

Desarrollo de aplicaciones en el sistema operativo OSO

Departamento de Ingeniería y Tecnología de Computadores
Universidad de Murcia

19 de mayo de 2005

Índice

1. Introducción	1
2. Construcción de bibliotecas con TLIB	2
3. La biblioteca del sistema OSO.LIB	2
4. La biblioteca del lenguaje UTIL.LIB	4
5. Aplicaciones de ejemplo	13
5.1. El programa ascii.c	14
5.2. El programa shell.c	14

1. Introducción

Cuando desarrollamos un programa para un sistema operativo, una opción es hacer todo el desarrollo en lenguaje ensamblador, sin utilizar código externo y haciendo uso única y exclusivamente de los servicios proporcionados por las llamadas al sistema de dicho sistema operativo. Esta opción, sin embargo, no es muy adecuada, ya que lo ideal es poder utilizar un lenguaje de más alto nivel que permita además (y tal vez esto sea lo más importante) utilizar y construir fácilmente bibliotecas que proporcionen tanto los servicios de las llamadas al sistema como otros servicios auxiliares (manejo de cadenas, funciones matemáticas, etc.).

Ya que las bibliotecas son importantes, en primer lugar vamos a ver cómo se construyen en un lenguaje de alto nivel, en nuestro caso, C. Distinguiremos entre dos tipos de bibliotecas: la *biblioteca del sistema*, que proporciona funciones para un uso directo de las llamadas al sistema, y la *biblioteca del lenguaje*, que proporciona funciones para el manejo de cadenas, salida por pantalla, entrada por teclado, etc. (es decir, funciones que no usan llamadas al sistema o funciones que, antes o después de hacer una llamada al sistema, hacen un tratamiento más o menos complejo de la información).

Una vez hayamos construido la biblioteca del sistema y la biblioteca del lenguaje, pasaremos a describir su uso para la implementación de programas en la sección 5.

2. Construcción de bibliotecas con TLIB

Una biblioteca está formada por la reunión de varios módulos objeto (.OBJ en nuestro caso) en un único fichero (.LIB en nuestro caso). El nombre de este nuevo fichero aparecerá en la línea de órdenes del montador junto con el nombre de los módulos objeto a montar. El montador extraerá de la biblioteca solamente los módulos objeto cuyas etiquetas hayan sido referenciadas en los módulos objetos que aparecen de forma explícita en la línea de órdenes del montador. Así, el programador se olvida de qué etiquetas (referencias de código o de datos) pertenecen a un módulo u otro y se asegura de que el montador sólo añade los módulos necesarios. Cuando se manejan varios módulos relacionados entre sí (como es nuestro caso) el uso de bibliotecas facilita el desarrollo de las aplicaciones.

El gestor de bibliotecas asociado al entorno TCC/TASM/TLINK de Borland es TLIB. Su uso es el siguiente:

```
TLIB biblioteca órdenes, fichero_listado
```

Si no se indican órdenes se obtiene simplemente información del contenido de la biblioteca en el fichero de listado (que puede ser CON para un listado por pantalla). Las órdenes son de la forma <operación>nombre_de_módulo, donde la operación puede ser una de las siguientes:

- «+»: añade el módulo objeto indicado a la biblioteca.
- «-»: borra el módulo indicado de la biblioteca.
- «*»: extrae el módulo (el fichero .OBJ) de la biblioteca sin borrarlo.
- «-+»: alternativamente «+-», reemplaza el módulo existente en la biblioteca.
- «-*»: alternativamente «*-», extrae el módulo y lo borra de la biblioteca.

Por ejemplo, para añadir el módulo CREAPROC.OBJ, borrar el READ.OBJ y reemplazar el WRITE.OBJ por una nueva versión en la biblioteca OSO.LIB, se ejecutaría:

```
TLIB OSO +CREAPROC-READ-+WRITE
```

Si la lista es muy larga se puede incluir en un fichero y ejecutar TLIB @fichero para que se lea del mismo (si no cabe en una línea del fichero, puede escribirse & al final antes de pasar a la siguiente línea).

Para ver el contenido de la biblioteca OSO.LIB por la consola ejecutaríamos:

```
TLIB OSO,CON
```

que nos dice, por cada módulo .OBJ contenido en OSO.LIB:

- su tamaño en bytes.
- referencias (etiquetas de código o de datos) públicas ofrecidas a otros módulos.

