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 <string.h>
#include <stdlib.h>

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

    if(setuid(0) == -1){ /*Compruebo si existe un error.*/
        /*Distingo el tipo de error con variable "errno" del sistema.*/
```

```
        if(errno == EAGAIN){
                perror("Error tipo -> EAGAIN");
                exit -1;
        }
        if(errno == EINVAL){
                perror("Error tipo -> EINVAL");
                exit -1;
        }
        if(errno == EPERM){
                perror("Error tipo -> EPERM");
                exit -1;
        }
    }
    return 0;
}
```

OUTPUT:

```
$ ./ejer1
Error tipo -> EPERM: Operation not permitted
```

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 <string.h>
#include <stdlib.h>

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

    if(setuid(0) == -1){ /*Compruebo si existe un error.*/
        /*Distingo el tipo de error con variable "errno".*/
        if(errno == EAGAIN){
                perror("Error tipo -> EAGAIN");
                exit -1;
        }
        if(errno == EINVAL){
                perror("Error tipo -> EINVAL");
                exit -1;
        }
        if(errno == EPERM){
                perror("Error tipo -> EPERM");
                exit -1;
        }

        printf("ERROR(%d): %s\n", errno, strerror(errno));
    }

    return 0;
}
```

OUTPUT:

```
$ ./ejer2
Error tipo -> EPERM: Operation not permitted
ERROR(1): Operation not permitted
```

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>
#include <stdlib.h>

const int MAX_ERRORES=255;

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

    for(int i = 0; i < MAX_ERRORES; i++){
        printf("ERROR(%d): %s\n", i, strerror(i));
    }
    return 0;
}
```

OUTPUT:

```
$ ./ejer3
ERROR(0): Success
ERROR(1): Operation not permitted
ERROR(2): No such file or directory
ERROR(3): No such process
ERROR(4): Interrupted system call
ERROR(5): Input/output error
ERROR(6): No such device or address
ERROR(7): Argument list too long
ERROR(8): Exec format error
ERROR(9): Bad file descriptor
ERROR(10): No child processes
ERROR(11): Resource temporarily unavailable
…
ERROR(134): Unknown error 134
…
ERROR(254): Unknown error 254
```

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.

```
uname()  returns  system  information  in the structure pointed to by buf.  The
utsname struct is defined in <sys/utsname.h>:

struct utsname {
   char sysname[];    /* Operating system name (e.g., "Linux") */
   char nodename[];   /* Name within "some implementation-defined network" */
   char release[];    /* Operating system release (e.g., "2.6.28") */
   char version[];    /* Operating system version */
   char machine[];    /* Hardware identifier */

#ifdef _GNU_SOURCE
   char domainname[]; /* NIS or YP domain name */
#endif
};
```

Ejercicio 5. 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.

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

int main (int argc, char *argv[]){
    struct utsname info;

    if(uname(&info) == -1){
        printf("ERROR(%d): %s\n", errno, strerror(errno));
    }
    else {
        printf("Sysname: %s\n", info.sysname);
        printf("Nodename: %s\n", info.nodename);
        printf("Release: %s\n", info.release);
        printf("Version: %s\n", info.version);
        printf("Machine: %s\n", info.machine);
    }

    return 0;
}
```

OUTPUT:
```
$ ./ejer5
Sysname: Linux
Nodename: luis
Release: 5.9.10-arch1-1
Version: #1 SMP PREEMPT Sun, 22 Nov 2020 14:16:59 +0000
Machine: x86_64
```

Ejercicio 6. 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 <sys/utsname.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

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

    printf("Longitud maxima de Argumentos: %d\n", sysconf(_SC_ARG_MAX));
    printf("Numero maximo de hijos: %d\n", sysconf(_SC_CHILD_MAX));
    printf("Numero maximo de ficheros: %d\n", sysconf(_SC_OPEN_MAX));

    return 0;
}
```

OUTPUT:
```
$ ./ejer6
Longitud maxima de Argumentos: 2097152
Numero maximo de hijos: 47336
Numero maximo de ficheros: 1024
```

Ejercicio 7. 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 <sys/utsname.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

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

