LIBRERíA DE SOPORTE AL LABORATORIO S.O.

SISTEMA OPERATIVO ONION V.4.0.1

Artiaga, Ernest
Doreste, Luis
García, Jordi
Herrero, Josep R.

Setembre 98

Includes

new_maq.h

```
#ifndef NEW_MAQH
#define NEW_MAQH


/* variables globals */
  int *pantalla;
  int *reset_bios = (int *)0xFFFF0000;
  int addr_6845;

  /* NUCLEO SE ENCARGA DE INICIALIZAR... */

  int p_stat_imp;
  int p_dat_imp;
  int p_cont_imp;

              unsigned char taula_teclat[256] = {
-1, 27, '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '\'', '-', '\b', /
*15*/
'\t', 'q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p', '[', ']', '\n',    /
*29*/
-1, 'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l', '¤', '{', '§', -1, '}',   /
*44*/
'z', 'x', 'c', 'v', 'b', 'n', 'm', ',', '.', '-', -1, '*', -1, ' ',      /*58*/
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 26, '7', '8',           /*73*/
'9', '-', '4', '5', '6', '+', '1', '2', '3', '0', '.', -1, -1, '<', -1, -1,
-1, -1,    /*89*/
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,    /
*107*/
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,    /
*125*/
-1,

-1, -1, '!', '"', 'ú', '$', '%', '&', '/', '(',  ')', '=', '?', '¨', '\b', /
*15*/
-1, 'Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P', -1, -1, '\n',        /
*29*/
-1, 'A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', '¥', -1, -1, -1, -1,      /
*44*/
'Z', 'X', 'C', 'V', 'B', 'N', 'M', ';', ':', '_', -1, '*', -1, ' ', -1, -1,/
*60*/
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 26, '7', '8', '9', '-', '4', '5', '6',
/*78*/
'+', '1', '2', '3', '0', '.', -1, -1, '>', -1, -1, -1, -1, -1, -1, -1, -1, -
1,    /*96*/
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,    /
*114*/
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 };

#endif
```

nucleo.h

```c
/**** Uso de procesos */
struct info_proc {
              int  quantum;
              int  prioritat;
              long t_cpu;
};
void multiplexar ();
/* crear_pcb_nuc(rutina, quantum, prioritat) */
int  crear_pcb_nuc (void (*)(), int, int);
int  destruir_pcb_nuc (int);
int  quisoc (void);
int  info_proc_nuc (int, struct info_proc *);
int  modif_proc_nuc (int, struct info_proc *);

/**** Sincronizacion */
int wait (int);
int signal (int);
int init_sem (int, int);
int sleep (char *);
void wakeup (char *);
void preemtion ();
void no_preemtion ();

/**** Teclado */
char llegir_teclat_nuc ();

/**** Pantalla */
int escriure_pantalla_nuc (int, int, char, unsigned char);
int scroll_nuc (int, int);

/**** Linea serie */
char llegir_l_s_nuc ();
void escriure_l_s_nuc (char);

/**** Impresora */
int escriure_impressora_nuc (char);

/**** Disco */
int motor_on_nuc (int);
void motor_off_nuc ();
int posicionar_pista_nuc (int, int);
int llegir_sector_nuc (int, int, char *);
int escriure_sector_nuc (int, int, char *);
int recalibrar_nuc (int);
void retardar_nuc (int);

/**** Uso de memoria */
char *(assignar_mem (int,long));
int alliberar_mem (int, char *, long);
long mem_lliure (int);

/**** Otros */
void beep ();
void reset ();
```


