SIN 142 – Sistemas Distribuídos
Projeto 2 - Controle Distribuído de Concorrência

Nome: Filipe Brener
Matrícula: 5952

Descrição do problema

Neste trabalho o era necessário projetar e implementar o algoritmo centralizado de exclusão mútua distribuída. O programa Coordenador deverá como já se diz no nome, coordenar o acesso dos processos do programa Criador no arquivo resultados.txt para criar, alterar ou adicionar no arquivo.

Tecnologia Usada

A tecnologia usada foi a linguagem de programação Python em sua versão mais recente 3.9.2, para a criação de ambos os programas, coordenador_de_regiao_critica.py e criador_de_processos.py. Foi usada uma abordagem de algoritmo centralizado para um sistema distribuído, onde os programas se comunicam via socket usando o protocolo TCP.

Forma com que foi abordado o problema e decisões tomadas em cada programa, destacando as threads e suas principais responsabilidades:

coordenador_de_regiao_critica.py

O coordenador_de_regiao_critica.py atua com 3 threads principais para a realização da solução do problema.

- **recv_connection** – É a thread responsável por receber as conexões dos processos do Criador, e para cada conexão nova feita é inicializado uma nova thread **handle_client.**
- **handle_client** – a thread responsável por lidar com o c*lient*, onde a relação do número de threads e o número de *clients* é de 1 para 1. O client que no caso é um processo do Criador que se conectou via socket com o Coordenador. A **handle_client** ficará responsável por receber as mensagens de REQUEST enviadas pelos clients e armazena-los em uma fila que seguirá a filosofia FIFO. A thread também ficará responsável por receber o RELEASE dos clients, logo após cada RELEASE recebido, é enviado para o próximo REQUEST da fila, caso exista, um GRANT, liberando assim o Criador entrar na região crítica. Um caso especifico a fila está vazia e o Coordenador recebe um RELEASE, este como se trata do primeiro e único da fila, recebe o GRANT do coordenador logo após ser alocado na lista, sem a necessidade de um prévio RELEASE.
- **interface –** a thread responsável pela interface é na verdade a **Main Thread** cujo possui um método que atua de forma bloqueante cujo é responsável de mostrar para

o usuário 3 opções, 1) imprimir a fila de pedidos atual, 2) imprimir quantas vezes cada processo foi atendido, 3) encerrar a execução.

É usado um monitor na thread **handle_client** para garantir a exclusão mútua entre suas múltiplas instancias à região crítica. Também é utilizado um monitor na **interface** já que, de forma concorrente às instancias da thread **handle_client**, é acessado uma lista que contém os IDs dos processos que estão na fila para receber o GRANT.

criador_de_processos.py

O criador_de_processos.py conta com apenas 2 diferentes threads para a resolução do problema, porém a thread **processo** terá N instancias, onde N é a quantidade de processos que executarão todo o processo.

- **processo –** a thread responsável por realizar a conexão com o Coordenador, executar o processo de mandar a REQUEST, receber o GRANT, realizar a escrita do horário atual e do seu PID no arquivo resultado.txt, esperar K segundos e por fim enviar o RELEASE para o Coordenador, esse processo se repete R vezes. Após a realização de todas as repetições a thread encerra a conexão com o Coordenador.
- **Main Thread** - apenas com 2 responsabilidades a **Main Thread** fica por responsável por inicializar as N threads dos processos e logo após a inicialização ela mostra no terminal a porcentagem de conclusão considerando todas as tarefas de todos os processos. Após a finalização dos processos a Main Thread indica que a execução foi completa.

Como citado anteriormente, N é o número de processos que executarão R vezes e esperarão K segundos para mandar o RELEASE, todas essas variáveis são impostas de forma ***hard coded*** pelo programador.

Testes de Estabilidade do Sistema

Por fim, foram feitos testes de estabilidade do sistema levando em consideração as 3 variáveis que interferem diretamente no tempo de execução de todo o processo. O processo é considerado como completo quando o arquivo resultado.txt contém N*R linhas que contenham um processo e o horário em que ele escreveu no arquivo esses dados. Para o cálculo do tempo de execução é subtraído a primeira hora da última hora encontradas no arquivo, por fim é gravado no mesmo o tempo de execução e os parâmetros em que foram executadas, ou seja, o valor de K, N e R.
K = tempo em segundos que o processo vai esperar antes de dar o RELEASE.
N = número de processos.
R = número de vezes que os processos vão executar.

