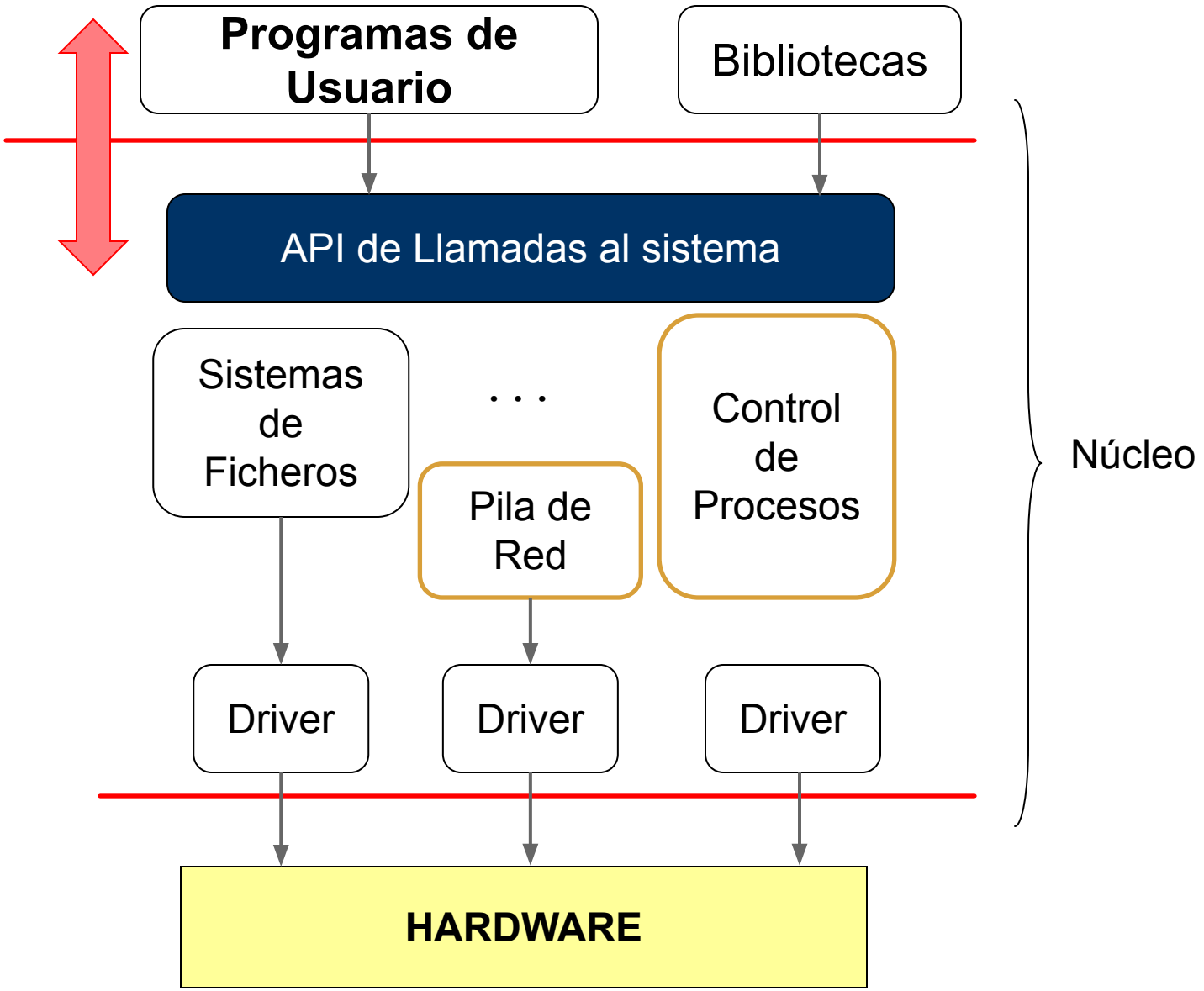
TEMA 2.1. Introducción a la Programación de Sistemas

PROFESORES:

