

UNIVERSIDADE FEDERAL DO ABC
SISTEMA Distribuídos

Relatório Final - Sistemas Distribuídos
Práticas 3, 4, 5 e Projeto

Guilherme Gimenes Liao
Rodrigo Martins de Souza

RA: 11201920847
RA: 11201920260

Santo André, SP
São Bernardo do Campo, SP
2025

PRÁTICA 3

1. Ephemeral versus Persistent (Regular) znodes

1.1 Inicialize a interface de comando de linha (CLI): `bin/zkCli.sh -server 127.0.0.1:2181` ou `bin/zkCli.cmd`.

```
Welcome to ZooKeeper!  
2025-07-16 10:44:50,219 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1171]  
- Opening socket connection to server localhost/127.0.0.1:2181.  
2025-07-16 10:44:50,220 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1173]  
- SASL config status: Will not attempt to authenticate using SASL (unknown error)  
2025-07-16 10:44:50,226 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1007]  
- Socket connection established, initiating session, client: /127.0.0.1:50420, server: localhost/127.0.0.1:2181  
JLine support is enabled  
2025-07-16 10:44:50,255 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1454]  
- Session establishment complete on server localhost/127.0.0.1:2181, session id = 0x100002047b70000, negotiated timeout  
= 30000  
  
WATCHER::  
  
WatchedEvent state:SyncConnected type:None path:null zxid: -1  
[zk: localhost:2181(CONNECTED) 0] _
```

1.2 Crie um znode persistente para este exercício: `create /mcta025 data`

```
[zk: localhost:2181(CONNECTED) 0] create /mcta025 data  
Created /mcta025
```

1.3 Crie um outro znode persistente: `create /mcta025/ex1 data`

```
[zk: localhost:2181(CONNECTED) 1] create -e /mcta025/ex1 data  
Created /mcta025/ex1
```

1.4 Crie um znode efêmero: `create -e /mcta025/ex1/ephemeral data`

```
[zk: localhost:2181(CONNECTED) 11] create -e /mcta025/ex1/ephemeral data  
Created /mcta025/ex1/ephemeral
```

1.5 Liste os filhos do seu sistema de arquivo (na API, `getChildren`): `ls <path>`

```
[zk: localhost:2181(CONNECTED) 12] ls  
ls [-s] [-w] [-R] path  
[zk: localhost:2181(CONNECTED) 13] ls /  
[mcta025, zookeeper]  
[zk: localhost:2181(CONNECTED) 14] ls /mcta025  
[ex1]  
[zk: localhost:2181(CONNECTED) 15] ls /mcta025/ex  
Node does not exist: /mcta025/ex  
[zk: localhost:2181(CONNECTED) 16] ls /mcta025/ex1  
[ephemeral]  
[zk: localhost:2181(CONNECTED) 17]
```

1.6 Saia do cliente usando o comando `quit`

1.7 e 1.8

Reinicie a CLI e liste novamente os filhos do sistema de arquivo. O que aconteceu?
Crie um novo znode efêmero: `create -e /mcta025/ex1/ephemeral1 data`. Então, crie um filho para o znode que acabou de ser criado: `create /mcta025/ex1/ephemeral1/child data`. O que aconteceu?

Um znode efêmero não pode atuar como um "pai" para outros znodes. Eles devem estar no final de um "galho" na hierarquia do ZooKeeper, ou seja, só podem ser nós folhas no espaço hierárquico do ZooKeeper.

```
WATCHER::

WatchedEvent state:SyncConnected type:None path:null zxid: -1
[zk: localhost:2181(CONNECTED) 0] ls /
[mcta025, zookeeper]
[zk: localhost:2181(CONNECTED) 1] ls /mcta025
[ex1]
[zk: localhost:2181(CONNECTED) 2] ls /mcta025/ex1_
[]
[zk: localhost:2181(CONNECTED) 3] ls /mcta025/ex1/ephemeral1 data
'ls path [watch]' has been deprecated. Please use 'ls [-w] path' instead.
Node does not exist: /mcta025/ex1/ephemeral1
[zk: localhost:2181(CONNECTED) 4] ls -e /mcta025/ex1/ephemeral2 data
org.apache.commons.cli.UnrecognizedOptionException: Unrecognized option: -e
[zk: localhost:2181(CONNECTED) 5] create -e /mcta025/ex1/ephemeral1 data
Created /mcta025/ex1/ephemeral1
[zk: localhost:2181(CONNECTED) 6] create -e /mcta025/ex1/ephemeral1/child data
Ephemerals cannot have children: /mcta025/ex1/ephemeral1/child
[zk: localhost:2181(CONNECTED) 7]
```

2. Sequential Suffix

2.1 e 2.2:

Crie um outro znode persistente: `create /mcta025/ex2 data`.

Crie znodes sequenciais (com sufixo sequencial): `create -s /mcta025/ex2/child data`.

Repita o comando várias vezes. Qual o comprimento do sufixo?

Podemos ver o comprimento de 10 dígitos do sufixo.

```
Ephemerals cannot have children: /mcta025/ex1/ephemeral1/child
[zk: localhost:2181(CONNECTED) 7] create /mcta025/ex2 data
Created /mcta025/ex2
[zk: localhost:2181(CONNECTED) 8] create -s /mcta025/ex2/child data
Created /mcta025/ex2/child0000000000
[zk: localhost:2181(CONNECTED) 9] create -s /mcta025/ex2/child data
Created /mcta025/ex2/child0000000001
[zk: localhost:2181(CONNECTED) 10] create -s /mcta025/ex2/child data
Created /mcta025/ex2/child0000000002
[zk: localhost:2181(CONNECTED) 11] create -s /mcta025/ex2/child data
Created /mcta025/ex2/child0000000003
[zk: localhost:2181(CONNECTED) 12] create -s /mcta025/ex2/child data
Created /mcta025/ex2/child0000000004
[zk: localhost:2181(CONNECTED) 13]
```

2.3 Crie znodes sequenciais efêmeros: create -s -e /mcta025/ex2/child data. Repita o comando várias vezes

```
[zk: localhost:2181(CONNECTED) 14] create -s -e /mcta025/ex2/child data
Created /mcta025/ex2/child000000000005
[zk: localhost:2181(CONNECTED) 15] create -s -e /mcta025/ex2/child data
Created /mcta025/ex2/child000000000006
[zk: localhost:2181(CONNECTED) 16] create -s -e /mcta025/ex2/child data
Created /mcta025/ex2/child000000000007
[zk: localhost:2181(CONNECTED) 17] create -s -e /mcta025/ex2/child data
Created /mcta025/ex2/child000000000008
[zk: localhost:2181(CONNECTED) 18]
```

