Trabajo Práctico 0

[6620] Organización de computadoras
Curso 2
Segundo cuatrimestre de 2020

Nombre y apellido	Padrón	E-mail	User Slack
Maria Sol Fontenla	103870	msfontenla@fi.uba.ar	Maria Sol Fontenla
Stephanie Izquierdo	104196	sizquierdo@fi.uba.ar	Stephanie Izquierdo
Agustina Segura	104222	asegura@fi.uba.ar	Agustina Segura

1 Introducción

El objetivo del presente trabajo es el de implementar un programa capaz de codificar y decodificar según el método de base 64, utilizando el programa QEMU.

2 Descripción general del trabajo

El trabajo se desarrolló en el lenguaje de programación C, y se compiló en el simulador de entorno de desarrollo QEMU. Para su correcta ejecución en el simulador, se necesita calcular previamente el endianness de la máquina MIPS, ya que la codificación requiere recorrer posiciones de memoria contiguas. Debido a esto, era necesario saber si la memoria se ordenaba de manera big endian o little endian para saber cómo recorrerla. Para esto se realizó un algoritmo aparte que nos permitió calcular el ordenamiento en memoria. Para compilar el programa, utilizamos la línea:

```
gcc -Werror tpOrga.c -o tp -g -lm
```

3 Supuestos

- Si se utiliza el comando -d (decode) se debe pasar por terminal el archivo previamente codificado en base 64 y no por entrada estándar.
- Solo se puede codificar / decodificar un archivo por separado, no los dos juntos.
- El archivo a decodificar debe ser un archivo codificado en base64.
- Al codificar, el texto se genera sin salto de línea debido a que este no pertenece a los caracteres representables en 64 bits.
- Si por terminal se ingresa el comando -o (output file) seguido del nombre de un archivo, el texto ingresado se codificará y guardará en el archivo y no se mostrará por salida estándar.

3.1 Criterios de diseño

- Para la parte de codificacion se va a ir leyendo de a 3 bytes del archivo, cargando esos bytes en un buffer. Luego se invocara a la funcion de codificacion en donde a cada letra se la pasara a binario. Al finalizar el pasaje de todo el buffer a binario, se invocara a la funcion separador de a seis en donde se ira tomando de a 6 bits para luego obtener un buffer de 4 bytes.
- Para la decodificacion se realizo algo similar. En vez de leer de a 3 bytes en este caso, leemos de a 4. Volvemos a realizar el pasaje de cada posicion del buffer a binario y una vez finalizado se invocara a la funcion separador de a ocho, en donde se ira tomando de 8 bits para volver nuevamente a un buffer de 3 y volver al buffer original.
- En el caso de que llegue un archivo invalido, imprimira un error y devolvera 0.
- Ademas, con respecto a los formatos de los archivos soportados, cuando se trata de archivos binarios estos se deben ingresar en el programa por stdin, ya que cuando recibe el nombre de un archivo, el programa lo abre como archivo de texto. Lo mismo pasa con los archivos que son muy grandes y no entran en memoria.
- Decidimos utilizar la biblioteca math.h para poder utilizar la funcion pow la cual nos permite obtener un numero elevado a cierta potencia para poder asi obtener los numeros en decimal almacenados como binarios. Optamos por hacerlo de esta manera, en lugar de realizar un shift ya que al tener el numero binario dentro de un vector, preferimos iterar sobre este e ir elevando cada posicion en lugar de hacer un shift, aunque sabemos que esto es menos eficiente.

- Cuando se imprime la version del programa, se muestra por pantalla "codificacion y decodificacion en base 64", para asi indicar que funcionamiento cumple nuestro codigo.

3.2 Casos de prueba

Para testear el programa, se realizo un script de bash que al ejecutarse nos muestra la salida esperada y la salida obtenida codificando, y luego la codificacion seguida de su decodificacion en base64. Ademas, añadimos casos de prueba con los generadores de caraterers /dev/zero y /dev/null.

3.3 Conclusiones

Pudimos familiazrizarnos con la herramienta de software qemu, instalandola y ejecutadonla. Tambien pudimos familiarizarnos con la codificacion y decodificacion en base 64.