Programación en C moderno

Álvaro Neira Ayuso <alvaro@soleta.eu>

f) Ejemplo 5: listas.

- Estructura list_head.
- Añadir elementos a la lista con list_add.
- Eliminar elementos de la lista con list_del.
- Referencias a elementos de una lista (&).

Estructura list_head

- Como todo lenguaje de programación C, también necesita listas de elementos.
- Existen varias implementaciones para dichas listas de elementos.
- Entre las cuales tenemos las listas list_head.
- Para utilizarlas hay que incluir la biblioteca list.h.

```
#include <linux/list.h>
```

Estructura list_head

```
struct list_head {
    struct list_head *next, *prev;
};
```

- Las listas contienen el elemento anterior y posterior a dicho elemento.

Declarar list_head

- Para utilizar las listas debemos actualizar nuestras estructuras.
- Debemos añadir en nuestras estructuras el elemento list_head para direccionarlos a él

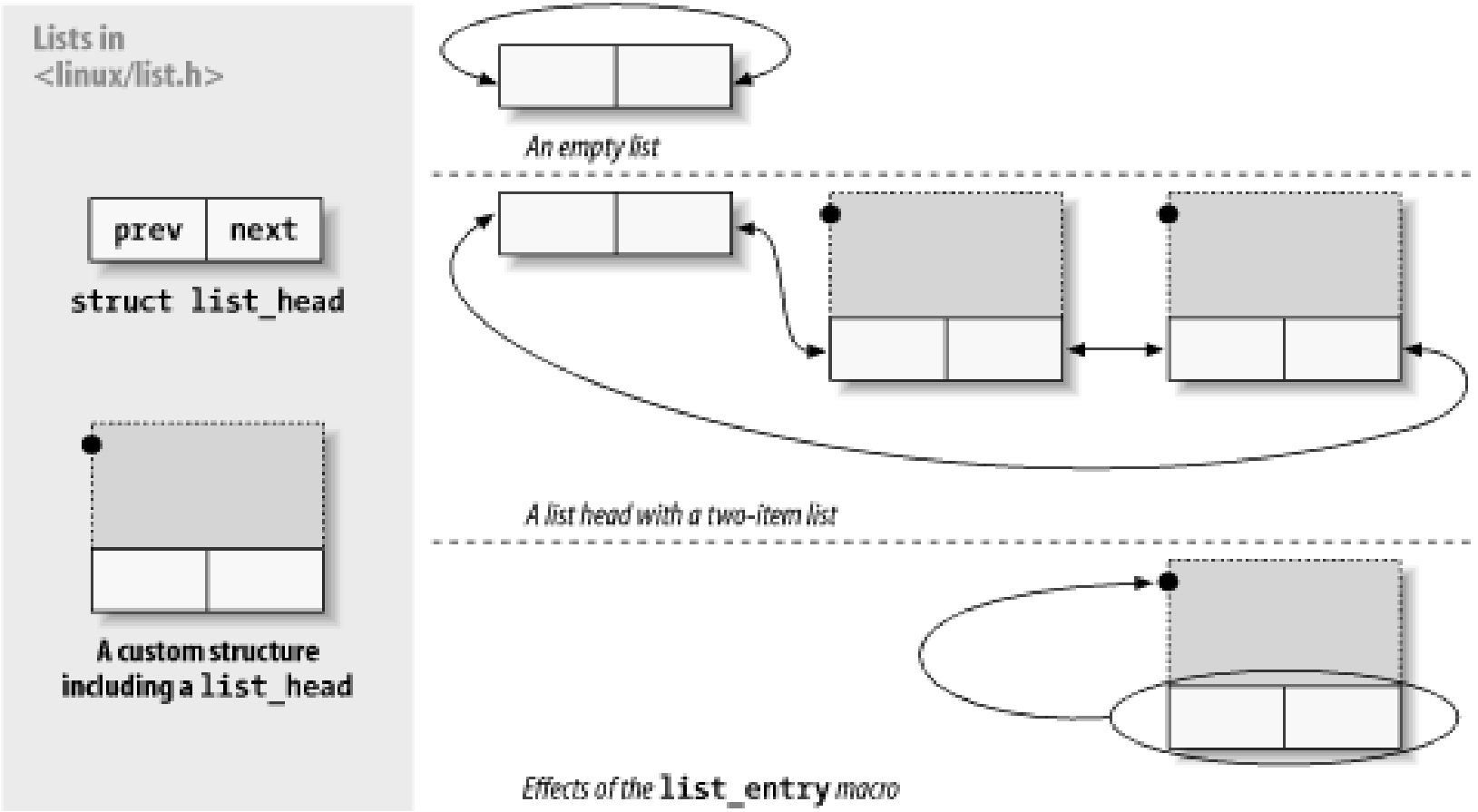
Declarar list_head

```
struct coche {
    struct list_head   head;

    uint32_t     id;
    const char      *matricula;
    const char      *marca;

    uint32_t     flags;
};
```

Declarar list_head



Inicializar list_head

- Para iniciar nuestras listas tenemos una macro que nos proporciona la biblioteca

```
struct list_head    lista;

INIT_LIST_HEAD(&lista);
```

- Es obligatorio la inicialización de las listas.

