

Implementação de Algoritmos Distribuídos

ATIVIDADE 2

Brenda Martinez
Análise e Desenvolvimento de Sistemas
março/2025

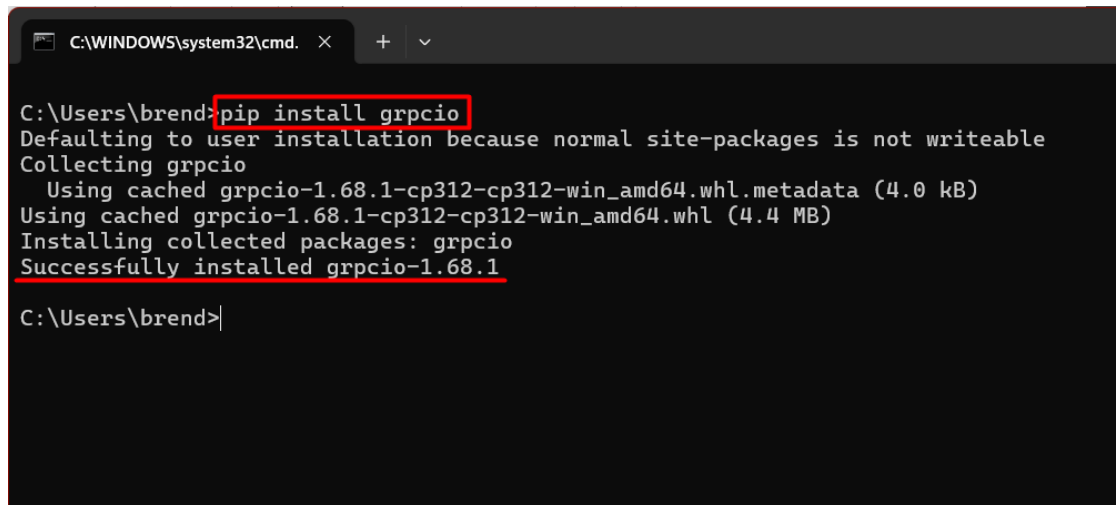
Sumário

Tecnologias.....	1
Questão 1 – Clocks e Sincronização de Tempo	2
Como Executar o Sistema (Windows).....	2
Explicação do Código	2
Questão 2 – Estado Global e Captura de Estado.....	4
Como Executar o Sistema (Windows).....	4
Explicação do Código	4
Questão 3 – Algoritmos de Eleição - Bully	7
Como Executar o Sistema (Windows).....	7
Explicação do Código	8
Questão 4 – Detecção de Falhas em Sistemas Distribuídos.....	10
Como Executar o Sistema (Windows).....	10
Explicação do Código	12

Tecnologias

- Essa documentação foi produzida utilizando o sistema operacional **WINDOWS**.
- Linguagem: Python 3
- Bibliotecas: time, random, socket, threading, sys
- Instalando uma biblioteca python (Windows):
 - Necessário ter o python previamente instalado

- Abrir o CMD e executar o comando “`pip install {nome da biblioteca}`”



```
C:\WINDOWS\system32\cmd. x + v

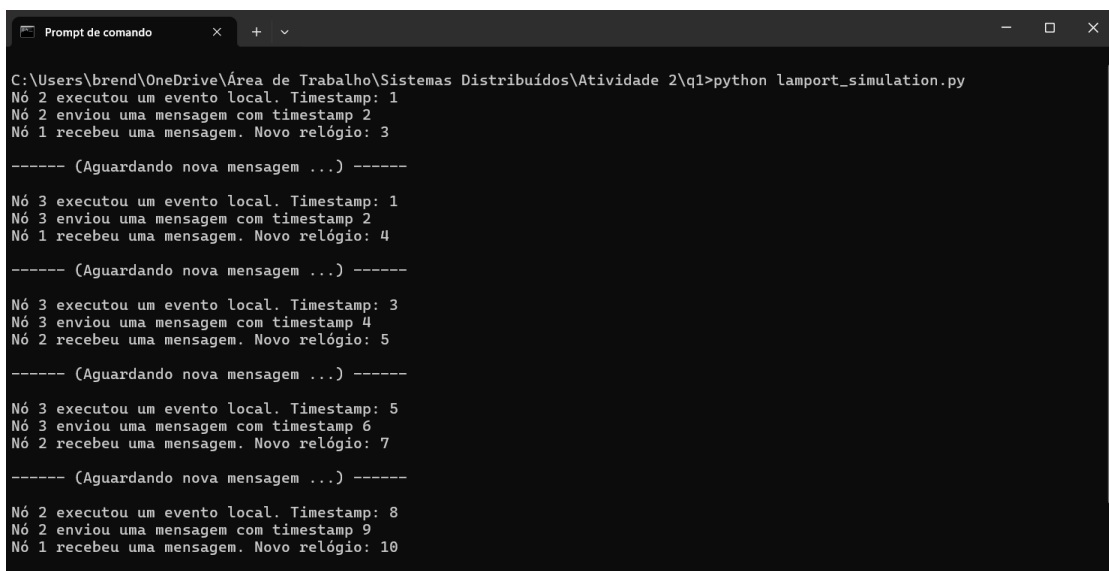
C:\Users\brend>pip install grpcio
Defaulting to user installation because normal site-packages is not writeable
Collecting grpcio
  Using cached grpcio-1.68.1-cp312-cp312-win_amd64.whl.metadata (4.0 kB)
Using cached grpcio-1.68.1-cp312-cp312-win_amd64.whl (4.4 MB)
Installing collected packages: grpcio
Successfully installed grpcio-1.68.1

C:\Users\brend>
```

Questão 1 – Clocks e Sincronização de Tempo

COMO EXECUTAR O SISTEMA (WINDOWS)

