questa direttiva può essere utilizzata solo per semplici modifiche nella forma *x binop=expr*, come shift, operatori binari, incrementi o espressioni in cui la variabile x non compare all'interno di expr. E' stata sviluppata appositamente per gestire la race condition, ovvero la possibilità che due thread vadano a scrivere sulla stessa variabile contemporaneamente, occupandosi di gestire la zona critica autonomamente.

- direttiva **flush**: esegue l'operazione flush di OpenMP, che rende consistente (aggiorna) la memoria locale del thread con quella globale condivisa tra tutti i thread.

```
#pragma omp flush ([variabile0, variabile1, … , variabileN])
```

- direttiva **ordered**: permette l'esecuzione sequenziale di un blocco di codice.

```
#pragma omp ordered
{/*codice*/}
```

questa direttiva costringe i thread ad avanzare uno alla volta, con l'effetto di rendere sequenziale l'esecuzione del blocco di codice inserito nella direttiva ordered.

L'utilizzo di queste direttive viene evidenziato in questa versione del problema dei cinque filosofi a cena. Questo problema è un esempio di condivisione delle risorse da parte di processi concorrenti, in cui vanno gestite correttamente le possibili problematiche: ci sono cinque filosofi attorno ad un tavolo, dove ci sono cinque forchette, ma ogni filosofo necessita di *due* forchette per poter mangiare. I filosofi possono alternativamente pensare o mangiare, oppure aspettare di avere disponibili le forchette per poter mangiare. I problemi principali sono due: lo stallo (deadlock), in cui tutti aspettano una forchetta ma non liberano quella che hanno in mano, e la "morte per fame" (starvation), dove può esserci qualche filosofo che non riesce ad accedere alle forchette, quindi non mangia mai.
La soluzione si basa, essenzialmente, sul prendere -per mangiare- le due forchette insieme e rilasciarle -per pensare- nello stesso modo, quindi si ha bisogno di una **zona critica** in cui nessun altro filosofo può prendere (o posare) le forchette mentre la sta prendendo (posando) un altro, ed ogni filosofo deve aspettare che il vicino abbia finito per poter prendere le forchette. In OpenMP è semplice creare una zona critica, come visto precedentemente:

```
#include <iostream>
#include <time.h>
#include <stdlib.h>
#include <unistd.h>
#include <omp.h>
using namespace std;
```

```
#define NUMERO_FILOSOFI 5
bool forks[NUMERO_FILOSOFI];
int main()
{
  srand(time(NULL));
  for(int i=0; i<NUMERO_FILOSOFI; i++) forks[i]=false;
  omp_set_num_threads(NUMERO_FILOSOFI); //1 filosofo=1 thread
  #pragma omp parallel shared(forks)
  {
     while(true)
     {
         int id=omp_get_thread_num();
         cout<<"il filosofo "<<id<<"sta pensando"<<endl;
                 //simuliamo il tempo del filosofo pensante
         sleep(rand()%10/1000.0);
         cout<<"il filosofo "<<id<<"ha fame"<<endl;

          #pragma omp flush //aggiorniamo tutte le variabili condivise
         #pragma omp critical  //ecco la nostra zona critica!
         {
                    if(!forks[i] && !forks[(i+1)%NUMERO_FILOSOFI])
                   {
                        cout<<"il filosofo "<<id<<"sta mangiando"<<endl;
                        //blocchiamo le forchette
                        forks[id]=true; forks[(id+1)%NUMERO_FILOSOFI]=true;
                        //simuliamo il tempo che il filosofo blocca le forchette mangiando
                        sleep(rand()%20/1000.0);
                        cout<<"il filosofo "<<id<<"è sazio"<<endl;
                    }
         }
                 //aspettiamo che tutti i thread siano entrati nella zona critica, in modo
da avere
                 //non solo 1 thread alla volta che blocca le forchette, ma dando la pos-
sibilità anche agli
                 //altri filosofi di mangiare, ma solo se le forchette sono libere.
                 #pragma omp barrier
                 //rilasciamo le forchette, stavolta senza una zona critica, dato che la
                 //riacquisizione si svolge all'inizio e non c'è il rischio di ottenere
valori non validi.
                 if(forks[i] && forks[(i+1)%NUMERO_FILOSOFI])
                 { forks[i]=false; forks[(i+1)%NUMERO_FILOSOFI]=false; }
            }

  }
  return 0;
}
```

Come possiamo vedere, OpenMP stesso crea una zona critica e si occupa di gestirla, implementando la soluzione più adatta, e tutto in maniera automatica! Semplice, no?

