TEMA 2.1. Introducción a la Programación de Sistemas

PROFESORES:

Rubén Santiago Montero

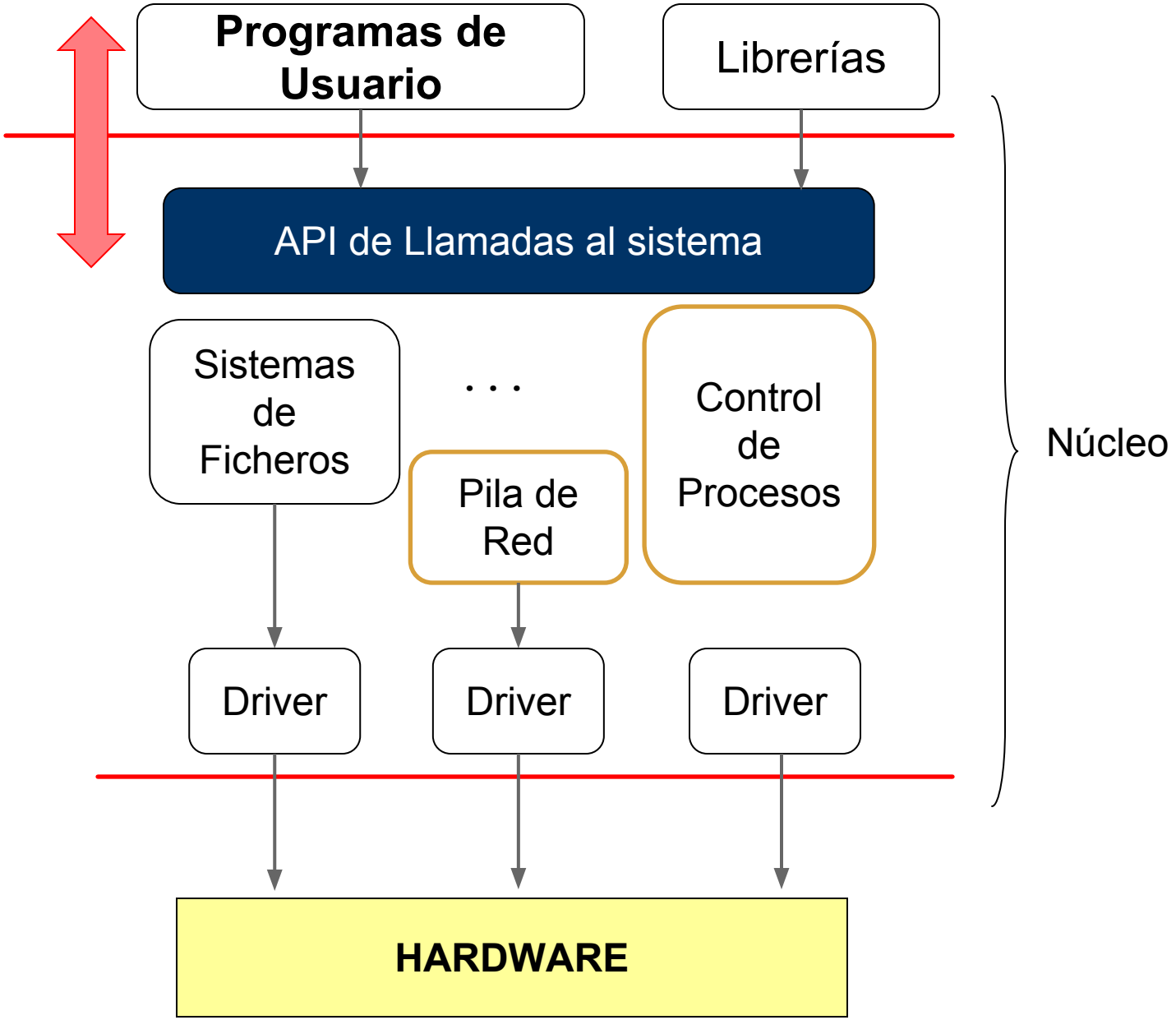
Eduardo Huedo Cuesta

OTROS AUTORES:

