UNIVERSIDADE FEDERAL DE MINAS GERAIS
NOME: Eduardo Henrique Dias Melgaço
MATRÍCULA: 2017023501

Redes de Computadores
Trabalho Prático 2 - Documentação

Objetivo

O objetivo deste trabalho prático é implementar um servidor que armazena arquivos enviados pelo cliente. A comunicação é feita em duas etapas, na primeira dela o servidor e o cliente trocam informações necessárias para o estabelecimento da conexão do cliente e informações do arquivo a ser enviado. Na segunda fase, o cliente fragmenta o arquivo em pedaços de até 1024 bytes, e envia para o servidor. Quando o servidor recebe todo o arquivo, o cliente encerra sua comunicação.

Detalhes de Implementação

O projeto foi criado utilizando Python e a biblioteca padrão socket, para executá-lo devemos primeiro instanciar o servidor e por fim o cliente. Abaixo é exibido como rodar o projeto:

```
# Execução do servidor
python servidor.py 51511
# Execução de um cliente
python cliente.py 127.0.0.1 51511 img-exemplo.jpg
```

Para executar o servidor, precisamos passar uma porta por parâmetro, que será será alocada para um socket que se comunica com TCP. Já a execução do cliente, precisa de 3 parâmetros: o endereço de IP, a porta do servidor que será utilizada para conexão e o nome do arquivo a ser enviado. Nas próximas seções, serão discutidos detalhes sobre a implementação de cada um dos 3 módulos criados.

Utils

Neste módulo definimos constantes que são utilizadas pelos dois módulos e um dicionário que mapeia uma chave para o número identificador de uma mensagem. Por fim, definimos 3 funções:

- send_message: Essa função é utilizada para enviar mensagens que utilizam o protocolo TCP. Ela recebe uma mensagem do formato string ou uma lista de mensagens e à converte para uma única string que separa as mensagens com o caractere "|". Essa estratégia foi adotada para enviar várias mensagens utilizando o mesmo send.
- recv_message: Recebe uma string que contém várias mensagens separadas por "|" e retorna uma lista
- is_valid_file: Recebe o nome de um arquivo e valida se este nome atende aos critérios especificados.

Cliente

No começo da execução, o cliente espera o servidor mandar a mensagem init, que indica que a comunicação deve ser estabelecida. A criação desta mensagem adicional foi necessária para que um padrão de comunicação fosse estabelecido. Esse padrão ocorre para todas as mensagens, onde o cliente espera a recepção de uma mensagem para enviar outra. Após o recebimento desta mensagem, o cliente envia as mensagens do tipo "hello" e "info file" e recebe a mensagem do tipo "ok". Quando essa mensagem é recebida, a função "send_packages_sliding_window" é executada.

Essa função utiliza a estratégia de janela deslizante para fazer a transferência de arquivos. Como pode ser visto na implementação da função, o cliente envia X pacotes por UDP ao servidor, onde X é o tamanho da janela. Após o envio de X pacotes, o cliente só envia um novo pacote, quando recebe um "ack" referente a pacotes anteriores do servidor.

O socket TCP do cliente possui um timeout de 1 segundo. Caso o servidor não envie um "ack" em até um segundo, um erro é lançado e o cliente reconhece que houve algum erro no envio de algum pacote da janela atual, então ele começa a reenviar os pacotes desta janela. Por fim, o cliente recebe a mensagem do tipo "fim", o que significa que o arquivo foi enviado com sucesso.

Um detalhe importante de implementação é que quando o cliente termina de enviar todo o conteúdo para o servidor, ele ainda envia pacotes vazios enviados para cada "ack" recebido após o o envio do último pacote de dados.

Servidor

Ao receber a conexão de um cliente, o servidor instancia uma nova thread para lidar com a conexão de forma isolada. Estabelecida a conexão, o servidor envia e recebe mensagens do cliente, usando o protocolo TCP. Após a troca de todas informações do arquivo, uma mensagem do tipo "ok" é enviada e a transferência começa a ser feita. Quando o servidor está pronto para receber arquivos, ele executa a função "create_file_receiver". Essa função cria um socket udp com a porta do cliente e cria uma pasta output. Após a criação do arquivo de escrita, o servidor recebe os arquivos à partir da estratégia de janela deslizante, ou seja, ele recebe X pacotes e após isso envia uma confirmação de pacote para cada pacote recebido posteriormente. Caso o servidor falhe ao enviar uma mensagem do tipo "ack", o ponteiro de escrita do arquivo é resetado para a posição do último ack.

Para determinar quando deve parar de receber blocos, o servidor checa se o tamanho dos pacotes recebidos é maior ou igual ao tamanho total do arquivo. Por fim, quando essa condição é satisfeita, uma mensagem do tipo "fim" é enviada para sinalizar ao cliente que a transferência de arquivos terminou e foi bem sucedida.

Dificuldades e Desafios

O maior desafio de implementação percebido foi implementar uma forma de recuperação em caso de erro de conexão. No começo estava tentando implementar a janela deslizante do tipo go-back-n, porém terminei desenvolvendo uma janela deslizante padrão que apenas reinicia o envio de pacotes quando uma mensagem do tipo "ack" não é enviada. Não consegui usar o comando TC para simular instabilidade de rede, então para simular um possível envio de erro eu lançava uma exceção no envio de uma mensagem de confirmação por parte do servidor. Esse bloco que lança a exceção está comentado e pode ser visto dentro da função "create_file_receiver".

Exemplo de execução

Logs do servidor

```
Connected to:
::ffff:127.0.0.1:61023
Thread Number: 1
Received hello
Received info_file
Receiving package # 0
Receiving package # 1
Receiving package # 2
Receiving package # 3
Sending ack # 0
Receiving package # 4
Sending ack # 1
Receiving package # 5
Sending ack # 2
Receiving package # 6
Sending ack # 3
Receiving package # 7
Sending ack # 4
Receiving package # 8
Sending ack # 5
Receiving package # 9
Sending ack # 6
Receiving package # 10
Sending ack # 7
Receiving package # 11
Sending ack # 8
Receiving package # 12
Sending ack # 9
Receiving package # 13
Sending ack # 10
Receiving package # 14
```

Logs do cliente

```
Waiting for connection response
Received init
Sending hello
Received connection, udp port 51514
Sending file info
Received ok
Sending package # 0
Sending package # 1
Sending package # 2
Sending package # 3
Received ack # 0
Sending package # 4
Received ack # 1
Sending package # 5
Received ack # 2
Sending package # 6
Received ack # 3
Sending package # 7
Received ack # 4
Sending package # 8
Received ack # 5
Sending package # 9
Received ack # 6
Sending package # 10
Received ack # 7
Sending package # 11
Received ack # 8
Sending package # 12
Received ack # 9
Sending package # 13
Received ack # 10
Sending package # 14
Received fim, closing connection...
```