rut_nuc.h

```c
/**** Definición de constantes */

#define MAXFLOPPIES     2
#define OP_LECTURA      0x00
#define OP_ESCRIPTURA   0x01

/**** Definición tipos de datos */

typedef struct {
  int sectors_track;
  int sides;
  int tracks_side;
  int factor;
  int gap;
  int rate;
  int specify;
  char *name;
} DISKINFO;

/**** Macros útiles */

#define FLOPPY_VALID( d ) (( d >= 0 ) && ( d < MAXFLOPPIES ) && \
  ( floppy_type[d] != NULL ))

#define PISTA_VALIDA( d, p ) ( FLOPPY_VALID( d ) && \
  ( p >= 0 ) && ( p < floppy_type[d]->tracks_side ))

#define SECTOR_VALID( d, p, c, s ) ( PISTA_VALIDA( d, p ) && \
  ( c >= 0 ) && ( c < floppy_type[d]->sides ) && \
  ( s >= 1 ) && ( s <= floppy_type[d]->sectors_track ))

#define ST0 (info_disc[0])
#define ST1 (info_disc[1])
#define ST2 (info_disc[2])
#define ST3 (info_disc[3])
#define ST4 (info_disc[4])
#define ST5 (info_disc[5])
#define ST6 (info_disc[6])
extern int inf_pos [2];
extern int inf_trans [7];

/**** Variables globales */

extern int info_disc[7];
extern DISKINFO *floppy_type[MAXFLOPPIES];

/**** Manipulación de interrupcciones */
int inhibir ();
void desinhibir (int);
int in_port (int);
void out_port (int, int);
void vector_int (int, void (*),());
void strobe (int, char );
void eoi ();
int emmascarar (int);
```

```
/**** Rutinas de teclado */
char llegir_teclat ();

/**** Rutinas de pantalla */
void escriure_pantalla (int, int, char, unsigned char);
void posicionar_cursor (int, int);
void scroll (int, int);

/**** Rutinas de impresora */
int escriure_impressora (char);

/**** Rutinas de disco */
void motor_on (int);
void motor_off ();
void posicionar_pista (int, int);
void programar_disc (int, int, int, int, int);
int programar_dma (int, char *);
void comanda_disc (int);
int comprovar_posicionat ();
int comprovar_transferenia ();
int llegir_estat ();
void recalibrar (int);


/**** Rutina de reloj */
void timer (int);


/**** Rutinas de la linea serie */
void ini_ls (int);
int escriure_ls (char);
char llegir_ls ();
int consulta_ls_iir();
int consulta_ls_lsr();
int consulta_ls_msr();


/**** Rutinas varias */
void beep ();
void reset ();
```


Rutines.h

```
#ifndef RUTINES
#define RUTINES

/**** Rutinas necesarias para la utilizacion del debug */
void bug();

/**** Manipulación de strings */
int longitud (char *);
void copiar (char *, char *, int);
int comparar (char *, char *, int);
int buscar (char *, char, int);
void majuscula (char *, int);
void minuscula (char *, int);
void omplir(char *, char, int);

/**** Manipulación de las direcciones */
unsigned long distancia (char *, char *);
char *(cota(char *, long));

/**** Manipulación de mapa de bits */
int get_bit (char *, int);
void set_bit(char *, int, int);

/**** Rutinas de conversión */
int long_a_ascii (long, char *, int, int);
int int_a_ascii (int, char *, int, int);
long ascii_a_long  (char *, int, int);
int ascii_a_int (char *, int, int);

/**** Rutinas de manipulación  de colas */

struct item
              {
              struct item *seguent;
              struct item *anterior;
              unsigned long pes;
              };
struct cua
              {
              struct item *primer;
              struct item *ultim;
              };

void inicialitzar_cua (struct cua *);
void encuar (struct cua *, struct item *);
struct item *(primer (struct cua *));
int buida (struct cua *);
struct item *(extreure (struct cua *, unsigned long));
struct item *(buscar_item (struct cua *, unsigned long));
struct item *(seguent (struct cua *, struct item *));
struct item *(anterior (struct cua *, struct item *));
void insertar (struct cua *, struct item *, unsigned long);
struct item *(extirpa (struct cua *, struct item *));
struct item *(cap(struct cua*));

#endif
```

RUT NUC

Rutinas de uso de las interrupciones

int inhibir();

Descripción: **Inhibe las interrupcciones. Devuelve el valor de los flags antes de inhibir las interrupciones.**

Código:

```
SOLIB_TEXT SEGMENT BYTES PUBLIC 'CODE'
assume cs:SOLIB_TEXT

public _inhibir
_inhibir proc far

              pushf
              cli
              pop ax
              ret

_inhibir endp

SOLIB_TEXT ENDS
END
```

void desinhibir (flags)
int flags;

Descripción: **Carga la palabra de estado con el parámetro flags, posiblemente para desinhibir las interrupcciones.**

Código:

```
FLAGS EQU [bp+6]

SOLIB_TEXT SEGMENT BYTES PUBLIC 'CODE'
assume cs:SOLIB_TEXT

public _desinhibir
_desinhibir proc far

              push          bp
              mov           bp, sp
              push          ax
              mov           ax, FLAGS
              push          ax
              popf
              pop           ax
              pop           bp
              ret

_desinhibir endp

SOLIB_TEXT ENDS
END
```


void vector_int (num_int, rutina)
int num_int;
void (*rutina)();