3. La biblioteca del sistema OSO.LIB

Para la construcción de la biblioteca del sistema, vamos a implementar tres funciones, que colocaremos en tres ficheros .c distintos, y un fichero cabecera .h con los prototipos de estas tres funciones. Cada una de esas

funciones se corresponderá con una de las tres llamadas al sistema que implementa la versión por defecto de OSO. A continuación se muestra el código fuente de estos cuatro ficheros que se encuentran en el directorio `oso/aplis/osolib` de los fuentes de OSO:

```

#include "oso.h"

int crearproceso (char far * programa)
{
    asm {
        les bx, dword ptr programa
        mov ah, 0
        int 22h
    }
}

/* $Id: crearproc.c 983 2005-04-19 18:13:02Z piernas $ */

```

```

#include "oso.h"

int read(int fd, char far * buffer, int count)
{
    asm {
        mov ah, 1
        mov cx, count
        mov dx, fd
        les bx, dword ptr buffer
        int 22h
    }
}

/* $Id: read.c 983 2005-04-19 18:13:02Z piernas $ */

```

```

#include "oso.h"

int write(int fd, char far * buffer, int count)
{
    asm {
        mov ah, 2
        mov cx, count
        mov dx, fd
        les bx, dword ptr buffer
        int 22h
    }
}

/* $Id: write.c 983 2005-04-19 18:13:02Z piernas $ */

```

```

#ifndef __OSO_H
#define __OSO_H

#define NULL (void *)0

int crearproceso (char far * programa);
int read (int fd, char far * buffer, int count);
int write (int fd, char far * buffer, int count);

#endif

/* $Id: oso.h 983 2005-04-19 18:13:02Z piernas $ */

```

En los tres primeros ficheros encontramos las implementaciones de las funciones `crearproceso`, `read` y `write` que realizan las tres llamadas al sistema existentes: crear proceso, leer de un fichero (en la versión actual, sólo del teclado) y escribir en un fichero (actualmente, sólo en la pantalla).

Para construir la biblioteca `OSO.LIB` debemos compilar cada módulo con `TCC`, para crear el fichero `.OBJ` correspondiente, y después juntarlos todos en un único fichero con `TLIB`. Todo esto se hace de forma automática mediante el siguiente fichero `Makefile` que también se encuentra en el directorio anterior:

```
all: oso.lib

oso.lib: creaproc.obj read.obj write.obj
       tlib oso.lib + creaproc + read + write

creaproc.obj: creaproc.c
           tcc -mt! -c creaproc.c

read.obj: read.c
         tcc -mt! -c read.c

write.obj: write.c
         tcc -mt! -c write.c

clean:
      del *.obj
      del *.map
      del *.lib

# $Id: Makefile 983 2005-04-19 18:13:02Z piernas $
```

4. La biblioteca del lenguaje `UTIL.LIB`

La biblioteca del lenguaje (o *biblioteca de C* en nuestro caso) puede ser tan grande como queramos, sólo es cuestión de implementar funciones que realicen cierta actividad útil apoyándose o no en otras funciones.

A continuación mostramos cómo se puede construir una biblioteca del lenguaje que sea apropiada para `OSO`. Nuestra biblioteca implementa 11 funciones: 8 para el manejo de cadenas, 1 para leer de teclado y 2 para la salida por pantalla.

Las funciones para el manejo de cadenas son: `itoa`, `strcat`, `strchr`, `strcmp`, `strcpy`, `strlen`, `strncmp` y `strncpy`. Estas funciones hacen exactamente lo mismo que las funciones del mismo nombre que podemos encontrar en cualquier biblioteca estándar de C.

La función para leer de teclado es `gets`. Esta función recibe dos argumentos: un puntero a una zona de memoria en donde se guardará la cadena leída (la cual terminará en `'\0'` y no incluirá el carácter de retorno de carro) y un número máximo de caracteres a leer (que no podrá ser mayor que el tamaño de la zona de memoria en donde se almacenarán los caracteres). La función devolverá el número de caracteres que se hayan podido leer de teclado antes de pulsar la tecla «return».

