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 se recomienda** el uso de IDEs como Eclipse.

Gestión de errores

Usar `perror(3)` y `strerror(3)` para gestionar los errores en los siguientes casos. En cada ejercicio, añadir las librerías necesarias (con `#include`).

Ejercicio 1. Añadir el código necesario para gestionar correctamente los errores generados por `setuid(2)`. Consultar en el manual el propósito de la llamada y su prototipo.

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

int main() {
  if(setuid(0) == -1){
       perror("Ha habido un error ");
  }
  return 1;
}
```

Ejercicio 2. Imprimir el código numérico de error generado por la llamada del código anterior y el mensaje asociado.

```c
#include <stdio.h>

#include <errno.h>

#include <string.h>

#include <unistd.h>

int main() {

  if(setuid(0) == -1){

       char *s = strerror(errno);

       printf("Mensaje asociado al código de error %d: %s", errno, s);

  }

  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 <string.h>

#include <unistd.h>



int main() {

   for (int errnum = 0; errnum < 256; ++errnum) {

       char *s = strerror(errnum);

       printf("Mensaje asociado al código de error %d: %s \n", errnum, s);

   }

}
```

Salida:

…

Mensaje asociado al codigo de error 128: Key has been revoked

Mensaje asociado al codigo de error 129: Key was rejected by service

Mensaje asociado al código de error 130: Owner died

Mensaje asociado al código de error 131: State not recoverable

Mensaje asociado al código de error 132: Operation not possible due to RF-kill

Mensaje asociado al código de error 133: Memory page has hardware error

Mensaje asociado al código de error 134: Unknown error 134

Mensaje asociado al código de error 135: Unknown error 135

Mensaje asociado al código de error 136: Unknown error 136

Mensaje asociado al código de error 137: Unknown error 137

Mensaje asociado al código de error 138: Unknown error 138

Mensaje asociado al código de error 139: Unknown error 139

Mensaje asociado al código de error 140: Unknown error 140

…

Información del sistema

Ejercicio 4. Consultar la página de manual de `uname(1)` y obtener información del sistema.

Comando: uname -a

Salida: Linux pto0607 5.15.0-39-generic #42-Ubuntu SMP Thu Jun 9 23:42:32 UTC 2022 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 <stdio.h>

#include <errno.h>

#include <string.h>

#include <unistd.h>

#include <sys/utsname.h>
```

```c
#define GNU_SOURCE

int main() {

    struct utsname info;

   if(uname(&info)==-1){

       perror("Error");

   }

   else{

       printf("Nombre del SO: %s \n", info.sysname);

       printf("Nombre del host: %s \n", info.nodename);

       printf("Release del SO: %s \n", info.machine);

       printf("Version del SO: %s \n", info.version);

       printf("Hardware del SO: %s \n", info.machine);

       printf("Dominio: %s \n", info.__domainname);

   }

}
```

Salida:

Nombre del SO: Linux

Nombre del host: pto0607

Release del SO: x86_64

Version del SO: #42-Ubuntu SMP Thu Jun 9 23:42:32 UTC 2022

Hardware del SO: x86_64

Dominio: nis.lab.fdi

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 abiertos.

```c
#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <sys/utsname.h>

int main() {

   long arc_max = sysconf(_SC_ARG_MAX);
```

```
   long fich_max = sysconf(_SC_OPEN_MAX);

   long hij_max = sysconf(_SC_CHILD_MAX);

   printf("La longitud máxima de argumentos es de %ld \n", arc_max);

   printf("El número máximo de ficheros abiertos por proceso es %ld \n", fich_max);

   printf("El número máximo de procesos simultáneos es %ld \n", hij_max);

}
```

Salida:

La longitud máxima de argumentos es de 2097152

El número máximo de ficheros abiertos por proceso es 1024

El número máximo de procesos simultáneos es 62433

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 <sys/utsname.h>

int main() {

   long enl_max = pathconf(".",_PC_LINK_MAX);

   long nombre_max = pathconf(".",_PC_NAME_MAX);

   long ruta_max = pathconf(".",_PC_PATH_MAX);



   printf("La número máximo de enlaces es %ld \n", enl_max);

   printf("La longitud máxima del nombre de fichero es %ld \n", nombre_max);

   printf("La longitud máxima de la ruta relativa es %ld \n", ruta_max);

