Mutual Exclusion

Samara Ribeiro Silva
Instituto Tecnológico de Aeronáutica, Laboratório de Processamento Distribuído (CES-27). Professor Celso Hirata e Professora Juliana de Melo Bezerra, São José dos Campos, São Paulo, 26 de setembro de 2021.

1. Introdução

Este trabalho tem como objetivo trabalhar com algoritmos de exclusão mútua para sistemas distribuídos. Os algoritmos de exclusão mútua tem como objetivo evitar *race condition* e existem várias classes de algoritmos: *Token-based*, *Timestamp-based*, *Quorum-based*.

O escolhido para implementação neste laboratório foi o algoritmo de Ricart-Agrawala que é da classe *Timestamp-based* e utiliza o relógio lógico de Lamport para determinar a preferência de acesso a sessão crítica (critical session CS).

Na inicialização do algoritmo todos os processos estão no estado RELEASED, ou seja, a CS está liberada. Se algum processo deseja acessar a CS este processo entra no estado WANTED e deve enviar mensagem para todos os outros processos solicitando acesso e informando seu identificador (ID), o seu relógio lógico do momento da solicitação (T) e o tipo da mensagem (REQUEST). A entrada na CS é liberada somente após o recebimento das respostas (REPLY) de todos outros processos. Então o processo solicitante entra no estado HELD e acessa a CS.

Ao receber uma mensagem de REQUEST o processo verifica seu estado. Se ele está no estado RELEASED ele encaminha uma mensagem de resposta REPLY liberando o acesso do processo solicitante. Caso o estado seja HELD o processo responderá apenas após liberar a CS, então o processo solicitante é inserido numa fila. E por fim, caso o estado seja WANTED o processo responde a requisição de imediato se o tempo T recebido for menor que o seu T ou se os Ts forem iguais e o ID do processo solicitante for menor que o seu ID, caso contrário o processo solicitante é inserido em uma fila e fica aguardando a liberação da CS para receber o REPLY.

Ao liberar a CS o processo passa para o estado de RELEASED e envia mensagem resposta REPLY para todos os processos presentes em sua fila de espera.

2. Implementação

A linguagem de programação utilizada para a implementação foi o *golang*. O trabalho está dividido em dois arquivos *Process.go* e *SharedResource.go*. Os códigos da atividade dirigida foram utilizados como base para a implementação do *Process.go*.

Na função *main* do *Process.go* as variáveis globais são iniciadas e as conexões dos servidores são realizadas através da função *initConnections().* A função *readInput(ch)* "escuta" o teclado e a função *doServerJob* "escuta" a conexão UDP.

Ao receber input do teclado a função *doClientJob(input)* é chamada. Essa função verifica o estado do processo e descarta o input quando o estado é HELD ou WANTED, se o estado for RELEASED e o *input* for o seu ID será executada um INTERNAL EVENT atualizando o clock e imprimindo mensagem no seu terminal, se for diferente a função WantCS() é chamada.

A função *WantCS()* incrementa o *logicalClock* em uma unidade, muda o estado do processo para WANTED, imprime mensagem no seu terminal e envia mensagem utilizando uma *struct* (com os campos ID, Time, Tipo e Msg) para todos outros processos através da função *sendMsg(item, processo destino).* A função *sendMsg* transforma o item da mensagem a ser enviada para formato *json* e envia para o processo de destino.

Como a função *doServerJob* "escuta" a conexão UDP, será por essa função que as mensagens dos outros processos chegará. Essa função lê e converte a mensagem para a estrutura inicial (struct) e chama a função *recMsg(mensagem recebida)*.

A função *recMsg(mensagem recebida)* atualiza o logicalClock para $logicalClock = máx(logicalClock, logicalClock\,recebido) + 1$ e verifica o tipo da mensagem. Se o tipo for REPLY a variável que conta o número de replies recebidos é incrementada em uma unidade e é impressa no terminal uma mensagem informando o RECEIVED REPLY. Se o número de replies recebido for igual a $N - 1$, onde N é a quantidade de processos existentes, o estado do processo é autorizado a entrar na CS. O estado é atualizado para HELD, uma mensagem é enviada para o processo *SharedResouse* e após um tempo (5 segundos) a função *exitCS()* é chamada. A função *exitCS()* altera o estado para RELEASED e envia mensagem para todos na fila de espera do processo. Se a mensagem recebida por *recMsg()* for do tipo REQUEST uma mensagem de REPLY é enviada se o estado for RELEASED, se o estado for HELD o processo solicitante é inserido na fila de espera e se o estado for WANTED são realizadas as comparações pertinentes em relação ao tempo de requisição.

