

TP3 - Sistemas Digitais 23.1 - Sem arquivos de Log

PedroGomes 119042303,
FilipePrates 116013311

O Objetivo desse trabalho é projetar e implementar o algoritmo centralizado de exclusão mútua distribuída.

Código Fonte: <https://github.com/Pedro-gomes8/DistributedSystems/tree/main/TP3>

O Servidor

A implementação do servidor foi feita na linguagem de programação Go, com ajuda da biblioteca de RPC (remote procedure call) chamada gRPC. Debaixo dos panos essa biblioteca nos ajuda à criar múltiplas threads que escutam os clientes em diferentes sockets, e reage às suas mensagens.

Tal biblioteca permite a criação de um “contrato de comunicação”: o PROTOBUFF, escrito em linguagem neutra, gera automaticamente o código para tratar as mensagens entre diferentes linguagens através de stubs e do marshalling.

O servidor possui um semáforo que controla o acesso dos clientes ao arquivo de escrita. Uma mensagem de ‘GRANT’ significa que o semáforo foi corretamente adquirido e ele só é solto por uma mensagem de ‘RELEASE’ vinda daquele processo que possui o ‘GRANT’.

Outros processos que tentarem escrever no arquivo estarão aguardando até que o acquire desse semáforo retorne, indicando que é a vez dele de escrever. A implementação desse semáforo possui um outro semáforo para controle da fila de requisições.

Não implementamos temporizadores ou maneiras robustas de tratar falhas, se o processo que possui o ‘GRANT’ falhar e não retornar o ‘RELEASE’, o servidor se vê impossibilitado de continuar à tratar os clientes (o que acontece na prática ao parar o script dos Clientes no meio, ao reiniciar, o servidor está esperando ainda um Release de um processo que não existe mais, é necessário parar e recomençar também o processo do servidor).

Os Clientes

Para ver a biblioteca de RPC do Go em ação, foi decidido escrever o código dos clientes em python, e se utilizar dos stubs e do marshalling para traduzir de uma linguagem para outra na prática.

O código recebe como argumento os valores de **n** (número de processos), **r** (número de vezes que cada processo precisa acessar a zona crítica), e **k** (número de segundos cada processo espera antes de começar um novo pedido). Para **n** vezes, começa um processo que roda a função *clientRequests* que efetivamente envia as requisições de acesso e age quando recebe as notificações de GRANT.

```
def main():
    if len(sys.argv) != 4:
        print(
            "Usage: python3 main.py <noOfProcesses> <noOfRequestsPerProcess> <secondsToSleep>"
        )
        sys.exit(1)
    n_processes, r_requests, k_seconds = [int(x) for x in sys.argv[1:]]
    processes = []
    for i in range(n_processes):
        p = multiprocessing.Process(
            target=clientRequests, args=(r_requests, k_seconds, i, write)
        )
        processes.append(p)
    for p in processes:
        p.start()
    for p in processes:
        p.join()
    print("done")
//---

def clientRequests(requests, seconds, id, writefunc):
    with grpc.insecure_channel("localhost:50051") as channel:
        stub = tp3_pb2_grpc.Tp3RPCStub(channel)
        for _ in range(requests):
            message = tp3_pb2.ClientMessage(ProcessId=id)
            granted = stub.Request(message)
            if granted.granted:
                print(f"Pid {id} granted message received")
                writefunc(f"Process Id: {id} - {datetime.datetime.now()}\n")
                time.sleep(seconds)
                stub.Release(message)
```

Resultados

Primeira rodada de testes.

▼ Teste 0 de funcionamento.

$n = 2, r = 10, e k = 2$

```
PS C:\Users\Filipe P\Documents\_Pessoal\_UFRJ\DistributedSystems\TP3> go run server/main/main.go
Initializing server
Please choose the corresponding command:
1: Print the current queue
2: Print how many times each process has been served
3: Forcefully stop the server: Cancels all RPCs and open connections and stops the server
What would you like to do? |
```

```
PS C:\Users\Filipe P\Documents\_Pessoal\_UFRJ\DistributedSystems\TP3> python client/main.py 2 10 2
Pid 0 granted message received
Pid 1 granted message received
Pid 0 granted message received
Pid 1 granted message received
Pid 0 granted message received
Pid 1 granted message received
Pid 0 granted message received
Pid 1 granted message received
Pid 0 granted message received
Pid 1 granted message received
Pid 0 granted message received
Pid 1 granted message received
Pid 0 granted message received
```

