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.

Professor: **José Marcos Nogueira e Marcos Vieira**
Estagiários em Docência: **Alisson e Henrique**

Trabaho Prático no. 1

Em grupo de dois alunos - 05/10/2010 - Valor: 20 pontos

Data de entrega: 29/10/2016

1. Motivação

Suponha que um professor armazene em um repositório, que está em um computador da universidade, arquivos relacionados à sua disciplina, a saber: slides de aulas, livros, listas de exercícios, entre outros. Aproximando-se a data da prova, você gostaria de acessar tais conteúdos para estudar, porém "não existe" uma ferramenta que lhe permite realizar a transferência de arquivos do computador da universidade para o seu computador pessoal. Os arquivos não podem ser copiados diretamente por dispositivos de armazenamento, somente pela rede. O professor, gente fina, permite a você o acesso necessário aos arquivos da máquina da universidade. Entretanto, ele solicita a você, aluno da disciplina de redes de computadores, que desenvolva um programa capaz de acessar o computador da universidade, listar o conteúdo disponível do repositório e realizar o download do arquivo desejado. Em complemento, o professor solicita que sua aplicação, ao transferir arquivos, seja capaz de medir a vazão alcançada na transferência de um arquivo.

2. Especificação

Neste trabalho, você deverá desenvolver dois programas usando o modelo de operação **cliente-servidor**, no modo **requisição-resposta** sobre o protocolo TCP e a versão 6 do protocolo IP (IPv6). O cliente (sua máquina) poderá solicitar do servidor ("máquina da universidade", mas pode ser a sua própria) os arquivos disponíveis para *download*. O cliente também poderá baixar um arquivo do servidor, enviando para este o nome do arquivo desejado.

O processo de comunicação entre cliente e servidor deverá seguir um padrão simples, ilustrado a seguir. O cliente deve se conectar ao servidor, enviar uma cadeia de caracteres (*string*) com o comando desejado e aguardar a resposta.

O cliente tem dois comandos à disposição:

- **list**: solicita, recebe e exibe na tela a lista dos arquivos disponíveis em um diretório para *download* (se não houver arquivos no diretório, imprime uma mensagem de erro); e
- **get**: solicita ao servidor e recebe um arquivo.

O programa cliente deverá se chamar **clienteFTP** e o programa servidor deverá se chamar **servidorFTP**. Os comandos deverão seguir o seguinte formato:

- clienteFTP **list** <nome ou IPv6 do servidor> <porta do servidor> <tam_buffer>
- clienteFTP **get** <nome do arquivo> <nome ou IPv6 do servidor> <porta do servidor> <tam_buffer>

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

- servidorFTP <porta do servidor> <tam_buffer> <diretório a ser utilizado>

Operação detalhada dos comandos list e get

Abaixo são apresentados os fluxogramas (não exaustivos) da operação esperada para os dois comandos no cliente (figura 1) e no servidor (figura 2). Nos fluxogramas não foram identificados todos os pontos de detecção e tratamento do erros que podem acontecer ao operar com *threads*, ponteiros, arquivos e *sockets*.

