Taller1. Entorno de ejecución de programas

Juan Francisco Cardona Mc
Juan Pablo Gaviria*

Introducción

Normalmente cuando ejecutamos un programa, ya sea a través de una línea de comandos, o de un *click* en cualquier entorno gráfico, se desatan una serie de tareas que permiten hacer que el programa seleccionado se convierta en un proceso. Estas labores son ocultadas por el sistema operativo, evitando al usuario tener que preocuparse de labores tan complicadas e insidiosas.
Es muy sencillo ejecutar un programa usando el sistema operativo, pero no hace mucho tiempo cuando la tarea de lanzar y ejecutar un simple programa era una tarea de científicos de la computación, dicha labor requería una cuidadosa planificación y puesta en marcha. El operador deberían indicar muchas cosas sobre los programas que se ejecutaban, como indicar en que unidad de disco se encontraba el programa incluyendo detalles tan específicos como la cara inicial, la cara final, el sector inicial, el sector final, el cilindro inicial y el cilindro final del disco donde estaba ubicado el programa, además dar un *entorno de ejecución*, cuál sería la posición de memoría donde estaría ubicado el programa, señalar en cuál terminal y cuál teclado debería leer y escribir sus datos, etc. etc. El sistema operativo ayudó a ocultar esta complejidad construyendo varias abstracciones, que sirven para dos objetivos importantas, la primera la de ofrecer una máquina virtual y la segunda para gestionar los diferentes recursos. Para acceder a los servicios del sistema operativo se creó la abstracción de las llamadas al sistema y para permitir que nuestros programas se ejecuten creo la abstracción del entorno de ejecución.

Antes de mirar la iteracción con el entorno de ejecución es importante hablar un poco sobre los lenguajes de programación que utilizaremos durante los talleres.

*Instalación de mingw

Lenguajes de programación

El lenguaje de programación C

Gran parte de los sistemas operativos modernos son construidos utilizando el lenguaje de programación C.

Compilación en Linux

El `gcc` será el compilador utilizado en estos talleres, para compilar los programas usualmente lo haremos a través de la línea de comando, aunque usted puede utilizar cualquier *IDE*.

La sintaxis del compilador es generalmente a la siguiente:

```
gcc [-c|-S|-E] [-std=standard]
    [-g] [-pg] [-Olevel]
    [-Wwarn...] [-pedantic]
    [-Idir...] [-Ldir...]
    [-Dmacro[=defn]...] [-Umacro]
    [-foption...] [-mmachine-option...]
    [-o outfile] infile...
```

Esto nos indica que podemos cero o más opciones para compilar un o más archivos desde el parámetro `infile`.

Tenemos el programa LHolaMundo.c, este contiene el conocidisimo ejemplo.

Para compilarlo y generar código objeto, se hace:

```
$ gcc -c LHolaMundo.c
```

La opción `-c` le indica al compilador que queremos que compile nuestro archivo `infile` y genere un archivo objeto. El resultado de nuestra compilación es un archivo llamado HolaMundo.o

Para compilar nuestro programa y generar un ejecutable, lo podemos hacer utilizando de nuevo el compilador `gcc`, así:

```
$ gcc -o LHolaMundo LHolaMundo.o
```

La anterior instrucción transforma el archivo objeto `LHolaMundo.o` a un archivo ejecutable llamado `LHolaMundo`. En el mundo de Linux no existen las extensiones `.exe`, `.com` o `.bat` para indicar que los programas son ejecutables.

Para generar nuestro ejecutable, lo podiamos haber hecho directamente desde nuestro archivo fuente, así:

```
$ gcc -o LHolaMundo LHolaMundo.c
```

Para ejecutar nuestro programa, lo hacemos de la siguiente forma:

```
$ ./LHolaMundo
```

Compilación en Windows

En Windows se puede conseguir una gran variedad de compiladores de diferentes vendedores. En el curso se sugiere el uso de **MinGW** (`http://www.mingw.org/`). Pero también los ejemplos se harán con **Visual Studio 2010**.

Existen muchas diferencias en la programación de los diferentes entornos, pero hay una diferencia que queremos resaltar y se referiene a la extensión que tendrán los ejecutables: `.exe`[1].

Para el compilador de Windows, Visual Studio 2010, en primer lugar debemos abrir una consola especial de comandos, esta se abre siguiendo los siguientes menus `Todos los programas ->Visual Studio 2010 ->Herramientas de Visual Studio ->Símbolo del sistema de Visual Studio`.

Descargamos el programa ejemplo WHolaMundo.c

Una vez abierta para compilar nuestro programa de fuente a objeto, lo hacemos a través de los siguientes pasos:

```
C:> cl /c WHolaMundo.c
```

Una vez generado nuestro código objeto, vamos a crear el ejecutable, este se hace con la siguiente línea de comandos.

```
C:> cl /FeWHolaMundo.exe WHolaMundo.obj
```

También se puedo hacer de forma directa la compilación al archivo ejecutable:

```
C:> cl /FeWHolaMundo.exe WHolaMundo.c
```

Para ejecutarlo:

```
C:> WHolaMundo
```

Para instalar MinGW dirigirse a la pagina oficial y descargar la ultima versión. Ejecutar el archivo descargado, dar en el botón instalar y luego en continue para instalar con las opciones por defecto. Una vez terminada la instalación se le mostrará una ventana que le dará a elegir los componentes a instalar.

