Universidade Federal de Minas Gerais - Instituto de Ciências Exatas - ICEx
Departamento de Ciência da Computação - Graduação em Ciência da Computação
2o. Semestre de 2016

Disciplina: Redes de Computadores – DCC023 -60 horas-aula, 4 créditos.

Professores: José Marcos Nogueira e Marcos Vieira
Estagiários em Docência: Alisson e Henrique

Trabalho Prático no. 2

Em grupo de até dois alunos - 02/12/2016 - Valor: 20 pontos

Data de entrega: A DEFINIR

Especificação

Problema

Neste trabalho, você deverá desenvolver dois programas usando o modelo de operação **cliente-servidor**, operando para transmissão unidirecional e no modo **requisição-resposta** sobre o protocolo UDP e a versão 6 do protocolo IP (IPv6), utilizando um protocolo de janela deslizante. A implementação deve utilizar a biblioteca de sockets do Unix (Linux). Recomendamos que o aluno leia atentamente a seção 2.5.2 e as seções de 5.2.1 a 5.2.4 do livro texto do Peterson (5ª edição, 2013, versão português).

Operação

O programa cliente deverá se chamar **clienteUDP** e o programa servidor deverá se chamar **servidorUDP**. O processo de comunicação entre cliente e servidor deverá seguir um padrão semelhante ao do TP1, ilustrado nos quadros a seguir.

O comando para busca de arquivos pelo cliente deverá seguir o seguinte formato:

- clienteUDP <nome ou IPv6 do servidor> <porto> <nome_arquivo ><tam_buffer> <tam_janela>

Para ativar o servidor, deverá ser usado o seguinte comando:

- servidorUDP <porto do servidor> <tam_buffer> <tam_janela> <diretório a ser utilizado>

clienteUDP:

```
        processa argumentos da linha de comando:
        nome_ou_ip_do_servidor porto_servidor nome_arquivo tam_buffer tam_janela

        chama gettimeofday para tempo inicial
        envia string com nome do arquivo (terminada em zero)
        abre arquivo que vai ser gravado - pode ser fopen(nome,"w+")
        loop recv buffer até que perceba que o arquivo acabou
                escreve bytes do buffer no arquivo (fwrite)
                atualiza contagem de bytes recebidos
        fim_loop

        fecha arquivo
        chama gettimeofday para tempo final e calcula tempo gasto
        imprime resultado:
        "Buffer = \%5u byte(s), \%10.2f kbps (\%u bytes em \%3u.\%06u s)
fim_cliente.
```

servidorUDP:

```
        processa argumentos da linha de comando:
        porto_servidor tam_buffer tam_janela dir

        faz abertura passiva e aguarda conexão
        recebe o string com nome do arquivo
        abre arquivo que vai ser lido -- pode ser fopen(nome,"r")
        se deu erro, fecha conexão e termina
        loop lê o arquivo, um buffer por vez até fread retornar zero
                envia o buffer lido
                se quiser, contabiliza bytes enviados
        fim_loop

        fecha arquivo
        chama gettimeofday para tempo final e calcula tempo gasto
        se quiser, imprime nome arquivo e no. de bytes enviados
fim_servidor.
```

Note que neste TP o aluno implementará somente o que seria a função GET do TP1, portanto não é necessário identificar na mensagem qual comando será executado pelo programa.

De forma resumida, o cliente deve enviar um string com o nome do arquivo desejado, receber o arquivo em um buffer e salvar os dados no disco. Quando não houver mais bytes para ler, o cliente fecha o arquivo e gera uma linha com os dados da execução. O servidor por sua vez deve operar de forma complementar.

Como neste trabalho será usado UDP, os segmentos podem se perder ou ser corrompidos. Sua implementação deve lidar com estes erros fazendo a retransmissão dos pacotes. Para simplificar, você pode considerar uma temporização de um segundo. Com esta temporização o seu programa pode ficar bem lento, mas isso não será um problema[1]. Os segmentos também podem chegar fora de ordem, portanto o seu programa deverá ser capaz de identificar a sequência de bytes que está sendo enviada e colocar a sequência na posição correta do buffer.

Definição do protocolo

Neste trabalho, como não teremos TCP para garantir a ordem de recebimento dos pacotes, vocês serão responsáveis por criar um protocolo confiável sobre o UDP usando janela deslizante. Para isso, vocês devem definir quais serão os campos do cabeçalho do seu pacote (que será enviado dentro do pacote UDP) que encapsulará os dados do arquivo.

O protocolo nesse caso será unidirecional, isto é, ele deve ser construído para enviar dados apenas na direção do servidor para o cliente. A primeira mensagem, do cliente para o servidor, não precisa ser mandada com uma janela deslizante, mas precisa ser entregue sem erro (como no stop-and-wait). Mensagens de confirmação podem ser necessárias em diferentes momentos e uma técnica de detecção de erros também deverá ser utilizada, pois erros poderão ocorrer durante os testes. Os cabeçalhos devem ser de tamanho fixo e representados em binário, por ser mais simples e eficiente que usar strings.

