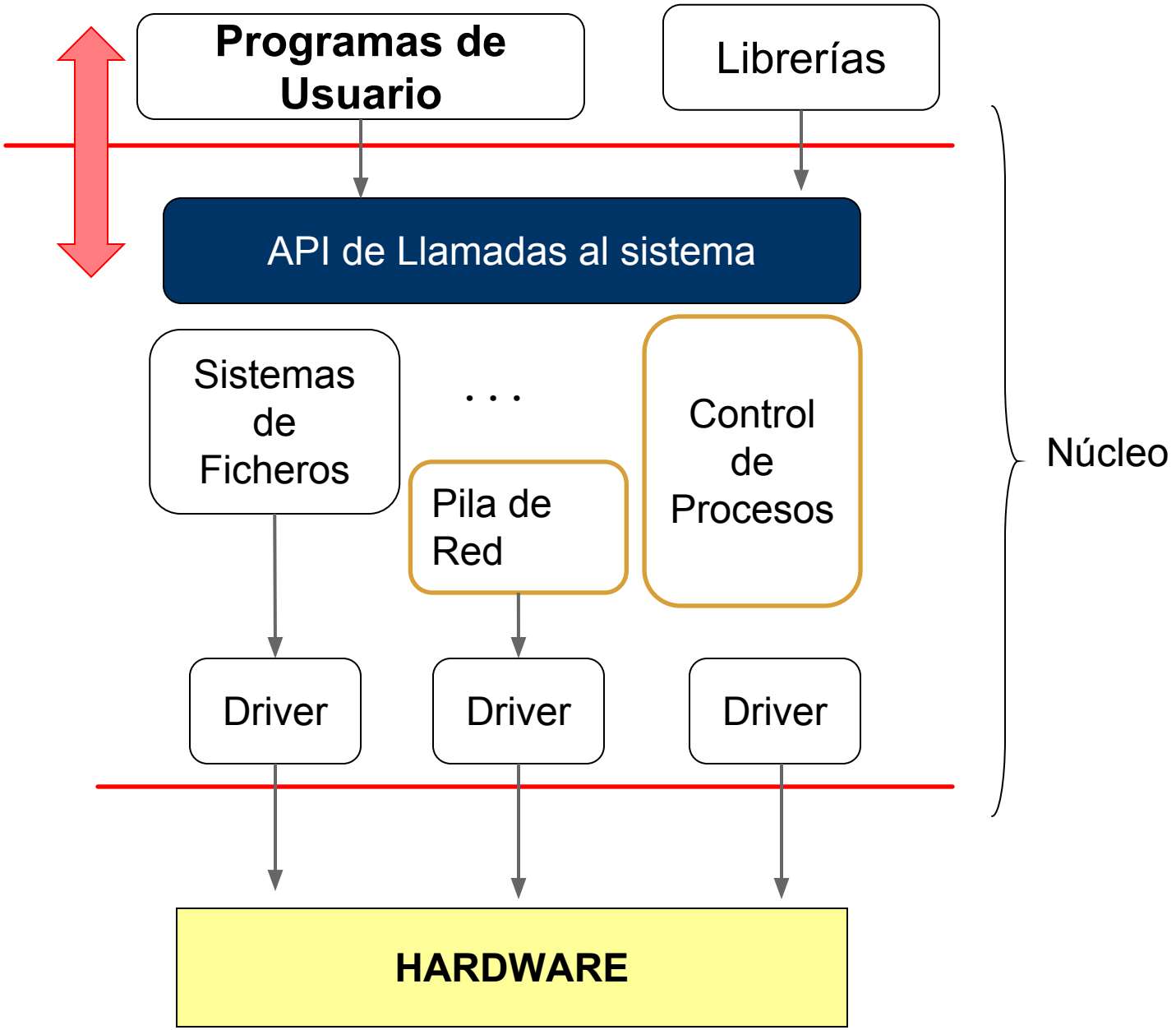
TEMA 5. Introducción a la Programación de Sistemas

PROFESORES:

Rubén Santiago Montero
Ignacio Martín Llorente
Juan Carlos Fabero Jiménez

Introducción: Arquitectura del sistema



Introducción: Estándares de Programación

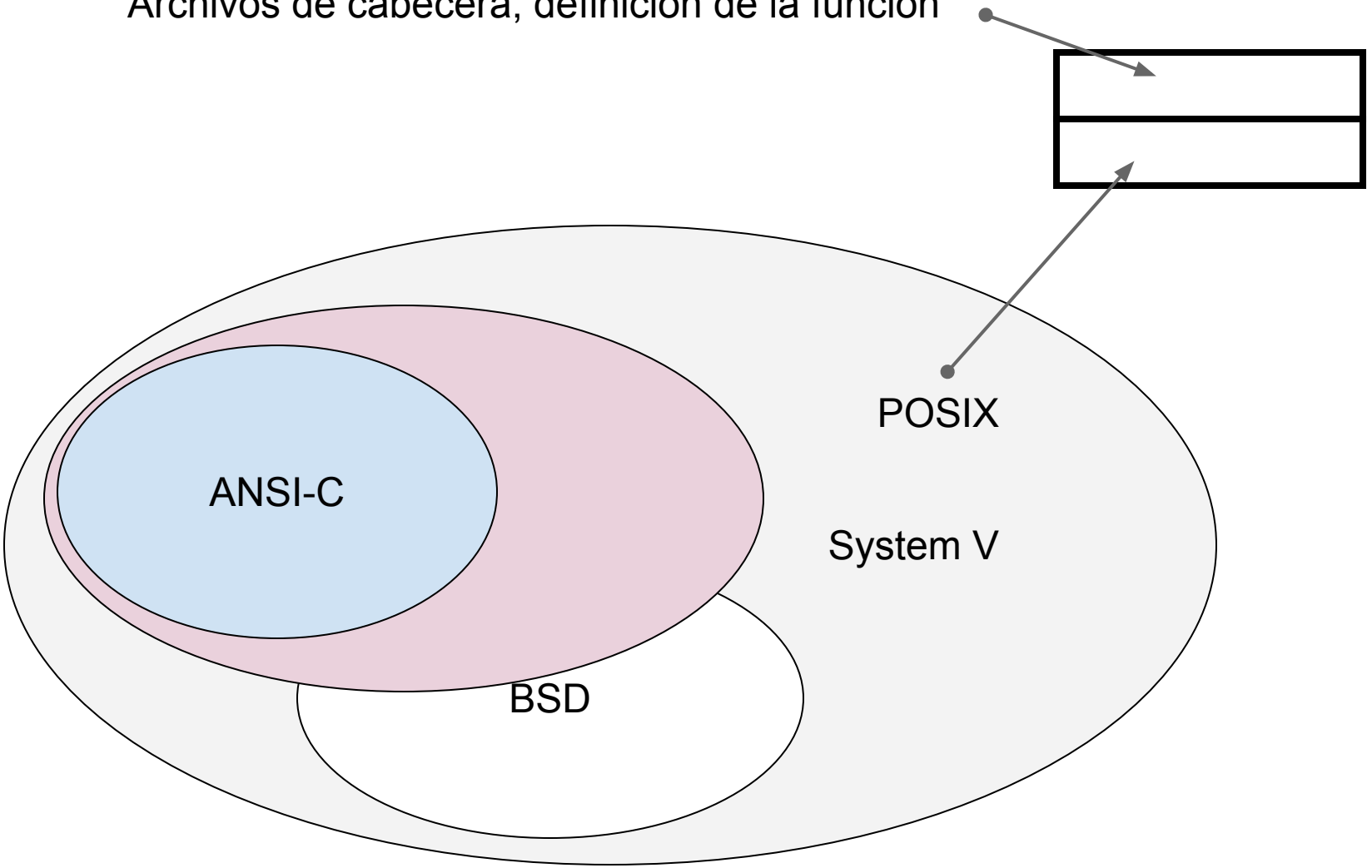
•**ISO-C**: Estándar de programación adoptado por la *American National Standards Institute (**ANSI**)* y posteriormente por la *International Standarization Organization (ISO).* Es el estándar más general. Puede usarse la opción -ansi en el compilador para cumplirlo estrictamente.

