Universidade Federal do Piauí - UFPI
Campus Senador Helvídio Nunes Barros - CSHNB
Sistemas de Informação - Sistemas Distribuídos - prof. Rayner Gomes

Aluno/a:_____ Mat:_____

Terceira Avaliação - Projeto de Desenvolvimento

Contexto do Problema

Atualmente, a arquitetura baseada em microserviços está se tornando um padrão '*de facto'*. Em resumo, microsserviços tem como base que toda aplicação complexa pode ser dividida entre componentes com funções bem definidas. Com a arquitetura de microsserviços espera-se que os componentes tenham baixo acoplamento, consequentemente, a substituição deles ocorrerá de forma mais rápida dando ao sistema mais flexibilidade e facilidade arquitetural para responder às constantes demandas por adaptações.

Ao estudante de Sistemas de Informação é altamente recomendado estudar esse estilo arquitetônico, pois o mesmo possui a promessa de entregar maior capacidade de lidar com heterogeneidade tecnológica, resiliência, escalabilidade e facilidade no desenvolvimento.

Uma vantagem de "componentizar" a aplicação em componentes é que cada componente possui um ciclo de vida e pode ser implantado em um ambiente computacional distinto, como, por exemplo, em uma máquina virtual ou em um *container,* em uma máquina real ou virtual, ainda, na infraestrutura local ou remota, como exemplo na nuvem.

Uma vez que os componentes estão em unidades de execução espalhadas localmente ou globalmente, a comunicação entre elas se torna um ponto crítico no ambiente real de execução. As tecnologias de comunicação baseadas na WEB, como SOAP e REST, estão maduras, e vêm sendo aplicadas por várias aplicações, especialmente as que exigem uma comunicação planetária, como por exemplo: Netflix, Youtube, Google Drive e etc.

Apesar das tecnologias de comunicação WEB estarem maduras, o seu uso nativamente por microsserviços tem um alto poder de gerar sobrecargas nas redes e, consequentemente, perdas de mensagens. Estudamos em sistemas distribuídos que existem dilemas sobre as promessas que um sistema faz quanto aos aspectos essenciais de desenvolvimento de um sistema distribuído. O Teorema de *Brewer*, ou, simplesmente, o **Teorema CAP**, diz que um sistema distribuído pode entregar apenas duas das três seguintes características desejadas: Consistência, Disponibilidade e Tolerância de partição (em inglês, *Consistency*, *Availability*, *and Partition Tolerance*, formando as iniciais de CAP).

A **Figura 1** mostra as possíveis escolhas embutidas no teorema CAP e os banco de dados (BD) associados a cada duas características atendidas. Importante notar que os microsserviços nasceram do desejo que tornar o processo de implantação, atualização e correção mais ágeis, uma vez que ele está inserido dentro de tendências como DevOps e metodologias ágeis; portanto, os sistema baseado em microsserviços devem reforçar a disponibilidade e a partição a faltas, assim a consistência é preterida em relação às outras duas.