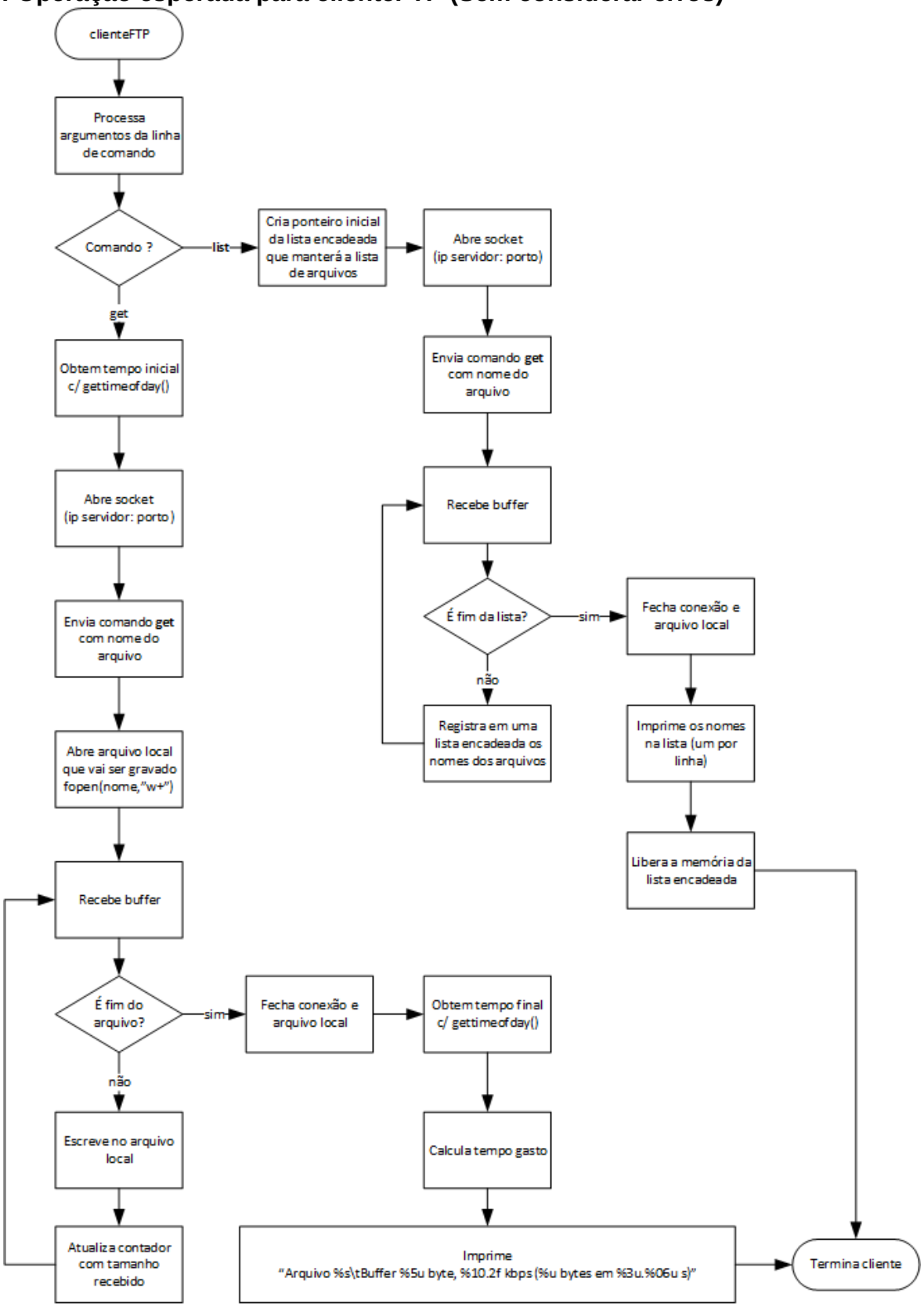
Códigos de erro (retorno dos comandos)

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

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 clienteFTP não existente
-999	Outros erros (não listados)

