OCCAM'S RAZOR

ISSN: 1998-0537

Porque lo más sencillo es lo más probable

Número 3 · 1ª Edición · 2008

VIM. Pequeño pero Matón
Todos los secretos de esta mala bestia

librerías
TCCLIB: Usando C como lenguaje de Script

tutorial
Diseña tus propias simulaciones Acústicas

en la práctica
Te contamos cómo se localiza el Software

historia
El telégrafo de Gauss

DIVULGACIÓN
Cómo aprenden las máquinas?

tecnología
Reconocimiento Biométrico a través del IRIS

... y nuestras secciones de siempre
- Trucos
- El rincón de los lectores

NO TE CORTES CON LA NAVAJA DE OCCAM

SUMARIO

Esta revista ha sido realizada con:

blender · GIMP · Emacs · Vim · LaTeX

Occam's Razor

Número 3, Año 2008

Dirección:
David Martínez Oliveira

Editores:
David Martínez Oliveira
Fernando Martín Rodríguez

Colaboradores:
Fernando Martín Rodríguez, Oscar Martínez Mozos, Silvia Carril Caldelas, Gonzalo Barrio, Francisco Miguel Bellas Aláez, Gavin Mathews, Laura Rodríguez González, Er Interprete, Er Escribano, Er de la Secció, Er del Aberno

Maquetación y Grafismo
DeMO LiR

Publicidad
Occam's Razor Direct
occams-razor@uvigo.es

Impresión
Por ahora tu mismo... Si te apetece

Editorial

Primer Aniversario

by The Occam Team

Primer Aniversario!!!. Bueno este es el número 3, pero eso no quita que ya haya pasado un año desde que empezamos este proyecto, con el primer número de *Occam's Razor*. Por si alguien no se enteró el genial Donald Knuth creador de TEXcumplió 70 años el pasado 11 de Enero, y desde aquí queremos dedicarle este número de Occam's Razor, el cual no existiría sin su genialidad (o al menos sería bastante diferente). Feliz Cumpleaños!

A modo de celebración de nuestro primer aniversario, hemos intentado que este número fuera más variado y un poco especial. Al final hemos terminado con más de 50 páginas que esperamos que disfrutéis.

La otra novedad que trae este número, y que anunciamos en un conocido foro tras el lanzamiento del número 2 de la revista, es el cambio de su licencia de distribución para hacer que *Occam's Razor* sea un poco más libre.

En otro orden de cosas, en este número 3 de *Occam's Razor* os encontraréis las secciones que por mérito propio se están convirtiendo en fijas: *Ratas de Biblioteca, Mala Bestia, Mú Rápido y el Reverso Tenebroso*. Ellas son las encargadas de mantener el contenido técnico relacionado directamente con el mundo de la tecnología informática.

Como decíamos más arriba este número es más variado que los anteriores gracias a nuevas colaboraciones externas que nos acercan a la tecnología desde otros puntos de vista y campos profesiones. Así Óscar Martínez nos proporciona una excelente introducción al mundo de la *Inteligencia Artificial* y Silvia Carril nos ayuda, con su artículo *Trujamana o Localizadora*, a comprender mejor una profesión con la que estamos en contacto cada día pero que sigue siendo una gran desconocida.

En esta misma línea de aplicación directa de la tecnología, Francisco Bellas nos cuenta cómo utilizar el programa *Odeon* para que aprendamos un poco de *Acústica Arquitectónica*.

Fernando Martín nos ofrece, en este número, dos excelentes artículos. Por una parte, Fernando continúa explicándonos como funcionan los sistemas de reconocimiento biométrico abordando en esta ocasión el análisis del iris. Por otra, Fernando en colaboración con Gonzalo Barrio, nos ofrece un interesantísimo y original artículo de investigación histórica sobre cómo pudo funcionar el *Telégrafo de Gauss*.

Como podéis ver seguimos evolucionando hacia la revista que queremos llegar a ser. Cada vez nos vamos acercando más a los contenidos Tecnológicos y Científicos (estos últimos nos están costando) que estamos buscando, siempre en torno a las ciencias de la computación que, cada vez más, se están convirtiendo en el nexo de unión de este mundo multidisciplinar.

Como siempre deseamos que este joven proyecto, que por ahora solo cuenta un año, os resulte interesante. Muchas gracias por vuestro apoyo!!!

The Occam's Razor Team

El Rincón de los lectores

Vuestros comentarios, sugerencias,...

por The Occam's Razor Team

SSH

enviado por Luis Rodríguez

Simplemente señalar que los usuarios de Windows pueden descargarse en el sitio anónimo de ftp del laboratorio de mi escuela (`ftp://ftp.lab.fi.uva.es/pub/ssh/Windows`) el software cliente Open SSH, que permite acceder mediante ssh a una máquina remota.
Un saludo y enhorabuena por la revista.

MÁS SSH

enviado por Cruz Enrique Borges Hernández

Muy buen artículo, pero creo q se os ha olvidado mencionar la kioslave fish que te permite acceder desde cualquier aplicación kde a fichero remotos a través de ssh. Es extremadamente útil ;)

–

Muchas gracias por ambos apuntes, aquí los ponemos para el resto de lectores.

ERROR EN ARTÍCULO SSH

enviado por Gerardo Elian Gidoni

Gerardo nos apuntó un error en el artículo SSH que seguro que muchos habéis detectado, y nos proporciona una detallada explicación que reproducimos a continuación... aún no sabemos como ha podido pasar :P.
Muchas gracias Gerardo por esta contribución.

–

Buenas,
Antes que nada recién acabo de conocer la revista y me han gustado mucho los artículos que leí.
Interesante proyecto, felicitaciones :-)
Por otro lado creo haber encontrado un error en el artículo sobre túneles SSH donde se habla de redirección de puertos remotos. (paginas 8 y 9).
Pego acá el fragmento.

–

Podemos utilizar el flag -R para conseguir un resultado similar. La sintaxis es idéntica, pero en este caso el puerto de redirección se abrirá en la máquina remota. Como podéis imaginar, nuestra máquina "proyecto-x" puede ser cualquier servicio interno de la red remota: servidores web, pop, etc... Por ejemplo, si la máquina "proyecto-x" ofrece un interfaz web, una vez establecido el túnel, solo tenemos que apuntar nuestro navegador a la url: "http://entrada/1234" si la redirección de puertos es remota, o a "http://localhost:1234" si optamos por la redirección local.

```
occam@razor$ ssh -R1234:proyecto-x:5000 entrada
Password:
occam@entrada$
....
```

[En otro terminal]

```
occam@razor$ nc entrada 1234
Bienvenido al Proyecto X
}
```

Creo que se ha confundido la idea de "-R". El parámetro "-R" sirve para hacer lo opuesto a "-L". Una situación acorde a la utilidad de "-R" sería:
Necesitamos acceder al puerto "5000" de "proyecto-x" por medio de "entrada" pero "entrada" no acepta conexiones desde el exterior, por lo que desde "razor" no podríamos iniciar la conexión. Entonces la idea sería que DESDE "entrada" iniciemos la conexion HACIA "razor", entonces haríamos

```
usuario@entrada$ ssh -R1234:proyecto-x:5000 razor
```

Esto nos permitiría acceder ahora a "proyecto-x" iniciando una conexion al puerto "1234" de "razor". Entonces ahora si haríamos:

```
occam@razor$ nc localhost 1234
Bienvenido al Proyecto X
```

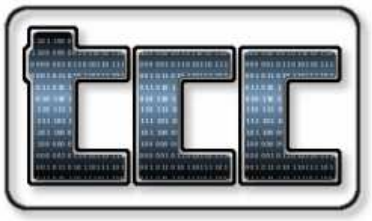
Además donde dice:
Por ejemplo, si la máquina "proyecto-x" ofrece un interfaz web, una vez establecido el túnel, solo tenemos que apuntar nuestro navegador a la url: `http://entrada/1234` si la redirección de puertos es remota
Esto ultimo tampoco seria cierto.
Espero no haberme confundido y que les sean útiles mis comentarios.
saludos!
- http://gerelblog.blogspot.com

ENVIADNOS...

Vuestros comentarios, sugerencias, ideas, críticas (constructivas claro), correcciones, soluciones, disoluciones o cualquier cosa que se os ocurra... a:

`occams-razor@uvigo.es`

LOS ESPERAMOS!!!!



TCClib

Utiliza C como tu lenguaje de Script

por Er Interprete

Pensando en incorporar un lenguaje de script a tu aplicación?. Python, Java, Ruby, Guile?... eso es para nenazas. Los programadores de verdad usan C incluso para sus scripts.

Bromas a parte, en este número vamos a hablar de tcclib, una librería que se distribuye junto a TCC (el Tiny C Compiler o Compilador C Pequeñito).

TCC es un compilador de C desarrollado por *Fabrice Bellard*. Es muy pequeño y rápido. Como os podéis imaginar no es gcc, pero para ciertas aplicaciones puede resultar muy útil.

Lo que tcclib nos permite, es compilar código C en nuestras aplicaciones de una forma muy fácil y sencilla. Ahí es nada!. Como siempre vamos directos al código.

COMPILANDO Y EJECUTANDO

Aquí tenéis el código de un pequeño programa que utiliza tcclib para leer un fichero de texto, compilarlo y ejecutarlo. Este ejemplo es una modificación del programa de prueba incluido con las fuentes de tcc, básicamente para que ocupe poco y nos quepa en esta sección :). Como siempre recordad que las comprobaciones de error han sido eliminadas.

```
#include <stdio.h>
#include <string.h>
#include <libtcc.h>

/* Aquí iría el API que ofrecemos
   a los scripts */
void mi_func (char *str) {
  printf ("PRINCIPAL(mu_funcion):%s", str);
}

int
main (int argc, char *argv[])
{
  TCCState      *s;
  FILE          *f;
  char           b[1024];
  int            (*un_script)(char *);
  unsigned long  v;

  /* Lee script */
  memset (b, 0, 1024);
  f = fopen (argv[1], "rt");
  fread (b, 1024, 1, f); fclose (f);

  /* Configuramos y compilamos */
  s = tcc_new ();
  printf ("Setting compiler %p\n", s);
  tcc_set_output_type (s, TCC_OUTPUT_MEMORY);
  tcc_compile_string (s, b);

  /* Añadimos nuestro especial API */
  tcc_add_symbol (s, "mi_func",
                  (unsigned long)&mi_func);
  tcc_relocate (s);

  /* Ejecutamos */
  tcc_get_symbol (s, &v, "mi_script");
  mi_script = (void*)v;
  mi_script ("Hola!!!");
  tcc_delete (s);
}
```

Sí, vale, esto parece un poco más complicado que lo que solemos incluir en esta sección, pero enseguida veremos que es mucho más sencillo de lo que parece.

TCC y tcclib compilan código C realmente rápido

Lo que este programa hace es cargar un fichero de texto que contendrá código C. La única condición que debe cumplir es definir la función "mi_script", que recibirá como parámetro una cadena de caracteres.

Por otra parte, el programa exporta una función, llamada mi_func que puede ser utilizada por nuestros scripts, para, por ejemplo, comunicarse con la aplicación principal... seguro que se os ocurren mil ideas.

LAS LLAMADAS A TCCLIB

Como podéis observar, utilizamos una variable del tipo TCCState para referirnos a lo que podríamos llamar nuestra instancia del compilador. Una vez creada (con tcc_new), procedemos a configurarla. En nuestro caso simplemente indicamos que queremos compilar en memoria (tcc_set_output_type).

tcclib es capaz de generar ejecutables, librerías dinámicas o código objeto, además de la compilación en memoria que a nosotros nos interesa.

Tras esto, utilizamos tcc_compile_string. Esta función, como os podréis imaginar, transforma una cadena de caracteres que contiene código C en código ejecutable... es decir, compila la cadena, como su propio nombre indica :).

Ahora que tenemos nuestro código compilado, nos interesan dos cosas. La primera es que nuestro script pueda llamar a funciones definidas por nuestro programa y por tanto se pueda utilizar como un lenguaje de script. La segunda es poder referenciar las funciones definidas por nuestro script y que hemos compilado en memoria, para, por tanto, ejecutar el código que contiene.
Lo primero se consigue con la llamada a la función tcc_add_symbol. Esta función recibe como parámetros el nombre de la función que se desea añadir (realmente un símbolo) al sistema, y un puntero al código de dicha función.
Para lo segundo (ejecutar el código que proporciona el script) utilizamos la función tcc_get_symbol, que realiza la operación complementaria a tcc_add_symbol. En el programa es necesario utilizar una variable auxiliar debido a como está definida esta función.
Una cosa importante. Antes de poder ejecutar la función tcc_get_symbol, es necesario ejecutar tcc_relocate. No vamos a entrar en explicaciones, pero esta función lleva a cabo el proceso de "relocación" que normalmente realiza el cargador de programas o el linker dinámico.

tcclib nos permite compilar, linkar y ejecutar de forma sencilla

Tened en cuenta que tcclib nos permite llevar a cabo todo un proceso que normalmente requiere de varias herramientas: compilador, linker, cargador.

EL SCRIPT

Ahora que ya sabemos como funciona nuestro programa, vamos a escribir nuestro primer script y ver si todo esto funciona.

```
int mi_script (char *str)
{
    printf ("Hola soy un script. "
            "Mi parámetro es: '%s'\n");
    mi_func ("Script llamando a tierra!!");
    return 0;
}
```

¿Simple no?, pero con bastante información. Lo primero que vemos es que se trata de código C estándar. Lo segundo es que estamos utilizando la función printf sin más. Esto es así porque nuestro programa principal incluye la librería C estándar. Para utilizar otras librerías, echadle un ojo a la función tcc_add_file.
Lo tercero es que estamos ejecutando la función mi_func que nuestro programa principal exporta. ¡Güay!. Basta de rollo, vamos a compilar y comprobar el resultado de una ejecución.

```
occam@razor $  gcc -o test test.c \
> /usr/local/lib/libtcc.a -ldl
occam@razor $ ./test script.c
Hola soy un script. Mi parámetro es: 'Hola!!!'
PRINCIPAL:Script llamando a tierra!!
```

Como podéis ver, libtcc es una librería estática y normalmente la tendréis instalada en /usr/local/lib. La librería dl (dynamic linking) también es necesaria. Si no la utilizáis obtendréis un error referente a funciones como dlopen o dlsym.

Usando TCC podéis utilizar el lenguaje C como lenguaje de script ejecutando código nativo!!

Lo segundo es que si habéis copiado los ejemplos al pié de la letra, veréis que vuestro programa no produce la salida esperada... veréis algunos "marcianitos". El script tiene un error. Es muy tonto, así que seguro que ya lo habéis visto. Lo interesante de esto es que solo tenéis que abrir el script con vuestro procesador de textos favorito. modificarlo y ya está... no hay que compilar nada, como si estuviéramos trabajando en Perl, Python, TCL o similares :).

COMENTARIOS FINALES

Al principio de este texto oa comentábamos que tcc es un compilador muy rápido y pequeño. Esto es algo muy interesante, pero tiene como principal contrapartida que el código generado, no está tan optimizado como el que pueda generar gcc. Este es el precio a pagar por la velocidad y la versatilidad de disponer de un compilador como parte de nuestros porgramas. Con esto queremos decir que, cada cosa es para lo que es, y no existen las panaceas :).
Por otra parte, en el momento de escribir este artículo, fué necesario obtener tcc de su CVS de desarrollo en savanah. El paquete en la web, no compila en los sistemas modernos y las distribuciones, al menos Ubuntu, solo incluyen el compilador tcc, y no la librería tcclib. El proceso de instalación es el habitual configure, make y make install.
La versión del CVS parece que también funciona en Windows, pero eso no lo hemos probado, así que si alguno se anima a probarlo y confirmarlo, esperamos sus comentarios.
Por último, y como viene siendo habitual, recomendaros que le echéis un ojo a libtcc.h. Veréis que hay varias funciones que aquí no hemos comentado pero que os resultarán muy útiles en vuestros programas. También os recomendamos una visita a la página del señor Fabrice Bellard (http://fabrice.bellard.free.fr/), donde encontraréis cosas realmente interesantes... A alguien le suena qemu?
Hasta el próximo número.

vim. Pequeño pero Matón

O lo quieres o lo odias

por Er Escribano

Sí compañeros, vim, la versión mejorada de vi es una mala bestia, ahí donde lo veis. En este artículo vamos a profundizar en las características menos conocidas y más potentes de este omnipresente editor de textos, que no son pocas.

Muchos os preguntaréis: ¿Por qué debería aprender a utilizar vim cuando tengo unos editores tan chulos que me hacen la vida tan sencilla y que puedo empezar a manejar con mi ratón sin conocer ningún oscuro conjunto de comandos?. A esta pregunta podríamos responder de tres formas.
La primera es que menudo pedazo de pulmones que tenéis para soltar del tirón semejante pregunta :).
La segunda es que uséis el sistema UNIX que uséis, lo más probable es que el editor que os encontréis seguro sea vi. Da igual que se trate de un Linux, un Solaris o un HP-UX, vi estará allí. Sí vale, vi no es vim, pero el uso básico de ambos editores es el mismo.
La tercera es que vim es una mala bestia. En seguida veréis la potencia de este pequeño editor de textos que, os enamorará o conseguirá que terminéis odiándolo completamente :).

LO BÁSICO

Por si algún despistado no ha utilizado nunca vim, vamos a incluir esta pequeña sección con lo básico para, simplemente, poder usarlo. Lo primero que debe saber el no iniciado sobre vim es que tiene tres modos de funcionamiento. A saber:
El modo edición/inserción. Este es el modo que cualquier usuario espera en un editor de textos... pulsas teclas y el texto comienza a aparecer en el documento.

Vim es mucho más fácil de utilizar de lo que parece

El modo comando. En este modo, cuando un usuario pulsa una tecla le indicará a vim que realice una determinada acción. Podéis pensar en ello como en "Hot-keys" para acceder a un menú, pero sin menú.
Finalmente, vim proporciona un modo de linea de comando en el que podemos pedirle que haga cosas más complejas que las que nos permite el modo comando.
Por defecto, vim arranca en el modo comando. Para poder escribir, necesitamos cambiar a modo edición/inserción. Esto se puede hacer con distintos comandos, es decir, pulsando alguna de estas teclas:

i	Inicia inserción en la posición del cursor
a	Inicia inserción después de la posición del cursor
A	Inicia inserción al final de la línea actual
o	Inicia inserción en una nueva línea, debajo de la actual
O	Inicia inserción en una nueva línea, encima de la actual

Para volver al modo comando, basta con pulsar la tecla ESC. Aunque existen multitud de comandos, los que probablemente mas utilicéis (y con los que se puede escribir cualquier texto) son los siguientes:

0	Mueve el cursor al principio de la línea
$	Mueve el cursor al final de la línea
dd	Borra/Corta la línea actual
D	Borra los caracteres desde la posición del cursor hasta el final de la línea
J	Junta la línea inferior y la línea actual
x	Borra el carácter sobre el que se encuentra el cursor
/cadena	Busca "cadena" en el texto hacia delante
?cadena	Busca "cadena" en el texto hacia atrás
n	Busca la siguiente ocurrencia de una búsqueda iniciada con / o ?
N	Busca la ocurrencia anterior.
yy	Copia la línea actual
p	Pega la/s líneas copiadas/cortadas debajo de la línea actual
P	Pega la/s líneas copiadas encima de la línea actual

Finalmente, para iniciar el modo de línea de comando, solo tenemos que pulsar el carácter ':' cuando nos encontremos en modo comando. En la parte inferior del texto aparecerán los ':' y a continuación podremos escribir nuestro comando. En esta sección para no iniciados, los comandos mínimos que tenéis que conocer son estos:

:w nombre	Graba los cambios en un fichero llamado nombre
:w	Graba los cambios
:q	Sale
:q!	Fuerza salida aunque existan cambios sin grabar.
:wq	Graba los cambios y sale

Bueno, hasta aquí lo más básico para poder trabajar con vim. Si os fijáis, aprendiendo estos pocos comandos podréis utilizar vim para cualquier tarea básica. Y como podéis comprobar no es tan terrible... de hecho, enseguida veréis que es una pasada.

ALGUNAS GENERALIDADES

Si os habéis fijado en los comandos que forman nuestra guía básica de supervivencia con vim, todos ellos guardan una cierta simetría. Por ejemplo, la diferencia entre mayúsculas y minúsculas es en general muy evidente (comandos 'o' o 'n'), la letra elegida para el comando suele estar relacionada con lo que hace (i insertar, a añadir, p pegar, n siguiente -next/,...), pero además, existen una serie de características generales en el modo comando.
La más importante es la capacidad de repetir un cierto comando tantas veces como deseemos, simplemente escribiendo un numero antes de ejecutar el comando. Por supuesto, esto hay que realizarlo en modo comando. Veamos algunos ejemplos:

Podemos repetir cualquier comando precediéndolo de un número

Si pulsamos 10 seguido del comando x, ejecutaremos el comando borrar caracteres 10 veces, con lo que borraremos 10 caracteres. Si queremos borrar las dos líneas justo debajo de nuestra posición pulsaremos "2dd". Y si queremos borrar las cuatro líneas de arriba pulsaremos "4DD"... No, si pulsáis 38FF no saldrá una maciza pechugona.
Este último comando "dd", no solamente borra líneas, sino que además las copia, de forma que se pueden pegar posteriormente. Es el equivalente al "Cortar" de los clásicos menús de edición de la mayoría de aplicaciones gráficas.
En general, cortar líneas para borrar trozos de nuestro texto o copiarlos es bastante tedioso, así que vim proporciona un modo de selección visual:

SHIFT+V	Activa el modo de selección visual para líneas. En este modo solo podremos seleccionar líneas completas.
CTRL+V	Idem pero para bloques... éste tenéis que probarlo. Pocos editores son capaces de hacer esto :).

Una vez que tengáis seleccionado vuestro bloque de texto (se mostrará en vídeo inverso), solo tenéis que ejecutar el comando que queráis sobre él. Así, "dd" cortará el bloque seleccionado o "yy" lo copiará, de forma que podremos "pegarlo" posteriormente con el comando "p" en otra parte del documento.
Enseguida veremos que esta forma de selección se puede utilizar directamente en el modo de línea de comandos.
Por último comentar que tenemos una tercera opción para manejar bloques dentro de nuestro documento. Son las marcadores o "bookmarks".
Para poner una marca en una línea solamente tenemos que pulsar la tecla m, seguida de un carácter. Este carácter sera el identificador de nuestra marca. Así la secuencia de teclas "ma", además de un homenaje tácito a la legendaria serie de televisión "El Equipo A", asignará el marcador "a" a la línea actual.
Pulsando la tecla "'" seguida del nombre del marcador, nos moveremos a la posición asociada al mismo. Como veremos muy pronto, podremos utilizar marcadores para referenciar bloques de texto, en comandos más complejos.

LINEA DE COMANDOS

Bien, vamos a empezar con las cosas interesantes. La mayoría de lo que comentaremos a continuación tiene que ver con el modo de línea de comandos de vim, que si recordáis se activa pulsando la tecla ":" cuando estamos en el modo comando.
Quizás el comando mas útil, o al menos el que yo más uso es el comando "s". Los que hayáis trabajado con sed o Perl lo conoceréis bien. Este comando permite localizar un patrón en el texto (utilizando expresiones regulares) y sustituirlo por otra cosa. Su sintaxis es esta:

```
s/origen/destino/flags
```

El flag que más nos interesará es el "g", que realiza la operación sobre todas las ocurrencias de "origen" en una determinada línea. Si no utilizamos este flag la sustitución se realizará, únicamente, sobre la primera ocurrencia.

El modo línea de comando es el que nos ofrece las posibilidades más interesantes

Veamos un ejemplo. Supongamos que estamos haciendo nuestra página web y decidimos que queremos meter las imágenes en un directorio en lugar de dejarlas en el directorio base con las páginas. Abrimos nuestra pagina en vim y ejecutamos:

```
:%s/src="\(.*\)"/src="images\/\1"/
```

El carácter "%" al principio de la línea indica que el comando se ejecutará sobre todo el fichero, de lo contrario, solo se ejecutaría sobre la línea actual. En lugar del "%" podemos proporcionar rangos específicos:

```
:1,5s/pepe/Manolo/
:1,$s/Gavilan/Paloma/
:10,$s/foo/bar/
:'a,'bs/AAAA/BBB/
```

Si os decimos que el carácter "$" representa el final del fichero y que 'a y 'b son marcas de las que hablamos antes.... estas expresiones no tienen ningún secreto verdad?
Si recordáis, algunas líneas antes hablábamos del modo de selección visual a los que accedíamos con las combinaciones de teclas SHIFT + V o CTRL + V. Si seleccionáis un bloque de texto de esta forma y entráis en modo linea de comandos (pulsando :) vim automáticamente seleccionará ese rango para la ejecución del comando que deseemos. Lo que veréis es algo como esto:

```
:'<,'>
```

Si, como os podéis imaginar <y >son dos marcas que vim genera automáticamente en el modo de selección visual.

El comando s/// da otra dimensión al concepto de buscar y sustituir

Por si alguno no está familiarizado con las expresiones regulares, vamos a hacer dos comentarios sobre el ejemplo anterior. El primero es que el uso de los paréntesis (que hay que escapar con el carácter \) permiten guardar "en memoria" el texto que cumple la expresión regular dentro de ellos. En nuestro ejemplo, nos estamos quedando con el nombre de la imagen, que debe estar entre comillas.
Para utilizar estos valores en la segunda parte del comando "s" utilizamos la expresión `\n`, donde n representa el paréntesis n-ésimo de la primera parte. Como podéis ver, las expresiones regulares pueden llegar a ser muy engorrosas debido a la necesidad de "escapar" los caracteres que tienen significado propio... pero son realmente potentes.
Por cierto, si no lo habéis probado, que sepáis que la línea de comandos de vim funciona como la de bash, es decir, con el cursor arriba podéis acceder a los comandos que acabáis de escribir (vamos la historia) y con el tabulador podéis autocompletar los comandos. Esto será más interesante un poco más adelante.

COMANDOS INTERESANTES

vim proporciona una inmensa cantidad de comandos, para que cualquiera pueda estar entretenido un buen rato leyendo. Para los que os guste leer, aquí esta el primer comando interesante:

```
:help
```

:)... Sí, podéis hartaros de leer la ayuda de vim o seguir leyendo este artículo. Comentar en este punto, que la ayuda de vim es excelente. Todo lo que comentemos en este artículo está en la ayuda on-line, pero aquí os lo presentamos más resumido y sencillo :).
Hay dos comandos especialmente útiles y potentes. El primero es ! y el segundo es r.
El comando !, permite ejecutar un comando shell, bueno, no solo eso, permite pasarle al comando shell parte del texto que estamos escribiendo y sustituirlo por la salida del programa que ejecutemos.
El comando r permite leer datos de un fichero, pero si lo que sigue es una ! y un comando shell, lo que obtendremos es la salida de ese comando.
Por ejemplo, para añadir la fecha actual a nuestro documento, solo tenemos que ejecutar:

```
:r ! date
```

O para incluir un bonito calendario en nuestro documento solo tenemos que :

```
:r ! cal 1 2008
```

Pero como os decíamos, el comando ! es realmente potente. Veamos un ejemplo de una aplicación realmente común... quedar a tomar unas birras. Alguien envía la convocatoria por correo (por ejemplo), y la gente se empieza a anotar y a proponer el día que le viene bien. El "pringao" responsable de organizar esto, es además un geek y mantiene su lista en un fichero de texto que edita con vim. El fichero sería algo como esto:

```
Pepe  14
Manolo 17
Josefina 11
Eufrasia 13
```

Si ahora queremos ordenar nuestro fichero por nombre, solo tenemos que ejecutar un comando como este.

```
:% ! sort
```

O si los quiere ordenar por la fecha que prefieran, solo tiene que ejecutar:

```
:% ! sort -k 2
```

:)... podéis ejecutar cualquier programa shell, pasarle un fragmento de texto para que lo procese y sustituir la salida del programa en el texto que estáis escribiendo...