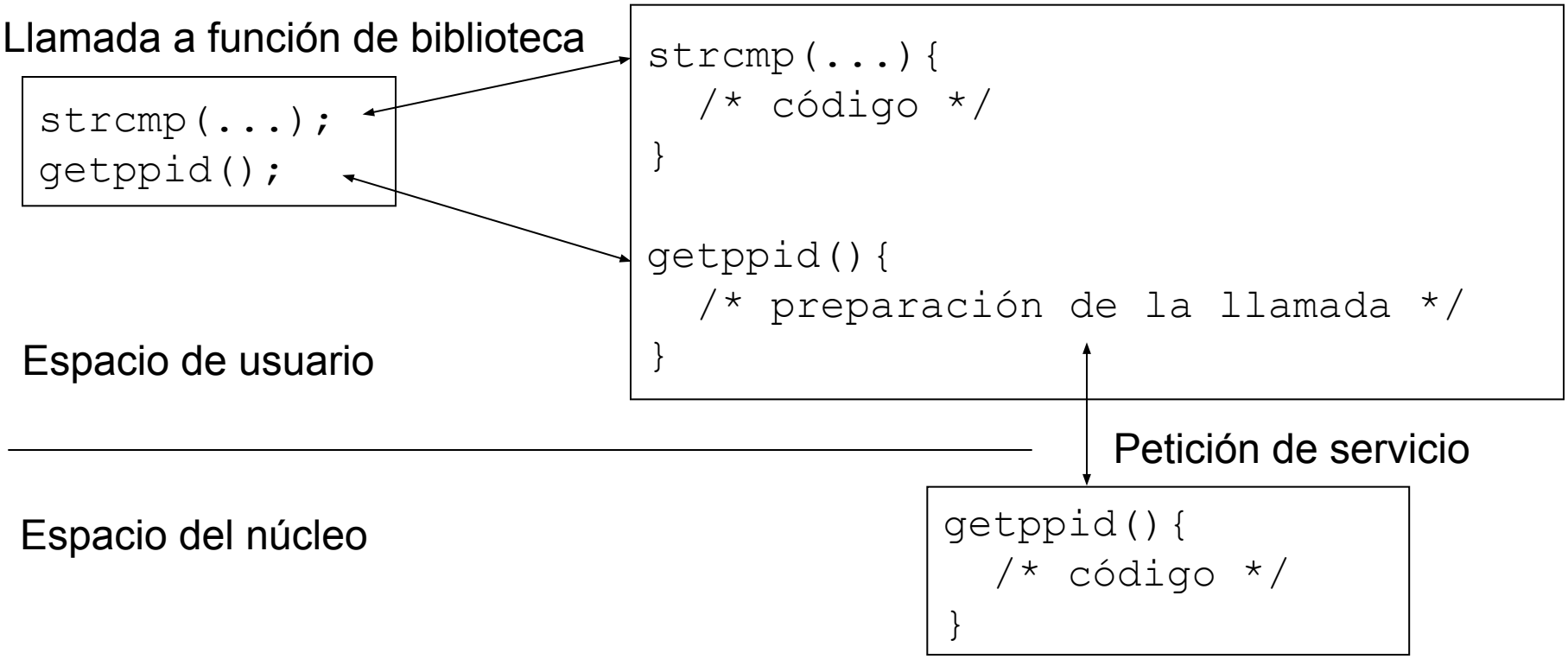
Rubén Santiago Montero
Eduardo Huedo Cuesta
Rafael Rodríguez Sánchez

Introducción: Arquitectura del sistema



Llamadas al Sistema y Funciones de Biblioteca

- Desde el punto de vista del programador no hay diferencia. Sin embargo:
  - Una llamada al sistema es un función de la biblioteca estándar de C que solicita un servicio del sistema (*trap*), que se resuelve en el núcleo del SO
  - Una función de biblioteca no interacciona de forma directa con el sistema (debe usar llamadas al sistema)



Llamadas al Sistema y de Biblioteca

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

Llamadas al Sistema y Funciones de Biblioteca

- Las llamadas al sistema y las funciones de biblioteca están documentadas en las páginas de manual (ver `man man`):
  - Sección 1: Comandos y aplicaciones
  - **Sección 2: Llamadas al sistema**
  - **Sección 3: Funciones de biblioteca**
  - Sección 4: Dispositivos y ficheros especiales
  - Sección 5: Formatos de ficheros y convenciones
  - Sección 6: Demostraciones y juegos
  - Sección 7: Miscelánea (convenciones y protocolos, juegos de caracteres, jerarquía del sistema de ficheros...)
  - Sección 8: Comandos de administración del sistema (superusuario)
  - Sección 9: Documentación del núcleo y 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)`
- El uso de `-k keyword` es útil para buscar páginas específicas
- Puede ser necesario consultar los ficheros en `/usr/include/`

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 necesita (`#include`)?
  - ¿Qué tipo de datos devuelve la función?
  - ¿Cuales son los argumentos de la función?
    - Tipos de datos
    - Paso por valor o por referencia
  - ¿Qué significado tiene el valor de retorno de la función?
  - ¿Qué significado tienen los argumentos de la función?
  - ¿Cómo se tiene que gestionar la memoria de las variables?
  - ¿Cómo de portable es la función?
  - ¿Cómo puede fallar la función?

