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

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.

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

int main() {
   if(setuid(0) == -1){
    return -1;
   }

   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.

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

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

int main() {
  if(setuid(0) == -1){
          perror("Error");
   printf("Salida: %s", strerror(errno));
  }
  return 0;
}
```

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

int main() {

    int i;
    for (i = 0; i < 255; i++){
        printf("Error %i : %s \n", i, strerror(i));
    }

    return 0;
}
```

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 Documents]$ 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.

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

```
int main(){

    struct utsname buf;
    if(uname(&buf) == -1) {
        perror("Error");
        return -1;
    }

    else {
        printf("Sysname: %s \n", buf.sysname);
        printf("Nodename: %s \n", buf.nodename);
        printf("Release: %s \n", buf.release);
        printf("Version: %s \n", buf.version);
        printf("Machine: %s \n", buf.machine);
    }

    return 0;
}
```

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.

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

int main(){

    long value = -1;

    value = sysconf(_SC_ARG_MAX);
    if (value == -1){
        printf("Error en ARG_MAX");
    } else {
        printf("Valor de ARG_MAX: %lo \n", value);
    }

    value = sysconf(_SC_CHILD_MAX);
    if (value == -1){
        printf("Error en CHILD_MAX");
    } else {
        printf("Valor de CHILD_MAX: %lo \n", value);
    }

    value = sysconf(_SC_OPEN_MAX);
    if (value == -1){
        printf("Error en OPEN_MAX");
    } else {
```

```
        printf("Valor de OPEN_MAX: %lo \n", value);
    }

    return 0;
}
```

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.

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

int main(){

    long value = -1;

    value = pathconf(".", _PC_LINK_MAX);
    if(value == -1) {
        printf("Error: %s", strerror(errno));
    } else {
        printf("Valor de LINK_MAX: %lo \n", value);
    }

    value = pathconf(".", _PC_PATH_MAX);
    if(value == -1) {
        printf("Error: %s", strerror(errno));
    } else {
        printf("Valor de PATH_MAX: %lo \n", value);
    }

    value = pathconf(".", _PC_NAME_MAX);
    if(value == -1) {
        printf("Error: %s", strerror(errno));
    } else {
        printf("Valor de NAME_MAX: %lo \n", value);
    }


    return 0;
}
```

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.

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*?qman1qq

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

int main(){

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

    printf("Grupo real: %d\n ", getgid());
    printf("Grupo efectivo: %d\n", getegid());


    return 0;
}

Podremos asegurar que tiene activado el bit setuid cuando el EUID o EGID se cambie 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 <string.h>
#include <stdio.h>
#include <pwd.h>
#include <sys/types.h>

int main(){

    uid_t uid = getuid();

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

        printf("Grupo real: %d\n", getgid());
        printf("Grupo efectivo: %d\n", getegid());

    struct passwd *buf;
    buf = getpwuid(uid);

    printf("Nombre usuario: %s\n", buf->pw_name);
    printf("Directorio home: %s\n", buf->pw_dir);
    printf("Decripcion usuario: %s\n", buf->pw_gecos);

    return 0;
```

```
}
```

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 <time.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>

int main(){

    time_t t;
    t = time(&t);

    if(t == -1){
        perror("Error");
        return -1;
    }

    printf("Tiempo desde el Epoch: %i\n", t);

    return 0;
}
```

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/time.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>

int main(){

    struct timeval tv1, tv2;

    if(gettimeofday(&tv1, NULL) == -1){
        perror("Error:");
        return -1;
    }
```

```
    int i = 0;
    while(i < 1000000){
        i++;
    }

    if(gettimeofday(&tv2, NULL) == -1){
        perror("Error:");
        return -1;
    }

    int result = tv2.tv_usec - tv1.tv_usec;
    printf("Tiempo restante: %i\n", result);

    return 0;
}
```

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

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

int main(){

    struct tm *t_struct;

    time_t t;
    t = time(&t);

    if(t == -1){
              perror("Error");
      return -1;
       }

    t_struct = localtime(&t);

    if(t_struct == NULL){
              perror("Error");
              return -1;
       }

    printf("Año actual: %i\n", 1900 + t_struct->tm_year);


    return 0;
}
```

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 <time.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <locale.h>

int main(){

    struct tm *t_struct;

    time_t t;
    t = time(&t);

    if(t == -1){
              perror("Error");
       return -1;
        }

    t_struct = localtime(&t);

    if(t_struct == NULL){
              perror("Error");
              return -1;
        }


    setlocale(LC_ALL, "es_ES");

    char buf[100];
      strftime(buf, 100, "%A, %d de %B de %Y, %H:%M\n", t_struct);

    printf("Fecha: %s", buf);

    return 0;
}
```

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"`.