DCC031 – Redes de computadores

Trabalho prático 0 – Codificação Base64

Professor: Daniel Fernandes Macedo

Data de entrega: 31/08/2011

Trabalho em grupos de dois alunos.

Valor do trabalho: 5 pontos

1. Descrição do trabalho

Neste trabalho iremos implementar o algoritmo de codificação e de decodificação de dados Base64 (http://en.wikipedia.org/wiki/Base64), a primeira etapa para o desenvolvimento do TP1 (iremos implementar um serviço de e-mail).

O algoritmo Base64 foi desenvolvido nos primórdios da Internet para codificar dados binários em dados ASCII, para que fossem enviados em protocolos de rede que enviavam dados no formato texto (e-mail, Usenet, dentre outros). Estes aplicativos não aceitavam caracteres diferentes dos especificados no padrão ASCII, pois tinham sido desenvolvidos somente para o envio de textos.

Com o avanço da Internet, as pessoas decidiram enviar dados em binário dentro destes protocolos (por exemplo, enviar uma planilha, uma foto ou um PDF em anexo a um e-mail). O problema é que o ASCII representa caracteres com apenas 7 bits, enquanto um dado em binário pode empregar todos os 8 bits de um byte para codificar informação. Dependendo dos programas empregados para o envio ou recepção de dados, estes poderiam descartar os bytes onde o bit mais significativo estava setado em um, ou mesmo zerá-lo antes de passar os dados para frente, corrompendo o arquivo.

O Base64, definido na RFC 4648, foi inventado para resolver esse problema. No Base64, convertemos o arquivo em uma sequência de caracteres, escolhidos entre 64 caracteres ditos *"imprimíveis"*: **A-Z**, **a-z**, **0-9**, **+** e **/**. Assim, quebramos o arquivo em unidades de 6 bits, que são convertidas uma a uma para um dos 64 caracteres possíveis, por isso o nome Base64. A tabela 1 (na próxima página) é empregada para a conversão.

Existe ainda mais um detalhe para que o Base64 funcione: os arquivos nos computadores são múltiplos de 8 bits, enquanto o base 64 codifica informações de 6 bits! Para resolver esse problema, codificamos informações em grupos de 3 bytes, ou 24 bits, o que gera 4 caracteres na codificação Base64. Se o arquivo tiver um número não múltiplo de 3 de bytes, convertemos os bytes restantes (1 ou 2 bytes), e adicionamos o caractere '**=**' no fim da mensagem Base64. Adicionamos um = se temos 2 bytes no último bloco, ou == se temos 1 byte somente no último bloco do arquivo de entrada. Por exemplo:

Value	Char	Value	Char	Value	Char	Value	Char
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Tabela 1: Tabela de conversão de binário para Base64.

```
Input: any carnal pleasure.  Output: YW55IGNhcm5hbCBwbGVhc3VyZS4=
Input: any carnal pleasure   Output: YW55IGNhcm5hbCBwbGVhc3VyZQ==
Input: any carnal pleasur    Output: YW55IGNhcm5hbCBwbGVhc3Vy
Input: any carnal pleasu     Output: YW55IGNhcm5hbCBwbGVhc3U=
Input: any carnal pleas      Output: YW55IGNhcm5hbCBwbGVhcw==
```

É importante notar que, como a divisão dos bytes não coincide com a divisão das unidades do Base64, o mesmo caractere pode ser codificado de formas diferentes. Por exemplo:

```
The Input: pleasure.   Encodes to cGxlYXN1cmUu
The Input:  leasure.   Encodes to bGVhc3VyZS4=
The Input:   easure.   Encodes to ZWFzdXJlLg==
The Input:    asure.   Encodes to     YXN1cmUu
The Input:     sure.   Encodes to     c3VyZS4=
```

Um último detalhe do Base64 é que ele foi desenvolvido na época dos terminais burros, que tinham telas de 25 linhas de 80 caracteres. Para que o arquivo fosse representado de forma "bonitinha" na tela, as linhas eram quebradas após o 64º caractere, seguidas pelos caracteres de nova linha e retorno de carro (CR+LF), representadas no C por '\r' e '\n'[1], respectivamente.

Assim, o texto abaixo seria codificado da seguinte forma em Base64:

[1] Para os mais piolhos de computador: no DOS e no Windows, arquivos de texto terminam com CR+LF, enquanto tradicionalmente no Unix esses arquivos terminam só com LF. Os editores mais recentes fazem essa distinção automaticamente, mas antigamente era necessário rodar programas para "converter" um arquivo DOS para Unix e Unix para DOS.