API del Sistema: Traza de Llamadas

- Trazar 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 al 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

- Imprimir un mensaje de error:

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

  - Imprime por la salida de error (`stderr`) un mensaje que describe el último error encontrado en una llamada al sistema o función de biblioteca
  - El formato de salida es:

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;
```

- Por convenio, las llamadas al sistema devuelven -1 cuando se produce un error
  - Muchas funciones de biblioteca también lo hacen
- Devolver una cadena que describe el número de error:

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

Información del Sistema

<sys/utsname.h>
SV+POSIX

- Obtener información sobre el sistema operativo:

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

struct utsname {
    char sysname[];     /* Nombre del SO (ej. "Linux") */
    char nodename[];    /* Nombre del host */
    char release[];     /* Release del SO (ej. "3.10.0") */
    char version[];     /* Versión del SO (fecha) */
    char machine[];     /* Hardware (ej. "x86_64") */
    char domainname[]; /* Nombre de dominio */
}
```

  - Devuelve la información en la estructura apuntada por `buffer` (paso por referencia)
  - Código de error: **EFAULT** (`buffer` no es válido)
- El comando `uname` proporciona acceso a esta funcionalidad
- Parte de la información está accesible vía `sysctl` y vía `/proc/sys/kernel/{ostype,hostname,osrelease,version,domainname}`

Información del Sistema

<unistd.h>
POSIX

- Obtener información de configuración del sistema:

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

  - El argumento `name` puede ser:
    - `_SC_ARG_MAX`: Longitud máxima de argumentos en funciones `exec()`
    - `_SC_CLK_TCK`: Número de ticks de reloj por segundo (Hz)
    - `_SC_OPEN_MAX`: Número máximo de ficheros abiertos por proceso
    - `_SC_PAGESIZE`: Tamaño de página en bytes
    - `_SC_CHILD_MAX`: Número máximo de procesos simultáneos por usuario
  - Devuelve el valor del parámetro o -1 en caso de error, pero no establece el valor de `errno`
- El comando `getconf` proporciona acceso a esta funcionalidad

Información del Sistema

<unistd.h>
POSIX

- Obtener información de configuración del sistema de ficheros:

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

  - El parámetro `name` puede ser:
    - `_PC_LINK_MAX`: Número máximo de enlaces
    - `_PC_NAME_MAX`: Longitud máxima del nombre de fichero
    - `_PC_PATH_MAX`: Longitud máxima de la ruta relativa
    - `_PC_CHOWN_RESTRICTED`: Devuelve un valor no nulo si el cambio del propietario del fichero está restringido
    - `_PC_PIPE_BUF`: Número máximo de bytes que pueden escribirse atómicamente en la tubería
  - La función devuelve el límite asociado con el parámetro o -1 en caso de que no exista (no modifica `errno`) o en caso de error (sí establece `errno`)
- El comando `getconf` proporciona acceso a esta funcionalidad

Información del Usuario

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

BSD+POSIX

- Obtener los identificadores de un proceso:

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

- Los procesos disponen de un identificador de usuario (**UID**) y de grupo (**GID**), que corresponden a los identificadores del propietario del proceso que, en general, se heredan del proceso que lo creó
  - Estos identificadores se denominan **UID y GID reales**
- Además, los procesos disponen de un identificador de usuario **efectivo** (**EUID**) y de grupo **efectivo** (**EGID**), que son los que se comprueban para conceder permisos
  - Generalmente, los identificadores reales y efectivos coinciden
  - Sin embargo, si el fichero de programa tiene los bits *setuid* o *setgid* activos, el EUID o el EGID del proceso creado se cambian al usuario o grupo del fichero

Información del Usuario

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

SV+POSIX+BSD

- Obtener **información de un usuario** de 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;   /* Descripción del usuario */
    char *pw_dir;     /* Directorio "home" */
    char *pw_shell;   /* Shell */
}
```

  - Devuelven un puntero a una estructura asignada estáticamente que puede sobreescribirse (hay versiones **reentrantes**)
  - Devuelven `NULL`, si no se encuentra el 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

- Obtener el tiempo en segundos desde el *Epoch:*

<time.h>
SV+BSD+POSIX

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

  - 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

- Obtener y establecer la fecha del sistema:

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

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

struct timeval {
    long tv_sec;  /* segundos relativos a Epoch */
    long tv_usec; /* microsegundos */
}
```

  - `gettimeofday` devuelve la fecha en la estructura apuntada por `tv` (paso por referencia)
  - 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 superusuario puede modificar la fecha del sistema

Información de la Hora del Sistema

<time.h>
SV+BSD+POSIX

- Obtener el tiempo desglosado UTC (*Coordinated Universal Time*) o en la zona horaria local:

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

  - Devuelven un puntero a una estructura asignada estáticamente que podría sobreescribirse (hay versiones reentrantes)