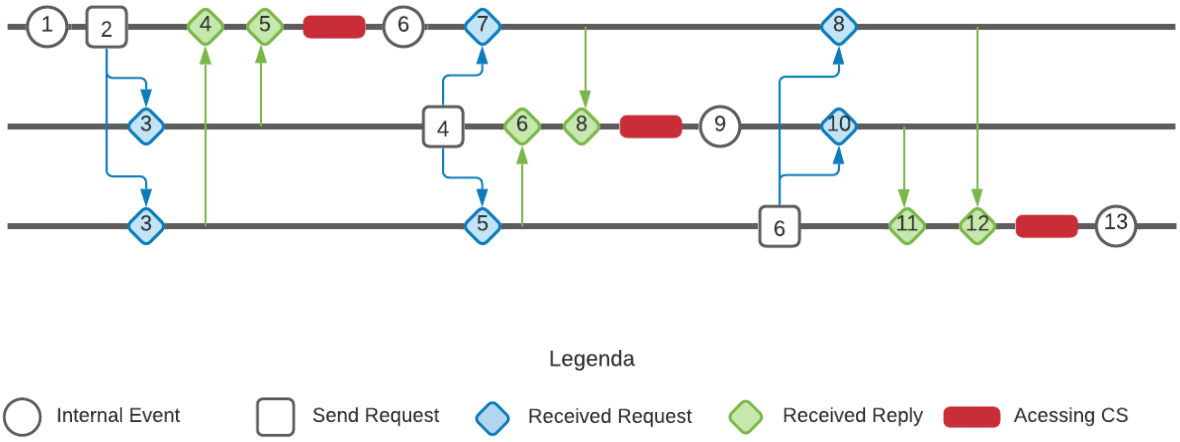
A função *main* do *SharedResourse.go* aguarda uma mensagem da conexão UDP e imprime no terminal o ID, o *logicalClock* e a *Msg*.

3. Testes

Foram realizados três testes para mostrar o correto funcionamento do código.

O primeiro teste pode ser observado nas figuras 1 a 5 e tem como objetivo mostrar casos triviais de INTERNAL EVENT, envio e recebimento de REQUESTS e REPLY e acesso da CS.

Figura 1: Esquema da saída do teste 1.



Na figura 1, é possível observar a dinâmica de atualização do *logicalClock* e o acesso a CS apenas após o recebimento das mensagens de REPLY.

Foram realizados as seguintes ações neste teste:

- O processo 1 realiza um INTERNAL EVENT.
- O processo 1 solicita acesso a CS enviando REQUEST para os demais processos.
- Os processos 2 e 3 atualizam seus *logicalClocks* ao receberem o REQUEST e enviam mensagem de REPLY, visto que estão no estado RELEASED.
- O processo 1 atualiza seu *logicalClock* ao receber as mensagens de REPLY dos processos 2 e 3. Vale ressaltar que a ordem de recebimento não é garantida.
- O processo 1 acessa a CS.
- O processo 1 libera a CS.
- O processo 1 realiza um INTERNAL EVENT.
- O processo 2 solicita acesso a CS enviando REQUEST para os demais processos.
- Os processos 1 e 3 atualizam seus *logicalClocks* ao receberem o REQUEST e enviam mensagem de REPLY, visto que estão no estado RELEASED.
- O processo 2 atualiza seu *logicalClock* ao receber as mensagens de REPLY dos processos 1 e 3.
- O processo 2 acessa a CS.
- O processo 2 libera a CS.
- O processo 2 realiza um INTERNAL EVENT.

- O processo 3 solicita acesso a CS enviando REQUEST para os demais processos.
- Os processos 1 e 2 atualizam seus *logicalClocks* ao receberem o REQUEST e enviam mensagem de REPLY, visto que estão no estado RELEASED.
- O processo 3 atualiza seu *logicalClock* ao receber as mensagens de REPLY dos processos 1 e 2.
- O processo 3 acessa a CS.
- O processo 3 libera a CS.
- O processo 3 realiza um INTERNAL EVENT.

Figura 2: *Print* do terminal do processo 1 no teste 1.