Elija los siguientes items de la lista para instalar las herramientas necesarias:

[1] Existe otra extensión `.com` que era utilizada en las versiones de DOS que no permitía relocalización de los programas

- mingw32-base, class: bin
- mingw32-gcc-g++, class: bin, dev, doc y man

Una vez marcados los items elija la opción Installation, Apply changes, luego pulse el boton Apply. Para utilizar el compilador de MinGW, se debe modificar la variable de entorno PATH, esto se hace desde la utilidad Sistema en el Panel de Control. Pulse en Configuración avanzada del sistema, luego en el botón Variables de entorno, busque la variable PATH y pulse en modificar. Agregue la carpeta de bin que se encuentra en la carpeta de instalación de MinGW (por defecto C:/MinGW/bin).

Una vez configurada la variable de entorno, abra una consola de comandos y realice los mismos paso que se hicieron para compilar el programa en el entorno Linux.

El lenguaje de programación C++

El lenguaje de programación C++, creado por Bjane Stroustrup en los laboratorios Bell como una extensión del lenguaje de programación C, también es una de los lenguajes más utilizados para la implementación de sistemas distribuidos.

Compilación en Linux

En linux el compilador de C++ es otro programa llamado `g++`, la sintaxis de este compilador es la siguiente:

```
g++ [-c|-S|-E] [-std=standard]
    [-g] [-pg] [-Olevel]
    [-Wwarn...] [-pedantic]
    [-Idir...] [-Ldir...]
    [-Dmacro[=defn]...] [-Umacro]
    [-foption...] [-mmachine-option...]
    [-o outfile] infile...
```

Como pueden notar la sintaxis es muy similar.

De nuevo utilizaremos una versión para C++ del programa clásico de Kerninghan y Richie LHolaMundo.cpp.

Para generar código objeto:

```
$ g++ -c LHolaMundo.cpp
```

Para crear el ejecutable:

```
$ g++ -o LHolaMundo LHolaMundo.o
```

Para crear el ejecutable directamente:

```
$ g++ -o LHolaMundo LHolaMundo.cpp
```

Para ejecutar nuestro programa:

```
$ ./LHolaMundo
```

Compilación en Windows

En Windows, en particular cuando se utiliza el compilador de Visual Studio 2010, la compilación se hace similar a la compilación hecha en C para Windows con dicho compilador. El programa utilizado en esta sección es WHolaMundo.cpp. Ya que, el compilador `cl` se guía por la extensión del fichero fuente para determinar si llama internamente el compilador del lenguaje C o C++.

En el caso de MinGW, la compilación se hace similar a la hecha en el entorno Linux.

El lenguaje de programación C#

El lenguaje de programación C#, desarrollado por Microsoft como parte de .NET, es un lenguaje orientado a objetos diseñado para construir aplicaciones que corran en el Framework .NET.

Compilación en Windows

Para compilar aplicaciones C# desde consola, se puede utilizar el compilador instalado desde el framework .NET en nuestro PC. Se debe buscar la versión, más reciente que se ha instalado. Para ello, dirigase a C:/Windows/Microsoft.NET/Framework64/ y elija la carpeta con la ultima versión. Para ejecutarlo desde consola escriba la ruta completa hasta el compilador (C:/Windows/Microsoft.NET/Framework64/[version]/csc.exe) o agregue esta ruta a la variable de entorno PATH (como se explica anteriormente en la instalación de MinGW).

Para compilar:

```
C:> csc.exe  Programa.cs
```

Para ejecutar:

```
C:> Programa.exe
```

Entorno de ejecución

Al igual que una persona cuando crece, el entorno que lo rodea le ayuda (o le impide) a comprender el mundo en que se desenvuelve, entender los símbolos de su medio, las conveciones para interactuar y los protocolos comunes. Una persona los adquiere con la educación.

De forma similar un proceso para llegar a ser ejecutado tiene también un entorno que le ayudará a comprender la cultura del sistema operativo en el cuál se ejecuta.

El entorno de ejecución para un proceso dependerá del sistema operativo en el cual se ejecutará, en algunos casos consistirá de los caminos de localización[2], el directorio de trabajo, los archivos de configuración, parámetros de inicialización que le permitirá al proceso definir cuáles son sus recursos (impresoras, archivos), sus permisos de ejecución, la ubicación de las bibliotecas de funciones que necesita, el responsable de la ejecución del proceso[3]. Lo anterior y otras cosas hacen parte del entorno del ejecución del programa.

El entorno de ejecución permite que el programador pueda hacer su aplicación personalizable dependiendo de la máquina (entorno) donde el programa se ejecutará. Existen varios mecanismos[4] que permiten modificar el entorno de ejecución:

- Lista de argumentos.
- Variables de ambiente.
- Archivos de configuración.
- Registro del sistema operativo.

Las dos primeras son los mecanismos más representantivos de los sistemas operativos; la tercera dependerá del diseñador del programa o la aplicación y la última no se encuentra disponible en todos los sistemas operativos.

No es necesario establecer el ambiente de un programa para cada ejecución del programa, el sistema operativo se encarga de ello, preconfigurando un ambiente por omisión para cada usuario que ejecute un programa determinado, o en algunas ocasiones, existe archivos de configuración que determina de antemano el entorno de ejecución.

En este primer taller se estudiará el entorno de ejecución de los programas, y el mecanismo para acceder a los servicios del sistema operativo a través de los programas.

Argumentos

Cuando ejecutamos un programa necesitamos en algunos casos pasarle un lista de argumentos que le indique al programa hacer algo. Suponga que nece-

[2] La ubicación de los archivos ejecutables.
[3] Usuario.
[4] También dependiente del sistema operativo donde se ejecutará

sita modificar varios archivos al tiempo, usted podría abrir su editor de texto favorito y dentro de este darle la orden de abrir el archivo y así por cada uno de ellos. Pero esto resulta un poco molesto después de un tiempo. Para evitar esta molestia, los sistemas operativos, a través del *shell* y de los lenguajes de programación permite el uso de la línea de argumentos, esta existe para indicarle a un programa desde el momento que se va ejecutar, un conjunto de cadenas de caracteres que pueden ser pasadas al entorno de ejecución del programa.
¿Cómo un programa puede acceder a la línea de comandos? esto depende tanto del sistema operativo, cómo en algunos casos, si el lenguaje de programación utilizado tiene características intrínsecas para interactuar con la línea de argumentos.