La dos funciones para escribir en pantalla son `puts` y `printf`. La primera función no devuelve nada y recibe un único parámetro que es la cadena a mostrar por pantalla. Esta cadena puede contener, si se desea, la secuencia de caracteres `"\r\n"` que produce un salto de línea y un retorno de carro. La segunda función hace exactamente lo mismo que la función del mismo nombre de la biblioteca estándar de C, aunque nuestra implementación tiene algunas limitaciones. Las limitaciones más importantes son que «sólo» soporta los caracteres de conversión `'c'`, `'s'`, `'p'`, `'n'`, `'o'`, `'x'`, `'X'`, `'d'`, `'i'` y `'u'`, y que no es capaz de mostrar enteros de tipo `long`.

A continuación se muestra el código fuente de los 11 módulos `.c` (uno por cada función de la biblioteca) y del fichero `util.h`, que contiene el prototipo de las 11 funciones y algunas macros que son necesarias para la

implementación de la función `printf`. Estos ficheros se encuentran en el directorio `oso/aplis/utillib` de los fuentes de OSO:

```
#include "util.h"

char * itoa(char * s, int n)
{
    char * p = s, * t = s, c;                                5

    if (n < 0)
    {
        *s++='-' ;
        n *= -1;                                           10
    }
    while(n > 0)
    {
        *s++ = (char)(n% 10 + '0');
        n /= 10;                                           15
    }
    if (s == p)
    {
        *s++ = '0';
        *s = '\0';
        return p;                                           20
    }
    *s-- = '\0';
    if (*t == '-')
        t++;
    while(t < s)
    {
        c = *t;
        *t++ = *s;
        *s-- = c;                                           30
    }
    return p;
}

/* $Id: itoa.c 983 2005-04-19 18:13:02Z piernas $ */          35
```

```
#include "util.h"

char * strcat(char *d, char *s)
{
    char *t = d;                                           5

    while(*d++)
        ;

    d--;
    while(*d = *s)
    {
        d++;
        s++;
    }
    return t;
}

/* $Id: strcat.c 983 2005-04-19 18:13:02Z piernas $ */      20
```

```

#include "util.h"

char * strchr(char *s, char c)
{
    while (*s && *s != c)
        s++;
    if (*s || c == '\0')
        return s;
    return NULL;
}
/* $Id: strchr.c 983 2005-04-19 18:13:02Z piernas $ */

```

```

#include "util.h"

int strcmp(char *r, char *s)
{
    while (*r && *r == *s)
    {
        r++;
        s++;
    }
    return (*r - *s);
}
/* $Id: strcmp.c 983 2005-04-19 18:13:02Z piernas $ */

```

```

#include "util.h"

char * strcpy(char *d, char *s)
{
    char *t = d;
    while (*d++ = *s++)
        ;
    return t;
}
/* $Id: strcpy.c 983 2005-04-19 18:13:02Z piernas $ */

```

```

#include "util.h"

unsigned int strlen(char * s)
{
    unsigned int n = 0;
    while(*s++)
        n++;
    return n;
}
/* $Id: strlen.c 983 2005-04-19 18:13:02Z piernas $ */

```

```

#include "util.h"

int strcmp(char *r, char *s, unsigned int n)
{
    if (n == 0)
        return 0;

    while (--n && *r && *r == *s)
    {
        r++;
        s++;
    }

    return (*r - *s);
}

/* $Id: strcmp.c 983 2005-04-19 18:13:02Z piernas $ */

```

```

#include "util.h"

char * strcpy(char *d, char *s, unsigned int n)
{
    char *t = d;

    while (n--)
    {
        *d++ = *s;
        if (*s)
            s++;
    }

    return t;
}

/* $Id: strcpy.c 983 2005-04-19 18:13:02Z piernas $ */

```

```

#include "..\osolib\oso.h"
#include "util.h"

int gets(char * cadena, unsigned int max)
{
    char c;
    unsigned int contador = 1;

    if (max == 0)
        return -1;

    while (contador < max)
    {
        read(0, cadena, 1);
        if (*cadena == '\r')
        {
            puts("\n\r");
            *cadena = '\0';
            return --contador;
        }
        write(1, cadena, 1);
        cadena++;
        contador++;
    }
}

