

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
- Gestión de errores
- Información del sistema
- Información del usuario
- Información horaria del sistema

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`).

[Primero consultamos el manual de:](#)

* `perror(3)`: # man 3 perror

```

sansen@MSI: ~/ASOR/SO
Linux Programmer's Manual
PERROR(3)

NAME
    perror - print a system error message

SYNOPSIS
    #include <stdio.h>

    void perror(const char *s);

    #include <errno.h>

    const char * const sys_errlist[];
    int sys_nerr;
    int errno; /* Not really declared this way; see errno(3) */

    Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

    sys_errlist, sys_nerr:
        Since glibc 2.19:
            _DEFAULT_SOURCE
        Glibc 2.19 and earlier:
            _BSD_SOURCE

DESCRIPTION
    The perror() function produces a message on standard error describing the last error encountered during a call to a system or library function.

    First (if s is not NULL and *s is not a null byte ('\0')), the argument string s is printed, followed by a colon and a blank. Then an error message corresponding to the current value of errno and a new-line.

    To be of most use, the argument string should include the name of the function that incurred the error.
  
```

* `strerror(3)`: # man 3 strerror

```

sansen@MSI: ~/ASOR/SO
Linux Programmer's Manual
STRERROR(3)

NAME
    strerror, strerror_r, strerror_l - return string describing error number

SYNOPSIS
    #include <string.h>

    char *strerror(int errnum);

    int strerror_r(int errnum, char *buf, size_t buflen);
    /* XSI-compliant */

    char *strerror_r(int errnum, char *buf, size_t buflen);
    /* GNU-specific */

    char *strerror_l(int errnum, locale_t locale);

    Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

    strerror_r():
        The XSI-compliant version is provided if:
            (_POSIX_C_SOURCE >= 200112L) && ! _GNU_SOURCE
        Otherwise, the GNU-specific version is provided.

DESCRIPTION
    The strerror() function returns a pointer to a string that describes the error code passed in the argument errnum, possibly using the LC_MESSAGES part of the current locale to select the appropriate language. (For example, if errnum is EINVAL, the returned description will be "Invalid argument".) This string must not be modified by the application, but may be modified by a subsequent call to strerror() or strerror_l(). No other library function, including perror(3), will modify this string.
  
```

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.

[Consultamos el manual de la llamada `setuid\(2\)`.](#)

Comando: **# man 2 setuid**

```
sansan@MSI: ~/ASOR/SO
SETUID(2) Linux Programmer's Manual SETUID(2)

NAME
    setuid - set user identity

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

    int setuid(uid_t uid);

DESCRIPTION
    setuid() sets the effective user ID of the calling process. If the calling process is privileged (more precisely: if the process has the CAP_SETUID capability in its user namespace), the real UID and saved set-user-ID are also set.

    Under Linux, setuid() is implemented like the POSIX version with the _POSIX_SAVED_IDS feature. This allows a set-user-ID (other than root) program to drop all of its user privileges, do some un-privileged work, and then reengage the original effective user ID in a secure manner.

    If the user is root or the program is set-user-ID-root, special care must be taken: setuid() checks the effective user ID of the caller and if it is the superuser, all process-related user ID's are set to uid. After this has occurred, it is impossible for the program to regain root privileges.

    Thus, a set-user-ID-root program wishing to temporarily drop root privileges, assume the identity of an un-privileged user, and then regain root privileges afterward cannot use setuid(). You can accomplish this with seteuid(2).
```

```
sansan@MSI:~/ASOR/SO/P1/Gestion_Errores$ ./ejer1
Operation not permitted
```

```
int main() {
    setuid(0);
    return 1;
}
```

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

```
sansan@MSI:~/ASOR/SO/P1/Gestion_Errores$ ./ejer2
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.

