Universidade Federal de Minas Gerais
DCC023: Redes de Computadores
Trabalho Prático

Introdução

Neste trabalho iremos desenvolver duas aplicações. Uma aplicação cliente que se conecta a um servidor e envia um *string* de texto criptografado, e uma aplicação servidora que recebe *strings* de clientes e retorna uma versão decodificada do mesmo. No texto abaixo, nomes de funções da biblioteca padrão relativas à tarefa sendo descrita estão referenciadas entre colchetes para facilitar o desenvolvimento do trabalho.

Objetivos

- Introduzir a interface de programação de soquetes POSIX.
- Introduzir os conceitos de aplicação cliente e aplicação servidor.
- Introduzir os conceitos de codificação e transmissão de dados.

Execução

- O trabalho é individual e vale 5 pontos.
- A data de entrega está disponível no Moodle ou no Plano de Curso.

Especificação

Cifra de César

Neste trabalho vamos usar um tipo de codificação simples para strings de texto denominado Cifra de César. O seu princípio de operação é escolher um certo inteiro X e trocar cada caractere do string pelo caractere X posições à frente no alfabeto. Considere que a letra [a] está à frente da letra [z]. Na prática, essa codificação é muito fácil de se quebrar, mas ilustra bem o princípio usado por algoritmos de criptografia. Na Internet, em grupos de discussão, uma versão muito popular usa X = 13 e é chamada de Rot13. Ela costuma ser usada para esconder spoilers ou a resposta de um quebra-cabeça, por exemplo.

Programa Cliente

O programa cliente receberá como parâmetros de linha de comando: o endereço IP da máquina onde o servidor estará executando, o número do porto em que ele estará esperando conexões, um *string* e um inteiro sem sinal. O *string* deve conter apenas os caracteres de [a] até [z], minúsculos e sem acento. (Note que o *string* não deve conter espaços, nem caracteres [\n] e [\r] de quebra de linha.) Ao executar, ele irá se conectar ao servidor [socket, bind, connect]. Após estabelecimento da conexão, o cliente irá enviar um inteiro de quatro bytes em network byte order [send, htonl/pack] indicando o tamanho do *string.* Em seguida, o cliente deve enviar o versão do string de entrada codificado usando a cifra de César (se você estiver usando C, note

que o caractere de terminação [\0] não deve ser enviado). Imediatamente após o *string*, o cliente deve enviar o valor de X como um outro inteiro de quatro bytes, também em network byte order [send, htonl/pack].

Após o envio da requisição, o cliente irá esperar do servidor um *string* de caracteres ASCII [recv]. O string recebido contém o mesmo número de caracteres do *string* enviado, e também contém apenas caracteres entre a e z, minúsculos e sem acento. Após recebimento do *string* do servidor, o cliente deve imprimi-lo na tela e fechar a conexão com o servidor [printf, close].

Programa Servidor

O programa servidor receberá como parâmetro de linha de comando o número do porto em que ele deve aguardar por conexões dos clientes [socket, bind, listen, accept]. Após o estabelecimento de uma conexão, o servidor deverá receber um inteiro de quatro bytes, network byte order [recv, ntohl/unpack] que indicará o tamanho do string que deverá ser lido em seguida [recv]. Após o recebimento do string, o servidor deverá receber outro inteiro de quatro bytes, network byte order [recv, ntohl/unpack], com o valor de X.

O servidor deve então decodificar o string, escrevê-lo na saída [print/printf] e enviá-lo decodificado de volta para o cliente. Depois disso, o servidor deve fechar a conexão [close].

Detalhes de Implementação

- O string deve ser tratado como código ASCII. Nesse caso, os caracteres entre [a] e [z] têm código ASCII entre 97 e 122.
- Clientes e servidores devem configurar um temporizador (*timeout*) de 15 segundos para detectar falhas de comunicação ao chamar a função [recv]. A configuração de temporizador é feita chamando-se a função [setsockopt]. No Linux, as opções disponíveis para uso na função [setsockopt] são descritas na seção [socket] do manual 7 (acesse usando [man 7 socket]). Procure por [SO_RCVTIMEO].
- Servidores e clientes devem imprimir cada um apenas uma linha de texto, como especificado acima, como resultado de cada conexão.
- O servidor deve conseguir atender múltiplos clientes simultaneamente (você deve utilizar threads [pthread/Threading] ou a função [select]).
- Sugestão. Para testar o correto funcionamento do seu cliente ou servidor, teste seu programa com o programa complementar de outros colegas. (Não é preciso, nem permitido, compartilhar código para fazer esse teste.)

Entrega e Avaliação

Você deve entregar o código fonte do seu programa. Seu programa será testado semi-automaticamente com as implementações de referência do cliente e do servidor implementados pelos professores.

- Você deverá entregar *apenas* dois arquivos chamados [cliente.c] e [servidor.c], ou [cliente.py] e [servidor.py].
- Caso opte por Python, use Python 3 (note que em Python 3 as strings são representadas em Unicode por default, então pode ser preciso transformar de/para ASCII em certos pontos).
- Atente para a forma como seu programa recebe os parâmetros da linha de comando e o que ele escreve na saída. Programas que não sigam o padrão descrito neste enunciado falharão na avaliação automática e serão penalizados na nota.

Testes

O professor irá disponibilizar *scripts* de teste no Moodle. O cliente deve ser executado na linha de comando recebendo os parâmetros mencionados. Por exemplo:

```
$ ./cliente 127.0.0.1 5000 stringdeteste 1
```

O servidor deve executar continuamente e deve requerer o número do porto como parâmetro:

```
$ ./servidor 5000
```