U.B.A. FACULTAD DE INGENIERÍA

Departamento de Electrónica

Organización de computadoras 66-20

TRABAJO PRÁCTICO #0

Infraestructura básica

Curso: 2018 - 2do Cuatrimestre

Turno: Martes

GRUPO N°	
Integrantes	**Padrón**
Verón, Lucas	89341
Gamarra Silva, Cynthia Marlene	92702
Gatti, Nicolás	93570
Fecha de entrega:	23-11-2018
Fecha de aprobación:	
Calificación:	
Firma de aprobación:	

Observaciones:

Índice

1. Enunciado del trabajo práctico

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`.

1.1. Diseño e implementación

El programa que contiene la lógica de codificador y decodificador se encuentra en el archivo ***encode.c***. El codificador transforma expresiones con caracteres ASCII en base64, mientras que el decodificador hace el proceso inverso. Ambas funciones se implementan con diversos ciclos y recorridos, así como utilización de instrucciones aritmético lógicas (sumas, &, shifts, etc) que relacionan la codificación base 64 con ASCII.

Se creó un achivo ***file.c*** para el manejo de los archivos a codificar/decodificar. Además se creó otro archivo llamado ***command.c*** cuya función es tomar los parámetros del programa y realizar las acciones pertinentes a la misma.

1.2. Parámetros del programa

Se detallan a continuación los parámetros del programa

- -h: Visualiza la ayuda del programa, en la que se indican los parámetros y sus objetivos.
- -V: Indica la versión del programa.
- -i: Archivo de entrada del programa.
- -o: Archivo de salida del programa.
- -a: Acción a llevar a cabo: codificación o decodificación.

Se indica a continuación detalles respecto a los parámetros:

- Si no se explicitan -i y -o, se utilizarán stdin y stdout, respectivamente.
- -V es una opción "show and quit ". Si se explicita este parámetro, sólo se imprimirá la versión, aunque el resto de los parámetros se hayan explicitado.
- -h también es de tipo "show and quitz se comporta de forma similar a -V.
- en caso de que se use la entrada estándar (con comando echo texto — ./tp0 etc) y luego se especifique un archivo de salida con -i, prevalecerá el establecido por parámetro.

1.3. Compilación del programa

La compilación del archivo fuente se realiza de la siguiente manera

```
$ make
```

Para proceder a la ejecución del programa, se debe llamar a (según el nombre elegido al compilar):

```
$ ./tp0
```

seguido de los parámetros que se desee modificar, los cuales se indicaron en la sección 1.2.

En caso de ser entrada estándar (stdin) se podrá ejecutar de la siguiente forma:

```
$ echo texto | ./tp0
```

También en este caso, se indican a continuación los parámetros a usar.

2. Pruebas con archivo bash test.sh

Se realizan corridas de prueba con el siguiente script

```
#!/bin/bash

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.bin -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 o;
done
```

El cual no presenta errores en ninguna de las corridas llevadas a cabo.

3. Pruebas realizadas

Todas las pruebas que se presentan a continuación, están codificadas en los archivos de prueba ***.txt de forma que puedan ejecutarse y comprobar los resultados obtenidos.

Se indicaran a continuación lo siguiente: comandos para ejecutarlas, líneas de código que las componen y resultado esperado.

3.0.1. Generales

- Mensaje de ayuda

```
$ ./tp0 -h  o ./tp0 --help

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
```

- Mensaje de version

```
$ ./tp0 -V  o ./tp0 --version
Version: 0.1

```

- Archivo de entrada no válido

```
$ ./tp0 -i archivoInvalido.txt

File Open Error; No such file or directory


```

Referencias

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

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

[3] Kernighan, B. W. - Ritchie, D. M. - *C Programming Language* - 2nd edition - Prentice Hall - 1988.