```

```

do
{
    read(0, &c, 1);
} while (c != '\r');
puts("\n\r");
*cadena = '\0';
return --contador;
}
/* $Id: gets.c 983 2005-04-19 18:13:02Z piernas $ */

```

```

#include "..\osolib\oso.h"
#include "util.h"

void puts(char * s)
{
    write(1, s, strlen(s));
}
/* $Id: puts.c 983 2005-04-19 18:13:02Z piernas $ */

```

```

#include "..\osolib\oso.h"
#include "util.h"

/* Memoria privada donde "printf" construye el resultado. */
static char bufprivado[1024];

/* Función auxiliar que permite extraer el siguiente número de la cadena
 * dada por "*s". Si el primer carácter es "*", se entiende que el número
 * es uno de los argumentos de la lista "*pargs". */
static int extrae_numero(const char **s, va_list * pargs)
{
    int i = 0;

    if (**s == '*')
    {
        ++(*s);
        return va_arg(*pargs, int);
    }

    while (isdigit(**s))
        i = i * 10 + *((*s)++) - '0';
    return i;
}

/* Función auxiliar que divide un número "n" por una base, devolviendo
 * la función el resto de la división entera y actualizando '*n' al
 * cociente de dicha división. */
static int dividir(int * n, int base) {
    unsigned resto;

    resto = ((unsigned int) *n) % (unsigned int) base;
    *n = ((unsigned int) *n) / (unsigned int) base;
    return resto;
}

#define CEROS    0x0001    /* Rellena con ceros. */
#define CONSIGNO 0x0002    /* Entero con signo. */
#define MAS      0x0004    /* Muestra un '+' en cantidades positivas. */
#define ESPACIO  0x0008    /* Deja un ' ' antes de cantidades positivas. */
#define IZQUIERDA 0x0010   /* Justifica a la izquierda. */

```

```

#define ESPECIAL 0x0020      /* Añade 0 si se especifica 'o', y 0x o 0X si
                             se especifica 'x' o 'X', respectivamente.*/
#define MAYUSCULAS 0x0040  /* Usa 'ABCDEF' en lugar de 'abcdef' */
#define PRECISION 0x0080   /* Se ha especificado una precisión. */
45

/* Funcion auxiliar para convertir números a cadenas con distintos
 * formatos. */
static char * trata_numero(char * str, int num, int base, int anchura,
                           int precision, int opciones)
{
    char c, signo, tmp[32];
    char * digitos = "0123456789abcdefx";
    int i;

    if (base < 2 || base > 16)
        return NULL;
55

    if (opciones & MAYUSCULAS)
        digitos = "0123456789ABCDEFX";
60

    if (opciones & IZQUIERDA)
        opciones &= ~CEROS;

    c = (opciones & CEROS) ? '0' : ' ';
65

    signo = 0;
    if (opciones & CONSIGNO)
    {
        if (num < 0)
        {
            signo = '-';
            num = -num;
            anchura--;
        }
        else if (opciones & MAS)
75
        {
            signo = '+';
            anchura--;
        }
        else if (opciones & ESPACIO)
80
        {
            signo = ' ';
            anchura--;
        }
    }
85

    if (opciones & ESPECIAL)
    {
        if (base == 16)
            anchura -= 2;
        else if (base == 8)
            anchura--;
90
    }

    i = 0;
95
    while (num != 0)
        tmp[i++] = digitos[dividir(&num,base)];

    if (i > precision)
        precision = i;
100
    anchura -= precision;

    if (!(opciones & (CEROS | IZQUIERDA)))
        while(anchura-- > 0)
            *str++ = ' ';
105