¿Cómo acceder a las línea de argumentos?

Casi todos los lenguajes de programación permiten manejar la línea de argumentos[5], o el sistema operativo permite acceder a estos recursos a través de las llamadas al sistema, vamos a ver en este caso a través de lenguaje de programación C o C++.

En C (o C++) la línea de comandos se presenta como parte de la función `main`:

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

Se observa que `main` puede llevar dos argumentos, el primer argumento `argc` de tipo entero, indica cuantos argumentos se le han pasado al programa (incluyendo el nombre del programa mismo) y al argumento `argv` es un vector de punteros que apuntan a cadenas de caracteres terminadas por el valor 0. De esta forma un programa sabe cuantos argumentos le han pasado. Miremos el siguiente programa que muestra cada uno de los argumentos pasados a él:

```
 1: /* LListaDeArgumentos.c
 2:  * Este programa se encarga de mostrar la lista de argumentos
 3:  * pasados al programa.
 4:  */
 5: #include <stdio.h>
 6:
 7: int
 8: main(int argc, char *argv[]) {
 9:
10:   int i;
```

[5]Esto se debe a que el entorno de ejecución preponderante era el entorno de lotes

```
11:
12:   fprintf(stdout,
13:    "Numero total de argumentos: %d\n", argc);
14:
15:   /* Recorre cada uno de los argumentos y los
16:    * muestra en la salida estandar */
17:   for (i = 0; i < argc; i++) {
18:     fprintf(stdout, "Argumento[%d]=%s\n", argv[i]);
19:
20:   }
21:
22:   return 0;
23: }
```

LListaDeArgumentos.c

Mire ahora como el anterior programa muestra sus argumentos:

```
$ ./LListaDeArgumentos argumento1 argumento2
Numero total de argumentos: 3
Argumento[0]=./LListaDeArgumentos
Argumento[1]=argumento1
Argumento[2]=argumento2
```

Observe que el primer argumento pasado es el nombre del programa mismo[6], y los restantes argumentos fueron los que se pasaron en la línea de ejecución.

Es obvio que se puede pasar una lista de argumentos con los nombres de todos los archivos que un programa de aplicación debe utilizar, pero que pasa cuando tenemos que indicar algo diferente, por ejemplo el nombre del dispositivo de salida. La primera solución que nos ocurre es utilizar argumentos posicionales, por ejemplo el 5-ésimo argumento señala el nombre del dispositivo de salida. Eso está bien en caso anterior cuando hay 4 argumentos y estos siempre sean obligatorios. Qué pasa si no los hay, o en algunas ocasiones hay cuatro o tres o dos o uno. Entonces, en este caso los argumentos posicionales no son útiles, que solución nos queda, podemos identificar de alguna forma que argumentos tienen un significado determinado ante-poniendo un carácter especial que identifique que ellos significan algo. Esto son los llamados *switches*, y aunque no son definidos por el sistema operativo, existen convenciones especiales para definirlos en cada sistema operativo.

Opciones[7] de línea de comando en Unix

En Unix las opciones de línea de comando están definidas anteponiendo a los argumentos que tienen un significado especial el caracter (-) y a veces una

[6] No todos los lenguajes de programación imprimimen como el primer argumento el nombre del programa.

[7] Conocidos como *Switches*

letra o una palabra que identifique y después están los argumentos. En algunas versiones de Linux, los switches son señalados tanto con un solo signo, como por dos seguidos (–).

Las *opciones* pueden ser más fácilmente programados, utilizando la función `getopt(3)` o la función `getopt_long(3)`[8]. La siguiente es la sinopsis de la función `getopt`

```
#include <unistd.h>

int getopt(int argc, char * const argv[],
           const char *optstring);

extern char *optarg;
extern int optind, opterr, optopt;
```

La función `getopt` analiza los argumentos de la línea de comandos. Los parámetros `argvc` y `argv` son el número y el vector de apuntadores que contiene los argumentos de la línea de comandos (que son pasados a la función `main`). Un elemento de opción, es un elemento de `argv` que comienze con '-' (y que no sea exactamente "-" ni "–"). Si la función `getopt` es llamada *repetivamente*, devuelve sucesivamente cada uno de los caracteres de opción.

Si `getopt(3)` encuentra otro carácter de opción, lo devuelve, actualizando la variable externe `optind` y una variable estática `nextchar` de forma que la siguiente llamada a `getopt(3)` pueda seguir la búsqueda en el siguiente carácter de opción o elemento de `argv`. Si no hay más caracteres de opción, `getopt(3)` devuelve -1. Entonces `optind` es el índice en `argv` del primer elemento de `argv` que no es opción. `opstring` es una cadena que contiene los caracteres de opción legítimos. Si un carácter de éstos es seguido por el carácter de dos puntos, la opción necesita un argumento.