•**POSIX** (*Portable Operating System Interface*): Derivado de diversas versiones de Unix. Es un superconjunto de la funcionalidad ofrecida por ANSI-C, extendiéndola. El objetivo es ofrecer soporte de bajo nivel para un tipo de sistemas operativos, y no un lenguaje de programación que pueda ejecutarse en diversos sistemas.

•**Berkely UNIX (BSD)**: La mayoría de los sistemas BSD (ej: SunOS) soportan toda la funcionalidad de los estándar ISO y POSIX. Las contribuciones más importantes de este sistema son los enlaces simbólicos, los sockets, la función select...

•**System V(SV)**: Puede considerarse como un superconjunto del estándar POSIX. La funcionalidad más importante es la comunicación entre procesos.

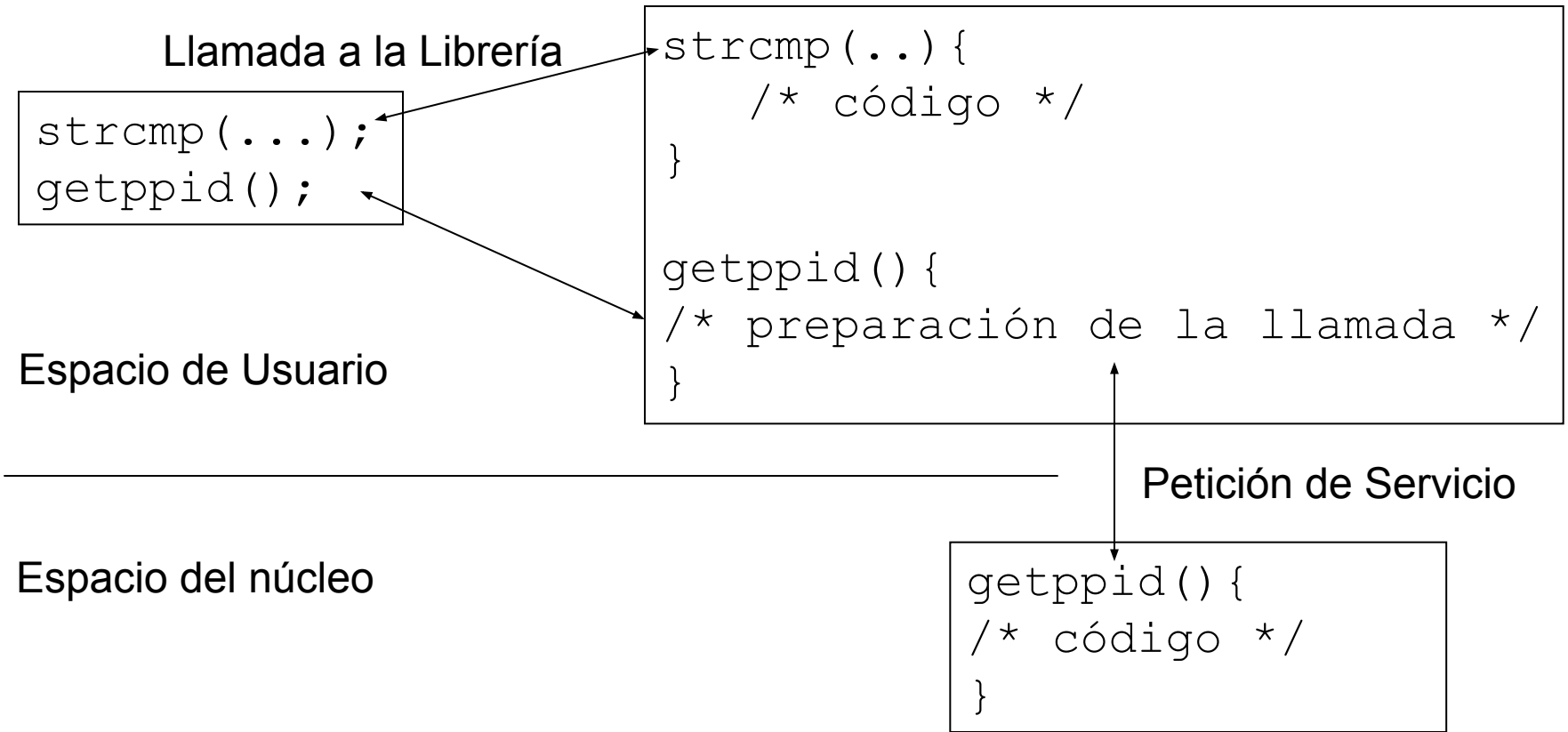
Introducción: Estándares de Programación