```

```

if (signo)
    *str++ = signo;

if (opciones & ESPECIAL) {
    if (base == 8)
        *str++ = '0';
    else if (base == 16)
    {
        *str++ = '0';
        *str++ = digitos[16];
    }
}

if (!(opciones & IZQUIERDA))
    while (anchura-- > 0)
        *str++ = c;

while (i < precision-- )
    *str++ = '0';

while (i-- > 0)
    *str++ = tmp[i];

while (anchura-- > 0)
    *str++ = ' ';

return (str);
}

/* Funcion auxiliar que analiza la cadena "fmt" sustituyendo los elementos
 * "%" que encuentra por el valor correspondiente. */
static int procesa_argumentos (const char * buf, const char * fmt, va_list args)
{
    unsigned int num;
    int i, base, len, * ip;
    char * str, * previo;
    const char * s;
    int opciones;      /* Opciones a tener en cuenta */
    int anchura;      /* Ancho del campo de salida */
    int precision;    /* N° mínimo de dígitos para números;
                       N° máximo de caracteres para cadenas */

    for (str = buf; *fmt; ++fmt)
    {
        if (*fmt != '%')
        {
            *str++ = *fmt;
            continue;
        }
        previo = fmt;

        /* Obtenemos el carácter de opción: +, -, #, 0 o ' '. */
        opciones = 0;
        repetir:
            ++fmt;
            switch (*fmt)
            {
                case '#':
                    opciones |= ESPECIAL;
                    goto repetir;
                case '0':
                    opciones |= CEROS;
                    goto repetir;
                case '-':

```

```

        opciones |= IZQUIERDA;
        goto repetir;
    case ' ':
        opciones |= ESPACIO;
        goto repetir;
    case '+':
        opciones |= MAS;
        goto repetir;
}
175

/* Obtenemos la anchura del campo. */
anchura = extrae_numero(&fmt, &args);
if (anchura < 0)
{
    anchura = -anchura;
    opciones |= IZQUIERDA;
}
180

/* Obtenemos la precisión. Por omisión es 1. */
precision = 1;
if (*fmt == '.')
{
    opciones |= PRECISION;
    ++fmt;
    precision = extrae_numero(&fmt, &args);
    if (precision < 1)
        precision = 0;
}
185

/* Tratamos el carácter que indica la conversión. */
switch (*fmt) {
    case 'c':
        if (!(opciones & IZQUIERDA))
            while (--anchura > 0)
                *str++ = ' ';
        *str++ = (unsigned char) va_arg(args, int);
        while (--anchura > 0)
            *str++ = ' ';
        continue;
    case 's':
        s = va_arg(args, char *);
        if (!s)
            s = "(null)";
        len = strlen(s);
        if (opciones & PRECISION)
            len = len > precision ? precision : len;
        if (!(opciones & IZQUIERDA))
            while (len < anchura--)
                *str++ = ' ';
        for (i = 0; i < len; i++)
            *str++ = *s++;
        while (len < anchura--)
            *str++ = ' ';
        continue;
    case 'p':
        str = trata_numero(str, (unsigned int) va_arg(args, void *), 16,
            anchura, precision, opciones | ESPECIAL);
        continue;
    case 'n':
        ip = va_arg(args, int *);
        *ip = (str - buf);
        continue;
    case 'o':
        base = 8;
        break;
    case 'x':
190
195
200
205
210
215
220
225
230
235

```

```

        opciones |= MAYUSCULAS;
    case 'x':
        base = 16;
        break;
    case 'd':
    case 'i':
        opciones |= CONSIGNO;
    case 'u':
        base = 10;
        break;
    default:
        while (previo <= fmt)
            *str++ = *previo++;
        continue;
}

num = va_arg(args, unsigned int);
str = trata_numero(str, num, base, anchura, precision, opciones);
}

*str = '\\0';

return str - buf;
}

/* Funcion "printf". Se apoya en la función auxiliar "procesa_argumentos"
 * que es la que hace realmente todo el trabajo. */
int printf(char *fmt, ...)
{
    va_list args;
    char * pcar;
    int total;

    va_start(args, fmt);
    total = procesa_argumentos (bufprivado, fmt, args);
    va_end(args);

    pcar = bufprivado;
    while (*pcar)
    {
        if (*pcar == '\\n')
            write(1, &"\\x", 1);
        write(1, pcar++, 1);
    }

    return total;
}

/* $Id: printf.c 1024 2005-05-18 13:30:20Z piernas $ */

```

```

#ifndef __UTIL_H
#define __UTIL_H

#define NULL (void *)0

/* Prototipos de las funciones implementadas */
int gets (char * cadena, unsigned int max);
char * itoa(char * cadena, int numero);
void puts(char * cadena);
char * strcat(char *d, char *s);
char * strchr(char *s, char c);
int strcmp(char *r, char *s);
char * strepy(char *d, char *s);
unsigned int strlen (char * cadena);