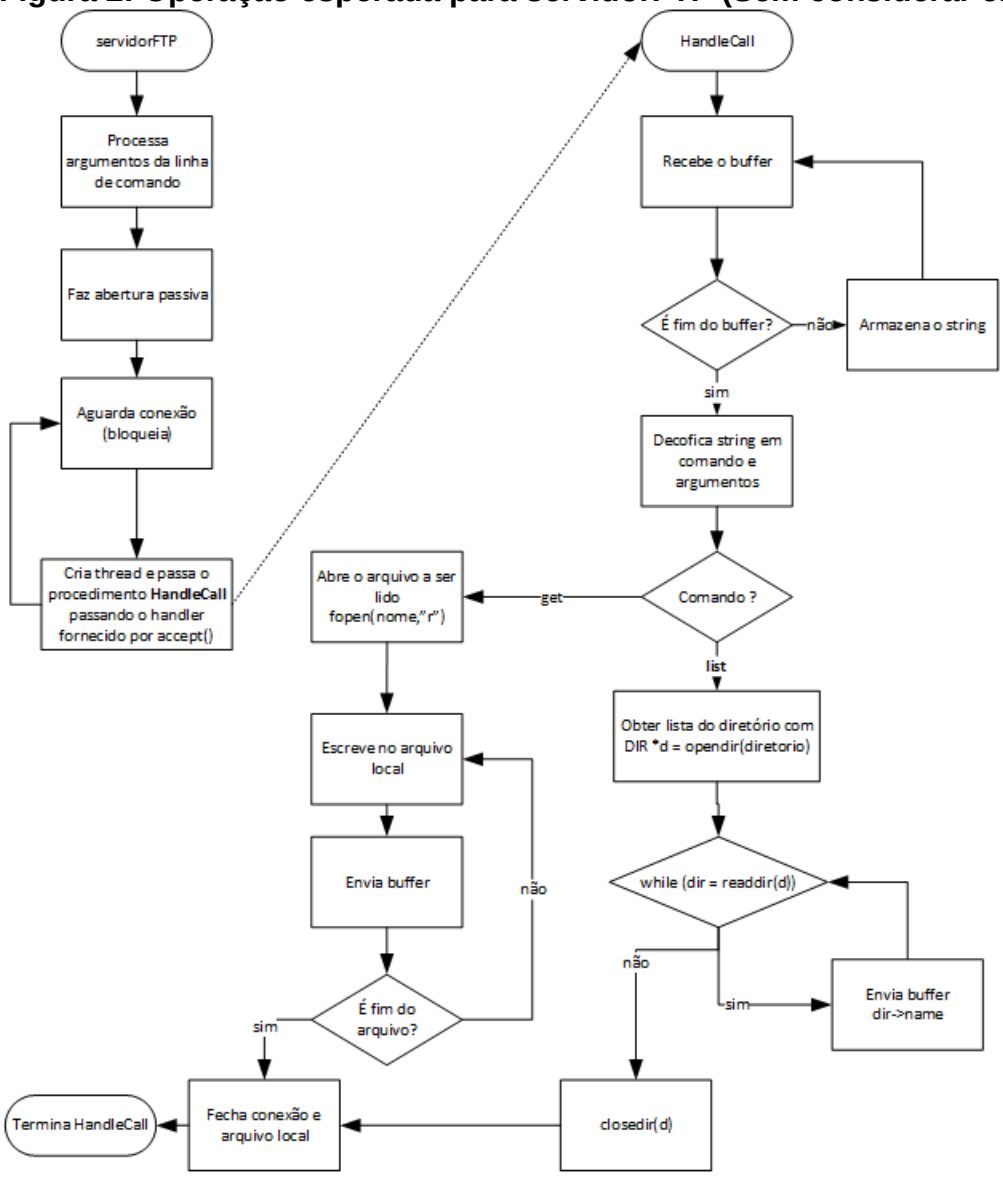
O aluno deverá acrescentar, se necessário, outros códigos de erro, que deverão estar documentados no arquivo README e no código fonte. As mensagens de erro deverão ter o seguinte formato: “Erro: %d - Descrição: %s”.

Figura 1 : Operação esperada para clienteFTP (Sem considerar erros)



O servidor deverá ser capaz de processar **múltiplas conexões simultâneas** e, portanto, ficará rodando até que receba um sinal SIGTERM ou SIGKILL.

Figura 2: Operação esperada para servidorFTP (Sem considerar erros)



Para obtenção da lista de arquivos sugere-se utilizar as funções da biblioteca *dirent.h* (você pode usar #**man 2 opendir** para obter mais informações).

Sockets devem ser utilizados no desenvolvimento do trabalho, utilizando a biblioteca de sys/sockets.h do Linux. Os programas deverão utilizar *sockets* com **IPv6**. Não serão aceitos programas somente com IPv4. Os testes serão rodados em um computador com IPv4 desabilitado.

3. Medições de desempenho

Uma vez que os programas estejam funcionando corretamente, o aluno deve avaliar o desempenho do funcionamento dos programas. Esta avaliação será feita mediante a conexão de **um único cliente** ao servidor. Não devem ser feitas medições com conexões simultâneas de diversos clientes.