El siguiente es un ejemplo de como utilizar las opciones. Supongamos que nuestra aplicación se le puede indicar explicitamente que se va a utilizar la impresora y cuál impresora utilizar, y si el programa activa el modo gráfico o el modo consola.

```
 1: /* LManejoDeArgumentos.c */
 2:
 3: #include <stdio.h>
 4: #include <unistd.h>
 5:
 6: int
 7: main(int argc, char *argv) {
 8:
 9:   extern char *optarg;
10:   extern int optind, opterr, optopt;
```

[8] Únicamente en Linux

```
11:   int option;
12:   /*
13:    * Verificando los argumentos
14:    */
15:
16:   while ((option = getopt(argc, argv, "p:gc")) != -1) {
17:
18:     switch (option) {
19:     case 'p':
20:       fprintf(stdout, "Activada la impresora: %s\n", optarg);
21:       break;
22:
23:     case 'g':
24:       fprintf(stdout, "Entorno gráfico activo\n");
25:       break;
26:
27:     case 'c':
28:       fprintf(stdout, "Entorno de consola activado\n");
29:       break;
30:
31:     default:
32:       fprintf(stdout, "No se reconoce la opcion\n");
33:       break;
34:     }
35:   }
36:
37:   return 0;
38: }
```

LManejoDeArgumentos.c

La ejecución del anterior programa puede ser así:

```
$ ./LManejoDeArgumentos -p nada -c
Activada la impresora: nada
Entorno de consola activado
```

Ejercicio 1. Compile y ejecute el anterior programa.
Ejercicio 2. Añada más opciones al anterior programa.
Ejercicio 3. . Escriba un programa que lea una lista de argumentos de los cuales una parte son opciones y los demás son nombres de archivos, imprimiendo todas las opciones (utilizando la función `getopt(3)` y listando los nombres de archivos utilizando las variables argc y argv.
Ejercicio 4. . Después de procesar la línea de argumentos, ¿la función `getopt(3)` modifica la línea de argumentos?

Opciones de línea de comandos en Windows NT

En Windows, las cosas son muy parecidas a Unix, pero debido a que en Windows se han mezclado muchas alternativas, existen varias convenciones, generalmente muchos programas diseñados antes que el sistema operativo Windows NT, aceptaban el mismo tipo de convención que el sistema operativo Unix, pero a medida que el sistema operativo DOS fue alcanzando su madurez la convención de las opciones cambiaron por el caracter '/'.

Ejercicio 5. Busque como implementar el programa de la sección anterior en el sistema operativo Windows.

Ejercicio 6. Existe alguna otra forma de obtener la línea de comandos en Windows.

Opciones de líneas de comandos en Java

En Java podemos procesar las líneas de comandos a través de el método de entrada `main` de la clase principal. Miremos el cuerpo el método main, en una clase cualquiera:

```
0:    public class ... {
1:
2:      public static void main(String []args) {
                ⋮
3:
4:       }
5:    }
```

Se puede observar en la línea 2, la firma del método `main`, este es el punto de entrada en la ejecución de una clase cualquiera. Este método no retorna ningún valor directamente al entorno de ejecución para indicar como terminó. Tiene un único parámetro llamado `args` de tipo arreglo de `String`, este parámetro es el mecanismo por el cual se le pasa los argumentos de la línea de comandos.

En el siguiente ejemplo muestra los parámetros de la línea de comandos en Java:

```
 1: // Lista de argumentos en Java
 2:
 3: public class ListaDeArgumentos {
 4:
 5:     public static void main(String []args) {
 6:
 7:     for (int i = 0; i < args.length; i++) {
 8:
 9:         System.out.println("args[" + i + "] = " + args[i]);
10:     }
11:     }
```

```
12: }
```

JListaDeArgumentos.java

Ejercicio 7. Compile y ejecute el anterior programa con 0 o más argumentos.

Opciones de líneas de comandos en C#

En C# podemos procesar las líneas de comandos a través del método de entrada `Main` de la clase principal. Miremos los posibles cuerpos que puede tener una clase cualquiera con el método `Main`.

Con un argumento y sin valor de retorno:

```
0:    public class ... {
1:
2:      public static void Main(string []args) {
3:          ⋮
4:       }
5:    }
```

Con un argumento y valor de retorno:

```
0:    public class ... {
1:
2:      public static int Main(string []args) {
3:          ⋮
4:       }
5:    }
```

Sin argumento y sin valor de retorno:

```
0:    public class ... {
1:
2:      public static void Main() {
3:          ⋮
4:       }
5:    }
```

Sin argumento y con valor de retorno:

```
0:    public class ... {
1:
2:      public static int Main() {
```

```
3:          ⋮
4:       }
5:    }
```

Cuando la firma del método `Main`, tiene un argumento este es del tipo arreglo de `string`, el argumento `args` contiene los argumentos pasados en la línea de comandos.

En el siguiente ejemplo muestra los parámetros de la línea de comandos en C#:

```
 1: using System;
 2:
 3: public class CSListaDeArgumentos {
 4:
 5:   public static void Main(string[] args) {
 6:
 7:      for (int i = 0; i < args.Length; i++) {
 8:         Console.WriteLine("args[{0}] = {1}", i, args[i]);
 9:       }
10:   }
11:
12: }
```

CSListaDeArgumentos.cs

Ejercicio 8. Compile y ejecute el anterior programa con 0 o más argumentos.

Variables de ambiente

Las variables de ambiente permite a un proceso conocer información acerca de su entorno. Existen varias formas de examinar dicho ambiente, cada una de ellas depende del sistema operativo[9].

Una variable de ambiente permite que un programa pueda ser utilizado en diferentes entornos sin tener que modificar el programa fuente y luego compilarlo para poder hacerlo funcionar en un nuevo ambiente de ejecución. Esto permite que el programa sea personalizable, independiente del ambiente que establezcamos.

Unix

Las variables de ambiente, se establecen a través de varias formas, pero se examinará aquí una única forma, a través del interpretador de comandos (*shell*).

