

# 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

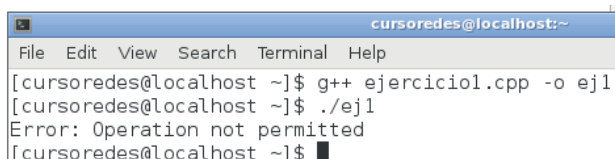
Usar las funciones disponibles en el API del sistema (perror(3) y strerror(3)) para gestionar los errores en los siguientes casos. En cada ejercicio, añadir las librerías necesarias (#include).

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

int main() {
    if(setuid(0) == -1){
        perror("Error");
    }
    return 1;
}
```

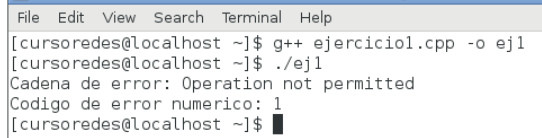


```
cursoredes@localhost:~
File Edit View Search Terminal Help
[cursoredes@localhost ~]$ g++ ejercicio1.cpp -o ej1
[cursoredes@localhost ~]$ ./ej1
Error: Operation not permitted
[cursoredes@localhost ~]$
```

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

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

int main() {
    if(setuid(0) == -1){
        printf("Cadena de error: %s\n", strerror(errno));
        printf("Codigo de error numerico: %i\n", errno);
    }
    return 1;
}
```



```
File Edit View Search Terminal Help
[cursoredes@localhost ~]$ g++ ejercicio1.cpp -o ej1
[cursoredes@localhost ~]$ ./ej1
Cadena de error: Operation not permitted
Codigo de error numerico: 1
[cursoredes@localhost ~]$
```

**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.

**PROGRAMA:**

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

int main() {
    for (int i = 0; i < 55; ++i){
        printf("Cadena asociada al error %i: %s\n", i, strerror(i));
    }
    return 1;
}
```

Lo he hecho solo hasta el 55, tan solo sería aumentar el bucle hasta 255.

**SALIDA:**

```
[cursoredes@localhost ~]$ ./ej3
```

Cadena asociada al error 0: Success

Cadena asociada al error 1: Operation not permitted

Cadena asociada al error 2: No such file or directory

Cadena asociada al error 3: No such process

Cadena asociada al error 4: Interrupted system call

Cadena asociada al error 5: Input/output error

Cadena asociada al error 6: No such device or address

Cadena asociada al error 7: Argument list too long

Cadena asociada al error 8: Exec format error

Cadena asociada al error 9: Bad file descriptor

Cadena asociada al error 10: No child processes

Cadena asociada al error 11: Resource temporarily unavailable

Cadena asociada al error 12: Cannot allocate memory

Cadena asociada al error 13: Permission denied

Cadena asociada al error 14: Bad address

Cadena asociada al error 15: Block device required

Cadena asociada al error 16: Device or resource busy

Cadena asociada al error 17: File exists

Cadena asociada al error 18: Invalid cross-device link

Cadena asociada al error 19: No such device

Cadena asociada al error 20: Not a directory

Cadena asociada al error 21: Is a directory

Cadena asociada al error 22: Invalid argument

Cadena asociada al error 23: Too many open files in system

Cadena asociada al error 24: Too many open files

Cadena asociada al error 25: Inappropriate ioctl for device

Cadena asociada al error 26: Text file busy

Cadena asociada al error 27: File too large

Cadena asociada al error 28: No space left on device

Cadena asociada al error 29: Illegal seek

Cadena asociada al error 30: Read-only file system

Cadena asociada al error 31: Too many links

Cadena asociada al error 32: Broken pipe

Cadena asociada al error 33: Numerical argument out of domain

Cadena asociada al error 34: Numerical result out of range

Cadena asociada al error 35: Resource deadlock avoided

Cadena asociada al error 36: File name too long

Cadena asociada al error 37: No locks available

Cadena asociada al error 38: Function not implemented

Cadena asociada al error 39: Directory not empty

Cadena asociada al error 40: Too many levels of symbolic links

Cadena asociada al error 41: Unknown error 41

Cadena asociada al error 42: No message of desired type

Cadena asociada al error 43: Identifier removed

Cadena asociada al error 44: Channel number out of range

Cadena asociada al error 45: Level 2 not synchronized

Cadena asociada al error 46: Level 3 halted

Cadena asociada al error 47: Level 3 reset

Cadena asociada al error 48: Link number out of range

Cadena asociada al error 49: Protocol driver not attached

Cadena asociada al error 50: No CSI structure available

Cadena asociada al error 51: Level 2 halted

Cadena asociada al error 52: Invalid exchange

Cadena asociada al error 53: Invalid request descriptor

Cadena asociada al error 54: Exchange full

## Información del sistema

**Ejercicio 4.** El comando del sistema `uname(1)` muestra información sobre diversos aspectos del sistema. Consultar la página de manual y obtener la información del sistema.

```
[cursoredes@localhost ~]$ 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 5.** Escribir un programa que muestre, con `uname(2)`, cada aspecto del sistema y su valor. Comprobar la correcta ejecución de la llamada.

