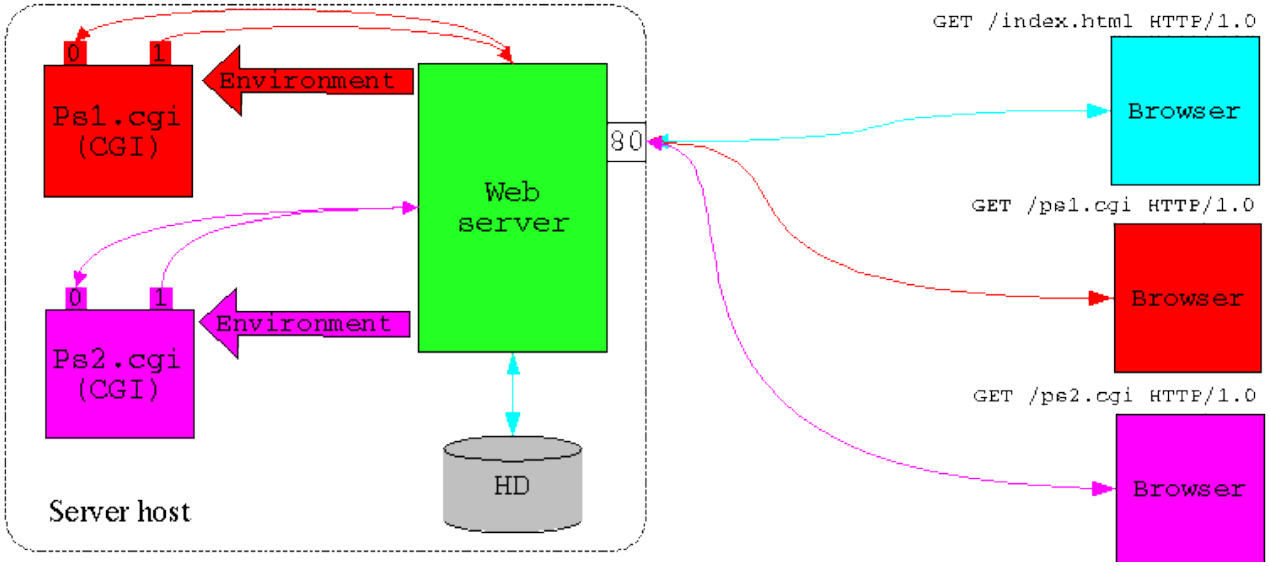
Lezione 5

Architettura CGI

Le richieste di risorse statiche vengono esaudite direttamente dal web server, mentre per quelle in cui c'è bisogno di eseguire un'applicazione separata, si crea un processo figlio e tramite la ridirezione dell'input/output la richiesta viene inviata all'applicazione. Schematicamente abbiamo la seguente situazione:



Vediamo dallo schema che il web-server deve essere in grado di:

- distinguere tra richieste di pagine statiche e invocazioni di applicazioni CGI: nella figura abbiamo aggiunto il suffisso .cgi per indicare le applicazioni CGI.
- distinguere tra applicazioni CGI;
- creare l'**environment** per l'applicazione CGI;
- creato un **processo** per ogni invocazione di applicazione CGI;
- "deviare" (**ridirezionare)** la connessione con il client nello stdin e stdout del processo creato.

Per distinguere le applicazioni CGI dalle pagine statiche, ci sono due metodi principali:

- i file che terminano con un suffisso particolare (generalmente `.cgi`);
- i file contenuti in particolari directory (generalmente la directory `cgi-bin`).