Información de la Hora del Sistema

<time.h>
SV+BSD+POSIX

- Formatear fecha y hora:

```
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
    - `%j`: Día del año 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`: Hora en a.m./p.m. (equivalente a "`%I:%M:%S %p`")
  - Devuelve la longitud de la cadena generada o 0 si supera los `max` bytes

Ejercicios: Preguntas Teóricas

¿Qué efecto tiene que un ejecutable tenga activado el permiso SETUID?

- ☐ El proceso se creará con permisos de superusuario.
- ☐ El proceso se creará con usuario (UID) igual al propietario del fichero.
- ☐ El proceso se creará con usuario efectivo (EUID) igual al propietario del fichero.

Una función de biblioteca...

- ☐ no puede usar internamente llamadas al sistema.
- ☐ es una función de la biblioteca estándar de C que solicita un servicio del sistema.
- ☐ no interacciona de forma directa con el sistema.

Material adicional

Estándares de Programación

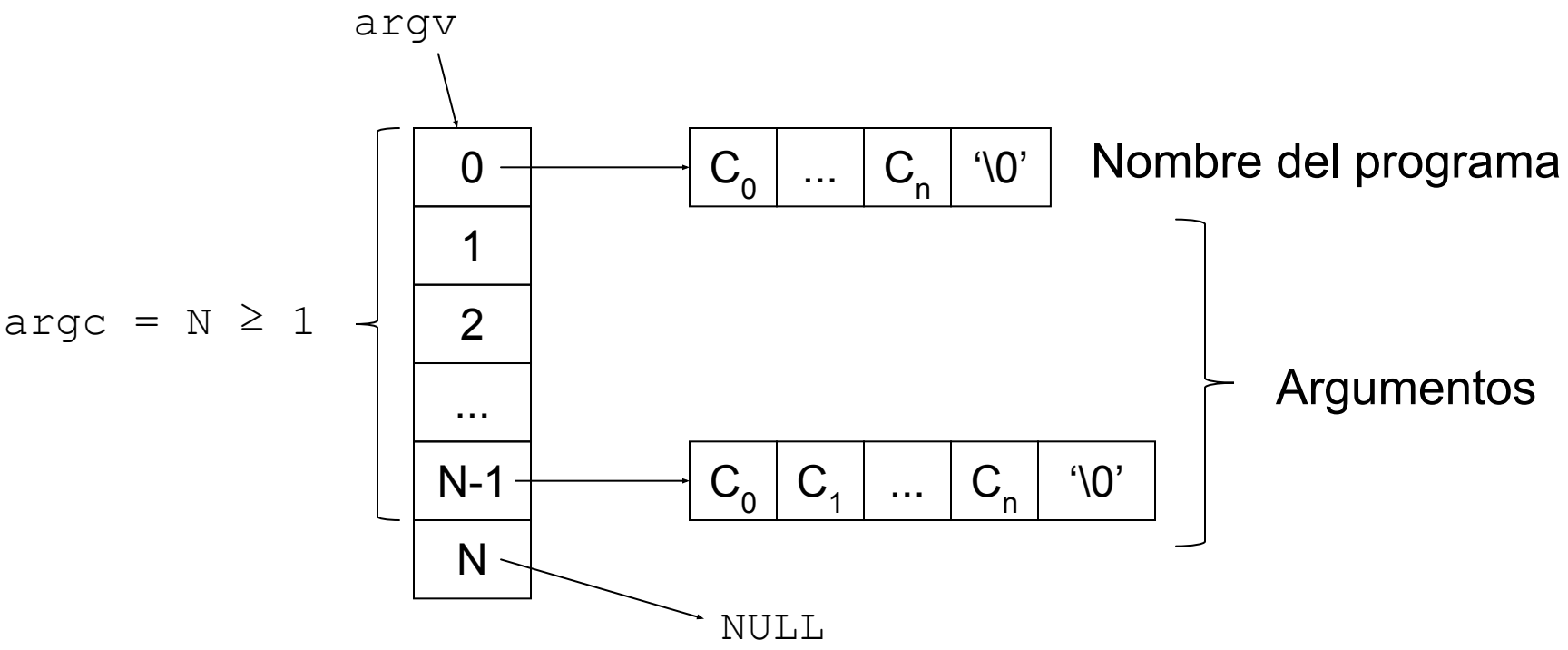
- **ANSI-C o ISO-C**: Estándar de programación adoptado por ANSI (*American National Standards Institute*) y posteriormente por ISO (*International Standardization Organization*). Es el estándar más general. La opción `-ansi` hace que el compilador lo cumpla de forma estricta.
- **BSD** (*Berkeley Software Distribution*): Desarrollado durante los 80 en la Universidad de California Berkeley. Sus contribuciones más importantes son los enlaces simbólicos, los sockets, la llamada `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 de biblioteca, 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 núcleo de Linux es GNU/Linux.

`standards(7)`

Argumentos del Programa

- Definición del **programa principal**:

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



Argumentos del 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` es equivalente a `-ofoo`
  - 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 con `--name=value`

Argumentos del 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 del 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 `opterr` no es cero se muestra un mensaje de error en la salida de error estándar