FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 0 — Infraestructura básica

[66.20] Organización de Computadoras
Segundo cuatrimestre de 2018

Alumno:	ALVAREZ WINDEY, Ariel Justo
Número de padrón:	97893
Email:	arieljaw12@gmail.com

Alumno:	CÁCERES, Julieta Agustina
Número de padrón:	96454
Email:	julieta.agustina.caceres@gmail.com

Alumno:	ROBLES, Gabriel
Número de padrón:	95897
Email:	gabyrobles93@gmail.com

Índice

1. Objetivo

El objetivo de este trabajo práctico fué desarrollar un programa en C que dado un imput lo codifique en base 64, a su vez este programa tuvo que tener la funcionalidad de decodificar desde base 64 a su formato original. Con la realización de este programa se esperaba familiarizarse con las herramientas de software que usaremos en la materia.

2. Diseño e implementación

Para la implementación de dicho programa se indentificó dos funcionalidades diferentes para las cuales se crearon los siguientes TDA:

- TDA Encoder:

  El cual posee la definición del struct: B64_encoder_t y las siguientes firmas:

  - int B64_encoder_create(B64_encoder_t * enc, FILE * input, FILE *output); Dicha función recibe como parámetro el puntero a struct y los punteros FILE desde donde leerá el input o hacia donde dirigirá el resultado de la codificación (en caso de ser la salida y/o entrada estándar se puede inicializar con NULL respectivamente), es la encargada de inicializar el encoder.
  - int B64_encoder_start(B64_encoder_t *enc); Esta función recibe como parámetro al encoder y comienza la codificación.
  - void _B64_encoder_write(B64_encoder_t * enc, const void * buff); Esta función privada es utilizada por el TDA en el proceso de codificación. Recibe como parámetro el encoder y el buffer donde se escribirá el resultado

- TDA Decoder: El cual posee la definición del struct: B64Decoder Y las siguientes fimas:

  - int decoder_create(B64Decoder * decoder, FILE * finput, FILE * foutput); Dicha función recibe como parámetro el puntero al struct decoder y los punteros FILE desde donde leerá el input o hacia donde dirigirá el resultado de la decodificación (al igual que el encoder, en caso de recibir los punteros FILE en NULL, inicializa con salida/entrada estándar). Es la encargada de inicializar el Decoder.
  - int decoder_start(B64Decoder * decoder); Esta función recibe como parámetr al decoder y comienza la decodificación.
  - int decoder_destroy(B64Decoder * decoder); Esta función recibe como parámetro el decoder y de finalizar el decoder.

3. Compilación

Para poder hacer uso del Makefile, primero es necesario crear la carpeta ‘gxemul‘ en la raiz del repositorio. La misma esta agregada a ‘.gitignore‘ ya que ahí estará el emulador y no queremos agregarlo al repositorio dado el tamaño del mismo.

Una vez creada la carpeta, descomprimimos el gxemul con su imagen ahí dentro.

Para bootear el emulador ejecutamos:

‘make gxemul‘

Nos pedirá usuario y contraseña:

‘user: root‘

‘password: orga6620‘

Configuramos el loopback desde el hostOS, es necesario hacerlo cada vez que reiniciamos nuestra computadora: ‘make loopback‘

Creamos el túnel desde el guestOS a nuestro hostOS, ejecutando en la terminal de NetBSD:

ssh -R 2222:127.0.0.1:22 <USER_NAME_HOST>@172.20.0.1

Nos pedirá la contraseña de nuestro usuario host. Una vez finalizado esta consola deberá quedar abierta, es nuestra conexión entre host y guest.

Con el comando ‘make ssh‘ ejecutado de hostOS creamos consolas remotas de guestOS.

Creamos la carpeta ‘tp0‘ en guestOS con el comando ‘mkdir tp0‘. La misma debe ubicarse en ‘/root/tp0‘.