[4] *GNU Make* - https://www.gnu.org/software/make/

[5] *Valgrind* - http://valgrind.org/

A. Código fuente

A.0.1. main.c

```
/**
 * Created by gatti2602 on 12/09/18.
 * Main
 */

#define ERROR 0
#define FALSE 0
#define TRUE 1


#include <stdio.h>
#include <string.h>
#include <getopt.h>

#include "command.h"
#include "file.h"
#include "encode.h"

/*enum ParameterState {
         OKEY = 0, INCORRECT_QUANTITY_PARAMS = 1, INCORRECT_MENU = 2, ERROR_FILE = 3,
    ERROR_MEMORY = 4
};
*/

/*
 * static char encoding_table[] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
                                   'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',
                                   'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X',
                                   'Y', 'Z', 'a', 'b', 'c', 'd', 'e', 'f',
                                   'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',
                                   'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
                                   'w', 'x', 'y', 'z', '0', '1', '2', '3',
                                   '4', '5', '6', '7', '8', '9', '+', '/'};
 */

/*
 * char* encode(char* buffer){
        unsigned char* buff = (char*)malloc(sizeof(char)*4);
        unsigned char b1 = buffer[0];
        unsigned char b2 = buffer[1];
        unsigned char b3 = buffer[2];

        unsigned char b1aux = b1 >> 2;
        buff[0] = encoding_table[(int)b1aux];
        unsigned char b2aux = b1 << 6;
        b2aux = b2aux >> 2;
        b2aux = b2aux | (b2 >> 4);
        buff[1] = encoding_table[(int)b2aux];
```

```
48          if(b2 != '\n'){
49                  if(b3 != '\n'){
50                          unsigned char b3aux = b3 >> 6;
51                          unsigned char b3aux2 = b2 << 4;
52                          b3aux2 = b3aux2 >> 2;
53                          b3aux = b3aux | b3aux2;
54                          buff[2] = encoding_table[(int)b3aux];
55                          unsigned char b4aux = b3 << 2;
56                          b4aux = b4aux >> 2;
57                          buff[3] =encoding_table[(int)b4aux];
58                          return buff;
59                  }
60                  unsigned char b3aux = b3 >> 6;
61                  unsigned char b3aux2 = b2 << 4;
62                  b3aux2 = b3aux2 >> 2;
63                  b3aux = b3aux | b3aux2;
64                  buff[2] = encoding_table[(int)b3aux];
65                  buff[3] = '=';
66                  return buff;
67          }
68          buff[2] = buff[3] = '=';
69          return buff;
70  }
71   */
72
73  /*
74   * Función de conveniencia.
75   */
76  /*
77  void executeEncode(File* input, File* output) {
78          FileOpenForRead(input);
79          int ichar = fgetc(input->file);
80
81          FileOpenForWrite(output);
82
83          char buffer[255];
84          int i = 0;
85          char* buff = (char*)malloc(sizeof(char)*3);
86          while(ichar != EOF){
87
88                  char character = ichar;
89                  buffer[i] = character;
90                  ichar = fgetc(input->file);
91                  i++;
92                  if(i == 3){
93                          buff = Encode(buffer);
94                          //Escribo en el output.
95                          fputs(buff,output->file);
96                          i = 0;
97                  }
98          }
99          if(i == 2){
100                 buff = Encode(buffer);
101                 fputs(buff,output->file);
102         }
103         //Cierro los archivos.
104         FileClose(input);
105         FileClose(output);
106 }
107 */
108 /*
109  * Función de conveniencia.
110  */
111 /*
112 void executeDecode(File* input, File* output) {
113         FileOpenForRead(input);
114         int ichar = fgetc(input->file);
```

```
115
116                 FileOpenForWrite(output);
117
118                 char buffer[255];
119                 int i = 0;
120                 char* buff = (char*)malloc(sizeof(char)*3);
121                 while(ichar != EOF){
122
123                         char character = ichar;
124                         buffer[i] = character;
125                         ichar = fgetc(input->file);
126                         i++;
127                         if(i == 4){
128                                 buff = Decode(buffer);
129                                 //Escribo en el output.
130                                 fputs(buff,output->file);
131                                 i = 0;
132                         }
133                 }
134                 //Cierro los archivos.
135                 FileClose(input);
136                 FileClose(output);
137 }
138 */
139
140 int main(int argc, char** argv) {
141
142     struct option arg_long[] = {
143             {"input",   required_argument,  NULL,  'i'},
144             {"output",  required_argument,  NULL,  'o'},
145             {"action",  required_argument,  NULL,  'a'},
146             {"help",    no_argument,        NULL,  'h'},
147             {"version", no_argument,        NULL,  'V'},
148     };
149     char arg_opt_str[] = "i:o:a:hV";
150     int arg_opt;
151     int arg_opt_idx = 0;
152
153     CommandOptions cmd_opt;
154     CommandCreate(&cmd_opt);
155
156     while((arg_opt =
157             getopt_long(argc, argv, arg_opt_str, arg_long,&arg_opt_idx)) != -1){
158         switch(arg_opt){
159                 case 'i':
160                         CommandSetInput(&cmd_opt, optarg);
161                         break;
162                 case 'o':
163                 CommandSetOutput(&cmd_opt, optarg);
164                 break;
165                 case 'h':
166                         CommandHelp();
167                         return 0;
168                 case 'V':
169                         CommandVersion();
170                         return 0;
171                 case 'a':
172                     CommandSetEncodeOpt(&cmd_opt, optarg);
173                                 break;
174                 default:
175                         CommandSetError(&cmd_opt);
176                         CommandHelp();
177                         break;
178         }
179     }
180     if(!CommandHasError(&cmd_opt))
181         CommandProcess(&cmd_opt);
```

```
182     else
183         return 1;
184     return 0;
185
186 }
```