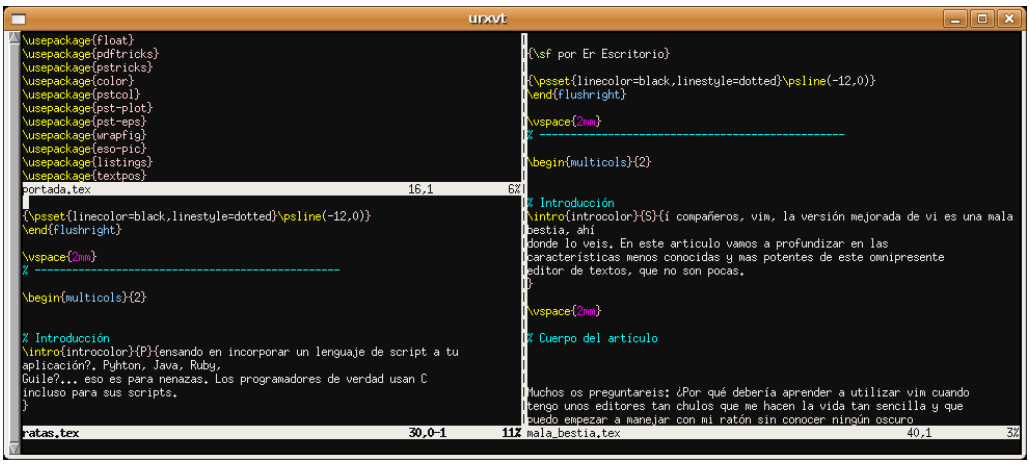
ALGO UN POCO MÁS VISUAL

Aunque en un primer momento, vim parece bastante austero visualmente hablando, como se suele decir, las apariencias engañan. Disponemos de varios comandos para dividir la pantalla y de esta forma ver varios ficheros a la vez.
Si queremos dividir la pantalla verticalmente solo tenemos que ejecutar un comando como este:

```
:vsplit otro_fichero.txt
```

Pero si lo que nos mola es hacerlo horizontalmente, no hay problema.

```
:split otro_fichero_mas.txt
```



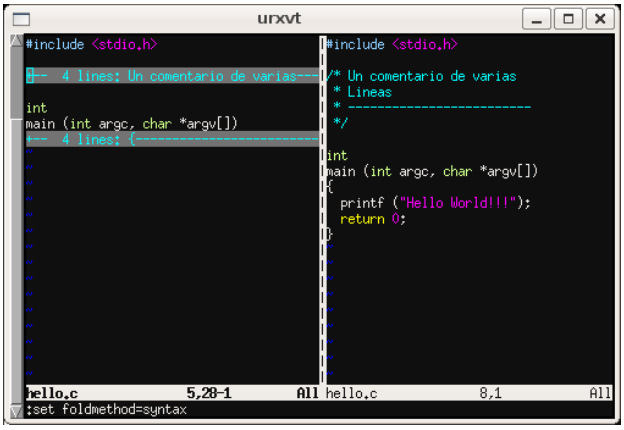
Para navegar a través de los distintos ficheros, en modo comando, pulsamos la tecla CTRL seguida de "ww".

vim nos permite colapsar y expandir bloques de texto de forma muy sencilla.

Pero eso no es todo. Como muchos habréis observado, los editores más modernos, sobre todo los orientados a programación, proporcionan una facilidad para colapsar bloques de texto de forma que podemos quitar de nuestra vista partes del texto que no nos interesan en un momento dado..., comentarios, funciones sobre las que no estamos trabajando, etc... Bien, pues esta funcionalidad también esta disponible en vim.
Para colapsar un bloque de texto, lo seleccionamos utilizando el modo de selección visual, por ejemplo, y pulsamos las teclas "zF". Para expandir un bloque de texto colapsado solo tenéis que situaros en la línea de turno y pulsar "za".
Podemos anidar estos bloques de texto e ir expandiéndolos uno a uno, o expandirlos todos de una sola vez utilizando el comando "zA"... como es lógico :). Para volver a colapsar un determinado bloque solo tenéis que volver a pulsar "zc" o "zC" si queremos aplicar la operación sobre todos los bloques recursivamente.

VIM Y LOS PROGRAMADORES

A juzgar por la cantidad de características relacionadas con la programación, muchos programadores deben utilizar vim. Vamos a ver algunas de las más importantes de estas características a continuación.
Como comentamos un poco más arriba, la capacidad de colapsar y expandir bloques de texto es muy útil cuando se escriben programas, permitiéndote disponer de más sitio para ver las cosas que realmente te interesan.
Esta funcionalidad, cuando estamos programando se vuelve mas útil ejecutando el siguiente comando:
:set foldmethod=syntax
A partir de ese momento, no necesitaremos seleccionar bloques de texto, vim los seleccionará por nosotros dependiendo del realzado de sintáxis que esté utilizando.



Existen otros métodos que pueden resultar mas útiles para otras aplicaciones.
También vimos como podemos ejecutar comandos shell desde vim, pero además, para el caso de compilación de programas, vim ofrece una serie de facilidades adicionales.
El primer comando que nos interesa es "make", que obviamente realiza la misma operación que el programa homónimo disponible en nuestros sistemas. Sin embargo, este comando nos proporciona algunas ventajas. Vamos a verlo.
Escribid un sencillo programa C, con algún error intencionado. Grabad el fichero con el nombre miprograma.c (esto se hace con el comando :w miprograma.c) y luego ejecutad lo siguiente

```
: make miprograma
```

Veremos una serie de errores, y tras pulsar una tecla volvemos a nuestro editor. Ahora escribid el siguiente comando:

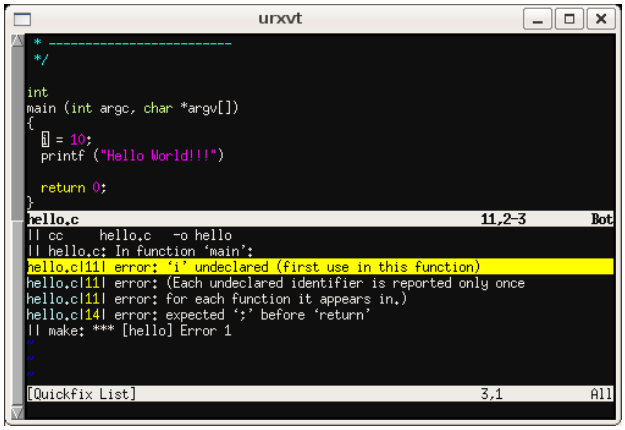
```
: copen
```

Mola!... todos los errores listados... pero lo mejor de todo es que podemos navegar por ellos, ya sea a través de la propia ventana (Quickfix List) o utilizando los comandos :cnext y :cprev.

Si, tenéis razón, es un poco pesado el escribir todo el rato esos comandos tan largos... así que vamos a mapear unas teclas al efecto.

```
: map <F7> :cn<CR>
: map <F8> :cp<CR>
```

Vaya!, no está nada mal. Como podéis ver asignar comandos a teclas o combinaciones de las mismas es bastante sencillo :). Más sobre esto dentro de un momento.
Una vez que hayáis corregido todos vuestros errores, podéis cerrar la ventana con el comando :close



ALGUNAS AYUDAS MÁS A LA PROGRAMACIÓN

Algo que siempre se necesita, especialmente los que utilizan Python. El indentado. Aunque existen multitud de opciones para controlar esta importante característica, los comandos <y >son especialmente útiles, permitiendo disminuir o aumentar el nivel de indentado de un determinado bloque de texto o linea.
Si se os ha ido de madre la edición, recordad que siempre podéis hacer un:

```
!%indent
```

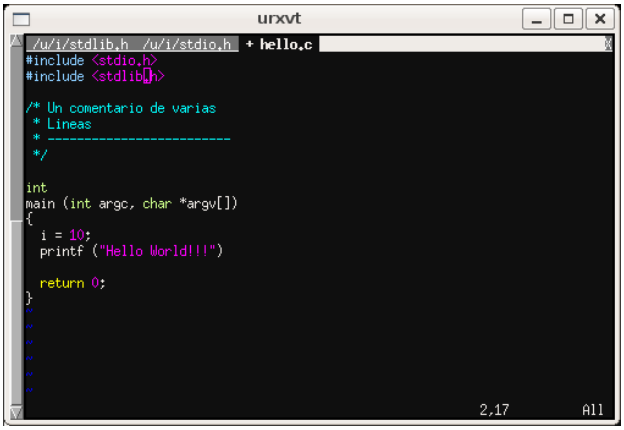
Para que alguien deje las cosas en orden por vosotros :).
La otra característica útil a la hora de escribir programas de un cierto tamaño, es la navegación a través del código. Aquí tenemos dos funcionalidades muy útiles: abrir ficheros cuyo nombre se encuentra en el texto, y los tags.

Las facilidades que vim ofrece a los programadores, hacen de él una excelente elección para esta tarea.

Vamos con la funcionalidad para abrir ficheros referenciados en nuestro texto. Un caso típico son los includes de los programas C. Para usar esta funcionalidad solo tenéis que poner el cursor sobre el nombre del fichero, en vuestro programa y pulsar las teclas “gf”.
Para que esta funcionalidad resulte útil necesitamos algo para poder navegar por los ficheros que tenemos abiertos en nuestro editor. Para ello, disponemos de dos opciones: los buffers y los tabs.
Para acceder a la lista de buffers abiertos en un determinado momento utilizamos el comando “:files” o “:buffers”. Este comando nos proporcionara una lista numerada. Podemos utilizar esos números junto con el comando :buffer para movernos a través de nuestra lista de buffers.
Como es habitual, disponemos de un montón de comandos para manipular estos buffers. Los más interesantes los comentamos a continuación:

:buffers	Lista de buffers
:buffer N o :b N	Edita el buffer N
:bn	Nos movemos al buffer siguiente
:bp	Nos movemos al buffer anterior
:sbn N	Divide la pantalla y va al buffer N

La otra opción para trabajar con varios ficheros a la vez es utilizar tabs. Los tabs, a diferencia de los buffers se muestran en la parte superior de la pantalla.



En nuestro ejemplo anterior, en el que abríamos un fichero .h, si pulsamos CTRL+W antes del comando “gf”, el fichero .h se abrirá en un nuevo tab. Podemos comprobar la lista de tabs, así como su contenido utilizando el comandos :tabs.
Y a que no os imagináis como podéis pasar de un tab a otro?... Si, claro que sí, utilizando los comandos :tabn y tabp.
El otro comando relativo a los tabs que os puede interesar es :tabe nombre-fichero. Como os imaginaréis, este comando abrirá el fichero indicado en un nuevo tab para su edición.

USANDO TAGS

Otra funcionalidad de interés para los programadores es el uso de los tags. Los tags nos permiten acceder de forma rápida a las definiciones de funciones dentro de un programa.

Los tags son generados por un programa externo llamado ctags. Podéis ejecutarlo desde la línea de comandos o directamente desde vim:

```
:!ctags *c *h
```

Los comandos mas interesantes para trabajar con tags son los siguientes:

:tag <tagname>	Localiza el tag tagname
CTRL-]	Nos mueve al tag sobre el que se encuentra el cursor
CTRL-T	Vuelve a la posición anterior
:ptag <tagname>	Localiza el tag tagname, y lo abre en una ventana de preview
CTRL-W }	Muestra el tab bajo el cursor en una ventana de preview
CTRL-W z	Cierra la ventana de preview abierta con el comando anterior

Si el fichero de tags para nuestro programa ha sido generado, podemos acceder a una determinada función al lanzar vim, utilizando la siguiente línea de comandos:

```
$ vim -t funcion
```

De esta forma, vim buscará que fichero define la función que pasamos como parámetro y lo abrirá automáticamente.

VIM PARA NO PROGRAMADORES

Hasta ahora hemos visto un montón de funcionalidades orientadas a la programación, pero vim es un editor de propósito general y ofrece otras funcionalidades que muchos usuarios genéricos agradecerán.
Lo primero que puede resultar interesante para el usuario común es la posibilidad de utilizar el ratón. Para ello ejecutamos el siguiente comando:

```
:set mouse=a
```

El uso del ratón se puede activar de forma independiente en cada uno de los modos de operación de vim (normal, visual, inserción,...). El carácter “a” en el comando anterior indica a vim que active el ratón en todos los modos.
Activar el ratón nos permitirá posicionar el cursor en nuestro texto o utilizar la rueda para desplazar el texto arriba o abajo. También podremos utilizar el ratón para seleccionar bloques de texto en modo visual. Allá cada cual, la verdad es que una vez que te acostumbras a utilizar las teclas, el ratón es mucho más lento :).
Otra funcionalidad que puede resultar útil es el uso de abreviaturas, sobre todo cuando estamos escribiendo un documento en el que ciertas palabras largas se repiten con mucha frecuencia. Para crear una abreviatura utilizamos el comando :iabbrev o simplemente iab. Por ejemplo:

```
:iab ester esternocleidomastoideo
```

A partir de ese momento, cada vez que escribamos la palabra “ester” en nuestro texto, se sustituirá automáticamente por el nombre completo de este conocido músculo.
Podemos eliminar las abreviaturas con el comando “:una” o “:unabbreviate” o eliminarlas todas con el comando “:iabclear”.
La “i” al principio del comando le indica a vim que la abreviatura se utilizará en modo inserción, es decir, cuando estamos escribiendo. Pero podemos crear abreviaturas para el modo línea de comandos. Esto se consigue sustituyendo la “i” inicial por una “c”.

```
:cab ls files
```

Ahora cada vez que escribamos ls en la línea de comandos este será sustituido por files.

MAPEADO

De forma similar funciona el mapeado, pero en este caso asociamos secuencias de caracteres a teclas concretas de nuestro teclado. Al igual que sucede con las abreviaturas, el mapeado de teclas se puede realizar de forma independiente para los distintos modos de operación. Concretamente.

normal	nmap o map
visual	vmap o map
insert	imap o map!
command-line	cmap o map!
operator-pending	omap o map!

La forma de utilizar estos comandos es muy sencilla, y ya la introdujimos de forma indirecta cuando hablamos de las facilidades que vim nos ofrecía para programar. Pero las posibilidades que el mapeado de teclas de vim nos ofrece van mucho más allá. Veamos solo un par de ejemplos:

```
:imap <F3> <Esc><Right>n<Insert>
```

Este comando nos permite utilizar la tecla F3 en modo edición a modo de “buscar siguiente”. Cada vez que pulsemos F3 la secuencia de teclas ESC, cursor derecho, comando “n” y vuelta a modo inserción se ejecutará. Si recordáis, el comando n nos permite movernos a la siguiente ocurrencia de una búsqueda, pero para ello debemos estar en modo comando (pulsando <ESC>).
Y aquí un par de mapeados que os pueden resultar útiles o cuanto menos familiares:

```
:map <S-S> <Esc>:w<CR>
:vmap <C-C> yy
:imap <C-V> <Esc>p<Insert>
```

Finalmente, y para terminar con este grupo de facilidades para lidiar con tareas repetitivas, vim, como la mayoría de editores de textos, permite “grabar” secuencias de operaciones y posteriormente repetirlas de forma muy sencilla.

Para empezar a grabar nuestras acciones tenemos que utilizar el comando “q”, seguido de un caracter que identificará la secuencia de acciones que realizaremos a continuación. Cuando hayamos terminado, simplemente debemos pulsar “q” de nuevo para finalizar la grabación.
A partir de este momento cada vez que pulsemos la tecla “@” seguida del identificador que seleccionamos al iniciar nuestra grabación, la secuencia de acciones grabadas será reproducida.
Esta funcionalidad, en combinación con la capacidad de vim para ejecutar un comando varias veces simplemente indicando el número de vez que lo queremos ejecutar, es una muy potente herramienta para llevar a cabo modificaciones mecánicas en un texto de forma muy rápida.

PERSONALIZANDO VIM

A lo largo de este artículo hemos visto un montón de características muy interesantes de vim, pero ninguna de ellas permanente, es decir, cuando cerremos la sesión actual, perderemos todas nuestra abreviaturas, mapeados de teclas y demás configuraciones.
Semejante mala bestia no podía haber pasado por alto algo tan obvio como esto, y por su puesto que no lo ha hecho. Para ello no tenemos más que editar el fichero .vimrc que encontraremos en nuestro directorio home. No tenéis más que editarlo y añadir cualquiera de los comandos que hemos visto en este artículo, o cualquier otros que hayáis encontrado vosotros mismos (hay un montón más :)
Uno de los comandos que es recomendable añadir a .vimrc es “syntax on”, en el caso que os hayáis percatado que el realzado de sintaxis no funciona por defecto en vuestra instalación de vim.
Otros comando de interés, y del que no hemos hablado explícitamente a lo largo de este artículo es “set”. Este comando nos permite establecer los valores de ciertas variables globales de vim que condicionan su comportamiento. Algunas de estas variables las hemos ido introduciendo junto con la funcionalidad relacionada, pero hay otras muchas. Aquí os incluimos una lista de las más útiles:

set number	Muestra números de línea
set tabstop=4	Modifica el tamaño de los tabuladores
set list	Muestra caracteres especiales
set incsearch	Realiza las búsquedas según la tecleamos
set laststatus=2	Muestra la barra de estado permanentemente
set autoindent	Activa la autoindentación. Útil para programar
set bg=dark	Modifica los colores si el fondo de pantalla es oscuro
set bg=light	Idem pero con fondo de pantalla claro.
set all	Muestra el estado de todas las variables.

No, no existe un comando unset. Normalmente para desactivar una opción se utiliza el mismo parámetro utilizado para activarla, pero precedido de “no”. Así, los siguientes comandos desactivan las opciones anteriores.

```
set nonumber
set nolist
set noincsearch
set noautoindent
```

Por supuesto todos estos comandos los podéis incluir en vuestro .vimrc para que cada vez que lo arranquéis todo esté como a vosotros os gusta :)

MÁS PERSONALIZACIÓN

Esta sección es solo para los más curiosos, o simplemente para los que hayáis leído hasta aquí. No vamos a profundizar en ellas porque normalmente, un usuario normal no tendrá que utilizarlas nunca, pero un usuario avanzado, debe, al menos, saber que existen.
Pues sí, vim proporciona su propio lenguaje de scripting, con el que poder añadir nuevos comandos si es que habéis encontrado algo que todavía no hace. El lenguaje es relativamente sencillo y en internet podréis encontrar varios tutoriales para iniciaros en él.
La otra funcionalidad que podéis personalizar es el realzado de sintaxis. Si bien, pocas cosas debe haber por ahí que vim no sea capaz de colorear, sois muy libres de crear vuestros propios ficheros de realzado de sintaxis para aplicaciones completas.
Si estáis en este punto, debéis saber que todos los ficheros que utiliza vim se encuentran en “/usr/share/vim/”. Allí encontraréis scripts, ficheros de sintaxis y un montón de sorpresas más.

PARA TERMINAR

Y para terminar un poco de humor. En vuestra sesión de vim teclead alguno de los siguientes comandos:

```
:help!
:help 42
:help holy-grail
:help quotes
```

Si queréis pasar un rato divertido no podéis dejar de instalar y probar vigor, una versión de vi con un asistente como el mítico clip aquel que no nos dejaba en paz.
También existen versiones “gráficas” de vim como gvim que nos proporciona un menú para acceder cómodamente a la mayoría de las funcionalidades que nos ofrece. Si con todo lo que os hemos contado, todavía os sigue pareciendo difícil de usar, probad “cream for vim”, una serie de macros orientados a facilitar la vida del principiante.
Finalmente, si queréis comparar ficheros y visualizar gráficamente sus diferencias... ahí está vimdiff.
Esto es todo... quién lo diría verdad?... Menuda mala bestia!!!.
Por cierto, hemos dejado lo mejor para la sección de trucos. No os la perdáis.

APUNTES EN INTELIGENCIA ARTIFICIAL: MACHINE LEARNING (APRENDIZAJE)

por Óscar Martínez Mozos
(omozos@googlemail.com)

En este artículo hacemos una pequeña introducción a la inteligencia artificial (IA). Esta disciplina comprende muchas y diversas áreas de conocimiento e investigación, y sus metodologías cada vez se usan más en la tecnología actual. Seguro que habréis oído hablar de coches inteligentes, búsquedas inteligentes en la web, etc. Aunque el nivel de inteligencia de estos sistemas es un poco *relativo*, si es cierto que utilizan métodos provenientes del área de inteligencia artificial. En este artículo también nos centraremos un poquito en un subcampo bastante importante de la IA llamado *machine learning* (aprendizaje).

¿QUÉ ES LA INTELIGENCIA ARTIFICIAL?

Esta es una buena pregunta que, por desgracia, no tiene una contestación clara. Probablemente la imagen más extendida de la inteligencia artificial podría ser la de un robot que se comporta como un humano además de parecerse a él, algo al estilo de la película *Yo, Robot*. Sin embargo, la IA comprende muchas otras áreas aparte de la robótica.

Buscar una única definición para la IA es difícil, así es que vamos a recoger diferentes ideas sobre lo que es un sistema inteligente y organizarlas un poco de acuerdo con su finalidad. Nos basaremos en la clasificación que Russell y Norvig presentan en su libro *Artificial Intelligence: A modern approach* [5]. Las distintas definiciones están organizadas en la Tabla 1, y a continuación pasamos a explicarlas en más detalle.

Los *sistemas que actúan como humanos* pretenden tener un comportamiento similar al de una persona. En este caso se supone que el hombre es el ser más inteligente, y comparamos la inteligencia de las máquinas con la suya. El científico Alan Turing propuso en 1950 el *test de Turing*. En este test, una persona entabla una conversación durante 5 minutos con el sistema artificial que queremos probar. El test se supone *superado* por el sistema artificial si durante más del 30 % del tiempo, la persona no sabe distinguir si las contestaciones provienen de una persona o de un sistema artificial. Existen varios sistemas para engañar a la gente durante una conversación, como por ejemplo el chatbot Mgonz o los programas Eliza y Alice. Sin embargo ninguno ha sido capaz de superar el 30 % ante un jurado preparado.

Es posible que algún lector cinéfilo haya pensado en la película *Blade Runner* al oir hablar del test de Turing. En esta película el policía caza-replicantes (Harrison Ford) aplica un test a las personas para detectar si son replicantes. El test de la película se basa en la diferente respuesta empática y emocional de un ser artificial con respecto a un humano. En realidad el test de empatía en *Blade Runner* está muy bien fundamentado, ya que uno de los principales problemas que se plantean en la inteligencia artificial es como hacer que un robot tenga sentimientos.

Siguiendo con la clasificación de sistemas inteligentes, los *sistemas que piensan como humanos* intentan modelar el funcionamiento de la mente humana. A esto se dedica más concretamente la *ciencia cognitiva*, un campo interdisciplinar que une la IA junto con la psicología y la neurociencia, con la finalidad de intentar entender como funciona el cerebro de las personas. Últimamente existe un gran interés en informática e IA por los denominados *sistemas cognitivos*.

La idea es crear sistemas artificiales que emulen la forma de entender el mundo tal y como lo hacen las personas. Algunos grandes proyectos europeos de investigación como CoSy [1] se dedican a este área.

Los *sistemas que piensan de forma razonada* se basan en las reglas de la lógica: "Sócrates es un hombre; todos los hombres son mortales; por lo tanto Sócrates es mortal". Un sistema basado en reglas de lógica pretende representar los distintos elementos que existen en el mundo y la relación entre ellos. El principal problema de este enfoque es que no parece muy factible representar *todo* lo que existe junto con *todas* sus relaciones.

Los sistemas que piensan de forma razonada se basan en las reglas de la lógica

Por último, los *sistemas que actúan de forma razonada* vendrían a estar representados por *agentes racionales*. Estos agentes perciben el entorno, razonan sobre él, y actúan de forma que maximizan la probabilidad de conseguir un objetivo. Pongamos por ejemplo un computador que juega al ajedrez, el famoso Deep Blue de IBM. Deep Blue tiene que localizar las piezas del contrario, entender la situación del juego, y generar un movimiento que maximice sus posibilidades de ganar. A lo mejor, alguno se preguntará porqué diferenciar los sistemas que actúan como humanos de los que actúan de forma razonada. Creo que la respuesta podría ser esta: los humanos, por suerte, también actuamos por sentimientos y, a veces, por intuición.

SISTEMAS QUE ACTUAN COMO HUMANOS	SISTEMAS QUE ACTUAN DE FORMA RAZONADA
SISTEMAS QUE PIENSAN COMO HUMANOS	SISTEMAS QUE PIENSAN DE FORMA RAZONADA

Tabla 1. Clasificación de Sistemas Inteligentes

La clasificación anterior es un poco teórica e incluso filosófica. Sin embargo, podemos presentar la IA simplemente listando las disciplinas que abarca. Algunos ejemplos son: lenguaje natural, representación del conocimiento, razonamiento, aprendizaje, robótica, planificación, visión artificial, teoría de juegos, y otras más. Todas ellas se basan fundamentalmente en la lógica y las matemáticas, siendo muy importantes la estadística y la probabilidad. En este artículo vamos a centrarnos en el aprendizaje, por ser una de las áreas más importante dentro de la IA. En la siguiente sección describiremos más detalladamente lo que se entiende por aprendizaje en IA, y después trabajaremos con un ejemplo práctico.

MACHINE LEARNING (APRENDIZAJE)

Para describir lo que es Machine Learning (ML), usaremos algunas de las ideas del libro de Tom Mitchell *Machine Learning* [4]. Según este libro, ML es el campo relacionado con las ideas necesarias para construir programas que automáticamente mejoran (o aprenden) con la experiencia. Esta definición puede ser un poco confusa, ya que puede dar la idea de un programa que mejora con el paso del tiempo. Sin embargo, la mayoría de las técnicas utilizadas en ML están relacionadas con aprendizaje en una sola vez, es decir, el sistema aprende de ejemplos que se le muestran una vez y luego aplica su aprendizaje posteriormente, pero no modifica lo aprendido la primera vez. Aunque también existen métodos para que el ordenador modifique sus modelos aprendidos con el paso del tiempo (aprendizaje continuo), éstos todavía forman una parte pequeña dentro de ML.

Otra cuestión importante es qué se entiende por aprender cuando hablamos de ordenadores. En general, cuando se habla de aprender, la idea es la de encontrar un modelo, basado generalmente en la estadística y la probabilidad, que explique *algo*, de forma que posteriormente el ordenador sea capaz de reconocer de nuevo ese *algo* cuando se le presente. Cuando hablo de *algo* me refiero a situaciones, objetos, acciones, o cualquier otra cosa que se os pueda ocurrir que pueda ser reconocido de alguna forma.

Seamos más concretos poniendo algunos ejemplos. Podemos querer programar un ordenador para que reconozca los distintos tipos de fruta cuando se le presenten. En este caso hablamos de reconocimiento de objetos. También podemos programar un reconocedor de lenguaje natural. En este caso clasificamos sonidos en palabras y palabras en frases. En cualquiera de los dos casos anteriores el ordenador primero tiene que aprender. En el caso de la fruta, tiene que aprender lo que es una manzana y como se diferencia de una naranja. En el caso de lenguaje natural, el sistema tiene que aprender cuales son los fonemas que forman una palabra. Por ejemplo, los fonemas "m" + "e" + "s" + "a", y por este orden, forman la palabra "mesa". Además, "mesa" es ya una palabra completa y no habrá que añadir más fonemas, excepto si es plural. Como vemos aquí, existen muchas y diferentes situaciones en los que el ordenador puede aprender.

Los sistemas supervisados usan ejemplos para aprender

En este artículo nos centraremos en el aprendizaje supervisado. En este tipo de aprendizaje, el ordenador crea su modelo usando ejemplos que se le muestran junto con su clasificación. En el ejemplo de la fruta, le enseñaríamos al ordenador cada vez una fruta y le diríamos su nombre, entonces el ordenador generaría un modelo en base a los ejemplos que le enseñamos con su correspondiente clasificación. Después de aprender, el ordenador debería de ser capaz de acertar el nombre de las piezas de fruta que le enseñemos posteriormente.

La clasificación de objetos nos sirve para introducir un nuevo problema muy importante en ML, y es, cuáles son las características o *atributos* que el ordenador debe usar para su modelo. Por ejemplo, en el caso de la fruta, el nombre del vendedor al que se la compramos no parece un buen atributo para clasificarla. Sin embargo, el color parece mucho más adecuado, o quizás la forma, o el tamaño, o quizás estas tres últimas al mismo tiempo.

Creo que ahora tenemos un poco más claro lo que esperamos de un algoritmo de aprendizaje. Primero, que sepa elegir los atributos adecuadas para su modelo. Y segundo, que sepa establecer los parámetros del modelo.

Cuando hablamos de aprender un modelo, nos referimos la mayor parte de las veces a entrenar un *clasificador*. Un clasificador, también llamado hipótesis, es una función que clasifica unos atributos de entrada dentro de uno de los posibles conceptos de salida. Pongamos un ejemplo extraído del libro *Data Mining*, de Witten y Frank [6]. La idea es aprender un clasificador para determinar cuando un día es bueno para jugar al fútbol. Para ello tenemos recogidas una seria de medidas durante 14 días con nuestra opinión sobre si el día es bueno o no.