**PROGRAMA:**

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

int main() {
    struct utsname s;
    uname(&s);
    printf("Sysname: %s\n", s.sysname);
    printf("Nodename: %s\n", s.nodename);
    printf("Release: %s\n", s.release);
    printf("Version: %s\n", s.version);
    printf("Machine: %s\n", s.machine);
    return 1;
}
```

**SALIDA:**

```
[cursoredes@localhost ~]$ ./ej5
Sysname: Linux
Nodename: localhost.localdomain
Release: 3.10.0-862.11.6.el7.x86_64
Version: #1 SMP Tue Aug 14 21:49:04 UTC 2018
Machine: x86_64
```

**Ejercicio 6.** Escribir un programa que obtenga, con `sysconf(3)`, 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.

**PROGRAMA:**

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

int main() {
    printf("La máxima longitud de los argumentos es :%lld\n", sysconf(_SC_ARG_MAX));
    printf("El número máximo de hijos es :%lld\n", sysconf(_SC_CHILD_MAX));
    printf("El número máximo de ficheros es :%lld\n", sysconf(_SC_OPEN_MAX));
    return 1;
}
```

**SALIDA:**

```
[cursoredes@localhost ~]$ ./ej6
La máxima longitud de los argumentos es :2097152
El número máximo de hijos es :3836
El número máximo de ficheros es :1024
```

**Ejercicio 7.** Escribir un programa que obtenga, con `pathconf(3)`, información de configuración del sistema de ficheros e imprima, por ejemplo, el número máximo de enlaces, el tamaño máximo de una ruta y el de un nombre de fichero.

**PROGRAMA:**

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

int main() {
    printf("El número máximo de enlaces es :%li\n", pathconf(".", _PC_LINK_MAX));
    printf("El tamaño máximo de la ruta es :%li\n", pathconf("/", _PC_PATH_MAX));
    printf("El tamaño maximo de nombre de fichero es :%li\n",
    pathconf(".", _PC_NAME_MAX));
    return 1;
}
```

**SALIDA:**

```
[cursoredes@localhost ~]$ ./ej7
El número máximo de enlaces es: 2147483647
El tamaño máximo de la ruta es: 4096
El tamaño maximo de nombre de fichero es: 255
```

## Información del usuario

**Ejercicio 8.** El comando `id(1)` muestra la información de usuario real y efectiva. Consultar la página de manual y comprobar su funcionamiento.

```
[cursoredes@localhost ~]$ id -a
uid=1000(cursoredes) gid=1000(cursoredes) groups=1000(cursoredes),10(wheel),983(wireshark)
```

**Ejercicio 9.** 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*?

### PROGRAMA:

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

int main() {
    printf("El usuario real es: %i\n", getuid());
    printf("El usuario efectivo es: %li\n", geteuid());
    return 1;
}
```

### SALIDA:

```
[cursoredes@localhost ~]$ ./ej9
El usuario real es: 1000
El usuario efectivo es: 1000
```

### RESPUESTA:

Lo podemos asegurar cuando el valor del real y del efectivo no coinciden.

**Ejercicio 10.** Modificar el programa anterior para que muestre además el nombre de usuario, el directorio *home* y la descripción del usuario.

### PROGRAMA

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

```
#include <unistd.h>
#include <sys/types.h>
#include <string.h>
#include <sys/utsname.h>
#include <pwd.h>

int main() {
    struct passwd *s = getpwuid(getuid());
    printf("Nombre de usuario: %s\n", s->pw_name);
    printf("Descripción del usuario: %s\n", s->pw_gecos);
    printf("Directorio home: %s\n", s->pw_dir);
    return 1;
}
```

### **SALIDA**

```
[cursoredes@localhost ~]$ ./ej10
Nombre de usuario: cursoredes
Descripción del usuario: cursoredes
Directorio home: /home/cursoredes
```

## **Información horaria del sistema**

**Ejercicio 11.** 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.

**Ejercicio 12.** Escribir un programa que muestre la hora, en segundos desde el Epoch, usando la función `time(2)`.

### **PROGRAMA:**

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

int main() {
    time_t t;
    t = time(NULL);
    printf("Tiempo en segundos desde el Epoch: %li\n", t);
    return 1;
}
```



**SALIDA:**

```
[cursoredes@localhost ~]$ ./ej12tIEMPO
Tiempo en segundos desde el Epoch: 1635174936
```

**Ejercicio 13.** 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.

**PROGRAMA:**

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

int main() {
    struct timeval tvI, tvF;
    gettimeofday(&tvI, NULL);
    for(int i = 0; i < 1000000; ++i){
    }
    gettimeofday(&tvF, NULL);
    printf("Tiempo en microsegundos que tarda un bucle en incrementar una variable un
millón de veces: %li\n", tvF.tv_usec - tvI.tv_usec);
    return 1;
}
```

**SALIDA:**

```
[cursoredes@localhost ~]$ ./ej13
Tiempo en microsegundos que tarda un bucle en incrementar una variable un millón de veces:
3086
```

**Ejercicio 14.** Escribir un programa que muestre el año usando la función `localtime(3)`.

**PROGRAMA:**

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

```
int main() {
    time_t t = time(NULL);
    struct tm * ano = localtime(&t);
    printf("Estamos en el año: %i\n", ano->tm_year + 1900);
    return 1;
}
```

**SALIDA:**

```
[cursoredes@localhost ~]$ ./ej10
Estamos en el año: 2021
```

**Ejercicio 15.** 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)`.

**PROGRAMA:**

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

int main() {
    setlocale(LC_TIME, "es_ES");
    time_t t = time(NULL);
    struct tm* t2 = localtime(&t);
    char f[1000];
    strftime(f, 1000,
        "%A, %d de %B de %Y, %H:%M:%S", t2);
    printf("Fecha y hora exacta: %s\n", f);
    return 1;
}
```

**SALIDA:**

```
[cursoredes@localhost ~]$ ./ej10
Fecha y hora exacta: lunes, 25 de octubre de 2021, 17:53:56
```

**Nota:** Para establecer la configuración regional (*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, “expo, o bien, `export LC_TIME="es_ES"`”.