Añadir elementos a la lista:

- La biblioteca list nos proporciona dos formas diferentes para añadir elementos en las listas:
- Añadir usando list_head es equivalente a enlazar nuestra lista con las lista comprendida en nuestra estructura

Añadir elementos a la lista con list_add

list_add(struct list_head *new, struct list_head *head);

- Donde el parámetro new es el nuevo elemento que queremos añadir a nuestra lista y el parámetro head es la lista donde queremos añadirlo.
- Se añade el elemento al final de lista.

Añadir elementos a la lista list_add_tail

- Donde el parámetro new es el nuevo elemento que queremos añadir a nuestra lista y el parámetro head es la lista donde queremos añadirlo.
- Se añade el elemento al principio de la lista.

Borrar elementos de la lista

```
list_del(struct list_head *entry);
```

- Donde el parámetro entry, es la lista que esta asociada a la estructura.

Mover elementos en la lista

```
list_move(struct list_head *entry, struct list_head *head);
```

- Donde el parámetro entry es el elemento contenido en la lista que queremos mover y el parámetro head es la lista la cuál queremos moverlo.
- Esta función borraría el elemento de la lista que lo contiene y lo movería al principio de la lista.

Mover elementos en la lista

list_move_tail(struct list_head *entry, struct list_head *head);

- Donde el parámetro entry es el elemento contenido en la lista que queremos mover y el parámetro head es la lista la cuál queremos moverlo.
- Esta función borraría el elemento de la lista que lo contiene y lo movería al final de la lista.

Como recorrer una lista

- En todo programa es necesario recoger los elementos de una lista para realizar algún tipo de tratamiento en ellos.
- Para realizar esa acción tenemos una serie de funciones que nos proporciona la biblioteca.

Como recorrer una lista

list_for_each_entry(type *cursor, struct list_head *list, member)

- Donde el parámetro cursor es el objeto que contiene nuestra lista, el parámetro list es la lista de objetos y el parámetro member es el nombre de la list_head dentro de la estructura.

Como recorrer una lista

list_for_each_entry_safe(type *cursor, type *next, struct list_head *list, member)

- Donde el parámetro cursor es el objeto que contiene nuestra lista, el parámetro next es el siguiente elemento de nuestra lista, el parámetro list es la lista de objetos y el parámetro member es el nombre de la list_head dentro de la estructura.

Ejemplo

Ejercicios

g) Ejemplo 6: paso de argumentos al programa y tratamiento.

- Los parámetros argc y argv y la función getopt_long

Pasar de argumentos

- C nos proporciona una serie de vías para acceder a argumentos que pasamos por línea de comando al programa
- En este apartado, vamos a estudiar las vías para poder realizar esto

Pasar argumentos de programa con argc y argv

- Declaramos que main tiene dos parámetros; uno un entero y otro un array de punteros a carácter:

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

- Al ejecutar el programa, argc tendrá el numero de argumentos y los argv[i] son los argumentos de línea de comando

Parar argumentos de programa usando argc y argv

- argv[0] es el nombre del programa que se ejecuta por lo que si argc es 1, no se le han pasado argumentos

- Al ejecutar el programa, argc será igual al número de argumentos y los argv[i] (hasta argc-1) son los argumentos pasados por línea de comandos

Ejemplo

Tratamiento de argumentos no abreviados

- Existen programas los cuales necesitamos pasar argumentos al programa de no abreviados
- Podríamos comprobar a partir de una función que nos proporciona la biblioteca getopt

Tratamiento de argumentos no abreviados

```
int getopt_long(int argc, char * const argv[], const char *optstring,
const struct option *longopts, int *longindex);
```

- El cuál el parámetro argc y argv son los argumentos que recibimos por main.
- El parámetro optstring son las opciones posibles que vamos a recibir y longopts es la estructura que definiremos las opciones largas que queremos traducir a cortas. El parámetro longindex devuelve el índice asociado a nuestra opción larga.