[9]No todos los sistema operativos soporta las variables de ambiente, algunos tiene mecanismos diferentes para realizar dicho trabajo

En Unix existen varios interpretadores de comandos. Cada uno de ellos tiene su propia manera de establecer las variables de ambiente. Por ejemplo en `bash`, se realiza de la siguiente forma:

```
$ tmp=valor
```

o.

```
$ export tmp=valor
```

En `tcsh`, se realiza de la siguiente forma[10]:

```
$ setenv tmp valor
```

En la anterior línea el proceso en Unix declará una variable de ambiente llamada `tmp`.
Pero como se utilizan las variables de ambiente dentro de los programas, esto se puede hacer a través de varias formas:

La función `getenv`

La función `getenv` permite obtener una variable de ambiente determinada en el sistema, para poder utilizar, esta variable de ambiente debio ser previamente establecida, como se vió en la sección anterior:

```
$ export LPRINTER=laser_sala512
```

Esto establece la variable de ambiente `LPRINTER`, podemos revisar que las variable está puesta en el sistema utilizando el comando `env`, que permite ver la variables definidas en el sistema:

```
$ env
PATH=/usr/bin:/bin:/usr/local/bin
SHELL=bash
EDITOR=emacs
LPRINTER=laser_sala512
```

Como vemos hemos establecido una variable de ambiente, ahora, como podremos utilizar dicha variable de ambiente dentro de un programa. A través de la llamada al sistema `getenv(3)`

[10] El resto de los talleres se trabajará con el intérpretador de comandos `bash`

```
#include <stdlib.h>

char *getenv(const char *nombre);
```

Esta función devuelve un puntero a posición al valor correspondiente a la variable de ambiente, o NULL si no existe dicha variable de ambiente.

Miremos el siguiente código:

```
 1:/* LVariableAmbiente.c */
 2:#include <stdio.h>
 3:#include <stdlib.h>
 4:
 5:int
 6:main(int argc, char *argv[]) {
 7:
 8:  char *env; /* Guarda la dirección de la variable */
 9:
10:  env = getenv("LPRINTER");
11:
12:  printf("%s\n", env);
13:}
```

LVariableAmbiente.c

En la línea 10, obtenemos el valor de la variable de ambiente, en la variable `env`[11] y luego imprimimos este valor en la línea 12.
El programa tendrá la siguiente salida, si la variable de ambiente `LPRINTER` esta definida:

```
$ ./LVariableAmbiente
laser_sala512
```

Ejercicio 9. Modifique el programa VariableAmbiente para que tome una lista de variables de ambiente a través de la línea de argumentos y mostrar si estas están definidas o no.

El parámetro `env`

La variables de ambiente se obtiene a través del parámetro `env` en la función `main` de un programa en C[12]. Tratemos de implementar nuestro propio comando `env` al que llamaremos `MiEnv`.

Miremos el siguiente código:

[11] Este valor no puede ser liberado, ni modificado
[12] También en C++

```
 1:/* LEnvironment.c */
 2:#include <stdio.h>
 3:
 4:int
 5:main(int argc, char *argv[], char *env[]) {
 6:
 7:  int i;
 8:
 9:  for (i = 0; env[i] != NULL; i++) {
10:    printf("%s\n", env[i]);
11:  }
12:
13:  return 0;
14:}
```

LEnvironment.c

En la línea 5, encontramos tercer parámetro que es una lista de cadenas de caracteres terminados en nulo, que contiene todas las variables de ambiente definidas para el programa que esta en ejecución. En la línea 9, vemos que la condición del programa comprueba que el valor del arreglo `env[i]` no sea nulo. No hay manera de saber de antemano cuantas variables de ambiente están definidas para el programa. En la siguiente figura se muestra la ejecución de nuestro programa.

```
$ ./LEnvironment
PATH=/usr/bin:/bin:/usr/local/bin
SHELL=bash
EDITOR=emacs
LPRINTER=laser_sala512
```

Ejercicio 10. Compile y ejecute el programa `LEnvironment`.
Ejercicio 11. Modifique el anterior programa `LEnvironment.c` para que imprima las variables de ambiente en orden alfabetico.

Algunas variables de ambientes importantes

La siguiente es una lista de algunas variables de ambiente de interés (la mayoría dependen en primer lugar del intérpretador de comandos *shell* y otras son dependientes al programa que se ejecutará):

`PATH`: Camino de búsqueda para comandos. Esta variable de ambiente contiene varios posibles caminos donde se debe buscar el programa cuando se ejecuta. Cada camino esta separado por dos puntos (:).

`PS1`: Es la cadena de caracteres que utiliza el intérprete de comandos para indicarle al usuario que esta esperando por un nuevo comando.

`HOME`: El directorio raíz del actual usuario.

Ejercicio 12. Escriba un programa que obtenga de la entrada estándar el nombre de la variable y que imprima el valor contenido de esta si existe.

Windows

Al igual que en Unix, Windows, tiene dos métodos para manipular variables de ambiente. El primer método es idéntico al método del parámetro `env`, por eso no es necesario explicar aquí. El segundo método es a través de las funciones `GetEnvironmentStrings`, `GetEnvironmentVariable` o `SetEnviromentString`.