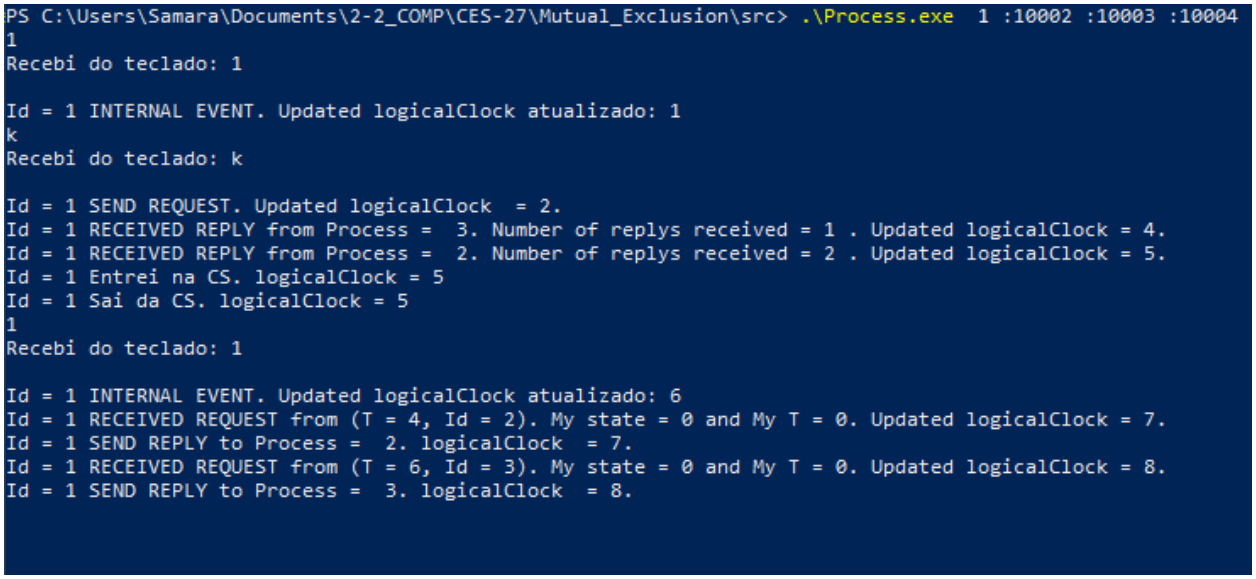


Figura 3: *Print* do terminal do processo 2 no teste 1.

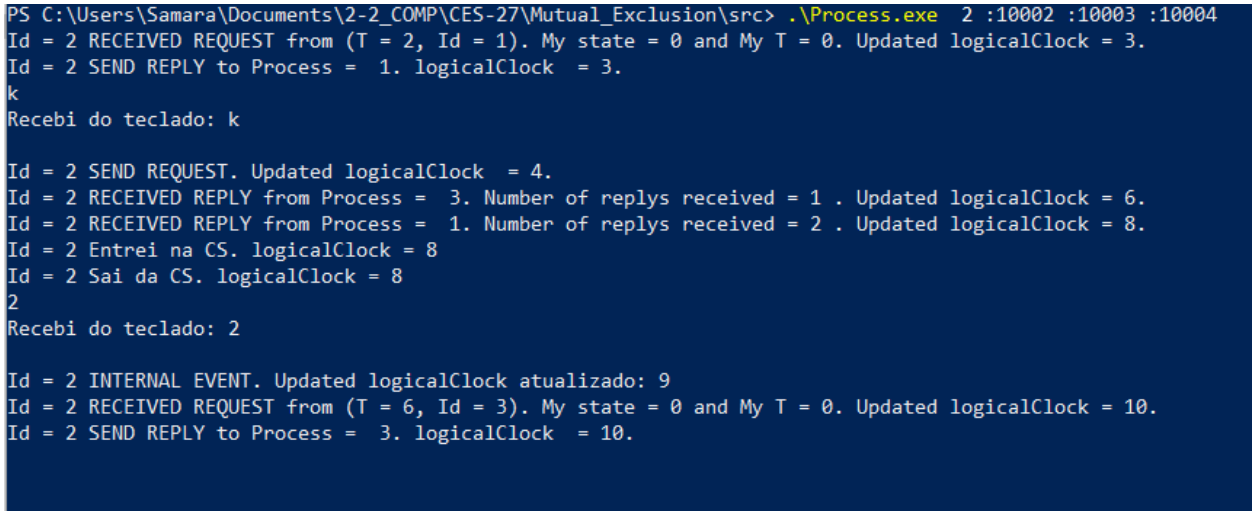


Figura 4: *Print* do terminal do processo 3 no teste 1.

