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 **2**

2. Diseño e implementación **2**

3. Compilación **2**

4. Corridas de prueba **3**

5. Código MIPS 32 **5**

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

5. Código MIPS 32

```
	.file	1 "main.c"
	.section .mdebug.abi32
	.previous
	.abicalls
	.rdata
	.align	2
$LC0:
	.ascii	"Opening input file %s\n\000"
```

```
9     .align  2
10  $LC1:
11    .ascii  "rb\000"
12    .text
13    .align  2
14    .globl  open_infile
15    .ent  open_infile
16  open_infile:
17    .frame  $fp,40,$ra    # vars= 0, regs= 3/0, args= 16, extra= 8
18    .mask 0xd0000000,-8
19    .fmask  0x00000000,0
20    .set  noreorder
21    .cpload $t9
22    .set  reorder
23    subu  $sp,$sp,40
24    .cprestore 16
25    sw  $ra,32($sp)
26    sw  $fp,28($sp)
27    sw  $gp,24($sp)
28    move  $fp,$sp
29    sw  $a0,40($fp)
30    la  $a0,__sF+176
31    la  $a1,$LC0
32    lw  $a2,40($fp)
33    la  $t9,fprintf
34    jal $ra,$t9
35    lw  $a0,40($fp)
36    la  $a1,$LC1
37    la  $t9,fopen
38    jal $ra,$t9
39    move  $sp,$fp
40    lw  $ra,32($sp)
41    lw  $fp,28($sp)
42    addu  $sp,$sp,40
43    j $ra
44    .end  open_infile
45    .size open_infile, .-open_infile
46    .rdata
47    .align  2
48  $LC2:
49    .ascii  "Opening output file %s\n\000"
50    .align  2
51  $LC3:
52    .ascii  "wt\000"
53    .text
54    .align  2
55    .globl  open_outfile
56    .ent  open_outfile
57  open_outfile:
58    .frame  $fp,40,$ra    # vars= 0, regs= 3/0, args= 16, extra= 8
59    .mask 0xd0000000,-8
60    .fmask  0x00000000,0
61    .set  noreorder
62    .cpload $t9
63    .set  reorder
64    subu  $sp,$sp,40
65    .cprestore 16
66    sw  $ra,32($sp)
67    sw  $fp,28($sp)
68    sw  $gp,24($sp)
69    move  $fp,$sp
70    sw  $a0,40($fp)
71    la  $a0,__sF+176
72    la  $a1,$LC2
73    lw  $a2,40($fp)
74    la  $t9,fprintf
75    jal $ra,$t9
```

```
  lw   $a0,40($fp)
  la   $a1,$LC3
  la   $t9,fopen
  jal  $ra,$t9
  move  $sp,$fp
  lw   $ra,32($sp)
  lw   $fp,28($sp)
  addu  $sp,$sp,40
  j $ra
  .end  open_outfile
  .size open_outfile, .-open_outfile
  .rdata
  .align  2
$LC4:
  .ascii  "version\000"
  .align  2
$LC5:
  .ascii  "help\000"
  .align  2
$LC6:
  .ascii  "input\000"
  .align  2
$LC7:
  .ascii  "output\000"
  .align  2
$LC8:
  .ascii  "action\000"
  .data
  .align  2
  .type long_options.0, @object
  .size long_options.0, 96
long_options.0:
  .word $LC4
  .word 0
  .word 0
  .word 86
  .word $LC5
  .word 0
  .word 0
  .word 104
  .word $LC6
  .word 1
  .word 0
  .word 105
  .word $LC7
  .word 1
  .word 0
  .word 111
  .word $LC8
  .word 2
  .word 0
  .word 97
  .word 0
  .word 0
  .word 0
  .word 0
  .rdata
  .align  2
$LC9:
  .ascii  "Vhi:o:a:\000"
  .align  2
$LC10:
  .ascii  "-\000"
  .align  2
$LC11:
  .ascii  "Can not open input file '%s'\n\000"
  .align  2
```

```
$LC12:
  .ascii  "Can not open output file '%s'\n\000"
  .align  2
$LC13:
  .ascii  "decode\000"
  .align  2
$LC14:
  .ascii  "encode\000"
  .align  2
$LC15:
  .ascii  "Unknown action '%s'\n\000"
  .align  2
$LC16:
  .ascii  "Execute tp0 -h for help\n\000"
  .align  2
$LC17:
  .ascii  "Unknown option '%c'\n\000"
  .align  2
$LC18:
  .ascii  "Encode mode\n\000"
  .align  2
$LC19:
  .ascii  "Decode mode\n\000"
  .align  2
$LC20:
  .ascii  "Fail creating decoder\n\000"
  .align  2
$LC21:
  .ascii  "\000"
  .text
  .align  2
  .globl  main
  .ent  main
main:
  .frame  $fp,104,$ra   # vars= 56, regs= 3/0, args= 24, extra= 8
  .mask 0xd0000000,-8
  .fmask  0x00000000,0
  .set  noreorder
  .cpload $t9
  .set  reorder
  subu  $sp,$sp,104
  .cprestore 24
  sw  $ra,96($sp)
  sw  $fp,92($sp)
  sw  $gp,88($sp)
  move  $fp,$sp
  sw  $a0,104($fp)
  sw  $a1,108($fp)
  sw  $zero,32($fp)
  sw  $zero,36($fp)
  sw  $zero,40($fp)
  sw  $zero,48($fp)
$L20:
  addu  $v0,$fp,48
  sw  $v0,16($sp)
  lw  $a0,104($fp)
  lw  $a1,108($fp)
  la  $a2,$LC9
  la  $a3,long_options.0
  la  $t9,getopt_long
  jal $ra,$t9
  sw  $v0,44($fp)
  lw  $v1,44($fp)
  li  $v0,-1      # 0xffffffffffffffff
  bne $v1,$v0,$L22
  b $L21
$L22:
```

```
210   lw  $v0,44($fp)
211   addu  $v0,$v0,-86
212   sw  $v0,84($fp)
213   lw  $v1,84($fp)
214   sltu  $v0,$v1,26
215   beq $v0,$zero,$L36
216   lw  $v0,84($fp)
217   sll $v1,$v0,2
218   la  $v0,$L37
219   addu  $v0,$v1,$v0
220   lw  $v0,0($v0)
221   .cpadd  $v0
222   j $v0
223   .rdata
224   .align  2
225 $L37:
226   .gpword $L24
227   .gpword $L36
228   .gpword $L36
229   .gpword $L36
230   .gpword $L36
231   .gpword $L36
232   .gpword $L36
233   .gpword $L36
234   .gpword $L36
235   .gpword $L36
236   .gpword $L36
237   .gpword $L32
238   .gpword $L36
239   .gpword $L36
240   .gpword $L36
241   .gpword $L36
242   .gpword $L36
243   .gpword $L36
244   .gpword $L25
245   .gpword $L26
246   .gpword $L36
247   .gpword $L36
248   .gpword $L36
249   .gpword $L36
250   .gpword $L36
251   .gpword $L29
252   .text
253 $L24:
254   la  $t9,version
255   jal $ra,$t9
256   sw  $zero,80($fp)
257   b $L19
258 $L25:
259   la  $t9,help
260   jal $ra,$t9
261   sw  $zero,80($fp)
262   b $L19
263 $L26:
264   lw  $a0,optarg
265   la  $a1,$LC10
266   la  $t9,strcmp
267   jal $ra,$t9
268   beq $v0,$zero,$L20
269   lw  $a0,optarg
270   la  $t9,open_infile
271   jal $ra,$t9
272   sw  $v0,32($fp)
273   lw  $v0,32($fp)
274   bne $v0,$zero,$L20
275   la  $a0,__sF+176
276   la  $a1,$LC11
```

```
277   lw   $a2,optarg
278   la   $t9,fprintf
279   jal  $ra,$t9
280   li   $v0,1      # 0x1
281   sw   $v0,80($fp)
282   b $L19
283 $L29:
284   lw   $a0,optarg
285   la   $a1,$LC10
286   la   $t9,strcmp
287   jal  $ra,$t9
288   beq  $v0,$zero,$L20
289   lw   $a0,optarg
290   la   $t9,open_outfile
291   jal  $ra,$t9
292   sw   $v0,36($fp)
293   lw   $v0,36($fp)
294   bne  $v0,$zero,$L20
295   la   $a0,__sF+176
296   la   $a1,$LC12
297   lw   $a2,optarg
298   la   $t9,fprintf