▼ results.txt (20 linhas 2x10) - 38,250 segundos

```
Process Id: 0 - 2023-07-16 19:08:14.385568
Process Id: 1 - 2023-07-16 19:08:16.402866
Process Id: 0 - 2023-07-16 19:08:18.413894
Process Id: 1 - 2023-07-16 19:08:20.431095
Process Id: 0 - 2023-07-16 19:08:22.444128
Process Id: 1 - 2023-07-16 19:08:24.464769
Process Id: 0 - 2023-07-16 19:08:26.477323
Process Id: 1 - 2023-07-16 19:08:28.484966
Process Id: 0 - 2023-07-16 19:08:30.499585
Process Id: 1 - 2023-07-16 19:08:32.515594
Process Id: 0 - 2023-07-16 19:08:34.533208
Process Id: 1 - 2023-07-16 19:08:36.548476
Process Id: 0 - 2023-07-16 19:08:38.556950
Process Id: 1 - 2023-07-16 19:08:40.564181
Process Id: 0 - 2023-07-16 19:08:42.571313
Process Id: 1 - 2023-07-16 19:08:44.588313
Process Id: 0 - 2023-07-16 19:08:46.604815
Process Id: 1 - 2023-07-16 19:08:48.611219
Process Id: 0 - 2023-07-16 19:08:50.623412
Process Id: 1 - 2023-07-16 19:08:52.635794
```

▼ serverlog.txt

```
Message Process ID Time
REQUEST 0 2023-07-16 19:08:08.381834
GRANTED 0 2023-07-16 19:08:08.384220
REQUEST 1 2023-07-16 19:08:08.396634
RELEASED 0 2023-07-16 19:08:08.401868
```

GRANTED 1 2023-07-16 19:08:08.401868
REQUEST 0 2023-07-16 19:08:08.402866
RELEASED 1 2023-07-16 19:08:08.413311
GRANTED 0 2023-07-16 19:08:08.413311
REQUEST 1 2023-07-16 19:08:08.414475
GRANTED 1 2023-07-16 19:08:08.430001
RELEASED 0 2023-07-16 19:08:08.430001
REQUEST 0 2023-07-16 19:08:08.431637
RELEASED 1 2023-07-16 19:08:08.442910
GRANTED 0 2023-07-16 19:08:08.442910
REQUEST 1 2023-07-16 19:08:08.444676
RELEASED 0 2023-07-16 19:08:08.459947
GRANTED 1 2023-07-16 19:08:08.460505
REQUEST 0 2023-07-16 19:08:08.464769
RELEASED 1 2023-07-16 19:08:08.474907
GRANTED 0 2023-07-16 19:08:08.475548
REQUEST 1 2023-07-16 19:08:08.477323
GRANTED 1 2023-07-16 19:08:08.481631
RELEASED 0 2023-07-16 19:08:08.481631
REQUEST 0 2023-07-16 19:08:08.485544
GRANTED 0 2023-07-16 19:08:08.497685
RELEASED 1 2023-07-16 19:08:08.497071
REQUEST 1 2023-07-16 19:08:08.499585
GRANTED 1 2023-07-16 19:08:08.514464
RELEASED 0 2023-07-16 19:08:08.513929
REQUEST 0 2023-07-16 19:08:08.516389
GRANTED 0 2023-07-16 19:08:08.531756
RELEASED 1 2023-07-16 19:08:08.531756
REQUEST 1 2023-07-16 19:08:08.533208
RELEASED 0 2023-07-16 19:08:08.546908
GRANTED 1 2023-07-16 19:08:08.547326
REQUEST 0 2023-07-16 19:08:08.548527
RELEASED 1 2023-07-16 19:08:08.553658
GRANTED 0 2023-07-16 19:08:08.554464
REQUEST 1 2023-07-16 19:08:08.556950
RELEASED 0 2023-07-16 19:08:08.563677
GRANTED 1 2023-07-16 19:08:08.563677
REQUEST 0 2023-07-16 19:08:08.564181
GRANTED 0 2023-07-16 19:08:08.570241
RELEASED 1 2023-07-16 19:08:08.569755
REQUEST 1 2023-07-16 19:08:08.571313
GRANTED 1 2023-07-16 19:08:08.587254
RELEASED 0 2023-07-16 19:08:08.586807
REQUEST 0 2023-07-16 19:08:08.588313
RELEASED 1 2023-07-16 19:08:08.601735
GRANTED 0 2023-07-16 19:08:08.601735
REQUEST 1 2023-07-16 19:08:08.604815
GRANTED 1 2023-07-16 19:08:08.609860
RELEASED 0 2023-07-16 19:08:08.609860
REQUEST 0 2023-07-16 19:08:08.611319
GRANTED 0 2023-07-16 19:08:08.622853
RELEASED 1 2023-07-16 19:08:08.622328
REQUEST 1 2023-07-16 19:08:08.623412
RELEASED 0 2023-07-16 19:08:08.634212

GRANTED 1 2023-07-16 19:08:08.634681
RELEASED 1 2023-07-16 19:08:08.642940

Minimamente funcional.