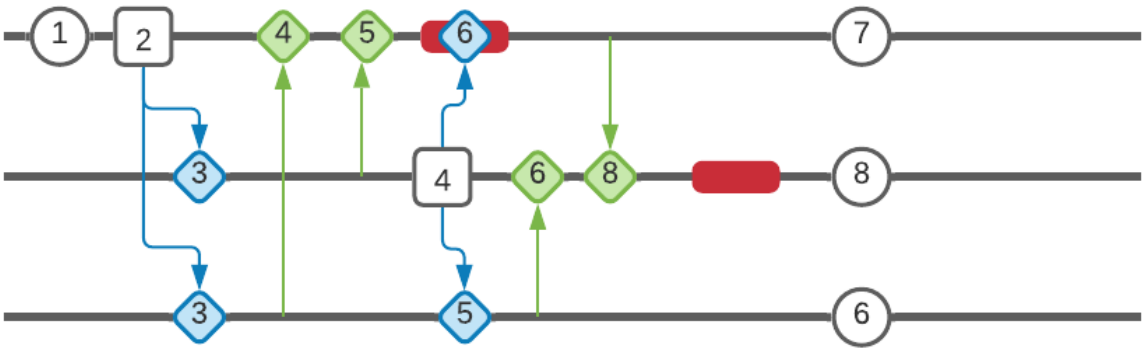
```
PS C:\Users\Samara\Documents\2-2_COMP\CES-27\Mutual_Exclusion\src> .\Process.exe  3 :10002 :10003 :10004
Id = 3 RECEIVED REQUEST from (T = 2, Id = 1). My state = 0 and My T = 0. Updated logicalClock = 3.
Id = 3 SEND REPLY to Process =  1. logicalClock  = 3.
Id = 3 RECEIVED REQUEST from (T = 4, Id = 2). My state = 0 and My T = 0. Updated logicalClock = 5.
Id = 3 SEND REPLY to Process =  2. logicalClock  = 5.
k
Recebi do teclado: k

Id = 3 SEND REQUEST. Updated logicalClock  = 6.
Id = 3 RECEIVED REPLY from Process =  2. Number of replys received = 1 . Updated logicalClock = 11.
Id = 3 RECEIVED REPLY from Process =  1. Number of replys received = 2 . Updated logicalClock = 12.
Id = 3 Entrei na CS. logicalClock = 12
Id = 3 Sai da CS. logicalClock = 12
3
Recebi do teclado: 3

Id = 3 INTERNAL EVENT. Updated logicalClock atualizado: 13
```

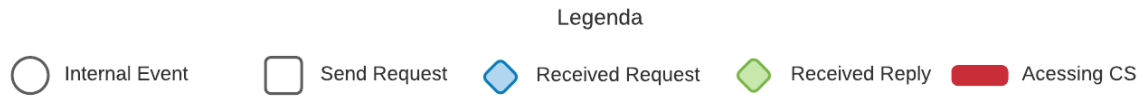
Figura 5: *Print* do terminal do processo *SharedResouce* no teste 1.

```
PS C:\Users\Samara\Documents\2-2_COMP\CES-27\Mutual_Exclusion\src> .\SharedResource.exe
Processo id = 1 entrou na CS. logicalClock = 5. Mensagem enviada: Olá =)
Processo id = 2 entrou na CS. logicalClock = 8. Mensagem enviada: Olá =)
Processo id = 3 entrou na CS. logicalClock = 12. Mensagem enviada: Olá =)
```

No segundo teste, representado nas figuras 6 a 10, pode-se observar que quando um processo está acessando a CS (estado HELD) ele só responde a mensagens de REQUEST após liberar a CS e o processo solicitante só será autorizado a acessar a CS após o recebimento de todas as mensagens de REPLY.

Figura 6: Esquema da saída do teste 2.





Os seguintes passos Foram realizados neste teste:

- O processo 1 realiza um INTERNAL EVENT.
- O processo 1 solicita acesso a CS enviando REQUEST para os demais processos;
- Os processos 2 e 3 atualizam seus *logicalClocks* ao receberem o REQUEST e enviam mensagem de REPLY, visto que estão no estado RELEASED.
- O processo 1 atualiza seu *logicalClock* ao receber as mensagens de REPLY dos processos 2 e 3.
- O processo 1 acessa a CS.
- O processo 2 solicita acesso a CS enviando REQUEST para os demais processos antes da liberação da CS pelo processo 1.
- Os processos 1 e 3 atualizam seus *logicalClocks* ao receberem o REQUEST.
- O processo 3 envia REPLY para o processo 2.
- O processo 1 libera a CS.
- O processo 1 envia REPLY para o processo 2.
- O processo 2 acessa a CS
- O processo 2 libera a CS.
- O processo 1 realiza um INTERNAL EVENT.
- O processo 2 realiza um INTERNAL EVENT.
- O processo 3 realiza um INTERNAL EVENT.