Ignacio Martín Llorente

Juan Carlos Fabero Jiménez

Introducción: Arquitectura del sistema



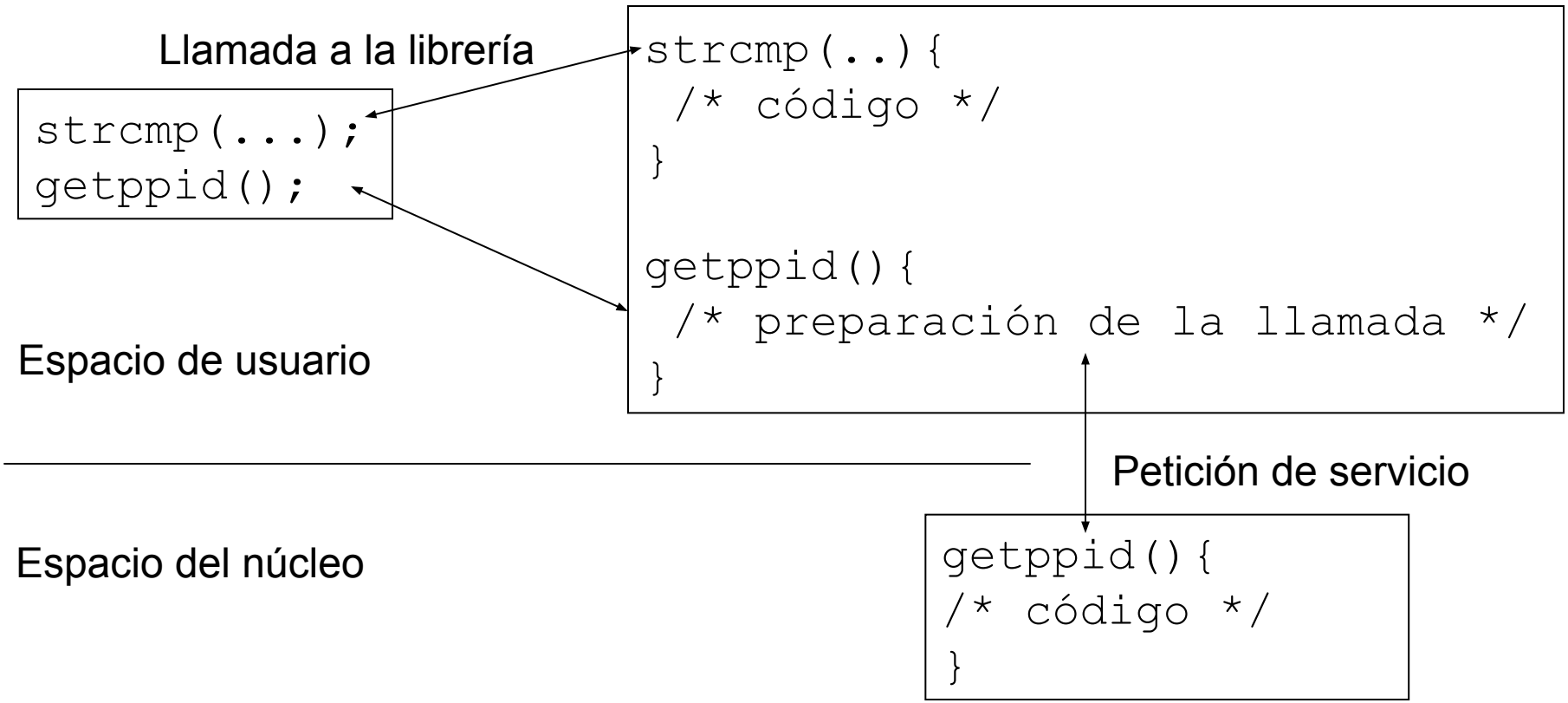
Introducción: Estándares de Programación

- **ANSI-C o ISO-C**: Estándar de programación adoptado por ANSI (*American National Standards Institute*) y posteriormente por ISO (*International Standardization Organization*). Es el estándar más general. La opción `-ansi` hace que el compilador lo cumpla de forma estricta.
- **BSD** (*Berkeley Software Distribution*): Desarrollado durante los 80 en la Universidad de California Berkeley. Sus contribuciones más importantes son los enlaces simbólicos, los sockets, la función `select`…
- **SVID** (*System V Interface Definition*): Descripción formal de las distribuciones comerciales de UNIX de la compañía AT&T, como System V Release 4 (SVr4). Su principal contribución son los mecanismos IPC.
- **POSIX** (*Portable Operating System Interface*): Estándares IEEE e ISO derivados de varias versiones de UNIX, principalmente de SVID. Incluye ANSI-C. Describe llamadas al sistema y librerías de C, especifica la semántica detallada de la *shell* y un conjunto mínimo de comandos, así como interfaces detallados para varios lenguajes de programación.
- **GNU** (*GNU's Not Unix*): Sistema operativo de tipo UNIX de software libre con licencia GNU GPL (*General Public License*). La combinación del software GNU y el kernel de Linux es GNU/Linux.

