Práctica 1

“Manejo de errores en Linux”

Sistemas Operativos

Dra. Mireya Paredes López

El objetivo de esta práctica es familiarizarse con la arquitectura del sistema operativo UNIX y específicamente con el manejo de errores de Linux. Además, el estudiante aprenderá a como programar y compilar desde la terminal de Linux utilizando el compilador GCC.

Leer el capítulo 1 del libro de Unix-Programación Avanzada y resolver los siguientes ejercicios. Los detalles de como y donde subir la práctica se encuentra en el syllabus del curso.

1. Escribir un programa que presente en el fichero estándar de salida una lista con todos los códigos de error que manejan las llamadas al sistema UNIX. Por cada código debe aparecer su número y la cadena descriptiva que el sistema le asocia. El nombre del programa será `ej1_1.c` y al ejecutarlo debe producir una salida como la siguiente:

   1: Operation not permitted
   2: No such file or directory
   3: No such process
   4: Interrupted system call
   5: Input/output error
   6: No such device or address
   7: Argument list too long
   8: Exec format error
   9: Bad file descriptor
   10: No child processes
   11: Resource temporarily unavailable
   12: Cannot allocate memory
   .
   .
   .
   132: Operation not possible due to RF-kill
   133: Memory page has hardware error
   134: (null)

2. Escribir una función de tratamiento genérico de errores para ser utilizada en los programas que se van a escribir en los capítulos posteriores. La interfaz de esta función debe ser:

```
void error (char* nfichero, int nro_linea, char* mensaje);
```

- **nfichero** es el nombre del fichero fuente desde donde se ha llamado a la función error. Utilizando la constante **__FILE__** que genera el preprocesador podemos hacer que este nombre se determine automáticamente.
- **nro_linea** es el número de línea del fichero fuente desde donde se ha llamado a la función errno. Utilizando la constante **__LINE__** generada por el preprocesador podemos hacer que este número se calcule automáticamente.
- **mensaje** es una cadena de caracteres que contiene un mensaje escrito por el usuario.

Como ejemplo, vamos a suponer que la línea 19 del fichero **prueba.c** es la siguiente:

```
error (__FILE__, __LINE__, "Error al abrir un fichero");
```

Si el último error producido es **ENOENT**, entonces la función escribirá el mensaje siguiente en el fichero estándar de salida de errores:

prueba.c (19). ERROR: ENOENT, No such file or directory. Error al abrir un fichero

La cadena **ENOENT** es el identificador del código de error. Este identificador, y no el número de error, es el empleado en el manual de UNIX cuando se describen los códigos de error que devuelven las llamadas al sistema. Por ellos, es más significativo imprimir este identificador que el número de error.

Los identificadores asociados a cada error están definidos en el fichero de cabecera **/usr/include/errno.h**. Estos identificadores se definen como constantes para el preprocesador por lo que no podemos utilizarlos directamente en el programa para imprimirlos. Para imprimirlos es necesario que sean tratados como cadenas de caracteres. Esto se puede conseguir declarando un array de la forma siguiente:

Ejemplo:

```
char errores [10][15] = { "", "EPERM", "ENOENT", "ESRCH",
"EINTR", "EIO", "ENXIO", "E2BIG", "ERFKILL", "EHWPOISON"};
```

Hacemos notar que en esta declaración cada cadena ocupa la posición que le corresponde según el número de error que tiene asociado.

La cadena **No such file or directory** es la cadena descriptiva del último error producido. Esta cadena se debe determinar consultando las variables **errno** y **sys_errlist.**

Como queremos que esta función se pueda utilizar en otros programas, se aconseja crear:

A. El fichero de cabecera HOME/include/mic.h que contenga la declaración de prototipo de **error** y de otras funciones que se creen más adelante.
B. La biblioteca(**) **HOME/lib/libmic.a** que contenga la función **error** y otras que se creen más adelante.

**Para conocer las herramientas de desarrollo del sistema UNIX es aconsejable leer primero el Apéndice B "Desarrollo de aplicaciones en el entorno UNIX". (documento anexo)

Para probar el correcto funcionamiento de la función podemos ejecutar el programa siguiente:

```
#include "mic.h"
main()
{
      extern int sys_nerr;
      int i;
      char mensaje [100];
      for(i = 0; i<sys_nerr; i++) {
            sprint(mensaje, "Error nro. %d", i);
            error(__FILE__, __LINE__, mensaje);
      }
}
```

Para que el programa pueda ser compilado es necesario que **mic.h** y **ej1_2.c** se encuentren la misma carpeta.

La forma de compilar el programa será:

$gcc ej1_2.c ../lib/limbic.a -o eje1_2

La salida que produce el programa es:

$./eje1_2

ej1_2.c (19). ERROR: EPERM, Operation not permitted. Error nro. 1

ej1_2.c (19). ERROR: ENOENT, No such file or directory. Error nro. 2

…

3. Añadir a la biblioteca **libmic.a** la función **error_fatal** que se declara a continuación

   ```
   void error_fatal (char* mensaje, int codigo_salida);
   ```

   Esta función debe imprimir en el fichero estándar de salida de errores la información siguiente:

   - Identificador asociado al código del último error producido.
   - Cadena que asocia el sistema al último error producido.
   - mensaje del usuario.

   Ejemplo, si *errno* vale 2, la llamada
   `error_fatal("No se puede abrir datos.dat", -1);`
   produce la salida:
   ERROR FATAL: ENOENT, No such file or directory. No se puede abrir datos.dat

   La función `error_fatal` termina llamando a la función `exit` y pasándole a ésta el código de salida `codigo_salida`. Por lo tanto, `error_fatal` se debe utilizar para terminar un programa y devolverle al sistema un código numérico elegido por el usuario. La interpretación de este código es responsabilidad del programador. Para probar esta función escribir el programa `eje1_3.c` donde se debe hacer una llamada a `error_Fatal`.
   La forma de compilar el programa será:

   $gcc ej1_3.c ../lib/limbic.a -o ej1_3

   ¿Dónde guarda el sistema operativo los códigos que le envía la función `error_fatal` a través de `exit`?