MEMORIA

1. INTRODUCCIÓN

Nuestra opción elegida ha sido "**Realización de un historial de comandos para Minix".**

Decidimos elegir esta opción, porque nos pareció interesante estudiar y principalmente aprender como en un sistema operativo, en este caso minix, a través de algunas modificaciones en distintas funciones se puede crear dicho historial .

La opción elegida nos ha ayudado a conocer más a fondo el sistema operativo MINIX, principalmente los archivos **sunkeyboard.c** que es el archivo desde el que llamamos a funciones definidas en tty.c, éste es el encargado de controlar la terminal y **tty.c** que gestiona las entradas y salidas del **kernel**, ya que son los que están más relacionados con nuestro proyecto.

2. OBJETIVO

Nuestro proyecto consiste en hacer un historial de comandos para Minix.
Es un historial que recuerde los comandos usados anteriormente. Se usará la flecha de arriba si se quiere ver los comandos utilizados anteriormente y si se quiere retroceder entonces se utilizará la flecha hacia abajo.

3. FUNDAMENTOS TEÓRICOS Y DESCRIPCIÓN DEL TRABAJO REALIZADO

3.1. Variables Globales usadas en "sunkeyboard.c"

A continuación explicaremos las variables globales que hemos añadido a nuestra práctica. Las hemos declarado en este mismo fichero como globales ya que son usadas en varias funciones y no se pueden pasar como argumento.

Definimos dos constantes que son el tamaño del buffer que vamos a usar para guardar todos los caracteres que se van escribiendo por pantalla:

```
#define palabras 50        /*Número de comandos a guardar*/
#define longpala 100       /*Longitud del comando*/
```

Las variables utilizadas en nuestra práctica son:

```
int letra1,letra2;         /*controlan si es flecha para arriba o para abajo*/
int pulsacion=-1;          /*Contador que controla el comando que hay que
                             mostrar*/
int contador=0;            /*Comandos que hay en el buffer*/
```

```
int linea=0,letras=0;          /*Número de líneas y caracteres en el buffer*/
int tipo=0;                    /*Variable que indica si es flecha hacia arriba o hacia
                                 abajo*/
int bandera=0;                 /*Indica si es la última palabra en el buffer*/
int blanco=0;                  /*Indica si es la primera palabra en el buffer*/
int primera=0;                 /*Indica si es la primera palabra que se escribe, la
                                 cual eliminamos porque es el nombre de usuario*/
char buffer[palabras][longpala];       /*Matriz donde guardamos los
                                         comandos*/
```

3.2. Funciones Modificadas en "sunkeyboard.c"

En este fichero hemos modificado las siguientes funciones:

• *kb_read(tty_t *tp):* Kb_read coge el código de rastreo del buffer circular del teclado y coloca el correspondiente código ASCII en su buffer local, el cual es lo suficientemente grande para almacenar secuencias de escape que deben ser generadas en respuesta a algunos códigos de rastreo provenientes del teclado numérico. Entonces se llama a *in_process* para colocar el carácter en la cola de salida. En esta función (*kb_read*), *lock* y *unlock* son usados para proteger el decremento de la variable *kb->icount* de la aparición de una posible interrupción de teclado al mismo tiempo. La llamada a *make_break* devuelve el código ASCII como un entero. Teclas especiales como las del teclado numérico y teclas de función, tienen valores superiores a 0xFF en este punto. Códigos dentro del rango comprendido entre *HOME* e *INSRT* (0x101 y 0x10C respectivamente, definidos en *include/minix/keymap.h*) resultan de apretar una tecla del teclado numérico, y son convertidos a una secuencia de tres caracteres de escape. En este momento, la secuencia es pasada a *in_process*. Códigos mayores que el de *INSRT*, no son pasados a *in_process*, pero se realiza un chequeo para comprobar si los códigos corresponden a *ALT-LEFT-ARROW*, *ALT-RIGHT-ARROW*, o desde *ALT-F1* hasta *ALT-F12*, en cuyo caso se llama a *select_console* para activar **consolas virtuales**.