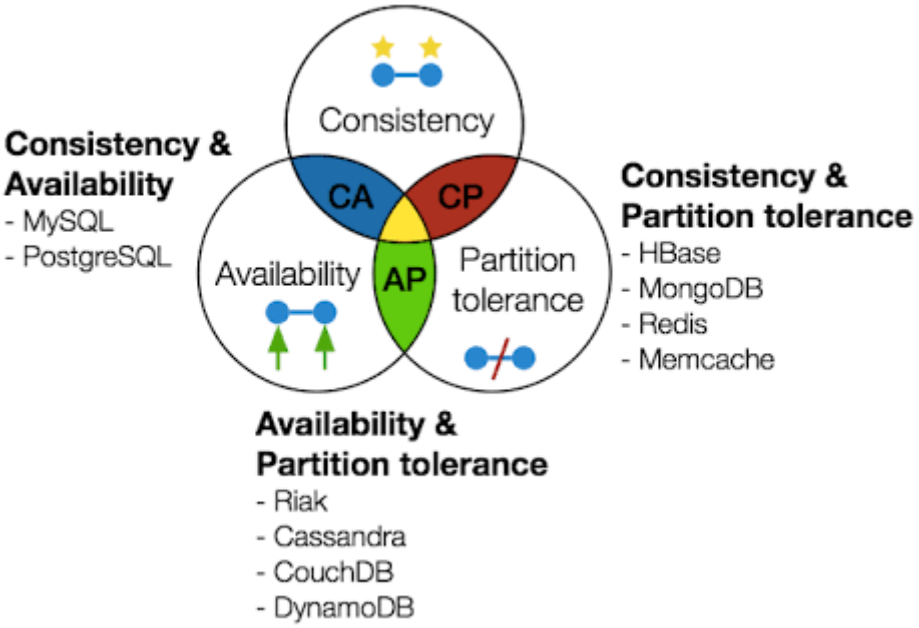

Fig 1. Teorema CAP - Exemplos de BD - Fonte: shorturl.at/ghrwy

Uma técnica para aumentar a disponibilidade e a partição da falha é utilizar um sistema de mensageria em vez da comunicação direta entre os componentes, reduzindo sobrecarga de comunicações cruzadas (conexões “macarronicas”) e possíveis rejeições de mensagens devido a sobrecarga na rede.

Um sistema de mensageria é um sistema simples que exerce um papel de *broker* na entrega de mensagens entre os componentes na rede. Existem vários sistemas de mensageria modernos, entre eles o Apache Kafka, RabbitMQ, Amazon SQS, ActiveMQ, Amazon SNS e outros; dentre esses os dois primeiros são os mais utilizados..

Ao trazer o tema de mensageria neste trabalho, damos a oportunidade de estudar um *hot topic.* Ademais, ele intercepta conceitos de engenharia de *software*, redes de computadores, sistemas distribuídos e estatística.

Sobre o Trabalho

O RabbitMQ (https://www.rabbitmq.com/) é um dos principais sistemas de mensageria, vários vídeos e tutoriais podem ser facilmente encontrados na *Internet*. Nesse trabalho, vamos desenvolver um sistema similar, contudo, bem mais simples. Manteremos a ideia primordial de um sistema de mensageria contendo os seguintes componentes: produtor, trocador, filas e consumidor. A seguir esses elementos são explicados:

- O **produtor** envia mensagens, é o elemento gerador, onde a mensagem é criada e submetida no sistema. No sistema de mensageria o produtor não envia direto para o consumidor, por outro lado, o produtor envia a mensagem ao trocador.
- O **consumidor** recebe as mensagens, porém as mensagens não são endereçadas a ele, o sistema de mensageria utiliza filas e chaves para associar os produtores e os consumidores.
- O **trocador** é o 'cérebro' do sistema de mensageria. Ele é o responsável por receber as mensagens, logo todos os produtores enviam as mensagens para ele. Uma vez que a mensagem é recebida, ele deve enviar a respectiva fila. Constantemente, o trocador deve enviar as mensagens nas filas para os consumidores. A ligação entre as mensagens e os consumidores é pelo tópico. O tópico é um texto, uma chave, utilizada para direcionar as mensagens às filas e das filas para os consumidores.
- A **mensagem** pode ser binária ou textual. Nosso sistema apenas manipula mensagens textual. Uma mensagem precisa ter dois campos, a chave/tópico e a carga útil. A chave é um texto para sua identificação, também chamada de tópico, e a carga útil refere-se a mensagem a ser transmitida do produtor para o consumidor.

Importante, o produtor e o consumidor não são processos exclusivos, ou seja, um produtor pode ser um consumidor, assim o conceito de Servante é perfeitamente aplicável. Por fim, o produtor ao enviar uma mensagem pode associar a mensagem a um tópico, ou classificá-la com "fanout". Toda mensagem fanout deverá ser enviada para todas as filas e consequentemente ela vai ser entregue a todos consumidores.

Ilustração dos Sistema de Mensageria

A figura abaixo foi produzida pela ferramenta RabbitMQ Simulator. Essa ferramenta pode ajudar a entender o funcionamento de como o sistema de mensageria funciona. A área de desenho é usada para representar a topologia de mensagens. Arrastando os elementos de mensagens da caixa de ferramentas à esquerda para a tela novos elementos podem ser adicionados ao sistema. Para conectar nós, mantenha pressionada a tecla ALT (ou tecla SHIFT) e arraste de um nó de origem para conectá-lo a um nó de destino.