Llamadas al Sistema y Librerías

Desde el punto de vista del programador no existe ninguna diferencia. Sin embargo:

- Una llamada al sistema es un función de la librería C que solicita un servicio del sistema (*trap*). Esta petición se resuelve en el núcleo del sistema operativo
- Una llamada a una librería estándar no interacciona de forma directa con el sistema (debe usar llamadas al sistema)



Llamadas al Sistema y Librerías

	Llamadas al Sistema	Llamadas a Librerías
Sección de manual	2	3
Área de ejecución	Usuario/Kernel	Usuario
Espacio de parámetros	No se reserva	Dinámico/Estático
Código de error	-1 + `errno`	NULL + no `errno`

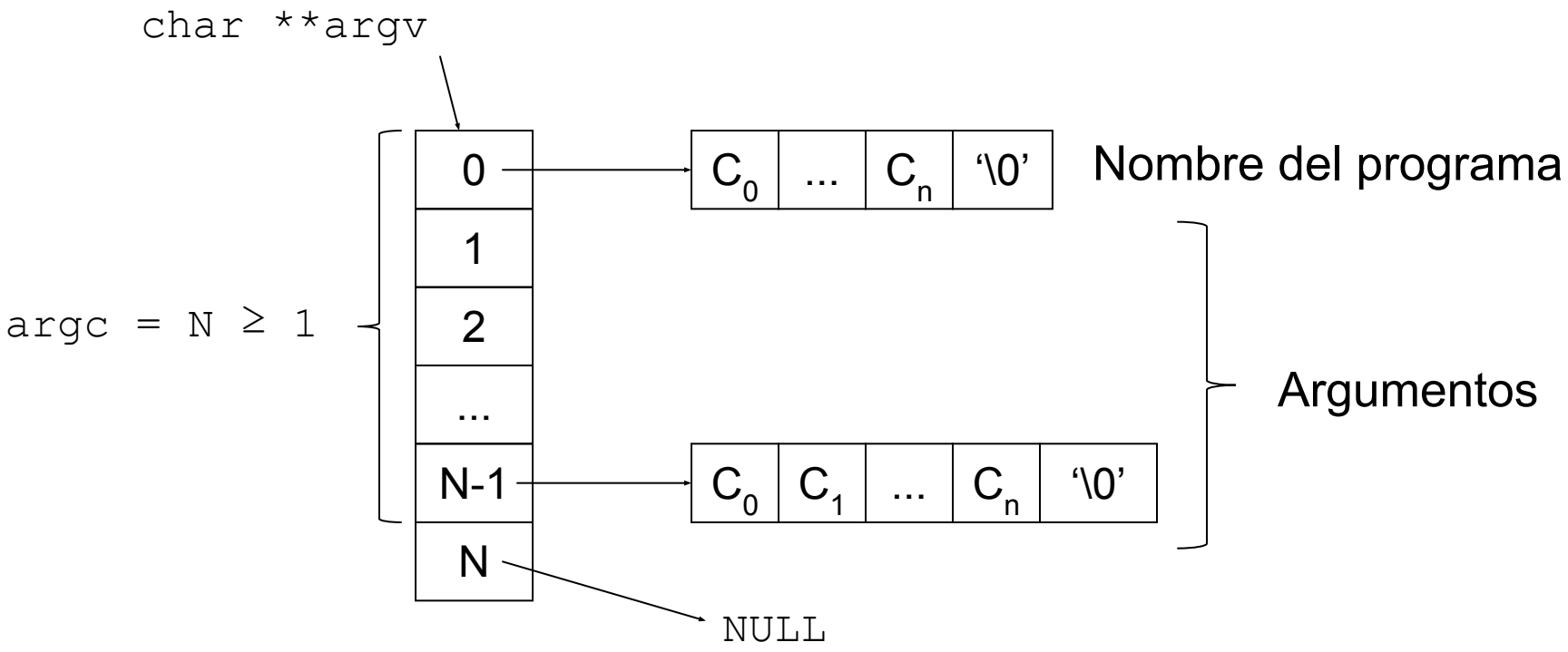
Llamadas al Sistema y Librerías

- Las funciones de sistema y librería están documentadas en las páginas de manual (ver `man man`):
  - Sección 1: Comandos y aplicaciones
  - **Sección 2: Llamadas al sistema**
  - **Sección 3: Funciones y librerías**
  - Sección 4: Dispositivos y ficheros especiales
  - Sección 5: Formatos de ficheros y convenciones
  - Sección 6: Demostraciones y juegos
  - Sección 7: Miscelánea: Descripción de protocolos de red, ASCII, códigos...
  - Sección 8: Comandos de administración (super-usuario)
  - Sección 9: Documentación del kernel o desarrollo de drivers
- El formato general de consulta es: `man [section] comando`
- La sección del manual se especifica seguida del comando, en la forma: `open(2)`
- Es útil usar la opción `-k keyword`
- Puede ser necesario consultar los archivos en `/usr/include/`

Argumentos de un Programa

Formato de cabecera del **programa principal**:

```
int main(int argc, char **argv);
int main(int argc, char *argv[]);
```



Argumentos de un Programa

POSIX recomienda las siguientes convenciones para los argumentos de línea de comandos:

- Los argumentos se consideran opciones si empiezan con un guión (`-`)
- Los nombres de las opciones son un único carácter alfanumérico
- Se pueden indicar varias opciones tras un guión en un solo elemento si las opciones no toman argumentos. Por tanto, `-abc` es equivalente a `-a -b -c`
- Ciertas opciones requieren un argumento, como, `-o name`. El espacio entre la opción y el argumento es opcional. Por tanto, `-o foo` y `-ofoo` son equivalentes
- Normalmente, primero se indican las opciones y después el resto de argumentos
- El argumento `--` termina las opciones. Los argumentos que le siguen se tratan como no opciones, incluso si empiezan por un guión
- Un único guión se interpreta como un argumento ordinario. Por convención, se usa para especificar `stdin` o `stdout`