Figura 7: *Print* do terminal do processo 1 no teste 2.

```
PS C:\Users\Samara\Documents\2-2_COMP\CES-27\Mutual_Exclusion\src> .\Process.exe  1 :10002 :10003 :10004
1
Recebi do teclado: 1

Id = 1 INTERNAL EVENT. Updated logicalClock atualizado: 1
k
Recebi do teclado: k

Id = 1 SEND REQUEST. Updated logicalClock  = 2.
Id = 1 RECEIVED REPLY from Process =  3. Number of replys received = 1 . Updated logicalClock = 4.
Id = 1 RECEIVED REPLY from Process =  2. Number of replys received = 2 . Updated logicalClock = 5.
Id = 1 Entrei na CS. logicalClock = 5
Id = 1 RECEIVED REQUEST from (T = 4, Id = 2). My state = 2 and My T = 2. Updated logicalClock = 6.
k
Recebi do teclado: k

x ignorado.
Id = 1 Sai da CS. logicalClock = 6
Id = 1 SEND REPLY to Process =  2. logicalClock  = 6.
1
Recebi do teclado: 1

Id = 1 INTERNAL EVENT. Updated logicalClock atualizado: 7
```

Observe que na figura 7 ao receber uma entrada do teclado enquanto o estado do processo era HELD a entrada foi ignorada.

Figura 8: *Print* do terminal do processo 2 no teste 2.

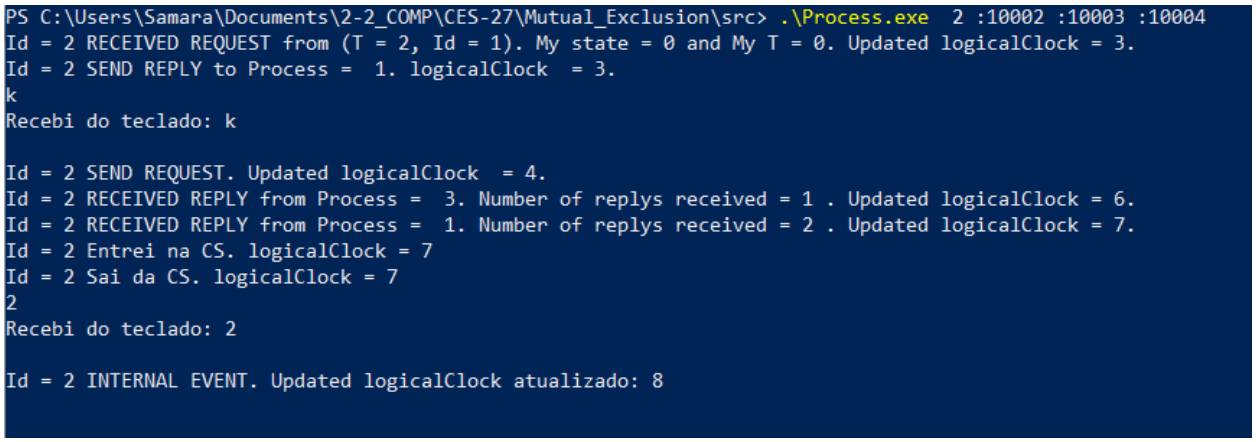


Figura 9: *Print* do terminal do processo 3 no teste 2.