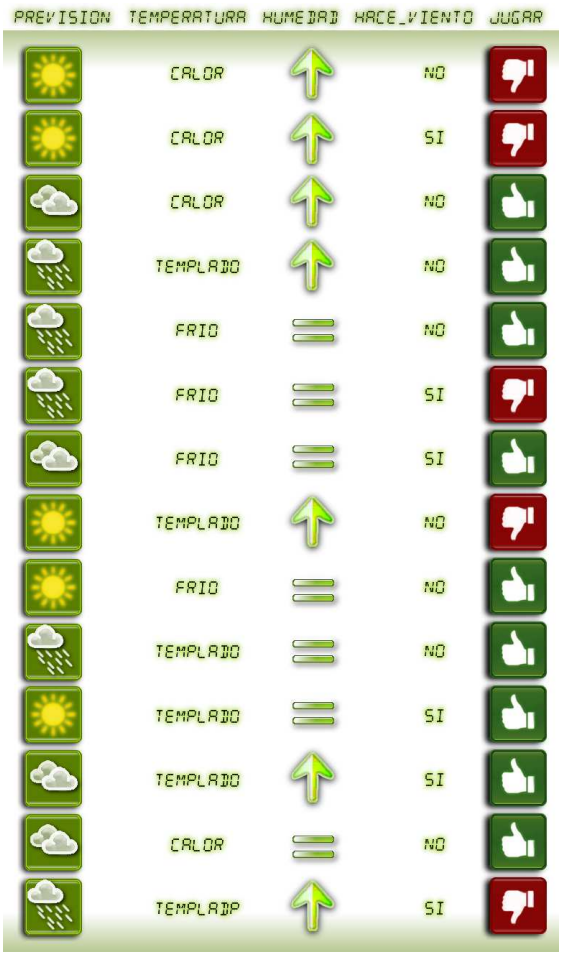


Tabla 2. Valores de los Atributos para 14 días.

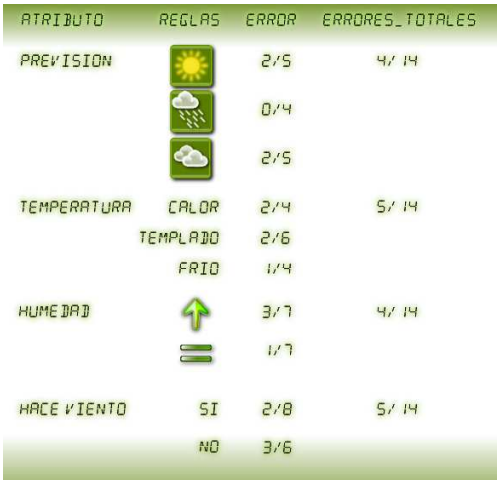


Tabla 3. Conjunto de Reglas.

La Tabla 2 muestra los atributos de cada día junto con su clasificación para jugar al fútbol (SI o NO), en el atributo `jugar`. Ahora tenemos que escribir un programa que analice los datos y nos devuelva nuestro clasificador.

UN CLASIFICADOR SENCILLO: UNA REGLA

Uno de los clasificadores mas sencillos que podemos obtener es una simple regla `if-then-else`. Este clasificador se denomina también $1R$ (1-Rule). La idea es elegir un sólo atributo de entre todos para clasificar el día. Por supuesto, habrá que elegir el atributo con el que menos nos equivoquemos. La idea del clasificador $1R$ es la siguiente [6].

Cogemos un atributo, `previsión`, y establecemos uno de sus valores, `soleado`, y miramos el valor que se repite más en el atributo que queremos aprender, en nuestro caso `jugar`. Al atributo a aprender también se le denomina *concepto* y a sus valores se les denomina *clases*. Bien, pues de acuerdo con la Tabla 2, si sólo elegimos el atributo `previsión` junto con su valor `soleado`, entonces la clase que más se repite en el concepto `jugar` es `NO` (3 veces `NO` y 2 veces `SI`). Como `NO` es el más repetido, creamos una regla del tipo:

```
if previsión = soleado then jugar = NO
```

Si aplicamos sólo esta regla, nos habremos equivocado en dos ocasiones de las 5 posibles cuando `previsión=soleado` (las dos veces en que `jugar=SI`), es decir, cometemos un error de 2/5.

Ahora repetimos el proceso con el resto de atributos y sus valores, obteniendo la Tabla 3. Vemos que en esta tabla también hemos calculado un error total por cada conjunto de reglas correspondiente a un solo atributo. Para ello simplemente sumamos los errores de cada regla.

Ahora sólo nos falta elegir el conjunto de reglas correspondiente al atributo con menor error total. En nuestro caso podemos elegir entre el atributo `previsión` o el atributo `humedad`, pues los dos tienen un error total de 4/14. Siguiendo la idea de *Occam's razor*, elegimos la regla más simple:

```
if humedad = alta then jugar = NO
elseif humedad = normal then jugar = SI
```

Alternativamente, la regla eligiendo el atributo `previsión` sería como sigue:

```
if previsión = soleado then jugar = NO
elseif previsión = nublado then jugar = SI
elseif previsión = lloviendo then jugar = SI
```

El pseudocódigo para calcular el conjunto de reglas final podría ser algo parecido al siguiente:

```
Para cada atributo,
  Para cada valor del atributo,
    encontrar la clase más frecuente
    generar una regla asignando esta clase
  Calcular el ratio de error total del \
  conjunto de reglas
Elegir el conjunto de reglas con menor error total
```

Bueno, pues ya hemos programado un algoritmo de aprendizaje para saber si jugar al fútbol o no. Ahora sólo sería cuestión de utilizar la regla final elegida y aplicarla a cada nuevo día.
Vamos a repasar cuales son las diferentes componentes del sistema de aprendizaje de 1R. Primero tenemos los datos iniciales (Tabla 2) también llamados *datos de entrenamiento*. Luego tenemos la forma del modelo (o hipótesis) que queremos aprender, en este caso una regla `if-then-else`. Luego tenemos el método para elegir los parámetros de nuestro modelo (pseudocódigo). Y finalmente tenemos los parámetros de nuestro modelo, que son el atributo elegido (`humedad`) y sus valores (`alta, normal`). Este esquema de aprendizaje es el más común en los métodos de Machine Learning.
Y llegados a este punto terminamos esta breve introducción a la inteligencia artificial y al aprendizaje. Esperamos que os haya gustado y que en alguno se haya despertado el interés por la IA. También nos gustaría mucho que nos dierais vuestra opinión para saber si queréis que sigamos con este tema.

ALGUNAS NOTAS ADICIONALES

- El libro *Artificial Intelligence: A modern approach* [5] es una lectura básica para todo aquel que quiera introducirse en la IA. Este libro es usado por más de 1000 universidades en 91 países. Del mismo modo los libros *Machine Learning* [4] y *Data Mining* [6] son referencia obligada en el área de aprendizaje. Todos los métodos explicados en el libro *Data Mining*, incluido 1R, están progamados en lenguaje Java y pueden ser encontrados en el software de aprendizaje de libre distribución WEKA [2]. De hecho, parte del libro explica como usar WEKA.

- La vida de Alan Turing es muy interesante. Trabajó como desencriptador durante la segunda guerra mundial y fue perseguido por sus tendencias homosexuales. Su muerte aún no esta aclarada. Además del test de Turing, Alan Turing también creo la *máquina de Turing*, base la informática actual. Podéis encontrar más información en [3] y por supuesto también en Wikipedia.

- El chatbot Mgonz y los programas Eliza y Alice (o sus diversas implementaciones) pueden encontrarse fácilmente en Internet. Intentad charlar con alguno si tenéis tiempo.

- Cognitive Systems for Cognitive Assistants (CoSy) [1] es el proyecto dentro del cual estoy realizando mi doctorado. Este proyecto ganó el premio a la mejor propuesta en el marco europeo FP5. Mi labor se centra en el escenario de exploración y representación de entornos cerrados (Explorer).

Sobre el Autor

Me llamo Óscar Martínez Mozos y trabajo como investigador en la universidad "Albert-Ludwigs-Universitaet Freiburg", en Freiburg, Alemania. Actualmente escribo mi tesis doctoral sobre robótica móvil, bajo la supervisión del catedrático Wolfram Burgard, un investigador importante en este tema. Anteriormente estuve realizando investigación en visión artificial y natural en el Instituto de Bioingeniería en la universidad Miguel Hernández, en Elche, Alicante. Mis áreas de investigación son, principalmente, aprendizaje, robótica móvil, visión, e inteligencia artificial en general. Para más información podéis visitar mi página web:
http://www.informatik.uni-freiburg.de/ omartine/
También podéis poneros en contacto conmigo en la dirección de email: `omozos@googlemail.com`
Estaré encantado de contestar a vuestros correos sobre este o cualquier otro tema.

Referencias

[1] http://www.cognitivesystems.org/.
[2] http://www.cs.waikato.ac.nz/ml/weka/.
[3] http://www.turing.org.uk/.
[4] Tom M. Mitchell. *Machine Learning*. McGraw Hill, 1997.
[5] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2003.
[6] Ian H. Witten and Eibe Frank. *Data Mining*. Morgan Kaufmann, 2000.

Crea tu Propio Superdemonio (y II)

escuchando en varios puertos a la vez

por er del aberno

Sí, ya estamos muy cerca de disponer de nuestra propia versión de inetd. Nuestro propio superdemonio. Como os adelantábamos en el número anterior una de las cosas que nos falta por hacer es que nuestro servidor pueda escuchar en varios puertos a la vez y ejecutar distintos programas en función del puerto utilizado.

Por cierto, antes de empezar, coged la última versión de nuestro demonio, la que hicimos en el número anterior, ya que la tomaremos como base en este artículo y la iremos modificando poco a poco. Os recomendamos que descarguéis el paquete asociado de nuestra web, para ir probando sobre la marcha lo que aquí contamos... hace que todo esto sea mucho más fácil de entender.

MANOS A LA OBRA

Lo primero que tenemos que hacer, y esto no tiene nada que ver con las redes, es proporcionar a nuestro programa la lista de los puertos, y de los programas asociados a cada puerto que queremos que maneje. Nuestro querido superdemonio inetd hace esto utilizando el fichero /etc/inetd.conf que ya conocemos.

Nosotros, para mantener nuestro programa sencillo simplemente tomaremos esa información de la línea de comandos. Cada cual que lo haga como prefiera.

La función que interpreta la línea de comandos y las constantes y variables asociadas son estas:

```
#define N_PUERTOS 10

static int    s[N_PUERTOS];
static char *prg[N_PUERTOS];

void
procesa_params (int argc, char *argv[])
{
  int    i, j = 1;

  for (i = 0; i < N_PUERTOS;
       prg[i] = NULL, s[i++] = -1);

  i = 0;
  while (argv[j] && i < N_PUERTOS)
    {
      s[i] = crea_server_socket (atoi(argv[j++]));
      prg[i++] = strdup (argv[j++]);
    }
}
```

ATENCIÓN:

Niños, no hagáis esto en vuestras casas, el uso de variables globales suele producir graves efectos secundarios... y puede haceros perder horas buscando errores... Nosotros es que somos asín y necesitamos que los listados no sean muy largos para que los podáis seguir, pues nos hemos permitido algunas licencias.

Como podéis ver simplemente declaramos dos matrices. La primera contendrá nuestros sockets de servidor, en los que vamos a esperar las conexiones, y la segunda contendrá los nombres de los programas a ejecutar cuando se acepte una conexión en uno de los sockets s. Sí, podéis crear una estructura para manejar las dos cosas juntas... es más largo (por eso no lo hemos hecho), pero queda mejor :)

Ahora añadimos una llamada a esta nueva función al principio de nuestro main, pasándole como parámetros argc y argv.

```
int main (int argc, char *argv[])
{

  procesa_params (argc, argv);
  ...
}
```

SOCKETS BLOQUEANTES

Hasta este momento, hemos estado utilizando una semántica bloqueante con nuestros sockets. Lo que esto significa es que si queremos leer algo de la red y no hay nada que leer, nuestro proceso quedará bloqueado en esa instrucción hasta que recibamos algún dato.

Los sockets bloqueantes bloquean nuestro proceso hasta que puedan completar una cierta operación

Aceptar conexiones es un caso particular de lectura, y por lo tanto, cuando ejecutamos la llamada al sistema accept y no existe ninguna conexión pendiente, nuestro proceso queda bloqueado en esa instrucción hasta que alguien se conecte con nosotros. Si queremos aceptar conexiones en varios puertos diferentes... pues tenemos un problema.
Si escribimos un código como el siguiente, que es lo único que podemos hacer por el momento:

```
int  sa1, sa2, s1, s2;

sa1 = crea_server_socket (2000);
sa2 = crea_server_socket (3000);

while (1) {
  s1 = acepta_conexion (sa1);
  s2 = acepta_conexion (sa2);
  ...
}
```

Como podéis observar, estamos creando nuestros dos sockets de servidor y a continuación comenzamos a aceptar conexiones.
El problema es que si nuestro socket es bloqueante, nuestro programa se quedará parado en el primer accept (el del puerto 2000).
Pero si en ese momento llega un intento de conexión al segundo puerto (el 3000), nuestro programa no podrá atenderlo, ya que está parado, esperando conexiones en el primer puerto (el 2000)... es decir, todavía no se ha ejecutado el segundo acepta_conexion.
De la misma forma, si recibimos un intento de conexión en el puerto 2000, pasaremos a esperar conexiones en el puerto 3000, de forma que un nuevo intento de conexión al puerto 2000 tampoco sería atendido.
Existen tres formas fundamentales de lidiar con esta situación.
La primera consiste en crear una proceso o mejor una hebra, por cada puerto en el que queramos aceptar una conexión. Todas ellas se ejecutarán concurrentemente y no habrá problema con quedarse bloqueado en la llamada al sistema accept.
La segunda consiste en hacer que nuestros sockets utilicen una semántica no bloqueante. En este caso, el proceso no se bloqueará en las llamadas al sistema como accept o read. Si hay algo que leer o una conexión que aceptar el resultado será el habitual, sino recibiremos un error indicando que no hay nada que hacer. Para el problema que nos ocupa, estaríamos continuamente ejecutando accept y, la mayor parte del tiempo obteniendo como resultado un error indicando que no hay nada que hacer. Como los sockets son ahora no bloqueantes, si no hay nada que hacer con uno de ellos, pasaremos al siguiente y así sucesivamente.
La desventaja de esta forma de trabajar es que nuestro programa tiene que hacer lo que se conoce como espera activa, es decir, estaremos comprobando el estado de nuestros sockets continuamente, consumiendo CPU en el proceso.
La tercera forma de manejar múltiples conexiones, y en general la buena, aunque eso siempre depende de la aplicación, es utilizar la llamada al sistema select. Esta es la que nosotros utilizaremos con nuestro superdemonio.

SELECT

La llamada al sistema select nos permite "monitorizar" conjuntos de descriptores de ficheros. En lo que a nosotros respecta esos descriptores de ficheros serán sockets, pero en general pueden ser de cualquier tipo, una FIFO, un dispositivo en /dev o la entrada estándar.
Podemos monitorizar estos conjuntos de descriptores respecto a tres tipos diferentes de eventos, a saber: lectura, escritura y excepciones. Nosotros solo usaremos el evento de lectura, si bien los demás funcionan de la misma forma.

La llamada al sistema select nos permitirá manejar varios sockets a la vez

Cuando ejecutamos una llamada a select desde nuestro programa, éste pasará al estado "dormido", hasta que alguno de los eventos configurados ocurra o pase un cierto tiempo (ahora veremos como configurarlo). Una vez que uno de esos eventos ocurre, podremos llevar a cabo nuestra lectura, escritura o manejo de excepción sin tener que esperar.
La forma de utilizar select consta de tres partes:

- Configuración de la lista de descriptores de fichero a monitorizar y del timeout
- Ejecución de select y gestión de errores
- Comprobación del estado de los sockets y acción pertinente.

AÑADIENDO SELECT A NUESTRO PROGRAMA

Con todo lo que hemos visto hasta el momento, la función main de nuestro nuevo servidor tendrá la siguiente apariencia. La reproducimos entera por comodidad.

```
int main (int argc, char *argv[])
{
  fd_set              rfds;
  struct timeval      tv;
  int                 n, max, i;
  int                 terminar = 0;

  procesa_params (argc, argv);

  while (!terminar)
    {
      max = prepara_select (&rfds, &tv);
      if ((n = select (max, &rfds,
                       NULL, NULL, &tv)) < 0)
        perror ("select:");
      else
        {
          if (!n)
            {
              printf ("Me aburro!\n");
              continue;
            }
          /* Comprobamos sockets */
          procesa_conexiones (&rfds);
        }
    }
  return 0;
}
```

Lo primero que encontramos es nuestra función "procesa_params" que ya hemos comentado. A continuación nos encontramos un bucle infinito, en el que estaremos infinitamente aceptando conexiones y ejecutando procesos.

En el código se pueden apreciar claramente las tres partes necesarias para el uso de select. La función prepara_select se encarga de la configuración de la lista de descriptores de ficheros a monitorizar y la configuración del timeout.

Esta función también nos devuelve el valor máximo en la lista de descriptores de ficheros creada, valor que select espera como primer parámetro.

A continuación nos encontramos con la ejecución de select propiamente dicha. En este caso solamente estamos monitorizando un conjunto de descriptores para el evento de lectura. Los dos NULLs que siguen a continuación contendrían, respectivamente, las listas de descriptores para escritura y para excepciones.

El timeout de select nos permite ejecutar código mientras esperamos por la red.

La llamada al sistema select devolverá un valor negativo en caso de error (como es habitual), el valor 0 si ha vencido el timeout configurado con la variable tv sin recibir ningún evento y un número positivo, indicando cuantos de nuestros descriptores de ficheros han recibido un evento en el caso de que algún evento haya sucedido.

Como podéis ver el las líneas de código anteriores, en el caso de recibir un evento la función procesa_conexiones es ejecutada. En breve veremos lo que hace dicha función.

CONFIGURANDO SELECT

Para la configuración de select utilizamos una sencilla función llamada "prepara_select". Esta función recibe dos parámetros. El primero es una variable del tipo fd_set en el que almacenaremos la lista de descriptores de fichero que queremos monitorizar para los eventos de lectura.

El macro FD_SET nos permite determinar los socket que select monitorizará

Si quisiéramos monitorizar los otros tipos de eventos (escritura, excepción), tendríamos que utilizar dos variables adicionales.

La segunda variable es la que nos permite configurar el timeout, es decir, el tiempo que nuestro proceso permanecerá dormido como máximo. Si durante ese tiempo no recibimos ningún evento, select retornará el valor 0 y nos permitirá llevar a cabo las acciones que consideremos oportunas. Normalmente volver a configurar y ejecutar select.

Ambos parámetros se pasan como punteros a la función para permitir que ésta los modifique.

```
int
prepara_select (fd_set *rfds, struct timeval *tv)
{
  int  i, max;

  max = 0;
  FD_ZERO(rfds);
  for (i = 0; i < N_PUERTOS && s[i] != -1; i++)
    {
      FD_SET(s[i],rfds);
      if (s[i] > max) max = s[i];
    }
  max++;

  tv->tv_sec = 4;
  tv->tv_usec = 0;

  return max;
}
```

Como podéis ver en el código de prepara_select, el manejo de las variables del tipo fd_set se lleva a cabo con los macros FD_ZERO y FD_SET. El primero de ellos "vacía" la lista de descriptores que contiene la variable, mientras que el segundo añade un descriptor de fichero a la lista que se indica.

Es necesario saber cual es el mayor valor del descriptor de fichero que se añade a la lista. Este es el primer parámetro que select espera.

Tras inicializar nuestra lista de descriptores de ficheros, configuramos el timeout a utilizar. En este ejemplo es de 4 segundos. Observad que select permite especificar el timeout con una precisión de microsegundos (sí para eso es el campo tv_usec de la estructura timeval).

PROCESANDO CONEXIONES

Tras la ejecución de select, si se ha producido algún evento, tenemos que procesarlo. Los casos de error y timeout se procesan directamente en la función main. El tercer caso, un intento de conexión a alguno de nuestros puertos lo procesamos en una función a parte.

```
void
procesa_conexiones (fd_set *rfds)
{
  int  i, as;

  for (i = 0; i < N_PUERTOS; i++)
    if (s[i] != -1 && FD_ISSET(s[i],rfds))
      {
        printf ("Conexion a %d\n", i);
        as = acepta_conexion (s[i]);
        printf ("Conexion aceptada en %d\n", as);
        procesa (as, s[i], prg[i]);
        printf ("Ejecutando '%s'\n", prg[1]);
      }
}
```

Esta función, simplemente recorre la lista de nuestros sockets, para averiguar cual de ellos fue el responsable del evento que estamos procesando. Esta comprobación se lleva a cabo con el FD_ISSET, el cual nos indicará si el descriptor de fichero indicado ha recibido un evento dentro de la lista de descriptores (nuestra variable rfds) que se pasa como segundo parámetro.

Sabremos que socket ha producido el evento gracias al macro FD_ISSET

Si hemos recibido uno de esos eventos de lectura, eso significa que alguien está tratando de conectarse a uno de nuestros sockets, así que simplemente aceptamos la conexión con la misma función "acepta_conexion" que definimos en el artículo anterior y ejecutamos el proceso asociado a ese puerto con la función "procesa". Esta función "procesa" ha sido modificada ligeramente para que funcione con el nuevo esquema de gestión de conexiones, y la veremos un poco más adelante.
Para los más impacientes, esta modificación consiste, simplemente, en que la función "procesa" cierra el socket de comunicación en el proceso padre después de crear el proceso hijo. Si no hacemos esto, todas las conexiones permanecerán abiertas, puesto que el socket sigue abierto en el proceso padre, aunque el proceso hijo lo haya cerrado (todos los descriptores de ficheros se cierran automáticamente cuando un proceso muere).
Si queréis ver cual es el efecto de esto, solo tenéis que comentar la línea con el close al final de la función "procesa".

LA NOCHE DE LOS MUERTOS VIVIENTES

Bien, ya casi hemos terminado. En este punto todo debería funcionar perfectamente. Para comprobarlo simplemente compilamos nuestro nuevo super-server y lo ejecutamos.

```
$ make server\_simple4
$ ./server_simple4 2000 ./echo.pl 4000 ./echo.pl
```

Si ahora nos conectamos al puerto 2000 o al puerto 4000 de localhost, nuestro pequeño servidor de echo escrito en Perl, se ejecutará y nos responderá con lo mismo que nosotros le digamos.
Realizad unas cuantas conexiones y luego ejecutad ps:

```
$ ps -e
13614 pts/8    Z+     0:00 [echo.pl] <defunct>
13620 pts/8    Z+     0:00 [echo.pl] <defunct>
13627 pts/8    Z+     0:00 [echo.pl] <defunct>
$
```

Sí, efectivamente. Son zombies (eso es lo que significa la Z de la tercera columna).
En la sección consultorio del primer número de "Occam's Razor" le contábamos a Van Helsing como tratar con estos malditos, pero por si alguien no tiene a mano ese legendario número, vamos a contaros como solucionar este problemilla aquí.
En nuestro caso particular, lo solución que vamos a adoptar es la de proporcionar una manejador de señales para poder ejecutar la llamada al sistema waitpid, cada vez que algún proceso hijo muera, y de esta forma dejar que vaya hacia a luz... a un mundo mejor...
El manejador de señales sería algo tal que así:

```
void handler (int s)
{
  pid_t pid;
  int  status;

  pid = waitpid( -1, &status, WNOHANG ) ;

  printf ("Child done with status %d\n", status);
}
```

Y en nuestra función main, al principio de la misma, tendremos que añadir una línea como esta:

```
signal (SIGCHLD, handler);
```

La señal SIGCHLD (de children... niños) se envía cada vez que un proceso hijo termina su ejecución. Sin embargo, esta ejecución no es completa hasta que el proceso padre se da por enterado... es decir, ejecuta la llamada al sistema wait. Si el proceso padre no ejecuta esa llamada al sistema, el proceso hijo mantiene sus recursos en memoria aún cuando su ejecución ya ha terminado, convirtiéndose en un proceso zombie... ni vivo ni muerto.

Si no tenemos cuidado nos invadirán los zombies

Nuestro manejador de señales nos va a asegurar que cada vez que uno de nuestros hijos pase a mejor vida, nosotros nos enteremos y podamos ejecutar la llamada al sistema wait y dejar así que el pobre descanse en paz.

UN POCO MÁS DE FLEXIBILIDAD. TCCLIB

¿Habéis leído ya la sección ratas de biblioteca?... Pues a que estáis esperando. En este número os contamos como utilizar tcclib para generar código en tiempo de ejecución a partir de fuentes C. Pues bien, no estaría nada mal utilizar esta librería en nuestro superservidor para poder escribir servicios de red mú rápido... vamos a ver como hacerlo.
Para que el código no nos quede demasiado largo declararemos algunas variables globales más y almacenaremos el código C a ejecutar en un fichero llamado "script.c" que se encontrará en el mismo directorio que nuestro servidor.

TCCLib nos permitirá implementar nuestros servicios como ficheros de texto

También aprovecharemos nuestra sencilla función para interpretar la línea de comandos. Ahora, las cadenas que acompañan a los puertos van a representar funciones en el fichero script.c, en lugar de procesos externos.
Para inicializar tcclib y cargar y compilar el código en el fichero script.c vamos a declarar una nueva variable global, definir un tipo para referenciar nuestros punteros a funciones y escribir una pequeña función que inicialice nuestro compilador run-time:

```
typedef int (*MIFUNC)(void);
static  TCCState            *tcc;

int carga_codigo (TCCState *s, char *nf)
{
  FILE          *f;
  char           b[1024];

  /* Lee fichero */
  memset (b, 0, 1024);
  f = fopen (nf, "rt"); fread (b, 1024, 1, f);
  fclose (f);

  tcc_set_output_type (s, TCC_OUTPUT_MEMORY);
  tcc_compile_string (s, b);
  tcc_relocate (s);
  return 0;
}
```

De forma mú rápida, hemos declarado el tipo de las funciones que implementarán nuestros nuevos servicios. Se trata de funciones que retornan un entero y no reciben ningún parámetro.
Dependiendo de vuestras necesidades concretas podéis cambiar este tipo según os convenga. A continuación declaramos la variable que contendrá nuestro entorno de compilación en memoria, y que inicializaremos en la función carga código. Una vez más recordad que ésta no es la mejor forma de hacer esto. Las variables globales dan problemas.
Para simplificar el resto del código definimos también una sencilla función para obtener los punteros a las funciones en nuestro fichero script.c.

```
MIFUNC lee_funcion (TCCState *s, char *nombre_func)
{
  MIFUNC        un_script;
  unsigned long v;

  tcc_get_symbol (s, &v, nombre_func);
  un_script = (void*)v;

  return un_script;
}
```

Ahora tenemos que hacer algunos cambios a nuestras funciones para adaptarlas a la nueva funcionalidad.

PEQUEÑOS CAMBIOS

Lo primero que vamos a hacer es añadir una nueva variable en la que almacenar el puntero a la función en "script.c" que asociaremos a cada servicio. Esto es fácil, junto al resto de variables s y prg añadimos esta nueva línea:

```
static MIFUNC mifunc[N_PUERTOS];
```

Ahora ya podemos modificar nuestra función procesa_params para que resuelva la dirección en memoria de la función que se recibe a través de la línea de comandos. La versión modificada de procesa_params será la siguiente:

```
void
procesa_params (int argc, char *argv[])
{
  int   i, j = 1;

  for (i = 0; i < N_PUERTOS;
       prg[i] = NULL, s[i++] = -1);

  i = 0;
  while (argv[j] && i < N_PUERTOS)
    {
      s[i] = crea_server_socket (atoi(argv[j++]));
      prg[i] = strdup (argv[j++]);
      mifunc[i++] = lee_funcion (tcc, prg[i]);
    }
}
```

Como podéis observar simplemente hemos añadido una nueva línea dentro del bucle principal de la función que inicializa nuestra matriz de punteros a funciones. Ahora, por cada puerto en el que nuestro servidor escuchará tendremos un socket, una cadena con el nombre de una función y el puntero a esa función en memoria.