- Las opciones se pueden proporcionar en cualquier orden o aparecer varias veces. La interpretación se deja al programa

Las opciones largas (extensión de GNU) consisten en dos guiones seguidos de un nombre (que puede abreviarse) compuesto por caracteres alfanuméricos y guiones

- Se puede especificar un argumento para una opción larga con `--name=value`

Argumentos de un Programa

<unistd.h>
POSIX

- Procesar los argumentos de un programa:

```
extern char *optarg;
extern int optind, opterr, optopt;
int getopt(int argc, const char *argv [],
           const char *options);
```

  - `options`: Cadena que contiene las opciones válidas para el programa. Si al carácter le sigue `':'`, indica que esa opción usa un argumento
  - `optind`: Índice que apunta al primer argumento que no es una opción
  - `opterr`: Si el valor de esta variable no es nulo, `getopt()` imprime un mensaje de error cuando encuentre una opción desconocida
  - `optopt`: Cuando se encuentra una opción desconocida o se detecta la falta de un argumento, la opción en cuestión se almacena en esta variable. Útil para mostrar mensajes propios de error
  - `optarg`: Apunta al valor del argumento de la opción

Argumentos de un Programa

Funcionamiento de `getopt(3)`:

- Permuta los contenidos a medida que los trata de forma que los argumentos no-opciones se encuentran al final del array `argv`
- Devuelve el siguiente carácter opción
- Si no hay más devuelve -1. Para comprobar que no existen más argumentos no-opciones comparar `argc` con `optind`
- Cuando la opción tiene un argumento, `getopt()` establece el puntero `optarg` (normalmente no es necesario copiarlo ya que es un puntero a `argv`, que no se modifica)
- Cuando se encuentra una opción no válida o una opción que le falta argumento, devuelve el carácter `'?'` y establece `optopt` a la opción incorrecta
- En caso de error si `opterror` no es cero se muestra un mensaje de error en la salida de error estándar

API del Sistema

- *Application Programming Interface (API):* Conjunto de funciones y rutinas agrupadas con un propósito común
- Consideraciones generales en el uso de un API:
  - ¿Qué fichero de cabecera necesito (`#include`)?
  - ¿Qué tipo de datos devuelve la función?
  - ¿Cuales son los argumentos de la función?
    - Tipos de datos
    - Paso por valor o por referencia (Entrada/Salida)
  - ¿Qué significado tiene el valor de retorno de la función?
  - ¿Qué significado tienen los argumentos de la función?
  - ¿Cómo tengo que gestionar la memoria de las variables?

