Arquitecturas de Computadoras

TP Especial

Mini Sistema operativo

Integrantes

MAGNORSKY, Alejandro
MATA SUÁREZ, Andrés
MERCHANTE, Mariano

Cátedra

VALLÉS, Santiago Raúl
EL NOMBRE DEL NUEVO NO ME LO SE NI LO DICE IOL

Fecha de entrega

Viernes 18 de junio de 2010

Índice general

Inconvenientes surgidos - FALTA BOCHA**********

Dada la imposibilidad de uso de las librerías de C oficiales, en determinadas situaciones nos vimos obligados a ajustar el diseño de nuestro sistema de acuerdo con la implementación de nuestra propia libreria de C. Tales percances y sus resoluciones son expuestos a continuación:

Alocación en memoria

Quizás este fue el problema con el que más frecuentemente tuvimos que lidiar. En otras circunstancias, el uso de la función de reserva de espacio en memoria `malloc` de C nos hubiera permitido no muy dificilmente solucionar la cuestión. Sin embargo, al no tener la bases suficientes como para implementar una función similar a la mencionada en nuestra librería, en los casos que así lo dispusieron optamos por reservar cierta cantidad fija en memoria y sacrificar la manipulación dinámica de ese espacio, mediante la declaración de vectores.

Máxima cantidad de caracteres ingresados por el usuario

La función `scanf` de `lib.c`, en un principio, solicita al usuario ingrese una string, invocando a la función `getString` de la misma librería. Esta última función recibe un parámetro de tipo `char *` donde se guardará cada caracter leído con `getchar`, dejando un último lugar para el caracter nulo (`'\0'`). No devuelve nada; el parámetro recibido es también de retorno.

Ahora bien, es tarea de la función invocadora (en este caso, `scanf`) pasarle a `getString` un `char *` con suficiente espacio reservado para que la función se encargue de rellenarlo. Si el parámetro no tiene suficiente espacio para guardar todo lo introducido por el usuario, se producirá un error de segmentación en memoría y el sistema se reiniciará.

Y es aquí donde se toma la decisión. Con las constantes globales, `MAX_ARGUMENTS` y `MAX_ARGUMENT_LENGTH`, se limita la cantidad de argumentos y caracteres por argumento que puede introducir el usuario, respectivamente:

en shell.h:

```
. . .
#define MAX_ARGUMENT_LENGTH 32
#define MAX_ARGUMENTS 8
. . .
```

Dicho esto, dentro de scanf y antes de la llamada a getString, se declara la variable que posteriormente será utilizada como el parámetro de retorno que esta última función requiere de la siguiente manera:

en libc.c:

```
. . .
#define MAX_USER_LENGTH MAX_ARGUMENTS*MAX_ARGUMENT_LENGTH
. . .
int scanf(const char * str, ...){
    . . .
    char strIn[MAX_USER_LENGTH+1];
    . . .
}
```

Así, se reserva el suficiente espacio para que el usuario pueda meter MAX_USER_LENGTH caracteres, más uno para hacer la string null-terminated.

Bibliografía consultada

- **OSDev.org**
  http://wiki.osdev.org/Main_Page

- **Programación del PIC**
  http://www.beyondlogic.org/interrupts/interupt.htm#6
  http://www.osdever.net/tutorials/pdf/pic.pdf

- **ASCIIs y ScanCodes**
  http://www.win.tue.nl/ aeb/linux/kbd/scancodes-1.html
  http://www.cdrummond.qc.ca/cegep/informat/Professeurs/Alain/files/ascii.htm
  http://www.ee.bgu.ac.il/ microlab/MicroLab/Labs/ScanCodes.htm

- **Manejo de excepciones**
  http://pdos.csail.mit.edu/6.828/2008/readings/i386/s09_08.htm

- **Command bytes del teclado**
  Command bytes
  http://www.arl.wustl.edu/ lockwood/class/cs306/books/artofasm/Chapter_20/CH20-2.html
  LEDs http://wiki.osdev.org/PS2_Keyboard#Keyboard_LEDs
  Reboot http://wiki.osdev.org/Reboot