Descripción: **Procedimiento que inicializa el vector de interrupciones. Recibe como parámetro el numero de la interrupci¢n (8 para el reloj, 9 para el teclado....) y un puntero lejano (tipo far; CS:IP) a la rutina de servicio de la interrupción.**

Código:

```
void vector_int(num_int, rutina)
int num_int;               /* numero de interrupción */
void (*rutina)();          /* direccion de la rutina de servicio */
         {
         long *vect_int;

         (long)vect_int = (long)(num_int * 4);
         *vect_int = (long)rutina;
         }
```

int in_port (port)
int port;

Descripción: **Lee de un puerto. Lee el contenido del puerto que se le pasa como parámetro.**

Código:

```
PORT EQU [bp+6]

SOLIB_TEXT SEGMENT BYTES PUBLIC 'CODE'
assume cs:SOLIB_TEXT

public _in_port
_in_port proc far

              push          bp
              mov           bp, sp
              push          dx
              mov           dx, PORT
              in            al, dx
              xor           ah, ah
              pop           ax
              pop           bp
              ret

_in_port endp

SOLIB_TEXT ENDS
END
```

void out_port (port, valor)
int port;
int valor;

Descripción: **Escribe en un puerto. Escribe el valor en el puerto que se le pasa como parámetro.**

Código:

```
PORT EQU [bp+6]
VALOR EQU [bp+8]

SOLIB_TEXT SEGMENT BYTES PUBLIC 'CODE'
assume cs:SOLIB_TEXT

public _out_port
_out_port proc far

              push          bp
              mov           bp, sp
              push          dx
              push          ax
              mov           dx, PORT
              mov           ax, VALOR
              out           dx, al
              pop           ax
              pop           dx
              pop           bp
              ret

_out_port endp

SOLIB_TEXT ENDS
END
```

void eoi ();

Descripción: **Procedimiento envía al puerto 20H el valor 20H, con lo cual indicamos la finalización de una interrupción de tipo general.**

Código:

```
#define PORT_INT 0x20
#define EOI 0x20

#include "rut_lib.h"

void eoi()
              {
              out_port(PORT_INT, EOI);
              }
```


void strobe (port, mascara)
int port;
char mascara;

Descripción: **Procedimiento que efectúa el strobe sobre un puerto. Se realiza escribiendo primero uno y después ceros en los bits indicados en la máscara.**

Código:

```
PORT EQU [bp+6]
MASC EQU [bp+8]

SOLIB_TEXT SEGMENT BYTES PUBLIC 'CODE'
assume cs:SOLIB_TEXT

public _strobe
_strobe proc far

              push          bp
              mov           bp, sp
              push          dx
              push          ax
              mov           dx, PORT
              in            al, dx
              xor           ax, MASC
              out           dx, al
              nop
              nop
              xor           ax, MASC
              out           dx, al
              pop           ax
              pop           dx
              pop           bp
              ret

_strobe endp

SOLIB_TEXT ENDS
END
```

Rutina de teclado

char llegir_teclat()

Descripción: **Procedimiento que lee un carácter del teclado. Se encarga de leer del puerto de datos del teclado y detecta si la tecla esta pulsada o si ya ha sido liberada. Devuelve el carácter ASCII en el primer caso y un -1 en el otro caso.**

Código:

```
#define DAT_TEC 0x60

#include "rut_lib.h"

extern char taula_teclat[];

char llegir_teclat()
              {
              register int i;

              i=in_port(DAT_TEC);
              strobe(0x61, 0x80);
              return(i & 0x80) ? -1 : taula_teclat[i-1(;
              }
```


Rutina de pantalla

void escriure_pantalla(fila, columna, caracter, atributo)
int fila, columna;
char caracter,
unsigned char atributo;

Descripción: **Procedimiento que escribe un carácter con su atributo en la posición indicada por los parámetros fila (0 a 24) y columna (0 a 79).**

El atributo del carácter puede ser:

```
NORMAL -------------->0x7
INVERSE ------------->0x70
BLINK---------------->0x87
UNDERSCORE ---------->0x5f
```

Código:

```
#define POSICIO fila*80+columna

#include "rut_lib.h"

extern int *pantalla;

void escriure_pantalla ( fila, columna, caracter, atributo)
int fila, columna;
char caracter,
unsigned char atributo;
              {
              *(pantalla+POSICIO)=caracter+(atributo<<8);
              }
```