API del Sistema: Traza

Traza de las llamadas al sistema realizadas por un programa:

```
strace [opciones] comando [argumentos]
```

- Ejecuta el comando hasta que termina, interceptando las llamadas al sistema que realiza y las señales que recibe
- Permite analizar el comportamiento de programas de los que no se dispone el código fuente
- En cada línea se muestra la llamada al sistema realizada, los argumentos de la llamada y el valor de retorno
- Opciones:
  - `-c`: Recopila el tiempo, las llamadas y errores producidos mostrando un resumen
  - `-f`: Traza los procesos hijos a medida que se crean
  - `-T`: Muestra el tiempo de cada llamada
  - `-e trace=`*`call`*: Selección del tipo de llamadas a sistema trazadas (`process`, `network`, `IPC`, `signal` o `file`)
  - `-e write=`*`fd`*: realiza un volcado completo de los datos escritos en el descriptor de ficheros

Gestión de Errores

<stdio.h>
<errno.h>
<string.h>
POSIX+ANSI-C

- Imprime por pantalla un mensaje de error perteneciente a la **última llamada al sistema** realizada:

```
void perror(const char *s);
```

- El formato de salida es:

Cadena `s`	`:`		Mensaje de error	`\n`

- En la cadena debe incluirse el nombre de la función que produjo el error
- El código de error se obtiene de la variable `errno`, que se fija cuando se produce un error (pero no se borra cuando la llamada tiene éxito):

```
int errno;
```

- La siguiente función devuelve una cadena que describe el número de error:

```
char *strerror(int errnum)
```

- Por convenio, las llamadas al sistema devuelven -1 cuando se ha producido un error. Habitualmente algunas llamadas de librería también lo hacen

Información del Sistema

<sys/utsname.h>
SV+POSIX

- Obtención de información sobre el kernel actual:

```
int uname(struct utsname *buffer);
```

- La información se devuelve en la estructura `buffer`, de la forma:

```
struct utsname {
        char sysname[];
        char nodename[];
        char release[];
        char version[];
        char machine[];
}
```

- Código de error: **EFAULT** (`buffer` no es válido)
- Parte de la información también se puede acceder mediante

```
/proc/sys/kernel/{ostype,hostname,osrelease,version,domainname}
```

Información del Sistema

<unistd.h>
POSIX

- Obtención de información sobre el sistema operativo:

```
long sysconf(int name);
```

- El argumento `name` puede ser:
  - `_SC_ARG_MAX`: Longitud máxima de los argumentos de las funciones `exec()`
  - `_SC_CLK_TCK`: Número de ticks de reloj por segundo (Hz)
  - `_SC_OPEN_MAX`: Número máximo de ficheros que puede abrir el proceso
  - `_SC_PAGESIZE`: Tamaño de página en bytes
  - `_SC_CHILD_MAX`: El número máximo de procesos simultáneos por usuario
- La función devuelve el valor del parámetro o -1 en caso de error, en este caso no se instancia la variable `errno`

Información del Sistema

<unistd.h>
POSIX

- Obtención de información sobre el sistema de ficheros:

```
long pathconf(char *path, int name);
long fpathconf(int filedes, int name);
```

- El parámetro `name` puede ser:
  - `_PC_LINK_MAX`: Número máximo de enlaces al archivo/directorio
  - `_PC_NAME_MAX`: Longitud máxima del nombre de archivo en el directorio indicado por `path`
  - `_PC_PATH_MAX`: Longitud máxima del path relativo
  - `_PC_CHOWN_RESTRICTED`: Devuelve un valor no nulo si no puede efectuarse un cambio de permisos sobre el archivo
  - `_PC_PIPE_BUF`: Tamaño de la tubería asociada a `path` o `filedes`
- La función devuelve el límite asociado con el parámetro, -1 en caso de que no exista (no modifica `errno`), en caso de error devuelve -1 e instancia la variable `errno`

