Relatório do EP2 de MAC 0216

Nome: Odair Gonçalves de Oliveira
Número USP: 13671581

Resumo

Este relatório refere-se ao Exercício Programa 2 (EP2) da disciplina Técnicas de Programação I (MAC 0216), orientada pelo professor Daniel Macedo Batista, do curso de Ciência da Computação do Instituto de Matemática e Estatística da Universidade de São Paulo (IME - USP). O EP2 envolveu a implementação de um sistema de chat, com cliente e servidor, usando bash script como ferramenta principal além de contar com uma implementação de um bot do Telegram que faz o envio de informações sobre esse sistema para um chat no Telegram. Este documento tem como objetivo descrever a abordagem utilizada, os desafios enfrentados e as lições aprendidas durante o desenvolvimento do projeto.

Conteúdo

1 Introdução e Objetivos

O EP2 consistiu na implementação de um arquivo escrito em bash script, com o nome de ep2.sh, que permite rodar no terminal um serviço de chat com cliente e servidor. O objetivo principal é estabelecer uma comunicação entre diferentes terminais abertos ao mesmo tempo. Além disso, dentro de script, foi implementado um processo em paralelo para envio de informações sobre o sistema de chat direto para o Telegram a partir da criação de um bot conforme orientações disponíveis no enunciado do EP2.

Os objetivos educativos eram provocar uma maior familiarização com os comandos de bash e a construção de scripts. A *segunda parte* da disciplina de Técnicas de Programação I consistiu em nos aproximar da linguagem bash para aprendermos a executar algumas tarefas como: busca de arquivos, escrita de arquivos, alterações, exclusões, criações, listagens, entre outras, que manualmente ou feitos de forma individual poderiam ser extremamente complexas dependendo da dimensão do projeto em que estamos trabalhando. Ou seja, a ideia é nos ensinar a utilizar uma ferramenta que permite acelerar esses processos, além de condensá-los em um script que seja capaz de realizar tarefas mais elaboradas usando diversos desses comandos de forma conjunta. Assim sendo, o EP2 nos permitiu aplicar esse conhecimento de forma prática para aprofundar esse aprendizado.

2 Detalhes da Implementação

O EP2 consiste em um arquivo bash script, com final ".sh", que executa o que sistema de chat que foi descrito anteriormente. A ideia é poder rodar o script diretamente no terminal junto com um termo, "cliente"ou "servidor"para escolher o modo de execução do mesmo.

```
bash ep2.sh <opção>
```

2.1 Modo Servidor

O **modo servidor** deve ser iniciado antes do modo cliente, pois é nesse modo de execução do script que são criados os arquivos que irão guardar as informações dos usuários que poderão vir a realizar o login. Esse modo conta com algumas opções exclusivas que permitem o controle do sistema.

As principais tarefas desse modo são criar os arquivos, sendo que em um deles ficarão salvas as informações dos usuários e senhas criados, enquanto no outro arquivo ficará salvo uma lista dos clientes logados. Vale destacar que essa implementação não considera que poderão haver vários servidores rodando ao mesmo tempo, pois os arquivos que são criados ao iniciar o servidor tem um nome padrão e são todos salvos na pasta **tmp** do computador. Ou seja, caso houvesse mais de um servidor rodando, eles estariam referenciando os mesmos arquivos, porém, como foi descrito que poderíamos supor que isso não iria ocorrer, o tratamento para esse caso não foi feito. Assim, concluindo, esse modo gerencia esses dois arquivos principais e é responsável por excluir eles ao final da execução.

As opções disponíveis para o usuário que estiver rodando o modo servidor são:

- `list:`

  Permite listar na tela o nome dos usuários que estão logados no sistema. Faz isso verificando um arquivo que guarda essa informação ("logados.txt").

- `reset:`

  Faz a remoção de todos os usuários que foram criados na instância do servidor. Essa tarefa é feita limpando o arquivo que guarda as informações dos usuários criados. No enunciado é dito que não é preciso considerar o caso de o reset ser feito quando há usuários logados, mas decidi que o reset iria limpar também o arquivo que guarda as informações dos usuários logados também.

- `time:`

  Esse comando permite imprimir na tela a quanto tempo o servidor foi iniciado em segundos. Vale destacar que o **reset** não fazer esse timer ser reinicializado.

- `quit:`

  Encerra o servidor e apaga os arquivos que guardam as informações dos usuários.

Além disso, é nesse modo que ocorre a verificação de usuários logados no sistema junto do envio das mensagens ao Telegram caso haja usuários logados. Esse processo acontece em paralelo e é feito em um loop que verifica se existe alguma informação escrita no arquivo que guarda uma lista dos usuários logados ("logados.txt"). As mensagens enviadas nesse processo em paralelo informam o nome dos usuários que estão logados no momento. Outro ponto importante é a falta de segurança envolvida nesse sistema de chat, já que os usuários e senhas dos mesmos ficam em um arquivo que pode ser acessado facilmente, porém, como questões como essa não eram pertinentes ao EP2, não vi grandes problemas em deixar dessa forma mesmo.