Código fuente

Directorio src

libc.c:

```
#include "../include/kernel.h"
#include "../include/shell.h"
#include "../include/kc.h"

#define MAX_USER_LENGTH MAX_ARGUMENTS*MAX_ARGUMENT_LENGTH

int getchar(){
        return getc(stdin);
}

int getc(int fd){
        int character = -1;
        while (character == -1)
                read(fd, &character, 1);
        return character;

}


int atoi(const char *str){
        int i, acum, flagNegative = 0;
        for (i = 0, acum = 0; (!i && str[i]=='-') ||
           isDigit(str[i]) && str[i] != '\0'; i++){
                if (str[i]=='-')
                        flagNegative = 1;
                else
                        acum = acum * 10 + str[i] - '0';
        }
        if (i==strlen(str))
                return flagNegative * -1 * acum + !
                   flagNegative * acum;
        return 0;
}


/*
** NOTE:
**      This implementation of 'getString' is aimed for
```

```
   being used only by 'scanf' function (where
** 'ans' is always a vector of MAX_STRLEN chars), as i
   counter is only compared against MAX_STRLEN
** constant (in the 'else if' clause), regardless the
   length of 'ans'.
*/
void getString(char * ans){
        int i, j;
        char c;
        for (i=0, j=0; (c = getchar()) != '\n' ; j++){
                printf("%c", c);
                if(c == '\b'){
                        if (i)
                                ans[--i] = '\0';
                } else if (i<MAX_USER_LENGTH){
                        ans[i] = c;
                        i++;
                }
        }
        ans[i] = '\0';
}


/*
** NOTE:
**      As we can't make use of 'malloc' function, we
   decided to set MAX_USER_LENGTH constant with the
** max quantity of characters the user can enter. Then, '
   scanf' reserves MAX_USER_LENGTH+1 bytes for
** vector strIn (strIn[MAX_USER_LENGTH] = '\0').
*/

int scanf(const char * str, ...){
        int i, j, k, acum, strLen, strTrueLen, strInLen,
           flagNegative;
        /*      strLen:         'str' length. Scanf's
           string.
        **      strTrueLen:     final 'str' length, once
           every %x in 'str' has been replaced with values.
        **      strInLen:       'strIn' length, user-
           entered string.
        */
        char strIn[MAX_USER_LENGTH+1];  /* Last position is
```

```
            reserved for '\0'; user can effectively enter
           MAX_USER_LENGTH chars */
71          void ** argv = (void **)(&str);
72          getString(strIn); /* User enters a string */
73          strLen = strlen(str);
74          strTrueLen = strLen;    /* At start, boths lengths
              are equals, because no %x has been replaced */
75          strInLen = strlen(strIn);
76
77          for (i=0, j=0 ; i<strLen && j<strInLen; i++, j++){
78                  if ( i+1 != strLen && str[i] == '%' )
79                          switch(str[++i]){
80                                  case 'd':
81                                          for(k=0, acum = 0,
                                             flagNegative =
                                             0; ( strIn[j] ==
                                              '-' && !k ) ||
                                             isDigit(strIn[j
                                             ]) ; k++, j++){
82                                               if (strIn[j
                                                  ] != '-'
                                                  ){
83                                                       acum
                                                          =
                                                          acum
                                                          *
                                                          10
                                                          +
                                                          strIn
                                                          [
                                                          j
                                                          ]
                                                          -
                                                          '
                                                          0
                                                          '
```

```
                                                    ;
84                                          } else
85                                                  flagNegative
                                                    =
                                                    1;
86                                  }
87                                  j--;
88                                  if (!k){
89                                          return -1;
90                                  } else if (!isDigit
                                        (strIn[j]) &&
                                        flagNegative){
91                                          return -1;
92                                  }
93                                  *(*((int **)++argv)
                                        ) = flagNegative
                                         * -1 * acum + !
                                        flagNegative *
                                        acum;
94                                  strTrueLen =
                                        strTrueLen - 2 +
                                         k;  /* Sub 2
                                        ('%' and 'd')
                                        and add k (qty.
                                        of read digits -
                                         1) */
95                                  break;
96                          case 'c':
97                                  *(*((char **)++argv
                                        )) = strIn[j];
98                                  strTrueLen--;
99                                  break;
100                         case 's':
101                                 argv++;
102                                 /* NOTE:
103                                 **      i+1 can be
                                        equal to strLen
                                        as 'str' is a
                                        null-terminated
```

```
104                                    const char *.
                                    ** So, str[strLen]
                                       = '\0';
105                                 */
106                                 if ( i+1 <= strLen)
                                       {
107                                      for (k=0; k<
                                          MAX_USER_LENGTH
                                           && strIn[j
                                          ] != str[i
                                          +1] ; k++,
                                          j++)
108                                                ((
                                                   char

                                                   *)
                                                   (*((
                                                   char

                                                   **)

                                                   argv
                                                   ))
                                                   )[