Ad esempio il web server Apache, ha il file di configurazione `/etc/httpd/conf/httpd.conf` in cui possiamo specificare i suffissi che rappresentano le applicazioni CGI nel seguente modo:

```
AddHandler cgi-script .cgi
```

Inoltre possiamo specificare le directory che contengono solo applicazioni CGI e quindi indipendentemente dal suffisso vengono trattati come tali, nel seguente modo:

```
ScriptAlias /cgi-bin/ "/home/httpd/cgi-bin/"
```

Dobbiamo ricordarci anche che ogni directory in cui sono contenuti delle applicazioni CGI deve avere impostato l'opzione per eseguire i CGI stessi.

```
#
# "/home/httpd/cgi-bin" should be changed to whatever your
ScriptAliased
# CGI directory exists, if you have that configured.
#
<Directory "/home/httpd/cgi-bin">
    AllowOverride None
```

```
    Options ExecCGI
    Order allow,deny
    Allow from all
</Directory>
```
Per avere una directory di applicazioni CGI nella home di ogni utente bisogna aggiungere la seguente dichiarazione:
```
<Directory /home/*/public_html/cgi-bin>
    Options ExecCGI
    SetHandler cgi-script
</Directory>
```

Esempio

Il primo esempio consiste nel programma C per visualizzare la pagina html con scritto “Hello World!”.

```
/*********************************************************
*  cgiweb.c                                              *
*********************************************************/
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#define BUFLEN 1024
char buf[BUFLEN];
main(int argc, char *argv[])
{
    write(1,"Content-type: text/html\015\012",25);
    write(1,"Status: 200 OK\015\012",17);
    write(1,"\015\012",2);
    while(1,"<html><body>Hello World!</body></html>",38);
    close(file);
}
```

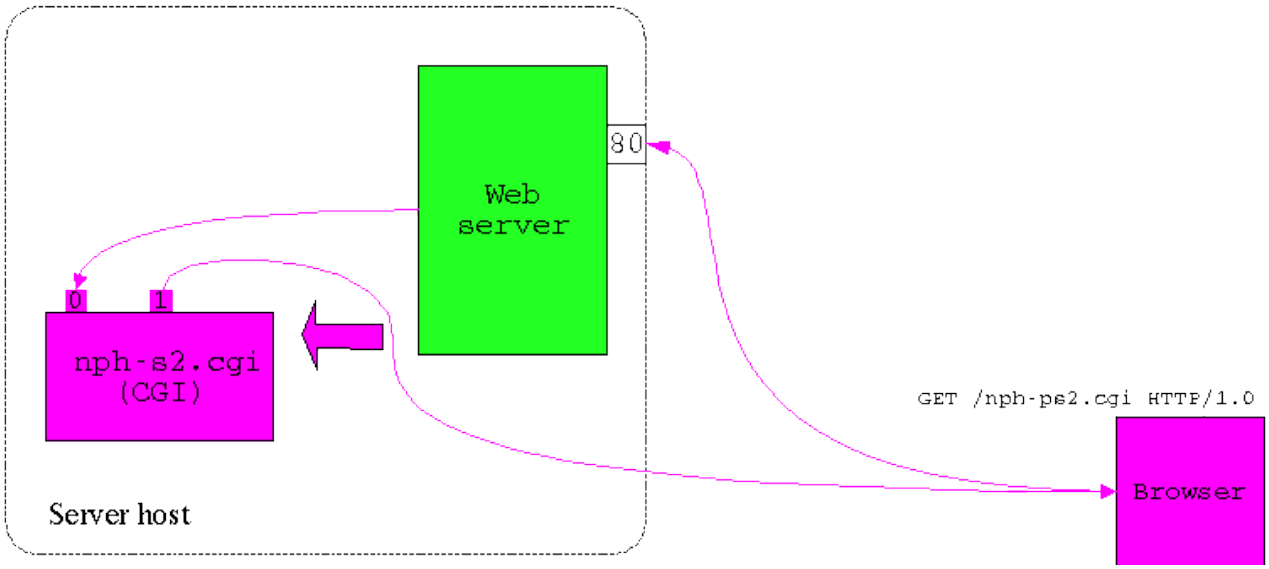
L'esempio seguente visualizza il file env.html:

```
/*********************************************************
*  cgiweb.c                                              *
*********************************************************/
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#define BUFLEN 1024
char buf[BUFLEN];
main(int argc, char *argv[])
{
    int n=0,i=0;
    int file;

    file=open("/public/taw/lezione3/env.html", O_RDONLY);
    if (file<0)
    {
      write(1,"Content-type: text/html\015\012",25);
      write(1,"Status: 404 NOT FOUND\015\012",24);
      write(1,"\015\012",2);
      return 0;
    }
    write(1,"Content-type: text/html\015\012",25);
    write(1,"Status: 200 OK\015\012",17);
    write(1,"\015\012",2);
    while ((n = read(file, buf, BUFLEN)) > 0)
       write(1, buf, n);
    close(file);
}
```