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. Esta petición se resuelve en el núcleo del sistema operativo (trap).
- Una llamada a una librería estándar no interacciona de forma directa con el 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

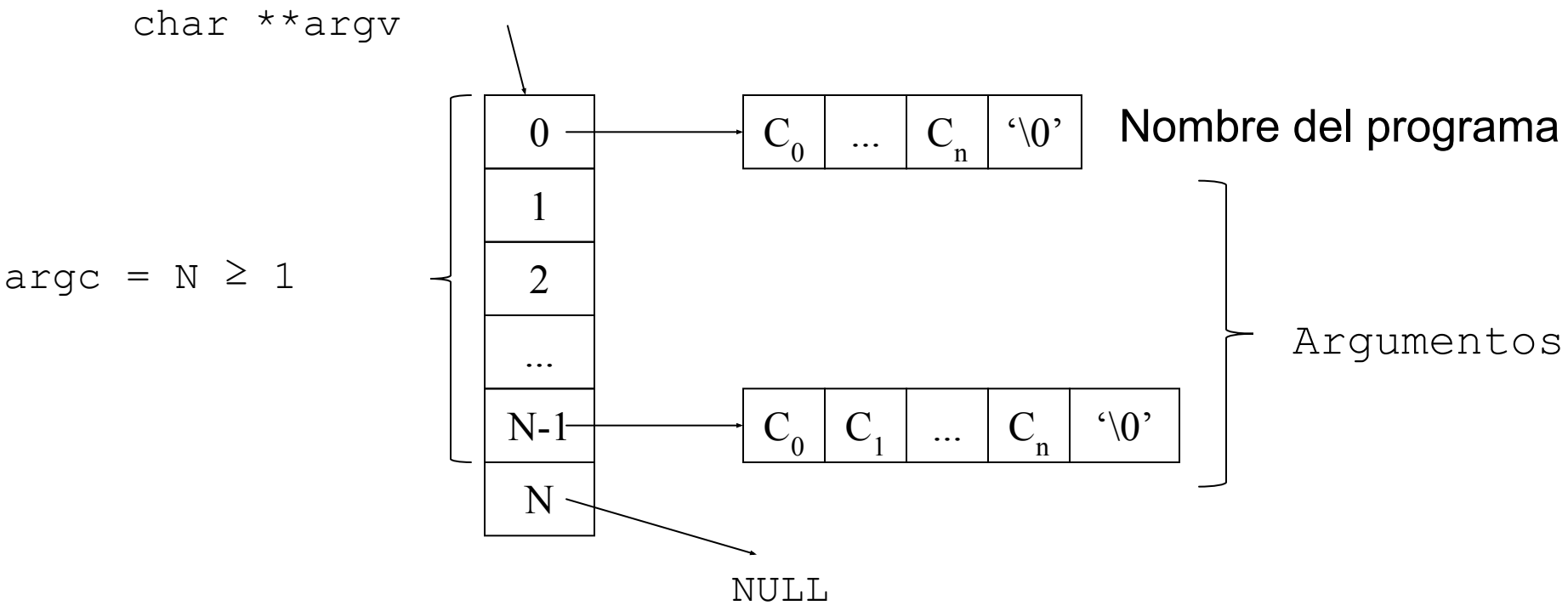
Llamadas al Sistema y Librerías

- Las funciones de sistema y librería están documentadas en las páginas de manual:
  - sección 1: Comandos y Aplicaciones.
  - **sección 2: Llamadas al sistema.**
  - **sección 3: Funciones y Librerías.**
  - sección 4: Dispositivos.
  - sección 5: Formatos de ficheros.
  - 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.
- 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

Comandos en línea de un programa según el estándar **POSIX:**

- Un argumento se considera **opción si comienza por '-'.**
- Si la opción no tiene argumentos **pueden agruparse**. Ej:'-a -b -c' '-abc'.
- Las opciones son **un único carácter** alfanumérico.
- Las **opciones** pueden necesitar **argumento**s. Ej: '-o'. En este caso, las opciones y sus argumentos **pueden o no aparecer separadas** Ej:'-o salida' '-osalida'.
- En principio se procesan primero las opciones con argumentos y después las que usan argumentos, a pesar que el usuario los intercambios.
- **Extensiones GNU**, opciones largas '--' seguidas por una cadena alfanumérica, que puede ser abreviada, siempre que la abreviatura sea única. Ej:'--name=value'.

