

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 librería 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 difícilmente 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 carácter leído con `getchar`, dejando un último lugar para el carácter 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 memoria 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:

```
1  . . .
2  #define MAX_ARGUMENT_LENGTH 32
3  #define MAX_ARGUMENTS 8
4  . . .
```

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:

```
1  . . .
2  #define MAX_USER_LENGTH MAX_ARGUMENTS*MAX_ARGUMENT_LENGTH
3  . . .
4  int scanf(const char * str, ...){
5      . . .
6      char strIn[MAX_USER_LENGTH+1];
7      . . .
8  }
```

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:

```
1  #include "../include/kernel.h"
2  #include "../include/shell.h"
3  #include "../include/kc.h"
4
5  #define MAX_USER_LENGTH MAX_ARGUMENTS*MAX_ARGUMENT_LENGTH
6
7  int getchar(){
8      return getc(stdin);
9  }
10
11 int getc(int fd){
12     int character = -1;
13     while (character == -1)
14         read(fd, &character, 1);
15     return character;
16 }
17
18
19
20 int atoi(const char *str){
21     int i, acum, flagNegative = 0;
22     for (i = 0, acum = 0; (!i && str[i]!='-') ||
23         isDigit(str[i]) && str[i] != '\0'; i++){
24         if (str[i]!='-')
25             flagNegative = 1;
26         else
27             acum = acum * 10 + str[i] - '0';
28     }
29     if (i==strlen(str))
30         return flagNegative * -1 * acum + !
31             flagNegative * acum;
32     return 0;
33 }
34
35 /*
36 ** NOTE:
37 **      This implementation of 'getString' is aimed for
```

```

    being used only by 'scanf' function (where
37  ** 'ans' is always a vector of MAX_STRLen chars), as i
    counter is only compared against MAX_STRLen
38  ** constant (in the 'else if' clause), regardless the
    length of 'ans'.
39  */
40  void getString(char * ans){
41      int i, j;
42      char c;
43      for (i=0, j=0; (c = getchar()) != '\n' ; j++){
44          printf("%c", c);
45          if(c == '\b'){
46              if (i)
47                  ans[--i] = '\0';
48              } else if (i<MAX_USER_LENGTH){
49                  ans[i] = c;
50                  i++;
51              }
52          }
53          ans[i] = '\0';
54      }
55
56
57  /*
58  ** NOTE:
59  **      As we can't make use of 'malloc' function, we
    decided to set MAX_USER_LENGTH constant with the
60  ** max quantity of characters the user can enter. Then, '
    scanf' reserves MAX_USER_LENGTH+1 bytes for
61  ** vector strIn (strIn[MAX_USER_LENGTH] = '\0').
62  */
63
64  int scanf(const char * str, ...){
65      int i, j, k, acum, strLen, strTrueLen, strInLen,
        flagNegative;
66      /*      strLen:          'str' length. Scanf's
        string.
67      **      strTrueLen:      final 'str' length, once
        every %x in 'str' has been replaced with values.
68      **      strInLen:        'strIn' length, user-
        entered string.
69      */
70      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                                     ;
85                                     } else
86                                     flagNegative
87                                     =
88                                     1;
89
90                                     }
91                                     j--;
92                                     if (!k){
93                                     return -1;
94                                     } else if (!isDigit
95                                     (strIn[j]) &&
96                                     flagNegative){
97                                     return -1;
98                                     }
99                                     *( * ((int **)) ++ argv
100                                     ) = flagNegative
101                                     * -1 * acum + !
102                                     flagNegative *
103                                     acum;
104                                     strTrueLen =
105                                     strTrueLen - 2 +
106                                     k; /* Sub 2
107                                     ('%' and 'd')
108                                     and add k (qty.
109                                     of read digits -
110                                     1) */
111                                     break;
112 case 'c':
113     *( * ((char **)) ++ argv
114     ) = strIn[j];
115     strTrueLen--;
116     break;
117 case 's':
118     argv++;
119     /* NOTE:
120     **      i+1 can be
121     equal to strLen
122     as 'str' is a
123     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

```

```

117                                     works this way to keep
                                         analogy with 'printf'
                                         function. If it detects
                                         a '%' and next character
118                                     ** is neither 'd' nor 'c'
                                         nor 's', then it skips
                                         both characters ('%' and
                                         the next one).
119                                     */
120                                     default:
121                                         j--;
122                                         strTrueLen-=2;
123                                         break;
124                                     }
125     else
126         if (str[i] != strIn[j])
127             return -1;
128     }
129     if (strInLen != strTrueLen)
130         return -1;
131     printf("\n"); /* The user entered '\n', so it
132                    must be used */
133     return 0;
134 }
135
136 int read(int fd, int * c, size_t count){
137     return _int_80_hand(0, fd, c, count);
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.
249           So 'seconds%32' makes sure pow's return value
250           fits an integer
251         **          - '(int)(seconds/32)' is added to avoid
252           cases like:
253
254           pow(2,59) = pow(2,27)
255           --> acum1 would be the same for seconds=
256           n and seconds=n-32
257         **          Thanks to the addition:
258         **          pow(2,59) + 1 != pow(2,27)
259           + 0 --> acum1 can be assigned 60 different
260           values
261         */
262         acum1 = pow(2, seconds%32) + (int)(seconds/32);
263
264         /* NOTE 2:
265         **          - Max value an integer can handle is 2^32.
266           But 3^20 < 2^32 < 3^21. So 'seconds%20' makes
267           sure pow's return value fits an integer.
268         **          - '(int)(seconds/20)' is added to avoid
269           cases similar to the ones in *NOTE 1*
270         */
271         acum2 = pow(3, seconds%20) + (int)(seconds/20);
272
273         return acum2%acum1;
274     }
275
276     void wait(int seconds){
277         int i, lastSec = -1;
278         _outport(0x70, 0);
279         int actSec = _inport(0x71);
280
281         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     *****
281
282 *setup_IDT_entry
283 * Inicializa un descriptor de la IDT
284 *
285 *Recibe: Puntero a elemento de la IDT
286 *         Selector a cargar en el descriptor de interrupcion
287 *         Puntero a rutina de atencion de interrupcion
288 *         Derechos de acceso del segmento
289 *         Cero
290 *****
291 */
292
293 void setup_IDT_entry (DESCR_INT *item, byte selector, dword
294     offset, byte access,
295     byte cero) {
296     item->selector = selector;
297     item->offset_l = offset & 0xFFFF;
298     item->offset_h = offset >> 16;
299     item->access = access;
300     item->cero = cero;
301 }

```