2.2 Modo Cliente

O **modo cliente** é o modo no qual é possível atuar como um usuário do sistema de chat. Esse modo só pode ser utilizado caso o servidor já tenha sido inicializado antes e isso é feito verificando a existência dos arquivos que o modo servidor cria na pasta tmp. Ou seja, caso existam arquivos chamados "usuarios.txt"e "logados.txt"na pasta do tmp do computador em que este bash esteja rodando, o modo cliente poderá ser ativado mesmo sem o início do servidor. Ao iniciar esse modo, cria-se um arquivo temporário com um nome aleatório que irá guardar o caminho para o pipe com o nome do usuário que fizer login durante aquela instância atual de execução do modo do cliente.

O modo cliente possui dois tipos de opções: as opções para o usuário logado e as opções para o usuário que não está logado. Para facilicitar o uso de um possível usuário que nunca usou o sistema antes, após o modo cliente ser iniciado, é printado na tela um cabeçalho que explica quais são as opções disponíveis. Veja a seguir uma lista delas:

- `create usuario senha: (Sem login)`

  Registra no arquivo "usuarios.txt", que guarda as informações dos usuários criados, esse novo usuário adicionado e sua senha. Antes, verifica se o nome de usuário está disponível, se não estiver, avisa o cliente e espera um novo comando ser executado.

- `passwd usuario antiga nova: (Sem login)`

  Atualiza a senha do 'usuario' que deve ser a 'antiga' e a substitui no arquivo "usuarios.txt"pela nova. Note que há duas verificações a serem feitas aqui. Se o usuário realmente existe e se a senha antiga informada está correta. Se estiver tudo certo, atualizamos a senha do usuário no arquivo. Tal alteração é feita de forma que apenas a informação sobre o usuário 'usuario' é alterada, ou seja, se houverem outras palavras nesse arquivo que sejam iguais a senha 'antiga' elas não serão alteradas.

- `login usuario senha: (Sem login)`

  Esse comando realiza o login do usuário 'usuario' no sistema, marcando seu nome no arquivo "logados.txt". Antes, verifica se o usuário existe. Se existir, verifica se ele já não está logado, se estiver, avisa-se com uma mensagem na tela. Se não estiver, verifica-se a senha e se estiver correta, o login é feito e envia-se uma mensagem ao Telegram informando o nome do usuário que fez login e a data em que isso ocorreu. Essa função então cria um pipe com o nome do usuário (uma espécie de arquivo que fica dentro da pasta **tmp**) e salva esse caminho dentro do arquivo temporário criado no início da execução. Após feito o login de um usuário, não é possível realizar outro login (um login duplo por exemplo) pois ocorrem verificações no arquivo temporário: se estiver vazio, isso indica que não há nenhum usuário logado naquela instância e se não estiver, então há um usuário logado.

- `quit: (Sem login)`

  Encerra o modo cliente e apaga o arquivo temporário criado no ínicio e faz o logout do usuário, caso exista algum usuário logado.

- `list: (Com login)`

  Lista todos os usuários logados.

- `logout: (Com login)`

  Realiza o logout do usuário da instância atual. Isso é, verifica se há algum usuário logado, se estiver, apaga o nome do mesmo da lista de logados (no arquivo "logados.txt"), apaga o pipe que foi criado no login e limpa o conteúdo do arquivo temporário que guardava o endereço desse pipe.

- `mensagem usuario mensagem: (Com login)`

  Envia uma mensagem para o pipe com nome 'usuario'.

Além de suportar essas funções, em um loop que ocorre em processo paralelo, o script verifica se algum conteúdo (mensagem) foi enviado para o pipe do usuário da instância atual. Se foi, ele imprime o que foi enviado, se não, não faz nada, apenas continua o loop para a próxima verificação.

2.3 Comentários gerais da implementação

Por conta de ocorrerem alguns processos em parelelo em segundo plano no código e existir uma questão do escopo das variáveis utilizadas, foi necessário usar mais arquivos do que eu esperava (já que de início pensei ser possível fazer sem usar arquivo algum, talvez seja possível fazer assim, me falta conhecimento para afirmar ou negar isso, porém a saída mais fácil foi fazer utilizando os arquivos para não ter esse problema de escopo).