Unos pequeños cambios y tendremos nuestro server funcionando de nuevo

La función procesa_conexiones también necesita una pequeña modificación. En este caso sustituiremos la líneas:

```
procesa (as, s[i], prg[i]);
```

por

```
procesa (as, s[i], mifunc[i]);
```

Es decir, en lugar de pasar el nombre del programa externo a ejecutar, pasamos el puntero a la función a ejecutar.
Finalmente, la función procesa también debe ser modificada. la nueva versión quedaría tal que así:

```
int procesa (int s, int sa, MIFUNC la_funcion)
{
  pid_t    pid;

  if ((pid = fork ()) < 0)
    fprintf (stderr, "No puedo crear el proceso");
  else
    {
      if (!pid) /* Proceso hijo */
        {
          close (sa);

          dup2(s, 0);
          close (s);
          dup2(0, 1);
          dup2(0, 2);

          la_funcion ();
          exit (0);
        }
    }
  close (s);
  return 0;
}
```

Como podéis ver el tercer parámetro de la función ahora es un puntero a una función, en lugar del nombre del proceso externo a ejecutar, y la llamada al sistema execv ha sido sustituida por la llamada a la función que recibimos como parámetro.

En unas pocas líneas de texto podemos implementar algunos servicios comunes

Para minimizar el número de cambios, hemos mantenido la llamada a fork para crear un nuevo proceso, si bien en este caso la creación de una hebra sería menos costosa... Bien, ahí tenéis una cosilla para entreteneros.
Finalmente, el bloque de inicialización en nuestro programa principal también se verá ligeramente modificado... no así el bucle principal que no requiere ningún cambio. Este bloque de inicialización será ahora:

```
main ...

  tcc = tcc_new ();
  carga_codigo (tcc, "script.c");
  signal (SIGCHLD, handler);
  procesa_params (argc, argv);
```

PROBANDO NUESTRO SUPERSERVER

Para probar nuestro super servidor de la muerte, tenemos que escribir un fichero script.c con las funciones en C que queremos ejecutar cada vez que un cliente se conecte a alguno de nuestros servicios.
Nosotros hemos escrito este sencillo script.c

```
#include <time.h>
#include <string.h>

int serv_echo (void)
{
  char buffer[1024];
  int  len;

  len = read (0, buffer, 1024);
  write (1, buffer, len);

  return 0;
}

int serv_time (void)
{
  time_t  hora;
  char   *la_hora;

  time (&hora);
  la_hora = (char*) ctime (&hora);
  write (1, la_hora, strlen (la_hora));
}

int serv_chargen (void)
{
  while (1)
    {
      if ((write (1, "AAAAAAAAAAAAAAAAAAAA", 20)) < 0)
        break;
    }
}
```

El fichero define tres funciones que ofrecen tres sencillos servicios que normalmente inetd implementa de forma interna: el servicio de echo, el servicio de hora y el servicio de generación de caracteres.
Ahora ya podemos ejecutar nuestro flamante servidor con, por ejemplo, esta línea de comandos:

```
./server_tcclib 2000 serv_echo 3000 serv_time 4000 serv_chargen
```

Ya podéis utilizar nc o telnet para comprobar como van nuestros nuevos servicios.

ESTO ES TODO

Bueno, esperamos que os haya resultado interesante esta breve explicación de cómo programar vuestro propio superdemonio. El resultado final os proporciona un esqueleto general para poder implementar sencillos protocolos mú rápido, sin tener que escribir ni una línea de código relacionada con la red.
Hemos dejado algunas cosas en el tintero para que os entretengáis con ellas... siempre es más divertido descubrir las cosas por uno mismo que que nos las cuenten. Con lo que os hemos contado hasta aquí, podéis descargar las fuentes de inetd (o su hermano mayor xinetd) y estudiar el código y averiguar hasta el último detalle de esta aplicación UNIX que ha estado ahí desde el principio.
Como siempre, estaremos encantados de saber de vuestros éxitos así como de cualquier uso curioso o mejora que hagáis sobre estos sencillos esqueletos de programa que os ofrecemos. Por supuesto, serán publicados en el próximo número con vuestras explicaciones.
En el próximo número nos introduciremos en un nuevo y excitante campo de la programación... pero esto es una sorpresa.

Trujamana o Localizadora

Introducción a la Localización de Software

por Silvia Carril Caldelas (silvia@ilialang.com)

Trujamán, na. (Del ár. hisp. turğumán, este del ár. clás. turğumān, intérprete, este del arameo rabínico tūrgmān[ā] y siriaco targmānā, y estos del acadio targamānu[m] o turgamānu[m]). 2. p. us. intérprete (|| de lenguas)

Hace años que ante la pregunta: "¿En qué trabajas?" he desistido de intentar dar una explicación lógica e inteligible a mi interlocutor. El responder "Soy localizadora de software" da pie a las reacciones y expresiones faciales más sorprendentes.

Lo cierto es que la mía es una profesión cada vez más en auge y, definitivamente, una cuyos resultados son disfrutados a diario por millones de usuarios de ordenadores y, sin embargo, continúa siendo una gran desconocida... no sólo para los propios usuarios sino también para programadores e ingenieros informáticos.

La globalización (ese concepto tan de moda ahora) aúna a gentes y países de todo el mundo. No obstante, aunque cada vez nos parecemos más un español de España y un americano de los EE.UU., continúa habiendo grandes diferencias, singularidades y "barreras" (por llamarle de algún modo). Una de ellas es el idioma y el uso que cada individuo hace del mismo en su vida cotidiana. Uno de esos usos es el que hacemos a diario millones de usuarios informáticos totalmente ajenos al hecho de que en esa aplicación de software que estamos utilizando sin darle mayor relevancia se ha invertido mucho tiempo y esfuerzo para lograr que llegue a nosotros en nuestro propio idioma.

¿QUÉ ES LA LOCALIZACIÓN?

El concepto de localización o internacionalización es muy simple, es decir, es la adaptación de aplicaciones de software a otros entornos, particularmente de otros países y culturas. Entre otros aspectos es responsable de asegurar aspectos, obvios para los usuarios de idiomas con base en el alfabeto romano, como la capacidad para representar y visualizar los sistemas alfabéticos de un idioma en cuestión. Y ésa es tan sólo la punta del iceberg...

La diferencia entre internacionalización (generalmente, abreviado como i18n o I18n o I18N) y localización (generalmente, abreviado como L10n o l10n) es sutil pero importante. La internacionalización es la adaptación de productos para su uso potencial virtualmente en cualquier parte del mundo, mientras que la localización implica la inclusión de características particulares para su uso con una configuración regional específica. Ambos procesos se complementan entre sí y son necesarios conjuntamente para lograr un sistema que funcione de modo global.

Los procesos de internalización y localización se abrevian como i18n y L10n

Así pues, la internacionalización y la localización son algo mucho más amplio y complejo que la traducción. Abarcan no solamente el aspecto esencial de la transposición de un idioma a otro del código que compone una aplicación sino que van más allá. Además de los aspectos puramente lingüísticos, en su aplicación también tratan cuestiones culturales y técnicas. Estos incluyen entre otros:

- La adaptación de formatos de hora y fecha para cada país, así como los diferentes calendarios existentes. Por ejemplo, el formato de fecha habitual en EE.UU. es especificar mes/día/año (10/12/2007), mientras que en España lo habitual es el formato día/mes/año (12/10/2007).

- La adaptación de formatos de número y separadores. Mientras que en inglés los números decimales se representan mediante un punto, en español se utiliza una coma. Así 225.32 en inglés se representa como 225,32 en español.

- La adaptación de formatos de moneda. De nuevo, en la mayoría de los países anglosajones el símbolo que representa la moneda del país se sitúa antes de la cifra unido a ésta sin espacios (£225.32). Lo correcto en castellano es que el símbolo vaya después de la cifra separado por un espacio en blanco (225,32 €).

- La representación de las horas también es otro factor que suele generar serios problemas en la localización de software. Aunque cada vez más se tiende a utilizar el formato inglés a.m. y p.m. lo correcto en español es el uso del formato de 24 horas seguido de h.

- La correcta adaptación de las fórmulas de cortesía. Bien conocida es la tendencia de los países anglosajones y latinoamericanos en el uso de fórmulas de cortesía del tipo Dr., Ing., etc.

- Los formatos de dirección son otro aspecto de vital importancia tanto en aplicaciones de software tradicionales como en las aplicaciones web. Con mucha frecuencia nos encontramos campos del tipo Estado/Condado, o códigos postales que no permiten la introducción de los formatos propios del país, etc.

- Asimismo, la localización dentro de un mismo idioma pero a sus diferentes variantes es esencial para evitar la alienación de los usuarios de cada país/cultura destino. Todos sabemos bien que mientras en España se usan ordenadores no así en Argentina, Colombia, etc. donde lo habitual es utilizar computadores. Esta situación es también similar entre el inglés de América, Reino Unido, Australia, Sudáfrica, etc.

Así pues, un localizador o localizadora es algo más que un simple traductor. Podría argumentarse que estos mismos principios son igualmente aplicables a la traducción; no obstante, estos se limitan exclusivamente a la única lengua de destino con que los traductores suelen trabajar, mientras que en la localización el abanico de idiomas y culturas destino es mucho más amplio.

Junto a esta apreciación cultural, un buen localizador o localizadora debe asimismo poseer conocimientos relativamente genéricos en programación y otros aspectos técnicos, que le ayudarán enormemente a la hora de detectar problemas en la localización.

Lamentablemente, pese a vivir en los tiempos que corren la mayoría de las aplicaciones se desarrollan desde un principio sin previsión ni preparación alguna para su adaptación a otros mercados lo que dificulta enormemente su posterior localización. Cuatro de los aspectos fundamentales que causan problemas a la hora de realizar el proceso de localización son la codificación dura (hard coding) de las cadenas de texto, la falta de espacio suficiente para garantizar la correcta visualización de la traducción de las cadenas de texto (en términos generales los textos traducidos al castellano son un 20 % más largos que el texto original en inglés) y la concatenación de cadenas de texto así como el uso variables. Estos problemas podrían evitarse fácilmente con el consiguiente ahorro de tiempo, esfuerzo y dinero si existiese desde un principio claridad en los objetivos empresariales a la hora de iniciar un proyecto determinado.

DIFICULTADES EN EL PROCESO DE LOCALIZACIÓN

Si nos internamos en las profundidades de la propia traducción de las cadenas de texto que componen una aplicación de software, el localizador o localizadora se enfrenta al problema de la falta de contexto. Al afrontar una traducción (en el sentido tradicional del concepto) el traductor dispone de un contexto e información tanto previa como posterior que el texto le facilita. En el caso de la traducción de aplicaciones de software esto, salvo contadas excepciones, es pura utopía. Incluso en aquellos casos en que es posible la introducción de campos de comentarios para facilitar el proceso de traducción (como es el caso de .NET, Passolo y algún otro) estas ayudas a la traducción no suelen llegar al traductor responsable de la localización de esa aplicación específica.

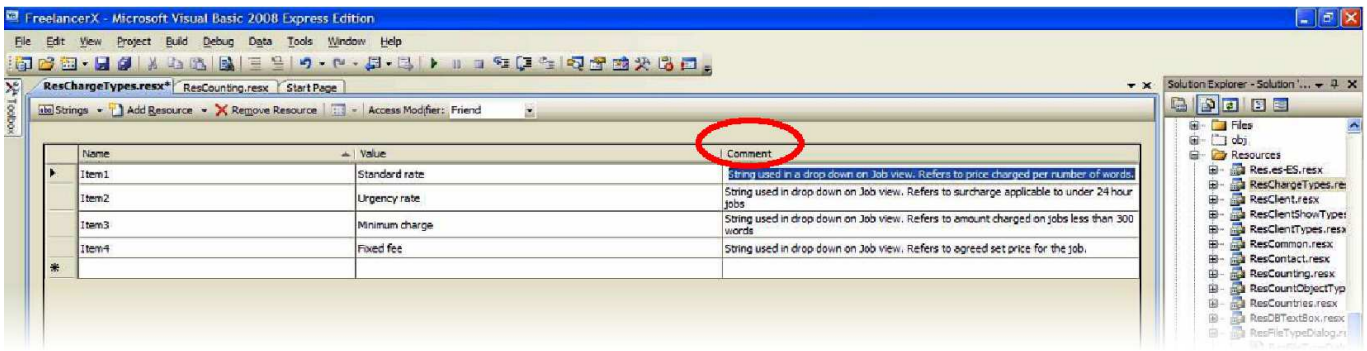


Figura 1. Comentario en .Net.

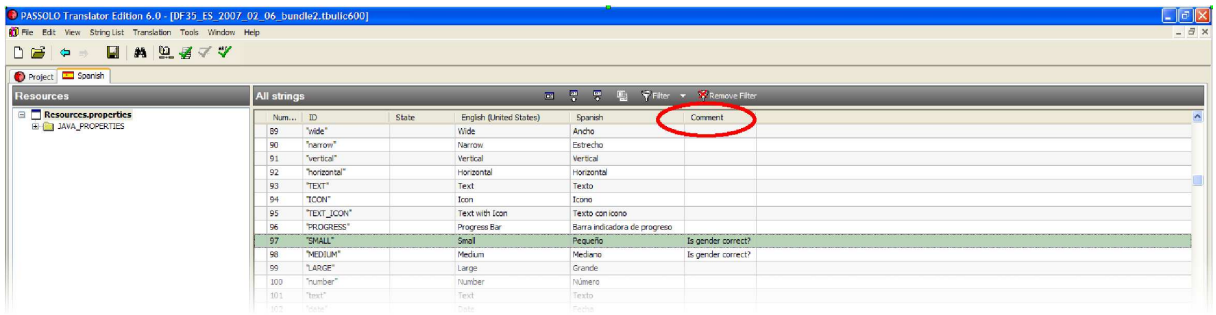


Figura 2. Ejemplo de cómo usar el campo de comentarios en Passolo para confirmar la traducción elegida.

Como resultado en ocasiones nos encontramos ante traducciones incorrectas y los usuarios nos sorprendemos ante el hecho de que se haya podido traducir un término de determinada manera cuando resulta evidente que es incorrecto. Éste es el caso del ejemplo a continuación tomado del último proyecto de localización en el que he trabajado.

En el ejemplo que se ilustra se muestra claramente la confusión que puede generar la traducción de una cadena sin disponer de su contexto en la aplicación. Aquí la traducción del inglés "languages" se prestaba a una traducción incorrecta debido a la propia naturaleza de la aplicación, una herramienta para la comparación de archivos de recursos creados en diversos lenguajes de programación.

```
IDD_OpenFileWithEncoding DIALOGEX 0, 0, 285, 33
STYLE DS_SETFONT | DS_3DLOOK | DS_FIXEDSYS | WS_CHILD | \
                   WS_VISIBLE | WS_CLIPSIBLINGS
FONT 8, "MS Shell Dlg", 0, 0, 0x0
BEGIN
  LTEXT           "&Encoding:",IDC_EncodingLabel,5,2,42,8
  COMBOBOX        IDC_Encodings,53,0,150,135,CBS_DROPDOWNLIST |\
                                      WS_VSCROLL | WS_TABSTOP
  PUSHBUTTON      "Open from URL...",IDC_OpenUrl,211,0,62,14,\
                                      NOT WS_VISIBLE
  CONTROL         "&Languages",IDC_ShowLanguagesForEncodings, \
                  "Button",BS_AUTORADIOBUTTON | \
                  WS_GROUP,53,17,51,10
  CONTROL         "&Codepages",IDC_ShowCodepagesForEncodings, \
                  "Button",BS_AUTORADIOBUTTON,107,17,52,10
END
```

¿Cómo determinar si la traducción correcta en castellano debe ser "idioma" o "lenguaje"? ¿Se refiere la cadena a un idioma humano o a un lenguaje de programación?

Afortunadamente, al tratarse de una aplicación pequeña que permite su instalación en el ordenador del usuario, logramos resolver la diatriba simplemente buscando la instancia de esa cadena en la propia aplicación.

Sin embargo, esto no siempre resulta posible ya que en ocasiones no es factible instalar la aplicación para aclarar la duda directamente, debido a los requisitos de hardware o cuando se trata de aplicaciones cliente servidor. En estos casos es únicamente el programador que ha escrito esa cadena quien puede colaborar con el localizador y arrojar luz para resolver la cuestión. Una comunicación interdepartamental fluida puede reducir considerablemente el tiempo de desarrollo de las aplicaciones además de proporcionar un resultado con una calidad claramente mejorada. De lo contrario el localizador se enfrenta a un escollo insalvable y la única opción disponible es aventurarse con una traducción para empezar a rezar a continuación.

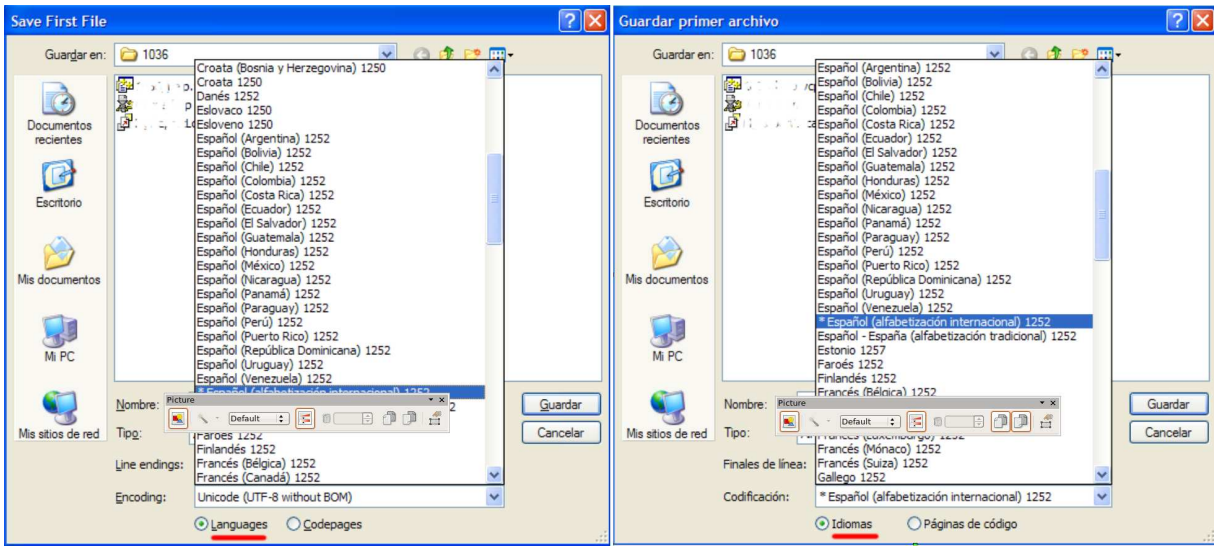


Figura 3. Ejemplo de traducción del término original "languages" en un contexto particular dentro de la aplicación.

Las traducciones incorrectas debido a situaciones similares a la del ejemplo abundan. He aquí un par de muestras tomadas de la localización de juegos:
"Complete": ¿Cómo traducirlo? Puede indicar que se ha superado un objetivo, un nivel o una misión determinada, o puede ser una instrucción (completar).
"All": Según a lo que se refiera será "todo, todos, todas"

La concatenación de cadenas de caracteres en los programas puede dificultar el proceso de localización

Otros ejemplos resultado de la falta de previsión para la internacionalización de los productos son la concatenación de cadenas y el uso de variables. A continuación se incluye una muestra de las hojas de cálculo utilizadas habitualmente por los localizadores para solicitar la aclaración de dudas. Estos ejemplos, están tomados también de uno de mis proyectos de localización más recientes (ver tabla al final de esta página).
Podemos verlo aún más claro en dos ejemplos ficticios tomados de Internet uno de variables y otro de concatenación en la localización de videojuegos para ilustrar este concepto.
En la cadena "John plays for `{TEAM_NAME}`." como se puede deducir, la variable es el nombre de un equipo de fútbol de la liga española. "John juega en el `{TEAM_NAME}`" es una traducción válida si el equipo es el Betis, por ejemplo. Pero ¿y si resulta que John juega en la Real Sociedad? "John juega en el `{REAL SOCIEDAD}`". Como siempre el problema de género nos juega una mala pasada, algo que no ocurre en inglés al ser un idioma mucho más sintético. Como siempre con un poco de imaginación y recursos se puede encontrar una solución factible del tipo: "John juega en el equipo: `{TEAM_NAME}`".
Veamos un ejemplo de problemas de concatenación en las siguientes cadenas:

1	Player %s wins a	El jugador %s consigue una
2	Red	roja
3	Yellow	amarilla
4	Green	verde
5	Motorcycle	moto

Si no se indica al traductor que la cadena 1 va a ir unida a la cadena 2, 3 ó 4 y luego a la 5, "red, yellow, green" podrían traducirse en masculino o femenino. Si sabemos que estos adjetivos acompañan a "motorcycle" ya sabemos que irán en femenino.
Durante el juego, el jugador realiza una acción y es premiado con una moto verde. El programa une entonces las cadenas de la siguiente forma: 1 + 4 +5:
El resultado en inglés:

> "Player %s wins a green motorcycle."

El resultado en español:

> "El jugador %s consigue una verde moto."

Lo que el jugador puede interpretar como una "mala traducción" es en realidad un problema de internacionalización.

Resource ID	String ID	English text	Query	ACCT Subject (Yes/No)	Category
000000067 .ProgramCurrentCostMetricsHelp.jsp	68	Filters for Programs or Projects where the actual costs are greater than the earned value by the specified dollar amount.	Should dollars be replaced with euros?	No	symbols
000000062. RequestSummaryBarChartHelp.jsp	191	# of Requests	Could you confirm that '#' means number and whether it would be okay to use the Spanish equivalent symbol °?	No	Convention
000000050. KPMXresourcespropertiesjs/49	CREATED_RESOLVED.TXT	created and resolved per	Please clarify what follows after this option in order to translate correctly.	No	String not completed
000000033. KEXPtResourcesproperties/1574	RED_EXCEPTION.TXT	of Tasks must have Exceptions to turn the Project Health	Please clarify what precedes and follows this string: i.e. is there a percentage value " %" before "of tasks"? Also, does a condition color follow "Project Health"?	No	
000000012. WorkPlanResources_en.properties/239	warning. ActualsStartWarning warning. ActualsFinishWarning	{0} constraint {1} cannot be satisfied because the constraint date {2} is {3} than the Actual Start Date for the task {4}. The Actual Start Date takes precedence and is reflected in the Scheduled Start Date. Resolution: If the Actual Start Date is in error, correct it and reschedule. Otherwise, changing or removing the constraint definition so that it does not conflict with the Actuals date will eliminate this warning.	I need to know what variables 0, 1, 3 and 4 will be replaced with in order to translate correctly.	No	Variables

En ocasiones la aparición de una traducción incorrecta o fuera de contexto se debe a situaciones como las descritas anteriormente pero, en otras ocasiones, se deben a la automatización de los procesos de traducción como es el caso del uso de las herramientas y memorias de traducción. ¡Atención a este punto! Hablamos de herramientas de traducción y no de traducción automática, dos conceptos diametralmente diferentes.

TRADUCCIÓN AUTOMÁTICA

El concepto de traducción automática no requiere mayor explicación ya que es ampliamente conocido por todos. Y abundan los ejemplos de traducciones atroces realizadas mediante todo tipo de "artilugios" informáticos, del tipo Bable Fish, etc. Ni que decir tiene que aunque cada vez este tipo de aplicaciones están más y mejor desarrolladas, los resultados generados no pueden equipararse a la traducción humana a ningún nivel. Aunque es cierto que en determinados contextos pueden resultar útiles para algunos usuarios al ayudarles a captar la idea general de un texto, no pueden considerarse traducciones propiamente dichas. Por eso no nos extenderemos más en su tratamiento.

HERRAMIENTAS Y MEMORIAS DE TRADUCCIÓN

Volviendo al punto de las memorias de traducción. Éste es, de por sí, un tema lo suficientemente extenso como para escribir uno y muchos más artículos sobre él. Para los no expertos en la materia: las herramientas de traducción asistida (generalmente conocidos en sus siglas en inglés como herramientas o aplicaciones CAT o Computer Assisted Translation) son aplicaciones de software cuya finalidad es la de asistir al traductor "humano" durante el proceso de traducción; una memoria de traducción es básicamente una base de datos usada por las herramientas CAT específicas diseñadas para ayudar a los traductores "humanos". Estas bases de datos contienen los segmentos de texto en el idioma origen junto con su traducción a uno o varios idiomas. Se diferencian de las bases de datos terminológicas en que suelen contener bloques de texto, párrafos, oraciones y frases, mientras que las otras contienen únicamente palabras o términos.

Figura 4: Base de datos terminológica Multiterm 5.0

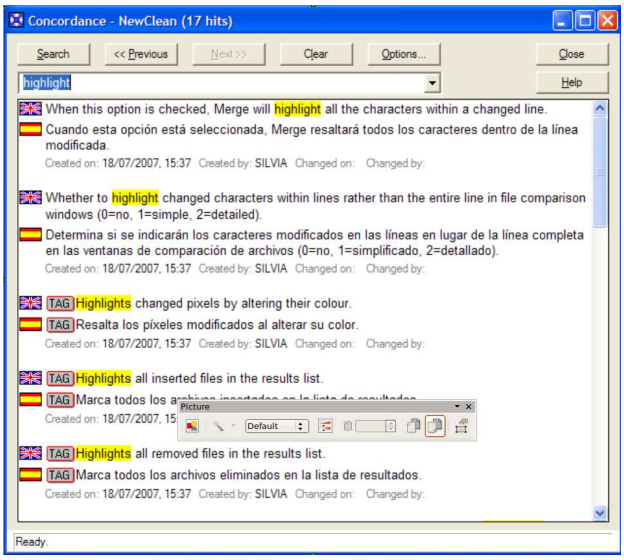


Figura 5: Memoria de traducción Trados Workbench 6.5 con los segmentos guardados.

Las herramientas CAT y las memorias de traducción representan una ayuda valiosísima a la tarea del traductor o localizador, ya que permiten reutilizar los segmentos coincidentes que hayan sido traducidos previamente. Ello supone un considerable ahorro de tiempo además de contribuir a mantener la consistencia en la traducción de los términos más habituales específicos de un determinado proyecto.

Las memorias de traducción son una herramienta imprescindible

El proceso es simple. A grandes rasgos y generalizando un poco, ya que cada herramienta presenta peculiaridades específicas en su funcionamiento, se puede decir que funcionan del siguiente modo. El traductor abre el documento que va a traducir en la herramienta CAT (algunas usan su propia interfaz mientras que otras se integran e interactúan con procesadores de texto del tipo MS Word) y procede a escribir la traducción para cada segmento que aquélla abre. Cada vez que se cierra o se guarda un segmento, la herramienta de traducción almacena el texto en el idioma original con su traducción correspondiente en la memoria. A medida que se avanza en la traducción, la herramienta CAT junto con la memoria de traducción mostrará al traductor los segmentos coincidentes que se hayan traducido anteriormente, bien en un 100 % de concordancia o en porcentajes inferiores, para reutilizarlos. Asimismo, permiten realizar búsquedas específicas de términos frecuentes entre todos los segmentos traducidos almacenados en la memoria. La herramienta de traducción define cada segmento que se traduce de acuerdo con las definiciones proporcionadas por el usuario. La gran mayoría de ellas usan los signos ortográficos como marcadores de segmentos, es decir, puntos, puntos y comas, etc.

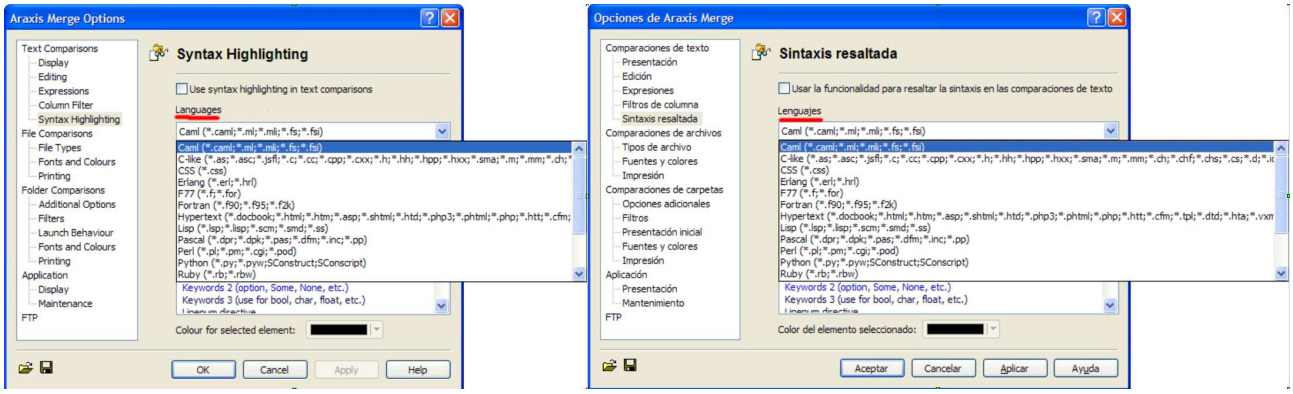


Figura 6. Ejemplo de traducción del término original "languages" en otro contexto diferente dentro de la misma aplicación.