```

```

int strcmp(char *r, char *s, unsigned int n);
char * strcpy(char *d, char *s, unsigned int n);
int printf(char * fmt, ...);

/* Macro para comprobar si un carácter es dígito o no */
#define isdigit(c) ((c) >= '0' && (c) <= '9')

/* Macros para acceder a una lista variable de argumentos en una función.
 * Estas macros con "va_start", "va_arg" y "va_end". Definimos primero el
 * tipo para la lista variable de argumentos. */
typedef void * va_list;

/* va_start se define como una función que devuelve "void". Simplemente,
 * tenemos que hacer que "la" contenga la dirección de "param" más el
 * tamaño del tipo de "param". Puesto que el tipo puede ser "char" y en la
 * pila siempre se apila en múltiplos de palabra, debemos asegurarnos de que
 * el tamaño del tipo sea múltiplo de palabra. Eso se consigue sumando lo que
 * hay en la 6ª línea. */
#define va_start(la, param)\
    (\
        (void)\
        (\
            (la) = (va_list)((char *)&(param) +\
                ((sizeof(param) + sizeof(int) - 1) & ~(sizeof(int) - 1)))\
        )\
    )

/* Esta es complicada. En teoría habría que devolver sólo "(tipo *)(la)". El
 * problema es que "la" se debe quedar apuntando al siguiente parámetro. Por
 * eso, en la 5ª línea actualizamos "la". Esta línea devuelve un puntero que
 * es "la" más el tamaño del tipo. A este puntero hay que restarle lo que se
 * ha incrementado (6ª línea) para obtener el valor deseado. */
#define va_arg(la, tipo)\
    (\
        *(tipo *)\
        (\
            (\
                (char *)la += ((sizeof(tipo) + sizeof(int) - 1) & ~(sizeof(int) - 1))\
            )\
            - ((sizeof(tipo) + sizeof(int) - 1) & ~(sizeof(int) - 1))\
        )\
    )

/* va_end se define como una función que devuelve "void", de ahí la
 * definición no sea "{}" sino lo que aparece a continuación. */
#define va_end(la) ((void)0)

#endif

/* $Id: util.h 1024 2005-05-18 13:30:20Z piernas $ */

```

Para construir la biblioteca UTIL.LIB correspondiente, podemos hacer uso también del fichero Makefile que encontraremos en el directorio oso/aplis/utillib.

5. Aplicaciones de ejemplo

En esta sección vamos a describir dos programas de ejemplo. El primero, ASCII.COM, es un pequeño programa que no hace uso de ninguna biblioteca y que muestra indefinidamente todos los caracteres ASCII. El segundo, SHELL.COM, es un pequeño intérprete de órdenes que, además, será el primer programa en «espacio de usuario» ejecutado por el sistema operativo OSO.

5.1. El programa `ascii.c`

El programa `ascii.c` es muy sencillo ya que no hace uso de ninguna biblioteca, simplemente, muestra indefinidamente todos los caracteres ASCII, uno tras otro, en una posición establecida de la pantalla. Su código fuente es el siguiente:

```
/* Sencillo programa que muestra de forma indefinida los
 * caracteres ASCII en una posición concreta de la pantalla,
 * escribiendo directamente en la RAM de video. */
void main(void){
    unsigned long i;
    unsigned char j;
    unsigned char far *pantalla = (char far *)0xb8000000;

    for(;;) {
        for (j = 0; j < 255; j++) {
            for (i = 0; i < 2000000; i++){
                pantalla[2500] = j;
            }
        }
    }
}

/* $Id: ascii.c 983 2005-04-19 18:13:02Z piernas $ */
```

El fichero ejecutable `ASCII.COM` se construye a partir de dos módulos objeto: `ASCII.OBJ` y el módulo de arranque `C0T.OBJ`. El código fuente de este segundo módulo es:

```
model tiny
.code
    EXTRN _main:NEAR
.startup
    call _main
infinito:
    jmp infinito
END

; $Id: c0t.asm 983 2005-04-19 18:13:02Z piernas $
```

Como podemos ver, si se regresa de la función `main` el programa quedará ejecutando un bucle infinito. Esto se hace así porque no existe la llamada al sistema `exit` que permita terminar correctamente la ejecución de un proceso.