299   jal  $ra,$t9
300   li   $v1,1      # 0x1
301   sw   $v1,80($fp)
302   b $L19
303 $L32:
304   lw   $v0,optarg
305   beq  $v0,$zero,$L36
306   lw   $a0,optarg
307   la   $a1,$LC13
308   la   $t9,strcmp
309   jal  $ra,$t9
310   bne  $v0,$zero,$L34
311   li   $v0,1      # 0x1
312   sw   $v0,40($fp)
313   b $L20
314 $L34:
315   lw   $a0,optarg
316   la   $a1,$LC14
317   la   $t9,strcmp
318   jal  $ra,$t9
319   bne  $v0,$zero,$L35
320   sw   $zero,40($fp)
321   b $L20
322 $L35:
323   la   $a0,__sF+176
324   la   $a1,$LC15
325   lw   $a2,optarg
326   la   $t9,fprintf
327   jal  $ra,$t9
328   la   $a0,__sF+176
329   la   $a1,$LC16
330   la   $t9,fprintf
331   jal  $ra,$t9
332   li   $v0,1      # 0x1
333   sw   $v0,80($fp)
334   b $L19
335 $L36:
336   la   $a0,__sF+176
337   la   $a1,$LC17
338   lw   $a2,44($fp)
339   la   $t9,fprintf
340   jal  $ra,$t9
341   la   $a0,__sF+176
342   la   $a1,$LC16
343   la   $t9,fprintf
```

```
344   jal $ra,$t9
345   li  $v1,1     # 0x1
346   sw  $v1,80($fp)
347   b $L19
348 $L21:
349   lw  $v0,40($fp)
350   bne $v0,$zero,$L38
351   la  $a0,__sF+176
352   la  $a1,$LC18
353   la  $t9,fprintf
354   jal $ra,$t9
355   addu  $v0,$fp,56
356   move  $a0,$v0
357   lw  $a1,32($fp)
358   lw  $a2,36($fp)
359   la  $t9,B64_encoder_create
360   jal $ra,$t9
361   addu  $v0,$fp,56
362   move  $a0,$v0
363   la  $t9,B64_encoder_start
364   jal $ra,$t9
365   lw  $v0,32($fp)
366   beq $v0,$zero,$L39
367   lw  $a0,32($fp)
368   la  $t9,fclose
369   jal $ra,$t9
370 $L39:
371   lw  $v0,36($fp)
372   beq $v0,$zero,$L38
373   lw  $a0,36($fp)
374   la  $t9,fclose
375   jal $ra,$t9
376 $L38:
377   lw  $v1,40($fp)
378   li  $v0,1     # 0x1
379   bne $v1,$v0,$L41
380   la  $a0,__sF+176
381   la  $a1,$LC19
382   la  $t9,fprintf
383   jal $ra,$t9
384   addu  $v0,$fp,72
385   move  $a0,$v0
386   lw  $a1,32($fp)
387   lw  $a2,36($fp)
388   la  $t9,decoder_create
389   jal $ra,$t9
390   beq $v0,$zero,$L42
391   la  $a0,__sF+176
392   la  $a1,$LC20
393   la  $t9,fprintf
394   jal $ra,$t9
395   li  $v0,1     # 0x1
396   sw  $v0,80($fp)
397   b $L19
398 $L42:
399   addu  $v0,$fp,72
400   move  $a0,$v0
401   la  $t9,decoder_start
402   jal $ra,$t9
403   beq $v0,$zero,$L43
404   li  $v1,1     # 0x1
405   sw  $v1,80($fp)
406   b $L19
407 $L43:
408   lw  $v0,32($fp)
409   beq $v0,$zero,$L44
410   lw  $a0,32($fp)
```

```
411   la   $t9,fclose
412   jal  $ra,$t9
413 $L44:
414   lw   $v0,36($fp)
415   beq  $v0,$zero,$L41
416   lw   $a0,36($fp)
417   la   $t9,fclose
418   jal  $ra,$t9
419 $L41:
420   lw   $v0,36($fp)
421   bne  $v0,$zero,$L46
422   la   $a0,$LC21
423   la   $t9,puts
424   jal  $ra,$t9
425 $L46:
426   sw   $zero,80($fp)
427 $L19:
428   lw   $v0,80($fp)
429   move   $sp,$fp
430   lw   $ra,96($sp)
431   lw   $fp,92($sp)
432   addu   $sp,$sp,104
433   j  $ra
434   .end   main
435   .size  main, .-main
436   .ident   "GCC: (GNU) 3.3.3 (NetBSD nb3 20040520)"
```

