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](#_14a3ftqman5y)

[Gestión de errores](#_dngqig2y9aib)

[Información del sistema](#_d769h3pfq9ne)

[Información del usuario](#_rg73kma1zd4y)

[Información horaria del sistema](#_m7n4o8t1iyye)

# 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() {  char \*s;  if(setuid(0) < 0) {  perror();  }  **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.

|  |
| --- |
| #include <stdio.h>  #include <errno.h>  #include <unistd.h>  #include <string.h>  int main(){  if (setuid(0) < 0){  printf("ERROR(%d): %s\n", errno, strerror(errno));  }  return 1;  }  [cursoredes@localhost Prac1]$ ./ej2  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>  int main(){  int i;  for(i = 1; i <= 255; ++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.

|  |
| --- |
| [cursoredes@localhost Prac1]$ 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 en cada caso.

|  |
| --- |
| include <stdio.h>  #include <unistd.h>  #include <string.h>  #include <sys/utsname.h>  int main(){  struct utsname unameData;  uname(&unameData);  printf("system name = %s\n", unameData.sysname);  printf("node name = %s\n", unameData.nodename);  printf("release = %s\n", unameData.release);  printf("version = %s\n", unameData.version);  printf("machine = %s\n", unameData.machine);  return 1;  }  [cursoredes@localhost Prac1]$ ./ej5  system name = Linux  node name = localhost.localdomain  release = 3.10.0-862.11.6.el7.x86\_64  version = #1 SMP Tue Aug 14 21:49:04 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 <stdio.h>  #include <unistd.h>  #include <string.h>  int main(){  printf("longitud máxima de los argumentos: %d\n", sysconf(\_SC\_ARG\_MAX));  printf("número máximo de hijos: %d\n", sysconf(\_SC\_CHILD\_MAX));  printf("número máximo de ficheros %d\n", sysconf(\_SC\_OPEN\_MAX));  return 1;  }  [cursoredes@localhost Prac1]$ ./ej6  longitud máxima de los argumentos: 2097152  número máximo de hijos: 3836  número máximo 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 <stdio.h>  #include <unistd.h>  #include <string.h>  int main(){  printf("número máximo de enlaces: %d\n", pathconf("/", \_PC\_LINK\_MAX));  printf("tamaño máximo de una ruta: %d\n", pathconf("/", \_PC\_PATH\_MAX));  printf("nombre de fichero: %d\n", pathconf("/", \_PC\_NAME\_MAX));  return 1;  }  [cursoredes@localhost Prac1]$ ./ej7  número máximo de enlaces: 2147483647  tamaño máximo de una ruta: 4096  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.

|  |
| --- |
| [cursoredes@localhost Prac1]$ id -a  uid=1000(cursoredes) gid=1000(cursoredes) groups=1000(cursoredes),10(wheel),983(wireshark)  (man id) |

***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 <unistd.h>  #include <string.h>  int main(){  printf("uid real = %i\n", getuid());  printf("uid efectiva = %i\n", geteuid());  return 1;  }  [cursoredes@localhost Prac1]$ ./ej9  uid real = 1000  uid efectiva = 1000  En caso del que UID real y efectivo del usuario sean distintos. |

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

|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <string.h>  #include <pwd.h>  int main(){  printf("uid real = %i\n", getuid());  printf("uid efectiva = %i\n", geteuid());  int uid = getuid();  struct passwd \*pass = getpwuid(uid);  printf("Username: %s\n", pass->pw\_name);  printf("Home: %s\n", pass->pw\_dir);  printf("Info: %s\n", pass->pw\_gecos);  return 1;  }  [cursoredes@localhost Prac1]$ ./ej10  uid real = 1000  uid efectiva = 1000  Username: cursoredes  Home: /home/cursoredes  Info: cursoredes |

# 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.

|  |
| --- |
| [cursoredes@localhost Prac1]$ date  Wed Dec 2 15:35:38 CET 2020  (man date) |

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

|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <string.h>  #include <time.h>  int main(){  time\_t segundos = time(NULL);  printf("segundos desde el Epoch: %d\n", segundos);  return 1;  }  [cursoredes@localhost Prac1]$ ./ej12  segundos desde el Epoch: 1606920226 |

***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 <unistd.h>  #include <string.h>  #include <sys/time.h>  int main(){  struct timeval tv1, tv2; //st.tv\_usec para milisegundos  gettimeofday(&tv1, NULL);  int start = tv1.tv\_usec;  int i;  for(i = 0; i < 1000000; ++i){  i += 1;  }  gettimeofday(&tv2, NULL);  int end = tv2.tv\_usec;  printf("Ha tardado en incrementar la variable: %d\n", end - start);  return 1;  }  [cursoredes@localhost Prac1]$ ./ej13  Ha tardado en incrementar la variable: 2017 |

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

|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <string.h>  #include <time.h>  int main(){  time\_t t = time(NULL);  struct tm \*year = localtime(&t);  printf("Año actual: %d\n", 1900 + year->tm\_year);  return 1;  }  [cursoredes@localhost Prac1]$ ./ej14  Año actual: 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 <stdio.h>  #include <unistd.h>  #include <string.h>  #include <time.h>  #include <locale.h>  int main(){  char buf[80];  time\_t t = time(NULL);  struct tm \*date = localtime(&t);  setlocale(LC\_ALL, "es\_ES");  strftime(buf, 80, "%A, %d de %B, %H:%M", date);  printf("Hoy estamos en el año %d, %s\n", date->tm\_year + 1900, buf);  return 1;  }  [cursoredes@localhost Prac1]$ ./ej15  Hoy estamos en el año 2020, miércoles, 02 de diciembre, 17:11 |

***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".