Tenemos los siguientes comandos útiles que podemos ejecutar desde el hostOS:

- ‘make c2guest‘: envía los archivos ‘*.c‘ a la carpeta ‘tp0‘ de guestOS
- ‘make h2guest‘: envía los archivos ‘*.h‘ a la carpeta ‘tp0‘ de guestOS
- ‘make make2guest‘: envía el archivo ‘Makefile‘ a la carpeta ‘tp0‘ de guestOS

Tenemos los siguientes comandos útiles que podemos ejecutar desde el guestOS:

- ‘make asm‘: genera el código assembly del programa.
- ‘make bin‘: compila los archivos y genera un binario ‘tp‘.
- Para poder correr los casos de prueba tal cual el enunciado es necesario mover el binario a la carpeta correspondiente con el comando ‘cp tp0 /usr/bin‘.

4. Corridas de prueba

A continuación se pueden ver las corridas de prueba propuestas en el enunciado, ejecutadas en el emulador de NetBSD propuesto por la cátedra.

En la Figura 1 observamos la salida del comando help

```
# tp0 -h
Usage:
 tp0 -h
 tp0 -V
 tp0 [options]
Options:
 -V, --version          Print version and quit.
 -h, --help             Print this information.
 -i, --input            Location of the input file.
 -o, --output           Location of the output file.
 -a, --action           Program action: encode (default) or decode.
Examples:
 tp0 -a encode -i ~/input -o ~/output
 tp0 -a decode
#
```

Figura 1: Comando help

En la Figura 2 observamos la salida del comando ls -l para un archivo vacío codificado.

```
# touch /tmp/zero.txt
# tp0 -a encode -i /tmp/zero.txt -o /tmp/zero.txt.b64
Opening input file /tmp/zero.txt
Opening output file /tmp/zero.txt.b64
Encode mode
# ls -l /tmp/zero.txt.b64
-rw-r--r--  1 root  wheel  0 Sep 23 02:58 /tmp/zero.txt.b64
#
```

Figura 2: Codificación de archivo vacío

En la Figura 3 observamos la salida para la codificación del caracter 'M'

```
# echo -n M | tp0
Encode mode
TQ==
#
```

Figura 3: Codificación de M

En la Figura 4 observamos la salida para la codificación de la cadena 'Ma'

```
# echo -n Ma | tp0
Encode mode
TWE=
#
```

Figura 4: Codificación de Ma

En la Figura 5 observamos la salida para la codificación de la cadena 'Man'

```
# echo -n Man | tp0
Encode mode
TWFu
#
```

Figura 5: Codificación de Man

En la Figura 6 observamos la salida para la codificación y decodificación de la cadena 'Man'

```
# echo Man | tp0 | tp0 -a decode
Encode mode
Decode mode
Man
#
```

Figura 6: Codificación y decodificación de Man

En la Figura 7 observamos la salida para la codificación bit a bit

```
# echo xyz | tp0 | tp0 -a decode | od -t c
Encode mode
Decode mode
0000000    x   y   z  \n
0000004
```

Figura 7: Codificación bit a bit

En la Figura 8 observamos que la codificación no genera líneas de más de 76 caracteres

```
# yes | head -c 1024 | tp0 -a encode
Encode mode
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5
CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5
CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5
CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5
CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5
CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5
CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5
CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5
CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5
CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5Cg==#
```

Figura 8: Líneas de 76 caracteres

En la Figura 9 verificamos que se hayan decodificado 1024 bytes

```
# yes | head -c 1024 | tp0 -a encode | tp0 -a decode | wc -c
Encode mode
Decode mode
    1024
#
```

Figura 9: Decodificación de 1024 bytes

En la Figura 10 observamos la última corrida de prueba, con archivos de tamaño creciente:

```
# ./run_maximum_test.sh
ok: 1
ok: 2
ok: 3
ok: 4
ok: 5
ok: 6
ok: 7
ok: 8
ok: 9
ok: 10
ok: 11
```

Figura 10: Archivos de tamaño creciente

Donde podemos ver a continuación el código del archivo *run_maxiumum_test*

```
# cat run_maximum_test.sh
N=1;
while :; do
head -c $N </dev/urandom >/tmp/in.bin;
./tp0 -a encode -i /tmp/in.bin -o /tmp/out.b64;
./tp0 -a decode -i /tmp/out.b64 -o /tmp/out.bin;
if diff /tmp/in.bin /tmp/out.bin; then :; else
echo ERROR: $N;
break;
fi
echo ok: $N;
N=$((N+1))
rm -f /tmp/in.bin /tmp/out.b64 /tmp/out.bin
done
#
```