E' anche possibile gestire la **concorrenza in modo manuale, utilizzando i lock.** OpenMP mette a disposizione delle funzioni specifiche per creare, bloccare / sbloccare, testare e distruggere lock, di semplice utilizzo.
Un possibile sorgente per la risoluzione del problema dei cinque filosofi può essere il seguente:

```
#include <iostream>
#include <time.h>
#include <stdlib.h>
#include <unistd.h>
#include <omp.h>
using namespace std;

//filosofi.cpp
```

```
#define NUMERO_FILOSOFI 5
//gli oggetti di tipo omp_lock_t sono i lock utilizzati da OpenMP
omp_lock_t locks[NUMERO_FILOSOFI];

omp_lock_t *get_right_fork(int id)
{
  return &locks[(id+1)%NUMERO_FILOSOFI];
}

omp_lock_t *get_left_fork(int id)
{
  return &locks[id%NUMERO_FILOSOFI];
}

void eat(int id)
{
  cout<<"il filosofo "<<id<<" sta mangiando"<<endl;
 sleep(rand()%20/1000.0);
  cout<<"il filosofo "<<id<<" è sazio"<<endl;
}

void think(int id)
{
  cout<<"il filosofo "<<id<<" sta pensando"<<endl;
 sleep(rand()%10/1000.0);
  cout<<"il filosofo "<<id<<" ha fame"<<endl;
}

int main()
{

  srand(time(NULL));
  for(int i=0; i<NUMERO_FILOSOFI; i++)
     omp_init_lock(&locks[i]);  //inizializziamo i thread con omp_init_lock(omp_lock_t*)

  omp_set_num_threads(NUMERO_FILOSOFI); //1 filosofo=1 thread
  #pragma omp parallel
  {
     while(true)
     {
         int id=omp_get_thread_num();
         think(id);
         if(id&1) //controllo per evitare lo stallo
         {
                     omp_set_lock(get_left_fork(id)); //blocchiamo un lock...
              omp_set_lock(get_right_fork(id));
          }
          else
          {
                     omp_set_lock(get_right_fork(id));
              omp_set_lock(get_left_fork(id));
          }
          eat(id);
          omp_unset_lock(get_left_fork(id));  //... e lo sblocchiamo!
          omp_unset_lock(get_right_fork(id));
     }
  }
  for(int i=0; i<NUMERO_FILOSOFI; i++) omp_destroy_lock(&locks[i]); //distruggiamo i lock
  return 0;
}
```

Le funzioni sui lock sono molto chiare, l'unico parametro che hanno è un omp_lock_t*, su cui operano l'operazione descritta nel nome. Notare che omp_set_lock è una funzione bloccante, che rimane in attesa che venga sbloccato per poterlo bloccare. Una funzione non utilizzata qui è la funzione omp_test_lock(omp_lock_t*), che restituisce il valore del lock (zero se libero, uno se bloccato) e lo blocca nel caso sia libero; può essere utile come condizione di un ciclo while per gestire operazioni che i thread possono eseguire nel frattempo che il lock si libera.

Pro e Contro

OpenMP viene **fortemente utilizzato in ambiente HPC** (High performance computing, cioè il calcolo ad alte prestazioni fatto sui supercomputer) per il calcolo con dati sulla stessa macchina: se gli scienziati lo utilizzano efficacemente, perchè noi no? A parte questa precisazione, OpenMP può essere utile quando abbiamo bisogno di velocizzare un'applicazione all'ultimo minuto: l'esempio di Dijkstra con l'aggiunta del pragma parallelo è un esempio eclatante! Questo è possibile perchè **non si ha bisogno, con OpenMP, di dover riprogettare l'applicazione**, e questo è indubbiamente un vantaggio! Inoltre, se si compila senza specificare l'opzione -fopenmp i pragma vengono ignorati, rendendo seriale il codice parallelo: ottima cosa se il codice ha subito un'ottimizzazione dell'ultimo momento!