Um grande desafio deste trabalho é o tratamento de temporizações. Isso pode ser feito de diferentes maneiras, usando uma temporização associada ao **recvfrom** (man socket) ou usando alarmes e tratadores de sinais com sockets. A primeira é mais simples, mas não é garantida em ambientes que não o Linux e o FreeBSD (Mac).

Código fornecido com este enunciado

No Moodle encontra-se um conjunto de arquivos auxiliares para este trabalho. Esse conjunto contém:

- um exemplo de um programa que utiliza sinais e temporização no seu funcionamento (impaciente.c) que serve de exemplo no uso dessas funções;
- um módulo de encapsulamento das funções da interface de sockets (tp_socket.[ch]) que deve ser usado no trabalho.

Esse módulo de encapsulamento deve ser utilizado no trabalho, no lugar das chamadas às funções da biblioteca de sockets. Na avaliação, seu programa (que deverá usar essas funções) será ligado a uma versão deste módulo que servirá para simular erros no canal de transmissão em certas situações. Programas que sejam desenvolvidos usando as funções da biblioteca diretamente não serão avaliados.

Códigos de erro (retorno dos comandos)

Abaixo é apresentada uma lista inicial dos códigos esperados para as mensagens de erro que deverão ser fornecidas pelos programas.

[1] O aluno poderá fazer a retransmissão adaptativa (seção 5.2.6 do Peterson, 5ª edição, 2013, português). Este método é melhor porém muito mais complexo que um temporizador fixo.

Código de Erro	Descrição do erro
-1	Erros nos argumentos de entrada
-2	Erro de criação de socket
-3	Erro de bind
-4	Erro de listen
-5	Erro de accept
-6	Erro de connect
-7	Erro de comunicação com servidor/cliente
-8	Arquivo solicitado não encontrado
-9	Erro em ponteiro
-10	Comando de clienteUDP não existente
-999	Outros erros (não listados)

O aluno deverá acrescentar, se necessário, outros códigos de erro, que deverão estar documentados no arquivo README e no código fonte. As mensagens de erro deverão ter o seguinte formato: "Erro: %d - Descrição: %s".

Medições de desempenho

Uma vez que os programas estejam funcionando corretamente, deve-se desenvolver uma avaliação do desempenho do par de programas semelhante ao que foi feito no TP1. Deve-se medir o *throughput* da comunicação, isto é, a taxa de transferência obtida entre cliente e servidor (basicamente, o número total de bytes enviados dividido pelo tempo medido no cliente).

As medições devem verificar como o desempenho varia quando diferentes tamanhos de buffer são utilizados e diferentes tamanhos de janelas. Em particular, deve-se **incluir nas medições os casos com mensagens de tamanho de buffer igual a 100, 1.000 e 4.000 bytes**. Outros valores podem ser escolhidos conforme suas observações indicarem a necessidade. Já o tamanho da janela deve ser variado em potências de 2, iniciando em 2 e até que o desempenho atinja o máximo para aquele tamanho de mensagem. Cabe ao aluno determinar quando isso ocorre experimentalmente.

O tamanho do arquivo pode ser escolhido de forma a garantir um tempo de teste nem muito longo nem muito curto. Para testes em que o par de programas executa na mesma máquina, um arquivo de aproximadamente 3 MB pode ser um bom ponto de partida.

Entregáveis

O aluno deverá entregar um **relatório de medições** e o **código fonte** dos programas na linguagem C ou C++. A entrega será eletrônica via Moodle e deve constar de um arquivo .zip, com a máscara **tp2_<nome_do_aluno>_<curso-bcc/eca>.zip** (ex: *tp1_carlos_roberto_eca.zip*)**.** contendo os seguintes documentos:

1. todos os arquivos fonte desenvolvidos (arquivos de terminação .c/.cpp, .h e Makefile);
2. um arquivo denominado README, com uma breve descrição do programa e orientação para compilação e execução. Deve indicar a necessidade de instalação de alguma biblioteca. O arquivo README deverá conter o nome do(s) aluno(s) que fez(fizeram) o TP;
3. uma cópia eletrônica (em .pdf) do relatório, com, **no máximo, 8 páginas**;
4. arquivos contendo os dados coletados, seja como planilhas ou arquivos texto contendo os dados na forma gerada pelos programas.

Não incluir nesse conjunto outros arquivos, como por exemplo, arquivos objeto (.o), arquivos de backups (~* e *~) e os executáveis utilizados! O arquivo zip não deve conter subdiretórios. Todos os arquivos enviados deverão estar no mesmo diretório. Um único Makefile deve ser fornecido para compilar os fontes do cliente e do servidor, que deverão ser gerados no diretório corrente. Não devem ser criados diretórios separados para os fontes, binários, README e relatório.