1. Abra o CMD do Windows e entre na pasta onde está o arquivo “`lamport_simulation.py`”.
2. Execute o sistema utilizando o comando “`python lamport_simulation.py`”



```
Prompt de comando x + v

C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuídos\Atividade 2\q1>python lamport_simulation.py
Nó 2 executou um evento local. Timestamp: 1
Nó 2 enviou uma mensagem com timestamp 2
Nó 1 recebeu uma mensagem. Novo relógio: 3

----- (Aguardando nova mensagem ...) -----

Nó 3 executou um evento local. Timestamp: 1
Nó 3 enviou uma mensagem com timestamp 2
Nó 1 recebeu uma mensagem. Novo relógio: 4

----- (Aguardando nova mensagem ...) -----

Nó 3 executou um evento local. Timestamp: 3
Nó 3 enviou uma mensagem com timestamp 4
Nó 2 recebeu uma mensagem. Novo relógio: 5

----- (Aguardando nova mensagem ...) -----

Nó 3 executou um evento local. Timestamp: 5
Nó 3 enviou uma mensagem com timestamp 6
Nó 2 recebeu uma mensagem. Novo relógio: 7

----- (Aguardando nova mensagem ...) -----

Nó 2 executou um evento local. Timestamp: 8
Nó 2 enviou uma mensagem com timestamp 9
Nó 1 recebeu uma mensagem. Novo relógio: 10
```

EXPLICAÇÃO DO CÓDIGO

Relógio de Lamport

O código para esse sistema é composto de dois arquivos: o arquivo [LamportClock.py](#), que define a classe LamportClock: uma representação do relógio de Lamport; e o

arquivo [lamport_simulation.py](#), que realiza a simulação do sistema de disparo de eventos utilizando o relógio de Lamport.

[LamportClock.py](#)

Essa classe representa o relógio de Lamport. Ela simula um contador lógico em um nó de um sistema distribuído. A explicação para cada linha do código consiste nos comentários da imagem abaixo.

```
1 class LamportClock:
2     def __init__(self, node_id):
3         self.node_id = node_id # identificador unico
4         self.clock = 0 # Contador logico. OBS timestamp é o valor de self.clock no momento do envio de uma mensagem
5
6     def increment(self):
7         self.clock += 1 # incrementa o relógio
8
9     def send_event(self):
10        self.increment() # incrementa o relógio antes do envio
11        print(f"Nó {self.node_id} enviou uma mensagem com timestamp {self.clock}") # simulacao da mensagem
12        return self.clock # retorna o timestamp da mensagem
13
14    def receive_event(self, received_clock):
15        self.clock = max(self.clock, received_clock) + 1 # atualiza o relógio p garantir a ordem causal
16        print(f"Nó {self.node_id} recebeu uma mensagem. Novo relógio: {self.clock}")
17
```

[lamport_simulation.py](#)

Nesse arquivo, é executada a simulação da troca de mensagens entre 2 nós aleatórios. Seu objetivo é simular a execução de eventos em um sistema distribuído onde três nós utilizam Relógios de Lamport para manter a ordenação dos eventos.

A explicação para cada linha do código consiste nos comentários da imagem abaixo.

```
1 import time
2 import random
3 from LamportClock import LamportClock
4
5 def lamport_simulation():
6     node1 = LamportClock(1)
7     node2 = LamportClock(2)
8     node3 = LamportClock(3)
9
10    # simulacao de eventos
11    for _ in range(5):
12        sender, receiver = random.sample([node1, node2, node3], 2) # seleciona aleatoriamente 2 nos
13        sender.increment() # incrementa o relógio do sender
14        print(f"Nó {sender.node_id} executou um evento local. Timestamp: {sender.clock}") # realiza evento
15        msg_clock = sender.send_event() # sender envia o evento e retorna o timestamp
16        receiver.receive_event(msg_clock) # receiver recebe a mensagem e atualiza o relógio
17        print("\n----- (Aguardando nova mensagem ...) ----- \n")
18        time.sleep(1)
19
20 if __name__ == "__main__":
21     lamport_simulation()
22
```

Questão 2 – Estado Global e Captura de Estado

COMO EXECUTAR O SISTEMA (WINDOWS)

1. Abra o CMD do Windows e entre na pasta onde está o arquivo “chandy_lamport_snapshot_sim.py”.
2. Execute o sistema utilizando o comando “python chandy_lamport_snapshot_sim.py”

```
Prompt de Comando

C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuidos\Atividade 2\q2>python chandy_lamport_snapshot_sim.py

Recebendo mensagens. . .
. . . . .

Enviando mensagens. . .

Jogador 1 enviou 'Ataque' para Jogador 2
Jogador 2 recebeu 'Ataque' de Jogador 1
Jogador 2 enviou 'Defesa' para Jogador 3
Jogador 3 recebeu 'Defesa' de Jogador 2
Jogador 3 enviou 'Toma Poca' para Jogador 1
Jogador 1 recebeu 'Toma Poca' de Jogador 3

[SNAPSHOT] Jogador 2 iniciou a captura de estado global

Jogador 2 enviou MARCADOR para Jogador 1
Jogador 1 recebeu MARCADOR de 2 e salvou estado: {'vida': 63, 'pontuacao': 1468, 'itens': ['arco', 'poção']}
Jogador 1 enviou MARCADOR para Jogador 2
Jogador 2 ignorou MARCADOR de 1, pois já recebeu anteriormente
Jogador 1 enviou MARCADOR para Jogador 3
Jogador 3 recebeu MARCADOR de 1 e salvou estado: {'vida': 70, 'pontuacao': 611, 'itens': ['flecha', 'flecha']}
Jogador 3 enviou MARCADOR para Jogador 1
Jogador 1 ignorou MARCADOR de 3, pois já recebeu anteriormente
Jogador 3 enviou MARCADOR para Jogador 2
Jogador 2 ignorou MARCADOR de 3, pois já recebeu anteriormente
Jogador 2 enviou MARCADOR para Jogador 3
Jogador 3 ignorou MARCADOR de 2, pois já recebeu anteriormente
```

```
Prompt de Comando

Jogador 3 enviou MARCADOR para Jogador 2
Jogador 2 ignorou MARCADOR de 3, pois já recebeu anteriormente
Jogador 2 enviou MARCADOR para Jogador 3
Jogador 3 ignorou MARCADOR de 2, pois já recebeu anteriormente

Recebendo mensagens. . .
. . . . .

Enviando mensagens. . .

Jogador 1 enviou 'Toma Poca' para Jogador 3
Jogador 3 recebeu 'Toma Poca' de Jogador 1
Jogador 2 enviou 'Defesa' para Jogador 3
Jogador 3 recebeu 'Defesa' de Jogador 2
Jogador 3 enviou 'Vida baixa. Use a poção!' para Jogador 2
Jogador 2 recebeu 'Vida baixa. Use a poção!' de Jogador 3
Jogador 3 enviou 'Ataque' para Jogador 1
Jogador 1 recebeu 'Ataque' de Jogador 3

[ESTADO GLOBAL] Jogador 1: Estado = {'vida': 63, 'pontuacao': 1468, 'itens': ['arco', 'poção']}, Canais = {2: [], 3: ['Ataque']}
[ESTADO GLOBAL] Jogador 2: Estado = {'vida': 91, 'pontuacao': 253, 'itens': ['escudo', 'espada']}, Canais = {1: [], 3: ['Vida baixa. Use a poção!']}
[ESTADO GLOBAL] Jogador 3: Estado = {'vida': 70, 'pontuacao': 611, 'itens': ['flecha', 'flecha']}, Canais = {1: ['Toma Poca'], 2: ['Defesa']}
```