Possiamo anche inviare direttamente la risposta al cliente senza bisogno di passare dal server, in questo caso bisogna inviare glie header esattamente come previsto dal protocollo HTTP perché lo standard CGI prevede che per le applicazioni che iniziano per `nph-` (not parse headers), l'output venga inviato direttamente al client senza che il server esamini il contenuto dello header.

```
/*********************************************************
*  nph-cgiweb.c                                          *
*********************************************************/
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#define BUFLEN 1024
char buf[BUFLEN];
main(int argc, char *argv[])
{
    int n=0,i=0;
    int file;

    file=open("/public/taw/lezione2/env.html", O_RDONLY);
    if (file<0)
    {
      write(1,"HTTP/1.0 404 NOT FOUND\015\012",24);
      write(1,"\015\012",2);
      return 0;
    }
    write(1,"HTTP/1.0 200 OK\015\012",17);
    write(1,"Content-type: text/html\015\012",25);
    write(1,"\015\012",2);
    while ((n = read(file, buf, BUFLEN)) > 0)
       write(1, buf, n);
    close(file);
}
```

Questo comportamento è illustrato dal seguente schema:



Invocazione di un'applicazione CGI

Un'applicazione CGI può essere invocata da un client in questi modi:

- tramite link in questo caso il metodo di invocazione e GET (o HEAD);
- tramite la sottomissione di un FORM in questo caso il metodo può essere sia GET che POST.

Nel caso del metodo GET, il server riceve una richiesta del tipo:

```
GET /nome-cgi HTTP/1.0
...
```

oppure nel caso siano specificati degli argomenti da passare alla applicazione:

```
GET /nome-cgi?
nome_arg1=val_arg1&nome_arg2=val_arg2&..&nome_argn=val_arg_n HTTP/1.0
...
```

Nel caso del metodo POST, il server riceve sempre una richiesta del tipo:

```
POST /nome-cgi HTTP/1.0
....
Content-type: ....
Content-lenght: nnnn

XXX
```

e gli eventuali argomenti opportunamente codificati rappresentano il contenuto della risorsa inviata al server, quindi compaiono dopo lo header e sono indicati da XXX nell'esempio. La codifica può essere:

- `application/x-www-form-urlencoded`
- `multipart/form-data`

Nel primo caso, la codifica è la stessa che per il metodo GET, quindi `XXX` sarebbe uguale a `nome_arg1=val_arg1&nome_arg2=val_arg2&..&nome_argn=val_arg_n` . Tutti i caratteri con significato particolare vengono codificati in ASCII esadecimale anteponendo il carattere `%`, quindi ad esempio se nel valore compare il simbolo & questo viene sostituito dai caratteri `%26`.
Nel secondo caso, la codifica la esamineremo successivamente.

environment

Tutti i dati dello header della richiesta del client vengono analizzati dal server che li deve rendere disponibili alla applicazione CGI. Il passaggio di queste informazioni e di altre che solo il server conosce avviene attraverso l'ambiente passato al processo dell'applicazione CGI. La lista completa delle informazioni passate al processo è:

Le seguenti informazioni non dipendono dalla richiesta e sono le stesse per ogni richiesta inviata:

- `SERVER_SOFTWARE`

  Il nome e la versione del web server. Formato nome/versione;

- `SERVER_NAME`

  Il nome, DNS alias, o IP address della macchina che ospita il server.