2.4 Remova alguns znodes com o comando: delete .

```
[zk: localhost:2181(CONNECTED) 20] delete /mcta025/ex2/child000000000001
[zk: localhost:2181(CONNECTED) 21] delete /mcta025/ex2/child000000000002
[zk: localhost:2181(CONNECTED) 22] delete /mcta025/ex2/child000000000003
[zk: localhost:2181(CONNECTED) 23] delete /mcta025/ex2/child000000000004
```

2.5 Crie mais alguns znodes sequenciais. Como fica a numeração deles?

Podemos ver que mesmo após remover alguns znodes, os seguintes continuam a contagem, pois o ZooKeeper garante que o sufixo sequencial seja sempre único para um determinado znode pai.

```
[zk: localhost:2181(CONNECTED) 20] delete /mcta025/ex2/child000000000001
[zk: localhost:2181(CONNECTED) 21] delete /mcta025/ex2/child000000000002
[zk: localhost:2181(CONNECTED) 22] delete /mcta025/ex2/child000000000003
[zk: localhost:2181(CONNECTED) 23] delete /mcta025/ex2/child000000000004
[zk: localhost:2181(CONNECTED) 24] create -s -e /mcta025/ex2/child data
Created /mcta025/ex2/child000000000009
[zk: localhost:2181(CONNECTED) 25] create -s -e /mcta025/ex2/child data
Created /mcta025/ex2/child000000000010
[zk: localhost:2181(CONNECTED) 26]
```

2.6 Crie znodes sequenciais com outro prefixo: create -s /mcta025/ex2/son data. Como fica a numeração deles?

O contador sequencial pertence ao znode pai e não ao prefixo do filho. Qualquer znode sequencial criado sob /mcta025/ex2 usará e incrementará o mesmo contador.

```
[zk: localhost:2181(CONNECTED) 26] create -s -e /mcta025/ex2/son data
Created /mcta025/ex2/son000000000011
[zk: localhost:2181(CONNECTED) 27] create -s -e /mcta025/ex2/son data
Created /mcta025/ex2/son000000000012
[zk: localhost:2181(CONNECTED) 28] create -s -e /mcta025/ex2/son data
Created /mcta025/ex2/son000000000013
[zk: localhost:2181(CONNECTED) 29]
```

2.7 Crie outros znodes sequenciais em /mcta025. A numeração (isto é, o escopo) está relacionada com a numeração anterior?

Número por znode, utiliza ordem de criação para usar o sufixo. Com modo sequencial que da a ordenação dos eventos

```
[zk: localhost:2181(CONNECTED) 29] create -s -e /mcta025/child data
Created /mcta025/child000000000003
```

3. Watches

3.1 Todas as operações de leitura no ZooKeeper tem a opção de se configurar um watch. Abra um segundo cliente em paralelo, que chamaremos de Cliente 2. O cliente inicial será chamado de Cliente 1.

```
Welcome to ZooKeeper!
2025-07-16 11:08:27,411 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1171] - Opening socket connection to server localhost/[0:0:0:0:0:0:0:1]:2181.
2025-07-16 11:08:27,412 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1173] - SASL config status: Will not attempt to authenticate using SASL (unknown error)
2025-07-16 11:08:27,418 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1007] - Socket connection established, initiating session, client: [/0:0:0:0:0:0:0:1]:50559, server: localhost/[0:0:0:0:0:0:0:1]:2181
JLine support is enabled
2025-07-16 11:08:27,428 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1454] - Session establishment complete on server localhost/[0:0:0:0:0:0:0:1]:2181, session id = 0x100002047b70002, negotiated timeout = 30000

WATCHER::

WatchedEvent state:SyncConnected type:None path:null zxid: -1
[zk: localhost:2181(CONNECTED) 0] set /mcta025/ex3 newData
```

3.2 e 3.3

Crie um novo znode através do Cliente 1: create /mcta025/ex3 data.

Configure um watch de dados do nó que acabou de ser criado no Cliente 1: get -w /mcta025/ex3

```
Created /mcta025/ex3
[zk: localhost:2181(CONNECTED) 30] create /mcta025/ex3 data
Created /mcta025/ex3
[zk: localhost:2181(CONNECTED) 31] get -w /mcta025/ex3_
data
```

3.4 No Cliente 2, modifique os dados do nó: set /mcta025/ex3 newData. O que acontece no Cliente 1?

O Watcher notifica o cliente 1 de uma mudança no diretório modificado pelo cliente 2

```
[zk: localhost:2181(CONNECTED) 32] _
WATCHER::

WatchedEvent state:SyncConnected type:NodeDataChanged path:/mcta025/ex3 zxid: 36
```

3.5 Repita no Cliente 2 o comando anterior. O que aconteceu no Cliente 1? Justifique.
Não ocorre nada, pois o watcher só é disparado uma única vez

```
[zk: localhost:2181(CONNECTED) 32] _
WATCHER::

WatchedEvent state:SyncConnected type:NodeDataChanged path:/mcta025/ex3 zxid: 36
```

3.6 É possível instalar watch de filhos de um nó. No Cliente 1: `ls -w /mcta025/ex3`.

Então, no Cliente 2, crie um novo filho dentro de `/mcta025/ex3`. O que aconteceu?

O Watcher notificou o Cliente 1, pois com o `ls -w` foi projetado para ativar quando um filho é criado ou deletado sob o nó vigiado, que ocorreu no Cliente 2.

```
WatchedEvent state:SyncConnected type:NodeDataChanged path:/mcta025/ex3 zxid: 36
ls -w /mcta025/ex3
[]
[zk: localhost:2181(CONNECTED) 33]
WATCHER::

WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/mcta025/ex3 zxid: 38
```

```
WATCHER::

WatchedEvent state:SyncConnected type:None path:null zxid: -1
[zk: localhost:2181(CONNECTED) 0] set /mcta025/ex3 newData
[zk: localhost:2181(CONNECTED) 1] set /mcta025/ex3 newDataData
[zk: localhost:2181(CONNECTED) 2] create /mcta025/ex3/child data
Created /mcta025/ex3/child
[zk: localhost:2181(CONNECTED) 3]
```

3.7 Crie um watch de dados para `/mcta025/ex3`. No Cliente 2, crie um filho em `/mcta025/ex3`. O que aconteceu? Justifique.