Algunas de estas herramientas también ofrecen la posibilidad de procesar el documento para realizar una pretraducción usando los segmentos almacenados en la memoria de traducción. Esta pretraducción podrá hacerse también de acuerdo con los criterios de concordancia establecidos. Así podrán usarse concordancias del 100 % o en franjas de porcentajes inferiores.

Muchas también ofrecen la posibilidad de trabajar individualmente o conectados en red compartiendo los contenidos almacenados en las memorias entre varios traductores.

Como puede verse, las memorias de traducción suponen un excelente reaprovechamiento de recursos y garantizan un mayor grado de coherencia interna de la terminología. No obstante, no están exentas de riesgo. En relación a lo que decíamos anteriormente, una misma cadena puede ser utilizada en un contexto diferente con un significado totalmente distinto, tal como se ilustra en la Figura 6. Es evidente que la traducción del término "languages" aquí no podría ser nunca la misma que la que se mostraba en uno de los ejemplos anteriores, aunque ambos están extraídos de la misma aplicación.

Esto resulta particularmente relevante cuando, por ejemplo, varios traductores están trabajando compartiendo en red una misma memoria global de traducción para un mismo cliente. Mientras que unos están realizando la traducción de las cadenas de software, otros están trabajando en los materiales colaterales (manuales de usuario, manuales técnicos, marketing, etiquetado y empaquetado, intranet, etc.). Incluso en estos casos aunque existan concordancias del 100 %, la traducción no tiene por qué ser necesariamente igual.

Las memorias de traducción facilitan la coherencia interna de la terminología

Lo que es más, probablemente no deberían ser iguales puesto que, por ejemplo, las terminaciones de los verbos en -ing del inglés podrán traducirse como un infinitivo en las cadenas de código de la aplicación pero como un sustantivo o incluso como una frase fórmula del tipo "Cómo..." en manuales de usuario o materiales de marketing.

Una misma cadena puede ser utilizada en un contexto diferente con un significado totalmente diferente

Otros ejemplos de diferentes usos dependiendo del ámbito de uso:

Texto Origen	Traducción para Documentación	Traducción para Software
Printing a range and a Graph	Impresión de un intervalo y de un gráfico o Cómo imprimir un intervalo y un gráfico	Imprimir un intervalo y un gráfico
Delete Links	Supresión de enlaces o Cómo...	Suprimir enlaces
Apply symbols	Aplicación de los símbolos o Cómo...	Aplicar símbolos

Por ello, es esencial disponer de los medios (bien sea mediante la instalación de la aplicación o la colaboración con los programadores responsables o el personal de marketing) para garantizar que una cadena de texto determinada es apropiada en el contexto en el que va a aparecer.

Este aspecto es de particular relevancia ya que durante la fase de testing o de control de calidad existe un probabilidad muy alta de que pase desapercibido puesto que este tipo de tareas, por lo general, suelen estar desempeñadas por testers que no hablan el idioma o idiomas a los que se ha traducido la aplicación.

EMPRESAS LÍDERES DEL MERCADO

Evidentemente, dependiendo de la herramienta utilizada, la facilidad o dificultad para identificar y procesar estas cadenas variará de forma considerable. En la actualidad, podría decirse que el líder del mercado para la traducción general así como para la localización de software comercial es SDL Trados, seguido de Transit y DejaVu. En un ámbito más técnico se encontrarían Catalyst y Passolo. Como nota de interés, la mayoría de estas aplicaciones son desarrolladas por empresas alemanas.

	Atril Deja-Vu X	TRADOS 7	STAR Transit XV	SDL Translation Suite 2006
Ayuda al traductor				
Revisor ortográfico	Sí	Sí	Sí	Sí
Tesauro	Sí	Sí	No	No
Glosario	Sí	Sí	Sí	Sí
BD terminológica	Sí	Sí	Sí	Sí
Diccionarios	Sí	Sí	Sí	Sí
Interfaz de traducción propia	Sí	Sí	Sí	Sí
Alineación	Sí	Sí	Sí	Sí
Extracción de términos	No	Sí	No	No
Interfaz de TA	No	No	Sí	Sí
Tipos de archivo				
DLL, EXE	No	Sí	No	No
Excel	Sí	Sí	Sí	Sí
PowerPoint	Sí	Sí	Sí	Sí
RC	Sí	Sí	Sí	Sí
RTF (Win Help)	Sí	Sí	Sí	Sí
RTF (Texto)	Sí	Sí	Sí	Sí
TXT	Sí	No	Sí	Sí
DOC	Sí	Sí	Sí	Sí
Framemaker	Sí	Sí	Sí	Sí
PageMaker	No	Sí	Sí	Sí
HTML	Sí	Sí	Sí	Sí
Reutilización				
Coincidencias parciales	Sí	Sí	Sí	Sí
Coincidencias idénticas	Sí	Sí	Sí	Sí
De mismo tipo de objeto	Sí	No	No	No
Basado en frases	Sí	Sí	Sí	Sí
Basado en párrafos	Sí	Sí	Sí	Sí

Fuente: www.traductores.org.ar

Por supuesto, en el ámbito de software libre también existen aplicaciones equivalentes como, por ejemplo, OmegaT. Así mismo y aunque no encuadradas bajo el grupo de herramientas CAT también existen bibliotecas de software del tipo de gettext que facilitan la tarea de localización de aplicaciones.

RETOS DEL LOCALIZADOR

La labor del localizador de software es silenciosa pero omnipresente en la actualidad. En su trabajo diario se enfrenta a numerosos retos y dificultades inherentes a la naturaleza de este trabajo. Puesto que el lenguaje evoluciona a una velocidad vertiginosa particularmente en el ámbito de la informática mediante la introducción de neologismos, el traductor/localizador debe tener una clara conciencia de su labor como lingüista además de como experto en otros aspectos más especializados del proceso de localización. Por ello, para ser un buen localizador no basta con ser un buen traductor. Además de ser un maestro del lenguaje también hay que poseer excelentes aptitudes técnicas y un gran entusiasmo, porque localización es el arte de la traducción convertido en ciencia.

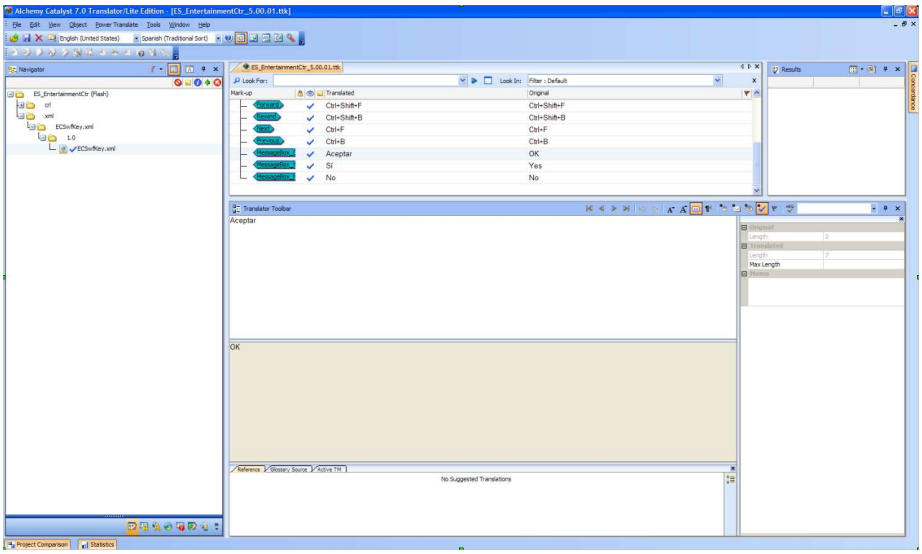
Y eso es algo que el localizador debe aprender mediante una sólida formación académica y amplia experiencia en el ámbito laboral, ya que no es algo que necesariamente venga predeterminado (del inglés *by default* que no por defecto!!...)

Sobre la Autora

Silvia Carril Caldelas es licenciada en Traducción e Interpretación por la Universidad de Vigo. Desde 1997 trabaja como especialista en localización de software para empresas tanto del Reino Unido como de España. Actualmente es la directora de ilia, compañía británica especializada en la localización y desarrollo de software.

`silvia@ilialang.com // www.ilialang.com`

Fig. 8: Volcado de pantalla de Catalyst 7.0



Introducción a la Simulación Acústica

Aprendemos a utilizar ODEON

por Francisco Miguel Bellas Aláez

Cuando se oye hablar de realidad virtual uno siempre piensa en mundo 3D creados mediante programas de renderizado. Sin embargo, también existe la virtualización sonora, es decir, crear entornos virtuales que transformen un sonido de una forma determinada. En este artículo voy a presentar uno de estos programas y, al mismo tiempo, voy a ir dando unas nociones básicas sobre análisis y diseño acústico de salas.

Podremos medir y comprobar mediante números y gráficas si una sala responde como nosotros queremos pero también podremos probar dicha sala escuchando un sonido, una conversación o una melodía como si estuviéramos en dicha sala.
Usaremos a lo largo del artículo el programa de análisis acústico Odeon que, aunque no es gratuito, si dispone de una versión demostración que permite trabajar con algunas salas predefinidas, lo cual servirá para asimilar los conceptos básicos del diseño y análisis de salas. Esta guía está basada en las prácticas de uno de los laboratorios de acústica de la ETSIT de Vigo.

INSTALACIÓN DE ODEON

Odeon es un programa de simulación acústica de salas que nos ayuda tanto al diseño de nuevas salas como al análisis acústico de salas ya construidas. En su versión completa, este programa permite crear salas en 3D incluyendo los materiales de que está compuesta para luego someter dicha sala a una serie de algoritmos de simulación acústica que nos permiten medir los principales parámetros de calidad acústica de salas. En este artículo trabajaré con la versión limitada (demostración) que permite sólo trabajar con las salas predefinidas.

Odeon ha sido creado por el Departamento de Tecnología Acústica de la Universidad Técnica de Dinamarca

Odeon ha sido creado por el Departamento de Tecnología Acústica de la Universidad Técnica de Dinamarca y puede ser descargado de la web:

`http://www.odeon.dk/`

en el apartado downloads. En el momento de escribir este artículo dicha página ofrece la versión 9.0 del programa. Os lo podéis bajar en un único fichero zip para Windows. Si tenéis Linux podéis probar a instalar en alguna máquina virtual tipo VirtualBox. Parece que se puede ejecutar directamente sobre wine, aunque no hemos probado todas sus funcionalidades.
La instalación es realmente simple. Desomprimís el zip en una carpeta y ejecutáis el programa setup.exe. Prácticamente la única opción que hay que seleccionar es vuestra carpeta destino que, curiosamente y al revés que en otros programas, no se instala en

```
C:\Archivos de programa\
```

si no directamente en el raíz del disco duro fuente. Cambiadlo a vuestro gusto y punto.

PARÁMETROS ACÚSTICOS DE INTERÉS

Existen diversas teorías acústicas que ayudan a obtener distintos parámetros de calidad de una sala. La teoría estadística permite mediante modelado probabilístico del campo acústico obtener la energía acústica en distintas partes de una sala y, por tanto, nos permite obtener niveles de reverberación. La teoría geométrica modela la propagación sonora mediante el trazado de rayos de forma similar a como se trabaja en óptica. Finalmente la teoría ondulatoria obtiene la ecuación de onda en la sala y obtiene la respuesta en frecuencia de la misma. Veamos algunos de los parámetros principales que manejan estas teorías:

- **Coeficiente de absorción:** Relación entre la energía incidente y reflejada en una superficie.
- **Tiempo de Reverberación (TR):** Se define como el tiempo en que tarda la energía acústica en atenuarse 60 dB una vez que cesa la emisión de la fuente (la millonésima parte en unidades naturales). Si en un recinto los materiales son acústicamente muy reflectantes este tiempo será mayor (se percibirá efecto de un eco muy corto) y si son absorbentes será menor. Dependiendo del destino de la sala, un TR muy bajo puede ser bueno para que se entienda mejor a un orador, pero puede darle mucha "frialdad" a una sinfonía.
- **Área de absorción equivalente:** Ponderación entre las diversas superficies que tiene una sala según su absorción y establecer un equivalente mediante una única superficie de determinada área que tenga la misma absorción.

COMENZANDO CON ODEON

A continuación vamos a familiarizarnos con el programa Odeon para realizar luego una serie de ejemplos guiados en los cuales quedará más claro los conceptos explicados previamente. Tras abrir el entorno de trabajo nos encontramos con un formulario con sus menús, iconos y un fondo de pantalla absolutamente horrible y que puede parpadear de forma muy incómoda dependiendo de la frecuencia de refresco de nuestro monitor.

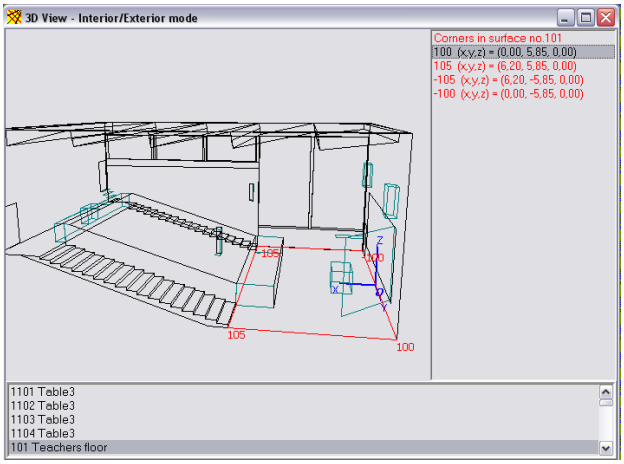


Figura 1. 3D View

Abramos a continuación una de las salas que vienen predefinidas (recordemos que en la versión demo sólo se puede trabajar con estas salas) entrando en el menú File->Open Room. Escogemos por ejemplo la sala auditorium 21 (los archivos son los de extensión .par). Se abre una ventana denominada 3D View en la que aparece el modelo 3D de dicha sala que podemos girar y colocar a nuestro aire mediante el ratón. También pulsando la barra espaciadora podemos cambiar ciertas vistas predeterminadas (planta, alzado, etc.)
Aparece también un nuevo menú, el 3D View que permite realizar distintas tareas con la ventana de representación 3D: Podemos ampliar o reducir con la opción zoom; mediante Highligh Surfaces se nos muestra una lista de superficies en la parte inferior. Pulsando en alguna de ellas la resalta en el dibujo; mediante Show Corner numbers and coor se pueden ver las coordenadas de las esquinas de la superficie seleccionada y se resalta la esquina seleccionada con un círculo azul.

La ventana 3D View nos muestra nuestra sala en 3D

Os recomiendo jugar con las distintas opciones de este menú que afectan exclusivamente a la visualización de la sala. También os recomiendo abrir distintas salas y ver las características de las mismas.
En el menú Toolbar hay algunas opciones más que son interesantes como primera toma de contacto. Una de ellas es Room Information que nos da datos sobre las dimensiones de la habitación. Otra es la vista 3D OpenGL que nos permite echar un vistazo a la sala como si estuviéramos dentro de ella. También es interesante la opción 3D Reflector Coverage que nos muestra mediante código de colores que zonas de la sala son reforzadas por los reflectores para distintas fuentes (sources). Hay más opciones pero que introduciremos en sucesivos apartados ya que es preferible verlas definiendo nosotros los parámetros que podamos y no limitándonos a visualizar algo ya hecho. De todas formas podéis probar la 3D Billiard o la 3D Investigate Rays que son simulaciones bastante espectaculares de la evolución de una onda o de rayos sonoros. Finalmente quedaros con la sala Example ya que es muy sencilla y para aprender el manejo de Odeon y los conceptos de acústica es la más clara. Luego experimentaremos con alguna sala más compleja.

DEFINICIÓN DE MATERIALES USADOS

Aunque la versión demo de Odeon no deje modificar la sala, sí permite cambiar los materiales de la misma para hacer distintas pruebas.
Si tenéis abierta la sala Example, podéis pinchar la opción del menú Toolbar->Material List y se abrirán dos ventanas. En la superior aparece la representación de la sala y en la inferior la lista de superficies a la izquierda y la librería de materiales disponibles en el programa a la derecha.

Podremos modidicar los materiales de nuestra sala

Pinchando sobre cualquier material de la lista de la derecha, aparece en la parte inferior su repuesta en frecuencia (en referencia al coeficiente de absorción) en las bandas de octava 63, 125, 250, 500, 1000, 2000, 4000 y 8000 Hz que son estándar en medidas acústicas.

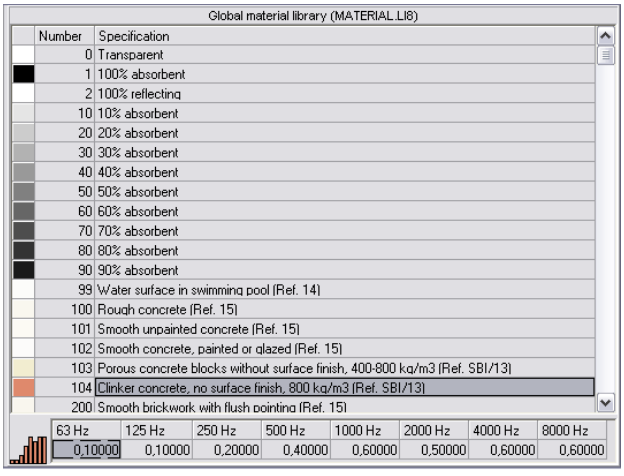


Figura 2. Lista de Materiales

También mediante un gráfico de barras pequeño a la izquierda se puede intuir de forma visual la respuesta en frecuencia del material.
Los primeros materiales de la lista son ideales. Es decir, son materiales con un porcentaje de absorción determinado en todas las frecuencias. Los materiales reales aparecen a partir del 99.
La parte interesante de esto es la posibilidad de asignarle los materiales que nos apetezca a las distintas superficies de la sala. Para ello lo primero que se hace es seleccionar el material en un lado y la superficie en el otro. Luego se pulsa en el icono de Asignación de Material que es el primero que aparece en la columna que separa la lista de materiales de la lista de superficies (en la misma ventana Material List)

Figura 3. Icono de Asignación de Materiales

Por ejemplo podemos tomar la superficie 1001 que es donde estaría la orquesta y asignarle el material 900 que es un material que simula la absorción de una orquesta con instrumentos. Luego seleccionamos la 1002, que es la audiencia y le podemos asignar uno de los valores entre el 903 y el 909 que son distintos tipos de zonas audiencia con sillas de distintos materiales. Por ejemplo el 903. Al resto de las superficies (techos y paredes) le podemos asignar un material poco absorbente como el 700.
El parámetro scatter indica la difusión en la reflexión del sonido. Poned todos a 0,1 salvo los materiales 1001, 1002 y 2004 que les podéis aumentar la difusión a 0,7.

FUENTES Y RECEPTORES

Una vez que tenemos definida la sala, para realizar algún tipo de simulación es necesario especificar dónde colocamos las fuentes sonoras y dónde se desea colocar el receptor o los receptores.
Seleccionamos del menú Toolbar la opción Source Receiver List. Se nos abre una ventana donde podemos especificar fuentes (sources) y receptores (receivers). Pulsando la letra P o en el icono New Point Source (similar a una diana roja) añadimos fuentes y pulsando R o sobre el icono del micrófono (New Receiver) añadimos receptores.


New Point Source


New Receiver

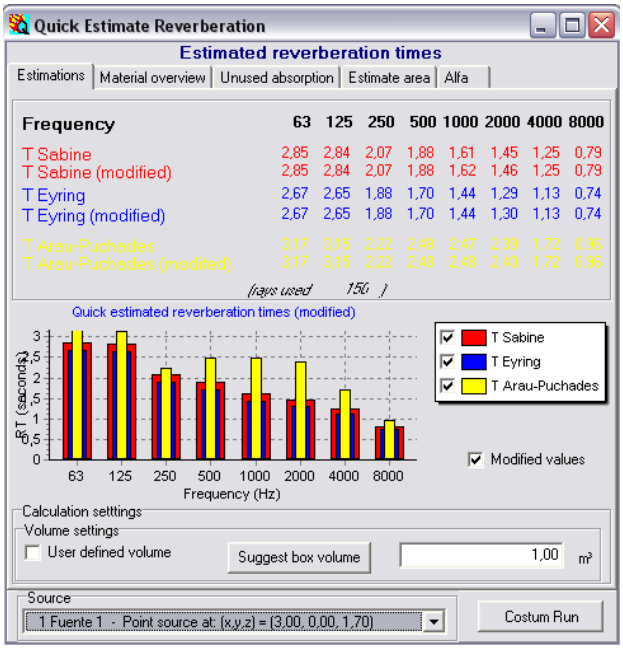
Pulsamos primero la P y se nos abre una ventana donde podemos introducir diversos parámetros. La descripción es el nombre que le queramos dar a la fuente. En las coordenadas colocaremos

$x = 3$ $y = 0$ $z = 1,7$
(usa la coma para los decimales)

A medida que modificamos las coordenadas podemos ver la fuente como un círculo rosa en la ventana 3DView. Mediante Azimut, elevación y rotación podemos indicar hacia dónde apunta la fuente. En principio podemos dejarlo a cero (En 3DView viene indicado por una línea que sale del círculo).
En la parte inferior indicamos una ganancia (Overall gain) de 65 dB que es la potencia de emisión. El apartado EQ es por si se desea ecualizar la fuente para darle cierta "forma en frecuencia".
Cerramos la ventana. Sí, en el icono X de la propia ventana. No hay botón aceptar así que no lo busquéis ;). Nos aparece la fuente creada en la lista.
Para definir receptores pulsamos la tecla R. Aparece una ventana similar pero más sencilla. Sólo se indica coordenada. Aparece también un icono de Open GL ya que una vez colocado el receptor podríamos comprobar la vista que tiene de la sala desde la posición indicada.
Crear dos receptores, uno en $x = 8, y = -5, z = 1,5$ y el otro en $x = 20, y = 2, z = 4$.

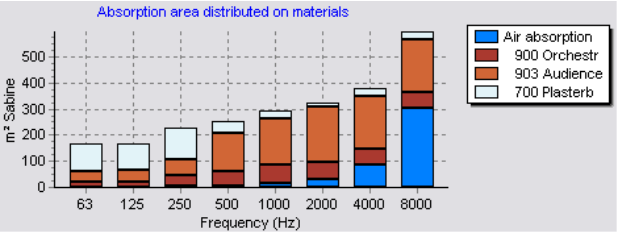
CÁLCULOS DE TIEMPO DE REVERBERACIÓN

Lo primero que haremos será hacer una estimación del tiempo de reverberación mediante teoría estadística. Para ello tenéis que abrir primero la Material list y luego seleccionar la opción Quick Estimate en el menú Materials.
Nos aparece un gráfico (pestaña Estimations) que nos muestra el tiempo de reverberación de la sala (RT) en segundos para distintas frecuencias según 3 métodos distintos: ecuación de Eyring, de Sabine y de Arau-Puchades. Estos métodos cambian en cuanto a precisión y complejidad de cálculo.



En la pestaña Material Overview podemos ver la absorción de cada uno de los materiales según la frecuencia.

Esto es muy útil ya que nos puede ayudar a ajustar los materiales de la sala si notamos que hay excesiva absorción o demasiada reflexión para algunas frecuencias. Cabe destacar también que el aire realmente afecta mucho en altas frecuencias y las atenúa por absorción. La pérdida de altas frecuencias provoca una pérdida en el "brillo" de la música.



Podemos ver también que las audiencia y la orquesta (900 y 903) apenas varían con la frecuencia (salvo en muy bajas). El 700 absorbe realmente poco salvo las frecuencias más bajas.
La medida está en m2 porque se usa el área de absorción equivalente para dar una medida de lo que absorbe cierto material.
La siguiente pestaña, Unused absortion, indica la absorción que podríamos añadir a la sala si sustituimos el material indicado por un material ideal totalmente absorbente.

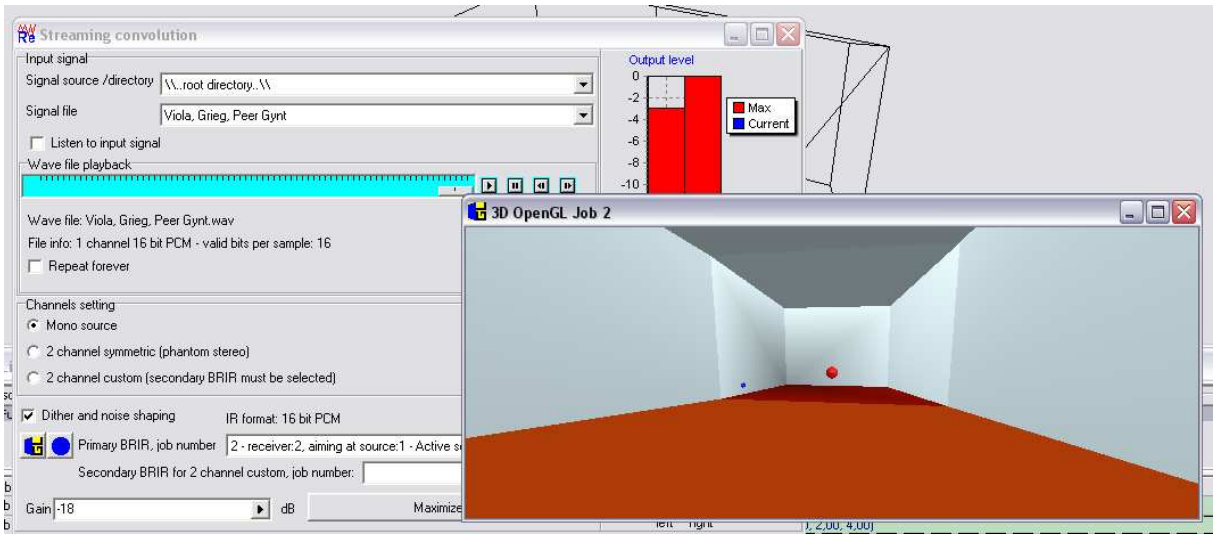
¿Y COMO SUENA LA SALA?

Las simulaciones y los cálculos para mejorar una sala están muy bien, pero en el fondo lo que nos gustaría es saber cómo suena. Para eso en Odeon existe la Auralización que crea un entorno sonoro virtual que simula el sonido en la sala. Es decir, que seleccionando un receptor podemos saber como se escucharía cierto sonido en dicha sala.

Sin entrar en detalles, para que el programa pueda hacer los cálculos hay que establecer una respuesta al impulso de la sala lo suficientemente larga. Para este caso vamos a ponerla de 2000ms. En Toolbar seleccionamos Calculation Parameters y cambiamos sólo Impulse Response (en la parte inferior).
Seleccionamos en Toolbar la opción Job List. Aquí se definen los trabajos de auralización. Seleccionamos el primero y activamos la fuente (justo encima de la lista de trabajos). Luego en Receiver Pointing to de Source seleccionamos la única fuente (si alguien cerró más pues que elija). En Single Point Response Receiver escogemos el primer receptor. Luego creamos un segundo trabajo igual pero seleccionando el segundo receptor.

Odeon nos permite simular como sonará nuestra sala

Ahora le mandamos realizar los cálculos necesarios para crear el sistema que realiza la respuesta al impulso de la sala. Para ello apretamos el icono Run All Jobs que lo tenéis tanto en los iconos de la derecha como en el menú Job Lists.
Una vez hechos los cálculos, si no hay errores los trabajos aparecerán en verde. Para escuchar el sonido de la sala se debe pulsar en la opción Streaming Convolution del menú Job Lists (también los tenéis en los iconos de la derecha).
Os sale una nueva ventana que permite en la parte superior seleccionar la señal de entrada. Por defecto trae 3 señales. Una de aplausos (bastante parecido a un impulso o delta), otra de música y otra de voz. Si marcáis Listen to input signal y le dais a play escucharéis el fichero original. Si lo desmarcáis escucharéis como se oiría ese sonido en esa sala en la situación de uno de los receptores (el que tengáis seleccionado).



