

Documentação TP de redes

O trabalho proposto consiste em implementar o paradigma de comunicação publish/subscribe. Para isso, alguns requisitos foram estabelecidos como o uso dos protocolos TCP/IP, a conexão de um número arbitrário de clientes e programação em paralelo. Ao longo do desenvolvimento do mesmo projetou-se uma arquitetura de cliente e servidor, bem como arquivos auxiliares nos quais algumas das funções foram sendo implementadas.

Para não ser necessário desenvolver o trabalho do zero, tomei como base o código de sala apresentado pelo professor Ítalo. Nele já haviam sido implementados arquivos referentes ao cliente e ao servidor. Dentre as funcionalidades já implementadas havia o sistema de comunicação multithread e a troca de mensagens. Dentre as alterações efetuadas no servidor temos as funções responsáveis por gerenciar os clientes e fazer o processamento das entradas. Para implementar a gestão de clientes foi utilizada uma lista na qual são armazenados dados do tipo clientes, ou seja, um identificador, seus respectivos soquetes, bem como uma lista de tags. Já o processamento das mensagens trocadas entre o servidor e o cliente é basicamente efetuado dentro da função *confirmalnscricao*, ela faz uma triagem para saber como se configura a entrada, e, portanto, que processos devem ser executados para o correto funcionamento do programa.

O desenvolvimento do software foi muito difícil, já que a descrição do mesmo foi grande, detalhada e envolvia conceitos os quais não tinha conhecimento prático prévio. Portanto, dentre os principais desafios para do desenvolvimento do trabalho estão: gerenciar os dados do cliente e fazer um processamento de mensagem eficiente, manter o cliente e o servidor conectados levando em consideração as restrições de conexão mencionadas e implementar um programa multithread.

Como uma forma de não deixar detalhes sem serem abordados, foi feito o download de uma cópia do arquivo da descrição do TP. Nesse arquivo, as partes referentes a requisitos foram grifadas de amarelo, e a medida que elas iam sendo implementadas elas iam sendo marcadas de azul para indicar que estava funcionando mas outras partes ainda poderiam influenciar o funcionamento do requisito, ou de verde significando q já havia implementado a funcionalidade por completo.

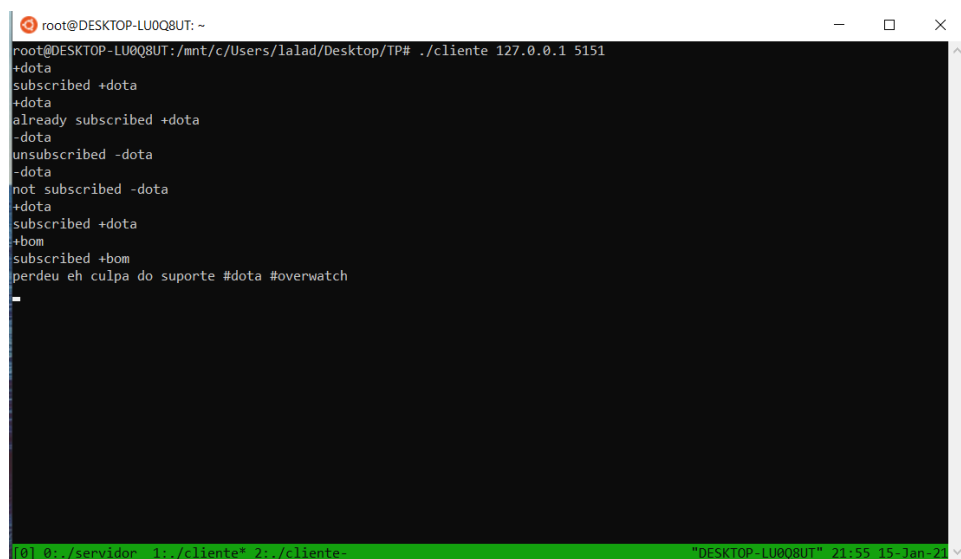
Para gerenciar os dados do cliente, e suas tags optei por usar listas encadeadas, já que elas permitem remover itens de qualquer posição sem desperdício de memória ou problemas com remoções e lixo. Para melhor efetuar a troca de mensagens entre o cliente e o servidor foi realizada uma pequena pesquisa informal sobre como geralmente funciona o publish/subscribe, tal pesquisa reforçou a ideia de usar listas e corroborou para entender melhor o processamento de threads.

Sobre o problema de manter os programas cliente e servidor funcionando até as condições de parada determinadas, a solução encontrada foi fazer uma análise visual do código. Com isso, adaptou-se o programa para que ele não encerrasse o socket dos clientes, ou do servidor antes do esperado.

Dentre os problemas enfrentados com as multithreads inclui-se o fato de desconectar clientes ao receber `##kill`, por exemplo, ou um conjunto de caracteres inválidos. O primeiro, a solução encontrada foi desconectar o servidor, já o segundo foi resolvido após as pesquisas supracitadas. Sobre a comunicação, quebrar o “tweet” em texto e tags foi uma tarefa trabalhosa, pois encontrar uma tag e entre duas tags diferentes ser obrigado a ter um espaço exigiu que eu fizesse duas verificações. A primeira ideia para solucionar este problema era usar regex, que evoluiu para o uso do strtok, e por fim desenvolvi minha função que tokeniza a mensagem e identifica as tags(vide função hashtag).