Un'ulteriore vantaggio per adottare OpenMP è la sua natura **cross-platform**: essendo uno standard è disponibile sui compilatori più popolari, di cui esistono versioni per molti sistemi operativi esistenti, quindi si può portare tranquillamente il codice scritto su Linux su pc Windows, quindi testarlo su Solaris o su Mac semplicemente ricompilando.

A fronte di una grande efficacia, si presentano problemi pratici come la **difficoltà di debuggare** eventuali problemi di sincronizzazione e di race condition: il debug su questo tipo di applicazioni può essere molto difficoltoso, ed **OpenMP non genera eccezioni** nel caso ci siano errori, nè è dotato di strumenti di debug, lasciando silenziosi bachi che si manifestano solo in produzione.

Un altro problema di OpenMP, comune alla maggior parte delle librerie di parallelizzazione, è quello della **dimensione dei dati**: se lanciamo un eseguibile OpenMP con dati di piccola dimensione, si perderà molto più tempo nello startup dei thread e della loro sincronizzazione che non nel calcolo vero e proprio. Naturalmente se si vuole parallelizzare un'applicazione è a causa di un'*effettiva necessità*, quindi bisogna valutare a priori se conviene o meno parallelizzare il codice, per evitare di trovarsi in queste situazioni.

Ah, e non pensate che questa libreria sia efficace per programmare anche su GPU! Per quello c'è CUDA, OpenMP serve **solo per calcolo multiprocessore su CPU!**

Conclusione

Sfruttando la potenza di più processori si ha la possibilità di velocizzare enormemente i calcoli, incrementando la velocità di esecuzione e un miglior sfruttamento della macchina stessa; ulteriori vantaggi e svantaggi li ho appena spiegati. Spero che questo articolo vi abbia incuriosito riguardo questa possibilità e alla sua implementazione tramite OpenMP. Questo articolo non vuol essere un invito ad utilizzare soltanto OpenMP, ma è un invito a sperimentare questa libreria come base per poter imparare a sviluppare in maniera parallela anche con altre librerie, come pthread, MPI, CUDA o OpenCL. Enjoy the parallelization!

Alessandro "alfateam123" Balzano

Riferimenti

http://openmp.org/ sito ufficiale di OpenMP

http://heather.cs.ucdavis.edu/~matloff/ParProcBook.pdf Ebook open source, mi ha aperto un mondo: shared memory, message passing, OpenMP, CUDA, MPI, algoritmi paralleli. Ve lo consiglio.

http://msdn.microsoft.com/it-it/magazine/cc163717%28en-us%29.aspx#S8 articolo su MSDN riguardo OpenMP. ah, e potete trovare molta documentazione sull'MSDN che vale sia per lo sviluppo con Visual Studio che con gcc.

http://www.michaelsuess.net/publications/suess_leopold_common_mistakes_06.pdf Pdf contenente esempi di errori comuni in OpenMP e indicazioni su come evitarli.

Note finali di UnderAttHack

Per informazioni, richieste, critiche, suggerimenti o semplicemente per farci sapere che anche voi esistete, contattateci via e-mail all'indirizzo **underatthack@gmail.com** Siete pregati cortesemente di indicare se non volete essere presenti nella posta dei lettori.

Allo stesso indirizzo e-mail sarà possibile rivolgersi nel caso si desideri collaborare o inviare i propri articoli.

Per chi avesse apprezzato UnderAttHack, si comunica che l'uscita (il numero 16) è prevista alla data di:

venerdì 25 Novembre 2011

Come per questo numero, l'e-zine sarà scaricabile nel formato PDF al sito ufficiale del progetto:
http://underatthack.org