Simulación con Odeon

En la parte inferior se puede seleccionar el otro trabajo, se puede disminuir la ganancia si el volumen está muy alto (o aumentarla en caso contrario) y se puede ver la sala desde la posición del receptor como si estuviéramos en la sala además de escuchándola (eso sí, con gráficos bastante pobres).
Ahora podéis probar a cambiar materiales o fuentes en la sala. Por ejemplo podéis coger en uno entre el 907 y el 909 para la zona de audiencia que simularía la sala con gente. O podéis probar a meter un material más absorbente en la pared posterior para comprobar como pierde fuerza el oyente trasero.
Incluso os recomiendo probar otras salas de las de demostración que las hay muy interesantes. Podéis ver si están más enfocadas a música o a oratoria... o incluso intentar mejorarlas para una de las dos funciones.

¿Y QUE MÁS PUEDO HACER?

Para abrir un poco más el apetito acústico vamos a realizar un ultimo experimento. Abre una de las salas predefinidas distinta a la Example, por ejemplo el Hagia Irene. Abre la Job List y aparecerán ya un montón de trabajos predefinidos y en algunas salas como esta, ya calculados (en verde). Además de escuchar como hicimos antes, podéis pulsar en la opción View Grid Response del menú Job List. Aparece una maya de colores sobre el edificio que representan los niveles de algún parámetro. Estos parámetros se cambian con los cursores hacia arriba y abajo y en frecuencia con los cursores izquierda y derecha. Algunos parámetros de fácil explicación son:

- **SPL:** Potencia sonora
- **T30:** una medida del tiempo de reverberación (distinta a la estadística que vimos)
- **LF:** Eficiencia Lateral. Es una relación entre el sonido que viene frontalmente (directo) y lateralmente (normalmente reflexiones).
- **STI:** Indice de transmisión del habla. Indica si se entiende bien al oyente. Puede tomar valores de 0 (no se entiende nada) a 1 (se entiende de maravilla). Las reflexiones estropean el parámetro.

Y con todos estos parámetros seguramente os preguntaréis: "Muy bien, muy bien, pero yo quiero diseñar una sala fabulosa ¿Cómo tienen que estar esos parámetros?" Bueno, como buen gallego la respuesta es "Depende ¿Para que la vas a usar?". No se si alguno de los lectores se habrá fijado pero en muchas ocasiones cuando va a una conferencia en una sala determinada la oratoria se oye relativamente mal, es decir, las frases del orador no se entienden demasiado bien y, sin embargo, si habéis escuchado música en dicha sala es posible que os haya gustado como suena.

Este problema está relacionado con la reverberación del sonido y es complicado conseguir una "sala perfecta" para todo tipo de uso. Cuanta más reverberación tenga una sala, peor es la calidad de dicha sala para la oratoria. Parámetros relacionados con la inteligibilidad (comprensión) de la palabra como el STI caen de forma estrepitosa cuanto mayores sean las reflexiones. Es decir, se entiende mal a un orador.
Esto es lógico si pensamos que el eco es una forma retardada del sonido, entonces si un orador dice dos palabras seguidas como "! Buenos días!" y resulta que en la sala en la que estamos escuchando hay un cierto nivel de eco, el "Buenos" lo oiremos bien, pero resulta que al mismo tiempo que oímos la versión directa del "días" nos llega la versión retardada del "buenos" que será más molesto y hará que se entienda peor cuanto más potente sea dicho eco. Si esto lo extrapolamos a una charla completa, puede llegar a no entenderse nada de lo que se dice.
Sin embargo, si disminuimos mucho la reflexión de las superficies de la sala y tratamos de dar algún tipo de concierto musical en la misma vamos a notar que el sonido está "apagado" sin fuerza. Un cierto nivel de reverberación para la música (ojo, tampoco debe ser excesiva) va a provocar un sonido más envolvente, con mayor "calidez". De ahí que sea complicado realizar una sala multiusos.
Entonces ¿cómo podemos mejorar una sala o simplemente una habitación? Pues lo más sencillo y económico normalmente es el cambio de materiales en las distintas superficies. ¡No! No hace falta que tiréis las paredes. Se puede disminuir los ecos en una sala con mucha rever metiendo unas buenas cortinas en lugares estratégicos. O si me quiero montar un estudio casero puedo utilizar hueveras para cubrir las paredes ya que simulan (aunque remotamente) a una sala anecoica (que absorbe muy bien los distintos ecos de los instrumentos usados). Además, cuando se realza una grabación es muy fácil introducir digitalmente los distintos efectos de rever que puedan interesar para un acabado "ideal".
Si por el contrario lo que quiero es aumentar dicha reverberación ya dependemos un poco más de cómo está construida y de qué materiales consta. Quizá llegue con sacar esas cortinas del fondo y dejar los cristales a la vista ya que el sonido lo refleja muchísimo más un cristal que una cortina (y además las cortinas eran horrorosas, así que dos pájaros de un tiro). O quitar algunos elementos de la sala si la tenemos recargada con muchos muebles o recovecos por los que se pueda perder el sonido.
Se podría escribir un libro más que un artículo si quisiéramos explorar todo lo que permite este programa. Para empezar habría que adentrarse en la acústica y definir muchos más parámetros y conceptos para luego ponerlos en práctica sobre el programa. Quién sabe, quizá en un futuro artículo avancemos un poco más.

BIBLIOGRAFÍA

Ingeniería Acústica. M. Sobreira, E. Alexandre. Servicio de Publicaciones de Teleco Vigo. 2003.
Apuntes de Laboratorio de Acústica Arquitectónica. Sin editar.
http://www.wikipedia.org (Versión inglesa y española) y http://www.odeon.dk/

El telégrafo de Gauss

Una historia de Investigación Histórica

por Fernando Martín y Gonzalo Barrio
(fmartin@tsc.uvigo.es)

Si recordáis el primer artículo que escribí para Occam's Razor, se titulaba "Mi Historia de las Telecomunicaciones". En aquel texto ya se mencionaba que el primer sistema de telecomunicación de la historia fue el telégrafo construido por los físicos C. F. Gauss y W. Weber en 1822.

Este artículo pretende describir aquel primitivo sistema esperando que su comprensión os resulte tan interesante como a mí. Debo comentar que el interés por el tema procede del señor Gonzalo Barrio (licenciado en física y empleado de la Fundación Empresa Universidad Gallega, Feuga, www.feuga.es). Gonzalo está realizando su tesis doctoral en un programa de la Universidad de Santiago llamado "Historia de la Ciencia y la Tecnología" y decidió encaminar sus esfuerzos a investigar la labor del gran genio alemán. Una parte de ese trabajo es la que describe este artículo: el estudio y descripción del funcionamiento del telégrafo de Gauss y Weber. Cuando Gonzalo me pidió que colaborara en su trabajo, me sentí halagado por su confianza en que encontraríamos la solución pero ante la poca información técnica disponible yo no pensaba que acabaríamos resolviendo el problema.

Nuestra primera fuente de información sobre el telégrafo de Gauss fue esta foto (procedente del museo dedicado a Gauss en Gottingen, la ciudad donde trabajó Gauss durante más tiempo y donde murió en 1855).

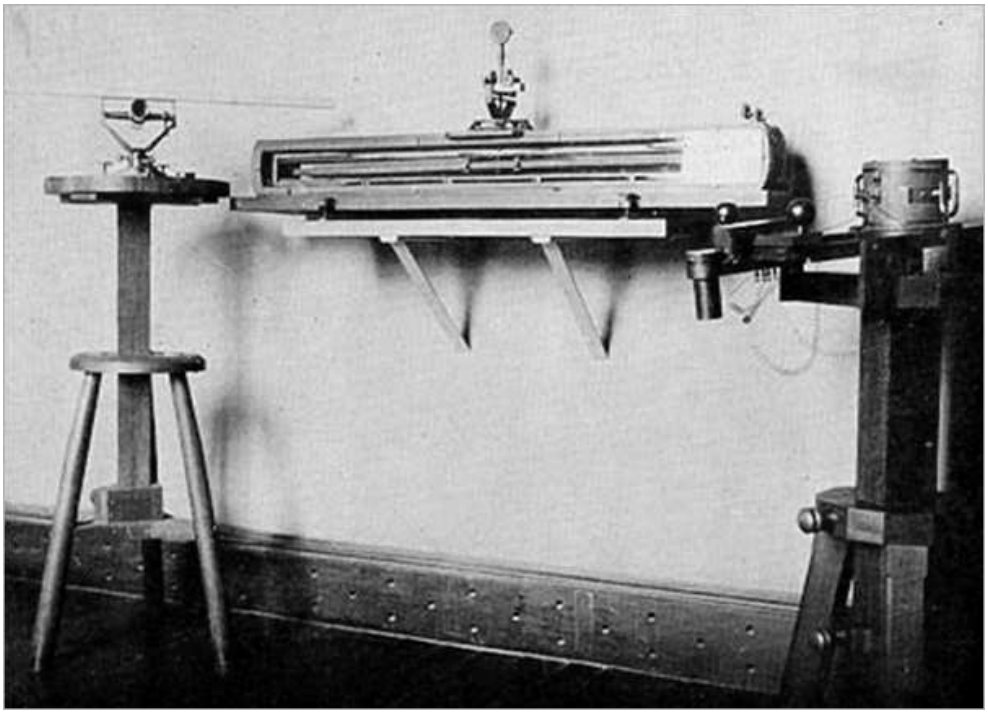


Figura 1. Telégrafo de Gauss

Desde luego sólo con esta foto era imposible saber nada. Después logramos más información: más fotos procedentes del mismo museo, una carta de Gauss comentando sus pruebas al astrónomo Olbers y también el CD "Genial Gauss Gottingen", editado en Alemania para conmemorar el 250 aniversario de su muerte.

EL TELÉGRAFO DE MORSE

Nunca hubiéramos entendido el telégrafo de Gauss sin haber examinado primero el de Morse (posterior). El siguiente esquema pertenece a un telégrafo de la época de Morse y es mucho más comprensible porque ya es un esquema eléctrico:

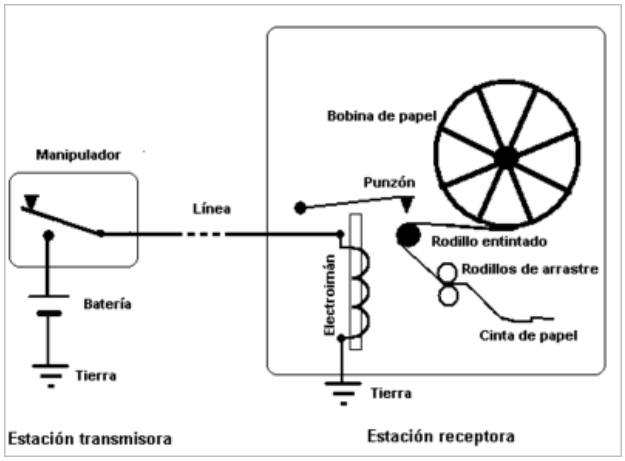


Figura 2. El telégrafo de Morse

Vemos que el emisor se basa en una batería y un pulsador. El negativo está conectado permanentemente a tierra y el pulsador permite aplicar tensión o no a la línea. Si pulsamos durante un segundo tendremos un pulso de tensión en la línea... una señal así:

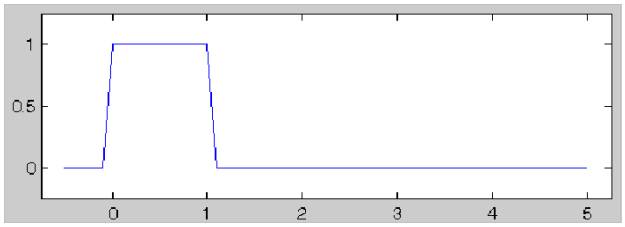


Figura 3. Pulso de tensión generado por una pila y un conmutador.

Ese pulso va a viajar hasta el otro lado (porque el hilo metálico y la tierra forman una línea de transmisión. Una vez que el pulso llega a la bobina del electroimán se generará una fuerza magnética que atraerá a la pieza metálica que sostiene el punzón. Eso hará que se escriba sobre el papel durante el tiempo que se ha apretado el pulsador. Como el rollo de papel está continuamente en movimiento, si pulsamos sólo un instante el receptor obtiene un punto (se ha transmitido un pulso corto). Si pulsamos durante un poco más de tiempo se transmitirá un pulso más largo y el receptor obtendrá una raya. Con eso ya se ha conseguido una comunicación puesto que somos capaces de enviar dos símbolos diferentes. El telégrafo de Morse usa, por tanto, una modulación temporal (la información va en la duración de las señales).

Además, como sabréis, Morse desarrolló un código prefijo óptimo con 26 símbolos (las letras del alfabeto). Se trata de un código de longitud variable más evolucionado que los de longitud fija (Gauss y Weber usaban palabras código de 5 bits para un alfabeto de 32 mensajes posibles). Para poder decodificar un código de longitud variable se debe cumplir que ninguna palabra código sea prefijo de otra (código prefijo). Además, es un código de entropía mínima (las palabras código más cortas corresponden a los símbolos más probables logrando una longitud media óptima desde un punto de vista estadístico). **El método matemático para crear códigos prefijo óptimos no se conoció hasta que lo desarrolló Huffman en 1952. ¡Más de cien años después!**

Wilhem Weber (1804-1891) fue un físico alemán recordado por sus trabajos sobre el campo magnético. De hecho, hoy en día a la unidad de flujo magnético se le llama Weber. Weber llegó a ser profesor de física en Gottingen con sólo 27 años. Gauss comenzó a colaborar con él al ver que era un científico de su nivel y juntos desarrollaron varios trabajos. La capacidad de Gauss para idear nuevos artefactos se complementaba muy bien con la capacidad práctica de Weber para crearlos y probarlos.

El primer mensaje telegráfico de la historia lo recibió Weber y decía textualmente: `Michelman Kommt` ("Michelman viene" en alemán). En él Gauss informaba a Weber de que su común ayudante Michelman había salido del observatorio (donde trabajaba Gauss) en dirección a la facultad de física (lugar de trabajo de Weber).

EL TELÉGRAFO DE GAUSS

A partir del telégrafo de Morse y de la información disponible dedujimos que el telégrafo de Gauss respondía a un esquema eléctrico como éste:

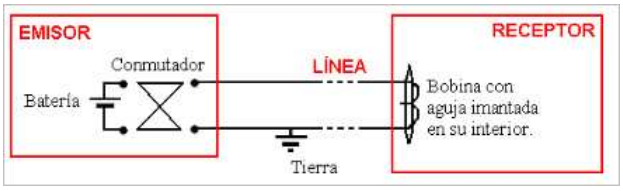


Figura 4. Esquema Eléctrico del Telégrafo de Gauss.

El emisor consiste en algún tipo de generador o acumulador de energía (Gauss en su carta a Olbers le llama "pila voltaica"). De hecho, en las fotos se ve un objeto cilíndrico con aspecto de primitivo condensador que podría ser usado para acumular la tensión que va a generar los pulsos (podría ser un dispositivo muy similar a la botella de Leyden que ya se conocía desde 1746, podéis ver una descripción sencilla en: `http://es.wikipedia.org/wiki/Botella_de_Leyden`)

Figura 5. Emisor.

El código de Gauss y Weber se basaba en mover la aguja receptora (horizontal aunque en el esquema aparece vertical) a la izquierda o a la derecha, de modo que la modulación de la señal se va a basar en su signo (es una modulación en amplitud y no temporal como la del telégrafo de Morse). De esta forma, la conexión de la pila emisora a la línea tenía que consistir en un interruptor relativamente complejo (Gauss le llama "conmutador") que permitiera tres estados:

- **Neutro**: el operador no hace nada los dos bornes de la pila están abiertos.
- **Enviando pulso positivo**: el operador ordena mover la aguja receptora a la derecha el borne positivo se une a la línea y el negativo a tierra. Se envía un pulso positivo (su duración no es importante pero será el tiempo que el operador mantiene el pulsador en esta posición):

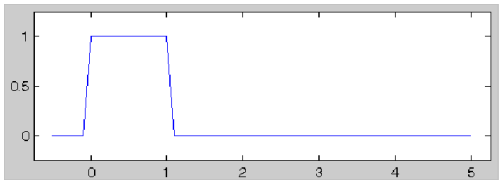


Figura 6. Pulso Positivo.

- **Enviando pulso negativo**: el operador ordena mover la aguja receptora a la izquierda el borne positivo se une a tierra y el negativo a la línea. Se envía un pulso opuesto al de Morse:

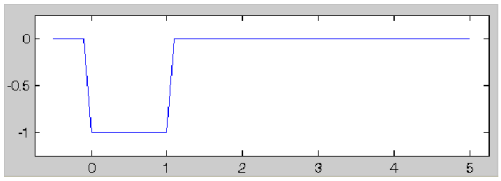


Figura 7. Pulso Negativo.

Nótese que pulsos opuestos generarán campos magnéticos también opuestos y movimientos de la aguja hacia un lado o a otro. En el CD se dispone de figuras de un manuscrito que detalla el código original y se ve que lo que hoy llamaríamos bits aparece codificado con los signos "+" y "-" en vez de "1" y "0".

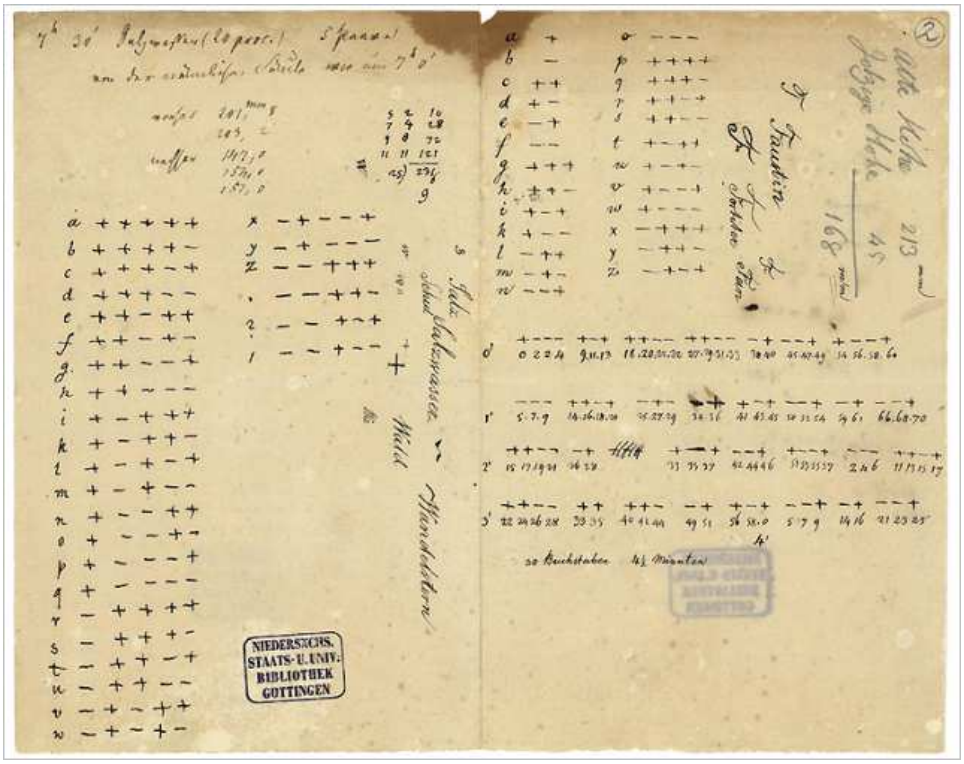


Figura 8. Código Original.

El receptor posee una bobina y una aguja imantada. Gauss le llama galvanómetro y, de hecho, es un dispositivo similar a los medidores de D'Arsonval (desarrollados por Jacques-Arsène d'Arsonval en el siglo XIX) con disposición de la aguja horizontal en vez de estar montada sobre un cuerpo giratorio (otra diferencia es que en el medidor de D'Arsonval el imán es fijo y la bobina es móvil).

Figura 9. Bobina receptora (izquierda). Anteojo usado para ver el movimiento de la aguja.

En este telégrafo era necesario un anteojo para ver las desviaciones de la aguja que debían de ser muy débiles. Lo que no es de extrañar si se usaba un condensador como fuente de energía: se generaba un pulso no muy potente que debía atravesar más de dos kilómetros de un cable que seguramente tenía una atenuación por metro muy alta (existe un dibujo del tendido inicial realizado sobre los tejados, esta instalación unía el laboratorio de Gauss con el de Weber). Otra razón es el gran peso de la aguja (Gauss habla de un imán de una libra) y el hecho de si se pudo incluir o no un núcleo de hierro en el interior de la bobina. De incluirse, el campo magnético es mucho mayor, Gauss habla de una bobina de 50 vueltas y no dice nada sobre un núcleo, incluirlo parece difícil si tiene que haber una aguja dentro.

CONCLUSIONES

Como conclusión final pensad que podemos establecer a Gauss y Weber como los padres de las telecomunicaciones. Su prototipo no fue muy tenido en cuenta en su país, a diferencia del de Morse que logró el interés del gobierno norteamericano de la época. Después vinieron otros muchos sistemas (ved, por ejemplo, el dossier sobre el tema en el primer número de Occam's Razor).

Como conclusiones más personales a ambos nos ha parecido apasionante escudriñar en la historia para intentar explicar el funcionamiento de viejos pero ingeniosos sistemas. Es una faceta investigadora un poco diferente a las habituales pero igualmente gratificante.

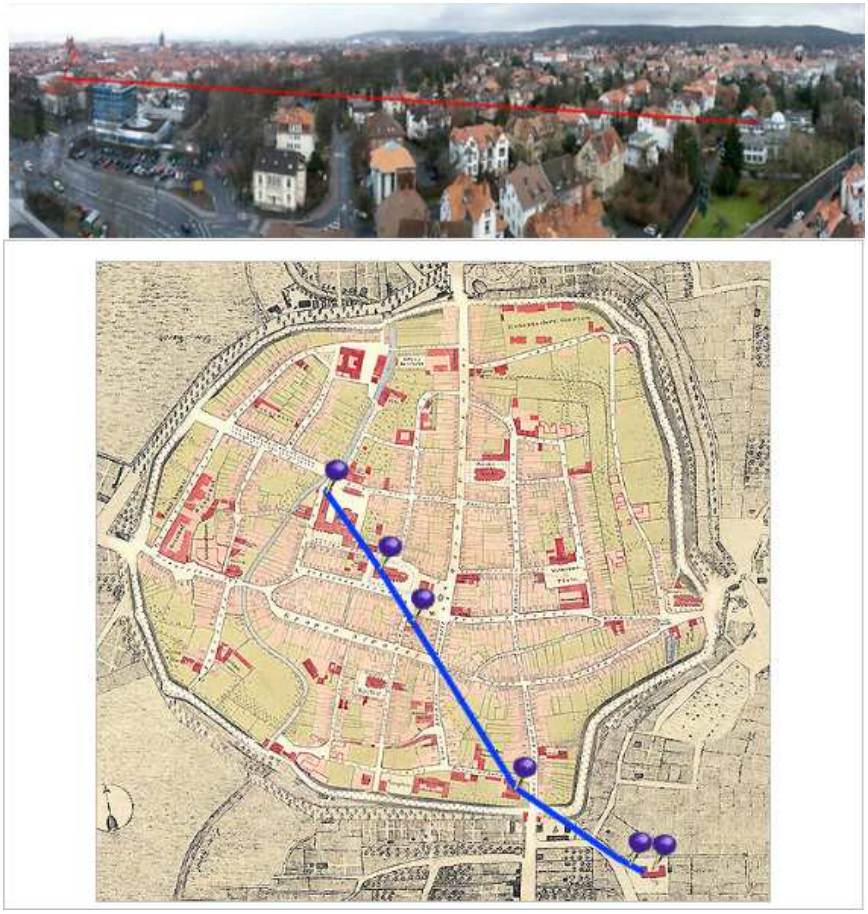


Figura 10. Tendido inicial. Arriba: puntos inicial y final (laboratorio de física de Weber a la izquierda, observatorio astronómico de Gauss a la derecha). Abajo: mapa de la época detallando el recorrido del cable.

Poniendo Contraseñas a Ejecutables

O como parchear el segmento .note.ABI-tag

por Er de la Sección

Cuando se dispone del código fuente de una determinada aplicación (como sucede en los sistemas libres), añadir una contraseña a cualquiera de ellos para no permitir su uso por desconocidos es sencillo. Pero cuando el código fuente no está, la cosa es un poco más... interesante.

En este artículo vamos a describir una sencilla técnica para parchear ejecutables. El ejemplo que utilizaremos, proteger con una contraseña, no es que sea especialmente útil, pero es una de las cosas más sencillas que podemos hacer para ilustrar el proceso.
Una vez sepáis como funciona todo esto, estamos seguros de que se os ocurrirán montones de ideas interesantes con las que divertiros parcheando ejecutables. Bien, vamos a ello.

EL FORMATO ELF

El formato ELF (Executable and Linkable Format o formato ejecutable y enlazable), es el formato utilizado por la mayoría de los sistema UNIX para sus programas, es decir, todo ejecutable en nuestro sistema se almacena en el disco utilizando este formato (bueno, podemos utilizar otros formatos, pero no es lo habitual).
Cuando compilamos nuestros programas a partir del código fuente, en el lenguaje que más rabia nos dé, iniciamos un proceso bastante complejo que termina cuando nuestro programa está cargado en memoria.
Sí, vale, tras compilar nuestro programa obtenemos un fichero ejecutable en el disco y parece que ahí se termina todo, sin embargo, el sistema todavía tiene que hacer unas cuantas cosas al cargarlo en memoria para que ese fichero se convierta en la imagen de un proceso en ejecución.

ELF es el formato ejecutable más común en los sistemas UNIX

Así, el fichero que almacenamos en disco es el resultado del proceso de enlazado (link), pero el proceso de carga para su ejecución, requiere que el sistema lleve a cabo ciertas operaciones como la relocalización de código o la carga de librerías dinámicas externas y su enlazado en tiempo de ejecución. Estas tareas las realiza el denominado enlazador dinámico (dynamic linker en inglés :).
Nosotros, en este artículo, vamos a trabajar sobre los ficheros ejecutables. Básicamente vamos a hacer de enlazadores, añadiendo código a un fichero ELF, y modificándolo para que, cuando el programa se cargue en memoria y el enlazador dinámico lleve a cabo su tarea, todo esté correcto.

CABECERA Y PUNTO DE ENTRADA

Como suele ser habitual, los formatos de fichero suelen tener una cabecera. Esta cabecera nos proporciona información para verificar que el fichero es de ese formato y para acceder al resto de información que el fichero contiene.
Pues bien, el formato ELF empieza con una pequeña cabecera.
Antes de continuar, permitirnos hacer un pequeño comentario. En este artículo no vamos a describir todos los detalles del formato ELF, solamente aquellos que tengan que ver con el tema que nos ocupa. Los que estéis interesados en conocer todos los detalles sobre ELF (merece la pena) deberéis descargaros la especificación de este formato, la cual encontréis fácilmente en Internet.
Tras este breve inciso, veamos la cabecera típica de un ejecutable ELF. Para ello vamos a utilizar la herramienta readelf, que nos permite visualizar la información interna de los ficheros de una forma sencilla.

```
occam@razor: $ readelf -h /bin/ls
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x8049b40
  Start of program headers:          52 (bytes into file)
  Start of section headers:          76752 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         8
  Size of section headers:           40 (bytes)
  Number of section headers:         27
  Section header string table index: 26
```

De todos esos campos en la cabecera del fichero a nosotros nos van a interesar dos de ellos. El primero, como os podéis imaginar es la *Dirección del Punto de Entrada* (Entry Point Address).
Este campo nos dice cual es la dirección de memoria en la que empezará la ejecución del programa, una vez haya sido cargado y procesado.

El segundo campo que nos va a interesar es el desplazamiento del *Comienzo de las cabeceras de programas* (Start of program headers). En seguida veremos que esto.