Figura 11: Script *run_maxiumum_test.sh*

5. Código MIPS 32

```
1         .file   1 "main.c"
2          .section .mdebug.abi32
3          .previous
4          .abicalls
5          .rdata
6          .align  2
7  $LC0:
8          .ascii  "rb\000"
9          .text
10         .align  2
11         .globl  open_infile
12         .ent    open_infile
13 open_infile:
14         .frame  $fp,40,$ra              # vars= 0, regs= 3/0, args= 16, extra=
       8
15         .mask   0xd0000000,-8
16         .fmask  0x00000000,0
17         .set    noreorder
```

```
18          .cpload $t9
19          .set    reorder
20          subu    $sp,$sp,40
21          .cprestore 16
22          sw      $ra,32($sp)
23          sw      $fp,28($sp)
24          sw      $gp,24($sp)
25          move    $fp,$sp
26          sw      $a0,40($fp)
27          lw      $a0,40($fp)
28          la      $a1,$LC0
29          la      $t9,fopen
30          jal     $ra,$t9
31          move    $sp,$fp
32          lw      $ra,32($sp)
33          lw      $fp,28($sp)
34          addu    $sp,$sp,40
35          j       $ra
36          .end    open_infile
37          .size   open_infile, .-open_infile
38          .rdata
39          .align  2
40  $LC1:
41          .ascii  "wt\000"
42          .text
43          .align  2
44          .globl  open_outfile
45          .ent    open_outfile
46  open_outfile:
47          .frame  $fp,40,$ra              # vars= 0, regs= 3/0, args= 16, extra=
        8
48          .mask   0xd0000000,-8
49          .fmask  0x00000000,0
50          .set    noreorder
51          .cpload $t9
52          .set    reorder
53          subu    $sp,$sp,40
54          .cprestore 16
55          sw      $ra,32($sp)
56          sw      $fp,28($sp)
57          sw      $gp,24($sp)
58          move    $fp,$sp
59          sw      $a0,40($fp)
60          lw      $a0,40($fp)
61          la      $a1,$LC1
62          la      $t9,fopen
63          jal     $ra,$t9
64          move    $sp,$fp
65          lw      $ra,32($sp)
66          lw      $fp,28($sp)
67          addu    $sp,$sp,40
68          j       $ra
69          .end    open_outfile
70          .size   open_outfile, .-open_outfile
71          .rdata
72          .align  2
73  $LC2:
74          .ascii  "version\000"
75          .align  2
76  $LC3:
77          .ascii  "help\000"
78          .align  2
79  $LC4:
80          .ascii  "input\000"
81          .align  2
82  $LC5:
83          .ascii  "output\000"
```

```
84          .align  2
85  $LC6:
86          .ascii  "action\000"
87          .data
88          .align  2
89          .type   long_options.0, @object
90          .size   long_options.0, 96
91  long_options.0:
92          .word   $LC2
93          .word   0
94          .word   0
95          .word   86
96          .word   $LC3
97          .word   0
98          .word   0
99          .word   104
100         .word   $LC4
101         .word   1
102         .word   0
103         .word   105
104         .word   $LC5
105         .word   1
106         .word   0
107         .word   111
108         .word   $LC6
109         .word   2
110         .word   0
111         .word   97
112         .word   0
113         .word   0
114         .word   0
115         .word   0
116         .rdata
117         .align  2
118 $LC7:
119         .ascii  "Vhi:o:a:\000"
120         .align  2
121 $LC8:
122         .ascii  "-\000"
123         .align  2
124 $LC9:
125         .ascii  "Can not open input file '%s'\n\000"
126         .align  2
127 $LC10:
128         .ascii  "Can not open output file '%s'\n\000"
129         .align  2
130 $LC11:
131         .ascii  "decode\000"
132         .align  2
133 $LC12:
134         .ascii  "encode\000"
135         .align  2
136 $LC13:
137         .ascii  "Unknown action '%s'\n\000"
138         .align  2
139 $LC14:
140         .ascii  "Execute tp0 -h for help\n\000"