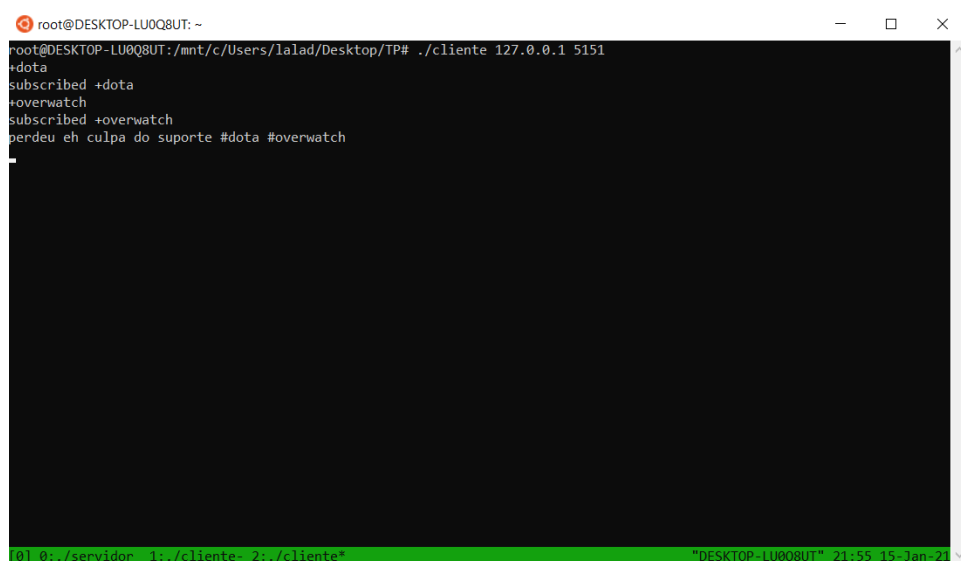
Um outro problema ao longo a construção do código foram os testes. O código foi constantemente testado ao longo de todo o seu desenvolvimento. Quando as tarefas indicadas pela descrição do trabalho haviam sido concluídas, utilizou-se do tmux para testar o servidor conectado a vários clientes. A princípio o código entregue junto com este documento cumpre os requisitos

esperados, por exemplo, respondendo a mensagem +dota com subscribed +dota, entre outras funcionalidades. No entanto, não foi possível fazer uso do script de teste. Outros colegas meus tiveram problemas similares e, no entanto, não foram adequadamente respondidos no fórum de dúvidas da disciplina. Os casos de testes deveriam ter sido atualizados no dia 14 de janeiro, no entanto, aparentemente os testes eram exatamente iguais aos de antes, já que não havia novos arquivos e eles testam apenas a inscrição e desinscrição das tags. Outros problemas com os testes é o fato de eles falarem no README “Para executar os testes, você precisa estar executando o programa `tmux` no terminal”, no entanto essa é a primeira função a ser chamada pelo arquivo .sh. O teste chega a rodar no meu computador, mas eles falham. Quando fui verificar o motivo do pelo qual manualmente o programa funciona, mas os testes não notei que o arquivo output gerado estava cheio de números ao invés da string subscribe +dota, mas no entanto, ao printar a mensagem no cliente python a mensagem chega corretamente, conforme o esperado na forma de char. Não sei o que está ocasionando este erro. Ele foi mencionado no Moodle mas não foram explicados possíveis problemas ou a forma adequada de corrigi-lo. O ambiente de programação utilizado foi o Sublime, juntamente com o terminal WSL (Linux) no Windows 10.



```
root@DESKTOP-LU0Q8UT: ~
root@DESKTOP-LU0Q8UT:/mnt/c/Users/lalad/Desktop/TP# ./cliente 127.0.0.1 5151
+dota
subscribed +dota
+dota
already subscribed +dota
-dota
unsubscribed -dota
-dota
not subscribed -dota
+dota
subscribed +dota
+bom
subscribed +bom
perdeu eh culpa do suporte #dota #overwatch
```

Figura 1: Exemplo de entradas e saídas no Cliente 1



```
root@DESKTOP-LU0Q8UT: ~
root@DESKTOP-LU0Q8UT:/mnt/c/Users/lalad/Desktop/TP# ./cliente 127.0.0.1 5151
+dota
subscribed +dota
+overwatch
subscribed +overwatch
perdeu eh culpa do suporte #dota #overwatch
```

Figura 2: Exemplo de entradas e saídas no Cliente 2

```
root@DESKTOP-LU0Q8UT: ~
ConfirmaInscricao
Mensagem enviada = +dota
Output = -subscribed +dota
-
ConfirmaInscricao
Mensagem enviada = +bom
Output = -subscribed +bom
-
ConfirmaInscricao
Mensagem enviada = perdeu eh culpa do suporte #dota #overwatch
Publish
Visitando cliente 0 de id 0
Inicio lista
dota
bom
Fim lista
Lista do cliente que enviou
Inicio lista
dota
bom
Fim lista
Visitando cliente 1 de id 1
Inicio lista
dota
overwatchDJ
Fim lista
Achou a tag dota
Output = -tag enviada-

[0] 0:./servidor* 1:./cliente 2:./cliente- "DESKTOP-LU0Q8UT" 21:55 15-Jan-21
```

Figura 3: Execução de comunicação no servidor

```
root@DESKTOP-LU0Q8UT: ~
root@DESKTOP-LU0Q8UT:/mnt/c/Users/lalad/Desktop/TP# ./cliente 127.0.0.1 5151
+dota
subscribed +dota
+dota
already subscribed +dota
-dota
unsubscribed -dota
-dota
not subscribed -dota
+dota
subscribed +dota
+bom
subscribed +bom
perdeu eh culpa do suporte #dota #overwatch
##kill
root@DESKTOP-LU0Q8UT:/mnt/c/Users/lalad/Desktop/TP#
```

Figura 4: Exemplo de encerramento dos clientes com o comando #kill