El módulo `C0T.OBJ` también se utilizará para construir el ejecutable del siguiente programa y podrá ser utilizado en la construcción de otros programas, aunque habrá que modificarlo adecuadamente cuando se añada la llamada al sistema `exit`.

5.2. El programa `shell.c`

Lo último que hace el sistema operativo OSO al inicio de su ejecución es cargar en memoria y ejecutar el programa `SHELL.COM`. El proceso creado tiene como PID 1 y es el primer proceso que se ejecuta en «modo usuario». Este proceso desempeña un papel similar al del proceso `init` de los sistemas Unix.

En nuestro caso, el programa `shell.c` es un pequeño intérprete de órdenes que ejecuta el programa cuyo nombre se introduce por teclado cuando se pulsa la tecla «return».

Ya que el programa se ejecuta en «modo usuario», no puede acceder directamente al hardware¹ y debe hacer

¹Al menos en teoría, no debería acceder directamente al hardware. La realidad es que sí puede al ejecutarse en el «modo real» del procesador que no ofrece ningún tipo de protección.

uso de los servicios proporcionados por el sistema operativo a través de la interfaz de llamadas al sistema.

Vamos a mostrar ahora el fuente del fichero `shell.c` y después comentaremos los aspectos más interesantes:

```
/* Pequeño intérprete de órdenes. Ejecuta el programa cuyo nombre se
 * introduce por teclado y cuando termina (se pulsa sólo return) se
 * queda indefinidamente dibujando un "molinillo" en la tercera
 * posición de la esquina superior izquierda de la pantalla. El código
 * máquina de este programa (SHELL.COM) es el código del primer
 * proceso que ejecuta el sistema operativo OSO cuando termina de
 * cargarse. */
#include "osolib\oso.h"
#include "utilib\util.h"

static char molinillo[] = {'|', '/', '-', '\\'};

void main(void)
{
    unsigned char j = 0;
    unsigned char far *pantalla = (char far *)0xb8000000;
    char cadena[80];

    for(;;)
    {
        printf("Orden (return para terminar) >");
        gets(cadena, 80);
        if (*cadena == '\0')
            break;
        if (crearproceso(cadena) < 0)
        {
            printf("Error al ejecutar <%s>\n", cadena);
        }
    }

    for(;;)
    {
        pantalla[4] = molinillo[j];
        j = (j + 1) % 4;
    }
}

/* $Id: shell.c 1024 2005-05-18 13:30:20Z piernas $ */
```

La única función que existe, la función `main`, hace uso de las funciones que se implementan en la biblioteca del sistema y en la biblioteca del lenguaje para desempeñar su trabajo. Para usar correctamente las funciones de estas bibliotecas, es **fundamental** incluir sus respectivos ficheros cabecera (líneas 8 y 9 del fichero fuente).

Cuando el programa «termina» (porque se introduce una línea en blanco) se queda ejecutando un bucle infinito en el que se accede «directamente» a la memoria de vídeo para dibujar un molinillo en la tercera posición de la esquina superior izquierda de la pantalla.

Para crear el ejecutable `SHELL.COM`, podemos hacer uso del siguiente fichero `Makefile` que encontraremos en el directorio `oso/aplis` de los fuentes de nuestro sistema operativo:

```
all: shell.com ascii.com

copia: all
    copy shell.com a:
    copy ascii.com a:

c0t.obj: c0t.asm
    tcc -mt! -c c0t.asm

shell.com: c0t.obj shell.obj osolib\oso.lib utilib\util.lib
```

```

tlink /t c0t.obj shell.obj, shell.com,, osolib\oso.lib utillib\util.lib

shell.obj: shell.c
tcc -mt! -c shell.c
15

ascii.com: c0t.obj ascii.obj
tlink /t c0t.obj ascii.obj, ascii.com

ascii.obj: ascii.c
tcc -mt! -c ascii.c
20

clean:
del *.obj
del *.map
del *.com
25

# $Id: Makefile 1019 2005-05-12 09:54:46Z piernas $

```

De este fichero es interesante observar el uso de las bibliotecas en las líneas 10 y 11 para la construcción del ejecutable.