Rutina del reloj

void timer(comptador)
int comptador;

Descripción: **Procedimiento que inicializa el controlador del reloj. Se le pasa como parámetro un entero que indica el numero de ciclos básicos de reloj. Si el computador tiene el valor máximo (65535) el tic será de 50 milisegundos aproximadamente.**

Código:

```
#define TIM_COMAN 0x36
#define TIM_CTL 0x43
#define TIMER_0 0x40
#define PORT_INT 0x20
#define TIMER_EOI 0x60

#include "rut_lib.h"

void timer(c)
int c
              {
              register int f;

              f=inhibir();
              out_port(TIM_CTL, TIM_COMAN);
              out_port(TIMER_0, c);
              out_port(TIMER_0, c >>8);
              out_port(PORT_INT,TIMER_EOI);
              desinhibir(f);
              }
```


Rutinas varias

void beep ()

Descripción: **Procedimiento que hace sonar el altavoz de la máquina.**

Código:

```
#define CONTTEC 0x61

#include "rut_lib.h"

void beep()
     {
     register int i,j;
     int f;

     f=inhibir();
     for(i=0;i<=100;i++)
            {
            out_port(CONTTEC,(in_port(CONTTEC) & 0xfe) | 2);
            for(j=0;j<=50;j++);
            out_port(CONTTEC,in_port(CONTTEC) & 0xfd);
            for(j=0;j<=50;j++);
            }
     desinhibir(f);
     }
```

void reset()

Descripción: **Función que realiza el boot en caliente de la máquina, cargando el DOS desde disco.**

Código:

```
extern_reset_bios:DWORD

SOLIB_TEXT SEGMENT BYTES PUBLIC 'CODE'
assume cs:SOLIB_TEXT

_reset proc far
public _reset
              push          ds
              mov           ax, 40h
              mov           ds, ax
              mov           ds:72h, 1234h
              pop           ds
              jmp           dword ptr _reset_bios
_reset endp
SOLIB_TEXT ENDS
END
```

RUTINES

Rutinas de uso de colas

Rutinas que manipulan colas doblemente encadenadas y ordenadas por pesos.

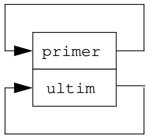
Una cola consta de:
- un descriptor de cola:

```
struct cua {
        struct item *primer;
        struct item *ultim;
}
```

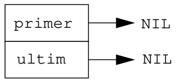
- elementos:

```
struct item {
        struct item *seguent;
        struct item *anterior;
        unsigned long pes;
        }
```

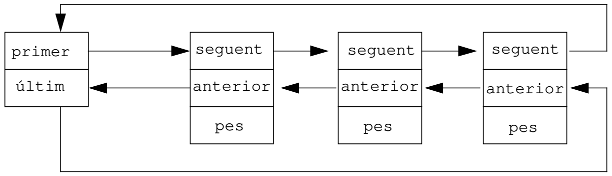
Una cola antes de ser utilizada se ha de inicializar, en caso contrario será ignorada. La representación gráfica de una cola es la siguiente:



cua inicializada



cua sin inicializar



cua con elementos (items)

Para poder tener colas de un tipo de objetos determinados, se ha de incluir la estructura de tipo item dentro del objecto. Ejemplo:

```
struct pcb {
        struct item element_cua;
        int identificador;
        int prioritat;
        int quantum;
        int status;
        long cpu_time;
        .
        .
        };
```

void inicialitzar_cua (cua)
struct cua *cua;

Descripción: **Inicializa una estructura de tipo cola, haciendo que el puntero a primero y a último de la cola apunten a la misma estructura.**

Código:

```
void inicialitzar_cua(c)
struct cua *c;
        {
        c ->ultim = c->primer = (struct item *)c;
        }
```

void encuar(cua, item)
struct cua *cua;
struct item *item;

Descripción: **Procedimiento que inserta al final de la cola el elemento especificado (item).**

Código:

```
#define NIL (long)0

void encuar(cua, item)
struct cua *cua;
struct item *item;
        {
        struct item *j;

        if((j=c->ultim) != NIL)
                {
                i -> seguent = j -> seguent;
                i -> anterior = j;
                (j -> seguent) -> anterior = i;
                j -> seguent = i;
                }
        }
```

void insertar(cua,item,pes)
struct cua *cua;
struct item *item;
unsigned long pes;

Descripción: **Procedimiento que inserta un elemento en una cola ordenada por pesos.**