EXPLICAÇÃO DO CÓDIGO

Algoritmo de Chandy-Lamport

O código para esse sistema também é composto por dois arquivos: o arquivo [JogadorProcess.py](#), que define a classe JogadorProcess: uma simulação de um processo, utilizando algumas características de um jogador em um jogo, conforme exemplo dado em sala de aula; e o arquivo [chandy_lamport_snapshot_sim.py](#), que realiza a simulação

de uma troca de mensagens entre jogadores (processos) antes e após o início de um snapshot, ou captura de estado global.

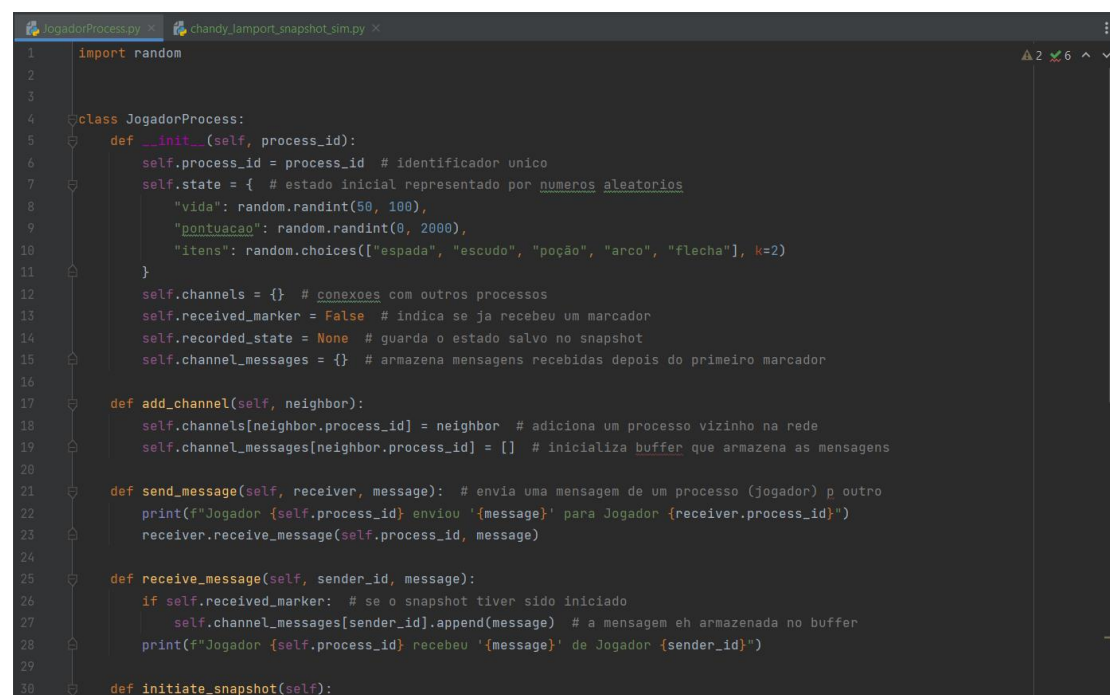
JogadorProcess.py

Essa classe define um processo, que, no contexto desse algoritmo, simula um jogador em um servidor de jogos online. A escolha da simulação ser feita baseada em um jogador, deve-se a melhor compreensão da troca de mensagens e o armazenamento do estado global, pois utilizando uma abstração, a compreensão e explicação do código tornou-se complicada e prolixa.

No início da classe, são definidos os atributos que aquele processo contém. Entre esses atributos estão o identificador do processo, o estado do processo, os canais de conexão, o estado global, e variáveis que armazenam as mensagens recebidas.

Após as definições dos atributos de um jogador, foram implementados os métodos necessários e comuns para o funcionamento do algoritmo de Chandy-Lamport, que propõe a captura do estado em um determinado momento por um dos processos ativos.