Relatório de desempenho

O relatório com os resultados das medições de desempenho deverá conter as seguintes seções:

1. **Introdução**: descrição do objetivo do trabalho
2. **Metodologia**: dados sobre os experimentos, como a configuração das máquinas utilizadas e a localização das mesmas na rede. Indicar também como foram feitas as medições, quantas vezes o teste foi executado, se foram execuções diferentes do programa ou apenas um loop ao redor do programa todo para fazer todas as “n” medições de tempo.
3. **Implementação**: cabeçalho do protocolo, detalhes da implementação do protocolo de janela deslizante, decisões de projeto envolvidas, como a técnica utilizada para fazer o enquadramento do arquivo (indicar ao receptor quando o arquivo acaba) e o tratamento de temporizações.
4. **Resultados**: apresentar a informação coletada, tanto na forma de tabelas quando na forma de gráficos. As mesmas observações feitas para o TP1 valem para este relatório. Notem que agora duas grandezas estão sendo variadas: tamanho das mensagens e tamanho da janela.
5. **Análise**: para cada experimento, discutir os resultados observados. Os resultados foram de acordo com o esperado? Você é capaz de explicar por que as curvas se comportam como o fazem? Houve algum elemento claramente de destaque nos resultados que merece uma análise especial (por exemplo, um pico/vale inesperado nos gráficos, desvios muito significativos nas medições)?
6. **Conclusão**: como todo trabalho técnico, algumas palavras finais sobre o resultado do trabalho, tanto das observações quanto do seu aprendizado sobre o assunto.

O relatório não deve incluir a listagem dos programas. Relatórios que apresentem um bom trabalho de análise de diversos cenários poderão receber pontos extras.

Critérios de Avaliação

Os seguintes critérios serão utilizados na avaliação dos programas e do relatório:

- Entregável
  - Padronização do nome do arquivo;
  - Inexistência de subdiretórios no arquivo ZIP
  - Presença do arquivo README
- Programa:
  - Corretude e unicidade do Makefile;

    - Corretude da compilação;
    - Nomeação dos programas;
    - Formato de entrada da linha de comandos;
    - Suporte a IPv6;
    - Corretude no envio de arquivos (se retorna os arquivos diretório);
    - Corretude no comando list (se retorna corretamente os arquivos solicitados);
    - Código de erro incompatível com a relação fornecida nesta especificação;
    - Clareza, indentação e comentários no programa;
    - Linguagens de programação utilizadas .
- Relatório:
    - Formato e tamanho;
    - Aderência das conclusões aos dados apresentados;
    - Utilização dos tamanhos de buffer como solicitados;
    - Clareza e objetividade.

Todos os programas serão avaliados em um ambiente com instalação padrão do sistema operacional UBUNTU 14.04.5, versão desktop com o ambiente de desenvolvimento denominado *build-essential*[2] do Ubuntu. Qualquer ferramenta ou biblioteca que não esteja contida nesta instalação deverá ser fornecida pelo aluno.

Referências

- PETERSON, L. L. & DAVIE, B. S. "Redes de Computadores: uma abordagem de sistemas", Ed. Campus. 2013. ISBN 978-85-352-4897-5. Tradução da Quinta edição em inglês.
- TANENBAUM, A.S. "Redes de Computadores", Campus, ISBN: 8535211853. 5a. edição, 2011.
- KUROSE, J. & ROSS, K. "Redes de Computadores e a Internet: uma nova abordagem". 3ª ed. São Paulo:Addison Wesley, 2006. ISBN 8588639181.
- STALLINGS, W. "*Redes e sistemas de comunicação de dados: teoria e aplicações corporativas*". Elsevier, Rio de Janeiro, 2005 (Tradução 5a. edição). ISBN 85-352-1731-2
- DONAHOO, M. J. AND CALVERT, K. L. "*TCP/IP sockets in C: practical guide for programmers"*. Morgan Kaufmann. 2009
- STEVENS, W. R., FENNER, B & RUDOFF, A. M. 2003. "*UNIX Network Programming"*, Vol. 1 (3 ed.). Pearson Education.
- COMER, D. E.. 1991. "*Internetworking with TCP/IP"* (2nd Ed.), Vol. I. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
- KERRISK, M. 2010. "*The Linux Programming Interface: A Linux and UNIX System Programming Handbook" (1st ed.)*. No Starch Press, San Francisco, CA, USA.
- HOLZMANN, GERARD J. "*Design and Validation of Computer Protocols"*. Prentice Hall, Published November 1991, 512 pages, ISBN 0135399254 (Hardcover), 0135398347 (Paperback).
- STROUSTRUP, BJARNE. "*The C++ Programming Language"*. Addison-Wesley, 2013, 4th Edition.
- https://computing.llnl.gov/tutorials/pthreads/
- http://long.ccaba.upc.edu/long/045Guidelines/eva/ipv6.html
- http://man7.org/linux/man-pages/man3/getaddrinfo.3.html
- Manual do linux (manpages)

[2] http://packages.ubuntu.com/trusty/devel/build-essential