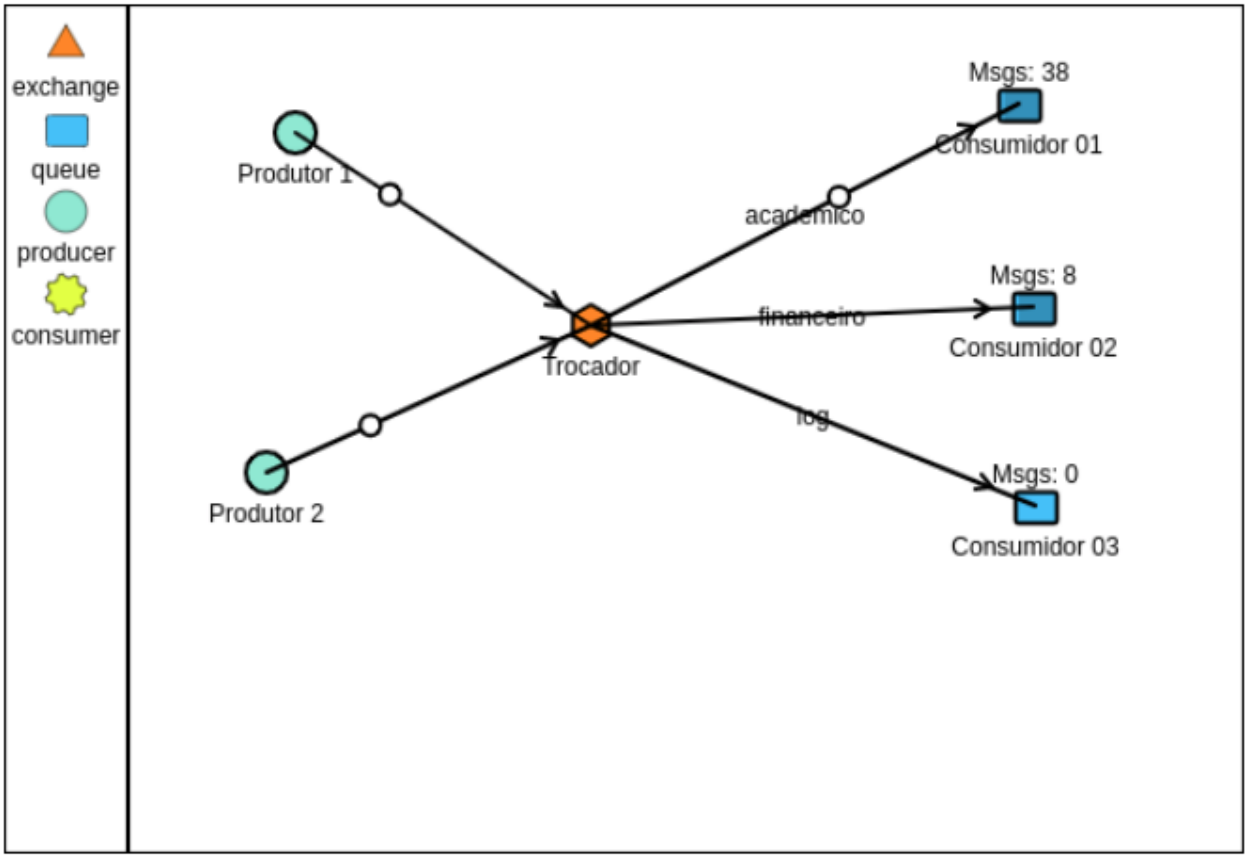

Fig 2. RabbitMQ Simulator - Fonte: http://tryrabbitmq.com

Infraestrutura de Desenvolvimento

A implementação de nosso sistema de mensageria contará com o uso de containers/Docker para simular diversos computadores. O sistema deve ser desenvolvido em Python. Nenhuma biblioteca dos sistemas de mensagens deverá ser utilizada. Logo, todos os processos serão abrigados na mesma máquina, mas cada em um c*ontainers* diferente. A Tabela 1 apresenta as especificações que devem ser seguidas.

Especificação	Opções
Sistema operacional	Windows ou Linux
Arquitetura de comunicação	TCP/IP
Linguagem de programação	Python, versão > 3.0
Comunicação entre processos	Sockets ou RPC
Interface de uso	Modo texto
Interface de teste	Modo texto

Tabela 1: Especificação da infraestrutura de execução.

Sobre o Desenvolvimento

O sistema contará com os elementos e as funcionalidades:

1. O produtor: O produtor precisará saber qual tópico as mensagens dele serão rotuladas. O rótulo pode ser passado durante a inicialização do processo, ou o processo pode perguntar durante sua inicialização. Outro parâmetro necessário é a taxa de envio das mensagens, a taxa deve estar em quantidades por minuto, por exemplo, 10 mensagens por minuto. As mensagens podem ser geradas aleatoriamente, é interessante que o

tamanho delas sejam variadas. O usuário poderá iniciar quantos produtores quiser, para isto, basta criar um novo container e inicializar um novo produtor. O produtor pode ser finalizado com o comando CTRL-C, neste caso o trocador deverá remover a Thread associada ao produtor. Semelhantemente, o consumidor pode ser finalizado com o mesmo comando, neste caso o trocador deverá remover a Thread associada ao consumidor. Por fim, do lado do trocador, quando a Thread é finalizada, ela deve emitir uma mensagem informando o seu encerramento.

