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>
#include <stdio.h>
int main() {
   if (setuid(0)){
       perror("Error en setuid");
   }
   return 1;
}
Salida al ejecutar:
Error en setuid: 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 <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
int main() {
  if (setuid(0)){
        printf("Código de error: %i \n", errno);
        printf("Cadena asociada: %s \n", strerror(errno));
  }
  return 1;
}

Salida:
Código de error: 1
Cadena asociada: 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 <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
int main() {
  for (int i = 0; i < 255; ++i){
        printf("Error número %i: %s\n", i, strerror(i));
  }
  return 1;
}

Salida:
(No se pone la salida entera porque es muy larga. Hasta 133 errores. Luego saca “unknown error”
```

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.

```
Consultamos con man 1 uname
Vamos haciendo el comando uname con todas las opciones del manual -s, -n, -k…

Nombre del kernel
uname -s
Linux

Nombre del nodo en la red
uname -n
```

```
Ssoo

Nombre del kernel release
uname -r
4.15.0-24-generic

Version del kernel
uname -v
#26-Ubuntu SMP Wed Jun 13 08:44:47 UTC 2018

Hardware de la máquina
uname -m
X86_64

Tipo de procesador
uname -p
X86_64

Plataforma hardware
uname -i
x86_64

Sistema operativo
uname -o
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 en cada caso.

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/utsname.h>
#include <stdlib.h>
int main() {
  struct utsname buf;
  if(uname(&buf)){
          perror("Error en uname");
          exit(1);
  }
  printf("Sistema operativo: %s\n", buf.sysname);
  printf("Nombre de nodo: %s\n", buf.nodename);
  printf("Release: %s\n", buf.release);
  printf("Version: %s\n", buf.version);
  printf("Machine: %s\n", buf.machine);
  return 0;
}

Salida:
Sistema operativo: Linux
Nombre de nodo: ssoo
Release: 4.15.0-24-generic
Version: #26-Ubuntu SMP Wed Jun 13 08:44:47 UTC 2018
```

```
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 <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main() {
        long valor;
  if((valor = sysconf(_SC_ARG_MAX)) == -1){
        perror("Error en sysconf ARG_MAX");
        exit(1);
  }
  else printf("ARG_MAX: %li\n", valor);

  if((valor = sysconf(_SC_CHILD_MAX)) == -1){
        perror("Error en sysconf CHILD_MAX");
        exit(1);
  }
  else printf("CHILD_MAX: %li\n", valor);

  if((valor = sysconf(_SC_OPEN_MAX)) == -1){
        perror("Error en sysconf OPEN_MAX");
        exit(1);
  }
  else printf("OPEN_MAX: %li\n", valor);
  return 0;
}
Salida:
ARG_MAX: 2097152
CHILD_MAX: 15549
OPEN_MAX: 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 <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main() {
        long valor;
  if((valor = pathconf(".", _PC_LINK_MAX)) == -1){
        perror("Error en sysconf LINK_MAX");
        exit(1);
  }
```

```
  else printf("LINK_MAX: %li\n", valor);

  if((valor = pathconf(".", _PC_PATH_MAX)) == -1){
          perror("Error en sysconf PATH_MAX");
          exit(1);
  }
  else printf("PATH_MAX: %li\n", valor);

  if((valor = pathconf(".", _PC_NAME_MAX)) == -1){
          perror("Error en sysconf NAME_MAX");
          exit(1);
  }
  else printf("NAME_MAX: %li\n", valor);
  return 0;
}

Salida:
LINK_MAX: 65000
PATH_MAX: 4096
NAME_MAX: 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.

```
Comando: id
Salida: uid=1001(usuarioso) gid=1001(usuarioso) grupos=1001(usuarioso),27(sudo)

Tanto los id's reales como los efectivos son los de usuarioso (1001). Muestran también ids de los grupos donde está usuarioso.
```

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 <unistd.h>
#include <sys/types.h>
#include <stdio.h>
int main() {
  printf("User ID real: %i\n", getuid());
  printf("User ID efectivo: %i\n", geteuid());
  printf("Group ID real: %i\n", getgid());
  printf("Group ID efectivo: %i\n", getegid());
  return 0;
}
Salida:
User ID real: 1001
```

```
User ID efectivo: 1001
Group ID real: 1001
Group ID efectivo: 1001

Repuesta:
Podríamos asegurarlo cuando el UID real y el efectivo no coincidan. Se habrá puesto el UID
efectivo al del dueño 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 <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <pwd.h>
int main() {
  uid_t realUID = getuid();
  printf("User ID real: %i\n", realUID);
  struct passwd* pw;

  if((pw = getpwuid(realUID)) == NULL){
                perror("Error getpwuid");
                exit(1);
  }
  printf("Nombre de usuario: %s\n", pw->pw_name);
  printf("Home: %s\n", pw->pw_dir);
  printf("Descripcion: %s\n", pw->pw_gecos);

  printf("User ID efectivo: %i\n", geteuid());
  printf("Group ID real: %i\n", getgid());
  printf("Group ID efectivo: %i\n", getegid());

  return 0;
}

Salida:
User ID real: 1001
Nombre de usuario: usuarioso
Home: /home/usuarioso
Descripcion: Usuario SO,,,
User ID efectivo: 1001
Group ID real: 1001
Group ID efectivo: 1001
```

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.

man 1 date

Por ejemplo, el comando date +%d/%m/%y
Produce la salida:
29/11/20

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

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
	time_t tiempo = time(NULL);
	if(tiempo == (time_t) -1){
		perror("Error en time");
		exit(1);
	}
	printf("Tiempo en segundos desde Epoch: %li\n", tiempo);

  return 0;
}
```

Salida:
Tiempo en segundos desde Epoch: 1606651824

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 <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
int main() {
	struct timeval timeIni;
	if(gettimeofday(&timeIni, NULL)){
		perror("Error en gettimeofday (primera llamada)");
		exit(1);
	}
	int variable = 0;
	for (int i = 0; i < 1e6; ++i){
		variable++;
	}
  struct timeval timeFin;
```

```
        if(gettimeofday(&timeFin, NULL)){
                perror("Error en gettimeofday (segunda llamada)");
                exit(1);
        }
        printf("Segundos transcurridos: %li\n", timeFin.tv_sec - timeIni.tv_sec);
        printf("Microsegundos transcurridos: %li\n", timeFin.tv_usec - timeIni.tv_usec);
  return 0;
}

Salida:
Segundos transcurridos: 0
Microsegundos transcurridos: 1654
```

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

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
        time_t segEpoch = time(NULL);
        if(segEpoch == (time_t) -1){
                perror("Error en funcion time");
                exit(1);
        }
        struct tm* time  = localtime(&segEpoch);
        printf("Estamos en el año: %i\n", 1900 + time->tm_year);
  return 0;
}
Salida:
Estamos en el año: 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)`.

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"`.

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <locale.h>
int main() {
```

```
        time_t segEpoch = time(NULL);
        if(segEpoch == (time_t) -1){
                perror("Error en funcion time");
                exit(1);
        }
        struct tm* time  = localtime(&segEpoch);
        char* s;
        setlocale(LC_ALL, "");
        strftime(s, 200, "%A, %e de %B de %Y, %R", time);
        printf("%s\n", s);
  return 0;
}
```

Salida:
domingo, 29 de noviembre de 2020, 13:46