                                                   k]
                                                    =

                                                   strIn
                                                   [j
                                                   ];
109                                        ((char *)(*((
                                              char **)
                                              argv)))[k] =
                                               '\0';
110                                 }
111                                 j--;
112                                 strTrueLen =
                                       strTrueLen -2 +
                                       k;
113                                 break;
114
115                           /* NOTE:
116                           **      'default' case
```

```
                                        works this way to keep
                                        analogy with 'printf'
                                        function. If it detects
                                        a '%' and next character
117                                  ** is neither 'd' nor 'c'
                                        nor 's', then it skips
                                        both characters ('%' and
                                         the next one).
118                                  */
119                                  default:
120                                          j--;
121                                          strTrueLen-=2;
122                                          break;
123                          }
124                  else
125                        if (str[i] != strIn[j])
126                                  return -1;
127          }
128          if (strInLen != strTrueLen)
129                  return -1;
130          printf("\n");    /* The user entered '\n', so it
                must be used */
131          return 0;
132  }
133
134
135  int read(int fd, int * c, size_t count){
136          return _int_80_hand(0, fd, c, count);
137  }
138
139
140  int strlen(const char * str){
141          int i;
142          for (i = 0; str[i] != '\0'; i++);
143                  return i;
144  }
145
146  int isDigit(int a){
147          if( (a >= '0') && (a <= '9') )
148                  return 1;
149          return 0;
150  }
151
```

```
152 int putchar(int c){
153         return putc(stdout, c);
154 }
155
156 int putc(int fd, int c){
157         return write(fd, &c, 1);
158 }
159
160 int printf(const char * str, ...){
161         int c;
162         int wait = 0;
163         void ** argv = (void **)(&str);
164
165         for(c=0;str[c] != NULL;c++){
166                 if(wait){
167                         switch(str[c]){
168                         case 'd':
169                                 putInt( *( (int*) argv));
170                                 break;
171                         case 'c':
172                                 putchar(*((char*)argv));
173                                 break;
174                         case 's':
175                                 printf(*((char**)argv));
176                                 break;
177                         default:
178                                 break;
179                         }
180                         wait = 0;
181                 } else {
182                         if(str[c] == '%'){
183                                 wait = 1;
184                                 argv++;
185                         }
186                         else
187                                 putchar(str[c]);
188                 }
189         }
190 }
191
192 int write(int fd, int * c, size_t count){
193         return _int_80_hand(1, fd, c, count);
194 }
```

```
195
196 int pow(int n, int exp){
197         int i;
198         int out = 1;
199         for(i=0;i<exp;i++)
200                 out*=n;
201         return out;
202 }
203
204 void putInt(int n){
205
206         int i,j;
207
208         if(n<0){
209                 putchar('-');
210                 n = -n;
211         }
212
213
214         for(i=0;n/pow(10,i) > 0 ;i++); /* char quantity */
215         for(j=i-1;j>0;j--){
216                 putchar( (n/pow(10,j))+'0');
217                 n = n%pow(10,j);
218         }
219         putchar((n%10)+'0');
220 }
221
222 int strcmp(const char * str1, const char * str2){
223
224         int i = 0;
225         int out = 0;
226         while(str1[i] != NULL || str2[i] != NULL){
227
228                 if( str1[i] != str2[i])
229                         return str1[i] - str2[i];
230
231                 i++;
232         }
233
234         if(str1[i] == NULL && str2[i] != NULL)
235                 return -str2[i];
236         else if( str1[i] != NULL && str2[i] == NULL)
237                 return str1[i];
```

```
238
239         return out;
240 }
241
242 int rand(){
243         _outport(0x70, 0);
244         int seconds = _inport(0x71);
245         int acum1, acum2;
246
247         /* NOTE 1:
248         **      - Max value an integer can handle is 2^32.
                So 'seconds%32' makes sure pow's return value
                fits an integer
249         **      - '(int)(seconds/32)' is added to avoid
                cases like:
250         **                      pow(2,59) = pow(2,27)
                        --> acum1 would be the same for seconds=
                n and seconds=n-32
251         **              Thanks to the addition:
252         **                      pow(2,59) + 1 != pow(2,27)
                + 0  --> acum1 can be assigned 60 different
                values
253         */
254         acum1 = pow(2, seconds%32) + (int)(seconds/32);
255
256         /* NOTE 2:
257         **      - Max value an integer can handle is 2^32.
                But 3^20 < 2^32 < 3^21. So 'seconds%20' makes
                sure pow's return value fits an integer.
258         **      - '(int)(seconds/20)' is added to avoid
                cases similar to the ones in *NOTE 1*
259         */
260         acum2 = pow(3, seconds%20) + (int)(seconds/20);
261
262         return acum2%acum1;
263 }
264
265 void wait(int seconds){
266         int i, lastSec = -1;
267         _outport(0x70, 0);
268         int actSec = _inport(0x71);
269
270         for (i=0; i <= seconds; i++){
```

```
271                     lastSec = actSec;
272                     _outport(0x70, 0);
273                     actSec = _inport(0x71);
274                     if (lastSec == actSec)
275                             i--;
276             }
277     }
278
279     /*
        ************************************************************
280     *setup_IDT_entry
281     * Inicializa un descriptor de la IDT
282     *
283     *Recibe: Puntero a elemento de la IDT
284     *        Selector a cargar en el descriptor de interrupcion
285     *        Puntero a rutina de atencion de interrupcion
286     *        Derechos de acceso del segmento
287     *        Cero
288     ************************************************************
        */
289
290     void setup_IDT_entry (DESCR_INT *item, byte selector, dword
        offset, byte access,
291                           byte cero) {
292       item->selector = selector;
293       item->offset_l = offset & 0xFFFF;
294       item->offset_h = offset >> 16;
295       item->access = access;
296       item->cero = cero;
297     }
```