Essa operação modifica a estrutura de filhos do nó `/mcta025/ex3`, mas não altera os dados armazenados diretamente nele. Portanto, um evento que depende da alteração de dados não tem motivo para ser disparado.

```
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/mcta025/ex3 zxid: 38
ls -w /mcta025/ex3
[child]
[zk: localhost:2181(CONNECTED) 34] ls -w /mcta025/ex3/child
[]
[zk: localhost:2181(CONNECTED) 35]
```

3.8 Crie um watch de filhos para o nó `/mcta025/ex3` e um watch de filhos para um filho dele usando o Cliente 1. No Cliente 2, crie um filho no nó filho criado anteriormente. O que aconteceu? Agora crie um filho em `/mcta025/ex3`. O que podemos concluir dessas operações?

Um watch em `/mcta025/ex3` monitora apenas as alterações em seus filhos diretos. Ele não detecta as alterações nos filhos de seus filhos.

```
[zk: localhost:2181(CONNECTED) 34] ls -w /mcta025/ex3/child
[]
[zk: localhost:2181(CONNECTED) 35]
```

```
[zk: localhost:2181(CONNECTED) 5] create /mcta025/ex3/sibling data
Created /mcta025/ex3/sibling
[zk: localhost:2181(CONNECTED) 6]
```

```
WATCHER::

WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/mcta025/ex3 zxid: 41
```

```
[zk: localhost:2181(CONNECTED) 4] create /mcta025/ex3/child/grandchild data
Created /mcta025/ex3/child/grandchild
[zk: localhost:2181(CONNECTED) 5]
```

```
[zk: localhost:2181(CONNECTED) 35]
WATCHER::

WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/mcta025/ex3/child zxid: 40
```

3.9 Crie um watch de dados de um filho de /mcta025/ex3 e um watch de filhos para /mcta025/ex3 no Cliente 1. No Cliente 2, remova esse nó com o comando delete de um filho de /mcta025/ex3. O que aconteceu?

O watch de filhos que estava no nó pai /mcta025/ex3 é notificado porque sua lista de filhos foi alterada.

```
[zk: localhost:2181(CONNECTED) 36] ls -w /mcta025/ex3
[child, sibling]
[zk: localhost:2181(CONNECTED) 37] _

[zk: localhost:2181(CONNECTED) 6] delete /mcta025/ex3/sibling
[zk: localhost:2181(CONNECTED) 7]

WATCHER::
WatchedEvent state:SyncConnected type:NodeChildrenChanged path:/mcta025/ex3 zxid: 42
```

4. ZooKeeper Java Example

4.1 Execute o run.sh depois de acertar o valor da variável ZK para o diretório de instalação do ZooKeeper em seu computador.

```
2025-07-16 11:36:40,123 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1454]
- Session establishment complete on server localhost/[0:0:0:0:0:0:0:1]:2181, session id = 0x100002047b70003, negotiated
timeout = 4000
```

4.2 Em uma CLI, crie um znode: create /mcta025/ex4 aData. O que aconteceu?

A criação do znode serviu como um gatilho para iniciar um processo e configurar seu estado inicial.

```
create /mcta025/ex4 aData
Created /mcta025/ex4
[zk: localhost:2181(CONNECTED) 38]
```

```
timeout = 4000
Starting child
List content of file out.txt
```

```
Starting child
List content of file out.txt

C:\Users\g.liao\Downloads\MCTA025_pratica3_codigos\Codigos>type out.txt
aData
_
```

4.3 Depois, mude o valor do nó: set /mcta025/ex4 newData. O que aconteceu?

A alteração do znode permitiu reconfigurar a aplicação em tempo real, sem a necessidade de uma reinicialização manual.

```
[zk: localhost:2181(CONNECTED) 39] set /mcta025/ex4 newData
```

```
Stopping child
Starting child
List content of file out.txt

C:\Users\g.liao\Downloads\MCTA025_pratica3_codigos\Codigos>type out.txt
newData
```


4.4 Delete o nó: delete /mcta025/ex4. O que aconteceu?

A deleção do znode funcionou como um "interruptor" para desligar a aplicação de forma controlada.

```
[zk: localhost:2181(CONNECTED) 40] delete /mcta025/ex4
[zk: localhost:2181(CONNECTED) 41]
```

```
C:\Users\g.liao\Downloads\MCTA025_pratica3_codigos\Codigos>type out.txt
newData
Killing process
```

-----PRÁTICA 4-----

1 Barrier and Queue Tutorial.

1.1 Modificar a variável ZK de acordo com a instalação do Zookeeper em run barrier.sh.

```
run_barrier.bat
1  set ZK="C:\Temp\apache-zookeeper-3.9.3-bin"
```

1.2 O que significa a variável SIZE em run barrier.sh?

A quantidade de processos necessários para liberar a barreira.

```
export SIZE=2
echo "Group size = $SIZE"
```

1.3 Executar run barrier.sh. O que aconteceu?

Inicia nodes em /b1 que tem a função de sincronizar o início e fim de uma computação.

Como o número de processos definido na barreira foi 2, é necessário ter 2 processos em /b1 para dar continuidade e finalizar a operação.

[illegible]

```
WATCHER::
WatchedEvent state:SyncConnected type:None path:null zxid: -1
[zookeeper: localhost:2181(CONNECTED) 0] ls /
[zookeeper: localhost:2181(CONNECTED) 0] ls /b1
[zookeeper: localhost:2181(CONNECTED) 0] ls /b1/mcta25
[zookeeper: localhost:2181(CONNECTED) 0] ls /b1/mcta25/zookeeper
```



```
[zk: localhost:2181(CONNECTED) 3] ls /
[b1, mcta025, zookeeper]
[zk: localhost:2181(CONNECTED) 4] ls /b1
[]
```

```
[zk: localhost:2181(CONNECTED) 34] ls /b1
[L404A2-17.ad.ufabc.int.br0000000002, L404A2-17.ad.ufabc.int.br0000000003]
[zk: localhost:2181(CONNECTED) 35] ls /b1
[L404A2-17.ad.ufabc.int.br0000000002, L404A2-17.ad.ufabc.int.br0000000003]
[zk: localhost:2181(CONNECTED) 36] ls /b1
[L404A2-17.ad.ufabc.int.br0000000002, L404A2-17.ad.ufabc.int.br0000000003]
[zk: localhost:2181(CONNECTED) 37] ls /b1
[L404A2-17.ad.ufabc.int.br0000000002]
[zk: localhost:2181(CONNECTED) 38] ls /b1
[L404A2-17.ad.ufabc.int.br0000000002]
[zk: localhost:2181(CONNECTED) 39] ls /b1
[]
```

1.4 Modificar a variável ZK de acordo com a instalação do Zookeeper em run queue producer.sh.

1.5 Modificar a variável ZK de acordo com a instalação do Zookeeper em run queue consumer.sh.

1.6 O que significa a variável SIZE em run queue producer.sh? E em run queue consumer.sh?

Em producer, quantos itens serão produzidos.