Outro problema enfrentado foi a questão do loop em segundo plano das mensagens que o usuário da instância atual pode receber a qualquer momento. Pelo fato de ser um loop que roda infinitamente, ele acabava rodando muito e sobrecarregando o sistema (só notei porque meu notebook começou a ventilar). Por fim, para evitar isso, encontrei na internet uma estratégia de usar o 'sleep 1' dentro do loop para evitar isso. Vale falar que esse sobrecarregamento só ocorria quando o usuário havia feito login e não tinha recebido nenhuma mensagem ainda. Pois, após fazer login e receber uma mensagem, o loop com as condições ficava parado esperando receber algo do canal (usando o 'cat') então ele não ficava verificando continuamente. Se não recebesse nada, apenas ficava lá parado esperando. Isso acabou gerando um outro problema: quando o usuário realizada logout e o canal era destruído, o loop não reiniciava, ficava congelado esperando receber algo e torná-va impossível que outro usuário fizesse login e recebesse mensagens. Para corrigir isso decidi, na função de logout, mandar um (string vazia) para o canal do usuário antes e logo depois destruir o pipe, o que corrigiu o problema do loop de mensagens travado.

Tirando essas duas questões, tive alguns problemas em parar os processos que rodavam em paralelo. Pois, dependendo da condição do loop, e da forma que eu encerrava o programa (usando 'ctrl' + 'c' por exemplo), o processo em paralelo não parava e ficava rodando mesmo fechando o terminal. Mas acabei usando o 'trap "algum comando"EXIT' que fazia com o que o 'algum comando' fosse executado quando o programa fosse encerrado. Isso me ajudou a lidar com os arquivos criados e com os processos em segundo plano no caso de um encerramento não ideal (sendo o encerramento ideal o ato de acionar o 'quit').

3 Testes e Resultados

As configurações do computador em que os testes foram rodados estão expostas a seguir:

1. Processador: 12º Geração - IntelCore i5-12500H;
2. Memória RAM: 8 GB;
3. Sistema Operacional: Ubuntu 22.04.3 LTS;
4. Arquitetura: x86_64;

Os testes desse EP envolviam analisar o tempo que levava para as mensagens serem enviadas e chegarem até o Telegram. O teste consistia em realizar o login de um usuário com uma senha incorreta e avaliar quanto tempo levava para receber o alerta de que isso ocorreu no Telegram. Tal cálculo foi feito utilizado um cronômetro do meu celular. Veja a seguir uma tabela com os 7 dias de teste indicando em segundos quando tempo levava:

Dia da Semana	Data e Hora	Atraso (em segundos)
Terça	07/11/23 Manhã 09:10	2.45s
Terça	07/11/23 Tarde 13:47	1.75s
Terça	07/11/23 Noite 21:19	2.38s
Quarta	08/11/23 Manhã 09:30	2.93s
Quarta	08/11/23 Tarde 15:31	1.31s
Quarta	08/11/23 Noite 21:40	2.18s
Quinta	09/11/23 Manhã 08:45	2.18s
Quinta	09/11/23 Tarde 14:45	1.96s
Quinta	09/11/23 Noite 22:20	2.46s
Sexta	10/11/23 Manhã 07:15	1.52s
Sexta	10/11/23 Tarde 13:05	2.77s
Sexta	10/11/23 Noite 22:15	2.35s
Sábado	11/11/23 Manhã 09:00	2.83s
Sábado	11/11/23 Tarde 14:16	1.28s
Sábado	11/11/23 Noite 21:00	2.53s
Domingo	12/11/23 Manhã 10:30	1.67s
Domingo	12/11/23 Tarde 16:45	2.45s
Domingo	12/11/23 Noite 21:47	2.23s
Segunda	13/11/23 Manhã 08:30	2.11s
Segunda	13/11/23 Tarde 12:17	2.62s
Segunda	13/11/23 Noite 20:20	1.99s

Não consegui notar nenhuma regularidade ou padrão pelos testes realizados, mas vale destacar que executei os testes em locais diferentes com redes de internet diferentes. O que eu esperava era um tempo maior em horários de 'pico' no uso da rede social ou de internet, mas não consegui notar quais seriam esses horários.

4 Conclusão

Em conclusão, o EP2 permitiu um aprofundamento maior no uso de bash e no entendimento e na construção de scripts. Graças a esse exercício, pude ter uma ideia do potencial que esse tipo de ferramenta permite. Sei que um sistema de chat verdadeiro provavelmente conta com muito mais segurança e um trabalho mais elaborado do que o que foi realizado, mas a versão simples que fiz me permitiu ter uma ideia de alguns dos dilemas e questões que envolvem a elaboração de um verdadeiro. Em relação aos testes, esperava resultados com uma maior variedade e algum padrão observável, mas não consegui notar nada muito significativo, o que talvez possa ser explicado pelo fato de eu ter realizado os testes em locais diferentes com acessos a internet diferentes. Por fim, gostei o trabalho e com certeza vou aplicar esse conhecimento em bash em outros projetos futuros.