A explicação para cada linha do código consiste nos comentários da imagem abaixo.



```
1 import random
2
3
4 class JogadorProcess:
5     def __init__(self, process_id):
6         self.process_id = process_id # identificador unico
7         self.state = { # estado inicial representado por numeros aleatorios
8             "vida": random.randint(50, 100),
9             "pontuacao": random.randint(0, 2000),
10            "itens": random.choices(["espada", "escudo", "poção", "arco", "flecha"], k=2)
11        }
12        self.channels = {} # conexoes com outros processos
13        self.received_marker = False # indica se ja recebeu um marcador
14        self.recorded_state = None # guarda o estado salvo no snapshot
15        self.channel_messages = {} # armazena mensagens recebidas depois do primeiro marcador
16
17    def add_channel(self, neighbor):
18        self.channels[neighbor.process_id] = neighbor # adiciona um processo vizinho na rede
19        self.channel_messages[neighbor.process_id] = [] # inicializa buffer que armazena as mensagens
20
21    def send_message(self, receiver, message): # envia uma mensagem de um processo (jogador) p outro
22        print(f"Jogador {self.process_id} enviou '{message}' para Jogador {receiver.process_id}")
23        receiver.receive_message(self.process_id, message)
24
25    def receive_message(self, sender_id, message):
26        if self.received_marker: # se o snapshot tiver sido iniciado
27            self.channel_messages[sender_id].append(message) # a mensagem eh armazenada no buffer
28            print(f"Jogador {self.process_id} recebeu '{message}' de Jogador {sender_id}")
29
30    def initiate_snapshot(self):
```

```
JogadorProcess.py x chandy_lamport_snapshot_sim.py x
28 print(f'Jogador {self.process_id} recebeu '{message}' de Jogador {sender_id}')
29
30 def initiate_snapshot(self):
31     print(f'\n[SNAPSHOT] Jogador {self.process_id} iniciou a captura de estado global\n')
32     self.recorded_state = self.state.copy() # o processo salva o estado global
33     self.received_marker = True
34     self.send_marker() # o marcador eh enviado para os vizinhos
35
36 def send_marker(self):
37     for neighbor_id, neighbor in self.channels.items():
38         print(f'Jogador {self.process_id} enviou MARCADOR para Jogador {neighbor_id}')
39         neighbor.receive_marker(self.process_id)
40
41 def receive_marker(self, sender_id):
42     if not self.received_marker: # se o processo nunca recebeu o marcador antes
43         self.recorded_state = self.state.copy() # o processo salva o estado global
44         self.received_marker = True
45         print(f'Jogador {self.process_id} recebeu MARCADOR de {sender_id} e salvou estado: {self.recorded_state}')
46         self.send_marker() # o marcador eh enviado para os vizinhos
47     else:
48         print(f'Jogador {self.process_id} ignorou MARCADOR de {sender_id}, pois já recebeu anteriormente')
49
50 def show_snapshot(self):
51     print(
52         f'\n[ESTADO GLOBAL] Jogador {self.process_id}: Estado = {self.recorded_state}, Canais = {self.channel_messages}')
```

chandy_lamport_snapshot_sim.py

Nesse arquivo, é executada a simulação da troca de mensagens entre os 3 jogadores. Seu objetivo é simular a troca de mensagens (Estado Global) antes e após o início do snapshot, em um sistema distribuído onde três processos comunicam-se.

A explicação para cada pedaço do código consiste nos comentários da imagem abaixo.

```
JogadorProcess.py x chandy_lamport_snapshot_sim.py x
1 import time
2 from JogadorProcess import JogadorProcess
3
4
5 def chandy_lamport_snapshot_sim():
6     p1 = JogadorProcess(1)
7     p2 = JogadorProcess(2)
8     p3 = JogadorProcess(3)
9
10    # simulacao estabelecendo as conexoes entre os processos
11    p1.add_channel(p2)
12    p1.add_channel(p3)
13    p2.add_channel(p1)
14    p2.add_channel(p3)
15    p3.add_channel(p1)
16    p3.add_channel(p2)
17
18    print("\nRecebendo mensagens. . .\n")
19    print(". . . . .\n")
20    print("Enviando mensagens. . .\n")
21    p1.send_message(p2, "Ataque")
22    p2.send_message(p3, "Defesa")
23    p3.send_message(p1, "Toma Poca")
24
25    time.sleep(1)
26    p2.initiate_snapshot() # processo 2 inicia o snapshot
27
28    time.sleep(1)
29    print("\nRecebendo mensagens. . .\n")
30    print(". . . . .\n")
```

```

17
18     print("\nRecebendo mensagens. . .\n")
19     print(".....\n")
20     print("Enviando mensagens. . .\n")
21     p1.send_message(p2, "Ataque")
22     p2.send_message(p3, "Defesa")
23     p3.send_message(p1, "Toma Poção")
24
25     time.sleep(1)
26     p2.initiate_snapshot()_# processo 2 inicia o snapshot
27
28     time.sleep(1)
29     print("\nRecebendo mensagens. . .\n")
30     print(".....\n")
31     print("Enviando mensagens. . .\n")
32     p1.send_message(p3, "Toma Poção")
33     p2.send_message(p3, "Defesa")
34     p3.send_message(p2, "Vida baixa. Use a poção!")
35     p3.send_message(p1, "Ataque")
36
37     time.sleep(1)
38
39     # saída: estado global
40     p1.show_snapshot()
41     p2.show_snapshot()
42     p3.show_snapshot()
43
44     if __name__ == "__main__":
45         chandy_lamport_snapshot_sim()

```

Questão 3 – Algoritmos de Eleição - Bully

COMO EXECUTAR O SISTEMA (WINDOWS)

1. Abra o CMD do Windows e entre na pasta onde está o arquivo “bully_sim.py”.
2. Execute o sistema utilizando o comando “python bully_sim.py”

```

Prompt de Comando
C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuidos\Atividade 2\q3>python bully_sim.py
No 3 iniciou uma eleição.
No 3 -> No 4: Você tem um ID maior?
No 4 iniciou uma eleição.
No 4 -> No 5: Você tem um ID maior?
No 5 iniciou uma eleição.
No 5 tornou-se o novo líder!
No 1 reconhece no 5 como líder.
No 2 reconhece no 5 como líder.
No 3 reconhece no 5 como líder.
No 4 reconhece no 5 como líder.
No 5 falhou!
No 3 foi escolhido para iniciar a nova eleição após falha de 5.
No 3 iniciou uma eleição.
No 3 -> No 4: Você tem um ID maior?
No 4 iniciou uma eleição.
No 4 tornou-se o novo líder!
No 1 reconhece no 4 como líder.
No 2 reconhece no 4 como líder.
No 3 reconhece no 4 como líder.
No 5 reconhece no 4 como líder.
No 5 voltou a funcionar.