Teste Inicial Preliminar:

- N= 2, R = 100, K = 1
- Tempo de execução: 0:00:19.184312

Bateria de Testes 1:

- R= 100 - **fixado**
- K = 1 - **fixado**

Gráfico de tempo de execução por (N):

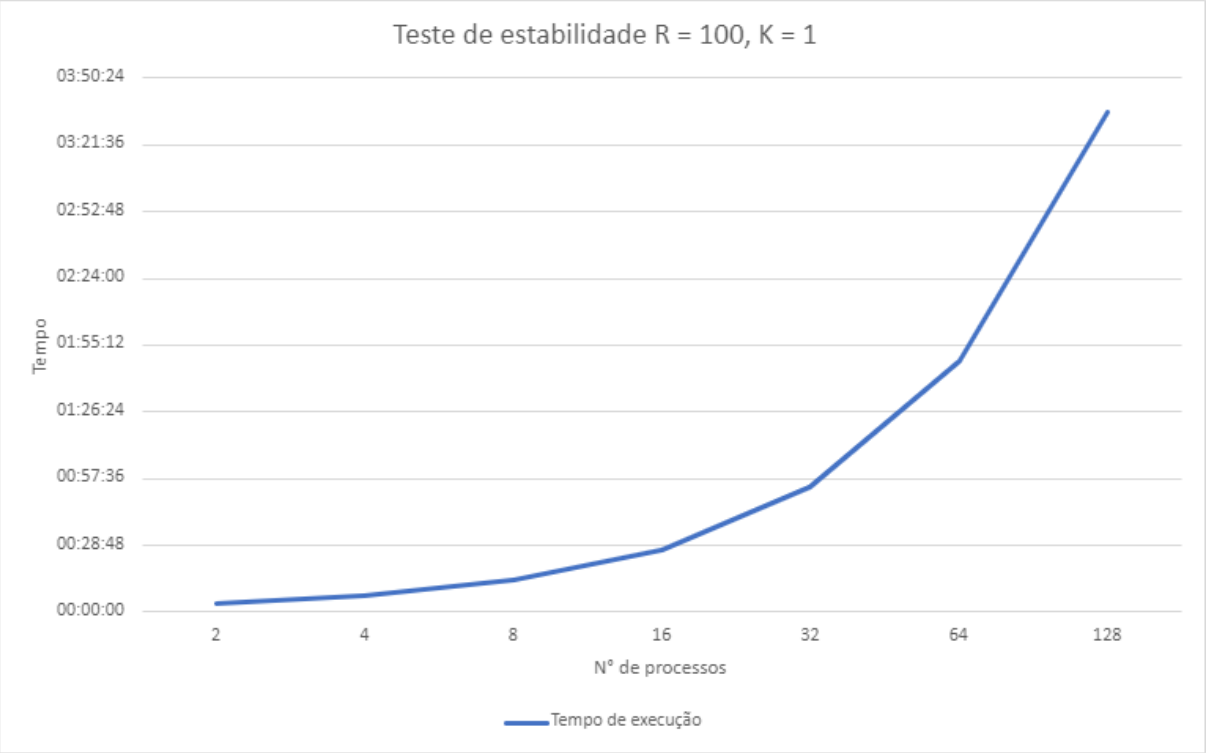


Tabela do Gráfico acima:

Teste de Estabilidade R = 100 K = 1	
N° de processos	**Tempo de execução**
2	00:03:21.586220
4	00:06:44.362419
8	00:13:29.013048
16	00:27:00.962213
32	00:54:14.120302
64	01:48:35.968695
128	03:36:00.956591

É possível observar que o tempo de execução assim como o número processos cresce de forma exponencial. Grande parcela de culpa se dá à variável K que, por mais que de uma

margem de segurança para que o sistema como um todo tenha um tempo garantido para que ele possa mandar suas mensagens, deixa o tempo de execução lento porque o sistema fica em cada passagem pelo processo ele fica 1 segundo ocioso. Mas como o teste é de estabilidade e não de escalabilidade conclui-se que o sistema, nessa configuração, se comportou bem diante dos testes.