▼ Teste 1 de escalabilidade do sistema.

n = 2, r = 10, e k = 2	38,250 segundos
n = 4, r = 10, e k = 2	78,484 segundos
n = 8, r = 10, e k = 2	158,932 segundos
n = 16, r = 10, e k = 2	320,285 segundos
n = 32, r = 10, e k = 2	640,785 segundos

▼ Test 2 de escalabilidade do sistema.

n = 2, r = 5, e k = 1	9,121 segundos
n = 4, r = 5, e k = 1	19,237 segundos
n = 8, r = 5, e k = 1	39,559 segundos
n = 16, r = 5, e k = 1	79,964 segundos
n = 32, r = 5, e k = 1	161,150 segundos
n = 64, r = 5, e k = 1	322,934 segundos

▼ Test 3 de escalabilidade do sistema.

n = 2, r = 3, e k = 0	0,053 segundos
n = 4, r = 3, e k = 0	0,087 segundos
n = 8, r = 3, e k = 0	0,181 segundos
n = 16, r = 3, e k = 0	0,618 segundos
n = 32, r = 3, e k = 0	1,726 segundos
n = 64, r = 3, e k = 0	3,335 segundos
n = 128, r = 3, e k = 0	11,911 segundos

Validação dos arquivos

Para validar que o acesso à zona crítica foi sequencial e organizado pelo servidor, olhamos os logs gerados no arquivo serverlog.txt e checamos utilizando o script disponibilizado pelo professor com ligeiras alterações.

```

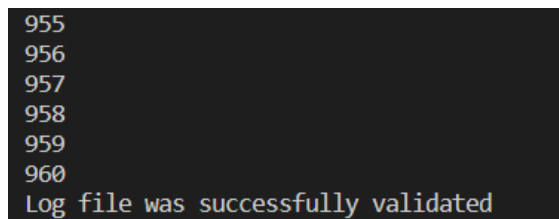
def validate():
    f = open("serverlog.txt", "r")
    lines = f.readlines()
    requests = []
    grants = []
    releases = []
    for index, line in enumerate(lines):
        if(index == 0): continue
        print(index)
        if ("REQUEST" in line):
            requests.append(int(line.split("-")[1]))
            continue
        if ("GRANTED" in line):
            if (len(grants) != len(releases)):
                print(line)
                print(len(grants))
                print(len(releases))
                raise Exception("Invalid log file: invalid grants and releases sequence")
            grants.append(int(line.split("-")[1]))
            continue
        if ("RELEASED" in line):
            if (len(releases) != len(grants) - 1):
                raise Exception("Invalid log file: invalid grants and releases sequence")
            releases.append(int(line.split("-")[1]))
            continue
        print(requests, grants, releases)
    for i in range(len(requests)):
        if (requests[i] != grants[i] or grants[i] != releases[i]):
            raise Exception("Invalid log file: invalid grants and releases sequence")

    print("Log file was successfully validated")

if __name__ == "__main__":
    validate()

```

Todos os arquivos a partir desse momento no relatório passaram com sucesso pelo processo de validação.



```

955
956
957
958
959
960
Log file was successfully validated

```

Após rodar os testes e tentativas de validar os logs, percebemos que a ordem entre escrever no arquivo de log

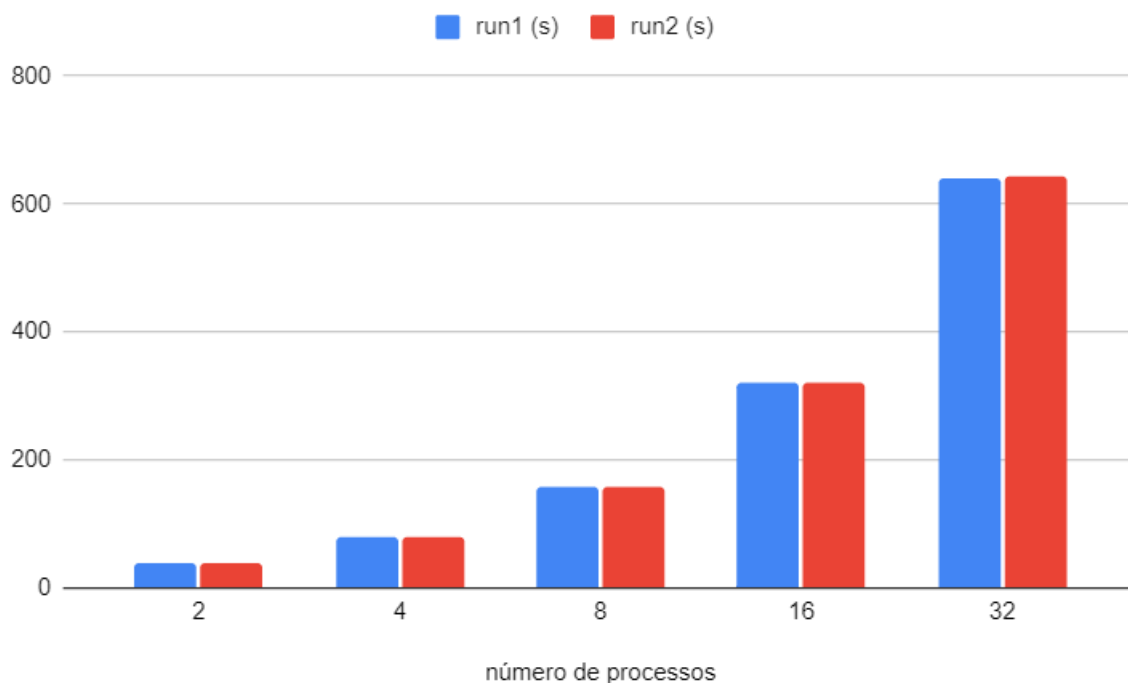
e enviar a mensagem estava equivocada - gerando 'GRANT's antes de 'RELEASE's. Após a correção rodamos novamente os testes.