- `GATEWAY_INTERFACE`

  La versione delle specifiche CGI che il server adotta.Formato: CGI/revisione.

Le seguenti informazioni sono specifiche della richiesta inviata:

- `SERVER_PROTOCOL`

  Il nome del protocollo e la versione specificato nella richiesta del client. Formato: protocollo/revisione

- `SERVER_PORT`

  Il numero di porta alla quale la richiesta è stata inviata, di solito 80.

- `REQUEST_METHOD`

  Il metodo specificato nella richiesta: GET, HEAD, POST, etc.

- `PATH_INFO`

  Il path specificato nella richiesta che segue il path del CGI stessto (extra path). In altre parole gli script CGI possono essere richiamati usando il loro path virtuale (cioè il path visto dal client) seguito da ulteriori informazioni alla fine del path virtuale. Le ulteriori informazioni sono presenti in PATH_INFO.

- `PATH_TRANSLATED`

  Il server mappa il contenuto di PATH_INFO trasformandolo nel path corrispondente della macchina locale.

- `SCRIPT_NAME`

  Il path virtuale dello script CGI richiamato. Usato per scrivere codice autoreferente.

- `QUERY_STRING`

  L'informazione che segue il simbolo '?' nell'URL della richiesta.

- `REMOTE_HOST`

  Il nome della macchina del client. Se il server non conosce il nome della macchina del client, dovrebbe impostare solo REMOTE_ADDR.

- `REMOTE_ADDR`

  L'IP address della macchina del client.

- `AUTH_TYPE`

  Il protocollo di autenticazione utilizzato per autenticare il client.

- `REMOTE_USER`

  L'username del client.

- `REMOTE_IDENT`

  Se il server HTTP supporta il protocollo RFC 931 per identificazione, allora questa variabile contiene il nome dell'utente remoto recupaerato dal server.

- `CONTENT_TYPE`

  Per richieste con inviano informazioni come HTTP POST e PUT, questa variabile dovrebbe contenere il tipo di contenuto.

- `CONTENT_LENGTH`

  La lunghezza del contenuto così come viene indicata dal client.

In aggiunta agli header appena visti, tutti glie eventuali altri header inviati dal client vengono messi nell'environment con il prefisso HTTP_ seguiti dal nome dell'header. Ogni carattere '-' nel nome dell'header viene sostituito dal carattere '_ '. Il server può escludere tutti gli header che ha già processato quali Authorization, Content-type, e Content-length. Se necessario il server può escludere ogni altro header se si superano eventuali limiti nell'uso delle risorse di sistema,

Esempi di questi variabili che possono essere escluse:

- `HTTP_ACCEPT`

  I tipi MIME che il cliente accetta come risposta. Ogni tipo dev'essere separato da una virgola.

  Formatp: tipo/sottotipo, tipo/sottotipo.

- `HTTP_USER_AGENT`

  Il nome del programma software usato come client. Formato: `software/version library/version`.

Dal punto di vista del programmatore di applicazioni CGI ci interessa sapere come accedere a queste informazioni. Da un programma scritto in C possiamo accedere alle variabili d'ambiente con la funzione `getenv`, mentre da uno script della shell basta anteporre il simbolo `$` ai nomi delle variabili.

stdin

Dallo standard input si può leggere tutto quello che il client invia dopo lo header al server. Quindi nulla per i metodi GET e HEAD. Mentre per i metodi POST e PUT potremmo leggere un contenuto di tipo `CONTENT_TYPE` e lunghezza `CONTENT_LENGTH`.

stdout

L'output dell'applicazione viene o meno interpretato dal server web. Se il nome dell'applicazione inizia per `nph-` allora l'output viene inviato direttamente al client. In questo caso è compito dell'applicazione CGI inviare tutti gli header in maniera corretta. In tutti gli altri casi è il server web che invia lo header di risposta e tutti gli altri campi dello header ritornato dall'applicazione eccetto per i seguenti campi:

1) Content-type:

E' il tipo MIME del documento ritornato dall'applicazione.