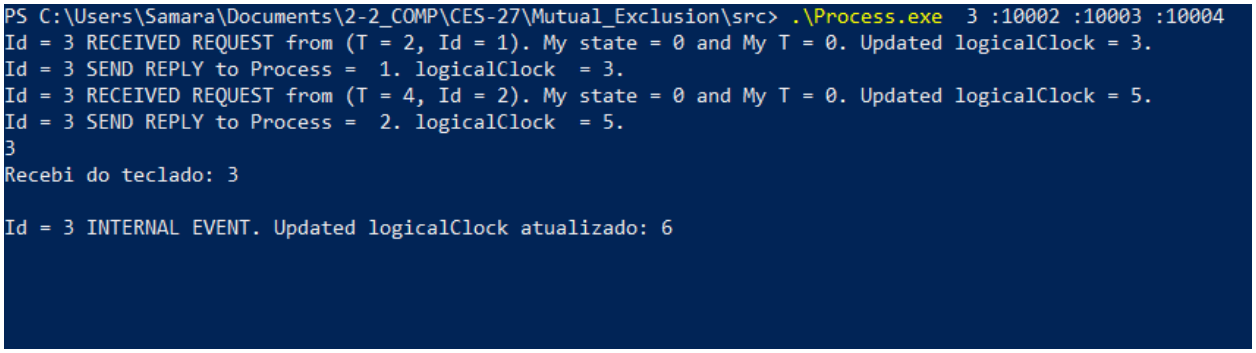
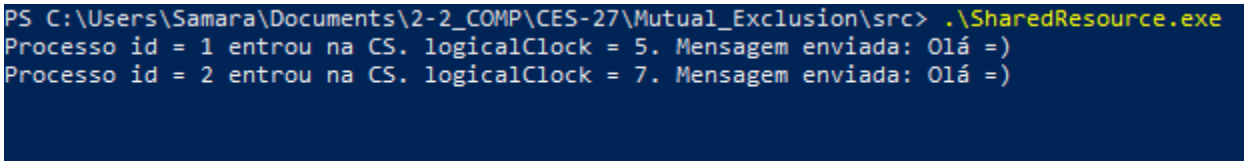
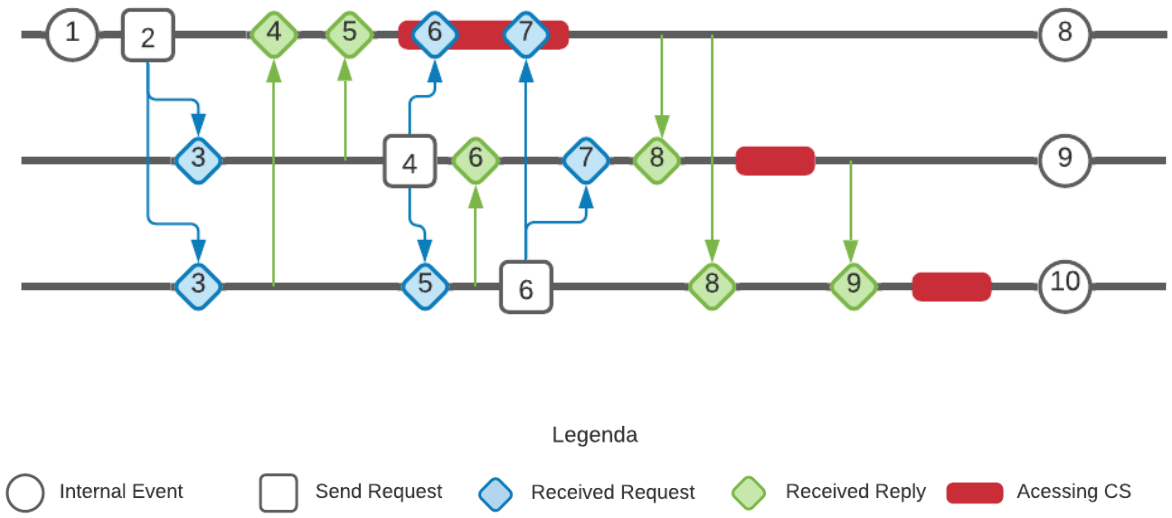


Figura 10: *Print* do terminal do processo *SharedResouce* no teste 2.



Os resultados obtidos no terceiro teste podem ser observados nas figuras 11 a 15. Este teste tem como objetivo observar o correto funcionamento da comparação dos tempos de requisição e da fila de espera.

Figura 11: Esquema da saída do teste 3.



Os seguintes passos Foram realizados neste teste:

- O processo 1 realiza um INTERNAL EVENT.
- O processo 1 solicita acesso a CS enviando REQUEST para os demais processos;
- Os processos 2 e 3 atualizam seus *logicalClocks* ao receberem o REQUEST e enviam mensagem de REPLY, visto que estão no estado RELEASED.
- O processo 1 atualiza seu *logicalClock* ao receber as mensagens de REPLY dos processos 2 e 3.
- O processo 1 acessa a CS.
- O processo 2 solicita acesso a CS enviando REQUEST para os demais processos antes da liberação da CS pelo processo 1.
- Os processos 1 e 3 atualizam seus *logicalClocks* ao receberem o REQUEST.
- O processo 3 envia REPLY para o processo 2.
- O processo 3 solicita acesso a CS enviando REQUEST para os demais processos antes da liberação da CS pelo processo 1 e com o processo 2 já em estado WANTED.
- Os processos 1 e 2 atualizam seus *logicalClocks* ao receberem o REQUEST.
- O processo 1 libera a CS.
- O processo 1 envia REPLY para o processo 2 e 3, uma vez que eles estavam na fila de espera.
- O processo 2 acessa a CS.
- O processo 2 libera a CS.
- O processo 2 envia REPLY para o processo 3.
- O processo 3 acessa a CS.
- O processo 3 libera a CS.
- O processo 1 realiza um INTERNAL EVENT.
- O processo 2 realiza um INTERNAL EVENT.
- O processo 3 realiza um INTERNAL EVENT.

