Controle de Envio de Arquivos

Execução: Individual
Data de entrega: 17 de maio de 2023 até 23h:59min

INTRODUÇÃO

Dois amigos querem transferir arquivos entre seus computadores em um sítio sem acesso a internet, logo decidem realizar a transferência na rede local utilizando sockets. Para simplificar o funcionamento da transferência, um computador age como o servidor e somente e o outro age como cliente. Estes arquivos devem ser ***Plain Text***, ou seja, texto puro. Nesta primeira versão não será possível enviar *.bin, *.sh entre outros tipos além de texto.

OBJETIVO

O objetivo deste trabalho prático é implementar um sistema de transferência de arquivo simples entre um servidor e um cliente utilizando sockets em C. O servidor deve aceitar a conexão de um único cliente, permitindo que ele envie um arquivo para o servidor. O servidor deve receber o arquivo e armazená-lo em seu sistema de arquivos. O cliente deve ser capaz de se conectar ao servidor, enviar um arquivo e receber uma confirmação do servidor de que o arquivo foi recebido com sucesso.

O servidor deve ser capaz de executar as seguintes funções:

- Aceitar a conexão de um único cliente;
- Receber o arquivo do cliente e armazená-lo no sistema de arquivos;
- Confirmar ao cliente que o arquivo foi recebido com sucesso.
- Receber pedido de encerramento de conexão.

O cliente deve ser capaz de executar as seguintes funções:

- Conectar-se ao servidor;
- Selecionar um arquivo para enviar ao servidor;
- Enviar o arquivo ao servidor;
- Receber uma confirmação do servidor de que o arquivo foi recebido com sucesso.
- Encerrar a conexão com o servidor.

PROTOCOLO

O protocolo de comunicação deve ser simples, com mensagens em texto puro. Cada mensagem deve ter um cabeçalho que indica o nome do arquivo utilizado no destino, seguido pelo conteúdo da mensagem. O conteúdo da mensagem deve conter os dados do arquivo. Caso o cliente queira encerrar a conexão, envie a mensagem “exit” para o servidor e a conexão deve ser encerrada. Os tipos válidos de arquivos podem ser vistos na Tabela I.

Tipos Válidos de Arquivos
*.txt
*.c
*.cpp
*.py
*.tex
*.java

Tabela I

O cliente e o servidor trocam mensagens curtas de até 500 bytes utilizando o protocolo TCP. As mensagens carregam textos codificados segundo a tabela ASCII. Apenas letras, números e espaços podem ser transmitidos. Caracteres acentuados e especiais não devem ser transmitidos.

As mensagens de comando são inseridas somente no cliente e a resposta vem do servidor. A seguir, estão descritas as ações que devem ser performadas, bem como a formatação de cada tipo de mensagem e a resposta desejada:

- **Selecionar arquivo a ser enviado:** o cliente seleciona o arquivo local a ser enviado ao servidor. Isso deve ser feito através do comando **“select file [nomearquivo]”**, onde o campo **nomearquivo** descreve o nome do arquivo com sua extensão**.** Não deve ser possível selecionar um arquivo que não tenha uma das extensões da Tabela I. Caso o arquivo seja válido, o próprio cliente deve retornar uma mensagem "**[nomearquivo] selected”.** Caso isso ocorra, o próprio cliente deve retornar uma mensagem "**[nomearquivo] not valid!”**. Caso o arquivo não exista na pasta, o próprio cliente deve retornar uma mensagem "**[nomearquivo] do not exist”.**
- **Enviar arquivo selecionado:** Após selecionar um arquivo válido, enviar o mesmo. Essa ação deve ser feita através do comando **“send file”**. Após enviado, o servidor deve retornar a resposta "**file [nomearquivo] received”**. Caso o envio falhe, o servidor deve retornar a mensagem "**error receiving file [nomearquivo]”**. Caso o cliente não tenha selecionado nenhum arquivo, o próprio cliente deve retornar a mensagem "**no file selected!”**. Caso o arquivo enviado já exista no servidor, o mesmo deve ser sobrescrito e o servidor deve retornar a resposta "**file [nomearquivo] overwritten".**
- **terminar sessão:** O cliente ao querer finalizar a sessão deve requisitar termino. Para isso, deve usar o comando **“exit”**. O servidor deve responder "**connection closed”**.