2) Location:

Indica al server web che la risposta dell'applicazione CGI indica dove trovare il documento vero e proprio invece che ritornarlo direttamente. In questo caso ci sono due possibilità:

1. l'argomento del campo Location è un URL, quindi il client ridireziona il client al nuovo indirizzo;
2. l'argomento del campo Location è un path, quindi il server ritorna il documento come se il client lo avessi richiesto direttamente.

3) Status:

Viene usato per indicare al server il tipo di risposta da inviare al client. Il formato dell'argomento è:
`Status: nnn XXXXX`
dove nnn è un numero di 3 cifre con il codice di stato e XXXXX è la stringa che descrive lo stato, ad esempio
`Status: 404 NOT FOUND`
viene trasformato dal server in
`HTTP/1.0 404 NOT FOUND`

Esempio accesso Environment

Un esempio di accesso alle variabili d'ambiente da uno script di shell è lo script seguente:

```
#!/bin/sh
echo Content-type: text/plain
echo
echo CGI/1.0 test script report:
echo
echo argc is $#. argv is "$*".
echo
echo SERVER_SOFTWARE = $SERVER_SOFTWARE
echo SERVER_NAME = $SERVER_NAME
echo GATEWAY_INTERFACE = $GATEWAY_INTERFACE
echo SERVER_PROTOCOL = $SERVER_PROTOCOL
echo SERVER_PORT = $SERVER_PORT
echo REQUEST_METHOD = $REQUEST_METHOD
echo HTTP_ACCEPT = "$HTTP_ACCEPT"
echo PATH_INFO = $PATH_INFO
echo PATH_TRANSLATED = $PATH_TRANSLATED
echo SCRIPT_NAME = $SCRIPT_NAME
echo QUERY_STRING = $QUERY_STRING
echo REMOTE_HOST = $REMOTE_HOST
echo REMOTE_ADDR = $REMOTE_ADDR
echo REMOTE_USER = $REMOTE_USER
echo CONTENT_TYPE = $CONTENT_TYPE
echo CONTENT_LENGTH = $CONTENT_LENGTH
```

Un esempio di accesso allea variabile d'ambiente `PORT` da un programma in C è il programma:

```
printf("La porta:%s\n",getenv("SERVER_PORT"));
```

Esempio Form HTML con metodo GET

Un esempio di una pagina html per creare un form (supponiamo che la pagina sia visibile con l'URL: `http://www.dsi.unive.it/~taw/Lezione3.html`):

```
<html>
<body>
<FORM METHOD="GET" ACTION="testform.cgi" ENCTYPE="application/x-www-form-urlencoded">
Nome:
<INPUT TYPE="text" NAME="nome" VALUE="">
<BR>
Cognome:
<INPUT TYPE="text" NAME="cognome" VALUE="">
<BR>
<TEXTAREA NAME="linee" ROWS="5">
Linea1
Linea2
Linea3
</TEXTAREA>
```

```
<BR>
<INPUT TYPE="submit" VALUE="ok">
</FORM>
</body>
</html>
```

Il browser visualizza la pagine predente nel seguente modo:

Nome:

Cognome:

```
 Linea1 Linea2
Linea3
```

ok

Se in corrispondenza dell'URL `http://www.dsi.unive.it/~taw/testform.cgi` troviamo uno script CGI col seguente codice:

```
#!/bin/sh
echo Content-type: text/plain
echo
echo argc is $#. argv is "$*".
echo
echo REQUEST_METHOD = $REQUEST_METHOD
echo PATH_INFO = $PATH_INFO
echo PATH_TRANSLATED = $PATH_TRANSLATED
echo SCRIPT_NAME = $SCRIPT_NAME
echo QUERY_STRING = $QUERY_STRING
echo CONTENT_TYPE = $CONTENT_TYPE
echo CONTENT_LENGTH = $CONTENT_LENGTH
cat
```

premendo il pulsante l'output che otterremo nella finestre del browser sarà:

```
argc is 0. argv is .
REQUEST_METHOD = GET
PATH_INFO =
PATH_TRANSLATED =
SCRIPT_NAME = testform.cgi
QUERY_STRING = nome=&cognome=&linee=Linea1%0D%0ALinea2%0D%0ALinea3%0D%0A
CONTENT_TYPE =
CONTENT_LENGTH =
```