Em consumer, quantos itens serão consumidos.

```
export SIZE=2
echo "Size = $SIZE"
```

1.7 Executar run queue consumer.sh e depois run queue producer.sh. O que aconteceu? Alterne a execução dos scripts.

Como o consumer tem um número definido de elementos que ele irá remover da fila, caso ele seja executado primeiro, sem nenhum elemento adicionado anteriormente, a ação fica em espera até que o número mínimo de elementos seja produzido.

Consumer:

```
- Session establishment complete on server localhost/127.0.0.1:2181, session id = 0x100002bd1fe0006, negotiated timeout = 4000
Input: localhost
Consumer
Going to wait
```

Producer adiciona elementos em /app3 e, em seguida, o Consumer retira eles da fila.

```
= 4000
Input: localhost
Consumer
List: [element0000000004, element0000000005]
Temporary value: /app3/element0000000004
Item: 10
List: [element0000000005]
Temporary value: /app3/element0000000005
Item: 11
```

```
[zk: localhost:2181(CONNECTED) 41] ls /
[app3, b1, mcta025, zookeeper]
[zk: localhost:2181(CONNECTED) 42] ls /app3
[]
```

O Consumer irá remover os elementos de menor índice:

```
[zk: localhost:2181(CONNECTED) 58] ls /app3
[element0000000006, element0000000007, element0000000008, element0000000009]
[zk: localhost:2181(CONNECTED) 59] ls /app3
[element0000000008, element0000000009]
```

Producer:

```
[ ]  
[zk: localhost:2181(CONNECTED) 47] ls /app3  
[element0000000002, element0000000003]  
[zk: localhost:2181(CONNECTED) 48]
```

2 Tutorial de lock baseado no Barrier and Queue Tutorial.

2.1 Modificar a variável ZK de acordo com a instalação do Zookeeper em run [lock.sh](#).

```
run_lock.bat  
1 set ZK="C:\Temp\apache-zookeeper-3.9.3-bin"
```

2.2 O que significa a variável WAIT em run lock.sh?

Tempo que o lock dura, no caso, 10 segundos.

```
set WAIT=10000  
@echo Wait time = $WAIT$
```

2.3 Executar várias instâncias de run lock.sh. O que aconteceu?

Quando várias instâncias de lock são executadas, o primeiro processo obtém o Lock e impede que os demais modifiquem o processo até que o lock anterior seja liberado.

```
My path name is: /lock/lock-0000000000  
List: [lock-0000000000]  
Suffix: 0, min: 0  
Lock acquired for lock-0000000000!  
Lock acquired!
```

```
My path name is: /lock/lock-0000000001  
List: [lock-0000000001]  
Suffix: 1, min: 1  
Lock acquired for lock-0000000001!  
Lock acquired!
```

```
My path name is: /lock/lock-0000000002  
List: [lock-0000000001, lock-0000000002]  
Suffix: 2, min: 1  
Watching /lock/lock-0000000001  
/lock/lock-0000000002 is waiting for a notification!  
Notification from /lock/lock-0000000001  
List: [lock-0000000002]  
Suffix: 2, min: 2  
Lock acquired for lock-0000000002!  
Lock acquired!
```

2.4 Executar várias instâncias de run lock.sh e, em seguida, matar alguma instância intermediária. O que aconteceu?

A instância intermediária notifica o próximo, que por sua vez vai refazer sua lista e definir seu antecessor como o primeiro lock.

```
My path name is: /lock/lock-0000000005  
List: [lock-0000000005, lock-0000000004, lock-0000000003]  
Suffix: 5, min: 3  
Watching /lock/lock-0000000004  
/lock/lock-0000000005 is waiting for a notification!  
Notification from /lock/lock-0000000004  
List: [lock-0000000005, lock-0000000003]  
Suffix: 5, min: 3  
Watching /lock/lock-0000000003  
/lock/lock-0000000005 is waiting for a notification!  
Not lowest sequence number! Waiting for a new notification.  
Notification from /lock/lock-0000000003  
List: [lock-0000000005]  
Suffix: 5, min: 5  
Lock acquired for lock-0000000005!  
Lock acquired!
```

PRÁTICA 5

1 Implementando eleição de coordenador.

1.1 Modificar a variável ZK de acordo com a instalação do Zookeeper em run leader.sh. Assegurar que exista um servidor do Zookeeper funcionando localmente.

```
1 set ZK="/home/ufabc/Documentos/apache-zookeeper-3.9.3-bin"
```

1.2 Executar várias instâncias de run leader.sh. O que aconteceu?

O node de menor id é eleito líder e os demais aguardam o processo do líder anterior ser encerrado e, com isso, serem eleitos em ordem.

```
ufabc@ufabc-OptiPlex-9010: ~/Documentos/Codigos
localhost/127.0.0.1:2181.
2025-07-30 08:40:40,761 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1173] - SASL config status: Will not attempt to authenticate using SASL (unknown error)
2025-07-30 08:40:40,769 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1007] - Socket connection established, initiating session, client: /127.0.0.1:54636, server: localhost/127.0.0.1:2181
2025-07-30 08:40:40,781 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1454] - Session establishment complete on server localhost/127.0.0.1:2181, session id = 0x1000010d5710004, negotiated timeout = 4000
My path name is: /election/n-0000000003 and my id is: 76476!
List: [n-0000000003]
Suffix: 3, min: 3
Become a leader: 76476!
I will die after 10 seconds!
Process 76476 died!
ufabc@ufabc-OptiPlex-9010: ~/Documentos/Codigos$

ufabc@ufabc-OptiPlex-9010: ~/Documentos/Codigos
localhost/127.0.0.1:2181.
2025-07-30 08:40:43,131 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1454] - Session establishment complete on server localhost/127.0.0.1:2181, session id = 0x1000010d5710005, negotiated timeout = 4000
Current leader with id: 76476
My path name is: /election/n-0000000004 and my id is: 420138!
List: [n-0000000004, n-0000000003]
Suffix: 4, min: 3
Watching /election/n-0000000003
/election/n-0000000004 is waiting for a notification!
List: [n-0000000004, n-0000000005]
Suffix: 4, min: 4
Become a leader: 420138!
I will die after 10 seconds!
Process 420138 died!
ufabc@ufabc-OptiPlex-9010: ~/Documentos/Codigos$

ufabc@ufabc-OptiPlex-9010: ~/Documentos/apache-zookeeper-3.9.3-bin/bin
Node does not exist: /leader
[zk: localhost:2181(CONNECTED) 3] get /leader
Node does not exist: /leader
[zk: localhost:2181(CONNECTED) 4] ls /
[election, zookeeper]
[zk: localhost:2181(CONNECTED) 5] ls /
[election, leader, zookeeper]
[zk: localhost:2181(CONNECTED) 6] get /leader
76476
[zk: localhost:2181(CONNECTED) 7] get /leader
76476
[zk: localhost:2181(CONNECTED) 8] get /leader
420138
[zk: localhost:2181(CONNECTED) 9] get /leader
Node does not exist: /leader
[zk: localhost:2181(CONNECTED) 10]
```