```
    printf("Numero máximo de enlaces: %ld\n", pathconf(".",_PC_LINK_MAX));
    printf("Tamaño máximo de ruta: %ld\n", pathconf(".",_PC_PATH_MAX));
    printf("Tamaño máximo de nombre de fichero: %ld\n", pathconf(".",_PC_NAME_MAX));

    return 0;
}
```

OUTPUT:

```
$ ./ejer7
Numero máximo de enlaces: 65000
Tamaño máximo de ruta: 4096
Tamaño máximo de nombre de fichero: 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.

```
Consultar el manual: $man 1 id

[luis@luis gestionErrores]$ id -a
uid=1000(luis)
gid=1000(luis)
groups=1000(luis),3(sys),19(log),90(network),94(floppy),96(scanner),98(power),98
3(rfkill),985(users),986(video),988(storage),990(optical),991(lp),995(audio),998
(wheel),999(adm)
```

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*?**

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

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

    printf("ID real: %d\n", getuid());
    printf("ID efectivo: %d\n", geteuid());
```

```
    return 0;
}
```

OUTPUT:
```
$ ./ejer9
ID real: 1000
ID efectivo: 1000
```

Si el fichero de programa tiene los bits setuid o setgid activos, el EUID o el EGID del proceso creado se cambian al usuario o grupo del fichero.

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

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

int main (int argc, char *argv[]){
    uid_t id = getuid();
    printf("ID real: %d\n", id);
    printf("ID efectivo: %d\n", geteuid());

    struct passwd *p = getpwuid(id);

    printf("Nombre de usuario real: %s\n", p->pw_name);
    printf("Directorio home real: %s\n", p->pw_dir);
    printf("Descripcion del usuario real: %s\n", p->pw_gecos);

    return 0;
}
```

OUTPUT:
```
$ ./ejer10
ID real: 1000
ID efectivo: 1000
Nombre de usuario real: luis
Directorio home real: /home/luis
Descripcion del usuario real: luis pozas
```

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)`.

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

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

    time_t t = time(NULL);

    /*Mirando en man me salia: char *ctime(const time_t *timep);*/
    char *date = ctime(&t);

    printf("La hora desde el Epoch: %s\n", date);

    return 0;
}
```

OUTPUT:

```
$ ./ejer12
La hora desde el Epoch: Sat Nov 28 09:30:15 2020
```

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.

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

```
const int MAX = 1000000;

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

    long int i = 0;

    struct timeval t_ini, t_fin;
    gettimeofday(&t_ini, NULL);

    while(i < MAX){
        i++;
    }

    gettimeofday(&t_fin, NULL);

    printf("Lo que tarda un bucle que incrementa una\n");
    printf("variable %d veces: %d microsegundos\n", i, (t_fin.tv_usec -
t_ini.tv_usec));

    return 0;
}
```

OUTPUT:

```
$ ./ejer13
Lo que tarda un bucle que incrementa una
variable 1000000 veces: 3816 microsegundos
```

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

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

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

   time_t t = time(NULL);
   struct tm *info = localtime(&t);

   printf("El año actual es: %i\n", 1900 + info->tm_year);

   return 0;
}
```

OUTPUT:
$ ./ejer14
El año actual es: 2020

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)`.

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


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

    time_t t = time(NULL);
    struct tm *info = localtime(&t);

    char c[200];
    strftime(c, sizeof(c), "%A, %d %B %Y %T", info);

    /*OJO: tm_year   The number of years since 1900.*/
    printf("El año actual es: %s\n", c);

    return 0;
}
```

OUTPUT:
```
$ ./ejer15
El año actual es: Saturday, 28 November 2020 09:33:51
```

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, `export LC_ALL="es_ES"`, o bien, `export LC_TIME="es_ES"`.