Documentação TP2
Sockets UDP, stop-and-wait

Luciano Otoni Milen [2012079754]

1- Introdução

Neste trabalho prático deve-se implementar dois programas que estabelecem uma conexão entre si via UDP. O servidor cria uma porta disponível que será acessada pelo cliente, que requisita um arquivo a ser transferido pelo servidor, onde há um tamanho de buffer especificado. A transferência deve ser realizada utilizando o método stop-and-wait, onde é assegurado que nenhuma informação será perdida por e que os pacotes serão transmitidos na ordem correta. É interessante notar que o stop-and-wait é o caso mais simples da janela deslizante, onde o tamanho da janela é 1. Após enviar um pacote, o servidor aguarda a resposta do cliente com um ACK. Somente após tal resposta o servidor continua a transmissão de dados. Em casos de *timeout*, por exemplo, o servidor retransmite o último pacote enviado enquanto aguarda um ACK do cliente.

2- Metodologia

Os experimentos foram realizados da seguinte forma:

a) Foi criado um arquivo teste que contém um texto qualquer;
b) O servidor é iniciado em uma porta aleatória (6001, por exemplo) e um tamanho_de_buffer, como 100 por exemplo;
c) O cliente é iniciado com o endereço do host , como localhost por exemplo. Os parâmetros de chamada são hostname , porta , nome_do_arquivo (teste , no caso) e tamanho_de _buffer (100). A conexão então é estabelecida;
d) O servidor recebe o nome_do_arquivo, confirma com um ACK e o abre no diretório em modo de leitura;
e) O cliente recebe o ACK do servidor e aguarda a transmissão dos *buffers* particionados do arquivo, enquanto houverem *bytes* a serem transmitidos;
f) Conforme os *buffers* chegam o cliente confirma o recebimento dos pacotes com um ACK e o número do ID do pacote recebido;
g) A conexão é encerrada quando o servidor envia uma confirmação do tipo FINAL e o cliente confirma um ACK, fechando o *socket*;
h) As estatísticas são impressas na tela e o arquivo de cópia salvo no cliente. É importante notar que o tratamento de falha no recebimento e envio dos pacotes é feito, conforme será descrito a seguir.

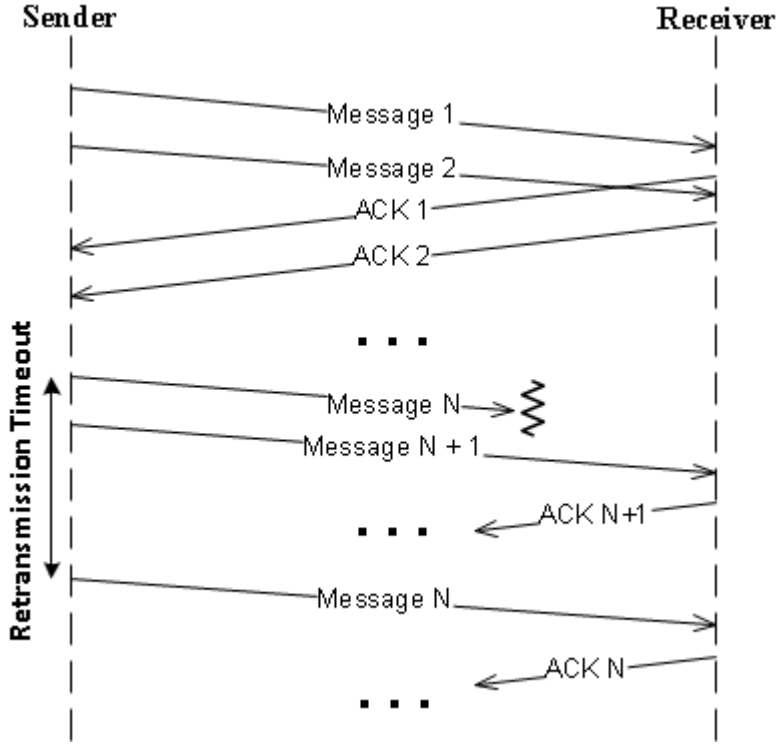
O programa foi elaborado e testado em uma máquina Ubuntu GNOME Linux 17.04 Intel® Core TM i7-3610QM CPU @ 2.30GHz × 8 com 8gb de memória RAM, em uma rede banda larga wireless de 15MB/s de download (2MB/s de upload). As medições foram feitas utilizando o *gettimeofday* para calcular o tempo decorrido na transferência. O teste principal foi executado 15 vezes e diversos testes menores foram feitos durante o desenvolvimento dos programas. O teste principal é composto por um arquivo sem formato específico contendo um texto Lorem Ipsum gerado aleatoriamente com 5 mil *bytes*.

Uma parte importante é a formatação dos pacotes. O esquema segue a tabela a seguir:

PACKET (B)			
`PACKET_ID`	`PACKET_TYPE`	`CHECKSUM`	`PACKET_DATA`
`4`	`1`	`1`	`buffSize`

O código está todo documentado, com o passo a passo do que é feito na execução. Para rodá-lo, basta utilizar o comando *make* no Linux para compilar os arquivos e executá-los na ordem servidor -> cliente, passando informações como hostname , porta, tamanho do buffer , arquivo da transferência. Após a transferência ser completada, um arquivo com o mesmo nome adicionado de "_" é gerado.