```
sansan@MSI:~/ASOR/SO/P1/Gestion_Errores$ ./ejer3
ERROR(0): Success
ERROR(1): Operation not permitted
ERROR(2): No such file or directory
ERROR(3): No such process
ERROR(4): Interrupted system call
ERROR(5): Input/output error
ERROR(6): No such device or address
ERROR(7): Argument list too long
ERROR(8): Exec format error
ERROR(9): Bad file descriptor
ERROR(10): No child processes
ERROR(11): Resource temporarily unavailable
ERROR(12): Cannot allocate memory
ERROR(13): Permission denied
ERROR(14): Bad address
ERROR(15): Block device required
ERROR(16): Device or resource busy
ERROR(17): File exists
ERROR(18): Invalid cross-device link
ERROR(19): No such device
ERROR(20): Not a directory
ERROR(21): Is a directory
ERROR(22): Invalid argument
ERROR(23): Too many open files in system
ERROR(24): Too many open files
ERROR(25): Inappropriate ioctl for device
ERROR(26): Text file busy
ERROR(27): File too large
ERROR(28): No space left on device
ERROR(29): Illegal seek
ERROR(30): Read-only file system
```

Información del sistema

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

Comando: # `man 1 uname`

```
sansan@MSI: ~/ASOR/SO/P1/Gestion_Errores
UNAME(1) User Commands

NAME
  uname - print system information

SYNOPSIS
  uname [OPTION]...

DESCRIPTION
  Print certain system information.  With no OPTION, same as -s.

  -a, --all
    print all information, in the following order, except omit -p and -i if unknown:

  -s, --kernel-name
    print the kernel name

  -n, --nodename
    print the network node hostname

  -r, --kernel-release
    print the kernel release

  -v, --kernel-version
    print the kernel version

  -m, --machine
    print the machine hardware name

  -p, --processor
    print the processor type (non-portable)

  -i, --hardware-platform
    print the hardware platform (non-portable)

  -o, --operating-system
    print the operating system

  --help display this help and exit
```

Comando: # `man 2 uname`

```
sansan@MSI: ~/ASOR/SO/P1/Gestion_Errores
UNAME(2) Linux Programmer's Manual

NAME
  uname - get name and information about current kernel

SYNOPSIS
  #include <sys/utsname.h>

  int uname(struct utsname *buf);

DESCRIPTION
  uname() returns system information in the structure pointed to by buf.  The utsname struct is defined in <sys/utsname.h>:

  struct utsname {
    char sysname[]; /* Operating system name (e.g., "Linux") */
    char nodename[]; /* Name within "some implementation-defined
                     network" */
    char release[]; /* Operating system release (e.g., "2.6.28") */
    char version[]; /* Operating system version */
    char machine[]; /* Hardware identifier */
    #ifdef _GNU_SOURCE
    char domainname[]; /* NIS or YP domain name */
    #endif
  };

  The length of the arrays in a struct utsname is unspecified (see NOTES); the fields are terminated by a null byte ('\0').

RETURN VALUE
  On success, zero is returned.  On error, -1 is returned, and errno is set appropriately.
```

Ejercicio 5. Escribir un programa que muestre, con `uname(2)`, cada aspecto del sistema y su valor. Comprobar la correcta ejecución de la llamada.

```
sansan@MSI:~/ASOR/SO/P1/Informacion_Sistema$ ./ejer5
Nombre del Sistema: Linux
Nodename: MSI
Release: 5.10.16.3-microsoft-standard-WSL2
Version: #1 SMP Fri Apr 2 22:23:49 UTC 2021
Machine: x86_64
```

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.