SECCIONES

El formato ELF, organiza toda la información que el sistema necesita para poder ejecutar un programa en dos estructuras principales: secciones y segmentos.
Las primeras, las secciones, realmente contienen pistas y ayudas para los cargadores de programas, depuradores, etc...
Veamos que apariencia tienen las secciones de nuestro querido `/bin/ls`:

```
occam@razor: $ readelf -S /bin/ls
There are 27 section headers, starting at offset 0x12bd0:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size
  [ 0]                   NULL            00000000 000000 000000
  [ 1] .interp           PROGBITS        08048134 000134 000013
  [ 2] .note.ABI-tag     NOTE            08048148 000148 000020
  [ 3] .hash             HASH            08048168 000168 000340
  [ 4] .dynsym           DYNSYM          080484a8 0004a8 0006d0
  [ 5] .dynstr           STRTAB          08048b78 000b78 0004bb
  [ 6] .gnu.version      VERSYM          08049034 001034 0000da
  [ 7] .gnu.version_r    VERNEED         08049110 001110 0000c0
  [ 8] .rel.dyn          REL             080491d0 0011d0 000028
  [ 9] .rel.plt          REL             080491f8 0011f8 000308
  [10] .init             PROGBITS        08049500 001500 000017
  [11] .plt              PROGBITS        08049518 001518 000620
  [12] .text             PROGBITS        08049b40 001b40 00cec8
  [13] .fini             PROGBITS        08056a08 00ea08 00001c
  [14] .rodata           PROGBITS        08056a40 00ea40 003c2f
  [15] .eh_frame_hdr     PROGBITS        0805a670 012670 00002c
  [16] .eh_frame         PROGBITS        0805a69c 01269c 0000ac
  [17] .ctors            PROGBITS        0805b748 012748 000008
  [18] .dtors            PROGBITS        0805b750 012750 000008
  [19] .jcr              PROGBITS        0805b758 012758 000004
  [20] .dynamic          DYNAMIC         0805b75c 01275c 0000e0
  [21] .got              PROGBITS        0805b83c 01283c 000008
  [22] .got.plt          PROGBITS        0805b844 012844 000190
  [23] .data             PROGBITS        0805b9e0 0129e0 00010c
  [24] .bss              NOBITS          0805bb00 012aec 000430
  [25] .gnu_debuglink    PROGBITS        00000000 012aec 000008
  [26] .shstrtab         STRTAB          00000000 012af4 0000db
```

Por cuestiones de espacio hemos eliminado las últimas columnas de la salida del comando, ya que no no nos son de utilidad. Aunque para parchear nuestro programa trabajaremos casi exclusivamente con los segmentos del programa, pero un par de comentarios sobre las secciones son necesarios... para que os pique la curiosidad.
Lo primero que hay que comentar es que las secciones tienen nombre, y espero que os hayáis fijado en el nombre de una de las primeras que os debería sonar, a no ser, claro, que seáis de esas personas que no leen los títulos de los artículos :).

Las secciones de tipo PROGBITS son las que se encuentran realmente en el fichero

Lo segundo es que hay varios tipos de secciones. Ya sabéis, si queréis conocer los detalles consultad la especificación. En lo que a nosotros respecta, el tipo interesante es PROGBITS.

Todas las secciones de este tipo, se encuentran físicamente en el fichero y serán cargadas en la posición de memoria que indica la columna Addr, cuando ejecutemos el programa. Además, la siguiente columna, Off, nos indica el desplazamiento en el fichero en el que se encuentran los datos asociados a esa sección y que serán cargados en la dirección de memoria respectiva.

Sobre la sección .note.ABI-tag, volveremos más tarde.

SEGMENTOS O CABECERAS DE PROGRAMA

Los segmentos o cabeceras de programas (Program Headers) son la otra estructura de datos principal, que el formato ELF maneja.

Estos segmentos son los que definen la estructura física real que un determinado programa tendrá en memoria... es decir, donde y como aparecerá cada parte del programa en la memoria del ordenador. Lo que se conoce como el mapa de memoria del proceso.

Veamos la lista de segmentos de `/bin/ls`:

```
occam@razor: $ readelf -l /bin/ls

Elf file type is EXEC (Executable file)
Entry point 0x8049b40
There are 8 program headers, starting at offset 52

Program Headers:
  Type          Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR          0x000034 0x08048034 0x08048034 0x00100 0x00100 R E 0x4
  INTERP        0x000134 0x08048134 0x08048134 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD          0x000000 0x08048000 0x08048000 0x12748 0x12748 R E 0x1000
  LOAD          0x012748 0x0805b748 0x0805b748 0x003a4 0x007e8 RW  0x1000
  DYNAMIC       0x01275c 0x0805b75c 0x0805b75c 0x000e0 0x000e0 RW  0x4
  NOTE          0x000148 0x08048148 0x08048148 0x00020 0x00020 R   0x4
  GNU_EH_FRAME 0x012670 0x0805a670 0x0805a670 0x0002c 0x0002c R   0x4
  GNU_STACK     0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .hash .dynsym .dynstr .gnu.version
.gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata
.eh_frame_hdr .eh_frame
   03     .ctors .dtors .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag
   06     .eh_frame_hdr
   07
```

Por cuestiones de espacio hemos eliminado la última columna de la salida del programa. Esta columna define la alineación de cada uno de los segmentos (byte, palabra, página,...), para nuestro cometido, no necesitamos esta información, pero existen otras técnicas de parcheado de binarios que si la requieren.

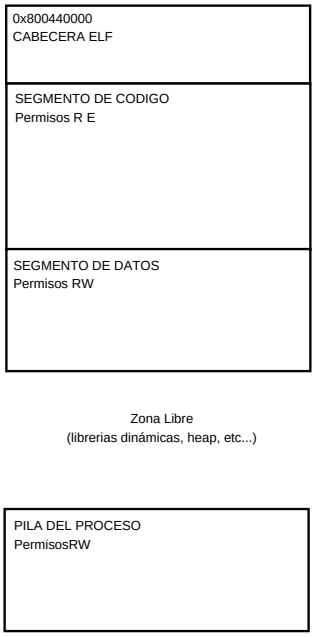
Lo primero que vemos es que la lista de segmentos es mucho más pequeña y además, que estos no tienen nombre. La utilidad `readelf` nos proporciona un mapeado de secciones y segmentos al final de la salida.

Estos mapeados se determinan a partir de las direcciones de secciones y segmentos del tamaño de los segmentos. Podéis hacer la prueba vosotros mismos.

El resto de la información asociada a cada segmento es bastante obvia: el desplazamiento dentro del fichero donde se encuentran los datos, las direcciones en las que se crearán los segmentos y su tamaño en disco y en memoria.
Observad que en general el tamaño en disco y en memoria son iguales, excepto para el segmento número 3 (empezamos a contar en 0 :). Si nos fijamos en las secciones asociadas a ese segmento nos encontramos la sección .bss. Esta sección es la que contiene los datos no inicializados, y por lo tanto, no necesitamos almacenar ninguna información en el fichero.
Cuando el programa se carga en memoria, el cargador se encarga de que haya espacio para esos datos.

MAPA DE MEMORIA DEL PROCESO

Bien. ¿Y qué aspecto tiene todo esto en memoria?. Pues básicamente el que nos ha mostrado readelf cuando le pedimos que nos listara los segmentos del programa, pero algunos comentarios nos van a ayudar a comprender la forma en la que parchearemos el programa en breve.
La siguiente figura muestra el mapa de memoria típico de un proceso en el sistema operativo GNU/Linux.



Hay dos cosas interesantes en esta figura. La primera es que, al menos en los sistemas GNU/Linux, los ejecutables se cargan a partir de una dirección física en memoria (0x800440000). Esto va a hacer nuestra vida mucho más sencilla.
La segunda es que el bloque inicial de segmentos (cabecera + código + datos) se corresponde exactamente con el contenido de nuestro fichero ejecutable. Esto es así debido al diseño del formato ELF, en el que se pretendía reducir lo más posible las operaciones a realizar para cargar un proceso en memoria. En la siguiente sección veremos como sacar partido de esto.
Finalmente, observad que en la figura hemos incluido los permisos de cada segmento (la columna Flg que nos proporciona readelf -l).

Si os fijáis, el primer segmento es el que contiene el código de nuestro programa y solamente tiene permisos de lectura (R) y (E), lo cual es lógico... no queremos que el programa se pueda modificar o automodificar... ¿o sí lo queremos?.

Los ficheros ELF se cargan directamente en memoria para su ejecución

El segundo segmento es el que contiene los datos, variables, sección bss, etc... Este, como es lógico, se inicializa con permisos de lectura (R) y escritura (W).
Como es habitual, la pila se sitúa en las posiciones altas de la memoria y "crece" hacia las bajas, y las librerías dinámicas y el heap (memoria dinámica... malloc y todo eso) utilizarán la memoria a continuación de los segmentos que componen el programa, creciendo hacia las direcciones altas.
De esta forma cuanto más uso hagamos de la memoria dinámica y de la pila, el hueco entre ambas se irá haciendo más pequeño.

PARCHEANDO EJECUTABLES

Después de esta "no corta" introducción vamos al tema que nos ocupa. Parchear ejecutables.
Existen varias formas de parchear un ejecutable ELF. Animamos a los interesados en conocer los detalles de las otras alternativas, a leer los textos de Silvio Cesare... totalmente imprescindibles. Lo que os hemos contado hasta ahora os resultará útil para comprender las distintas técnicas que el señor Cesare explica estupendamente.
Como os decíamos, nosotros vamos a utilizar la que quizás sea la técnica más sencilla: el parcheado del segmento .note.ABI-tag.
La técnica es muy sencilla, básicamente consiste en los siguientes pasos:

- Abrir el fichero a parchear
- Añadirle al final el código que queremos insertar
- Localizar el segmento que contiene la sección .note.ABI-tag
- Modificar el segmento para:
  - que el sistema cargue este segmento con el resto del programa cuando este es ejecutable
  - Tenga permisos de ejecución
  - Los offsets en memoria y en el disco apunten al código que hemos añadido
- Modificar el punto de entrada en la cabecera ELF para que la ejecución se inicie en el nuevo código insertado
- Añadir cualquier información extra que queramos.

No os preocupéis esto es mucho más sencillo de lo que parece :).

ALGUNAS ACLARACIONES

Antes de meternos con los fragmentos de código que hacen cada una de las operaciones que necesitamos, vamos a hacer algunas aclaraciones sobre el porqué de que esto se haga de la forma en la que se hace.

Lo primero es porque utilizamos la sección .note-ABI.tag.

Bien, hay varias razones. La primera es que, al menos en los sistemas GNU/Linux está siempre ahí, gcc la genera.

Añadir segmentos a un ejecutable es complicado, así que reutilizamos uno de los existentes

La segunda, y más importante, es que añadir un nuevo segmento a un ejecutable es bastante complicado, así que nos interesa reutilizar alguno de los que ya existen. El segmento asociado a la sección .note-ABI-tag, ya existe y no tiene ninguna utilidad en la práctica. Es decir, no vamos a romper el programa si lo utilizamos para otra cosa.

Utilizar alguno de los otros segmentos sería bastante más complicado. Bueno, si nuestro código es pequeño existen trucos, pero en el caso general no resulta práctico.

PREPARACIÓN

En nuestro parcheador, utilizamos un par de funciones de ayuda y una constante. Este primer bloque del listado lo podéis ver a continuación.

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

#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>

#include <elf.h>
#include <sys/mman.h>

#define ADDRESS 0x08044000

int get_file_size (int fd)  {
  struct stat _info;
  fstat(fd,&_info);
  return _info.st_size;
}

void PURGATUS_EST (int sys, int cond, char *msg) {
  if (cond) {
      if (sys)  perror (msg);
      else fprintf (stderr, "%s\n", msg);
      exit (1);
    }
  return;
}
```

Sí, existe un fichero elf.h que nos va a hacer nuestra vida más fácil a la hora de leer cabeceras y modificar segmentos.

Además necesitaremos algunas variables en nuestra función main.

```
int main (int argc, char *argv[]) {
  void          *data;
  int           size, size1;
  int           fd, fd1;
  int           i;
  Elf32_Ehdr    *elf_hdr;
  Elf32_Phdr    *elf_seg;
  unsigned char *code;

  PURGATUS_EST (0, argc!=4,
                "Usar: ./note-patch exe codigo clave");
```

Lo primero que hace nuestro programa es comprobar que recibe el número de parámetros correctos. Tal y como lo hemos programado, el programa espera como primer parámetro el nombre del ejecutable a parchear, como segundo parámetro, el código que queremos insertar (más tarde veremos como generarlo) y finalmente la clave.

Si recordáis, lo que pretendíamos hacer era poder añadir una clave a cualquier ejecutable en nuestro sistema, deforma que si la clave correcta no es proporcionada, el programa no se ejecuta. Bien, esa clave la grabaremos a fuego y sangre al parchear el ejecutable.

LEYENDO CÓDIGO Y MODIFICANDO EL EJECUTABLE

Lo siguiente que hará nuestro programa es cargar en memoria el código a insertar y a continuación abrir el fichero del ejecutable a parchear de forma que podamos modificarlo cómodamente.

La forma más sencilla de trabajar con un ejecutable para parchearlo es utilizando la llamada al sistema mmap. De esta forma mapeamos el fichero directamente en memoria, modificamos las posiciones de memoria que nos interese y luego cerramos el fichero.

Aquí podéis ver el fragmento de código que realiza estas dos acciones:

```
/* Leemos código a injectar */
PURGATUS_EST (1,
       (fd1 = open (argv[2], O_RDWR, 0)) < 0,
       "open1:");

size1 = get_file_size (fd1);
PURGATUS_EST (0, (code = malloc(size1)) == NULL,
              "No puedo reservar memoria");
read (fd1, code, size1);
close (fd1);

/* Abrimos ejecutable y lo mapeamos en memoria */
PURGATUS_EST (1,
     (fd = open (argv[1], O_APPEND | O_RDWR, 0)) < 0,
     "open2:");
/* Mapeamos fichero */
size = get_file_size (fd);
PURGATUS_EST (0,
        (data = mmap (0, size,
                PROT_READ | PROT_WRITE | PROT_EXEC,
                MAP_SHARED, fd, 0)) == 0,
          "mmap:");
```

La primera parte es trivial, simplemente leemos el fichero que contiene el código a injectar en un bloque de memoria dinámica que reservamos a tal efecto.

En el segundo bloque de instrucciones abrimos el fichero ejecutable. Notad el uso de O_APPEND, ya que necesitamos añadir nuestro código al final del ejecutable.

La instrucción mmap es la que nos va a permitir acceder a ese fichero que acabamos de abrir como si se tratara de una bloque de memoria. Observad que activamos todo los permisos. El de ejecución no lo necesitamos realmente, pero hay otras aplicaciones en las que hace falta y así ya sabéis como se usa :P.

PARCHEANDO SEGMENTOS

En este punto ya tenemos todo listo para parchear los segmentos de nuestro ejecutable. Veamos el fragmento de código y a continuación lo comentamos.

```
/* Patch code */
elf_hdr = (Elf32_Ehdr*) data;
/* Patch note segment */
elf_seg = (Elf32_Phdr*)((unsigned char*)elf_hdr +
          (unsigned int)elf_hdr->e_phoff);
for (i = 0; i < elf_hdr->e_phnum; i++) {
  if (elf_seg->p_type == PT_NOTE) {
    elf_seg->p_type = PT_LOAD;
    elf_seg->p_flags = 0x7;
    elf_seg->p_offset = size;
    elf_seg->p_vaddr =
              elf_seg->p_paddr = ADDRESS + size;
    elf_seg->p_filesz = elf_seg->p_memsz = size1;
    elf_seg->p_align = 0x1000;
    break;
  }
  elf_seg = (Elf32_Phdr*)((unsigned char*)elf_seg
            + elf_hdr->e_phentsize);
}
PURGATUS_EST (0, (i == elf_hdr->e_phnum),
              ".note.Abi-tag segment not found");
```

Lo primero que hacemos es asignar un puntero del tipo `Elf32_Ehdr` a la zona de memoria en la que hemos mapeado nuestro fichero ejecutable. Esto nos va a permitir acceder a la cabecera ELF a través de esta estructura de datos (definida en elf.h).
A continuación calculamos la posición de la tabla de segmentos, un bloque del fichero en el que se encuentra toda la información que nos proporciona el comando readelf -l.

Para parchear un fichero ELF, modificaremos una de las entradas de la tabla de segmentos

La tabla de segmentos se encuentra en el offset `e_phoff`, definido en la cabecera ELF. Este offset es relativo a la cabecera, por lo que tenemos que sumarle el tamaño de ésta.
Ahora ya solo tenemos que recorrer la tabla de segmentos y encontrar uno del tipo `PT_LOAD`. Como muchos os imaginaréis, esto debería ser un poco más complicado, debiendo existir comprobaciones extra con la información de secciones, de forma que si existieran varios segmentos de tipo NOTE, nos quedáramos con el correcto. En general, solo suele haber un segmento de tipo `PT_NOTE` que estará asociado a la sección .note.ABI-tag. Os queda como ejercicio el dejar esta parte del código en condiciones :).
El número de segmentos del programa lo encontramos en un campo en la cabecera ELF, y lo utilizamos en el bucle for para recorrerlas todas buscando la que nos interesa. Cuando encontramos una cabecera del tipo `PT_NOTE`, procedemos a parchearla de la siguiente forma:

- Cambiamos el tipo a `PT_LOAD` de forma que el segmento se cargará en memoria a partir del contenido del fichero.
- Activamos todos los permisos: lectura, escritura y ejecución. En general, solo los permisos de lectura y ejecución son necesarios, pero en nuestro ejemplo vamos a necesitar escribir en este segmento, así que activamos los tres.
- El offset o desplazamiento en el fichero, para este segmento pasa a ser el tamaño del fichero, o dicho de otra forma, nuestro código extra lo vamos a añadir al final del fichero.
- Actualizamos las direcciones de mapeo en memoria del segmento para que apunten a una zona libre. Para ello le sumamos el tamaño actual del fichero a la dirección base en la que todos los programas ELF se cargan :).
- Actualizamos el nuevo tamaño de este segmento, el cual será el tamaño del código que estamos añadiéndole al final.
- Y finalmente actualizamos el tipo de alienación, que para los segmentos del tipo `PT_LOAD` es habitualmente 0x1000.

Para navegar la tabla de segmentos utilizamos el campo `e_phnum` de la cabecera ELF que nos indica el tamaño de cada una de las entradas.
Finalmente, si no encontramos ningún segmento del tipo `PT_NOTE` terminamos con un error. Esto puede pasar si intentamos parchear un fichero dos veces. En la primera ejecución eliminamos la sección del tipo `PT_NOTE`, de forma que en la segunda ejecución no lo encontraríamos.

MODIFICANDO LA CABECERA Y GRABANDO EL RESULTADO

Para terminar con el proceso de parcheado del fichero ELF, tenemos que reflejar en la cabecera del mismo los cambios realizados, añadir cualquier información extra que nos interese y grabar los resultados.
Esto es lo que se consigue con este fragmento de código:

```
*((int*)((unsigned char*)code + size1 - 8)) =
          elf_hdr->e_entry;
elf_hdr->e_entry = ADDRESS + size;
/* Copiamos la clave al final del fichero */
memset (((unsigned char*)code + size1 - 16),
        0, 8);
strncpy (((unsigned char*)code + size1 - 16),
         argv[3], 8);

write (fd, code, size1);
close(fd);
free (code);
```

La modificación de la cabecera consiste en cambiar el punto de entrada del programa para que pase a ser el código que hemos inyectado, el cual se encuentra al final del fichero (ADDRES + size). Pero, necesitamos almacenar el punto de entrada real, de forma que nuestro código pueda ejecutar el programa real si la contraseña que escribimos es correcta.

A continuación almacenamos la contraseña que hemos proporcionado desde la línea de comandos y terminamos escribiendo y cerrando el fichero para que los cambios realizados tengan efecto en el fichero ejecutable real.

En estos momentos disponemos de una pequeña herramienta, que con unos pocos cambios, nos permitirá injectar fragmentos de código en cualquier ejecutable... básicamente tendréis que generalizar la última parte del programa que es específica para el código que nosotros vamos a insertar en este ejemplo.

CÓDIGO INYECTADO

El código a inyectar lo hemos escrito en ensamblador utilizando las llamadas al sistema del kernel Linux. En principio no podréis utilizar código normal compilado con cualquier herramienta.

Para ello, el parcheador que hemos visto en las secciones anteriores se vuelve muy complejo al tener que manejar las dependencias y la relocalización de código general... Además, para meteros en esos berenjenales necesitaréis un conocimiento del formato ELF muchísimo más profundo que lo que os hemos contado aquí. Vamos el código.

```
BITS 32

_start:
        jmp short get_fp
real_start:
        pop esi
        push esi
        pusha
        ; write (1, "passwd:", 8);
        mov eax, 4          ; write syscall
        mov ebx, 1          ; stdout -> ebx
        lea ecx, [esi]      ; passwd string
        mov edx, 8          ; string size size
        int 0x80
        ; ------------------------------------
        ; read (0, buffer, 8)
        mov eax, 3
        dec ebx
        lea ecx, [esi + 8]
        int 0x80
        ; ------------------------------------
        ; Compare passwd
        push esi
        xor ebx, ebx
        mov edi, ecx
        lea esi, [esi + 32]
        mov ecx, 8
        rep cmpsb
        jnz end

        ; ------------------------------------
        ; exit with error
        pop esi
        popa
        pop esi
        push dword [esi + 40]
        ret
end:
        pop esi
        mov eax, 4                  ; write syscall
        mov ebx, 1                  ; stdout -> ebx
        lea ecx, [esi + 16]         ; passwd string
        mov edx, 16                 ; string size size
        int 0x80

        mov eax, 0x01               ; exit syscall
        int 0x80
        ; ------------------------------------
get_fp:
        call real_start
        ; ------------------------------------
data0   db      'passwd:', 0
data1   db      0x0a, 0x0a, 0x0a, 0x0a, 0x0a,
                0x0a, 0x0a, 0x0a
        db      'Wrong Password!', 0x0a
        db      'sesame01'
entryp  db      '        '
```

Sí, visto así parece muy escandaloso...es lo que tiene el ensamblador, pero es uno de los programas más tontos que se pueden escribir en ensamblador.

Lo primero que vamos a comentar es el final del programa. La zona donde almacena sus datos. Como podéis ver, los últimos 8 bytes están reservados para almacenar el antiguo punto de entrada al programa, y los 8 anteriores almacenan la clave.

Estos son los dos valores que actualizamos desde nuestro programa parcheador, y tienen que estar al final por una muy buena razón.

CÓDIGO PIC

PIC significa *Position Independent Code*, es decir, código que puede ejecutarse en cualquier posición de memoria ya que no contiene ninguna referencia a ninguna posición de memoria absoluta.

¿Y porque estamos hablando de esto ahora?... pues porque el código que estamos inyectando tiene que ser de este tipo, ya que de lo contrario tendríamos que proporcionar información de relocación que haría todo este proceso muy complejo o inviable.

Respecto al código no tenemos grandes problemas... seleccionamos las instrucciones con cuidado para utilizar siempre desplazamientos en lugar de direcciones absolutas y ya está. El problema es el acceso a nuestros datos ya que, cuando se empiece a ejecutar el código no tendremos ni idea de en qué posición de memoria estarán.

Tendremos que recurrir a un truco para poder acceder a nuestros datos desde el ensamblador

Así que para acceder al bloque de datos utilizamos un bien conocido truco. Iniciamos nuestro programa con un salto a una instrucción call justo anterior a la zona de datos. La instrucción call a su vez saltará al punto de inicio de ejecución del programa, pero con la salvedad que en la pila tendremos la dirección de memoria siguiente a la instrucción call (la dirección a la que retornaríamos al ejecutar la instrucción ret), la cual, oh maravilla!, es el inicio de nuestro bloque de datos. Por eso, lo primero que hacemos al empezar el programa es leer un valor de la pila y volver a guardarlo para su uso posterior.

RESTO DEL PROGRAMA

Bien, el resto del programa es bastante tonto. En primer lugar encontramos un bloque de código encargado de mostrar el mensaje "password:" en pantalla, para perdir al usuario que introduzca la clave que hayamos elegido.

A continuación lee 8 caracteres del teclado y los almacena en un buffer. ¿Recordáis que marcamos el segmento con permisos de escritura?

Ahora ya tenemos localizadas en memoria la clave que introdujo el usuario y la clave que hemos elegido para proteger nuestro programa, así que solo tenemos que compararlas con la típica `rep cmpsb`.

Tras la comparación, si las claves no coinciden, el programa mostrará un mensaje de error y terminará con la llamada al sistema exit. Por el contrario, si las claves coinciden, continuaremos la ejecución en el punto de entrada original de la aplicación parcheada, el cual, si recordáis, hemos almacenado al final del fichero cuando lo parcheamos.

Para ejecutar el punto de entrada original, metemos su dirección en la pila y ejecutamos `ret`... pero vosotros podéis hacerlo como más rabia os dé.

Es necesario hacer un comentario final respecto al programa. Si os fijáis hay una instrucción `pusha` al principio de todo, justo depués de conseguir el puntero a nuestros datos y una instrucción `popa` justo antes de ejecutar el punto de entrada original.

Con esto restauramos los valores de registros y flags existentes al inicio del programa, de forma que el programa original se encuentre un entorno sano para su ejecución. Si no lo hacemos, obtendremos algúnos core dumps cuando el programa termina.

PROBANDO LA APLICACIÓN

Bueno, después de semejante rollo, vamos a probar nuestra aplicación. Para nuetras pruebas hemos elegido xeyes... pues porque es un programa simpático :).

Veamos la secuencia de comandos

```
occams@razor $ whereis xeyes
xeyes: /usr/bin/xeyes /usr/X11R6/bin/xeyes /usr/bin/X11/xeyes
/usr/share/man/man1/xeyes.1x.gz
occams@razor $ cp /usr/bin/xeyes xeyes-test

occams@razor $ readelf -l xeyes-test

Elf file type is EXEC (Executable file)
Entry point 0x8048f60
There are 7 program headers, starting at offset 52

Program Headers:
  Type         Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg
  PHDR         0x000034 0x08048034 0x08048034 0x000e0 0x000e0 R E
  INTERP       0x000114 0x08048114 0x08048114 0x00013 0x00013 R
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD         0x000000 0x08048000 0x08048000 0x02610 0x02610 R E
  LOAD         0x002620 0x0804b620 0x0804b620 0x00554 0x014cc RW
  DYNAMIC      0x00299c 0x0804b99c 0x0804b99c 0x00100 0x00100 RW
  NOTE         0x000128 0x08048128 0x08048128 0x00020 0x00020 R
  GNU_STACK    0x000000 0x00000000 0x00000000 0x00000 0x00000 RW

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .hash .dynsym .dynstr .gnu.version
.gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata
   03     .data .eh_frame .dynamic .ctors .dtors .jcr .got .bss
   04     .dynamic
   05     .note.ABI-tag
   06
occam@razor $ ./parcheador xeyes-test codigo sesamo01
occam@razor $ readelf -l xeyes-test

Elf file type is EXEC (Executable file)
Entry point 0x8046fec
There are 7 program headers, starting at offset 52

Program Headers:
  Type         Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg
  PHDR         0x000034 0x08048034 0x08048034 0x000e0 0x000e0 R E
  INTERP       0x000114 0x08048114 0x08048114 0x00013 0x00013 R
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD         0x000000 0x08048000 0x08048000 0x02610 0x02610 R E
  LOAD         0x002620 0x0804b620 0x0804b620 0x00554 0x014cc RW
  DYNAMIC      0x00299c 0x0804b99c 0x0804b99c 0x00100 0x00100 RW
  LOAD         0x002fec 0x08046fec 0x08046fec 0x0008c 0x0008c RWE
  GNU_STACK    0x000000 0x00000000 0x00000000 0x00000 0x00000 RW