Bateria de Testes 2:

- R= 1000 - **fixado**
- K = 0 - **fixado**

Gráfico de tempo de execução por (N):

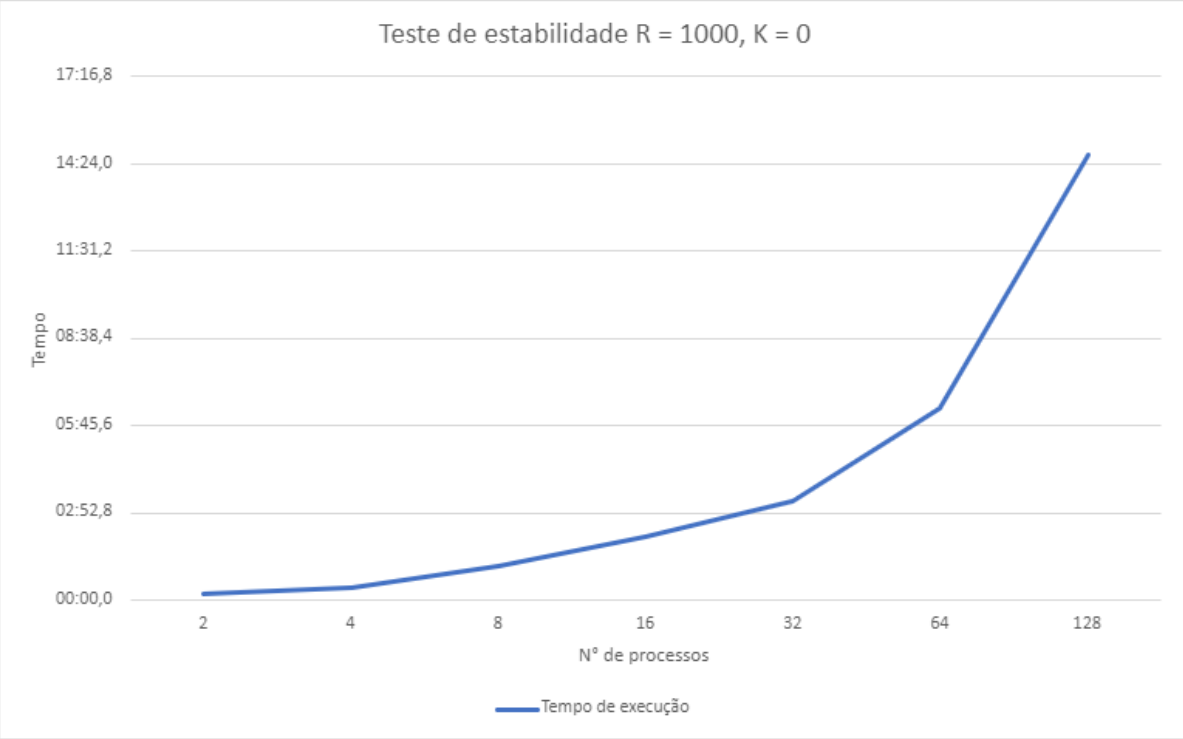


Tabela do Gráfico acima:

Teste de Estabilidade R = 1000 K = 0	
N° de processos	**Tempo de execução**
2	00:00:12.565824
4	00:00:24.818578
8	00:01:09.674660
16	00:02:05.012509
32	00:03:17.678534
64	00:06:20.407329
128	00:14:42.994379