```



```
Prompt de Comando
No 2 reconhece no 5 como líder.
No 3 reconhece no 5 como líder.
No 4 reconhece no 5 como líder.

No 5 falhou!

No 3 foi escolhido para iniciar a nova eleição após falha de 5.
No 3 iniciou uma eleição.
No 3 -> No 4: Você tem um ID maior?
No 4 iniciou uma eleição.

No 4 tornou-se o novo líder!

No 1 reconhece no 4 como líder.
No 2 reconhece no 4 como líder.
No 3 reconhece no 4 como líder.
No 5 reconhece no 4 como líder.

No 5 voltou a funcionar.
No 5 iniciou uma eleição.
No 5 tornou-se o novo líder!

No 1 reconhece no 5 como líder.
No 2 reconhece no 5 como líder.
No 3 reconhece no 5 como líder.
No 4 reconhece no 5 como líder.

C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuídos\Atividade 2\q3>
```

} simulação de recuperação do nó 5, antigo líder. Após a recuperação, o mesmo convoca uma nova eleição

EXPLICAÇÃO DO CÓDIGO

Algoritmo de Eleição de Bully

O código para esse sistema também é composto por dois arquivos: o arquivo [Process.py](#), que define a classe Process: uma representação de um processo em um sistema distribuído; e o arquivo [bully_sim.py](#), que realiza a simulação de uma eleição entre processos, bem como as simulações de falha e recuperação do nó líder.

Process.py

Essa classe define um processo, que possui um ID único e armazena uma referência da lista de processos em um sistema distribuído. Dentro dessa classe, estão definidos também os métodos necessários para a eleição, falha e recuperação do nó.

A explicação para cada linha do código consiste nos comentários das imagens abaixo.

```

4 class Process:
5     def __init__(self, process_id, processes):
6         self.process_id = process_id # identificador unico
7         self.processes = processes # lista de processos
8         self.is_leader = False # indica se o processo é o líder
9         self.alive = True # indica se o processo está ativo
10
11     def start_election(self, nodes):
12         if not self.alive:
13             return # se o processo está inativo não pode iniciar uma eleição
14
15         if self.is_leader:
16             return # se já é líder não precisa iniciar outra eleição
17
18         print(f'No {self.process_id} iniciou uma eleição.')
19
20         # lista de nós com ID maior e ativos, para garantir que apenas os maiores estejam competindo
21         higher_nodes = [node for node in nodes if node.process_id > self.process_id and node.alive]
22
23         if not higher_nodes: # se não tem processos com ID maior e ativo
24             self.become_leader() # o nó se proclama líder
25         else:
26             for node in higher_nodes:
27                 print(f'No {self.process_id} -> No {node.process_id}: Você tem um ID maior?') # desafio aos nós
28                 response = node.respond_to_election() # resposta se está ativo
29                 if response: # se o nó estiver ativo
30                     node.start_election(nodes) # o maior e ativo controla a eleição
31                 return # evita que vários processos tentem se tornar líder ao mesmo tempo
32
33         self.become_leader() # se nenhum nó responder o nó que iniciou a eleição vira líder

```

```

35     def respond_to_election(self):
36         return self.alive # retorna true se o processo estiver ativo
37
38     def become_leader(self):
39         self.is_leader = True # define o processo como líder
40         print(f'\nNo {self.process_id} tornou-se o novo líder!\n')
41         for p in self.processes:
42             if p.process_id != self.process_id:
43                 p.receive_leader_message(self.process_id) # notifica todos os outros nós sobre sua eleição
44
45     def receive_leader_message(self, leader_id):
46         print(f'No {self.process_id} reconhece o {leader_id} como líder.')
47         self.is_leader = False # o processo deixa de ser um líder
48
49     def fail(self):
50         self.alive = False # simula uma falha no processo, desativa o nó
51         self.is_leader = False
52         print(f'\nNo {self.process_id} falhou!\n')
53         self.initiate_new_election() # inicia uma nova eleição
54
55     def initiate_new_election(self):
56         active_nodes = [p for p in self.processes if p.alive]
57         if active_nodes:
58             starter = random.choice(active_nodes)
59             print(f'No {starter.process_id} foi escolhido para iniciar a nova eleição após falha de {self.process_id}.')
60             starter.start_election(self.processes)
61
62     def recover(self):
63         self.alive = True # simula uma recuperação do processo reativando o nó
64         print(f'\nNo {self.process_id} voltou a funcionar.\n')
65         self.start_election(self.processes) # inicia uma nova eleição

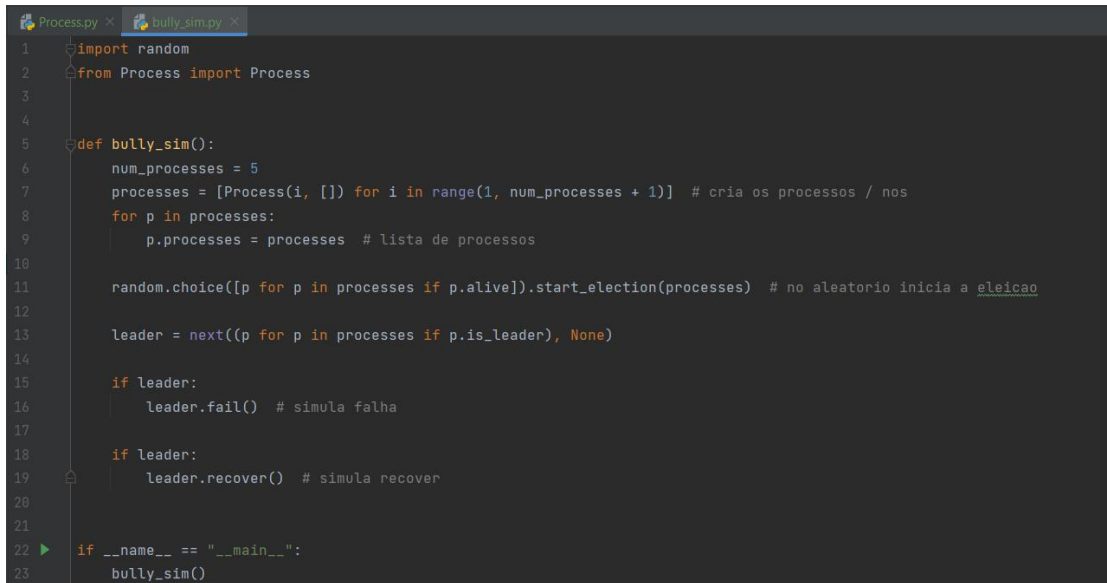
```