El siguiente código muestra como ver las variables de ambiente establecidas para el proceso actual.

```
 1:/* WEnvironment.c */
 2:#include <stdio.h>
 3:#include <Windows.h>
 4:
 5:int
 6:main(int argc, char *argv[]) {
 7:
 8: LPVOID lpvEnvironment;
 9: LPTSTR lpszVar;
10:
11: lpvEnvironment = GetEnvironmentStrings();
12:
13:
14: if (lpvEnvironment) {
15:
16: for (lpszVar = lpvEnvironment;
17:  *lpszVar;
18: lpszVar++) {
19:
20: while (*lpszVar)
21: putchar(*lpszVar++);
22:
23: putchar('\n');
24: }
25: }
26:
27:    FreeEnvironmentStrings(lpvEnvironment);
28:    return 0;
29:}
```

WEnvironment.c

En la línea 8 declaramos una variable de tipo `LPVOID lpvEnvironment`, esta variable es un puntero al bloque de variables de ambiente para el programa en ejecución. Estas variables están guardadas así:

```
nombre1=valor1\0
nombre2=valor2\0
nombre3=valor3\0
.
.
.
nombreN=valorN\0
\0
```

En la línea 9 utilizamos una variable temporal para guardar el valor de un caracter dentro del bloque de variables. En la línea 11 obtenemos el bloque de variables de ambiente a través de la función de Win32 `GetEnviromentStrings`, esta retorna la dirección donde esta almacenado temporalmente el bloque de las variables de ambiente. No se puede modificar debido a que está almacenado en un área de sólo lectura. Esta dirección únicamente puede ser liberada utilizando la llamada `FreeEnvironmentStrings`, como se muestra en la línea 27. Entre la línea 14 y 28 está el código que se encarga de recorrer el bloque e imprimir cada variable de ambiente en la consola.

La otra manera de obtener el valor de una variable de ambiente es a través de la función `GetEnvironmentVariable`.

```
DWORD
GetEnvironmentVariable(LPCTSTR lpName,
                       LPTSTR lpBuffer,
                       DWORD nSize);
```

Esta función tiene tres parámetros, el primer parámetro `lpName`, indica el nombre de la variable a obtener, el parámetro `lpBuffer` es el buffer donde será almacenada la variable de ambiente si existe y `nSize`, es el tamaño de `lpBuffer`. Si la variable de ambiente existe, la función retorna el número de TCHAR que conforma la variable de ambiente, sino retorna el valor de 0.

La siguiente es la forma de establecer una variable de ambiente:

```
C:\> SET VARIABLE=2000
```

El intérpretador de comandos de Windows, permite establecer variables a través del comando interno `SET`, en el ejemplo anterior hemos establecido al variable de ambiente `VARIABLE` con el valor de 2000. En Windows, las variables de ambiente no son sensitivas a mayúsculas o minúsculas.

Ejercicio 13. Compile y ejecute el programa `WEnvironment.c`.
Ejercicio 14. . Modifique el programa `WEnvironment.c` para que imprima todas las variables de ambiente definidas para el programa en orden alfabetico.

Miremos ahora como un programa puede obtener el valor de dicha variable:

```
 1:/* WVariableAmbiente.c */
 2:#include <stdio.h>
 3:#include <Windows.h>
 4:
 5:int
 6:main(int argc, char *argv[]) {
 7:
 8: TCHAR buffer[256];
 9:
10: if (GetEnvironmentVariable("VARIABLE",
11:                     buffer,
12:                                  256)) {
13:
14: printf("VARIABLE=%s\n\r", buffer);
15: }
16: else {
17: printf("VARIABLE no esta definida");
18: }
19:
20: return 0;
21:}
```

WVariableAmbiente.c

En la línea 10 se obtiene la variable de ambiente y si esta está defindia, retorna un valor diferente a cero y se puede imprimir su valor, en caso contrario se envia un mensaje que la variable no se encuentra definida.

Variables de ambiente en Java

En el pasado usted podia utilizar `System.getenv(.`^{en}`v-var-name")` para obtener las variables de ambiente. Sin embargo, el método ha sido desaprobado (*deprecated*) este lanzara una excepción si usted intenta invocarla:

```
Exception in thread "main" java.lang.Error: getenv no longer
supported, use properties and -D instead: PATH
   at java.lang.System.getenv(System.java:677)
   at Test.main(Test.java:4)
```

`System.getProperty()` sirve ahora como la forma preferencial para obtener valores que su programa requiere. Sun decidio volver obsoleto el método debido a que las variables de ambiente son especificas a la plataforma..

Con `System.getProperty()` obtendrá las variables definidas por por el sistema de propiedades de Java. Usted puede entrar el siguiente código para determinar los valores disponibles.

```
1:   public static void main( String [] args ) {
2:      java.util.Properties p = System.getProperties();
3:      java.util.Enumeration keys = p.keys();
4:      while( keys.hasMoreElements() ) {
5:         System.out.println( keys.nextElement() );
6:      }
7:    }
```

Variables de ambiente en C#

En C# a diferencia de Java si puede leer variables del ambiente a través del método estático `GetEnvironmentVariable` de la clase `System.Environment`, este método recibe un parámetro de tipo `string` que denota el nombre de la variable deseada, si esta existe, su valor es retornado, en caso contrario se retorna un referencia nula.

En el siguiente ejemplo, se muestra como obtener el valor de cualquier variable pasada a través de la línea de comandos.

```
 1: using System;
 2:
 3: public class Test {
 4:
 5:     public static void Main(string[] args) {
 6:
 7:     if (args.Length != 1) {
 8:        Console.WriteLine("Uso: CSLeerUnaVariable <var-ambiente>");
 9:        return;
10:        }
11:
12:     string variable = Environment.GetEnvironmentVariable(args[0]);
13:
14:     Console.WriteLine("{0}={1}", args[0], variable);
15:     }
16: }
```

CSLeerUnaVariable.cs

Qué pasa si queremos conocer todas las variables de ambiente que un programa en ejecución puede tener en un momento determinado, podemos utilizar otro método estático `GetEnvironmentVariables` de la misma clase para obtener la lista de todos los valores, como se muestra en el siguiente ejemplo:

```
 1: // CSLeerVariables.cs
 2:
 3: using System;
 4: using System.Collections;
 5:
 6: public class CSLeerVariables {
 7:    public static void Main(string []args) {
 8:       IDictionary dictionary = Environment.GetEnvironmentVariables();
 9:       foreach (string value in dictionary.Keys) {
10:          Console.WriteLine("{0}={1}", value, dictionary[value]);
11:    }
12:  }
13: }
```

CSLeerVariables.cs

Recursos

Un sistema operativo es un administrador de recursos, como todo buen administrador, este no nos permitirá acceder directamente a ellos y debemos solicitar acceder al recurso. Para ello, nos dará una referencia indirecta del recurso solicitado y dicha referencia será utilizada en operaciones posteriores.