1.3 Executar várias instâncias de run leader.sh e, em seguida, matar alguma instância intermediária. O que aconteceu?

O processo seguinte ao que foi interrompido vai refazer a lista.

```
ufabc@ufabc-OptiPlex-9010: ~/Documentos/Codigos
localhost/127.0.0.1:2181.
2025-07-30 08:43:10,273 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1173] - SASL config status: Will not attempt to authenticate using SASL (unknown error)
2025-07-30 08:43:10,278 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1007] - Socket connection established, initiating session, client: /127.0.0.1:58346, server: localhost/127.0.0.1:2181
2025-07-30 08:43:10,294 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1454] - Session establishment complete on server localhost/127.0.0.1:2181, session id = 0x1000010d5710007, negotiated timeout = 4000
My path name is: /election/n-0000000006 and my id is: 720627!
List: [n-0000000006]
Suffix: 6, min: 6
Become a leader: 720627!
I will die after 10 seconds!
Process 720627 died!
ufabc@ufabc-OptiPlex-9010: ~/Documentos/Codigos$

ufabc@ufabc-OptiPlex-9010: ~/Documentos/Codigos
localhost/127.0.0.1:2181.
2025-07-30 08:43:12,386 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1007] - Socket connection established, initiating session, client: /127.0.0.1:58354, server: localhost/127.0.0.1:2181
2025-07-30 08:43:12,409 [myid:localhost:2181] - INFO [main-SendThread(localhost:2181):o.a.z.ClientCnxn$SendThread@1454] - Session establishment complete on server localhost/127.0.0.1:2181, session id = 0x1000010d5710008, negotiated timeout = 4000
Current leader with id: 720627
My path name is: /election/n-0000000007 and my id is: 904696!
List: [n-0000000006, n-0000000007]
Suffix: 7, min: 6
Watching /election/n-0000000006
/election/n-0000000007 is waiting for a notification!
ufabc@ufabc-OptiPlex-9010: ~/Documentos/Codigos$

ufabc@ufabc-OptiPlex-9010: ~/Documentos/apache-zookeeper-3.9.3-bin/bin
Node does not exist: /leader
[zk: localhost:2181(CONNECTED) 4] ls /
[election, zookeeper]
[zk: localhost:2181(CONNECTED) 5] ls /
[election, leader, zookeeper]
[zk: localhost:2181(CONNECTED) 6] get /leader
76476
[zk: localhost:2181(CONNECTED) 7] get /leader
76476
[zk: localhost:2181(CONNECTED) 8] get /leader
420138
[zk: localhost:2181(CONNECTED) 9] get /leader
Node does not exist: /leader
[zk: localhost:2181(CONNECTED) 10] get /leader
948055
[zk: localhost:2181(CONNECTED) 11]
```

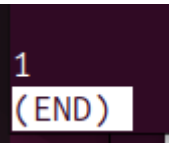
2 Criando um cluster para o Zookeeper – ensemble.

2.1 Em cada máquina abra /conf/zoo.conf. Adicione uma linha no formato server.id=host:port:port para cada máquina do ensemble, sendo que id é o identificador de cada máquina e que vai de 1 a 255.

Exemplo:server.1=172.17.36.230:2888:3888. O arquivo de configuração será igual em todas as máquinas do ensemble.

2.2 Adicione o arquivo myid contendo o id único daquele servidor no diretório especificado por dataDir em /conf/zoo.conf para cada máquina do ensemble.

```
ufabc@ufabc-OptiPlex-9010:/tmp/zookeeper$ echo 1 > myid
ufabc@ufabc-OptiPlex-9010:/tmp/zookeeper$ less myid
```



```
1
(END)
```

2.3 Inicialize o servidor do Zookeeper em várias máquinas do ensemble (pelo menos 3): /bin/zkServer.sh start. Leia as mensagens de log. Por que são geradas algumas exceções enquanto nem todos os servidores estão em execução? O que aconteceu depois que todos já estão executando? Descubra o líder enviando comandos svr no terminal de cada máquina (nc localhost 2181).

As exceções são normais e ocorrem porque, ao iniciar, cada servidor tenta se conectar imediatamente a todos os outros do cluster. Se os outros servidores ainda não estão no ar, a tentativa de conexão falha.