Figura 12: *Print* do terminal do processo 1 no teste 3.

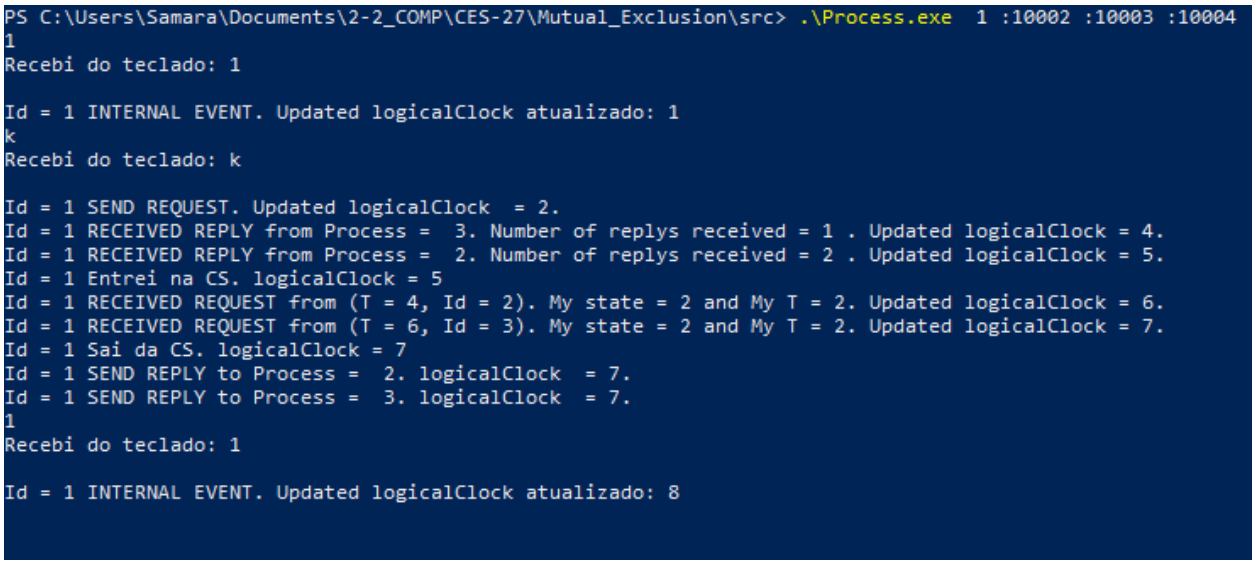


Figura 13: *Print* do terminal do processo 2 no teste 3.