bully_sim.py

Nesse arquivo acontece a simulação do algoritmo de eleição de bully. Primeiro, 5 processos são criados, para eficiência na visualização dos resultados. Após a criação dos processos, um nó aleatório convoca uma eleição e o nó com maior ID é eleito.

Para simular a falha e a recuperação do nó líder, uma variável “leader” foi criada para armazenar a informação de quem ganhou a eleição. Então, são chamadas as funções “fail” e “recover”, a fim de completar a simulação dos erros no nó líder, onde ambas funções convocam novas eleições.

A explicação para cada pedaço do código consiste nos comentários da imagem abaixo.

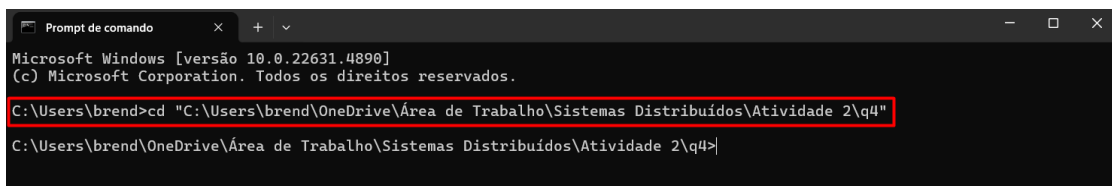


```
1 import random
2 from Process import Process
3
4
5 def bully_sim():
6     num_processes = 5
7     processes = [Process(i, []) for i in range(1, num_processes + 1)] # cria os processos / nos
8     for p in processes:
9         p.processes = processes # lista de processos
10
11     random.choice([p for p in processes if p.alive]).start_election(processes) # no aleatorio inicia a eleicao
12
13     leader = next((p for p in processes if p.is_leader), None)
14
15     if leader:
16         leader.fail() # simula falha
17
18     if leader:
19         leader.recover() # simula recover
20
21
22 if __name__ == "__main__":
23     bully_sim()
```

Questão 4 – Detecção de Falhas em Sistemas Distribuídos

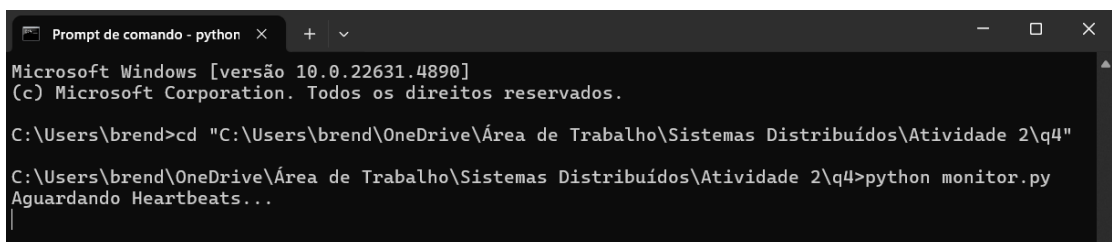
COMO EXECUTAR O SISTEMA (WINDOWS)

1. Abra o CMD do Windows e entre na pasta onde está o arquivo “monitor.py”.



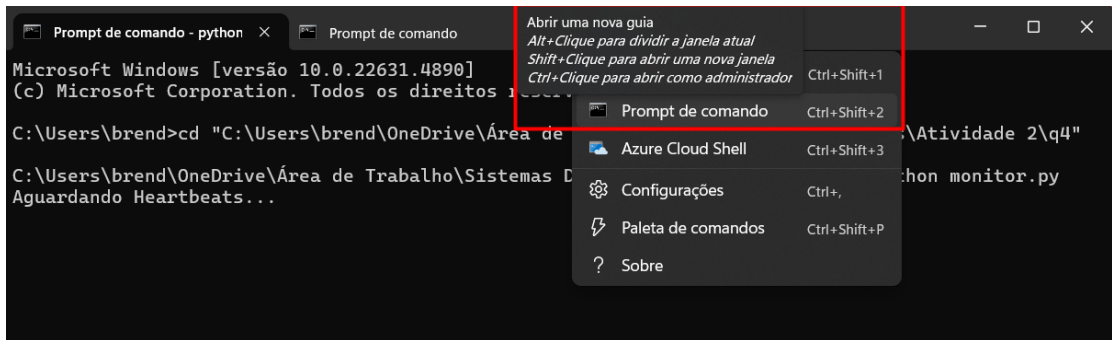
```
Prompt de comando
Microsoft Windows [versão 10.0.22631.4890]
(c) Microsoft Corporation. Todos os direitos reservados.
C:\Users\brend>cd "C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuídos\Atividade 2\q4"
C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuídos\Atividade 2\q4>
```

