Instituto de Matemática e Estatística da USP

MAC0216 - Técnicas de Programação I - 2s2023

EP2
Entrega até 8:00 de 9/10/2023
(INDIVIDUAL)

Prof. Daniel Macêdo Batista

1 Problema

A tarefa neste EP é implementar um sistema de chat (cliente e servidor) em bash script, em modo texto, para permitir a comunicação local de usuários logados no mesmo computador. Todos os códigos devem ser escritos em bash script para serem executados no GNU/Linux. O vídeo disponibilizado no YouTube em `https://youtu.be/UHBBcxs2Zd4` mostra como deve ser o comportamento do sistema (Note que o nome do arquivo está diferente do que precisa ser entregue. Além disso, não está sendo exibido no vídeo mas as mensagens para o Telegram que serão explicadas adiante neste enunciado precisam ser enviadas durante a execução).

O sistema estará implementado em um único arquivo `ep2.sh`. Para executar o servidor é necessário invocá-lo no shell da seguinte forma:

```
./ep2.sh servidor
```

Basta 1 invocação do servidor.

Para executar o cliente é necessário invocá-lo no shell da seguinte forma:

```
./ep2.sh cliente
```

Cada usuário que deseje usar o sistema de chat deverá invocar o script em modo cliente da forma acima. Nos exemplos acima considera-se que o script já está com permissão de execução e que ele está localizado no diretório local.

1.1 O servidor de chat

O servidor de chat deve ser iniciado antes de qualquer cliente ser iniciado. Uma vez inicializado, o servidor deve exibir na tela um prompt de comandos aguardando os comandos do usuário:

`servidor>`

O servidor deve suportar os seguintes comandos:

- `list`: lista os nomes de todos os usuários logados, um por linha. Se não tiver nenhum usuário logado, não precisa listar nada na tela;
- `time`: informa o intervalo de tempo, em segundos, desde que o servidor foi iniciado;
- `reset`: remove todos os usuários que foram criados nessa instância de execução atual do servidor;
- `quit`: finaliza o servidor.

Não se preocupe em tratar a execução dos comandos `reset` e `quit` se ainda houver algum cliente conectado. Considere que esses comandos só serão executados se não houver clientes conectados. Também não se preocupe em testar o sistema caso o servidor execute e já haja algum cliente ou servidor rodando. Considere que isso nunca vai acontecer.

1.2 O cliente de chat

Os clientes de chat devem ser iniciados após o servidor ser iniciado. Uma vez inicializado, o cliente deve exibir na tela um prompt de comandos aguardando os comandos do usuário:

`cliente>`

O cliente deve suportar os seguintes comandos sem necessidade do usuário estar logado:

- `create usuario senha`: cria um novo usuário de nome `usuario` e de senha `senha`. Ambas informações devem ser salvas no servidor e precisam ser mantidas enquanto o servidor estiver em execução. Só podem ser perdidas quando o servidor executar um `reset` ou um `quit`. Se o usuário `usuario` já existir, a string `ERRO` deve ser impressa;
- `passwd usuario antiga nova`: modifica a senha do usuário `usuario` de `antiga` para `nova`. A nova senha deve ser atualizada no servidor. Se o usuário não existir ou se a senha antiga estiver errada, a string `ERRO` deve ser impressa;
- `login usuario senha`: loga como o usuário de nome `usuario` com a senha `senha`. Se o usuário não existir, se a senha estiver errada ou se o usuário já estiver logado no sistema, a string `ERRO` deve ser impressa.
- `quit`: encerra a execução do cliente. Caso o usuário não tenha feito `logout`, faz `logout` antes de encerrar;

O cliente deve suportar os seguintes comandos após o usuário logar:

- `list`: lista os nomes de todos os usuários logados, um por linha, inclusive o próprio usuário;
- `logout`: desloga do sistema mas não encerra a execução do cliente;

- `msg usuario mensagem`: Escreve na tela do usuário `usuario` a mensagem `mensagem`. Se o usuário `usuario` não estiver logado, a string `ERRO` deve ser impressa. Na tela do `usuario` a mensagem deve ser impressa imediatamente com o prefixo `[Mensagem do remetente]:` , onde `remetente` deve ser o nome do remetente da mensagem;

Para todos os três comandos acima, se o usuário não estiver logado, deve ser impressa a string `ERRO`.

Para todos os comandos considere que a `mensagem` é o único parâmetro que poderá ter espaços em branco. Nenhum parâmetro terá tabulações ou quebras de linha.

