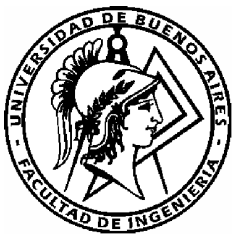

Facultad de Ingeniería Universidad de Buenos Aires

66.20 Organización de Computadoras

Trabajo Práctico N°0 : Infraestructura Básica

Integrantes:

Padrón	Nombre
76187	Lucas Hemmingsen

Índice

1. Objetivos

El objetivo del presente trabajo consiste en familiarizarnos con las herramientas con las cuales se trabajará en el transcurso de la cursada. Las mismas son:

- Compilador GCC
- Shell de Unix
- Emulador GXEmul
- Sistema de Composición LaTeX

2. Introducción Teórica

Para poner en práctica las herramientas citadas, el presente TP consiste en realizar un programa en lenguaje C que codifique información un conjunto arbitrario de bytes en un conjunto formado por caracteres ASCII utilizando el esquema de codificación *Base 64* A continuación se explican brevemente los conceptos y herramientas utilizadas necesarias para llevar a cabo esta tarea:

2.1. Shell de Unix

Un shell es un programa intérprete de comandos. Los comandos utilizados en el presente trabajo son los siguientes:

- ***pipe***: Toma lo que se encuentra a la izquierda del pipe como la salida estándar (*stdout*) por defecto y lo pasa como entrada estándar (*stdin*) a lo que se este a la derecha del pipe. Puede servir para pasarle a un programa un archivo y que este lo reciba por la entrada estándar
- **echo**: Imprime un texto en *stdout*.
- **cat**: Con este comando se pueden ver, así como también concatenar, varios archivos. Escribe en la salida estándar.

2.2. Compilador GCC

GCC es un compilador gratuito, Open Source y multiplataforma que permite compilar código C. El mismo es un compilador estándar, es decir, que respeta la norma ISOC99 o ANSI dependiendo de su versión, por lo cual se utilizan las bibliotecas estándar para trabajar con el mismo. En esta asignatura el mismo es de gran utilidad dado que permite tener acceso al código *assembly* equivalente a nuestro programa en C. A continuación se detallan brevemente algunos parámetros que podemos darle al GCC:

- **-Wall:** Activa todos los warnings.
- **-o file** Almacena la salida del programa en un archivo (*en este caso en file*)
- **-O0:** Desactiva las optimizaciones
- **-O3:** Activa todas las optimizaciones
- **-g:** Genera código agregando líneas para el *debbugger*
- **-S:** Detiene el compilador luego de generar el código *assembly*
- **-mrnames(solo para MIPS):** Indica al compilador que genera la salida utilizando nombre de registro en lugar de número de registro

Para generar el archivo "main.s" con el código *assembly* se debe ejecutar el siguiente comando:

```
gcc -Wall -O0 -S -mrnames main.c
```

Para compilar el código se debe ejecutar el siguiente comando:

```
gcc -Wall -O0 -o tp0 main.c
```

2.3. Emulador GXEmul

Es un emulador gratuito y Open Source de la arquitectura de computadores MIPS. Tiene como ventajas que puede correr algunos sistemas operativos sin modificaciones (especialmente *netBSD*), su portabilidad y velocidad de emulación dado que realiza la traducción binaria en tiempo real.

3. Desarrollo

A partir de los conceptos teóricos desarrollados en la sección anterior (2) se procedió a crear un programa en C que pudiese codificar información mediante el esquema de codificación *Base64* a partir de una muestra:

3.1. Modo de Operación

Para compilar la aplicación hay que posicionarse en el directorio en el que se encuentra el archivo "tp0.c", y ejecutar el siguiente comando:

```
gcc -Wall -O0 -o tp0 tp0.c
```

La aplicación desarrollada debe recibir ya sea por entrada estándar o mediante archivos de entrada, una muestra de N bytes.

- **-e –encode:** Por medio de este comando se indica que el algoritmo debe codificar la información pasada a *Base64*.
- **-d –decode:** Por medio de este comando se indica que el algoritmo debe decodificar la información pasada desde *Base64*.
- **-i –input file:** Por medio de este comando se indica el archivo de entrada para realizar la codificación o decodificación. En caso de no indicar un archivo de entrada, el programa tomará la entrada de *stdin*.
- **-o –output file:** Por medio de este comando se indica el archivo de salida para realizar la codificación o decodificación. En caso de no indicar un archivo de salida, el programa escribirá la salida en *stdout*.
- **-h –help:** Despliega la ayuda.
- **-v –version:** Muestra la versión.