Assim que um número mínimo de servidores está online, eles realizam um processo de eleição de líder. Um servidor se torna o Líder e os outros se tornam Seguidores, e o cluster começa a operar normalmente.

```
2025-07-30 09:07:29,936 [myid:] - INFO [ListenerHandler-/172.17.85.228:3888:o.a.z.s.q.QuorumCnxManager$Listener$ListenerHandler@1076] - Received connection request from /172.17.85.227:49838
2025-07-30 09:07:29,940 [myid:] - INFO [WorkerReceiver[myid=1]:o.a.z.s.q.FastLeaderElection$Messenger$WorkerReceiver@391] - Notification: my state:LOOKING; n.sid:2, n.state:LOOKING, n.leader:2, n.round:0x1, n.peerEpoch:0x0, n.zxid:0x3d, message format version:0x2, n.config version:0x0
2025-07-30 09:07:29,941 [myid:] - INFO [WorkerReceiver[myid=1]:o.a.z.s.q.FastLeaderElection$Messenger$WorkerReceiver@391] - Notification: my state:LOOKING; n.sid:1, n.state:LOOKING, n.leader:2, n.round:0x1, n.peerEpoch:0x0, n.zxid:0x3d, message format version:0x2, n.config version:0x0
2025-07-30 09:07:30,142 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.QuorumPeer@906] - Peer state changed: following
2025-07-30 09:07:30,142 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.QuorumPeer@1554] - FOLLOWING
2025-07-30 09:07:30,145 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.Learner@130] - leaderConnectDelayDuringRetryMs: 100
2025-07-30 09:07:30,146 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.Learner@131] - TCP NoDelay set to: true
2025-07-30 09:07:30,146 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.Learner@132] - zookeeper.learner.asyncSending = false
2025-07-30 09:07:30,146 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.Learner@133] - zookeeper.learner.closeSocketAsync = false
2025-07-30 09:07:30,147 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.BlueThrottle@141] - Weighed connection throttling is disabled
2025-07-30 09:07:30,148 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.AuthenticationHelper@66] - zookeeper.enforce.auth.enabled = false
```

```

sudo apt install mc
ufabc@ufabc-OptiPlex-9010:/tmp/zookeeper$ nc localhost 2181
srvr
Zookeeper version: 3.9.3-c26634f34490bb0ea7a09cc51e05ede3b4e320ee, built on 2024
-10-17 23:21 UTC
Latency min/avg/max: 0/1.85/20
Received: 22
Sent: 21
Connections: 2
Outstanding: 0
Zxid: 0x100000004
Mode: follower
Node count: 8
ufabc@ufabc-OptiPlex-9010:/tmp/zookeeper$

```

```

ufabc@ufabc-OptiPlex-9010:/tmp/zookeeper$ nc localhost 2181
srvr
Zookeeper version: 3.9.3-c26634f34490bb0ea7a09cc51e05ede3b4e3
-10-17 23:21 UTC
Latency min/avg/max: 0/3.3333/65
Received: 31
Sent: 30
Connections: 2
Outstanding: 0
Zxid: 0x100000004
Mode: leader
Node count: 8
Proposal sizes last/min/max: 90/48/90

```

2.4 Derrube algum servidor e observe as mensagens de log nos demais servidores. O que aconteceu?

Quando um servidor foi derrubado, os outros detectaram a falha. Se o servidor que caiu era o líder, os servidores restantes iniciaram uma nova eleição de líder para garantir que o serviço continuasse funcionando. Durante essa breve eleição, o cluster ficou temporariamente indisponível para novas requisições.

```

[532] - Processing srvr command from /127.0.0.1:46358
^Cufabc@ufabc-OptiPlex-9010:~/Downloads/apache-zookeep

```



```

nf, -720899=telnet close, 1751217000=hash, 2003003507=wchs, 2003003504=wchp, 1684632179=dirs, 1668247155=cons, 1835955314=mntr, 1769173615=
isro, 1920298859=ruok, 1735683435=gtmk, 1937010027=stnk]]
2025-07-30 09:11:58,907 [myid:] - INFO [NIOWorkerThread-5:o.a.z.s.c.FourLetterCommands@224] - The list of enabled four letter word command
s is : [[srvr]]
2025-07-30 09:14:15,907 [myid:] - INFO [NIOWorkerThread-5:o.a.z.s.NIOServerCnxn@532] - Processing srvr command from /127.0.0.1:54854
2025-07-30 09:14:15,421 [myid:] - WARN [RecvWorker:2:o.a.z.s.q.QuorumCnxManager$RecvWorker@1402] - Connection broken for id 2, my id = 1
java.io.EOFException: null
    at java.base/java.io.DataInputStream.readFully(DataInputStream.java:210)
    at java.base/java.io.DataInputStream.readInt(DataInputStream.java:385)
    at org.apache.zookeeper.server.quorum.QuorumCnxManager$RecvWorker.run(QuorumCnxManager.java:1390)
2025-07-30 09:14:15,421 [myid:] - WARN [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.Follower@128] - Except
ion when following the leader
java.io.EOFException: null
    at java.base/java.io.DataInputStream.readFully(DataInputStream.java:210)
    at java.base/java.io.DataInputStream.readInt(DataInputStream.java:385)
    at org.apache.jute.BinaryInputArchive.readInt(BinaryInputArchive.java:98)
    at org.apache.zookeeper.server.quorum.QuorumPacket.deserialize(QuorumPacket.java:86)
    at org.apache.jute.BinaryInputArchive.readRecord(BinaryInputArchive.java:136)
    at org.apache.zookeeper.server.quorum.Learner.readPacket(Learner.java:228)
    at org.apache.zookeeper.server.quorum.Follower.followLeader(Follower.java:124)
    at org.apache.zookeeper.server.quorum.QuorumPeer.run(QuorumPeer.java:1556)
2025-07-30 09:14:15,422 [myid:] - WARN [RecvWorker:2:o.a.z.s.q.QuorumCnxManager$RecvWorker@1408] - Interrupting SendWorker thread from Rec
vWorker, sid: 2, myid: 1
2025-07-30 09:14:15,422 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.Follower@142] - Discon
nected from leader (with address: /172.17.85.227:2888). Was connected for 405261ms. Sync state: true
2025-07-30 09:14:15,422 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.Follower@286] - shutdo
wn Follower
2025-07-30 09:14:15,422 [myid:] - WARN [SendWorker:2:o.a.z.s.q.QuorumCnxManager$SendWorker@1288] - Interrupted while waiting for message o
n queue
java.lang.InterruptedIOException: null
    at java.base/java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.awaitNanos(AbstractQueuedSynchronizer.java:1770)

```

Connection broken for id 2

```

ufabc@ufabc-OptiPlex-9010:/tmp/zookeeper$ nc localhost 2181
srvr
This ZooKeeper instance is not currently serving requests
ufabc@ufabc-OptiPlex-9010:/tmp/zookeeper$