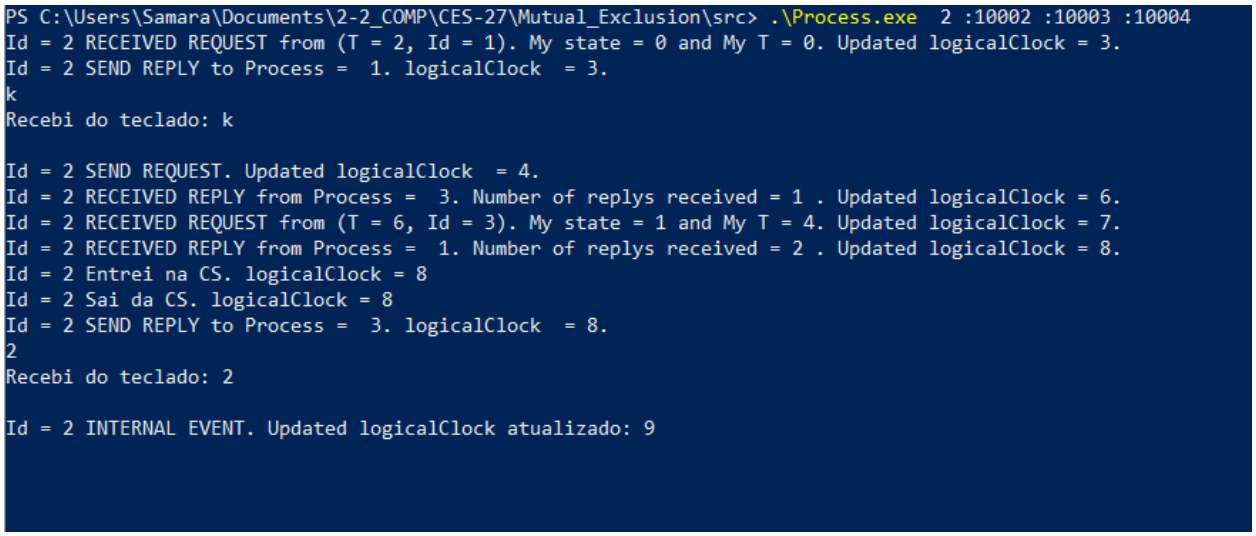


Figura 14: *Print* do terminal do processo 2 no teste 3.

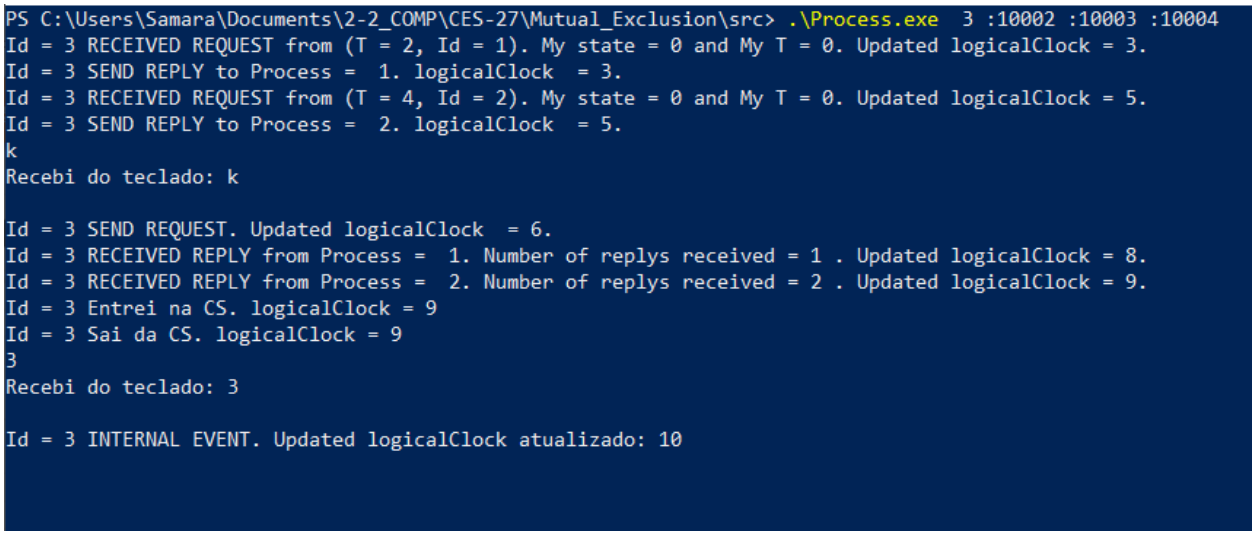
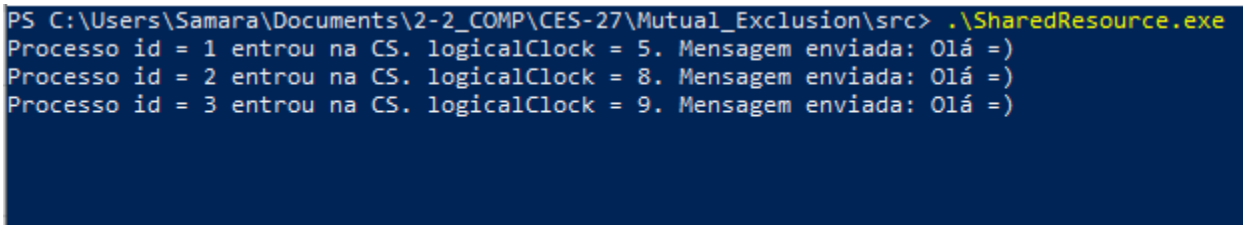


Figura 15: *Print* do terminal do processo *SharedResouce* no teste 3.



4. Conclusão

Este trabalho serviu para exercitar e fixar os conhecimentos da teoria sobre o algoritmos de exclusão mútua para sistemas distribuídos, em especial o algoritmo de Ricart-Agrawala.

Os resultados obtidos descritos na seção de testes foram condizentes com o esperado teoricamente e, portanto, corroboram para a classificação da implementação como adequada do algoritmo proposto.