```
sansan@MSI:~/ASOR/SO/P1/Informacion_Sistema$ ./ejer6
Longitud máxima de los argumentos: 2097152
Número máximo de hijos: 50169
Número máximo de ficheros abiertos: 4096
Número máximo de ficheros abiertos: 4096
```

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.

[Como path utilizamos el directorio actual](#)

```
sansan@MSI:~/ASOR/SO/P1/Informacion_Sistema$ ./ejer7
Número máximo de enlaces: 65000
Tamaño máximo de una ruta: 4096
Nombre de un fichero: 255
sansan@MSI:~/ASOR/SO/P1/Informacion_Sistema$
```

Información del usuario

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

```
sansan@MSI: ~/ASOR/SO/P1/Gestion_Errores
ID(1) User Commands

NAME
  id - print real and effective user and group IDs

SYNOPSIS
  id [OPTION]... [USER]

DESCRIPTION
  Print user and group information for the specified USER, or (when USER omitted) for the current user.

  -a      ignore, for compatibility with other versions
  -Z, --context
          print only the security context of the process
  -g, --group
          print only the effective group ID
  -G, --groups
          print all group IDs
  -n, --name
          print a name instead of a number, for -ugG
  -r, --real
          print the real ID instead of the effective ID, with -ugG
  -u, --user
          print only the effective user ID
  -Z, --zero
          delimit entries with NUL characters, not whitespace;
          not permitted in default format
  --help
          display this help and exit
```

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*?

```
sansan@MSI:~/ASOR/SO/P1/Informacion_Usuario$ ./ejer9
UID Real del Usuario: 1000
UID Efectivo del Usuario: 1000
```

El bit *setuid* es asignable a ficheros ejecutables, y permite que cuando un usuario ejecute dicho fichero, el proceso adquiera los permisos del propietario del fichero ejecutado.

Podemos ver que el bit está asignado (s) haciendo un ls:

```
sansan@MSI:~$ ls -l /bin/su
-rwsr-xr-x 1 root root 67816 Feb  7 2022 /bin/su
sansan@MSI:~$
```

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

```
sansan@MSI: ~/ASOR/SO/P1/Gestion_Errores
GETPWENT(3) Linux Programmer's Manual GETPWENT(3)

NAME
  getpwnam, getpwnam_r, getpwuid, getpwuid_r - get password file entry

SYNOPSIS
  #include <sys/types.h>
  #include <pwd.h>

  struct passwd *getpwnam(const char *name);
  struct passwd *getpwuid(uid_t uid);

  int getpwnam_r(const char *name, struct passwd *pwd,
                char *buf, size_t buflen, struct passwd **result);
  int getpwuid_r(uid_t uid, struct passwd *pwd,
                char *buf, size_t buflen, struct passwd **result);

  Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

  getpwnam_r(), getpwuid_r():
    _POSIX_C_SOURCE
    || /* glibc versions <= 2.19: */ _BSD_SOURCE || _SVID_SOURCE

DESCRIPTION
  The getpwnam() function returns a pointer to a structure containing the broken-out fields of the record in the password database (e.g., the local password file /etc/passwd, NIS, and LDAP) that matches the username name.

  The getpwuid() function returns a pointer to a structure containing the broken-out fields of the record in the password database that matches the user ID uid.

  The passwd structure is defined in <pwd.h> as follows:

  struct passwd {
    char *pw_name;      /* username */
    char *pw_passwd;    /* user password */
    uid_t pw_uid;       /* user ID */
    gid_t pw_gid;       /* group ID */
    char *pw_gecos;     /* user information */
    char *pw_dir;       /* home directory */
```

```
sansan@MSI:~/ASOR/SO/P1/Informacion_Usuario$ ./ejer10
UID Real del Usuario: 1000
UID Efectivo del Usuario: 1000
Nombre de Usuario: sansan
Directorio Home: /home/sansan
Info: , , ,
```

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.

```
sansan@MSI: ~/ASOR/SO/P1/Gestion_Errores
DATE(1) User Commands

NAME
    date - print or set the system date and time

SYNOPSIS
    date [OPTION]... [+FORMAT]
    date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]

DESCRIPTION
    Display the current time in the given FORMAT, or set the system date.

