

Trabajo Práctico N° 0: Infraestructura Básica

Martinez Ariel, *Padrón Nro. 88573*
`arielcorreofiuba@gmail.com.ar`

Nestor Huallpa, *Padrón Nro. 88614*
`huallpa.nestor@gmail.com`

Pablo Sivori, *Padrón Nro. 84026]*
`sivori.daniel@gmail.com`

1° Entrega: 28/03/2017

1er. Cuatrimestre de 2017
66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

En el presente trabajo práctico se describirán todos los pasos y conclusiones relacionadas al desarrollo e implementación de la codificación y decodificación de datos formateados en base 64.

Índice

1. Introducción

El objetivo del presente trabajo práctico es familiarizarse con el emulador gxemul mediante la implementación de un programa que codifica y decodifica datos en base 64.

2. Implementación

2.1. Lenguaje

Como lenguaje de implementación se eligió ANSI C ya que el mismo permite una alta portabilidad entre diferentes plataformas. El desarrollo del programa se realizó usando un editor de texto (gedit,vim, kwrite) y compilando los archivos fuente con GCC que viene en linux. Para compilar, ejecutar el siguiente comando:

```
$ make
```

2.2. Descripción del programa

Completar con el diseño del programa ...

La función `main` se encuentra en `tp0.c` y se encarga de interpretar las opciones y argumentos. En caso de ser una opción, como ayuda o versión, se imprime el mensaje correspondiente y finaliza la ejecución. Cuando no es una opción de ayuda o versión, se procede a procesar los datos de entrada. La salida de estas funciones proveen un código de error que sirve como salida del programa. Los mensajes de versión y ayuda se imprimen por `stdout` y el programa finaliza devolviendo 0 (cero) al sistema. Los mensajes de error se imprimen por la salida de errores (`stderr`) y el programa finaliza devolviendo 1 (uno) al sistema.

2.2.1. Errores posibles

1. El procesamiento de la entrada estándar causó el agotamiento del heap.
2. La invocación del programa es incorrecta.
3. Alguno de los archivos es inexistente.

Se contemplan otros errores gracias al uso de la variable externa `errno`. Cuando ocurre un error inesperado, el mismo es informado por `stderr` y finaliza el programa liberando la memoria que se había solicitado hasta el momento. (con la función `perror()`).

2.3. Desarrollo de actividades

1. Se instaló en un linux un repositorio de fuentes (GIT) para que al dividir las tareas del TP se pudiese hacer una unión de los cambios ingresados por cada uno de los integrantes más fácilmente.

2. Cada persona del grupo se comprometió a que sus cambios en el código fuente y los cambios obtenidos del repositorio que pudiesen haber subido los otros integrantes del grupo, sean compilados los sistemas operativos Linux y el NetBSD, asegurando así portabilidad entre plataformas planteada en el enunciado.
3. Se estableció que todos los integrantes en mayor o menor medida, contribuyan en el desarrollo de todas las partes del código para que nadie quede en desconocimiento de lo que se hizo en cada sección. Si bien cada parte del código fue realizada por diferentes integrantes (parseo de los argumentos, lectura de los ficheros, etc), todos nos familiarizamos con cada una de estas partes y cumplimos la función de testers de lo hecho por otros integrantes.
4. Se propuso como meta paralela, hacer el programa lo mas reutilizable posible tratando de que los métodos desarrollados, sean los suficientemente modulares como para su posible reutilización en los TPs venideros.
5. Para poder generar el código assembler a partir del código fuente, dentro de NETBSD se utilizó gcc con la siguiente opción:

gcc -S main.c

6. Para crear el presente informe se debe utilizar el comando make en el directorio informe.

2.4. Corridas de pruebas

Para correr las pruebas se debe ejecutar el comando `make` del directorio pruebas.

1. Prueba 1

Corriendo el script provisto por la catedra...

2. Prueba 2

Ejecutando el siguiente comando.

```
$ ./pruebas.sh
```

3. El código fuente, en lenguaje C

4. El código MIPS32 generado por el compilado

5. Conclusiones

1. Si bien lo solicitado por el programa no era excesivamente difícil, la realización completa del TP llevó cierta dificultad al tener que realizarlo en el contexto solicitado: alta portabilidad, desarrollo en C, e informe hecho en LaTeX.
2. En el primer caso la dificultad radicaba en tener configurado y funcionando el GXEmul dentro de un Linux, y lograr que en ambos casos el programa compile y corra sin problemas.
3. Debido a nuestro desconocimiento con LaTeX, tuvimos que invertir tiempo en encontrar forma de realizar el presente documento de la manera más correcta posible
4. En cuanto al trabajo grupal en si mismo, no hubo inconvenientes de ningún tipo ya que al ser el grupo relativamente chico y tener conocimiento del manejo del versionado de un proyecto ante cambios ingresado por los integrantes (por medio del GIT), la introducción de modificaciones y correcciones fué fluida.

6. Enunciado del trabajo practico

66:20 Organización de Computadoras
Trabajo práctico #0: Infraestructura básica
1^{er} cuatrimestre de 2017

\$Date: 2017/03/21 22:54:33 \$

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 grupo¹, 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 7/3 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 2017-03-19 15:14 /tmp/zero.txt.b64
```

Codificamos el carácter 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.

7. Fechas

- Entrega: 28/3/2017;
- Vencimiento: 10/4/2017.

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>.