}
```

Salida:

La número máximo de enlaces es 65000

La longitud máxima del nombre de fichero es 255

La longitud máxima de la ruta relativa es 4096

Información del usuario

Ejercicio 8. Consultar la página de manual de `id(1)` y comprobar su funcionamiento.

Comando: id

Salida:

uid=1565(usuario_vms) gid=100(users)grupos=100(users), 20(dialout), 24(cdrom), 25(floppy), 29(audio), 30(dip), 44(video), 46(plugdev), 108(netdev), 113(bluetooth), 118(scanner), 126(sambashare), 132(render), 138(vboxusers), 1025(vmwareusers)

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*?

```c
#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

int main() {

   uid_t id = getuid(), eid = geteuid();

   printf("El identificador de usuario es %d y el usuario efectivo %d \n", id, eid);

   return 1;

}
```

Salida:

El identificador de usuario es 1565 y el usuario efectivo 1565

¿Cuándo podríamos asegurar que el fichero del programa tiene activado el bit setuid?

Cuando el uid y euid no coinciden.

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

```c
#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <pwd.h>
```

```c
int main() {

   uid_t id = getuid();

   struct passwd *a;

   a = getpwuid(id);

   printf("Nombre de usuario %s \n", a->pw_name);

   printf("Directorio 'home' %s \n", a->pw_dir);

   printf("Descripción del usuario %s \n", a->pw_gecos);

   return 1;

}
```

Salida:

Nombre de usuario usuario_vms

Directorio 'home' /home/usuario_vms

Descripción del usuario Usuario VMs

Información horaria del sistema

Ejercicio 11. Consultar la página de manual de `date(1)` y familiarizarse con los distintos formatos disponibles para mostrar la hora.

date -I

2022-11-07

date -R

Mon, 07 Nov 2022 12:15:27 +0100

date --rfc-3339=date

2022-11-07

date --rfc-3339=seconds

2022-11-07 12:16:39+01:00

date --rfc-3339=ns

2022-11-07 12:16:46.422587048+01:00

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

```c
#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <time.h>

int main() {

   printf("Tiempo en segundos desde el EPOCH: %li \n", time(NULL));

   return 1;

}
```

Salida:

Tiempo en segundos desde el EPOCH: 1667819977

Ejercicio 13. Escribir un programa que mida, en microsegundos, lo que tarda un bucle que incrementa una variable un millón de veces usando `gettimeofday(2)`.

```c
#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <time.h>

#include <sys/time.h>

int main() {

   struct timeval ini, fin;

   long int a = 0;

   gettimeofday(&ini, NULL);

   for(int i = 0; i < 1000000; ++i)

       a  += 1;

   gettimeofday(&fin, NULL);

   printf("El bucle tarda %li microsegundos \n", fin.tv_usec - ini.tv_usec);

   return 1;

}
```

Salida:

El bucle tarda 4297 microsegundos

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

```
#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <time.h>

#include <sys/time.h>

int main() {

   time_t ti = time(NULL);

   struct tm *t;

   t = localtime(&ti);

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

   return 1;

}
```

Salida:

Año 2022

Ejercicio 15. Modificar el programa anterior para que imprima la hora de forma legible, como `"lunes, 29 de octubre de 2018, 10:34"`, usando `strftime(3)`.

```
#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <time.h>

#include <sys/time.h>

#include <locale.h>

int main() {

   time_t ti = time(NULL);

   struct tm *t = localtime(&ti);

   setlocale(LC_ALL, "");

   char* s;

   size_t a = strftime(s, 250, "%A, %d de %B de %Y, %H:%M \n", t);
```

```
    printf("%s \n", s);

    return 1;

}
```

Salida:

lunes, 7 de noviembre de 2022, 12:44

Nota: Para establecer la configuración regional (*locale*, como idioma o formato de hora) en el programa según la configuración actual, usar `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"`.