Argumentos de un Programa

<unistd.h>
POSIX

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

- `opterr:` Si el valor 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. Mensajes propios de error.
- `optind:` Almacena un índice que apunta al último argumento. Determina el comienzo de los argumentos que no son opciones
- `*optarg:` Apunta al valor del argumento de la opción
- `*pc:` Cadena que contiene las opciones válidas para el programa, si al carácter le sigue ':', indica que esa opción usa un argumento

Argumentos de un Programa

Funcionamiento:

•Permuta los contenidos a medida que los trata de forma que las **no-opciones** se encuentran **al final** del array **`argv`**.

•La función **devuelve el siguiente carácter opción**, si no hay más devuelve '-1'. Para comprobar que no existen más opciones **comparar `argc` con `optind`**.

•Cuando la opción tiene **un argumento** `getopt()` **inicializa el puntero** `optarg`, (normalmente no es necesario copiarlo ya que argv no se modifica).

•Cuando se encuantra una **opción no válida** o una opción que le **falta argumento** devuelve el carácter '?' y pone la variable `optopt` a la opción incorrecta. Si la cadena de opciones comienza con el carácter ':', la función `getopt()` retorna el carácter ':' para indicar la falta de argumento.

• En caso de error si `opterror` no es cero se **muestra un mensaje de error** en la salida de error estándar.

Gestión de Errores

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

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

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

- 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 errno, que se fija cuando se produce un error. (No cuando no se produce).
- Variables:

```
const char *sys_errlist[ ];
int sys_nerr;
int errno;
```

- Por convenio, las llamadas al sistema devuelven -1 cuando se ha producido un error. Habitualmente algunas llamadas de librería también lo hacen.
- El código de error de un programa es accesible mediante la variable de entorno ?.

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 include necesito?
  - ¿Qué tipo de datos devuelve la función?
  - ¿Cuales son los argumentos de la función?
    - Tipos de datos
    - Paso por valor o referencia (Entrada/Salida)
  - ¿Qué significado tiene el valor de retorno de la función?
  - ¿Qué significado tienen los argumentos de la función?
  - ¿Como 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.
- **Utilidad:** 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=process/network/IPC/signal/file`: Selección del tipo de traza
  - -e write=3: traza las llamadas a write sobre el descriptor de ficheros 3

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[SYS_NMLN];
        char nodename[SYS_NMLN];
        char release[SYS_NMLN];
        char version[SYS_NMLN];
        char machine[SYS_NMLN];
}
```

- Código de Error: **EFAULT** (buffer no 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 sobre la información del sistema operativo:

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

- donde el argumento `name` :
  - `_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 sobre la información del sistema de ficheros:

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

long fpathconf(int filedes, int name);
```

- El parámetro `name` :
  - `_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 del pipe asociado 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

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

<unistd.h> <sys/types.h>
BSD+POSIX

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

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

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

Información del Usuario

- Obtención de la **información de usuario** accediendo a la base de datos de contraseñas.

<pwd.h> <sys/types.h>
SV+POSIX+BSD

```
strcut 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).
- **Shadow passwords** es necesario utilizar las funciones getspnam

Información de la hora del sistema

<time.h>
SV+BSD+POSIX

- Tiempo en segundos desde 1 de Enero de 1970

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

- Si el puntero argumento de la función no es NULL el resultado se almacena también en 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*/

long tv_usec;/*µsecs*/

}
```

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

- **struct** timezone: Campo obsoleto que nunca ha de ser utilizado. Cuando algún argumento es NULL, no se modifica ni se devuelve ningún valor.
- 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);
```

- *Coordinate 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;   /* Ahorro de energía */
};
```

Información de la hora del sistema

- Conversión de la información temporal a cadena a medida:

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

<time.h>
SV+BSD+POSIX

- 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.