Dentro del sistema operativo los recursos son identificados sin ambigüedad, pero cuando accedemos a los servicios de un sistema operativo a través de un lenguaje de alto nivel, necesitamos una representación que sea entendia dentro del lenguaje de progamación que estemos utilizando, entonces, el identificador del recurso dependera tanto del sistema operativo, como del lenguaje de programación que estemos utilizando.

Unix

En Unix® cada recurso es identificado a través del tipo de dato entero (`int`). Cada vez que abrimos un archivo, obtenemos un valor entero que hace las veces de un recurso de archivo del sistema operativo, como en el siguiente código que creamos un socket y obtenemos un recurso de comunicación, que podemos utilizar para enviar y recibir mensajes.

```
int so; // Un socket, conexión para comunicación
so = socket(AF_INET, SOCK_STREAM, 0);
write(so, buffer, strlen(buffer));
```

Windows

En Windows® cada recurso es identificado a través de un manejador de objeto (o *Object Handle*). Un manejador de objeto es una referencia a un objeto, no el objeto mismo. Por lo tanto cuando abrimos o accedemos a un recurso en Windows, este nos entrega una referencia al recurso. Como la interfaz del Windows (Win32) está definida en C, los recurso son objetos[13] estos deben ser manejados de una manera muy especial. Se debe utilizar las instrucciones, que controlan el ciclo de vida de un objeto. Cada referencia maneja un valor interno llamado contador de referencia, que indica cuantas referencias de un recurso tiene un programa abierto, no se puede copiar una referencia a otra directamente, para ellos hay que utilizar la instrucción `DuplicateHandle`, esta se encarga de crear una referencia duplicada. Si no se va utilizar más un `Handle` hay que borrarlo explicitamente utilizando la instrucción `CloseHandle`. Recuerde siempre, si una llamada al sistema retorna un `Handle` y este no será utilizado dentro de nuestro programa debemos borrarlo llamado explicitamente `CloseHandle`.

Por ejemplo en el siguiente programa abrimos una archivo a través de la función `CreateObject` y escribimos en el algún tipo de mensaje. Los *Object Handle* están representados internamente a través de un apuntador a void, que a su vez es representado por el tipo de dato `HANDLE`.

```
DWORD dwByteEscritos; // Numero de bytes escritos

HANDLE hfile = CreateFile("Archivo.txt",
0, FILE_SHARE_WRITE,
NULL, OPEN_EXISTING, 0, NULL);

WriteFile(hfile, buffer, strlen(buffer), &dwByteEscritos,
NULL);
```

Ejercicio 15. ¿Existe un tipo de dato en C llamado `HANDLE`?, si no es así, como esta representado dicho tipo de dato[14]

Acceder a los recursos

Como ya vieron en los ejemplos anteriores para poder acceder a cualquier recursos del sistema operativo hay que utilizar las llamadas al sistema, estas son desde del punto de vista de la programación en C, funciones, como cualquier otra, como lo son la funciones de `printf()`. Pero tiene de especial que ellas llaman al sistema operativo para que realicen un labor especial que solo el sistema operativo puede realizar.

[13] Pero C no es un lenguaje de programación orientado a objetos

[14] Una posible pista puede ser utilizar el compilador de C con la opción de precompilación.

Ejercicio 16. Todas las llamadas al sistema nos indican cuando fallan, sin interrumpir el código que se está ejecutando[15] ¿Cómo puedo hacer las llamadas al sistema seguras? Muestre dos ejemplos de código, (en ambos sistemas operativos), como hacer que una llamada al sistema sea segura y si sucede algo, que nos indique que paso[16]

Manejo y reporte de errores

El sistema operativo Unix® y sus asociadas bibliotecas C ofrecen un rico conjunto de llamadas al sistema y funciones de biblioteca. Dentro de este conjunto de llamadas existen muy pocas funciones, las cuales no puedan retornar un error. Existen varias razones para que surjan los errores que incluyen el incorrecto uso de parámetros, el tamaño inadecuado de los buffers, la pérdida o el mal nombre de objetos del sistema de archivos, o simplemente la falta de acceso a un recurso. Debido a esto, un mecanismo surge para retornar las indicaciones de errores al usuario que invoca una función o una llamada al sistema.

Determinando el éxito o la falla

Cuando una función C es llamada, el programador está interesado en dos cosas al retornar esta:

- ¿Fue la llamada a la función exitosa?
- Si no, ¿Por qué la llamada falló?

Reglas generales para la indicación de errores

Las convenciones de Unix utilizadas por muchas de las llamadas y funciones de bibliotecas es que el valor de retorno indica la existencia general de éxito o falla. Los valores de retorno caen dentro de dos grandes categorías:

- El valor de retorno es un valor entero (`int` o `long`). Normalmente la falla es indicada por un valor de menos uno (-1).
- El valor de retorno es de tipo apuntador, tal como los apuntadores (char *), (void *) o un apuntador a una estructura. La falla es indicada por una apuntador de retorno nulo y el éxito por un apuntador no nulo.

Determinando las razones de una falla

La anterior discusión identifica que la gran mayoría de funciones retornan un indicador de:

- éxito,

[15] En la mayoría de las veces
[16] No vale utilizar la función perror().

- falla,
- y en casos raros, ninguna información.