3.2. Ejemplos

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

```
$tp0 -h
OPTIONS:
-e --encode Encodes to Base64
-d --decode Decodes from Base64
-i --input file Reads from file or stdin
-o --output file Writes to file or stdout
-v --version Show version string
-h --help Print this message and quit
```

Si quisieramos codificar y decodificar el arreglo de bytes "foo"debemos ejecutar:

```
$echo -n "foo" | tp0 -e | tp0 -d
foo
```

Lo que hace esta línea es tomar el arreglo de bytes "fooçomo entrada del programa para codificar, tomándolo desde la entrada estándar, codifica la información a *Base64* y escribe la salida por la salida estándar, para luego tomar esta salida como entrada estándar para decodificar la información, escribiendo la salida por salida estándar, siendo "foo.^{es}ta salida. Si se está codificando información y luego se desea ver la salida codificada, es recomendable especificar un archivo de salida y luego para visualizar el contenido del archivo binario en formato hexadecimal y separado en bytes, utilizar el comando *hexdump*:

```
$echo -n "foo" | tpo -e -o foo.txt
```

```
hexdump -C foo.txt
```

De cualquier manera, si se especifica un archivo, este puede verse con cualquier editor de texto, o por consola con el comando

```
cat foo.txt
```

4. Implementación

4.1. Determinaciones Previas

- Siempre que se ingrese el parámetro de ayuda (-h o –help) junto con cualquier otro, se ignorará este último. Por ejemplo "echo "abc" ./tp0 -h" sólo muestra el mensaje de ayuda.
- Si no se especifica codificar (-e) o decodificar (-d) el programa mostrará el menú de ayuda y finalizará la ejecución.
- Si alguno de los archivos pasados por parámetro no puede ser leído, la ejecución termina mostrando el error por stderr.
- Si se selecciona codificar y decodificar no se ejecutará la sentencia y mostrará el menú de ayuda.
- Si se ingresan archivos a través de los argumentos y además alguno por stdin, solo se tendrán en cuenta los primeros, ignorando el ingresado por entrada estándar.
- Si al decodificar la información se obtiene algún caracter invalido el programa mostrará un error por stderr y finalizará.