1.3 O monitoramento via Telegram

No geral, sistemas críticos precisam informar possíveis problemas constantemente para seus administradores. Para que esses problemas sejam visualizados o mais rápido possível, em um passado não muito distante eram enviadas mensagens para pagers ou mensagens SMS para telefones celulares. Nos últimos anos essas opções têm sido substituídas pelo uso de aplicativos de mensagem como o Telegram em smartphones.

Além de explorar o uso de diversos comandos do bash, e utilitários do Linux acessíveis pelo bash para permitir a conversa entre usuário, este EP também usará esses comandos e utilitários para que um bot envie alertas via Telegram. **Registros de sistemas computacionais**, como os que você terá que implementar aqui usando bash script via Telegram, são mais uma importante Técnica de Programação pois eles ajudam no monitoramento dos sistemas e facilitam na descoberta e correção de problemas. Um pouco mais sobre isso será visto na disciplina MAC0422 - Sistemas Operacionais.

Para que o computador possa enviar uma mensagem para você via Telegram ele precisa ter um usuário. Esse usuário vai ser de um tipo especial, chamado bot. Para criar um bot no Telegram, siga os passos descritos na página em:

`https://imasters.com.br/desenvolvimento/criando-um-bot-no-telegram-para-avisar-que-um-sistema-esta-fora-do-ar`

Leia a página até o trecho onde está escrito: "Esse é o chat id." Ao término dessa leitura e depois de ter seguido os passos você terá duas informações importantes: o token do bot e o ID do bot. Para confirmar que tudo deu certo, rode o seguinte comando no shell do seu computador conectado à Internet:

```
# Coloque o token e o ID nos lugares informados sem o '<' e o '>' abaixo
curl -s --data "text=oi" https://api.telegram.org/bot<token aqui>/sendMessage?chat_id=<ID aqui> 1>/dev/null
```

Você deverá receber um "oi" do seu bot no Telegram. Se você receber um erro de "Comando não encontrado" no shell, significa que você precisa instalar o curl no seu computador. Em um sistema operacional baseado em Debian, o comando `sudo apt-get install curl` é suficiente para a instalação.

Uma vez de posse do token ID e do ID do bot, coloque a URL como uma variável *hardcoded* no seu bash script para que o servidor possa enviar mensagens para você. Essas mensagens precisarão ser enviadas imediatamente pelo código do servidor nestas 3 situações:

- Usuário erra a senha: é necessário informar quem foi o usuário e qual o dia e hora em que o erro de senha ocorreu
- Usuário logou com sucesso: é necessário informar quem foi o usuário e qual o dia e hora em que ele logou
- Usuário deslogou: é necessário informar quem foi o usuário e qual o dia e hora em que ele deslogou

Além disso, de 1 em 1 minuto o servidor precisa enviar via Telegram a lista de todos os usuários conectados.

Note que as informações a serem enviadas via Telegram não devem ser exibidas no terminal. Tudo isso deve funcionar de forma transparente para o usuário que está executando o servidor e para os usuários que estiverem executando os clientes.

Não há um formato padrão para o envio das informações via bot do Telegram. O importante é que tudo que foi solicitado seja informado de uma forma que seja possível interpretar cada informação (Só apresentar nomes de usuários sem informação adicional não é uma boa prática).

2 Requisitos

O código deve ser escrito como **um único arquivo `ep2.sh`** e as primeiras linhas do arquivo devem ser obrigatoriamente:

```
#!/bin/bash
# AUTOR:
# <Seu nome, NUSP e endereço de e-mail>
#
# DESCRIÇÃO:
# <Explique o que os programas fazem (Não diga apenas que ele é o EP2 da
# disciplina tal, explique o que de fato os programas fazem)>
#
# COMO EXECUTAR:
# <Informe como executar o cliente e o servidor e quais os parâmetros
# de linha de comando necessários>
#
# TESTES:
# <Forneça explicações sobre o testa.c>
#
# DEPENDÊNCIAS:
# <Informe o que é necessário para rodar o programa. Informações como o
# SO onde você executou e sabe que ele funciona são importantes de serem
# colocadas aqui.>
```

Todo o código deve ser escrito em bash script para funcionar no shell sem qualquer interação com o usuário em modo gráfico. **Não é permitido utilizar comandos ou programas que já implementem, mesmo que parcialmente, um sistema de chat no shell. Programas que não respeitem esse requisito terão nota ZERO.**

Caso seja necessário criar arquivos ou diretórios temporários para garantir a execução correta do sistema, esses arquivos devem ser criados no diretório `/tmp/`. Certifique-se de remover tudo que tiver sido criado no sistema de arquivos quando os clientes e o servidor forem encerrados.