Las líneas que están en negrita es lo modificado en esta función. En la primera lo que cambiamos es que antes devolvía un valor (void) y ahora devuelve un (char), que se lo pasamos a la función guardar que es la siguiente línea que ha sido añadida y guarda los caracteres en nuestro buffer.

```
static void kb_read(tty_t *tp){

struct kb_s *kb;

    int scode;

    char ch;

    kb = kb_addr(tp);

    lock();
```

```
   while (kb->icount > 0) {
      scode = *kb->itail++;
      if (kb->itail == kb->ibuf + KB_IN_BYTES) kb->itail = kb->ibuf;
      kb->icount--;
      if (func_key(tp, scode)) continue;
      ch = scode;
      unlock();
      letra=in_process(tp, &ch, 1);
      guardar(letra);
 lock();
   }
   unlock();
}
```

•Func_key(tty_t *tp, int scode):

Todo este código es nuevo, y como se ve está en 4 cases distintos, esto es así, porque las teclas flecha arriba y flecha abajo son entendidas como 3 caracteres "1b", "5b" y "41" o "42", entonces vamos comprobando que se cumplen los tres cases y si es así ya hacemos lo correspondiente según sea flecha arriba o abajo.

```
static int func_key(tty_t *tp, int scode)
{
   switch(scode) {
   case 0x1f
      if (tty_line(tp) != 0) return FALSE;
      panic("TERMINAL PANIC", NO_NUM);
      break;
   case MLOGIN_HANGUP:
      hangup(tty_line(tp));
```

```
    break;
case 0x1b:
    letra1=1;
    break;
case 0x41:                          /*Flecha hacia arriba*/
    if ((letra1==1)&&(letra2==1)){
       tipo=0;
       if (pulsacion<contador-1){ /*Solo mostramos comandos guardados, sino pitido*/
         pulsacion++;
         bandera=0;
       }else
         printf("\a");
       sacardato(tp);
    }
   break;
case 0x42:                          /*Flecha hacia abajo*/
    if ((letra1==1)&&(letra2==1)){
       tipo=1;
       if ((pulsacion<=contador)&&(pulsacion>-1)){
         if(blanco==0)
           pulsacion--;
         else{
           blanco=0;
           bandera=0;
         }
```

```
            if (pulsacion<contador-1)
              bandera=0;
           }else
            printf("\a");
           sacardato(tp);
        }
       break;
    case 0x5b:
       if (letra1==1)
         letra2=1;
       break;
    default:
       return FALSE;
    }
    return TRUE;
}
```

3.3 Funciones Nuevas en "sunkeyboard.c"

•guardar(char letra):

Esta función es llamada en ***kb_read*** y se le pasa como argumento (letra) que es el carácter leído del teclado. Aquí lo que hacemos es ir guardando los caracteres en un buffer, hasta que se encuentra un "intro" lo que indica el fin de línea. Así por cada fila tendremos un comando distinto y luego irán saliendo según corresponda.

El código de la función es:

```
void guardar(char letra){

if (primera!=0){                    /*Quitamos la primera palabra, el usuario*/

if (letra=='\n'){

  buffer[linea][letras]='\0';       /*Ponemos fin de palabra*/

  int numero=strlen(buffer[linea]);

  if (numero>0){

    contador++;

    linea++;

  }

  pulsacion=-1;

  letras=0;

}else{

  if(pulsacion==-1)                 /*Si todavía  no se han pulsado flechas*/

    buffer[linea][letras]=letra;

  else{

    letras=strlen(buffer[contador-pulsacion-1]);

    buffer[contador-pulsacion-1][letras]=letra;

  }
```

```
  letras++;
}
}else{
if (letra=='\n')
  primera=1;
}
}
```

•sacardato(tty_t *tp):

Esta función es llamada desde ***func_key*** cuando se haya pulsado una de las dos flechas, lo que hace es controlar que comando hay que sacar según la flecha que haya sido pulsada, luego se llama a la función ***in_process*** para que lo muestre por pantalla, antes de esto borramos lo que hay en la pantalla, llamando a la función ***borrarletras.***