```

2.5 Levante o servidor que foi derrubado. O que aconteceu?

Ao ser levantado, o servidor que estava desligado não força uma nova eleição. Em vez disso, ele encontra o líder atual do cluster, conecta-se a ele como um seguidor e inicia um processo de sincronização para atualizar seus dados com todas as transações que ele perdeu enquanto estava offline. Após a sincronização, ele volta a ser um membro ativo do ensemble.

```

2025-07-30 09:19:32,962 [myid:] - INFO [LeaderConnector-/172.17.85.227:2888:o.a.z.s.q.Learner$LeaderConnector@380] - Successfully connecte
d to leader, using address: /172.17.85.227:2888
2025-07-30 09:19:33,029 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.QuorumPeer@920] - Peer
state changed: following - synchronization
2025-07-30 09:19:33,033 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.Learner@562] - Getting
a diff from the leader 0x100000004
2025-07-30 09:19:33,033 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.QuorumPeer@925] - Peer
state changed: following - synchronization - diff
2025-07-30 09:19:33,033 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.Learner@734] - Learner
received NEWLEADER message
2025-07-30 09:19:33,034 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.QuorumPeer@1909] - Dyn
amic reconfig is disabled, we don't store the last seen config.
2025-07-30 09:19:33,054 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.Learner@799] - Set the
current epoch to 2
2025-07-30 09:19:33,055 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.QuorumPeer@925] - Peer
state changed: following - synchronization
2025-07-30 09:19:33,055 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.Learner@805] - Sent NE
WLEADER ack to leader with zxid 200000000
2025-07-30 09:19:33,089 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.Learner@718] - Learner
received UPTODATE message
2025-07-30 09:19:33,090 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.CommitProcessor@490] -
Configuring CommitProcessor with readBatchSize -1 commitBatchSize 1
2025-07-30 09:19:33,090 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.CommitProcessor@451] -
Configuring CommitProcessor with 4 worker threads.
2025-07-30 09:19:33,090 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.FollowerRequestProcess
or@59] - Initialized FollowerRequestProcessor with zookeeper.follower.skipLearnerRequestToNextProcessor as false
2025-07-30 09:19:33,091 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.Learner@828] - 0 txns
have been logged asynchronously
2025-07-30 09:19:33,091 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.Learner@833] - 0 txns
have been committed
2025-07-30 09:19:33,092 [myid:] - INFO [QuorumPeer[myid=1](plain=[0:0:0:0:0:0:0:0]:2181)(secure=disabled):o.a.z.s.q.QuorumPeer@920] - Peer
state changed: following - broadcast
2025-07-30 09:19:34,166 [myid:] - INFO [NIOWorkerThread-6:o.a.z.s.q.Learner@161] - Revalidating client: 0x100003f5eb70000

```



```
ufabc@ufabc-OptiPlex-9010:/tmp/zookeeper$ nc localhost 2181
srvr
Zookeeper version: 3.9.3-c26634f34490bb0ea7a09cc51e05ede3b4e
-10-17 23:21 UTC
Latency min/avg/max: 0/0.0/0
Received: 1
Sent: 0
Connections: 1
Outstanding: 0
Zxid: 0x200000000
Mode: leader
Node count: 8
Proposal sizes last/min/max: -1/-1/-1
```

```
ufabc@ufabc-OptiPlex-9010:/tmp/zookeeper$ nc localhost 2181
srvr
Zookeeper version: 3.9.3-c26634f34490bb0ea7a09cc51e05ede3b4e320ee, built on 2024
-10-17 23:21 UTC
Latency min/avg/max: 0/0.3333/1
Received: 5
Sent: 4
Connections: 2
Outstanding: 0
Zxid: 0x200000001
Mode: follower
Node count: 8
ufabc@ufabc-OptiPlex-9010:/tmp/zookeeper$
```

```
2025-07-30 09:20:12,117 [myid:] - INFO [NIOWorkerThread-6:o.a.z.s.NIOServerCnxn@532] - Processing srvr command from /127.0.0.1:51780
```

-----PROJETO FINAL-----

Aplicação Proposta: Formador de Textos Distribuído com ZooKeeper

1. Visão Geral da Aplicação

A aplicação proposta, "Formador de Textos Distribuído", é um sistema de linha de comando multiusuário que permite a um grupo de participantes colaborar para construir um texto, palavra por palavra. O uso do Apache ZooKeeper serve como um serviço de coordenação centralizado, garantindo que as interações entre os múltiplos processos independentes ocorram de forma ordenada, consistente e sem conflitos.

O sistema designa papéis (um Moderador e vários Clientes), controla o fluxo da "rodada" de submissões e utiliza primitivas de coordenação distribuída para gerenciar o estado compartilhado da aplicação.

2. Arquitetura e Primitivas Utilizadas

O projeto é fundamentado em quatro primitivas de sincronização principais, cada uma implementada em sua respectiva classe Java e orquestrada pelo fluxo principal em Main.java.

2.1. Barreira de Sincronização (Barrier.java)

- **Propósito:** Garantir que a aplicação só inicie sua lógica principal após um número mínimo de participantes (processos) ter se conectado. Ela atua como um portão para sincronizar o início da computação.
- **Implementação:** A classe Barrier.java cria um nó pai persistente em /barrier. Cada processo que chama o método enter() cria um nó filho efêmero e sequencial e fica em estado de espera até que a quantidade de nós filhos atinja o tamanho (size) definido na inicialização.
- **Uso no Projeto:** É a primeira etapa do Main.java. Todos os processos são bloqueados na chamada barrier.enter(), e só prosseguem para a eleição de líder quando o quórum é atingido.

```
Conectado à barreira com o nó: /barrier/node-0000000000
Aguardando mais participantes... (1/3)
```

O primeiro participante se conecta ao servidor e espera a liberação da barreira.

```
[zk: localhost:2181(CONNECTED) 0] ls /
[zookeeper]
[zk: localhost:2181(CONNECTED) 1] ls /
[barrier, zookeeper]
[zk: localhost:2181(CONNECTED) 2] ls /barrier
[node-0000000000]
```

O nó pai barrier e o filho efêmero 0 são criados.

```
[zk: localhost:2181(CONNECTED) 6] ls /barrier
[node-0000000000, node-0000000001, node-0000000002]
```

Outros 2 participantes se conectam ao servidor.

```
Barreira superada! Iniciando eleição...
```

Com o tamanho definido atingido a barreira é liberada.

2.2. Eleição de Líder (Leader.java)

- Propósito: Atribuir de forma precisa e sem ambiguidades o papel de "Moderador" a um único processo do grupo. Isso centraliza as tarefas administrativas e de finalização da rodada.
- Implementação: A classe Leader.java implementa um algoritmo de eleição padrão. Cada processo cria um nó efêmero-sequencial em /election. O método elect() determina o líder verificando qual processo possui o nó com o menor número sequencial.
- Uso no Projeto: Executada logo após a superação da barreira. O resultado da chamada leader elect() no Main.java define o papel do processo para o resto de sua execução, determinando qual menu de opções será exibido.

```
[zk: localhost:2181(CONNECTED) 7] ls /election
[n-0000000000, n-0000000001, n-0000000002]
```

Após a barreira ser rompida, é iniciado o processo de eleição.

```
--- Eleição concluída. Você é o MODERADOR ---
```

Um dos participantes é designado o moderador.

```
Opções de Moderador:
1 - Verificar palavras restantes
2 - Verificar se o texto pode ser enviado
3 - Enviar texto final e reiniciar rodada
4 - Ver fila de palavras
9 - Sair
Sua escolha:
```

Esse participante tem acesso a uma série de comandos.

```
Opções de Cliente:
0 - Enviar palavra
9 - Sair
Sua escolha:
```

Os outros têm acesso a outros.