Nel browser l'URL visualizzato sarà:

`http://www.dsi.unive.it/~taw/lezione3/testform.cgi?nome=&cognome=&linee=Linea1%0D%0ALinea2%0D%0ALinea3%0D%0A`

Come si può vedere tutti i campi del form vengono codificati e aggiunti all'url con il quale viene invocata l'applicazione CGI. Dato che ci sono dei limiti nella lunghezza degli url (alcuni browser considerano al massimo url di 255 caratteri), questo metodo non può essere applicato sempre.

Notiamo inoltre che i valori dei parametri vengono codificati con la codifica url-encoded che trasforma gli spazi in '+' e tutti i caratteri che non sono lettere o numeri in codifica esadecimale preceduta dal simbolo '%'. I vari parametri sono separati dal simbolo '&'.

Esempio Form con metodo POST con codifica url-encoded

Il metodo GET va usato quando ci sono operazioni di lettura e ricerca. Non andrebbe mai usato quando ci sono operazioni che comportano una modifica alla base dati in senso lato. Inoltre possiamo usare il metodo POST:

- per trasferire molti dati dal client al server;
- per non far vedere all'utente del browser che dati vengono inviati.

Il codice HTML per usare il metodo POST è il seguente:

```
<html>
<body>
<form method="POST" action="testform" enctype="application/x-www-form-urlencoded">
NOME:
<input type="text" name="nome" value="">
<BR>
COGNOME:
<input type="text" name="cognome" value="">
<BR>
<textarea name="linee" rows=5>
Linea1
Linea2
Linea3
</textarea>
<BR>
<input type="submit" value="ok">
</form>
</body>
</html>
```

La visualizzazione del form da parte del browser **NON** cambia rispetto al metodo GET. Rispetto al metodo get il primo cambiamento riguarda l'URL visualizzato che sarà:
`http://www.dsi.unive.it/~taw/lezione3/testform.cgi`
Mentre la pagina di risposta sarà:

```
argc is 0. argv is .
REQUEST_METHOD = POST
PATH_INFO =
PATH_TRANSLATED =
SCRIPT_NAME = testform.cgi
QUERY_STRING =
CONTENT_TYPE = application/x-www-form-urlencoded
CONTENT_LENGTH = 57
nome=&cognome=&linee=Linea1%0D%0ALinea2%0D%0ALinea3%0D%0A
```

Come si può vedere tutti i campi del form vengono codificati e passati nello standard input dell'applicazione CGI. In questo caso non abbiamo il problema della lunghezza dell'url perché i parametri codificati vengono passati sullo standard input. In ogni caso la codifica dei parametri richiede del tempo di codifica al browser e un'eventuale decodifica da parte dell'applicazione CGI. Queste operazioni di codifice e decodifica posso essere onerose se i dati trasferiti sono molti come nel caso di trasferimento di file.

Esempio Form con metodo POST e codifica multipart

Nel caso di sottomissione di file, il metodo migliore è usare la codifica multipart. In questo caso i dati non vengono codificati, ma i singoli parametri compaiono nello standard input dell'applicazione separati da un'opportuno delimitatore.

```
<html>
<body>
<form method="POST" action="testform.cgi" enctype="multipart/form-data">
NOME:
<input type="text" name="nome" value="">
<br>
COGNOME:
<input type="text" name="cognome" value="">
<br>
File:
<input type="file" name="filename">
<br>
Ultimo campo:
<input type="text" name="end">
<br>
<input type="submit" value="ok">
</form>
</body>
</html>
```

Che produrrà il browser ci visualizzerà nel seguente modo:

NOME:
COGNOME:
File:
Ultimo campo:
ok

Ecco il risultato dell'invocazione dello script CGI come viene visualizzato dal browser:

```
argc is 0. argv is .
REQUEST_METHOD = POST
PATH_INFO =
PATH_TRANSLATED =
SCRIPT_NAME = testform.cgi
QUERY_STRING =
CONTENT_TYPE = multipart/form-data; boundary=---------------------------
24464570528145
CONTENT_LENGTH = 2921
-----------------------------24464570528145
Content-Disposition: form-data; name="nome"
alessandro
-----------------------------24464570528145
Content-Disposition: form-data; name="cognome"
roncato
-----------------------------24464570528145
Content-Disposition: form-data; name="filename"; filename="Gruppi200203.html"
Content-Type: text/html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <title>Laboratorio Ingegneria del Software: Composizione gruppi 2002/03</title>
  <meta http-equiv="content-type"
 content="text/html; charset=ISO-8859-1">
</head>
<body>
<h1>Laboratorio di Ingegneria del Software</h1>
<h2>Composizione gruppi 2002/03</h2>
Gruppo 1: Progetto 1<br>
```

<u>Antonello Mauro</u> 784113
Franchetto Stefano 783094
De Nes Francesco 784105
Galesso Daniele 786466

Gruppo 2: Progetto 2
<u>Rota Bulo' Samuel</u> srotabul@dsi.unive.it
Casagrande Massimo mcasagra@dsi.unive.it
Fornaro Emanuele efornaro@dsi.unive.it
Gerarduzzi Michele mgerardu@dsi.unive.it
Luisetto Andrea aluisett@dsi.unive.it
Niero Luca lniero@dsi.unive.it

Gruppo 3: Progetto 3
<u>Trolese Paolo</u>
Orsenigo Marco
Favaro Susanna
Busolin Katia

Gruppo 4: Progetto 2
<u>Pietro Ferrara</u> 784578
Teresa Scantamburlo 783915
Patrizia Zucconi 783916
Luigi Runfola 784386

Gruppo 5: Progetto 1
<u>Casotto Briano</u> bcasotto@dsi.unive.it 785251
Ravagnan Emiliano eravagna@dsi.unive.it 786353
Marvulli Donatella dmarvull@dsi.unive.it 783738
Ramelintsoa Carole cramelin@dsi.unive.it 786413
Scavazzon Marco mscavazz@dsi.unive.it 786242

Gruppo 6: Progetto 3
Fiori Alessandro afiori@libero.it 777191
Gastaldi Riccardo rigasta@tin.it
Bernacchia Francesco s.checco@libero.it 778611
Pozzobon Giovanni GiovanniP@aton.it 791120
Rampazzo Pietro rampaz79@tin.it
Cian Francesco francesco.cian@t-systems.it 778885

Gruppo 7: Progetto 4
<u>Piccin Massimiliano</u>
Carraretto Alessandro
Ministeri Dario
Rui Massimo

Gruppo 8: Progetto 4
<u>Corradin Michele</u>
 Borin Francesca
 Bordin Fabio
 Scomello Stefano

Gruppo 9: Progetto 5
<u>Carrer Andrea</u> 783089
Longo Irene 783528
Vianello Michele 784026
Molinari Marco 784162

Gruppo 10: Progetto 5
<u>Leonardo Scattola</u> lscattol@dsi.unive.it
Filippo Cervellin fcervell@dsi.unive.it
Roberto Fietta rfietta@dsi.unive.it
Luca Giacomazzi lgiacoma@dsi.unive.it
Lino Possamai lpossmai@dsi.unive.it

</body>

```
</html>
-----------------------------24464570528145
Content-Disposition: form-data; name="end"
valore ultimo campo
-----------------------------24464570528145--
```

Nota: se usiamo il tag file con la codifica url-encoded viene inviato solo il nome del file e non il contenuto.

Esercizio:

Negli esempi soprastanti c'è un errore, eseguire gli esempi di sopra e trovare l'errore.