Una vez que se ha descubierto que la función a fallado, usted necesita saber por qué. Por ejemplo, el comando de Unix `make(1)` necesita conocer de la llamada `open(2)` cuando esta falla.

- Fue imposible abrir un archivo `makefile` debido a que este no existe
- Faltan los permisos necesarios para abrir el `makefile` para lectura.

Las acciones para la falla *pueden* ser traídas por la acción realizada por el comando en cuestión. Por ejemplo, si este encuentra que el archivo `makefile` no existe, `make(1)` trata de abrir el archivo `Makefile`. Sin embargo, cuando este encuentra que faltan los permisos para abrir el archivo `makefile`, algunas implementaciones del `make(1)` reportan esto como un error al usuario.

La vieja variable `errno`

El método original que el programador utilizaba para obtener acceso al código de error era declarar una variable externa de tipo (int) llamada `errno`:

```
extern int errno;
```

Cuando un intento para abrir una archivo falla, un programa simplemente se puede consultar la variable externa para determinar la razón de la falla. En el siguiente ejemplo muestra como el comando `make(1)` podría ser escrito implementando el viejo método de la variable `errno`:

```
#include <errno.h>                /* Define ENOENT */
extern int errno;                 /* Código de error */
int fd;                           /* Descriptor de archivo */

/* Intento de abrir makefile */
if ((fd = open("makefile", O_RDONLY)) == -1) { /* Falla la apertura */
  if (errno == ENOENT)                        /* El archivo no existe */
     fd = open(``Makefile'', O_RDONLY));      /* No utilize Makefile */
}

if (fd == -1) {
   /* Si reporte la falla */
  ...
}
else {
   /* makefile o Makefile ha sido abierto */
}
```

Referenciando los código por nombre

Utilizar la convención de la variable externa `errno` requería que una serie de código de error sean adicionados a medida que el programa se escribía. Desde que los códigos de error numéricos podrían variar en diferentes plataformas Unix®, un conjunto de macros en C son definidas para referenciar los códigos de error (por ejemplo, el código de error `ENOMSG` es `83` para FreeBSD 3.4, `35` para HPUX y `42` para Linux). Los nombres de macros simbólicas puede ser utilizados para referenciar el mismo error en diferentes plataformas Unix®. Esa macros son definidas en el archivo de encabezados `errno.h`.

```
#include <errno.h>
```

Utilizar referencias de macros simbólicas para códigos de errores es importante, desde que ello le permite a que sus programas en C, sean portable a otras plataformas Unix®.

Examinando las fallas con valores de retorno enteros

El siguiente ejemplo indica cuando el valor de `errno` es válido:

```
extern int errno;
int fd;

if ((fd = open("makefile", O_RDONLY)) == -1) {
   /* Fallo: errno mantiene un código de error valido
  ...
}
else {
   /* Exitoso: fd mantiene un descriptor de archivo, la
      variable errno no tiene significado aquí */
}
```

Chequeando por fallas con resultado tipo puntero

```
FILE *fp = fopen(‘‘makefile’’, ‘‘r’’);

if (fp == NULL) {
   /* fopen fallo: el valor de errno mantiene un valor válido */
   ...
}
else {
   /* fopen exitoso: el valor de errno no tiene significado */
}
```

El nuevo valor `errno`

Debido a los problemas causados por la incorporación de hilos, el manejo de la variable externa `errno` se vio comprometida en mantener valores válidos. La forma de declarar la variable `errno` ha cambiado.

Declarando la variable `errno`

El nuevo valor `errno` es ahora definido de forma que es plataforma-dependiente. Esto significa que usted debe permitir que el sistema la defina por usted incluyendo únicamente le archivo `<errno.h>`. Usted no debe más declarar a `errno` como una variable entera externa.

Utilizando la nueva variable `errno`

```
 1: #include <stdio.h>
 2: #include <errno.h>
 3: #include <string.h>
 4: #include <unistd.h>
 5:
 6: int
 7: main()
 8: {
 9:   int saved_errno;
10:
11:   saved_errno = errno;
12:
13:   printf("errno = %d\n", errno);
14:
15:   errno = ENOENT;
16:   errno = 0;
17:
18:   exit(0);
19: }
```

Reportando los valores de `errno`

Cuando un error ocurre, es simple para los programadores examinar el caso especifico y actuar de acorde a ello. El problema llega a ser más complejo cuando se tiene que reportar este error al usuario. A los usuarios no les gusta memorizar códigos de error, así que un método debe existir para trasladar el código de error a mensaje entendible.

Se pueden reportar los mensajes de error de las siguientes formas:

- A través de la la función `perror(3)` para generar un mensaje de la variable `errno` y reportarla a `stderr`.

- A través del arreglo de mensajes `sys_errlist[]`
- A través de la función `strerror(3)` para retornar un mensaje para el código de error suministrado en el argumento de la función.

Ayudas

El trabajo del programador requiere de un gran esfuerzo individual que únicamente puede ser alcanzado si se cuenta con las ayudas necesarias que hagan nuestro trabajo más sencillo. Esto se logra a través de las ayudas del sistema. Pero hay que entender la idiosincrasia de cada sistema operativo.

Unix

Las ayudas se pueden obtener através del comando `man(1)`. Observe que cada vez que un libro sobre Unix informa sobre la existencia de una función o un comando siempre acompaña el nombre de un número entre paréntesis, este número indica las secciones en la cual esta dividido el manual.

Ejecute el siguiente comando:

```
$ man 1 intro
```

Este comando muestra en la pantalla la introducción a una sección específica de los manuales.

Ejercicio 17. ¿Cuántas secciones hay en linux?

Ejercicio 18. ¿Cómo funcionan las ayudas en Windows®?[17]

[17] Mire en la pagina web `http://msdn.microsoft.com`