Código:

```
#define NIL (long)0

void insertar(c,i,p)
struct cua *c;
struct item *i;
unsigned long p;
              {
              struct item *j,*k;

              if((j=c -> ultim)==NIL) return;
              i -> pes=p;
              k = (struct item *)c;
              while ((j!=(struct item *)c) && (j->pes < p))
                            {
                            k=j;
                            j=j->anterior;
                            }
              i->seguent=k;
              i->anterior=j;
              k->anterior=i;
              j->seguent=i;
              }
```

struct item *(primer (cua))
struct cua *cua;

Descripción: **Función que devuelve un puntero al primer elemento de la cola y lo saca de la cola. Si la cola esta vacía o no está inicializada devuelve NIL**

Código:

```
#define NIL (long)0

struct item *(primer (c))
struct cua *c;
              {
              struct item *j;

              if ((j=c->primer) == NIL) return(NIL);
              c->primer=j->seguent;
              (j->seguent)->anterior=(struct item *)c;
              return(j== (struct item *)c ? NIL : j);
              }
```


struct item *(extreure(cua, pes))
struct cua *cua;
unsigned long pes;

Descripción: **Función que extrae de una cola ordenada, un elemento de un peso determinado. Si la cola no está inicializada o el elemento no existe, devuelve NIL.**

Código:

```
#define NIL (long)0

struct item *(extreure(c, p))
struct cua *c;
unsigned long p;
  {  struct item *j;
     struct item *k;

     if((j=c->primer)== NIL)return(NIL);
     k=(struct item *)c;
     while ((j != (struct item *) c) && (j-> pes >p))
                            { k = j;
                              j=j->seguent;
                            }
     if (( j== (struct item *)c) || (j ->pes != p) return(NIL);
     k -> seguent = j -> seguent;
     (j-> seguent) -> anterior=k;
     return(j);
  }
```

struct item *(extirpar(cua,item))
struct cua *cua;
struct item *item;

Descripción: **Función que extrae de una cola un elemento. Si el elemento no está o la cola no está inicializada devuelve un NIL.**

Código:

```
#define NIL (long)0

struct item *(extirpar(c, e))
struct cua *c;
struct item *e;
            {  struct item *j, *k;

               if((j=c->primer)==NIL)return(NIL);
               k=(struct item *)c;
               while ((j!=(struct item *)c) && (j!=e))
                            { k=j;
                              j=j->seguent;
                            }
               if(j==(struct item *)c) return (NIL);
               k -> seguent=j-> seguent;
               (j-> seguent) -> anterior = k;
               return(j);
            }
```

struct item * (cap(cua))
struct cua *cua;

Descripción: **Función que devuelve la dirección del primer elemento de la cola, pero no lo extrae.**

Código:

```
struct item * (cap(c))
struct cua *c;
  {
  return ( c->primer);
  }
```

int buida (cua)
struct cua *cua;

Descripción: **Función que mira si la cola está vacía, y si lo está devuelve cierto. Si la cola no está inicializada también devuelve cierto.**

Código:

```
#define NIL (long)0

int buida (c)
struct cua *c;
  {
  return (( c->primer==NIL )||( c->primer == (struct item *) c));
  }
```

struct item *(buscar_item(cua,pes))
struct cua *cua;
unsigned long pes;

Descripción: **Busca el primer elemento de la cola con un peso igual al del parámetro pes.**

Código:

```
#define NIL (long)0

struct item *(buscar_item(c, p))
struct cua *c;
unsigned long p;
    {
    struct item *j;

    if((j=c->primer)== NIL)return(NIL);
    while ((j != (struct item *) c) && (j-> pes >p))
                        j=j->seguent;
    if (( j== (struct item *)c) || (j ->pes != p) return(NIL);
    return(j);
    }
```


struct item *(seguent(cua,item))
struct cua *cua;
struct item *item;

Descripción: **Obtiene el siguiente elemento de la cola.**

Código:

```
#define NIL (long)0

struct item *(seguent(c, i))
struct cua *c;
struct item *i;
            {
            if(i->seguent == (struct item *)c) return
((struct item *)NIL);
            return(i->seguent);
            }
```

struct item *(anterior(cua,item))
struct cua *cua;
struct item *item;

Descripción: **Obtiene el elemento anterior.**

Código:

```
#define NIL (long)0


struct item *(anterior(c, i))
struct cua *c;
struct item *i;
            {
            if(i->anterior == (struct item *)c) return
((struct item *)NIL);
            return(i->anterior);
            }
```