4.2. Código Fuente

```
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>

#define ALPHABET "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
#define ALPHABET_LENGTH 64
#define ENDING_CHAR '='
#define ENDING_CHAR_INT 61
#define ENCODE_SIZE 6
#define DECODE_SIZE 8
#define BUFFERSIZE 12

void error()
{
        fprintf(stderr, "Error al decodificar\n");
        exit(EXIT_FAILURE);
```

```
}

int indexOf(char aChar){
        int i;
        for(i= 0;i < ALPHABET_LENGTH;i++){
                char alph_char = ALPHABET[i];
                if(alph_char == aChar)
                        return i;
        }
        error();
        return -1;
}

int count = 0;
int acum = 0;
int length = 0;

void doEncode(int value)
{
        acum = (acum << 1) | value;
        count++;
        if(count == ENCODE_SIZE){
                ++length;
                putc((int)ALPHABET[acum], stdout);
                count = 0;
                acum = 0;
        }
}

void encode64(char * const input, size_t amount)
{
        int i,j;

        for (i = 0; i< amount; i++)
        {
                char aChar =  input[i];
                for (j = DECODE_SIZE -1 ; j >=0 ; --j)
                        doEncode(((aChar >> j) & 0x01));
        }
}

void doDecode(int value)
{
        acum = (acum << 1) | value;
        count++;
        if(count == DECODE_SIZE){
                ++length;
                putc(acum, stdout);
                count = 0;
                acum = 0;
```

```
        }
}

void decode64(char * const input, size_t amount)
{
        int i,j;
        char aChar;

        for (i = 0; i< amount && (input[i] != ENDING_CHAR); i++)
        {
                int index =  indexOf(input[i]);
                aChar =  index;
                for (j = ENCODE_SIZE - 1; j >= 0; j--)
                        doDecode(((aChar >> j) & 0x01));
        }
}

void process(int encode, char* buffer, size_t amount)
{
        if (encode)
                encode64(buffer, amount);
        else
                decode64(buffer, amount);
}

void closeProcess(int encode)
{
        int i;
        if(encode)
        {
                if(count < ENCODE_SIZE && count > 0)
                {
                        while(count != 0)
                        {
                                doEncode(0);
                        }

                        for (i = 0; i < 4 - length % 4; i++)
                                putc(ENDING_CHAR_INT, stdout);
                }
        }
        fflush(stdout);
}

void openInFile(char * file){
        FILE * ifp = freopen(file, "rb", stdin);
        if (ifp == NULL) {
          fprintf(stderr, "Can't open input file\n");
          exit(EXIT_FAILURE);
        }
```

```
}

void openOutFile(char * file){
        FILE * ofp = freopen(file, "wb", stdout);
        if (ofp == NULL) {
          fprintf(stderr, "Can't open output file\n");
          exit(EXIT_FAILURE);
        }
}

void print_help(){
        fprintf(stdout, "Modo de empleo: ./tp0 [OPTIONS]\n"
"OPTIONS:\n"
"-e --encode Encodes to Base64\n"
"-d --decode Decodes from Base64\n"
"-i --input file Reads from file or stdin\n"
"-o --output file Writes to file or stdout\n"
"-v --version Show version string\n"
"-h --help Print this message and quit\n");
}

void print_version(){
        fprintf(stdout, "Version 1.0\n");
}

void selectionError() {
        fprintf(stderr, "Error: decode and encode selected.\n");
        print_help();
        exit(EXIT_FAILURE);
}
int main(int argc,  char* const* argv)
{
        int encode = -1;
        int c;
        extern char *optarg;
        extern int optind, opterr, optopt;
        /* Una cadena que lista las opciones cortas válidas */
        const char* const opcionesCortas = "edi:o:vh";

        /* Una estructura de varios arrays describiendo las opciones largas */
        const struct option opcionesLargas[] = {
          { "help",          no_argument, NULL,'h'},
          { "version", no_argument, NULL,'v'},
          { "input",no_argument, NULL,'i'},
          { "output",no_argument, NULL,'o'},
          { "encode",no_argument, NULL,'e'},
          { "decode",no_argument, NULL,'d'},
          { NULL,     no_argument, NULL, 0 }
        };
```

```
        opterr = 0;

        while ((c = getopt_long(argc, argv, opcionesCortas, opcionesLargas, NULL)) != -1)
         switch (c)
           {
           case 'e':
             if (encode == -1)
                     encode = 1;
             else
                     selectionError();
             break;
           case 'd':
             if (encode == -1)
                     encode = 0;
             else
                     selectionError();
             break;
           case 'i':
             openInFile(optarg);
             break;
           case 'o':
             openOutFile(optarg);
             break;
           case 'h':
             print_help();
             exit(EXIT_SUCCESS);
           case 'v':
             print_version();
             exit(EXIT_SUCCESS);
           case '?':
             if (optopt == 'i' || optopt == 'o')
               fprintf (stderr, "Option -%c requires an argument.\n", optopt);
             else
               fprintf (stderr, "Unknown option character\n");
             print_help();
             exit(EXIT_FAILURE);
           default:
             abort ();
           }

    /*****************************************************************/

        char buffer[BUFFERSIZE];
        size_t amount;

        while (1) /* break with ^D or ^Z */
        {
                amount=fread(buffer, 1, BUFFERSIZE , stdin);
                process(encode, buffer, amount);
                if(feof(stdin))
```

```
                    break;
        }

        closeProcess(encode);
        fclose (stdin);
        fclose (stdout);

        return 0;
}
```

5. Pruebas

- **Codificación y Decodificación trivial:**
  Si ejecutamos la siguiente secuencia de comandos en el shell:

  ```
  $ echo -n foo | ./tp0 -e | ./tp0 -d
  foo
  ```

  Este caso demuestra que el programa al codificar y decodificar devuelve la misma información de entrada.
  Si ejecutamos la siguiente secuencia de comandos en el shell:

  ```
  $ echo -n foo | ./tp0 -e
  Zm9v
  ```

  nos devuelve la misma codificación que en el ejemplo del enunciado.
  Si ejecutamos la siguiente secuencia de comandos en el shell:

  ```
  $ echo -n Zm9v | ./tp0 -d
  foo
  ```

  volvemos a obtener la información pasada antes de codificarla.