141         .align  2
142 $LC15:
143         .ascii  "Unknown option '%c'\n\000"
144         .align  2
145 $LC16:
146         .ascii  "Fail creating decoder\n\000"
147         .text
148         .align  2
149         .globl  main
150         .ent    main
```

```
151 main:
152         .frame  $fp,104,$ra             # vars= 56, regs= 3/0, args= 24, extra
        = 8
153         .mask   0xd0000000,-8
154         .fmask  0x00000000,0
155         .set    noreorder
156         .cpload $t9
157         .set    reorder
158         subu    $sp,$sp,104
159         .cprestore 24
160         sw      $ra,96($sp)
161         sw      $fp,92($sp)
162         sw      $gp,88($sp)
163         move    $fp,$sp
164         sw      $a0,104($fp)
165         sw      $a1,108($fp)
166         sw      $zero,32($fp)
167         sw      $zero,36($fp)
168         sw      $zero,40($fp)
169         sw      $zero,48($fp)
170 $L20:
171         addu    $v0,$fp,48
172         sw      $v0,16($sp)
173         lw      $a0,104($fp)
174         lw      $a1,108($fp)
175         la      $a2,$LC7
176         la      $a3,long_options.0
177         la      $t9,getopt_long
178         jal     $ra,$t9
179         sw      $v0,44($fp)
180         lw      $v1,44($fp)
181         li      $v0,-1                  # 0xffffffffffffffff
182         bne     $v1,$v0,$L22
183         b       $L21
184 $L22:
185         lw      $v0,44($fp)
186         addu    $v0,$v0,-86
187         sw      $v0,84($fp)
188         lw      $v1,84($fp)
189         sltu    $v0,$v1,26
190         beq     $v0,$zero,$L36
191         lw      $v0,84($fp)
192         sll     $v1,$v0,2
193         la      $v0,$L37
194         addu    $v0,$v1,$v0
195         lw      $v0,0($v0)
196         .cpadd  $v0
197         j       $v0
198         .rdata
199         .align  2
200 $L37:
201         .gpword $L24
202         .gpword $L36
203         .gpword $L36
204         .gpword $L36
205         .gpword $L36
206         .gpword $L36
207         .gpword $L36
208         .gpword $L36
209         .gpword $L36
210         .gpword $L36
211         .gpword $L36
212         .gpword $L32
213         .gpword $L36
214         .gpword $L36
215         .gpword $L36
216         .gpword $L36
```

```
217         .gpword $L36
218         .gpword $L36
219         .gpword $L25
220         .gpword $L26
221         .gpword $L36
222         .gpword $L36
223         .gpword $L36
224         .gpword $L36
225         .gpword $L36
226         .gpword $L29
227         .text
228 $L24:
229         la      $t9,version
230         jal     $ra,$t9
231         sw      $zero,80($fp)
232         b       $L19
233 $L25:
234         la      $t9,help
235         jal     $ra,$t9
236         sw      $zero,80($fp)
237         b       $L19
238 $L26:
239         lw      $a0,optarg
240         la      $a1,$LC8
241         la      $t9,strcmp
242         jal     $ra,$t9
243         beq     $v0,$zero,$L20
244         lw      $a0,optarg
245         la      $t9,open_infile
246         jal     $ra,$t9
247         sw      $v0,32($fp)
248         lw      $v0,32($fp)
249         bne     $v0,$zero,$L20
250         la      $a0,__sF+176
251         la      $a1,$LC9
252         lw      $a2,optarg
253         la      $t9,fprintf
254         jal     $ra,$t9
255         li      $v0,1                   # 0x1
256         sw      $v0,80($fp)
257         b       $L19
258 $L29:
259         lw      $a0,optarg
260         la      $a1,$LC8
261         la      $t9,strcmp
262         jal     $ra,$t9
263         beq     $v0,$zero,$L20
264         lw      $a0,optarg
265         la      $t9,open_outfile
266         jal     $ra,$t9
267         sw      $v0,36($fp)
268         lw      $v0,36($fp)
269         bne     $v0,$zero,$L20
270         la      $a0,__sF+176
271         la      $a1,$LC10
272         lw      $a2,optarg
273         la      $t9,fprintf