2. O trocador: Este processo deverá ser o primeiro a ser executado. Suas credenciais devem ser conhecidas pelo produtor e consumidor. Portanto, deve ser um processo em um *container* exclusivo. O trocado deve gerenciar o uso de cada fila. A fim de não gerar um atraso sequencial, cada fila deve ser mantida por uma *Thread*.
3. Criação das filas. As filas podem ser criadas em dois momentos. Quando o produtor envia uma mensagem ou quando o consumidor conectar ao trocador. Em ambos os casos, o trocador ficará sabendo do tópico, caso a fila associada ao tópico não exista, ela será criada. Uma vez que a fila exista, as mensagens recebidas pelo produtor devem ser encaminhadas para as filas correspondentes.
4. Produtor de tópicos: Um produtor não precisa gerar apenas mensagens de um tópico, pode enviar a mensagens de qualquer tópico. Todas as mensagens enviadas com o tópico *fanout* serão encaminhadas pelo trocador a todas as filas.
5. Disciplina de Filas: Todas as mensagens exceto as de *fanout* devem seguir a disciplina de fila FCFS (*First Come, First Served*), contudo, as mensagens *fanout* devem seguir a disciplina LCFS (*Last Come, First Served*). Desta maneira, as mensagens *fanout* terão maior prioridade no sistema.
6. Monitoramento: O trocador, como já descrito até o momento, deverá receber as mensagens submetidas pelos produtores e encaminhar para os consumidores. A fim de reduzir o tempo de atendimento, cada conexão entre o trocador, produtor e o consumidor deverá ser salvaguardada por uma

Thread em particular. Por exemplo, se o sistema tiver 2 produtores e 2 consumidores então o trocador deverá ter no mínimo 4 Threads para lidar com a comunicação entre eles. Dessa maneira, a conexão deve ser mantida, logo o protocolo TCP deve ser utilizado. Além das Threads utilizadas para particularizar a comunicação, cada fila deve ser gerenciada por uma Thread. A Thread deve receber a mensagem das Threads ligadas aos produtores, entregar as Threads Gerenciadoras das Filas para que as mesmas sejam enfileiradas corretamente segundo os tópicos e enviadas as Threads ligadas aos consumidores. Importante notar que as filas são acessadas para inclusão e exclusão de mensagens, logo, alguma técnica de exclusão mútua deve ser implementada.

7. Monitoramento: Além das Threads descritas, o trocador deve contar com mais uma Thread para receber comandos de monitoramento. Uma porta dedicada deve ser fornecida ao usuário como meio de conexão remota. A Tabela 2 apresenta os possíveis parâmetros para obtenção de dados de monitoramento.

Parâmetro	Retorno
-p	retorna a quantidade de produtores conectados.
-c	retorna a quantidade de consumidores conectados.
-f	retorna as filas criadas.
-f <nome_fila> -t	retorna o tamanho da fila.
-f <nome_fila> -s	retorna a estatística da fila.
-s <SEGUNDOS>	retorna a estatística de todas as filas com o intervalo de SEGUNDOS. O usuário deverá sair deste modo quando digitar CTRL-C.

Tabela 2: Parâmetros para monitoramento.

A estatística na tabela compreende as seguintes informações:

- a taxa de mensagens recebidas por minuto.

- a taxa de mensagens entregues por minuto.
- o tamanho máximo que as filas alcançaram.