O aluno deve medir a taxa de transferência obtida na transferência do arquivo entre o cliente e o servidor, isto é, a vazão (*throughput)* da comunicação, definida como o número total de bytes enviados dividido pelo tempo medido no cliente. As medições devem verificar como a vazão varia

com o tamanho de buffer. Devem ser fornecidas, no mínimo, as medições para os casos nos quais o tamanho do buffer seja igual a 2^i bytes, onde $1 \leq i \leq 16$, $i \in N$. Outros valores podem ser escolhidos conforme suas observações indicarem necessidade. Para a medição do tempo no cliente sugere-se utilizar a função *gettimeofday()* da biblioteca *sys/time.h*.

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 os dois programas executam na mesma máquina, um arquivo de aproximadamente 3 MB pode ser um bom ponto de partida.

4. 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 **tp1_<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;
3. uma cópia eletrônica (em .pdf) do relatório, com, **no máximo, 8 páginas**.

Não incluir nesse conjunto outros arquivos, como por exemplo, arquivos objeto (.o), arquivos de backups (~* e *~) e os executáveis utilizados!

Relatório de desempenho

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

I. **Introdução**: descrição do objetivo do trabalho
II. **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.
III. **Resultados**: apresentar a informação coletada, tanto na forma de tabelas quando na forma de gráficos. Cada tabela deve conter, para o experimento em questão, uma linha para cada tamanho de mensagem, com o número de mensagens enviadas, o tempo total médio medido, o desvio padrão dos tempos medidos, o intervalo de confiança para o valor médio (com significância de 95%) e a banda média observada no experimento. Cada gráfico deve usar escalas adequadas (logarítmicas ou lineares, conforme o caso), os eixos devem ser identificados e possuir claramente a identificação das unidades em cada um. Os resultados devem ser apresentados por linhas retas interligando os pontos medidos, que devem ser destacados com marcas claras. Se um gráfico contiver mais do que um conjunto de pontos, as linhas devem ser claramente identificadas com marcações, cores e legenda. Se possível, cada gráfico deve incluir barras verticais indicando a variância de cada valor calculado.
IV. **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)?
V. **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. O relatório deverá ser feito utilizando o **formato padronizado da SBC** para publicação de artigos que pode ser obtido em http://www.sbc.org.br/documentos-da-sbc/category/169-templates-para-artigos-e-capitulos-de-livros.

5. Critérios de Avaliação

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

1) Programa:
   a) Corretude do Makefile;
   b) Corretude da compilação;
   c) Cliente e/ou servidor sem permitir múltiplas conexões no servidor;
   d) Nomeação dos programas;
   e) Formato de entrada da linha de comandos;
   f) Suporte a IPv6;
   g) Corretude no envio de arquivos (se retorna os arquivos diretório);
   h) Corretude no comando **list** (se retorna corretamente os arquivos solicitados);
   i) Código de erro incompatível com a relação fornecida nesta especificação;
   j) Clareza, indentação e comentários no programa;
   k) Linguagens de programação utilizadas .

2) Relatório:
   a) Formato e tamanho;
   b) Aderência das conclusões aos dados apresentados;
   c) Utilização dos tamanhos de buffer como solicitados;
   d) 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*[1] do Ubuntu. Qualquer ferramenta ou biblioteca que não esteja contida nesta instalação deverá ser fornecida pelo aluno.

Referências

- Donahoo, M. J. and Calvert, K. L. ***TCP/IP sockets in C: practical guide for programmers***. Morgan Kaufmann. 2009
- W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff. 2003. ***UNIX Network Programming***, Vol. 1 (3 ed.). Pearson Education.
- Douglas E. Comer. 1991. ***Internetworking with TCP/IP*** (2nd Ed.), Vol. I. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
- Michael Kerrisk. 2010. ***The Linux Programming Interface: A Linux and UNIX System Programming Handbook*** (1st ed.). No Starch Press, San Francisco, CA, USA.
- Bjarne Stroustrup. ***The C++ Programming Language***. Addison-Wesley, 2013, 4th Edition.
- Nivio Ziviani. ***Projeto de Algoritmos com Implementações em Pascal e C***. Cengage Learning, 3ª Ed, 2010.
- https://computing.llnl.gov/tutorials/pthreads/
- https://linux.die.net/man/3/opendir
- Manual do linux

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