DOCUMENTAÇÃO TP1 - REDES DE COMPUTADORES

Bárbara D'Avila Duarte – 2013062707
Guilherme Viegas de Faria – 2013

Considerações prévias

O trabalho teve seus testes finais realizado em um computador pessoal com as seguintes configurações:

Processador: Intel®Core™i7-4500U CPU @ 1.80GHz 2.4GHz
Intel®Core™i5-5257U CPU @ 2.70GHz 3.1GHz
Memória Instalada (RAM): 15,9 GB
8 GB 1867 MHz DDR3
Tipo de Sistema: Sistema Operacional de 64 bits, processador com base em x64
Sistema Operacional: Windows 10 Home Single Language
Apple macOS Sierra 10.12.4

Foi utilizado para a solução do problema a linguagem Python na versão 2.7.6.

Introdução

Este trabalho consiste em realizar os conceitos de comunicação bidirecional em sockets para os alunos da disciplina de Redes de Computadores, a fim de fazer duas entidades se comunicarem e trocarem dados ao mesmo tempo.

Sendo mais específicos, a ideia principal é fazer com que duas entidades se comuniquem entre si e troquem informações da seguinte maneira: cada entidade recebe um arquivo de entrada, esse arquivo de entrada deve ser enviado para a outra entidade, de acordo com um protocolo previamente definido, ao receber estes dados a entidade destino deve validar os dados recebidos, mais uma vez de acordo com o protocolo pré-estabelecido, e imprimí-los num arquivo de saída.

Ao final espera-se que a o arquivo de saída da primeira entidade seja igual ao arquivo de entrada da segunda entidade e o arquivo de saída da segunda entidade seja igual ao arquivo de entrada da primeira entidade.

Modelagem

A modelagem de solução do problema se baseia no conceito de programação em sockets com protocolo TCP/IP, onde há conexão entre as duas entidades. Para este problema onde o mesmo programa deve funcionar como "Cliente" e "Servidor" e também como, "Transmissor" e "Receptor" da mensagem deve verificar-se quem abre a conexão quem está transmitindo e quem está recebendo.

Para simplificar toda a modelagem focamos nossa solução na recepção de dados. Ou seja, o programa reage dado aquilo que ele recebe. Para exemplificar segue o pseudo código utilizado de base para a resolução do problema:

Abrir conexão

Montar o primeiro pacote (Independente se o arquivo de entrada está ou não vazio);
Enviar o primeiro pacote;
Enquanto não terminou de receber todos os dados ou não terminou de enviar todos os dados:
| **Receber** dados;

| **Se** ocorreu falha na comunicação:
| | **Enviar** último pacote de dados;
| **Se não** contínua execução habitual**:**
| | **Se** o flag não é de END:
| | | **Se** é AKC (confirmação de recebimento de dados):
| | | | **Se** checksum está correto:
| | | | | **Se** é um novo ACK (o ID do ACK é diferente do ID do ACK anterior):
| | | | | | **Montar** o próximo pacote;
| | | | | | **Enviar** o próximo pacote;
| | | | | | **Se** o pacote enviado era o último:
| | | | | | | **Receber** o último ACK;
| | | | | | | **Encerrar** o envio de dados (variável);
| | | | | | | **Se** já encerrou o envio de ACK:
| | | | | | | | **Encerrar** a conexão;
| | | | | | | **Inverte** o ID;
| | | | | **Se** é um ACK antigo (de um pacote já confirmado)
| | | | | | **Enviar** último pacote de dados;
| | | **Se** é DADO:
| | | | **Se** ocorreu falha na comunicação:
| | | | | **Enviar** último pacote de dados;
| | | | **Se não** contínua execução habitual**:**
| | | | | **Se** checksum está correto:
| | | | | | **Se** é um DADO repetido: (o ID do pacote de dado = ao ID do último pacote de dado):
| | | | | | | **Enviar** o ACK de confirmação (com ID = ID do pacote de dado);
| | | | | | **Se** é um DADO novo:
| | | | | | | **Gravar** o dado no arquivo de saída;
| | | | | | | **Enviar** ACK de confirmação (com ID = ID do pacote de dado);
| | **Se** o flag é de END:
| | | **Se** ocorreu falha na comunicação:
| | | | **Enviar** último pacote de dados;
| | | **Se não** contínua execução habitual**:**
| | | | **Se** o pacote tem DADOS:
| | | | | **Se** é um DADO repetido: (o ID do pacote de dado = ao ID do último pacote de dado):
| | | | | | **Enviar** o ACK de confirmação (com ID = ID do pacote de dado);
| | | | | | **Encerrar** o envio de ACK (variável);
| | | | | | **Se** já encerrou o envio de dados:
| | | | | | | **Encerrar** a conexão;
| | | | | **Se** é um DADO novo:
| | | | | | **Gravar** o dado no arquivo de saída;
| | | | | | **Enviar** ACK de confirmação (com ID = ID do pacote de dado);
| | | | | | **Encerrar** envio de ACK (variavel);
| | | | | | **Se** já encerrou o envio de dados:

| | | | | | | | **Encerrar** a conexão;
| | | | | **Se** o pacote não tem DADOS:
| | | | | | **Enviar** ACK de confirmação (com ID = ID do pacote de dado);
| | | | | | **Encerrar** envio de ACK (variavel);
| | | | | | **Se** já encerrou o envio de dados:
| | | | | | | **Encerrar** a conexão;

Solução Proposta

A proposta de solução deste problema é separada em um arquivo/programa principal: “dcc023c02.py”. Para este documento veremos portanto com mais detalhes a solução utilizada para este programa conforme descrito na modelagem. A solução portanto, pode ser separa em 8 tópicos relacionados e detalhados a seguir:

1. Comunicação:

A comunicação das entidades é estruturada para que o programa comunique “com ele mesmo”, ou seja, o programa funciona como SERVIDOR e também com o CLIENTE, de acordo com a passagem de parâmetro informada na chamada do programa. Além disso, para que a comunicação ocorra, também de maneira parametrizável, foi definido que o servidor deverá escutar no endereço IP fornecido na passagem de parâmetro na chamada do programa e no porto, também passado por parâmetro, mas restringido entre os números 51000 e 55000.