2. Execute o sistema de monitoramento utilizando o comando “python monitor.py”

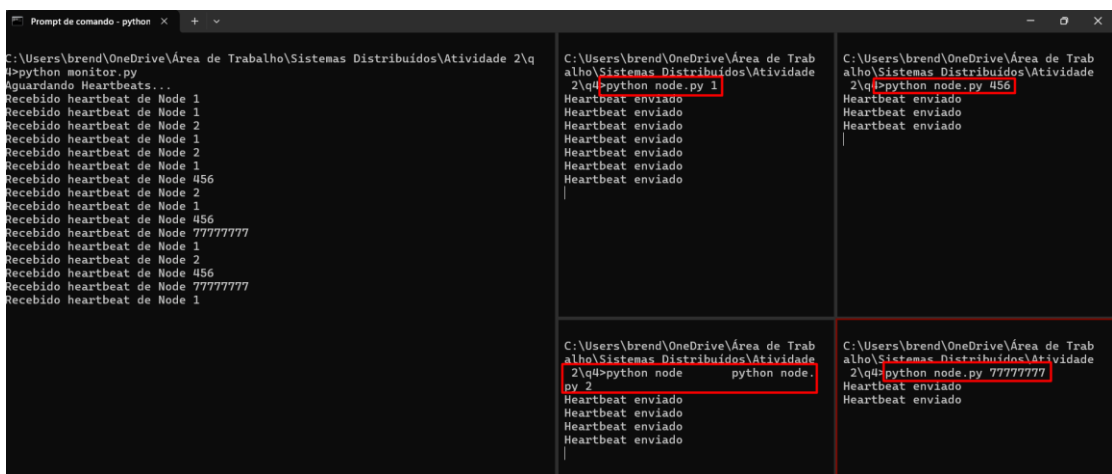


```
Prompt de comando - python
Microsoft Windows [versão 10.0.22631.4890]
(c) Microsoft Corporation. Todos os direitos reservados.
C:\Users\brend>cd "C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuídos\Atividade 2\q4"
C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuídos\Atividade 2\q4>python monitor.py
Aguardando Heartbeats...
```

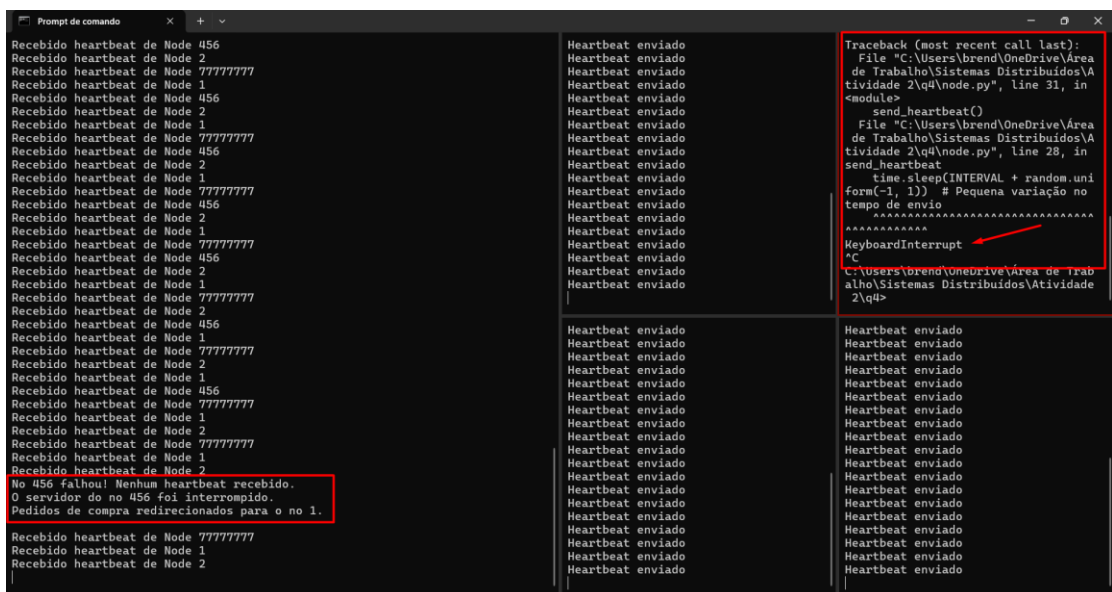
3. Abra novas abas no terminal para cada novo nó que será monitorado



- Para ativar um nó execute no terminal: “`python node.py <node_id>`”, onde `node_id` é um número inteiro que será o ID daquele nó.



- Para interromper um nó aperte os botões “Ctrl” + “C” do teclado. O monitor exibirá uma mensagem de falha no nó interrompido.



