Práctica 2.1: Introducción a la programación de sistemas UNIX

Objetivos

En esta práctica estudiaremos el uso básico del API de un sistema UNIX y su entorno de desarrollo. En particular, se usarán funciones para gestionar errores y obtener información.

Contenidos

Preparación del entorno para la práctica
Gestión de errores
Información del sistema
Información del usuario
Información horaria del sistema

Preparación del entorno para la práctica

Esta práctica únicamente requiere el entorno de desarrollo (compilador, editores y depurador), que está disponible en las máquinas virtuales de la asignatura y en la máquina física del laboratorio.

Se puede usar cualquier editor gráfico o de terminal. Además se puede usar tanto el lenguaje C (compilador `gcc`) como C++ (compilador `g++`). Si fuera necesario compilar varios archivos, se recomienda el uso de `make`. Finalmente, el depurador recomendado en las prácticas es `gdb`. **No está permitido** el uso de IDEs como Eclipse.

Gestión de errores

Usar las funciones disponibles en el API del sistema (`perror(3)` y `strerror(3)` para gestionar los errores en los siguientes casos. En cada ejercicio añadir las librerías necesarias (`#include`).

- Strerror: devuelve el mensaje asociado a un error.
- Perror: imprime un mensaje de error.

Ejercicio 1. Añadir el código necesario para gestionar correctamente los errores generados por la llamada a `setuid(2)`. Consultar en el manual el propósito de la llamada y su prototipo.

```
int main() {
   setuid(0);
   return 1;
}
```

```
#include <sys/types.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <stdio.h>

int main(){
        int resultado;
        resultado= setuid(0);
        if (resultado == -1){
```

```
        printf("Error: ", strerror(errno));
        perror("Motivo: ");
        }
        else{
        printf("todo ok");
        }

 return 0;

}
```

Ejercicio 2. Imprimir el código de error generado por la llamada del código anterior, tanto en su versión numérica como la cadena asociada.

Añadir en el printf del error “errno”

Ejercicio 3. Escribir un programa que imprima todos los mensajes de error disponibles en el sistema. Considerar inicialmente que el límite de errores posibles es 255.

```
#include <stdio.h>

#include <errno.h>

#include <unistd.h>

#include <string.h>

int main(){

 int i;

 for(i = 0; i < 255; i++)

  printf("Error: %s\n", strerror(i), i);

 return 0;

}
```

-> una vez ejecutado se podría cambiar el 255 a 132.

Información del sistema (SI)

Ejercicio 1. El comando del sistema `uname(1)` muestra información sobre diversos aspectos del sistema. Consultar la página de manual y obtener la información del sistema.

Uname - print system information

Obtener la información del sistema: $sudo uname -a

Linux localhost.localdomain 3.10.0-862.11.6.el7.x86_64 #1 SMP Tue Aug 14 21:49:04 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux

Ejercicio 2. Escribir un programa que muestre, con `uname(2)`, cada aspecto del sistema y su valor. Comprobar la correcta ejecución de la llamada en cada caso.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/utsname.h>
#include <sys/types.h>
#include <errno.h>
#include <unistd.h>

int main(){
 int comprueba;
 struct utsname u;
 comprueba= uname(&u);
 if(comprueba == 0){
  printf("Operating system name: %s\n", u.sysname);
  printf("Node name: %s\n", u.nodename);
  printf("Operating system release: %s\n", u.release);
  printf("Operating system version: %s\n", u.version);
  printf("Hardware identifier: %s\n", u.machine);
 }
 else
  printf("Error: %s\n", errno, strerror(errno));
return 0;
}
```

Ejercicio 3. Escribir un programa que obtenga, con `sysconf(3)`, la información de configuración del sistema e imprima, por ejemplo, la longitud máxima de los argumentos, el número máximo de hijos y el número máximo de ficheros.

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
int main(){
 long longMaxArg = sysconf(_SC_ARG_MAX);
 long numMaxHijos = sysconf(_SC_CHILD_MAX);
 long numMaxFich = sysconf(_SC_OPEN_MAX);
 printf("Longitud maxima de argumentos: %li\n", longMaxArg);
 printf("Numero maximo de hijos: %li\n", numMaxHijos);
 printf("Numero maximo de ficheros: %li\n", numMaxFich);
return 0 ;
}
```

Ejercicio 4. Repetir el ejercicio anterior pero en este caso para la configuración del sistema de ficheros, con `pathconf(3)`. Por ejemplo, que muestre el número máximo de enlaces, el tamaño máximo de una ruta y el de un nombre de fichero.