Listing 1: Código MIPS 32 generado por el compilador

66:20 Organización de Computadoras
Trabajo práctico #0: Infraestructura básica
2^{do} cuatrimestre de 2018

`$Date: 2018/09/08 23:16:30 $`

1. Objetivos

Familiarizarse con las herramientas de software que usaremos en los siguientes trabajos, implementando un programa (y su correspondiente documentación) que resuelva el problema piloto que presentaremos más abajo.

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes.

Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 6), la presentación de los resultados obtenidos explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo[1], y se valorarán aquellos escritos usando la herramienta TEX / LATEX.

4. Recursos

Usaremos el programa GXemul [1] para simular el entorno de desarrollo que utilizaremos en este y otros trabajos prácticos, una máquina MIPS corriendo una versión reciente del sistema operativo NetBSD [2].

En la clase del 21/8 hemos repasado los pasos necesarios para la instalación y configuración del entorno de desarrollo.

[1] `http://groups.yahoo.com/group/orga-comp`

5. Programa

Se trata de escribir, en lenguaje C, un programa para codificar y decodificar información en formato base 64: el programa recibirá, por línea de comando, los archivos o *streams* de entrada y salida, y la acción a realizar, codificar (acción por defecto) o decodificar. De no recibir los nombres de los archivos (o en caso de recibir `-` como nombre de archivo) usaremos los *streams* estándar, `stdin` y `stdout`, según corresponda. A continuación, iremos leyendo los datos de la entrada, generando la salida correspondiente. De ocurrir errores, usaremos `stderr`. Una vez agotados los datos de entrada, el programa debe finalizar adecuadamente, retornando al sistema operativo.

Estrictamente hablando, base 64 es un grupo de esquemas de codificación similares. En nuestra implementación, estaremos siguiendo particularmente el esquema establecido en [3], con el siguiente agregado: si se recibe una secuencia de caracteres inválida en la decodificación, debe asumirse como una condición de error que el programa deberá reportar adecuadamente y detener el procesamiento en ese punto.

5.1. Ejemplos

Primero, usamos la opción `-h` para ver el mensaje de ayuda:

```
$ 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
```

Codificamos un archivo vacío (cantidad de bytes nula):

```
$ touch /tmp/zero.txt
$ tp0 -a encode -i /tmp/zero.txt -o /tmp/zero.txt.b64
$ ls -l /tmp/zero.txt.b64
-rw-r--r--  1 user group 0 2018-09-08 16:21 /tmp/zero.txt.b64
```

Codificamos el caracter ASCII `M`,

```
$ echo -n M | tp0
TQ==
```

Codificamos los caracteres ASCII `M` y `a`,

```
$ echo -n Ma | tp0
TWE=
```

Codificamos `M a n`,

```
$ echo -n Man | tp0
TWFu
```

Codificamos y decodificamos:

```
$ echo Man | tp0 | tp0 -a decode
Man
```

Verificamos bit a bit:

```
$ echo xyz | tp0 | tp0 -a decode | od -t c
0000000    x   y   z  \n
0000004
```

Codificamos 1024 bytes, para verificar que el programa genere líneas de no mas de 76 unidades de longitud:

```
$ yes | head -c 1024 | tp0 -a encode
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkK
...
eQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5CnkKeQp5Cg==
```

Verificamos que la cantidad de bytes decodificados, sea 1024:

```
$ yes | head -c 1024 | tp0 -a encode | tp0 -a decode | wc -c
1024
```

Generamos archivos de tamaño creciente, y verificamos que el procesamiento de nuestro programa no altere los datos:

```
$ 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="‘expr $n + 1‘";
>         rm -f /tmp/in.bin /tmp/out.b64 /tmp/out.bin
> done
ok: 1
ok: 2
ok: 3
...
```

6. Informe

El informe deberá incluir al menos las siguientes secciones:

- Documentación relevante al diseño e implementación del programa;
- Comando(s) para compilar el programa;
- Las corridas de prueba, con los comentarios pertinentes;
- El código fuente, en lenguaje C, el cual también deberá entregarse en formato digital compilable (incluyendo archivos de entrada y salida de pruebas);
- El código MIPS32 generado por el compilador;
- Este enunciado.

El informe deberá entregarse en formato impreso y digital.

Referencias

[1] GXemul, `http://gavare.se/gxemul/`.

[2] The NetBSD project, `http://www.netbsd.org/`.

[3] RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies; sección 6.8, Base64 Content-Transfer-Encoding. `http://tools.ietf.org/html/rfc2045#section-6.8`.

[4] Base64 (Wikipedia). `http://en.wikipedia.org/wiki/Base64`.