Universidade Federal de Minas Gerais – Instituto de Ciências Exatas
Departamento de Ciência da Computação

Trabalho Prático Cliente/Servidor UDP

Nome: Júnio Leonardo Soares Salomé

Introdução

Nesta trabalho foi construído dois programas no modelo cliente-servidor, utilizando o protocolo UDP.

Customizou um protocolo de confirmação para garantir a ordem dos dados enviados.

Implementação

O protocolo de confirmação foi implementado enviando metadados sobre a ordem de envio dos pacotes, e configurado um timeout nas opções de inicialização do socket no servidor.

Acknowledge

Após um envio dos dados do servidor, o cliente responde com “ACK” para confirmar o recebimento. Enquanto o servidor não detecte o envio da confirmação, o servidor passa adiante para envio de novos pacotes.

Ordenação

Todos os pacotes são enviados em ordem verificando no cliente se houve a chegada de dados (enquanto isso, o servidor está esperando). Quando o arquivo chegar ao final, o identificador é enviado como uma tag “[END]”.

Temporização

As temporizações foram implementadas usando as opções setsockopt, configurando a struct timeval de acordo com os tempos desejados de timeout.

Compilação

Foi criado dois arquivos “Makefile” para compilar os programas.

Entrar na pasta cliente executar o comando “make” e na pasta servidor e executar comando “make”.

Serão criados os arquivos server e cliente em suas respectivas pastas.

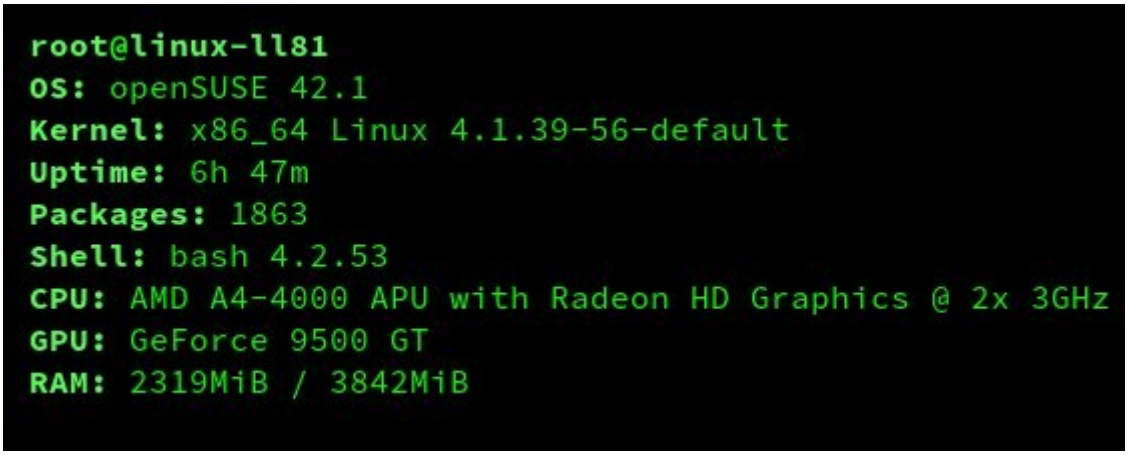
Execução

Após ter compilado os códigos executar os programas nas suas pastas.

./server 8888 512
./cliente 127.0.0.1 8888 nome_arquivo 512

O arquivo a ser baixado tem que estar na pasta onde está o executável do servidor.
Após ser feito o download ou tentativa o programa será finalizado.
Em caso de sucesso o arquivo estará na pasta onde está o executável do cliente.

Metodologia



Tanto o servidor quanto o cliente foram executados no mesmo computador e na mesma rede.

As execuções foram inicializadas simultaneamente utilizando a funcionalidade brodcasting do emulador de terminal, terminator, sendo assim, garantindo o disparo dos dois programas ao mesmo tempo.

Medições

Arquivo 2.1MB

Buffer = 512 byte(s), 8088.51 kbps (2142258 bytes em 2.069109 s)
Buffer = 1024 byte(s), 15533.79 kbps (2142258 bytes em 1.077393 s)
Buffer = 2048 byte(s), 28437.92 kbps (2142258 bytes em 0.588510 s)

Arquivo 23.9MB

Buffer = 512 byte(s), 17509.84 kbps (23916428 bytes em 10.670973 s)
Buffer = 1024 byte(s), 26702.49 kbps (23916428 bytes em 6.997362 s)
Buffer = 2048 byte(s), 54054.84 kbps (23916428 bytes em 3.456619 s)

Arquivo 228.7MB

Buffer = 512 byte(s), 21106.74 kbps (228726901 bytes em 84.661495 s)
Buffer = 1024 byte(s), 40384.89 kbps (228726901 bytes em 44.247439 s)
Buffer = 2048 byte(s), 75986.46 kbps (228726901 bytes em 23.516401 s)

Análises

O aumento do buffer implicou uma diminuição do tempo linearmente. Este resultado era esperado neste ambiente, pois os dados estão sendo enviados serialmente e estando na mesma rede, a chance de pacotes serem perdidos é muito baixa.

No entanto, num caso de rede instável, ter grandes chunks de dados podem ter o efeito inverso.

Outro fator importante, é que a medição de tempo leva em consideração as operações de entrada e saída, tanto do servidor quanto do cliente.

Conclusão

O trabalho foi interessante para entender o uso do UDP, assim como refletir sobre os diferentes casos de uso do protocolo, e formas de como utilizar, configurar e customizar para diferentes fins. Aplicações complexas possuem muitos trade-offs e requerimentos que certamente exigirão este conhecimento para a implementação de sistemas estáveis e de boa performance, importante tanto para os custos quanto para a experiência do usuário final.