Información del Usuario

<unistd.h>
<sys/types.h>

BSD+POSIX

- Los procesos disponen de un identificador de usuario (**UID**) y de grupo (**GID**), que corresponden a los identificadores del usuario que posee el proceso o, en general, a los del proceso que lo creó
  - Estos identificadores se denominan **UID y GID reales**

```
uid_t getuid(void);
gid_t getgid(void);
```

- Además los procesos disponen de un identificador de usuario *efectivo* (**EUID**) y grupo *efectivo* (**EGID**), que son los que se comprueban para conceder permisos
  - Generalmente ambos identificadores (UID y EUID) coinciden
  - Sin embargo, cuando se ejecuta un programa con el bit *setuid* activado, el proceso hereda los privilegios del propietario y grupo del archivo de programa

```
uid_t geteuid(void);
gid_t getegid(void);
```

Información del Usuario

<pwd.h>
<sys/types.h>

SV+POSIX+BSD

- Obtención de la **información de usuario** la base de datos de contraseñas:

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

```
struct passwd {
    char *pw_name;      /* Nombre de usuario */
    char *pw_passwd;    /* Contraseña */
    uid_t pw_uid;       /* Identificador de usuario */
    gid_t pw_gid;       /* Identificador de grupo */
    char *pw_gecos;     /* Nombre real de usuario */
    char *pw_dir;       /* Directorio "home" */
    char *pw_shell;     /* Shell*/
}
```

- La función devuelve NULL, si no encontró al usuario o si se produce algún error (**ENOMEM** si no puede reservar memoria para la estructura)
- Con **shadow passwords** es necesario utilizar las funciones `getspnam`

Información de la Hora del Sistema

- Tiempo en segundos desde el *Epoch*

```
time_t time(time_t *t);
```

<time.h>
SV+BSD+POSIX

- El *Epoch* se refiere a 1970-01-01 00:00:00 +0000, UTC
- Si `t` no es NULL el resultado también se almacena en la variable apuntada por `t`

- Funciones para fijar y obtener la fecha del sistema:

```
int gettimeofday(struct timeval *tv,
        struct timezone *tz);
int settimeofday(const struct timeval *tv,
        const struct timezone *tz);

struct timeval {
    long tv_sec;  /* secs relativos a Epoch */
    long tv_usec; /* µsecs */
}
```

<unistd.h> <sys/time.h>
SV+BSD

- La estructura `timezone` está obsoleta, y `tz` debe ponerse a NULL, de forma que la estructura correspondiente ni se modifica ni se retorna
- Únicamente el super-usuario puede modificar la fecha del sistema

Información de la Hora del Sistema

<time.h>
SV+BSD+POSIX

- Conversión de la información temporal a cadena:

```
char *ctime(const time_t *time);
```

- *Coordinated Universal Time (UTC)*:

```
struct tm *gmtime(const time_t *time);
```

- Tiempo relativo a la zona horaria especificada:

```
struct tm *localtime(const time_t *time);
```

```
struct tm {
    int tm_sec;    /* segundos 0-59 */
    int tm_min;    /* minutos 0-59 */
    int tm_hour;   /* horas 0-23 */
    int tm_mday;   /* día del mes 1-31 */
    int tm_mon;    /* mes 0-11 */
    int tm_year;   /* años desde 1900 */
    int tm_wday;   /* día de la semana (Dom.) 0-6*/
    int tm_yday;   /* día del año (1-1) 0-365*/
    int tm_isdst; /* horario verano/invierno */
};
```

Información de la Hora del Sistema

<time.h>
SV+BSD+POSIX

- Conversión de la información temporal a cadena a medida:

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

- El parámetro `format` es una cadena donde:
  - `%a`: Día de la semana abreviado (idioma sistema)
  - `%A`: Día de la semana completo
  - `%b`: Mes abreviado
  - `%B`: Mes completo
  - `%d`: Día del mes en decimal
  - `%H`: Hora en decimal (24)
  - `%I`: Hora en decimal (12)
  - `%M`: Minutos es decimal
  - `%S`: Segundos en decimal
  - `%n`: Retorno de carro
  - `%p`: PM, AM
  - `%r`: La hora en a.m./p.m. = `'%I:%M:%S %p'`
- La función devuelve el tamaño de la cadena generada, sin incluir el carácter de fin de cadena. Si la cadena no es suficientemente grande (`max`) devuelve 0