Os exemplos a seguir são da tela do cliente já após estar conectado ao servidor:

Exemplo 1:

Terminal Cliente	Resposta Cliente	Resposta Servidor
select file arquivooooo.txt	arquivooooo.txt does not exist	
select file arquivo.txt	arquivo.txt selected	
send file		file arquivo.txt received
send file		file arquivo.txt overwritten
exit		connection closed

Exemplo 2:

Terminal Cliente	Resposta Cliente	Resposta Servidor
send file	no file selected!	
select file half-life.exe	half-life.exe not valid!	
send file	no file selected!	
select file arquivo.txt	arquivo.txt selected	
send file		file arquivo.txt received
exit		connection closed

Detalhes de Implementação do Protocolo:

- Como dentro do arquivo de texto podem existir caracteres de quebra de linha `'\n'`, as mensagens serão terminadas com uma sequência definida `'\end'`. O caractere nulo `'\0'` para terminação de strings em C *não* deve ser enviado na rede.
- Atente-se que até as mensagens que são digitadas no terminal do cliente terminam em um `'\n'`, mas ela não é exatamente o que é enviado via socket!
- Dica: faça o tratamento de strings com a biblioteca `<string.h>`.
    - Nesse caso , o servidor deve enviar todas as respostas em uma mesma linha, conforme mostrado nos exemplos acima.
- O servidor deve desconectar o cliente caso receba uma mensagem com um comando desconhecido (exemplo: "`selec`" em vez de "`select`"), mas não precisa retornar mensagem inválida.

- Para funcionamento do sistema de correção semi-automática (descrito abaixo), seu servidor deve fechar todas as conexões e terminar sua execução ao receber a mensagem **“`exit`”** a qualquer momento.

DICAS

As dicas a seguir correspondem à manipulação de arquivos em C. Existem vários tutoriais na internet e nas aulas de C de alguns professores do departamento. Além disso se lembre de:

- Use da biblioteca `<string.h>` para verificar as extensões dos arquivos. Pode fazer sem ela se quiser, mas terá mais trabalho.
- Use as funções de File para verificar se o arquivo já existe, tanto na hora de escrever no servidor, quanto na hora de selecionar o arquivo no cliente.

IMPLEMENTAÇÃO

O aluno deve implementar tanto uma versão do servidor quanto uma versão do cliente. Ambos devem utilizar o protocolo TCP, criado com **[socket(AF_INET, SOCK_STREAM, 0)]** ou com **[socket(AF_INET6, SOCK_STREAM, 0)]**. Deve ser possível utilizar tanto o IPv4 quanto o IPv6.

O **cliente** deve receber mensagens do teclado e imprimir as mensagens recebidas na tela. O **servidor** deve imprimir na saída padrão todas as mensagens recebidas do cliente. **Não é necessário** que o servidor aceite mais de um cliente simultaneamente.

Seu servidor deve receber, **estritamente nessa ordem**, o tipo de endereço que será utilizado (**v4** para IPv4 ou **v6** para IPv6) e um número de porta na linha de comando especificando em qual porta ele vai receber conexões (Sugestão: utilize a porta 51511 para efeitos de padronização do trabalho). Seu cliente deve receber, **estritamente nessa ordem**, o endereço IP e a porta do servidor para estabelecimento da conexão.

A seguir, um exemplo de execução dos programas em dois terminais distintos:

IPv4:

```
no terminal 1: ./server v4 51511
no terminal 2: ./client 127.0.0.1 51511
```

IPv6:

```
no terminal 1: ./server v6 51511
no terminal 2: ./client ::1 51511
```

O servidor pode dar **bind** em todos os endereços IP associados às suas interfaces usando a constante **INADDR_ANY** para IPv4 ou **in6addr_any** para IPv6.