El código de la función es:

```
/*Función que controla el comando a mostrar, mirando si es el primero o el ultimo, que son especiales*/
void sacardato(tty_t *tp){
char *comando;
int longitud;
if ((pulsacion<contador)&&(bandera==0)){   /*Hay datos guardados y no es el ultimo*/
 if(pulsacion==contador-1)
   bandera=1;
 if (pulsacion==0){    /*Si es la primera pulsación hay que borrar las letras escritas en la
                              pantalla*/
   if (tipo==0)
     longitud=letras;
   else{            /*Si estamos bajando en el buffer, hay que borrar el anterior comando*/
     comando=buffer[contador-2];
```

```
      longitud=strlen(comando);
    }
  }else{
    if(tipo==0)
      comando=buffer[contador-pulsacion]; /*borramos anterior palabra según si subimos o bajamos*/
    else
      comando=buffer[contador-pulsacion-2];
    longitud=strlen(comando);
  }
  borrarletras(tp,longitud);    /*llamamos a la función borrarletras para eliminarlas de la pantalla*/
  if (pulsacion>-1){          /*mostramos el comando*/
    comando=buffer[contador-pulsacion-1];
    longitud=strlen(comando);
    (void) in_process(tp,comando,longitud);
  }
}else{
  if(pulsacion==contador-1){
  /*si es el ultimo comando borramos ese comando y ponemos la bandera blanco=1 esta lo que hace es controlar que ya hemos escrito todos los comandos del buffer y cuando pulsemos hacia atrás muestre el ultimo de nuevo*/
    blanco=1;
    comando=buffer[contador-pulsacion-1];
    longitud=strlen(comando);
    borrarletras(tp,longitud);
```

•borrarletras(tty_t *tp, int pos):

Esta función es la encargada de borrar lo que está escrito en la pantalla, se la llama desde la función ***sacardatos*** y se le manda a parte del terminal donde se esta ejecutando minix, la longitud de la palabra a borrar. Y lo que hacemos es crear una palabra con el "código de suprimir" con la longitud mandada en "pos", así se borrarán los caracteres que había.

El código de la función es:

```
/*Funcion que borra las letras de la pantalla*/

void borrarletras(tty_t *tp,int pos){

char *palabra,uno='\b';

palabra=&uno;

int longitud=0;

if (pos>0){

 longitud=pos;

 while(pos>0){

   palabra[pos-1]=uno;

   pos--;

 }   /*Hacemos que procese la palabra*/

 (void) in_process(tp,palabra,longitud);

}

}
```

3.4 Funciones Modificadas en "tty.c"

Para la práctica hemos tenido que modificar la función ***in_process***:

char in_process(*tp,*buf,count):

Lo único que modificamos aquí es que antes esta función estaba declarada como "int" y al no usarse ese valor, hemos hecho que devuelva un "char" y así guardarlo en el buffer.

```
        *tp->tty_inhead++ = ch;

        if (tp->tty_inhead == bufend(tp->tty_inbuf))

                tp->tty_inhead = tp->tty_inbuf;

        tp->tty_incount++;

        if (ch & IN_EOT) tp->tty_eotct++;

        if (tp->tty_incount == buflen(tp->tty_inbuf)) in_transfer(tp);

  }

 /* return ct;*/

  return ch;

}
```

CONCLUSIONES

Hemos tomado como referencia el ejemplo que está colgado en la página sobre este mismo proyecto. Al principio hemos encontrado bastantes dificultades, porque estábamos demasiado pérdidas, pero con la información que hemos buscado y también con ayuda de algunos compañeros hemos conseguido realizar nuestro proyecto.

BIBLIOGRAFÍA

http://sopa.dis.ulpgc.es/ii-dso/lecminix/manejado/keyboard/keyboard.pdf

http://www.infor.uva.es/~cevp/ASO/fichs/Proy_Ejemplo/Ej_Trabajo_Nivel_1_ImpHist.pdf