- **Selección de un algoritmo por defecto:**
  En el caso de no explicitar la opción de codificación o decodificación, el programa por defecto codificará la información que se le pase.

  ```
  $ echo -n hola | ./tp0
  aG9sYQ==
  ```

- **Caracteres fuera del alfabeto al decodificar:**
  Al decodificar, al pasarle como entrada caracteres que no se encuentran en el alfabeto, el programa indicará que hay un error al decodificar y detendrá la ejecución del mismo.

```
$ echo -n **foo | ./tp0 -d
Error al decodificar
```

- **Selección de un archivo inexistente:**

```
$ ./tp0 -e -i sdsd.txt
Can't open input file
```

  Si consideramos que el archivo "sdsd.txt" no existe, nos saldrá por stderr una línea indicando el error.

- **Codificación y decodificación de un archivo binario:**
  Se generó un archivo binario de 2KB y se encriptó el mismo utilizando nuestro programa y utilizando el programa base64 incluido en el sistema operativo y se compararon ambas salidas. Luego se desencriptaron los datos encriptados y se comparó con el archivo original.

```
$ dd if=/dev/urandom of=file.txt bs=2048 count=1
1+0 records in
1+0 records out
2048 bytes (2.0 kB) copied, 0.000291106 s, 7.0 MB/s
$ ./tp0 -e -i file.txt -o salida.txt
$ base64 -w0 file.txt > salidabase64.txt
$ diff salida.txt salidabase64.txt
$ ./tp0 -d -i salida.txt -o entrada.txt
$ diff file.txt entrada.txt
$
```

- **Script de pruebas:**
  En la realizacion de este trabajo práctico se utilizaron muchas pruebas en todas las etapas del desarrollo. Para facilitar las mismas se creó un script (adjunto en el presente trabajo) que realiza todas las pruebas automáticamente y entrega un informe detallando los resultados de los mismos. Para las pruebas realizadas en este trabajo se lo probó contra la versión del base64 instalada en el sistema operativo con la opcion -w0 para que no agrupe por líneas.
  A continuación se muestra la salida de dicho sistema, a modo de ejemplo:

El mismo prueba tanto los distintos modos de entrada y salida del ejecutable como también todos los casos de pruebas que se plantearon. Los archivos de prueba que se utilizaron en el trabajo son los siguientes (archivos adjuntos en la carpeta tests):

- **vacio.test:** Archivo vacío con el objetivo probar el funcionamiento si no hay entrada.
- **test_1_caracter:** Archivo con un sólo caracter con el objetivo de probar el funcionamiento con menos de 24 bytes.
- **test_2_caracteres:** Archivo con dos caracteres con el objetivo de probar el funcionamiento con menos de 24 bytes.
- **test_2_caracteres:** Archivo con 3 caracteres con el objetivo de probar el funcionamiento con 24 bytes.
- **Otros archivos de distintos valores para probar los resultados:**
  - **Archivos generados manualemente:** archivo_1.test, archivo_2.test
  - **Archivos binarios generados aleatoriamente (generados a partir de /dev/urandom):** random_50B.test, random_200B.test, random_1k.test, random_10k.test, random_20k.test

- **Selección de codificación y decodificación:**
  Si ejecutamos la siguiente sentencia diciéndole al programa que codifique y decodifique nos mostrará el menú de ayuda.

```
echo -n hola | ./tp0 -e -d
Modo de empleo: ./tp0 [OPTIONS]
OPTIONS:
-e --encode Encodes to Base64
```

```
-d --decode Decodes from Base64
-i --input file Reads from file or stdin
-o --output file Writes to file or stdout
-v --version Show version string
-h --help Print this message and quit
```

- **Ayuda:**
  $./tp0 -h
  Modo de empleo: ./tp0 [OPTIONS]
  OPTIONS:
  -e –encode Encodes to Base64
  -d –decode Decodes from Base64
  -i –input file Reads from file or stdin
  -o –output file Writes to file or stdout
  -v –version Show version string
  -h –help Print this message and quit

- **Versión:**
  $./tp0 -v
  Versión: 1.0

  Muestra la versión del programa.

6. Conclusión

En el presente trabajo se aprendió a utilizar las herramientas que utilizaremos en los próximos trabajos prácticos. Se aprendió a utilizar el emulador GXEmul y compilar programas para arquitectura MIPS, tanto a forma binaria como assembly MIPS.

Referencias

[1] "Documentación librería GetOpt"`http://www.gnu.org/software/libc/manual/html_node/Getopt.html`