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
int main(){
 long longMaxLink = sysconf(_PC_LINK_MAX);
 long longMaxPath = sysconf(_PC_PATH_MAX);
 long longMaxName = sysconf(_PC_NAME_MAX);
 printf("Numero maximo de enlaces: %li\n", longMaxLink);
 printf("Tamaño maximo de una ruta: %li\n", longMaxPath);
 printf("Tamaño maximo de un nombre de fichero %li\n", longMaxName);
return 0;
}
```

Información del usuario (SU)

Ejercicio 1. El comando `id(1)` muestra la información de usuario real y efectiva. Consultar la página de manual y comprobar su funcionamiento.

```
$man  1 id
$ id
uid=1000(cursoredes) gid=1000(cursoredes) groups=1000(cursoredes),10(wheel),983(wireshark)
```

Ejercicio 2. Escribir un programa que muestre, igual que `id`, el UID real y efectivo del usuario. ¿Cuándo podríamos asegurar que el fichero del programa tiene activado el bit *setuid*?

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
int main(){
 printf("Usuario real: %i\n", getuid());
 printf("Usuario efectivo: %i\n", geteuid());
return 0;
}
```

- ➔ Cuando el usuario real y el usuario efectivo es el mismo significa que el bit setuid está activado.

Ejercicio 3. Modificar el programa anterior para que muestre además el nombre de usuario, el directorio *home* y la descripción del usuario.

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <pwd.h>
int main(){
struct passwd *estructura;
```

```
 printf("Usuario real: %i\n", getuid());
 printf("Usuario efectivo: %i\n", geteuid());
 estructura = getpwuid(getuid());
 printf("Nombre de usuario: %s\n", estructura->pw_name);
 printf("Directorio home %s\n", estructura->pw_dir);
 printf("Descripcion de usuario: %s\n", estructura->pw_gecos);
return 0;
}
```

Información horaria del sistema (hsi)

Ejercicio 1. El comando `date(1)` muestra la hora del sistema. Consultar la página de manual y familiarizarse con los distintos formatos disponibles para mostrar la hora.

date - print or set the system date and time

Date - I

Date -R

Date -u

********info date**********

Ejercicio 2. Escribir un programa que muestre la hora, en segundos desde el Epoch, usando la función `time(2)`.

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
int main(){
 time_t t = time(NULL);
 printf("Desde el Epoch (1/1/1970) la hora es: %li\n", t);
return 0;
}
```

Ejercicio 3. Escribir un programa que mida, en microsegundos usando la función `gettimeofday(2)`, lo que tarda un bucle que incrementa una variable un millón de veces.

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

int main(){
 struct timeval tiempo;
 int rc = gettimeofday(&tiempo, NULL);
 int ini= tiempo.tv_usec; //microsegundos

 int i;
 for(i= 0; i < 1000000; ++i);

 int rc2 = gettimeofday(&tiempo, NULL);
 int fin = tiempo.tv_usec;
```

```
 printf("El bucle ha tardado: %i\n", fin - ini);

return 0;
}
```

Ejercicio 4. Escribir un programa que muestre el año usando la función `localtime(2)`.

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
int main(){
  time_t t = time(NULL);
  struct tm *informacion = localtime(&t);
  int year = informacion->tm_year;
  printf("El año es: %i\n", 1900 + year);
  return 0;
}
```

Ejercicio 5. Modificar el programa anterior para que imprima la hora de forma legible, como `"lunes, 29 de octubre de 2018, 10:34"`, usando la función `strftime(3)`.

Nota: Para establecer la configuración regional o *locale* (como idioma o formato de hora) en el programa según la configuración actual, usar la función `setlocale(3)`, por ejemplo, `setlocale(LC_ALL, "")`. Para cambiar la configuración regional, ejecutar, por ejemplo, `export LC_ALL="es_ES"`, o bien, `export LC_TIME="es_ES"`.