274         jal     $ra,$t9
275         li      $v1,1                   # 0x1
276         sw      $v1,80($fp)
277         b       $L19
278 $L32:
279         lw      $v0,optarg
280         beq     $v0,$zero,$L36
281         lw      $a0,optarg
282         la      $a1,$LC11
283         la      $t9,strcmp
```

```
284         jal     $ra,$t9
285         bne     $v0,$zero,$L34
286         li      $v0,1                   # 0x1
287         sw      $v0,40($fp)
288         b       $L20
289 $L34:
290         lw      $a0,optarg
291         la      $a1,$LC12
292         la      $t9,strcmp
293         jal     $ra,$t9
294         bne     $v0,$zero,$L35
295         sw      $zero,40($fp)
296         b       $L20
297 $L35:
298         la      $a0,__sF+176
299         la      $a1,$LC13
300         lw      $a2,optarg
301         la      $t9,fprintf
302         jal     $ra,$t9
303         la      $a0,__sF+176
304         la      $a1,$LC14
305         la      $t9,fprintf
306         jal     $ra,$t9
307         li      $v0,1                   # 0x1
308         sw      $v0,80($fp)
309         b       $L19
310 $L36:
311         la      $a0,__sF+176
312         la      $a1,$LC15
313         lw      $a2,44($fp)
314         la      $t9,fprintf
315         jal     $ra,$t9
316         la      $a0,__sF+176
317         la      $a1,$LC14
318         la      $t9,fprintf
319         jal     $ra,$t9
320         li      $v1,1                   # 0x1
321         sw      $v1,80($fp)
322         b       $L19
323 $L21:
324         lw      $v0,40($fp)
325         bne     $v0,$zero,$L38
326         addu    $v0,$fp,56
327         move    $a0,$v0
328         lw      $a1,32($fp)
329         lw      $a2,36($fp)
330         la      $t9,B64_encoder_create
331         jal     $ra,$t9
332         addu    $v0,$fp,56
333         move    $a0,$v0
334         la      $t9,B64_encoder_start
335         jal     $ra,$t9
336         lw      $v0,32($fp)
337         beq     $v0,$zero,$L39
338         lw      $a0,32($fp)
339         la      $t9,fclose
340         jal     $ra,$t9
341 $L39:
342         lw      $v0,36($fp)
343         beq     $v0,$zero,$L38
344         lw      $a0,36($fp)
345         la      $t9,fclose
346         jal     $ra,$t9
347 $L38:
348         lw      $v1,40($fp)
349         li      $v0,1                   # 0x1
350         bne     $v1,$v0,$L46
```

```
351         addu    $v0,$fp,72
352         move    $a0,$v0
353         lw      $a1,32($fp)
354         lw      $a2,36($fp)
355         la      $t9,decoder_create
356         jal     $ra,$t9
357         beq     $v0,$zero,$L42
358         la      $a0,__sF+176
359         la      $a1,$LC16
360         la      $t9,fprintf
361         jal     $ra,$t9
362         li      $v0,1                   # 0x1
363         sw      $v0,80($fp)
364         b       $L19
365 $L42:
366         addu    $v0,$fp,72
367         move    $a0,$v0
368         la      $t9,decoder_start
369         jal     $ra,$t9
370         beq     $v0,$zero,$L43
371         li      $v1,1                   # 0x1
372         sw      $v1,80($fp)
373         b       $L19
374 $L43:
375         lw      $v0,32($fp)
376         beq     $v0,$zero,$L44
377         lw      $a0,32($fp)
378         la      $t9,fclose
379         jal     $ra,$t9
380 $L44:
381         lw      $v0,36($fp)
382         beq     $v0,$zero,$L46
383         lw      $a0,36($fp)
384         la      $t9,fclose
385         jal     $ra,$t9
386 $L46:
387         sw      $zero,80($fp)
388 $L19:
389         lw      $v0,80($fp)
390         move    $sp,$fp
391         lw      $ra,96($sp)
392         lw      $fp,92($sp)
393         addu    $sp,$sp,104
394         j       $ra
395         .end    main
396         .size   main, .-main
397         .ident  "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```

Listing 1: Código MIPS 32 generado por el compilador