Limites:

- Cada mensagem possui no máximo 500 bytes.

Materiais para Consulta:

- Capítulo 2 e 3 do livro sobre programação com sockets disponibilizado no Moodle.
- Playlist de programação com sockets.

AVALIAÇÃO

O trabalho deve ser realizado individualmente e **deve ser implementado com a linguagem de programação C** utilizando somente a biblioteca padrão (interface POSIX de sockets de redes). Deve ser possível executar seu programa no sistema operacional **Linux** e **não deve utilizar bibliotecas Windows, como o winsock**. Seu programa deve interoperar com qualquer outro programa implementando o mesmo protocolo (você pode testar com as implementações dos seus colegas). Procure escrever seu código de maneira clara, com comentários pontuais e bem indentados. Isto facilita a correção dos monitores e tem impacto positivo na avaliação.

Correção Semi-automática

Seu servidor será corrigido de forma semi-automática por uma bateria de testes. Cada teste verifica uma funcionalidade específica do servidor. O seu servidor será testado por um cliente implementado pelo professor com funcionalidades adicionais para realização dos testes. Os testes avaliam a aderência do seu servidor ao protocolo de comunicação inteiramente através dos dados trocados através da rede (a saída do seu servidor na tela, e.g., para depuração, não impacta os resultados dos testes).

Para a correção os seguintes testes serão realizados **(com IPv4 e IPv6)**:

- Selecionar arquivo: **15%**
- Enviar arquivo: **15%**
- Tratamentos de seleção/envio: **30%**
- Tratamentos de recebimento: **30%**
- Cliente envia exit para o servidor e fecha a execução: **10%**

Obs.1: Caso os testes funcionem em apenas um tipo de endereço (IPv4 ou IPv6), a pontuação do respectivo teste será reduzida pela metade.

Obs.2: Considere para cada cenário acima todas as possibilidades possíveis (tentar enviar arquivo que não existe, sobrescrever arquivo, etc).

Obs.3: Não é necessário fazer tratamento para overflow de mensagens.

Entrega

Cada aluno deve entregar documentação em PDF de até 4 páginas (duas folhas), sem capa, utilizando fonte tamanho 10, e figuras de tamanho adequado ao tamanho da fonte. A documentação deve discutir desafios, dificuldades e imprevistos do projeto, bem como as soluções adotadas para os problemas. A documentação corresponde a 20% dos pontos do trabalho, mas só será considerada para as funcionalidades implementadas corretamente.

Será utilizado um sistema para detecção de código repetido, portanto não é admitido cola de trabalhos.

Será adotada média harmônica entre as notas da documentação e da execução, o que implica que a nota final será 0 se uma das partes não for apresentada.

Cada aluno deve entregar, além da documentação, o **código fonte em C** e um **Makefile** para compilação do programa. Instruções para submissão e compatibilidade com o sistema de correção semi-automática:

- O Makefile deve compilar o cliente em um binário chamado "`client`" e o servidor em um binário chamado "`server`".
- Seu programa deve ser compilado ao se executar apenas o comando "`make`", ou seja, sem a necessidade de parâmetros adicionais.
- A entrega deve ser feita no formato ZIP, com o nome seguindo o seguinte padrão: TP1_MATRICULA.zip
- Os nomes dos arquivos devem ser padronizado:
    - `server.c`
    - `client.c`
    - `common.c, common.h (se houver)`

Desconto de Nota por Atraso

Os trabalhos poderão ser entregues até a meia-noite do dia especificado para a entrega. O horário de entrega deve respeitar o relógio do sistema Moodle, ou seja, a partir de 00:01 do dia seguinte à entrega no relógio do Moodle, os trabalhos já estarão sujeitos a penalidades.

A fórmula para desconto por atraso na entrega do trabalho prático é:

$$desconto = 2^{d} - 1$$

onde d é o atraso em dias úteis. Note que após 3 dias, o trabalho não pode ser mais entregue.