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .hash .dynsym .dynstr .gnu.version .gnu.version_r
.rel.dyn .rel.plt .init .plt .text .fini .rodata
   03     .data .eh_frame .dynamic .ctors .dtors .jcr .got .bss
   04     .dynamic
   05     .note.ABI-tag
   06
edma@eve:/mnt/data/work/security/mine/elf/elf-tests$
```

Bueno, y ahora a buscar las diferencias :). Punto de entrada, tipo de segmento, permisos, tamaños,... Esas cosas :)

LIMITACIONES

Si habéis escrito los programas o los habéis descargado de la red probablemente, a estas alturas ya habréis encontrado muchas de las limitaciones de este ejemplo. Aquí os ponemos las principales:

- La clave se ve cuando se escribe.
- La clave tiene que ser de 8 caracteres. Ni más ni menos.

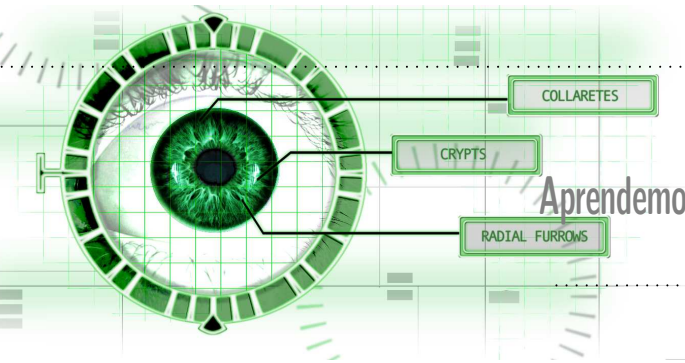
Bueno, esto era solo un ejemplo para ilustrar el proceso de manipulación de un fichero ELF. Si alguno soluciona algunos de estos problemas, que no dude en enviarnos sus soluciones para publicarlas en próximos números.

CONCEPTOS

Tenemos que admitir que nuestro ejemplo es bastante pobre, pero esperamos que ahora tengáis más claros algunos conceptos. El primero es cómo funcionan programas como los *packers* programas que comprimen ejecutables de una forma que siguen siendo ejecutables, o herramientas de protección para cifrar el ejecutable. Hablando de cifrado echadle un ojo al programa cryptelf de SLACKo que junto con los textos de Silvio Cesare nos han permitido escribir este artículo.

El otro concepto que esperamos que os haya quedado claro es que: Sí, se pueden escribir virus para linux :)

Nos leemos en el próximo número



Reconocimiento Biométrico a través del Iris

Aprendemos como funciona la última tecnología de control de acceso

por Fernando Martín Rodríguez

Bueno... aquí estoy de nuevo intentando continuar lo que empecé (una serie de artículos sobre biometría). Vaya por delante que empecé por las huellas porque era de lo que controlaba algo. Ahora estoy pisando terreno un poco pantanoso. Lo bueno es que escribir esto me va a obligar a buscar información sobre el tema y con eso aprenderemos todos.

Hoy veremos cómo se puede hacer reconocimiento biométrico utilizando el dibujo (o patrón) que todos tenemos en el iris del ojo. ¡¡¡Ojo (valga la redundancia) !!! No confundáis el iris con la retina; no lo digo por dar una lección de anatomía, sino porque la retina también es un rasgo biométrico empleado en algunos sistemas (de hecho, es muy fiable, el problema es encontrar una cámara capaz de fotografiar dentro del ojo). Como todos sabemos, el iris es la parte externa y coloreada del ojo (la que hace que tengamos los ojos verdes, azules o castaños).

INTRODUCCIÓN

Empecemos recordando las características que un rasgo debe cumplir para dar lugar a un buen sistema biométrico. Eran las siguientes:

- **Universalidad**: todas las personas deben poseer ese rasgo.
- **Unicidad**: no debe haber dos personas con los rasgos seleccionados iguales.
- **Permanencia**: deben ser invariantes con el tiempo.
- **Cuantificación**: admiten medidas cuantitativas.

Si en el artículo anterior veíamos que las huellas dactilares son el rasgo que mejor cumple estas características, podemos decir que el Iris las sigue a poca distancia. El dibujo del Iris está completo aproximadamente a los 18 meses de edad y permanece inalterable toda la vida (aclaración: el "dibujo" es constante, el color no se estabiliza del todo hasta la pubertad y con la vejez se puede perder un poco de coloración; el reconocimiento de iris no se puede basar en el color). Al igual que ocurre con las huellas dactilares, dos gemelos idénticos (con el mismo ADN) tendrán iris diferentes.

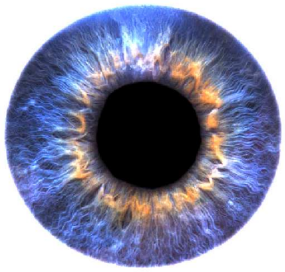


Figura 1. Imagen de un iris humano.

El reconocimiento automático a partir del iris no es tan popular en sistemas de seguridad como el de huellas debido a la relativa incomodidad de captación para el usuario (además, las huellas han sido mucho más estudiadas debido a sus aplicaciones legales). Sin embargo, si la imagen es captada con suficiente calidad, las tasas de reconocimiento llegan a ser superiores a las de huellas. Otra de las razones de su falta de popularidad es que el primer y mejor algoritmo de extracción y comparación de características de iris, debido a John Daugman, está patentado en Estados Unidos y licenciado sólo a algunos fabricantes de sistemas biométricos.

Las patentes de software-

Supongo que sabéis que Estados Unidos permite patentar software (y muchas otras cosas: algoritmos, fórmulas, estrategias de marketing ...). En resumen, permiten patentar ideas (en Europa sólo se permitía patentar dispositivos físicos, esto es: cacharros).
Pensad que las patentes han sido un gran instrumento: patentar un invento obliga a publicarlo, la patente dura 20 años que es el margen de beneficio del inventor (nadie puede fabricar sin pagarle derechos) pero después pasa a formar parte del dominio público. Sí, sí, nadie obliga a patentar pero si la competencia logra reproducir el producto y lo patenta antes, será el inventor el que tenga que pagar.
Yo (y muchos) siempre hemos pensado que una cosa es patentar el tubo trinitron, el transistor o un tipo de antena y otra es permitir patentar hasta la más mínima idea. Imaginaos que pasaría si alguien patentara el bucle for, el contador, el fichero.... La no existencia de patentes sobre el software permite la existencia de pequeñas empresas innovadoras que de otro modo podrían verse ahogadas por el pago de cánones o de gastos legales (cada nuevo producto obliga a un estudio de las posibles patentes infringidas).
Fijaos que dije que en Europa no se "permitía" patentar ideas. Existe desde hace tiempo cierto movimiento que presiona al parlamento europeo para copiar aquí la ley norteamericana. De momento, no lo han conseguido (la propuesta se ha rechazado dos veces). Podéis informaros más sobre este tema en la dirección:
http://lpf.ai.mit.edu/Patents/patents.html.

Indicador biométrico	TFR	TFA	Memoria (bytes)	Tiempo respuesta	Variabilidad intra-clase	Variabilidad inter-clase	Implantación en mercado	Facilidad de engaño
Huellas	3 %	$1:1^6$	100-1000	Depende del sistema	Media	Alta	52 %	Media
Retina	Bajas	aprox. 0 %	Poca	Bastante bajo	Muy baja	Muy alta	<1 %	Baja
Iris	Bajas	aprox. 0 %	1024 bits	Medio-bajo	Muy baja	Muy alta	7,3 %	Baja
Cara	Bajas	Bajas	>10,000	Alto(1:N) Bajo (1:1)	Muy alta	Alta	11,4 %	Muy alta
Voz	Bajas	Bajas	>1000	Alto (1:N) Bajo (1:1)	Alta	Media-alta	4,1 %	Muy alta

Tabla 1: Principales características de algunos indicadores biométricos.

Para terminar la introducción, os vamos a obsequiar con un cuadro comparativo de los diferentes sistemas biométricos (sí, sí, tal vez esto debería haber ido en el primer artículo de la serie). La columna TFA significa "Tasa de Falsa Aceptación" y es el porcentaje de veces que el sistema declara como iguales a dos individuos que no lo son. La TFR es la "Tasa de Falso Rechazo" y es el porcentaje de veces que dos individuos iguales son erróneamente considerados diferentes. Normalmente, se considera que la falsa aceptación es peor que el falso rechazo, aunque eso siempre depende de la aplicación.

Comparado con otros métodos de reconocimiento biométrico, el análisis del iris resulta una técnica muy fiable

La variabilidad intra-clase es la variación entre diferentes tomas de un rasgo para el mismo individuo. Por ejemplo, la cara es muy variable porque depende mucho del gesto, la posición ante la cámara, los cambios naturales debidos al paso del tiempo y otras circunstancias difíciles de controlar (afeitado o no, gafas, peinado...). Un rasgo es más adecuado cuando la variabilidad intra-clase es baja y la variabilidad inter-clase (las diferencias con otros individuos) es alta.
A partir de la tabla vemos que las huellas, el iris y la retina son rasgos muy fiables, mientras que la cara y la voz no lo son tanto.

El iris contiene un complejo patrón de líneas que se entrecruzan formando una textura

TÉCNICAS DE RECONOCIMIENTO

El iris contiene un complejo patrón compuesto por un conjunto de líneas que se entrecruzan formando una textura compleja. En procesado de imagen se habla de texturas cuando tenemos un patrón subyacente a una imagen donde no hay contornos claros que definan objetos pero sí muchas líneas entrecruzadas o no. Son ejemplo de texturas la huellas dactilares, las vetas naturales que se observan al fotografiar un trozo de madera, los hilos que forman una tela (fotografiada de cerca)... La textura de un iris humano tiene una enorme variabilidad entre individuos e incluso entre los dos ojos del mismo individuo.
Las técnicas de identificación mediante análisis del iris comprenden los siguientes pasos:

- **Captura**: para una buena captura se requiere iluminar el iris mediante una fuente en el infrarrojo cercano. Así se consigue no molestar al usuario. Esta iluminación permite capturar los detalles del patrón en iris muy oscuros. La información de color no se utiliza (se utilizan cámaras blanco y negro o se convierte la imagen a escala de grises).

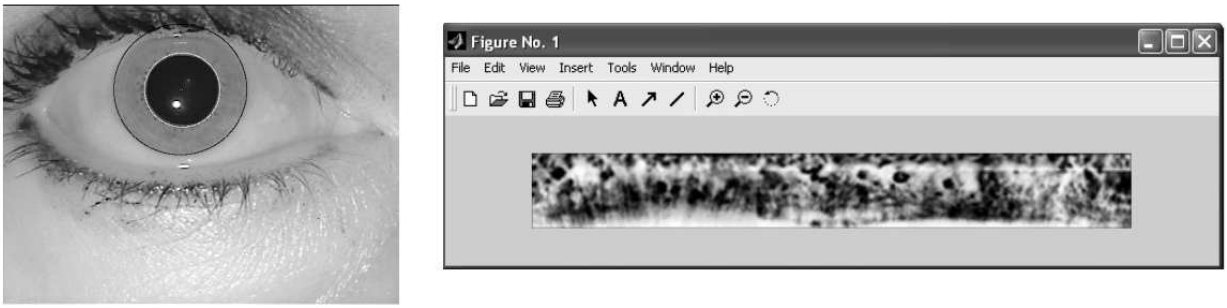


Figura 2. Segmentación, transformación polar e igualación de histograma del iris de la izquierda.

- **Segmentación**: los objetivos a lograr son la localización del centro, eliminación de la pupila y la esclerótica. Para hacer esto se suele utilizar el hecho de que la pupila es negra y la esclerótica blanca (figura 2, izquierda), mientras que el iris ocupa una tonalidad intermedia. Ojo: todos los que hayan estudiado procesado de imagen alguna vez saben que nunca es tan fácil como decir, esto es negro, aquello blanco... Estamos tratando con imágenes que tienen 256 niveles de gris y lo primero que haremos será establecer dos umbrales (blanco cuando el píxel tiene un valor>u1, negro cuando tiene un valor<u2). Con eso deberíamos tener una primera aproximación, esto es: un objeto similar a una corona circular que no es blanco ni negro pero que está rodeado por un objeto blanco y que rodea a un círculo negro. Aplicando un poco de geometría deberíamos lograrlo. Otra fuente de información que se puede usar a veces es el hecho de que el iris es una zona con color (no sabemos cuál pero tiene una tonalidad) mientras que el resto de la foto es gris más claro (hasta el blanco) o más oscuro (hasta el negro). A veces, aquí también se puede utilizar información de color ya que los puntos del iris tienen una tonalidad que su entorno no posee (porque es blanco o negro... en definitiva es gris más o menos brillante).

- **Transformación polar**: el iris es una corona circular y esa no es una forma muy adecuada para trabajar. El objetivo es "estirarlo" para convertirlo en un rectángulo. Para eso haremos una transformación de coordenadas: cada punto tendrá dos coordenadas polares (ángulo θ y radio ρ). Si representamos de nuevo la imagen pero llevando los ángulos al eje x y el radio al eje y lograremos un patrón rectangular (figura 2, derecha).

- **Normalización**: toda imagen capturada por una cámara está sujeta a variaciones de iluminación. Para evitar que eso cause problemas debemos transformar la imagen para tener siempre el mismo contraste (la diferencia entre el punto más oscuro y el más claro). Una de las normalizaciones más utilizadas es la "igualación de histograma" que consiste en transformar algunos de los niveles de gris en otros de forma que el histograma (el número de veces que se repite cada nivel) sea casi constante.

- **Extracción de características texturales**: ahora se trata de convertir la textura en un vector de números. Para eso suelen usarse los llamados filtros direccionales. Un ejemplo simple de filtro direccional es el gradiente. Aplicándolo en una textura nos daría una idea de la dirección de las líneas en cada punto (el vector gradiente saldría perpendicular a la dirección dominante). para extraer características de texturas se suelen usar los filtros de Gabor. En la técnica de Daugman (patentada) se procede a la cuantificación de la fase que resulta de la aplicación de filtros Gabor complejos. La fase de cada resultado local se codifica con 2 bits y cada iris queda codificado con 2048 bits. Otras técnicas utilizan filtros paso-banda ligeramente diferentes a los de Gabor y obtienen vectores reales en lugar de codificación binaria de la fase.

- **Comparación de los vectores codificados**: distancia de Hamming (número de bits diferentes) en la técnica de Daugman o distancias matemáticas clásicas (euclídea, norma 1, norma 2 ...) en los otros casos.

El proceso de reconocimiento se reduce finalmente a la comparación de los vectores obtenidos tras la cuantificación de la fase del resultado de la aplicación de los filtros de Gabor

PARA SABER MÁS

Sé lo que podéis estar pensando... la explicación anterior tiene una "densidad similar al hormigón", sobre todo para lectores que no compartan mi formación (ingeniería de telecomunicación). En fin... hablar de procesado de imagen a nivel técnico es lo que tiene. Si alguien es "muy cafetero" o "muy masoquista" puede ampliar datos con la siguiente bibliografía:

- En la segmentación hablé de elegir dos umbrales pero no dije cómo. Por prueba y error se pueden ajustar umbrales fijos. También se puede intentar calcularlos automáticamente. Existen innumerables publicaciones sobre cálculo de umbrales. La más clásica es el artículo de N. Otsu "A Threshold Selection Method for Gray Level Histograms" (publicado en la revista IEEE Transactions on System, Man and Cybernetics en enero de 1979). Una referencia de infinitamente menor calado pero que habla de obtener varios umbrales al mismo tiempo es el artículo "Analysis Tools for Gray Level Histograms" (publicado en el congreso SPPRA-2003 en junio de 2003). Sí, sí, una vez más me he referenciado a mí mismo (copia en www.gpi.tsc.uvigo.es).

- La igualación de histograma y, más en general, el uso de histogramas en procesado de imagen es una técnica clásica explicada en todos los libros de esta disciplina. Por ejemplo, os recomiendo "Fundamentals of Digital Image Processing" de A.K. Jain (Prentice-Hall) o "Digital Image Processing" de R.C. González (también editado por Prentice Hall).

- Para los filtros de Gabor he visto que hay un resumen cualitativo correcto en la wikipedia (`http://es.wikipedia.org/wiki/Filtro_de_Gabor`). Para mayores profundidades hay tutoriales en la red como:

  - `http://mplab.ucsd.edu/wordpress/tutorials/gabor.pdf`.
  - `http://www.cs.rug.nl/~imaging/simplecell.html`.

    Por cierto, no viene mucho al caso, pero Dennis Gabor también inventó los hologramas y recibió por ello el Premio Nobel de física. Además, publicó un curiosísimo artículo titulado "Theory of Communication" publicado por el IEEE en 1946 donde se propone una teoría de la información diferente a la de Shannon.

- Queda un poco como el villano de la historia, pero J.C. Daugman fue el que resolvió de una vez por todas el problema del reconocimiento de iris (lástima que no quisiera compartirlo). Su trabajo está descrito en varios artículos. Por ejemplo, éste lo escribió él mismo y se puede bajar de internet:

  `http://www.argus-solutions.com/pdfs/recog_persons_iris.pdf`.

  ¡¡¡Ojo!!! Que esté publicado no quiere decir que se puedan usar sus ideas sin pagarle, también está patentado (US Patent No. 4641349), Daugman fundó la empresa Iridian Technologies para explotar esta patente. Bueno... hasta donde yo sé, si en Europa no hay patentes intelectuales podríamos hacer productos usando este algortimo sin mayor problema. ¿Algún abogado me puede corregir? A la hora de comercializar esos productos en Estados Unidos sí que íbamos a tener lío... De todas formas no pienso intentarlo, no vaya a ser que tenga que escribir el próximo artículo desde Carabanchel.

- En nuestra Escuela algo se ha trabajado sobre este tema. El Proyecto Fin de Carrera "Desarrollo de un Algoritmo de Identificación de Personas Mediante su Iris" consiste en la implementación en matlab de un método de reconocimiento (distinto al de Daugman). Su autor es Óscar González Rey y su tutor el profesor José Luis Alba Castro.

ENSUCIÁNDONOS LAS MANOS

Yo, como todos, tiendo a la ley del mínimo esfuerzo, por eso quería dar por terminado este artículo. La dirección de la revista, muy sabiamente, me convenció de que trabajara un poco más.... y aquí está el resultado. He hecho un pequeño programa en matlab intentando poner en práctica parte de lo que os he contado.... A la izquierda el código, a la derecha los resultados parciales. ¡¡¡Ojo!!! falta la última parte.

```
%
% Extraccion de iris.
%
clear;close all; % Limpiar
fichero = 'Ojo1.jpg'; % Fichero de entrada
ImOjoRGB = imread(fichero); % Leer imagen
ImOjoHSV = rgb2hsv(ImOjoRGB); % Pasar a HSV
ImAux = imcomplement(ImOjoHSV(:,:,3)); % Negativo
ImDisc1 = ImAux.*ImOjoHSV(:,:,2);
% Iris = Poca luminacia y mucha saturacion
umbral = graythresh(ImDisc1);
% Umbral de saturacion (Otsu)
Masc0 = im2bw(ImDisc1,umbral); % Mascara previa
radio = 20;
ee = strel('disk',radio);
Masc1 = imopen(imfill(Masc0,'holes'),ee);
% Limpiar (empezamos eliminando la esclerotica)
% Ahora vamos a recortar el iris
[label,num] = bwlabel(Masc1);
% Etiquetar (separar objetos)
props = regionprops(label,'basic');
maxarea = -1;
idx = -1;
for i=1:num,
    % Buscar area maxima
    area = props(i).Area;
    if (area>maxarea)
        maxarea = area;
        idx = i;
    end
end
Rect = props(idx).BoundingBox;
IrisRGB = imcrop(ImOjoRGB,Rect);
IrisHSV = imcrop(ImOjoHSV,Rect);
ImDisc2 = imcrop(ImDisc1,Rect);
Masc2 = imcrop(Masc1, Rect);
% Recortar la zona de interes de cada imagen
%
% Ahora eliminar la pupila
IrisBN = rgb2gray(IrisRGB); % Pasar a B/N
U1 = 10;
U2 = 90; % Umbrales <<fijos>>
Masc3 = roicolor(IrisBN,U1,U2); % Mascara
```

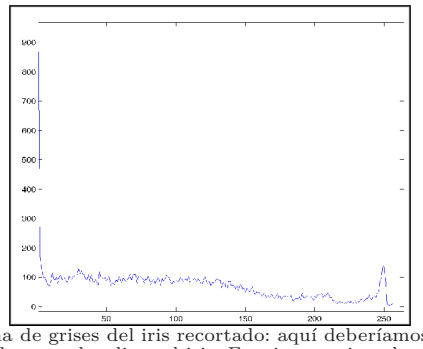
Imagen Original

Imagen "discriminante": resultado de multiplicar la saturación (pureza del color) por el negativo del brillo (el iris tiene mucha pureza de color y menos brillo que el resto de la imagen).

Máscara del iris: se obtiene procesando la imagen anterior. Primero se rellenan las regiones con huecos, se limpia un poco y conservamos la mayor mancha blanca.

Iris recortado: la máscara anterior nos permite eliminar la esclerótica... todavía falta quitar la pupila.

```
%
% Coger, de nuevo, la mayor area.
[label,num] = bwlabel(Masc3);
props = regionprops(label,'basic');
maxarea = -1;
idx = -1;
for i=1:num,
    area = props(i).Area;
    if (area>maxarea)
        maxarea = area;
        idx = i;
    end
end % Encontrar el area mayor
Masc3 = (label==idx);
Centro = props(idx).Centroid;
Xc = round(Centro(1));
Yc = round(Centro(2));
% Posicion del centro
radio = 10;
ee = strel('disk',radio);
Masc4 = imclose(Masc3,ee);
% Limpiar (rellenar huecos)
Imaux = double(imfill(Masc4,[Yc Xc]));
Masc5 = logical(Imaux-double(Masc4));
% Esta es la pupila
Imaux = double(imfill(Masc4,'holes'));
Masc6 = logical(Imaux-double(Masc5));
% Mascara rellena menos la pupila
%
% Ahora convertir el iris a un rectangulo.
props = regionprops(Masc5,'basic');
Centro = props(idx).Centroid;
Xc = round(Centro(1));
Yc = round(Centro(2));
% La mascara solo tiene un objeto
[alto,ancho] = size(Masc5);
Rmax = sqrt(alto^2+ancho^2)/2;
IrisCartesiano = [];
for tita=0:359, % Para cada angulo
  % Recorremos ese radio de la imagen.
  columna = [];
  for radio=0:Rmax,
    x = round(Xc+radio*cos(tita*pi/180));
    y = round(Yc+radio*sin(tita*pi/180));
    if (x>1)&(x<=ancho)&...
      (y>1)&(y<=alto)&...
      (Masc6(y,x)~=0)
        % Copiar punto en su nueva posicion
        valor = IrisBN(y,x);
    else
        %No hay punto para copiarlo
        valor = 0;
    end
    columna = [columna; valor];
  end
  IrisCartesiano = [IrisCartesiano columna];
end
%
% Ahora ecualizar el histograma
IrisNorm = histeq(IrisCartesiano);
% Esto ya lo hace el matlab por nosotros
```

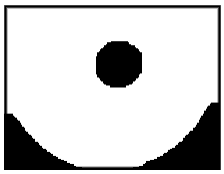


Histograma de grises del iris recortado: aquí deberíamos encontrar los umbrales para localizar el iris. Funciona casi cualquier par que empiece después del pico en cero (pupila negra) y acabe antes del pico en trono al 255 (restos de esclerótica).

Máscara inicial para eliminar la pupila.

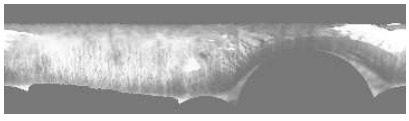
Máscara después de limpiarla.

Pupila localizada: hueco que contiene al centroide de la máscara total.

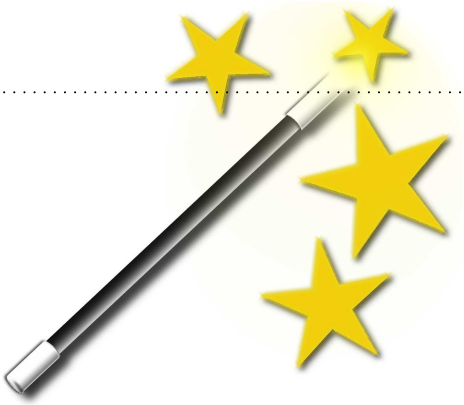


Si tenemos la pupila podemos rellenar toda la máscara, EXCEPTO, la pupila.

Iris rectangular: se obtiene recorriendo el iris mediante las coordenadas polares. Falta un "trozo" debido a que el párpado tapa casi medio iris.



Iris normalizado: al aplicar la ecualización logramos repartir los niveles de gris y ganamos contraste. Ojo, realmente antes de esto habría que quitar la zona negra arriba y abajo y seguramente reescalar a un tamaño fijo.



Con un par... de líneas

Chuletillas para hacer cosas mú rápido

por Tamariz el de la Perdíz

LÍNEA DE COMANDOS AVANZADA PARA CUALQUIER PROGRAMA DE CONSOLA

Utilizando la herramienta `cle` podemos añadir funcionalidades avanzadas de edición a cualquier programa en modo consola.
Probad, por ejemplo (previa instalación del paquete claro):

```
occam@razor # cle nc localhost 110
```

Ahora, tras escribir unos comandos, pulsad el cursor arriba, o la tecla inicio para editar la línea actual que estéis editando... Impresionante!.

CIFRANDO TU NUEVO DISCO

Los "device mappers" proporcionados por los kernel Linux 2.6 permiten hacer cosas bastante interesantes. Una de ellas es el cifrado de dispositivos de una forma sencilla. Si has instalado una distribución recientemente, esto funcionará directamente, sino, tendrás que instalar algunos paquetes para poder utilizar los "device mappers".

```
occam@razor # cfisk /dev/sdb
occam@razor # crypsetup create -y crypt /dev/sdb1
Password:
Password:
occam@razor # mfs.ext3 /dev/mapper/crypt
occam@razor # mount /dev/mapper/crypt \
> /mnt/mi_disco_seguro
occam@razor #
```

La secuencia de comandos anterior nos permite formatear el disco duro (cfdisk), activar el device mapper para una de las particiones (cryptsetuo) formatearla (mkfs.ext3) y montar la nueva partición cifrada.
Eso solo lo tenéis que hacer la primera vez. Para su uso normal solo tenéis que hacer lo siguiente:

```
occam@razor # crypsetup create crypt /dev/sdb1
occam@razor # mount /dev/mapper/crypt \
> /mnt/mi_disco_seguro
occam@razor #
```

VOLCADO HEXADECIMAL

La herramienta `od` es una de esas grandes desconocidas en los sistemas UNIX, que merece la pena conocer dada su versatilidad. Básicamente, esta herramienta nos permite realizar volcados de datos binarios.
Probablemente los parámetros que utilicéis más a menudo serán estos:

```
occams@razor # od -Ax -tx1z /bin/ls
```

El primer parámetro -Ax indica que la base del volcado será hexadecimal (otros valores posibles son d para decimal y o para octal). A continuación indicamos el formato deseado para el volcado, en este caso `x1z` es la cadena elegida. En ella, `x1` indica a od que vuelque cada línea de datos como valores hexadecimales de 1 byte. Si os interesa, por lo que sea, buscar valores de 16 ó 32 bits en vuestro chorro binario cambiad 1 por 2 ó 4. La `z` final indica a od que, además, vuelque esos datos como una cadena de caracteres.

```
occams@razor # nc servicio 10000 | od -Ax -tx1z /bin/ls
```

Cuando no hay otra cosa a mano, es una opción para depurar protocolos binarios :).

CIFRADO DE FICHEROS

Cifrar ficheros es realmente sencillo usando gpg. Alguna gente cree que para usar gpg tiene que generar distintos tipos de claves que no sabe muy bien como manejar... Bien, eso es cierto, para utilizar toda la potencia de gpg, pero si solo quieres cifrar un fichero, gpg ofrece un modo de cifrado simétrico... el de la palabra secreta de toda la vida.
Lo único que tenéis que hacer para cifrar y posteriormente descifrar un fichero es:

```
occams@razor $ gpg -c mi_fichero.tgz
occams@razor $ gpg  mi_fichero.tgz.gpg
```

Por defecto, gpg usa el algoritmo CAST5, pero podéis utilizar cualquiera disponible en vuestra versión. Para saber que algoritmos podéis usar:

```
occams@razor $ gpg --version
(...)
Supported algorithms:
Pubkey: RSA, RSA-E, RSA-S, ELG-E, DSA
Cypher: 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH
Hash: MD5, SHA1, RIPEMD160, SHA256, SHA384, SHA512
Compression: Uncompressed, ZIP, ZLIB, BZIP2
occams@razor $
```

Como podéis ver, DES ya ni aparece :)

UNDO EN VIM

Sí, se nos olvidó comentarlo en el artículo y ya no teníamos sitio. Pulsad "u" en modo comando para deshacer y "CTRL+R" para rehacer. ;)

Envía tus trucos

Puedes enviarnos esos trucos que usas a diario para compartirlos con el resto de lectores a la dirección:

occams-razor@uvigo.es

CALL FOR PAPERS

Julio 2008

Especial Seguridad

comunicación · privacidad · autenticidad
redes · legislación · hardware · vigilancia

Nº4

OCCAM'S RAZOR

no te cortes con la navaja de OCCAM

http://webs.uvigo.es/occams-razor

Nº3

Nº2

Nº1

¿CÓMO UTILIZAN LA TECNOLOGÍA?

Profesionales, Investigadores, Autodidactas, Científicos, Estudiantes, ...

Físicos, Químicos, Ingenieros, Matemáticos, Economistas, Genetistas, Arqueólogos,

ilósofos, Médicos, Linguistas, Diseñadores, Abogados, Astrónomos, Detectives, Res

tauradores, Biólogos, Músicos, Mecánicos, Estadísticos, Arquitectos, Artistas, Geól

ogos, Metereólogos, Deportistas, Antropólogos, Historiadores, Periodistas, Cartógra