2.1 Relatório

O desempenho, em termos de atraso entre o momento que o usuário erra a senha e o momento da chegada da mensagem no Telegram deverá ser avaliada durante 7 dias de semana em 3 horários diferentes: um de

manhã, um de tarde e um de noite. Tente repetir as medições sempre nos mesmos horários para que a comparação seja justa.

Para medir o atraso você terá que usar um cronômetro (um relógio de verdade, um software no seu computador ou uma app no *smartphone*). Logue no cliente, digite uma senha errada, inicie o cronômetro ao mesmo tempo em que pressiona o ENTER e pare quando a mensagem chegar no Telegram (Mantenha o Telegram aberto na conversa com o bot e desligue a proteção de tela para não correr o risco de você não ver quando a mensagem chegar). Coloque os valores numa tabela no seu relatório conforme apresentado na Tabela 1.

Tabela 1: Atraso em segundos

Dia da semana	**Data e hora**	**Atraso**
Domingo	dd/mm hh:mm manhã	Valor em segundos
Domingo	dd/mm hh:mm tarde	Valor em segundos
Domingo	dd/mm hh:mm noite	Valor em segundos
Segunda	dd/mm hh:mm manhã	Valor em segundos
Segunda	dd/mm hh:mm tarde	Valor em segundos
Segunda	dd/mm hh:mm noite	Valor em segundos
Terça	dd/mm hh:mm manhã	Valor em segundos
Terça	dd/mm hh:mm tarde	Valor em segundos
Terça	dd/mm hh:mm noite	Valor em segundos
Quarta	dd/mm hh:mm manhã	Valor em segundos
Quarta	dd/mm hh:mm tarde	Valor em segundos
Quarta	dd/mm hh:mm noite	Valor em segundos
Quinta	dd/mm hh:mm manhã	Valor em segundos
Quinta	dd/mm hh:mm tarde	Valor em segundos
Quinta	dd/mm hh:mm noite	Valor em segundos
Sexta	dd/mm hh:mm manhã	Valor em segundos
Sexta	dd/mm hh:mm tarde	Valor em segundos
Sexta	dd/mm hh:mm noite	Valor em segundos
Sábado	dd/mm hh:mm manhã	Valor em segundos
Sábado	dd/mm hh:mm tarde	Valor em segundos
Sábado	dd/mm hh:mm noite	Valor em segundos

Além das tabelas, o seu relatório precisa ter um cabeçalho com seu nome e NUSP e uma conclusão explicando se os resultados obtidos em termos de atraso seguem algum padrão que faça sentido considerando os diferentes dias da semana e horários. Também informe a configuração dos computadores usados (processador, memória e sistema operacional) onde todas as execuções foram realizadas. O ideal é que todas as medições sejam feitas no mesmo computador mas caso isso seja complicado para você, por conta da necessidade de execuções em horários específicos, tudo bem se forem usados computadores diferentes.

3 Entrega

Você deverá entregar um arquivo tarball comprimido (.tar.gz) contendo os seguintes itens:

- Implementação do `ep2.sh`;
- 1 arquivo .pdf com o relatório.

Todos esses arquivos devem estar presentes. Caso algum arquivo esteja faltando, o EP não será corrigido e a nota dele será zero.

O desempacotamento do tarball deve produzir um diretório contendo os itens. O nome do diretório deve ser ep2-seu_nome. Por exemplo: ep2-joao_dos_santos.

A entrega do tarball deve ser feita no e-Disciplinas.

O EP deve ser feito individualmente.

Obs.1: Serão descontados 2,0 pontos de EPs com tarballs que não estejam nomeados como solicitado ou que não criem o diretório com o nome correto após serem descompactados. Confirme que o seu tarball está correto, descompactando ele no shell (não confie em interfaces gráficas na hora de verificar o seu tarball pois alguns gerenciadores de arquivos criam o diretório automaticamente mesmo quando esse diretório não existe).

Obs.2: A depender da qualidade do conteúdo que for entregue, o EP pode ser considerado como não entregue, implicando em MF=0,0. Isso acontecerá também se for enviado um tarball corrompido, códigos fonte vazios ou códigos fonte que resolvam um problema completamente diferente do que foi solicitado.

Obs.3: O prazo de entrega expira às 8:00:00 do dia 09/10/2023.

4 Avaliação

80% da nota será dada pela implementação, e 20% pelo relatório. Os critérios detalhados da correção serão disponibilizados apenas quando as notas forem liberadas.