Práctica 2.1: Introducción a la programación de Sistemas UNIX

Objetivos
En esta práctica el alumno estudiará el uso básico y convenciones del API de un sistema UNIX y su entorno de desarrollo. En particular se usará las funciones disponibles para la gestión de errores y para obtener información.

Contenidos

Preparación del entorno para la práctica

La realización de esta práctica únicamente requiere del entorno de desarrollo (compilador, editores y utilidades de depuración). Estas herramientas están disponibles en las máquinas virtuales de la asignatura y en la máquina física de los puestos del laboratorio.

En la realización de las prácticas 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 alguna herramienta para la compilación de proyectos como `make`. Finalmente, el depurador recomendado en las prácticas es `gdb`. **No está permitido** el uso de IDEs como Eclipse.

Gestión de Errores

```
Para compilar: gcc “nombrearchivo.c” -o “archivosalida”
Para ejecutar: ./nombrearchivo
```

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

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

```
int main()
{
   /* Comprobar la ocurrencia de error y notificarlo con la llamada adecuada */
    setuid(0);
    return 1;
}
```

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

int main() {
        setuid(0);
        printf("ERROR: %s\n", strerror(errno));
        return 1;
}
```

// Si se llama a setuid() sin ser root devuelve -1 y pone el error en la variable ERRNO, que se imprime con strerror(errno).

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

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

int main() {
        setuid(0);
        printf("ERROR string: %s\nERROR code: %d\n", strerror(errno), errno);
        return 1;
}
```

Salida: ERROR string: No such file or directory ERROR code: 1

Ejercicio 3. Escribir un programa que recorra en un bucle todos los mensajes de error disponibles en el sistema y los imprima. Considerar inicialmente que el límite de errores posibles es 255. Según la salida del programa, determinar el número de errores definidos en el sistema.

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

int main() {
        int i;
        for(i = 0; i < 255; i++) {
                printf("ERROR string: %s\nERROR code: %d\n", strerror(i), i);
        }
        return 1;
}
```

Información del Sistema

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 -a
```

Ejercicio 2. Escribir un programa que muestre, claramente identificado, cada aspecto del sistema y su valor, comprobar la correcta ejecución de la llamada en cada caso. Consultar `uname(2)` para más información sobre la llamada al sistema.

```c
#include <sys/types.h>
#include <sys/utsname.h>
#include <string.h>
#include <stdio.h>

int main() {
        struct utsname mystruct;
        uname(&mystruct);
        printf("Kernel name: %s\n", mystruct.sysname);
        printf("Kernel network node hostname: %s\n", mystruct.nodename);
        printf("Kernel release: %s\n", mystruct.release);
        printf("Kernel version: %s\n", mystruct.version);
        printf("Kernel machine: %s\n", mystruct.machine);
        return 0;
}
```

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

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

int main() {
        printf("Long. Max. Argumentos: %i\n", sysconf(_SC_ARG_MAX));
        printf("Num. Max. Hijos: %i\n", sysconf(_SC_CHILD_MAX));
        printf("Num. Max. Ficheros: %i\n", sysconf(_SC_OPEN_MAX));
        return 0;
}
```

Ejercicio 4. Repetir el ejercicio anterior pero en este caso para la configuración del sistema de ficheros,

`pathconf(3)`. Por ejemplo que muestre el número máximo de enlaces, el tamaño máximo de una ruta y el de un nombre de fichero.

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

int main() {
        printf("Num. Max. Enlaces: %i\n", pathconf("/", _PC_LINK_MAX));
        printf("Tam. Max. Ruta: %i\n", pathconf("/", _PC_PATH_MAX));
        printf("Tam. Max. Nombre Fichero: %i\n", pathconf("/", _PC_NAME_MAX));
        return 0;
}
```

Información del Usuario

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

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

Ejercicio 3. Modificar el programa anterior para que se muestre además el nombre de usuario, el directorio home e información del usuario.

← Las variables GID y GROUP no existen →

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

int main() {
        printf("uid=%i(%s) gid=%i(%s) grupos=%i(%s) name=%s homedir=%s\n",
        getenv("UID"), getenv("USER"),
        getenv("GID"), getenv("USER"),
        getenv("GROUP"), getenv("USER"),
        getenv("USER"), getenv("HOME"));
        return 0;
}
```

Información Horaria del Sistema

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

Ejercicio 2. La función principal para obtener la hora del sistema es `time(2)`. Escribir un programa que obtenga la hora usando esta función y la muestre en el terminal.

```c
#include <stdio.h>
```

```c
#include <time.h>

int main () {
  time_t rawtime;
  struct tm * timeinfo;
  char buffer [80];

  time(&rawtime);
  timeinfo = localtime(&rawtime);

  strftime(buffer,80,"Now it's %d/%m/%y %H:%M",timeinfo);
  puts(buffer);

  return 0;
}
```

Ejercicio 3. Modificar el programa anterior para que muestre además la hora en formato *legible*, usando la función `ctime(3)`. ¿Dónde se reserva espacio para el valor de la cadena que devuelve la función? ¿Es necesario liberar el puntero?

Ejercicio 4. Cuando es necesario obtener la información horaria con precisión de microsegundos se puede usar `gettimeofday(2)`. Escribir un programa que mida cuánto tarda un bucle de 10.000 repeticiones en incrementar una variable en una unidad en cada iteración.

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

int main() {
        struct timeval st;
        gettimeofday(&st, NULL);
        long total;
        int i;
        for(i = 0; i < 10000; i++) {
                i = i + 1;
                total = total + st.tv_usec;
        }
        printf("Tiempo total:%i\n", total);
        return 0;
}
```

Ejercicio 5. Escribir un programa que muestre el año (p.ej. **"`Estamos en el año 1982`"**) usando la función `localtime(2)`.

```c
#include <unistd.h>
#include <stdio.h>
#include <time.h>

int main() {
          time_t rawtime;
```

```
          struct tm * timeinfo;
          char buffer [80];
          time(&rawtime);
          timeinfo = localtime(&rawtime);
          strftime(buffer,80,"Estamos en el año %Y\n",timeinfo);
          puts(buffer);
          return 0;
}
```

Ejercicio 6. Modificar el programa anterior para que usando la función `strftime(3)`, imprima además la hora en la forma: **"`Hoy es Lunes, 10:34`"**.

Nota: Para establecer la configuración regional o *locale* (como idioma o formato de hora) en el programa según la configuración actual, usar `setlocale(LC_ALL, "")`.

```
#include <unistd.h>
#include <stdio.h>
#include <time.h>
#include <locale.h>

int main() {
          time_t rawtime;
          struct tm * timeinfo;
          char buffer [80];
          setlocale(LC_ALL, "");
          time(&rawtime);
          timeinfo = localtime(&rawtime);
          strftime(buffer,80,"Estamos en el año %Y, hoy es %A,
%H:%M\n",timeinfo);
          puts(buffer);
          return 0;
}
```