Arquitetura do Trocador

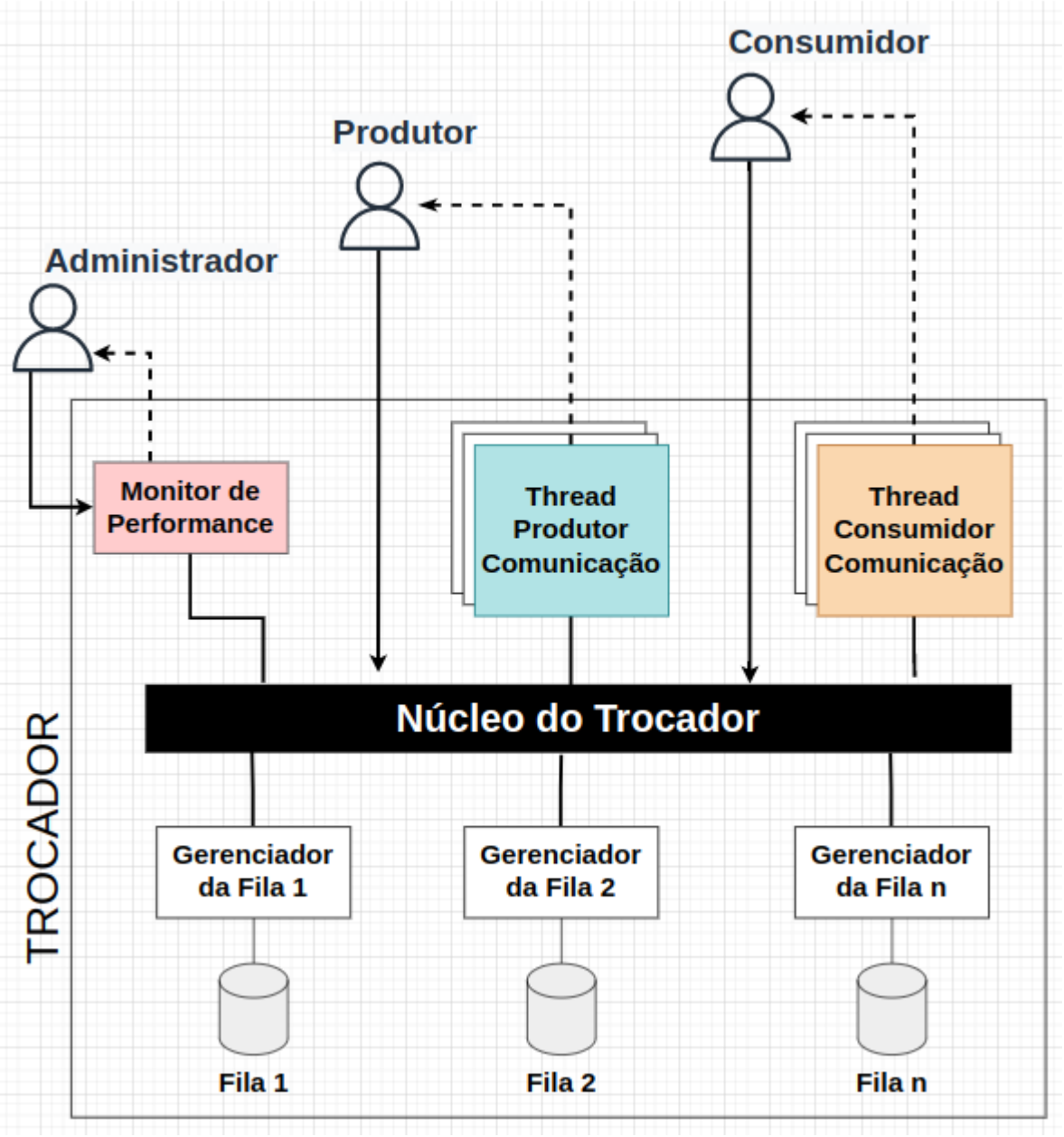

Fig 3. Arquitetura do Trocador

Relatório de Entrega

O relatório de entrega deve seguir o modelo de um artigo. Utilizar o modelo do IEEE Transactions (este modelo está disponível no Overleaf no endereço: shorturl.at/fhqrS). Logo, o relatório deverá constar:

1. **Resumo**: No máximo 300 palavras. Deve conter uma breve introdução; o que foi desenvolvido; os resultados e conclusões.
2. **Introdução**: A introdução já está incluída neste neste documento, os autores devem apenas resumir para que o resumo e a introdução não ultrapasse a uma página.
3. **Referencial Teórico**: Os autores não devem estender muito esta seção, apenas fazer uma tabela para comprar os principais *softwares* de mensagerias na atualidade. A última linha da tabela deve ser o próprio trabalho do aluno, deste modo uma comparação do projeto do aluno com os relacionados. Isso apenas dará ao leitor o que ainda precisa ser desenvolvido para que o atual *software* tenha a maturidade dos projetos atuais. Espera-se no máximo 1 página para esta seção.
4. **Desenvolvimento:** Descrever os principais aspectos da implementação do Produtor, Consumidor e do Trocador. Os autores podem utilizar ferramentas de modelagem como o UML (*Unified Modeling Language*) para descrever os componentes e suas interações. Sem limites de páginas.
5. **Execução e Resultados:** Nesta seção os autores deverão mostrar através de figuras o processo de configuração do ambiente, da execução dos códigos e das saídas. Também, devem ilustrar os testes das funcionalidades exigidas. Como a execução será feita no terminal, as telas salvas devem constar no *prompt* o nome de um aluno do grupo (ver exemplo da Figura 4). Importante, a sequência deve ser gradual e com posse dos arquivos fontes o usuário poderá executar conforme o apresentado. Sem limites de páginas.
6. **Conclusão:** As experiências obtidas neste trabalho devem ser reportadas nesta seção. A fim de guiar a construção desta seção os autores podem responder as perguntas:

a. Quais foram os desafios enfrentados?
b. O sistema funcionou como projetado?
c. Existe algum ponto de vulnerabilidade?
d. Analisando os programas relatados na Seção 3, qual o mínimo necessário para que o seu projeto possa ser usado em produção.
e. O sistema é seguro?
f. Qual a capacidade máxima do sistema? Quantos produtores e consumidores são possíveis? Há como equacionar?
g. Houve perda de mensagens? Porque?
h. A conclusão não há limites de páginas

7. **Referencial Teórico:** Deixe a referência de todas as citações, *softwares* ou dados externos.

Pontuação do Trabalho

Tarefas	Pontos
Relatório. Legível, segue o padrão exigido, segue as formatações.	5 pontos
Produtor, consumidor e trocador se comunicando.	1 ponto
Encaminhamento por tópicos.	1 ponto
Encaminhamento por “fanout”.	1 ponto
Estatísticas funcionando.	1 ponto
Produtores, Consumidores por Threads.	1 ponto.

Penalidades

Falhas	Pontos

Não utilização de *containers* para o produtor, consumidor e trocador.	1 ponto
Não utilização de Threads para gerenciar os produtores e consumidores.	1 ponto
Não utilização de Threads para gerenciar as filas.	1 ponto
O *prompt* não consta o nome do de algum aluno do grupo.	2 pontos
Fala na execução do código fonte. A execução dos comandos seguirá o apresentado na seção	1 ponto
Thread do consumidor ou produtor não informa o seu encerramento.	1 ponto

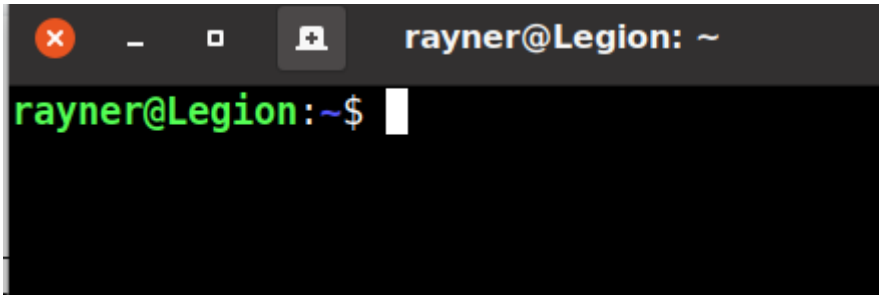

Fig 4. Exemplo do nome do aluno RAYNER no prompt de comando.

Sobre a Entrega

1. Equipe: O grupo pode ser formado por no máximo 2 alunos. Apenas um grupo poderá ter 3 alunos, caso um fique sem grupo.
2. Material de Entrega: Um arquivo ZIP com:
    a. Um arquivo formato texto (txt) com apenas o *link do* projeto no Overleaf.
    b. O PDF do relatório.
    c. E todas as fontes do projeto.

3. Data da Entrega: 03/10/2022 até às 23:55.
4. Local da Entrega: SIGAA.
5. Data da Apresentação: Caso tenha alguma falha na execução ou inconsistência no relatório, uma data será acordada entre os alunos do trabalho para as devidas explicações.

Bônus

- Imagina se o administrador dos sistema de mensageria pudesse ver os estados das filas e suas estatísticas *online* via WEB? Caso esta funcionalidade seja desenvolvida os autores terão 2 pontos de bônus.
- Projetos desenvolvidos usando RPC terão 2 pontos de bônus.
- Atenção, caso a nota seja 10, então os bônus serão transferidos para as outras notas, 2º ou 1º avaliação.

Bom Trabalho!!!!