2.3. Fila Distribuída (Queue.java)

- Propósito: Fornecer uma estrutura de dados compartilhada para que os clientes possam submeter palavras de forma ordenada e para que o Moderador possa consumi-las na ordem correta.
- Implementação: A classe Queue.java gerencia o znode /queue.

- Produce: O método `produce(String palavra)` cria um nó persistente-sequencial em `/queue`, armazenando a palavra como dado. A ordem é garantida pelo sufixo numérico do nó.
- Consume: O método `consumeAllStrings()` busca todos os nós filhos, ordena-os numericamente, lê os dados de cada um e os apaga em sequência.
- Uso no Projeto: Clientes usam `produce()` para enviar palavras. O Moderador usa `consumeAllStrings()` para gerar o texto final e `getAllStrings()` para inspecionar a fila.

```
Digite a palavra: Sistemas
```

```
Digite a palavra: distribuidos
```

Os participantes vão enviando suas palavras.

```
[zk: localhost:2181(CONNECTED) 19] ls /queue
[element0000000000, element0000000001]
```

Nós são criados em `/queue`.

```
[zk: localhost:2181(CONNECTED) 20] get /queue/element0000000000
Sistemas
[zk: localhost:2181(CONNECTED) 21] get /queue/element0000000001
distribuidos
```

Cada um deles contendo, de forma ordenada, as palavras enviadas.

```
Fila atual de palavras: [Sistemas, distribuidos]
```

O líder pode ver a fila atual de palavras. E se o mínimo de palavra foi atingido pode enviar.

2.4. Lock Distribuído (Lock.java)

- Propósito: Garantir exclusão mútua em uma seção crítica do código, prevenindo condições de corrida.
- Implementação: A classe `Lock.java` implementa uma receita de lock de exclusão. O método `lock()` cria um nó efêmero-sequencial em `/lock` e aguarda sua vez na fila (baseado no número sequencial). O `unlock()` apaga o nó, liberando o próximo processo da fila.
- Uso no Projeto: É utilizado pelos Clientes ao enviar uma palavra (case 0 em `Main.java`). O lock é adquirido antes de verificar se o limite de palavras foi atingido e se o cliente já enviou uma palavra. Isso torna a operação "verificar-e-enviar" atômica, prevenindo que mais palavras que o permitido sejam enviadas.

```
Sua escolha: 0
Lock adquirido!
Digite a palavra:
```

Um participante escolhe "Enviar palavra" e obtém o lock.

```
Sua escolha: 0
Aguardando envio por outro grupo...
```

Outro participante escolhe “Enviar palavra” antes do primeiro ter a enviado, e agora espera o envio do outro para poder enviar sua palavra.

```
Palavra enviada com sucesso!  
Lock liberado!
```

O primeiro envia uma palavra liberando o lock.

```
Lock adquirido!  
Digite a palavra:
```

O segundo obtém o lock e só agora pode enviar sua palavra.

3. Funcionalidades da Aplicação

O sistema opera com dois papéis distintos, cada um com seu próprio conjunto de funcionalidades, além de uma funcionalidade automática gerenciada pelo Moderador.

3.1. Funcionalidades do Moderador (Menu Manual)

- Verificar palavras restantes: Exibe quantas palavras ainda precisam ser enviadas para atingir o mínimo.
- Verificar se o texto pode ser enviado: Informa se a condição de minWords foi atendida.
- Enviar texto final e reiniciar rodada: Se o mínimo de palavras foi atingido, consome todas as palavras da fila, exibe o texto final, limpa o registro de submissões e reinicia a barreira, encerrando o programa.
- Ver fila de palavras: Exibe o conteúdo atual da fila sem consumir os itens.

3.2. Funcionalidade Automatizada do Moderador

Em segundo plano, uma thread monitora o número de submissões (/submissions). Quando todos os clientes conectados na barreira tiverem enviado uma palavra, o registro de submissões é automaticamente limpo. Isso permite que os clientes enviem mais de uma palavra por rodada, mas apenas uma por vez, garantindo a participação de todos.

3.3. Funcionalidades do Cliente (Menu Manual)

- Enviar palavra: Permite ao cliente submeter uma palavra. O sistema impede o envio caso o mínimo de palavras da rodada já tenha sido atingido ou se o cliente já tiver enviado uma palavra na "vez" atual.
- Sair: Encerra o processo do cliente e o remove da barreira.

4. Instruções de Execução

4.1. Pré-requisitos

- Java Development Kit (JDK) instalado e configurado no sistema.
- Apache ZooKeeper 3.9.3 baixado, configurado e em execução. O servidor deve estar acessível no endereço especificado no arquivo .bat.

4.2. Configuração e Execução

A execução é gerenciada pelo arquivo projeto.bat.

Configurar projeto.bat: Antes de executar, abra o arquivo projeto.bat em um editor de texto e ajuste as seguintes variáveis:

- set ZK=...: Caminho para a pasta de instalação do Apache ZooKeeper.
- set ZK_ADDR=...: Endereço do servidor ZooKeeper (ex: localhost).
- set BARRIER_SIZE=...: Número de participantes necessários para iniciar a aplicação (ex: 3).
- set MIN_WORDS=...: Número mínimo de palavras para que o texto final possa ser enviado (ex: 10).

Executar a Aplicação:

- Inicie o Servidor ZooKeeper: zkServer.bat (localizado no /bin do local de instalação do “apache-zookeeper-3.9.3-bin”)
- Execute o projeto.bat quantas vezes for necessário, definido pelo tamanho da barreira.
- Opcional: Para verificar os znodes e a criação dos nós em cada primitiva utilizada, execute e zkCli.bat e navegue ulizando o comando “ls /<path>”

O script irá primeiro compilar os arquivos .java e depois iniciar a aplicação. Os processos ficarão aguardando na barreira até que o número definido em BARRIER_SIZE seja atingido. Após isso, a eleição de líder ocorrerá e os menus serão exibidos.