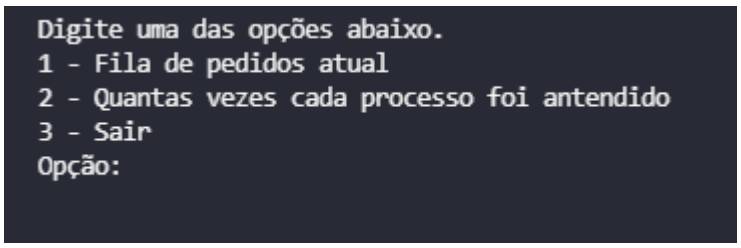
Assim como na primeira bateria de testes o tempo caminha junto com o número de processos em um padrão que cresce de forma exponencial, mas se compararmos os tempos da primeira bateria com a segunda bateria, vamos ver uma diferença considerável de

tempo, esse fator se dá devido o sistema não ficar ocioso em nenhum momento com o K valendo 0. E assim como na primeira bateria o sistema se comportou de forma satisfatória no quesito estabilidade, mesmo em casos mais extremos como no cenário de 128 processos repetindo 1000 vezes e com um intervalo de 0 segundos.

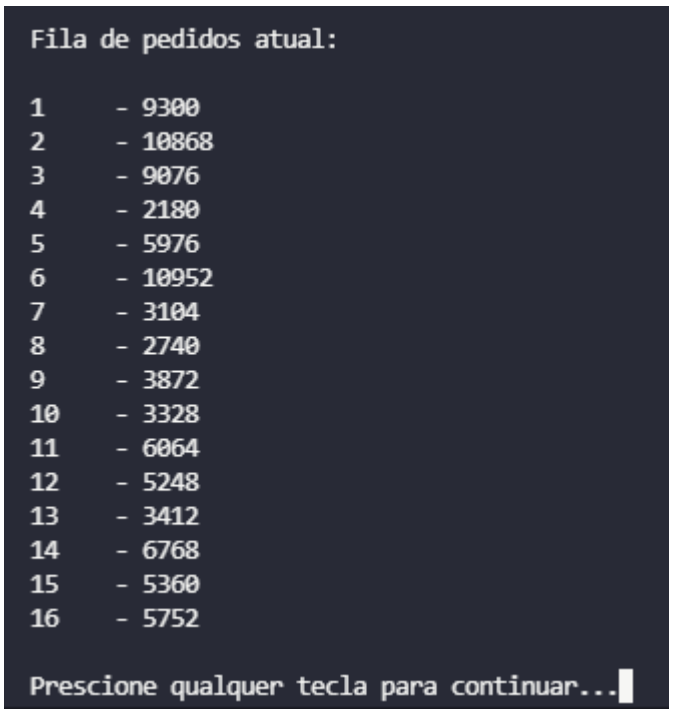
Saídas esperadas

Coordenador

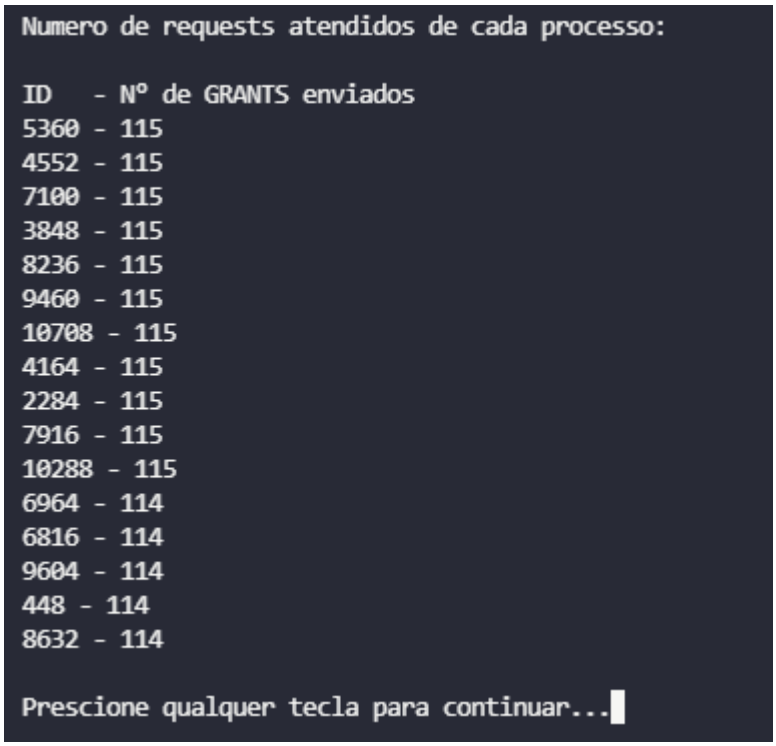
No **Coordenador** teremos 3 tipos de saídas, o **menu inicial**:



E ao usuário escolher a opção 1 no **menu inicial**, a próxima saída esperada será a **fila de pedidos atual**, onde à esquerda é exibido o número da posição na fila e à direita o ID do processo dono da REQUEST que está na fila:

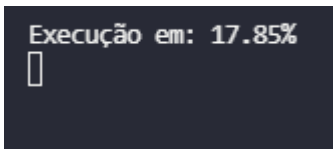


E por fim quando escolhido a opção 2 no menu inicial a saída esperada é a **quantidade de vezes que cada processo foi atendido**, ou seja, quantas vezes ele recebeu um GRANT do Coordenador:



Criador

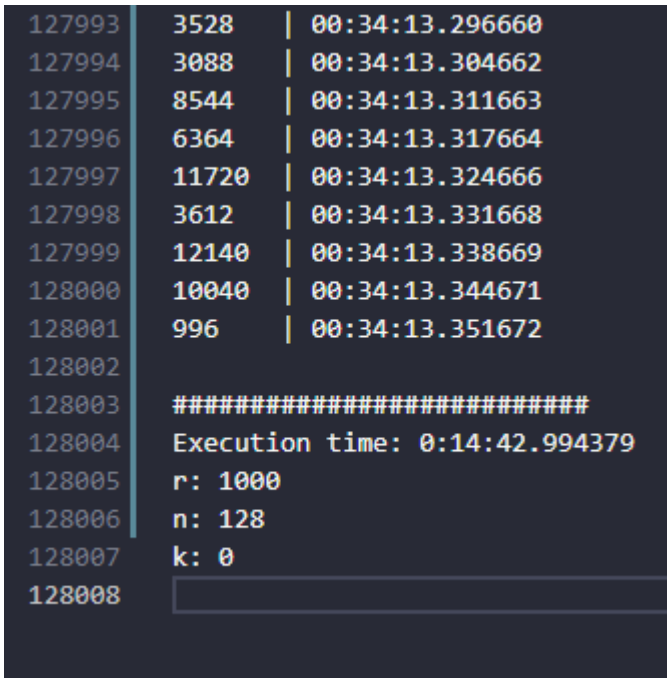
No **Criador** existe apenas a porcentagem de conclusão de todo o processo:



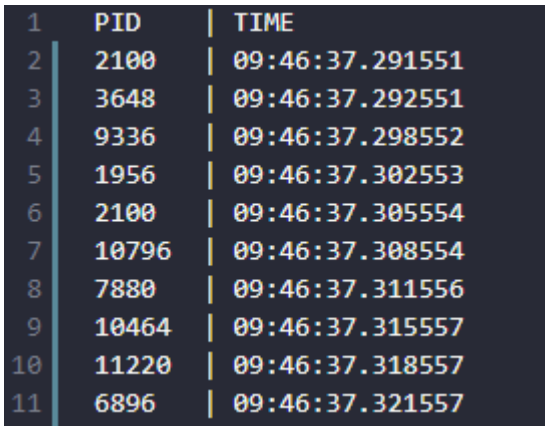
E por fim quando o **Criador** já fez todos os acessos que ele queria no **resultado.txt** é exibido que a execução foi completa:

resultado.txt

No **resultado.txt** é esperado as R*N linhas com as IDs dos processos e o horário em que foi gravado no arquivo, e ao final o cálculo de tempo de execução e em que cenário foi executado:



Pode-se observar que o esperado é 128.000 linhas, mas como exibido no arquivo, contém 128.001. Isso se dá ao fato de que a primeira linha é informada o padrão com que será gravado no arquivo:



Como executar

Para executar o sistema é necessário executar primeiro o **Coordenador**, para que ele possa começar a receber conexões via socket, e por fim executar o **Criador** que, como o **Coordenador** já estará operando, irá conseguir realizar as conexões com sucesso e executará seus processos com êxito.

Para acessar o código do projeto disponível no GitHub clique aqui!
Ou acesse: https://github.com/filipebrener/Controle-Distribuido-de-Concorrencia