Uma breve descrição da estrutura do código: uma biblioteca `common.h` junta as funções de uso comum do servidor e cliente. As funções específicas de cada um estão em seus respectivos arquivos. Além disso, todas importam a `tp_socket.h` fornecida pela especificação do trabalho prático.



Exemplo de comunicação via *stop-and-wait*. [3]

Basicamente a ideia seguida foi a de utilizar uma janela deslizante de tamanho 1 [1]. Desta forma somente 1 buffer pode ser recebido por vez, o que é de fato o procedimento utilizado no *stop-and-wait*.

Os tratamentos de erro são feitos na função `getBuff(char *buffer, int buffSize)`. São 3 casos possíveis:

1. O cliente recebe um pacote que já tinha sido recebido anteriormente, ou seja, o ACK do pacote não chegou no servidor. Neste caso, o cliente retransmite o ACK para o servidor para que a transferência possa continuar.
2. O pacote recebido é do tipo *FINAL_TYPE*. O cliente responde com um pacote do mesmo tipo para encerrar a conexão.
3. O caso esperado, o pacote veio corretamente. O cliente retorna um pacote com ACK para o servidor, incrementando a quantidade de ACKS enviados e o número do próximo *packID* a ser recebido, além de escrever o *buffer* no arquivo de saída.

As funções relacionadas à comunicação via *socket* são feitas através dos arquivos disponibilizados na especificação do trabalho. A função `timer(unsigned int sec)` é responsável pela temporização na transferência de arquivos. Utiliza a função `setsockopt()`[2] que seta no *socket* o valor máximo de segundos para tolerar até emitir um *timeout*.

3- Resultados

Seguem abaixo informações gráficas a respeito do desempenho dos programas. A tabela identifica alguns resultados obtidos nos testes realizados. Cada experimento foi executado 10 vezes e o valor mostrado representa a média obtida.

Experimento	Tamanho Mensagem (B)	Tamanho Buffer (B)	Número de Pacotes	Tempo Decorrido (s)	*Throughtput* (kbps)
1	200	100	5	1.024631	296.66
2	200	600	3	1.017626	391.19
3	200	1600	3	1.008671	394.63
4	1000	100	8	1.085503	1196.42
5	1000	600	4	1.001960	1568.55
6	1000	1600	3	1.000939	1991.98

7	4000	100	43	1.031465	3987.53
8	4000	600	9	1.008237	4584.41
9	4000	1600	5	1.002406	5566.21
10	3000000	100	3013	1.269929	237117.99
11	3000000	600	504	1.032570	292108.97
12	3000000	1600	191	1.036054	292091.95

Experimento 1: no caso mais simples, o resultado com um buffer de 50% do tamanho da mensagem o número de pacotes gerados totalizou em 5.

Experimento 2: tem-se um buffer 6x maior que no experimento 1. O tempo de transferência foi minimamente reduzido.

Experimento 3: neste teste o buffer é 16x maior que no primeiro. O tempo é significativamente menor, e o número de pacotes é igual no segundo teste.

Experimento 7: com uma mensagem 20x maior e um buffer de tamanho igual ao experimento 1, o tempo aumentou, conforme esperado.

Experimento 8: o buffer maior permite que a transferência seja ainda mais rápida.

Experimento 9: o tempo registrado é impressionante, pois mostra pouca alteração em relação ao experimento 3 mesmo tendo uma mensagem muito maior.

Experimento 10: com uma mensagem gigante, o tempo se alterou da mesma forma. É perceptível que o consumo de memória aumenta também.

Experimento 11 e 12: interessante observar que o tempo registrado é virtualmente o mesmo, mesmo o experimento 12 tendo mais que o dobro do buffer do 11.

Os resultados em geral foram conforme esperado. Quanto maior o buffer, mantendo a mensagem em um tamanho fixo, mais rápido a troca de mensagens é feita. Contudo, a relação não é linear. O tamanho da mensagem é bem mais significativo que o tamanho do buffer.

5- Conclusão

Neste trabalho foi possível entender como funciona o protocolo UDP. Sua implementação requer bem mais empenho que o TCP, feito no primeiro exercício. A troca de pacotes deve sempre ter a informação do tipo do pacote, seja ele ACK, DATA_TYPE ou FINAL_TYPE. É a única forma de garantir que a troca de mensagens seja feita na ordem correta.
É um protocolo difícil de implementar. Devem ser feitos tratamentos de erro, como o ACK não chegar a tempo do próximo envio de um pacote pelo servidor. O re-envio de pacotes não está 100%. Percebi que o primeiro pacote estava sempre dando diferença no ID, o que ocasionava no descarte dele e o servidor não re-enviava tal pacote perdido. Contudo, percebi que a transferência é feita de forma correta. O arquivo chega no cliente com todos seus dados. Creio que a falha seja na contabilização de retransmissão de pacotes.
Foi uma ótima oportunidade de aprender como funciona um protocolo tão trivial em redes de computadores. Entretanto, as dificuldades que a própria linguagem traz torna sua implementação consideravelmente complexa.

6- Referências

[1] https://en.wikipedia.org/wiki/Sliding_window_protocol#The_simplest_sliding_window:_stop-and-wait

[2] http://pubs.opengroup.org/onlinepubs/009695399/functions/setsockopt.html

[3] https://www.codeproject.com/KB/IP/UDP_RT/

Slides dos professores Marcos e Loureiro

https://linux.die.net/man/7/socket

https://stackoverflow.com