Caso todos os nós sejam interrompidos, o monitor enviará uma mensagem de falha

Prompt de comando		
<pre>Recebido heartbeat de Node 1 Recebido heartbeat de Node 13 Recebido heartbeat de Node 77777777 Recebido heartbeat de Node 2 Recebido heartbeat de Node 1 Recebido heartbeat de Node 13 Recebido heartbeat de Node 77777777 Recebido heartbeat de Node 2 Recebido heartbeat de Node 13 Recebido heartbeat de Node 77777777 Recebido heartbeat de Node 13 Recebido heartbeat de Node 2 Recebido heartbeat de Node 1 Recebido heartbeat de Node 77777777 Recebido heartbeat de Node 2 Recebido heartbeat de Node 1 Recebido heartbeat de Node 77777777 No 13 falhou! Nenhum heartbeat recebido. O servidor do no 13 foi interrompido. Pedidos de compra redirecionados para o no 1.</pre>	<pre>Traceback (most recent call last): File "C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuídos\Atividade 2\q4\node.py", line 31, in <module> send_heartbeat() File "C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuídos\Atividade 2\q4\node.py", line 28, in send_heartbeat time.sleep(INTERVAL + random.uniform(-1, 1)) # Pequena variação no tempo de envio AAAAAAAAAAAAAA KeyboardInterrupt ^C C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuídos\Atividade 2\q4></pre>	<pre>Traceback (most recent call last): File "C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuídos\Atividade 2\q4\node.py", line 31, in <module> send_heartbeat() File "C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuídos\Atividade 2\q4\node.py", line 28, in send_heartbeat time.sleep(INTERVAL + random.uniform(-1, 1)) # Pequena variação no tempo de envio AAAAAAAAAAAAAA KeyboardInterrupt ^C C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuídos\Atividade 2\q4></pre>
<pre>Recebido heartbeat de Node 2 Recebido heartbeat de Node 77777777 Recebido heartbeat de Node 1 Recebido heartbeat de Node 2 Recebido heartbeat de Node 77777777 Recebido heartbeat de Node 1 Recebido heartbeat de Node 2 Recebido heartbeat de Node 77777777 No 1 falhou! Nenhum heartbeat recebido. O servidor do no 1 foi interrompido. Pedidos de compra redirecionados para o no 2.</pre>	<pre>Traceback (most recent call last): File "C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuídos\Atividade 2\q4\node.py", line 31, in <module> send_heartbeat() File "C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuídos\Atividade 2\q4\node.py", line 28, in send_heartbeat time.sleep(INTERVAL + random.uniform(-1, 1)) # Pequena variação no tempo de envio AAAAAAAAAAAAAA KeyboardInterrupt ^C C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuídos\Atividade 2\q4></pre>	<pre>Traceback (most recent call last): File "C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuídos\Atividade 2\q4\node.py", line 31, in <module> send_heartbeat() File "C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuídos\Atividade 2\q4\node.py", line 28, in send_heartbeat time.sleep(INTERVAL + random.uniform(-1, 1)) # Pequena variação no tempo de envio AAAAAAAAAAAAAA KeyboardInterrupt ^C C:\Users\brend\OneDrive\Área de Trabalho\Sistemas Distribuídos\Atividade 2\q4></pre>
<pre>No 77777777 falhou! Nenhum heartbeat recebido. O servidor do no 77777777 foi interrompido. Crítico: nenhum no disponível para redirecionamento.</pre>		

Detecção de Falhas utilizando Heartbeat

monitor.py

PÁGINA 12

```

1 import time
2 import socket
3 import threading
4
5 HOST = "127.0.0.1" # o monitor estará escutando os heartbeats localmente
6 PORT = 5000 # o monitor receberá os heartbeats através dessa porta
7 TIMEOUT = 10 # tempo limite para considerar um nó como falho
8
9 nodes_status = {} # armazena o último tempo que cada nó enviou um heartbeat
10 lock = threading.Lock() # evita que múltiplas threads modifiquem o dicionário
11
12
13 def receive_heartbeat():
14     with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
15         s.bind((HOST, PORT)) # cria um socket UDP para escutar as mensagens
16         print("Aguardando Heartbeats...")
17
18         while True:
19             data, addr = s.recvfrom(1024) # recebe a mensagem
20             message = data.decode()
21             node_id = int(message.split()[-1]) # extrai o node_id da string enviada
22
23             with lock:
24                 nodes_status[node_id] = time.time() # armazena o tempo do último heartbeat para rastrear a atividade
25                 print(f"Recebido heartbeat de Node {node_id}")
26
27
28 def check_failures():
29     while True:
30         time.sleep(1) # verifica a cada segundo

```

```

28 def check_failures():
29     while True:
30         time.sleep(1) # verifica a cada segundo
31         current_time = time.time()
32         with lock:
33             for node_id in list(nodes_status.keys()): # verifica cada nó na lista
34                 if current_time - nodes_status[node_id] > TIMEOUT: # se não enviou nenhuma mensagem dentro do timeout
35                     print(f"No {node_id} falhou! Nenhum heartbeat recebido.")
36                     handle_failure(node_id) # inicia a reacção a falha
37                     del nodes_status[node_id] # remove o nó da lista
38
39     # obs.: essa reacção é apenas uma simulação para os prints
40
41 def handle_failure(node_id):
42     print(f"O servidor do nó {node_id} foi interrompido.")
43     available_nodes = [n for n in nodes_status.keys() if n != node_id] # verifica os nós ativos na lista
44     if available_nodes:
45         new_target = min(available_nodes) # seleciona o nó de menor ID
46         print(f"Pedidos de compra redirecionados para o nó {new_target}.\n") # redireciona os pedidos de compra
47     else:
48         print("Crítico: nenhum nó disponível para redirecionamento.\n") # se não tiver nós ativos, envia alerta crítico
49
50
51 threading.Thread(target=receive_heartbeat, daemon=True).start() # cria uma thread para receber mensagens
52 threading.Thread(target=check_failures, daemon=True).start() # cria uma thread para verificar falhas
53
54 while True:
55     time.sleep(1)

```

node.py

Já o arquivo node.py busca simular um nó em um sistema distribuído, esse nó envia os heartbeats para o monitor a cada 5 segundos e pode ser instanciado diversas vezes, bem como interrompido a qualquer momento. A explicação para cada linha do código consiste nos comentários da imagem abaixo.

```
monitor.py x node.py x
1 import time
2 import socket
3 import sys
4 import random
5
6 HOST = "127.0.0.1" # envia os heartbeats localmente
7 PORT = 5000 # envia os heartbeats atraves dessa porta
8 INTERVAL = 5 # envia um heartbeat a cada 5 segundos
9
10 if len(sys.argv) != 2: # mensagem de erro caso o usuario esqueça de atribuir um ID
11     print("Para ligar um servidor/no utilize: python node.py <node_id>")
12     sys.exit(1)
13
14 node_id = int(sys.argv[1]) # o id do no eh definido pelo terminal pelo usuario
15
16
17 def send_heartbeat():
18     with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s: # cria um socket UDP para envio de mensagens
19         while True:
20             try:
21                 message = f"Heartbeat from Node {node_id}".encode() # envia o heartbeat
22                 s.sendto(message, (HOST, PORT)) # local que recebe o heartbeat
23                 print(f"Heartbeat enviado")
24             except Exception as e:
25                 print(f"Falha ao enviar Heartbeat: {e}")
26
27             time.sleep(INTERVAL + random.uniform(-1, 1)) # variacao no tempo de envio para evitar que todos os nós
28                 # enviem mensagens ao mesmo tempo
29
30     send_heartbeat()
```