Universidade Federal de Minas Gerais
DCC023: Redes de Computadores - Turma TW (BSI) - 2019-1
Trabalho Prático 0

Introdução

Neste trabalho iremos desenvolver duas aplicações: um cliente e um servidor que se conectam para trocar algumas informações simples. O servidor implementará um contador global que pode ser atualizado pelos clientes. Neste texto, nomes de funções 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 em Python.

Execução

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

Especificação

Contador global

Neste trabalho vamos implementar um serviço simples de armazenamento de um valor global gerenciado por um servidor. O programa servidor fica à disposição de clientes para responder comandos sobre o valor armazenado. Esses valores será um inteiro positivo entre 0 e 999.999, que será sempre manipulado em módulo 1.000.000 - isto é , se o resultado da operação com o contador for um valor igual ou superior a 1.000.000, ele será representado apenas pelas seis casas decimais menos significativas. Além disso, operações que gerem valores negativos devem ser complementadas em base 1.000.000 - isto é, -1 se torna 999.999, -9 se torna 999.991, etc.

Protocolo de comunicação

O protocolo entre cliente e servidor será baseado em conexões e terá apenas dois tipos de mensagens:

- Mensagens do cliente para o servidor iniciam-se com um byte, cujo valor indica a operação a ser executada: zero indica subtração, 1 indica adição. Esse byte deve ser

seguido por um inteiro de 4 bytes codificado em *network byte order* [pack/unpack] contendo o valor a ser adicionado ou subtraído ao contador global.
- Mensagens do servidor para o cliente serão apenas o valor reportado do contador global após a execução de um comando. Esse valor será sempre representado como um string (em formato ASCII) com seis caracteres numéricos, de 000000 a 999999. (Note que o *string* não deve conter espaços, nem caracteres [\n] e [\r] de quebra de linha.)

Formato dos dados

Manipular bytes em uma mensagem de rede, para um protocolo simples como esse, é a única atividade que costuma ser mais trabalhosa em Python que em C/C++. Recomendo que todos verifiquem as funções pack/unpack para converter valores em Python para um formato binário específico. Além disso, é importante notar que Python 2.x processa strings em diretamente em ASCII, enquanto Python 3.x usa Unicode. Neste último caso, convém verificar também o uso das funções encode/decode. (https://docs.python.org/3.3/howto/unicode.html)

(Se você ainda não teve que lidar com esse tipo de detalhe, convém também ler "The Absolute Minimum Every Software Developer Absolutely Positively Must Know About Unicode and Character Sets".)

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]. Ele deve então executar um loop aceitando conexões e processando todos os comandos de uma conexão, até que o cliente feche essa conexão. Nesse momento ele deverá voltar a aceitar novas conexões. Segundo esse comportamento, a única forma de terminar o programa servidor será por um sinal de terminação, como um Control-C. O servidor não precisa exibir nenhuma mensagem durante sua operação.

Programa Cliente

O programa cliente receberá como parâmetros de linha de comando o endereço IP da máquina onde o servidor estará executando e o número do porto em que ele estará esperando conexões. Ele deve então se conectar ao servidor e passar a aceitar comandos pela entrada padrão, formatar esses comandos como mensagens para o servidor, enviar essas mensagens, receber cada resposta e exibi-la na saída.

Comandos serão fornecidos como um caracteres "+" ou "-", seguido de um espaço, seguido de um inteiro positivo entre 0 e 999.999. O programa cliente pode assumir que não há erros na digitação dos comandos. Ao encontrar o indicador de fim de arquivo, o cliente deve fechar a conexão e terminar sua execução.

Detalhes de Implementação

- Os programas devem ser implementados em Python 2 ou 3.
- 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].)
- Clientes devem imprimir apenas uma linha contendo o resultado de cada operação, como especificado acima, sem outras mensagens. Isso será importante para garantir a correção automática.
- Sugestão: Para testar o correto funcionamento do seu cliente ou servidor, teste seu programa com o programa complementar de outros colegas. (**Não é permitido compartilhar código para fazer esse teste**.)

Entrega e Avaliação

Você deve entregar apenas o código fonte dos seus programas (nenhum relatório) em um arquivo .zip ou .tar.gz. Eles serão testados semi-automaticamente com as implementações de referência do cliente e do servidor implementados pelo monitor/professor.

- Você deverá entregar ***apenas*** dois arquivos chamados [cliente.py] e [servidor.py].
- 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.