Ejemplo

Ejercicio

h) Ejemplo 7: E/S por ficheros.

- Abrir y cerrar ficheros.
- Lectura y escritura de caracteres en fichero.
- Lectura y escritura de cadenas en un fichero.
- Las funciones fprintf(), fwrite() y fread().

* E/S por ficheros

- Entrada y salida Se refiere a las operaciones que se producen a través de alguna vía de entrada como teclado o ficheros. Mostrando los resultado a través de la pantalla o otros ficheros.
- Existen varias bibliotecas que nos proporcionan dichas herramientas como stdio.h

Abrir y cerrar ficheros

FILE *fopen(const char *nombre, const char *modo);

- Abre un fichero cuyo nombre es la cadena apuntada por nombre, y adjudica un stream a ello. El argumento modo configura que acciones vamos a permitir realizar en dicho fichero

Abrir y cerrar ficheros

- r: Abre un fichero de texto para lectura
- w: Inicializa a longitud cero o crea un fichero de texto para escribir
- a: Añade; abre o crea un fichero de texto para escribir al final del fichero (EOF)
- r+: Abre un fichero de texto para actualización (lectura y escritura)
- w+: Inicializa a longitud cero o crea un fichero de texto para actualización
- a+: Añade; abre o crea un fichero de texto para actualización, escribiendo al final del fichero (EOF)

** EOF = End Of File

Abrir y cerrar ficheros

```
int fclose(FILE *stream);
```

- El stream apuntado por stream será despejado y el fichero asociado, cerrado.
- La función fclose retorna cero si el stream fue cerrado con éxito. Si se detectaron errores, entonces retorna EOF.

Ejemplo

Lectura de caracteres en fichero.

int fgetc(FILE *stream);

- Esta función obtiene el carácter siguiente (si está presente) como un unsigned char convertido a int.
- Si surge algún error dicha función devuelve EOF.

Escritura de caracteres en fichero.

```
int fputc(int c, FILE *stream);
```

- Esta función escribe el carácter indicado por c al stream de salida en la posición indicada por el indicador del steam, y avanza el indicador apropiadamente.

Ejemplo

Lectura de cadenas en fichero

```
char *fgets(char *cadena, int n, FILE *stream);
```

- Esta función lee como máximo uno menos que el número de caracteres indicado por n desde el stream apuntado por stream al array apuntado por cadena.
- Devuelve NULL cuando encuentra EOF o surge algún tipo de error.

Escritura de cadenas en ficheros.

int fputs(const char *cadena, FILE *stream);

- Esta función escribe la cadena apuntada por cadena al stream apuntado por stream.
- Retorna EOF si ocurre un error de escritura,

Ejemplo

Leer ficheros usando fread()

size_t fread(void *puntero, size_t tamanyo, size_t nmemb, FILE *stream);

- ptr: Puntero donde queremos guardar los datos leídos
- size: El tamaño de los datos que vamos a leer
- nmemb: El número de elementos que vamos a leer
- stream: El puntero al fichero que queremos leer

Leer ficheros usando fread()

- La función fread recibe, en el array apuntado por puntero, hasta nmemb de elementos cuyo tamaño es especificado por tamanyo, desde el stream apuntado por stream.
- Devuelve el número de bytes que hemos leído, en caso de error devuelve EOF.

Escribir en ficheros usando fwrite()

size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)

- ptr: Puntero el cuál va a ser escrita la información que contiene al fichero
- size: Tamaño de los elementos que va a ser escrito
- nmemb: El número de elementos que queremos escribir en nuestro fichero
- stream: El puntero al fichero en donde queremos escribir

Escribir en ficheros usando fwrite()

- La función fwrite envía, desde el array apuntado por puntero, hasta nmemb de elementos cuyo tamaño es especificado por tamaño, al stream apuntado por stream.
- La función fwrite retorna el número de caracteres escritos correctamente.

Ejemplo

Otra vía de escribir en ficheros

int fprintf(FILE *stream, const char *format, ...)

- stream -- Puntero del fichero que queremos añadir la cadena
- format -- El formato que definimos para escribir las cadenas en nuestro fichero.

Otra vía de escribir en ficheros

- Esta función envía datos al stream apuntado por stream, bajo el control de la cadena apuntada por formato que especifica cómo los argumentos posteriores son convertidos para la salida
- Retorna el número de caracteres transmitidos, o un valor negativo si un error.

Otra forma de escribir en ficheros

- %c Carácter
- %d or %i Decimal con signo
- %s Cadena de caracteres
- %u Decimal sin signo
- %p Dirección de un puntero

Ejemplo

Ejercicio