    Mandatory arguments to long options are mandatory for short options too.
```

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

```
sansan@MSI: ~/ASOR/SO/P1/Gestion_Errores
TIME(2) Linux Programmer's Manual

NAME
    time - get time in seconds

SYNOPSIS
    #include <time.h>

    time_t time(time_t *tloc);

DESCRIPTION
    time() returns the time as the number of seconds since the Epoch, 1970-01-01 00:00:00 +0000 (UTC).

    If tloc is non-NULL, the return value is also stored in the memory pointed to by tloc.

RETURN VALUE
    On success, the value of time in seconds since the Epoch is returned. On error, ((time_t) -1) is returned, and errno is set appropriately.
```

```
sansan@MSI:~/ASOR/SO/P1/Informacion_Horaria_Sistema$ ./ejer12
Seconds since 1970-01-01 00:00:00 +0000 (UTC): 1667736312
sansan@MSI:~/ASOR/SO/P1/Informacion_Horaria_Sistema$
```

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)`.

```
sansan@MSI:~/ASOR/SO/P1/Informacion_Horaria_Sistema$ ./ejer13
El bucle ha tardado (ms): 2798
sansan@MSI:~/ASOR/SO/P1/Informacion_Horaria_Sistema$
```

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

```
sansan@MSI:~/ASOR/SO/P1/Gestion_Errores
TIME(3) Linux Programmer's Manual

NAME
    asctime, ctime, gmtime, localtime, mktime, asctime_r, ctime_r, gmtime_r, localtime_r - transform date and time to broken-down time or ASCII

SYNOPSIS
    #include <time.h>

    char *asctime(const struct tm *tm);
    char *asctime_r(const struct tm *tm, char *buf);

    char *ctime(const time_t *timep);
    char *ctime_r(const time_t *timep, char *buf);

    struct tm *gmtime(const time_t *timep);
    struct tm *gmtime_r(const time_t *timep, struct tm *result);

    struct tm *localtime(const time_t *timep);
    struct tm *localtime_r(const time_t *timep, struct tm *result);

    time_t mktime(struct tm *tm);

Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

    asctime_r(), ctime_r(), gmtime_r(), localtime_r():
        _POSIX_C_SOURCE
        || /* Glibc versions <= 2.19: */ _BSD_SOURCE || _SVID_SOURCE

DESCRIPTION
    The ctime(), gmtime() and localtime() functions all take an argument of data type time_t, which represents calendar time. When interpreted as an absolute time value, it represents the number of seconds elapsed since the Epoch, 1970-01-01 00:00:00 +0000 (UTC).

    The asctime() and mktime() functions both take an argument representing broken-down time, which is a representation separated into year, month, day, and so on.
```

```
sansan@MSI:~/ASOR/SO/P1/Informacion_Horaria_Sistema$ ./ejer14
Estamos en el 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)`.

```
sansan@MSI: ~/ASOR/SO/P1/Gestion_Errores
STRFTIME(3) Linux Programmer's Manual STRFTIME(3)

NAME
    strftime - format date and time

SYNOPSIS
    #include <time.h>

    size_t strftime(char *s, size_t max, const char *format,
                    const struct tm *tm);

DESCRIPTION
    The strftime() function formats the broken-down time tm according to the format specification format and places the result in the character array s of size max. The broken-down time structure tm is defined in <time.h>. See also ctime(3).

    The format specification is a null-terminated string and may contain special character sequences called conversion specifications, each of which is introduced by a '%' character and terminated by some other character known as a conversion specifier character. All other character sequences are ordinary character sequences.

    The characters of ordinary character sequences (including the null byte) are copied verbatim from format to s. However, the characters of conversion specifications are replaced as shown in the list below. In this list, the field(s) employed from the tm structure are also shown.
```

```
sansan@MSI:~/ASOR/SO/P1/Informacion_Horaria_Sistema$ ./ejer15
Hoy estamos a Sunday, 06 de November de 2022, 13:46
sansan@MSI:~/ASOR/SO/P1/Informacion_Horaria_Sistema$
```

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"`.