"*Man is distinguished, not only by his reason, but by this singular passion from other animals, which is a lust of the mind, that by a perseverance of delight in the continued and indefatigable generation of knowledge, exceeds the short vehemence of any carnal pleasure.*"

Sua representação em Base64 (fonte menor para preservar a quebra de linha, não mostrando o CR+LF no fim):

```
TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWFzb24sIGJ1dCBieSB0aGlz
IHNpbmd1bGFyIHBhc3Npb24gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaCBpcyBhIGx1c3Qgb2Yg
dGhlIG1pbmQsIHRoYXQgYnkgYSBwZXJzZXZlcmFuY2Ugb2YgZGVsaWdodCBpbiB0aGUgY29udGlu
dWVkIGFuZCBpbmRlZmF0aWdhYmxlIGdlbmVyYXRpb24gb2Yga25vd2xlZGdlLCBleGNlZWRzIHRo
ZSBzaG9ydCB2ZWhlbWVuY2Ugb2YgYW55IGNhcm5hbCBwbGVhc3VyZS4=\
```

2. Programa a ser desenvolvido

Neste trabalho prático iremos desenvolver dois programas, em C, chamados de "encode" e de "decode". O encode receberá dois parâmetros **pela linha de comando**: primeiro, o nome do arquivo binário a ser codificado, e em seguida o nome do arquivo de saída Base64. O decode terá a mesma forma, passando de Base64 para binário. Não é necessário que os programas imprimam nada na tela. Uma execução dos dois programas seria a seguinte:

```
[~/] encode arquivo.pdf arquivo.b64

[~/] decode arquivo.b64 arquivo.pdf
```

3. Entrega do código

O código deve ser entregue em um arquivo Zip (não pode ser RAR nem .tgz nem .tar.gz) no Moodle, contendo um arquivo **readme.txt** com o nome dos integrantes. Incluam todos os arquivos (.c, .h, makefile, não incluam executáveis ou arquivos objeto) EM UM ÚNICO DIRETÓRIO. Um makefile deve ser fornecido para a compilação do código. Este makefile, quando executado sem parâmetros, irá gerar os dois programas, encode e decode, EXATAMENTE com esses nomes. Submissões onde os programas não seguem as especificações de parâmetros e nomes, makefiles não funcionando ou arquivos necessários faltando não serão corrigidos. O julgamento do trabalho será feito em Linux.

Os programas desenvolvidos deverão rodar em um computador **Linux**. Neste primeiro trabalho isso **não deve ser** um problema, mas sugiro fortemente a execução em um computador Linux mesmo para o TP0 (os trabalhos subsequentes DEVERÃO executar em Linux, e ocorrem diferenças em plataformas diferentes quando executamos programas de rede).

4. Documentação

A documentação deve ser breve, descrevendo as decisões de implementação mais importantes e o funcionamento do código, de forma que o professor possa entender o que foi feito no código sem ter que entender linha a linha dos arquivos. Implementações modularizadas deverão mencionar quais funções são implementadas em cada módulo ou classe.

Como o Base64 é muito importante em redes, existem diversas implementações prontas do mesmo na Internet, de forma que a documentação deve mostrar da melhor forma possível que foram os alunos que escreveram o código, que ele não foi baixado da Internet. Tal tipo de cópia é fácil de ser identificado por ferramentas automáticas (eu vou rodá-las!), e porque os códigos da Internet são, em geral, altamente otimizados, o que os tornam visivelmente diferentes do código de um aluno de graduação típico). No caso de entregas de códigos que levantem suspeita, irei fazer uma entrevista com os membros do grupo para ver se os mesmos conseguem explicar como o código escrito funciona.

Apresentem exemplos de execução dos dois programas, mostrando a linha de comando a ser executada, um arquivo de entrada simples e a sua saída.

A documentação será entregue impressa, na secretaria do DCC, até as 17:00 da data final de entrega, ou em sala de aula. Trabalhos que forem entregues no Moodle mas sem a documentação impressa **não serão** corrigidos.

5. Desconto de nota por atrasos

Trabalhos entregues com atraso sofrerão as seguintes penalidades:

- **Um dia de atraso**: correção valendo somente 50% da nota.
- **Dois dias de atraso**: correção valendo somente 25% da nota.
- **Atrasos superiores a dois dias**: o trabalho não será corrigido.

6. Referências

Base 64:

- RFC 4648: http://tools.ietf.org/html/rfc4648
- Wikipedia: http://en.wikipedia.org/wiki/Base64

Programação em C:

- http://www.dcc.ufla.br/~giacomin/Textos/tutorialc.pdf
- ftp://ftp.unicamp.br/pub/apoio/treinamentos/linguagens/**c**.pdf
- http://www.cs.cf.ac.uk/Dave/C/CE.html
- http://www2.its.strath.ac.uk/courses/c/
- http://www.lysator.liu.se/c/bwk-tutor.html
-

Uso do programa make e escrita de Makefiles:

- http://comp.ist.utl.pt/ec-aed/PDFs/make.pdf
- http://haxent.com.br/people/ruda/make.html
- http://informatica.hsw.uol.com.br/programacao-em-c16.htm
- http://www.gnu.org/software/make/manual/make.html
- http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/