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.

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

int main() {
  char *s;
  if(setuid(0) == -1){
      perror(s);
  }
  return 1;
}
```

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

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

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

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.

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

const int MAX = 255;

int main() {

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

   return 1;
}
```

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 -a
Linux pto0813 5.8.0-63-generic #71~20.04.1-Ubuntu SMP Thu Jul 15 17:46:08 UTC 2021 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.

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

int main() {
   utsname buf;
   if(uname(&buf)==-1){
       printf("Error %d: %s\n", errno, strerror(errno));
       return -1;
   }

   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 1;
}
```

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.

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

int main() {
   if(sysconf(_SC_ARG_MAX)==-1){
       printf("Error\n");
       return -1;
   }
```

```
   printf("Max args: %ld\n", sysconf(_SC_ARG_MAX));
   printf("Max children: %ld\n", sysconf(_SC_CHILD_MAX));
   printf("Max open files: %ld\n", sysconf(_SC_OPEN_MAX));

   return 1;
}
```

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

int main() {
   if(pathconf("/", _PC_LINK_MAX)==-1){
       printf("Error\n");
       return -1;
   }

   printf("Max links: %ld\n", pathconf("/", _PC_LINK_MAX));
   printf("Max path: %ld\n", pathconf("/", _PC_PATH_MAX));
   printf("Max name: %ld\n", pathconf("/", _PC_NAME_MAX));

   return 1;
}
```

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*?

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <sys/utsname.h>
int main() {

    printf("Real uid: %d\n", getuid());
    printf("Effective uid: %d\n", geteuid());
    printf("Real gid: %d\n", getgid());
    printf("Effective gid: %d\n", getegid());
    //si uid y euid son diferentes, setuid está activado
    return 1;
}
```

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

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

int main() {

    printf("Real uid: %d\n", getuid());
    printf("Effective uid: %d\n", geteuid());
    printf("Real gid: %d\n", getgid());
    printf("Effective gid: %d\n", getegid());

    passwd *p = getpwuid(getuid());

    printf("Username: %s\n", p->pw_name);
    printf("Home: %s\n", p->pw_dir);
    printf("Info: %s\n", p->pw_gecos);
    printf("Password: %s\n", p->pw_passwd);

    return 1;
}
```

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

int main() {
    printf("Secs since Epoch %ld\n",time(NULL));

    return 1;
}
```

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 <stdio.h>
#include <time.h>
#include <unistd.h>
#include <sys/time.h>

int main() {
    timeval tv;
    gettimeofday(&tv, NULL); int start = tv.tv_usec;
    for(int i = 0; i < 1000000; ++i);
    gettimeofday(&tv, NULL); int end = tv.tv_usec;
    printf("Duracion del bucle: %li usegundos\n", end - start);
    return 1;
}
```

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

```
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <sys/time.h>

int main() {
    time_t t = time(NULL);
    tm *info = localtime(&t);
    int year = info->tm_year + 1900;
    printf("Year: %i\n",year);
    return 1;
}
```

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 <stdio.h>
#include <time.h>
#include <unistd.h>
#include <sys/time.h>

int main() {
    time_t t = time(NULL);
    tm *info = localtime(&t);
    char s[100];
    strftime(s, 100, "%A, %B %d, %H:%M", info);
    printf("Fecha: %s\n", s);
    return 1;
}
```

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"`.