2. Enquadramento:

O enquadramento foi feito da seguinte forma:

- Primeiramente a chave SYNC, um hexadecimal, foi "pakeado" para um inteiro sem sinal (I) na representação de rede (*big-endian* '>'). (4bytes, 32bits)
- Posteriormente o tamanho LENGTH da mensagem foi transformado em um inteiro hexadecimal e enviado pela rede como um *unsigned short* (>H). (2bytes, 16bits)
- O campo ID foi transformado em um *unsigned char* (>B). (1byte, 8bits)
- Cada possibilidade de flab foi representada como string equivalente a sequência binária que posteriormente foi transformado em um inteiro binário (int(string,2)) e enviado como um *unsigned char* pela rede .
- Por fim, foi calculado checksum de todo pacote, cabeçalho (SYNC, SYNC, checksum(0), LENGTH, ID, FLAG) mais mensagem. Checksum "pakeado", como um *unsigned short,* e acoplado ao cabeçalho do pacote.

3. Nó Transmissor:

Como a modelagem é baseada na recepção de informações, o nó transmissor inicia quando o programa recebe um ACK. Ao receber um ACK o programa é acionado para montar o próximo pacote de dados.

Após a montagem dos dados o pacote é enviado para o programa receptor até que a recepção do ACK de confirmação do pacote , o ACK que possui o mesmo ID do pacote, é recebido. A cada segundo que o ACK não é recebido pelo programa transmissor o quadro de dados é retransmitido.

4. Retransmissão:

O terceiro desafio encontrado foi a retransmissão de dados, dado que a modelagem se baseia na recepção, é necessário verificar a recepção ou não do ACK para retransmitir o dado. Este desafio foi contornado acionando um *timer* na recepção de um quadro, sempre que o programa fica mais de 1 segundo sem receber informações, AKC ou dados, ele reenvia o último pacote de dados afim de restabelecer a comunicação com seu par.

5. Nó Receptor:

O nó receptor é o mais simples dado à modelagem proposta. Ao receber a mensagem o nó envia o ACK de confirmação do pacote e grava o dado recebido no arquivo de saída. O nó receptor não se preocupa se o ACK de confirmação foi corretamente recebido no outro extremo. Caso um ACK se perca, o último pacote será transmitido e o nó receptor tornará a reenviar um ACK de confirmação.

6. Encerramento da conexão:

Para indicar o fim da transmissão e recepção de dados foram criadas duas variáveis booleanas de controle: end_receber_ack e end_receber_dados.

Essas variáveis tem atribuição "False" ao longo de grande parte do programa.

End_receber_dados tem o valor "True" atribuído quando recebe o ACK do último pacote de dados enviado. Indicando assim que esta entidade da conexão já encerrou o envio de todos os dados.

Já para end_receber_ack ter o valor "True" atribuído é necessário que um pacote com flag "END" seja recebido, o pacote pode ter dado ou não. Após a recepção de tal pacote, um ACK de confirmação do pacote de dado recebido é enviado, último ACK que o programa enviará, e o valor "True" é atribuído à variável end_receber_ack.

No momento em que as duas variáveis de controle possuem valor "True" o programa encerra a conexão das entidades e encerra em seguida.

7. Execução do código:

Deve-se chamar, por linha de comando, o programa "DCC023C02" como descrito abaixo:

python dcc023c02.py arg

Onde arg possui os comando à seguir:

Para criar a entidade SERVIDOR:

-s <PORTO> <ARQUIVO DE ENTRADA> <ARQUIVO DE SAÍDA>

Para criar a entidade CLIENTE:

-c <IP>:<PORTO> <ARQUIVO DE ENTRADA> <ARQUIVO DE SAÍDA>

8. Testes realizados:

Primeiramente foram realizados testes simulando o truncamento de um pacote. Nessas circunstâncias o código se apresentou robusto o suficiente para verificar e descartar pacotes com erro. Posteriormente foi criadas versões modificadas do programa a fim de verificar o que ocorreria, por exemplo, caso um pacote se perdesse ou demora-se demasiado tempo para ir de um nó a outro. Novamente, o código mostrou-se bastante robusto para reenviar a mensagem nesses casos. O código modificado funcionava da seguinte forma: o programa era pausado (*time.sleep()*) de tempos em tempos, aleatoriamente.

Conclusões

O objetivo era a implementação, utilizando linguagem Python, de uma entidade bidirecional que se comunicava "com ela mesma" a fim de que os arquivos de entrada das entidades se tornassem os arquivos de saída da entidade na outra ponta da comunicação.

O objetivo foi alcançado após algumas tentativas. As maiores dificuldades encontradas foram a própria modelagem da solução do problema, o enquadramento das informações do cabeçalho conforme o protocolo e a retransmissão de dados ao não receber o ACK de confirmação, a solução destes problema estão descritos em detalhes nos tópicos "MODELAGEM" e "SOLUÇÃO PROPOSTA" deste documento.

Apesar das dificuldades encontradas, todos os problemas foram corrigidos e o programa está executando conforme esperado.