Segunda Rodada de testes:

```
Fixed order of log messages
main
Pedro-gomes8 committed 2 hours ago
Showing 1 changed file with 1 addition and 1 deletion.
TP3/server/tools/TP3/semaphore/semaphore.go
180 180 @@ -180,8 +180,8 @@ func (s *Semaphore) Release(n int32) {
181 181     s.mu.Unlock()
182 182     panic("semaphore: released more than held")
183 183 }
184 184 - s.notifyWaiters()
185 185 + writingLog.WriterLog("RELEASED", id)
186 186 + s.notifyWaiters()
187 187 + s.mu.Unlock()
188 188 }
```

▼ Teste 1 de escalabilidade do sistema.

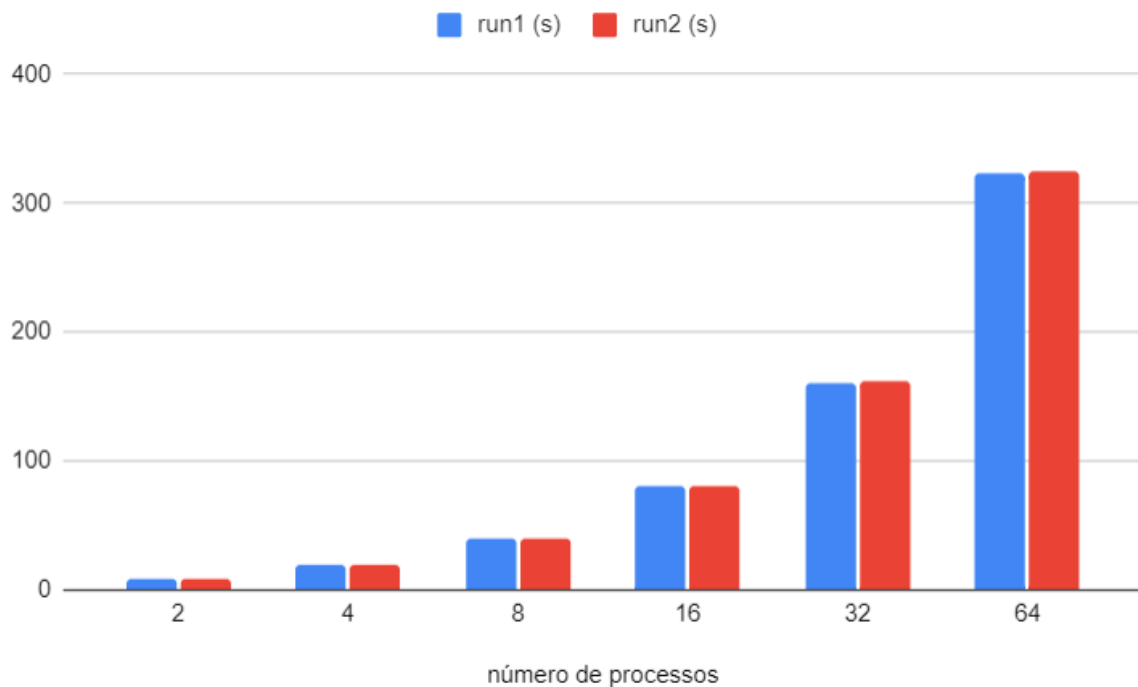
- ▼ $n = 2, r = 10, e k = 2$ - 38,241 segundos
- ▼ $n = 4, r = 10, e k = 2$ - 78,687 segundos
- ▼ $n = 8, r = 10, e k = 2$ - 159,482 segundos
- ▼ $n = 16, r = 10, e k = 2$ - 319,984 segundos
- ▼ $n = 32, r = 10, e k = 2$ - 642,384 segundos



▼ Test 2 de escalabilidade do sistema.

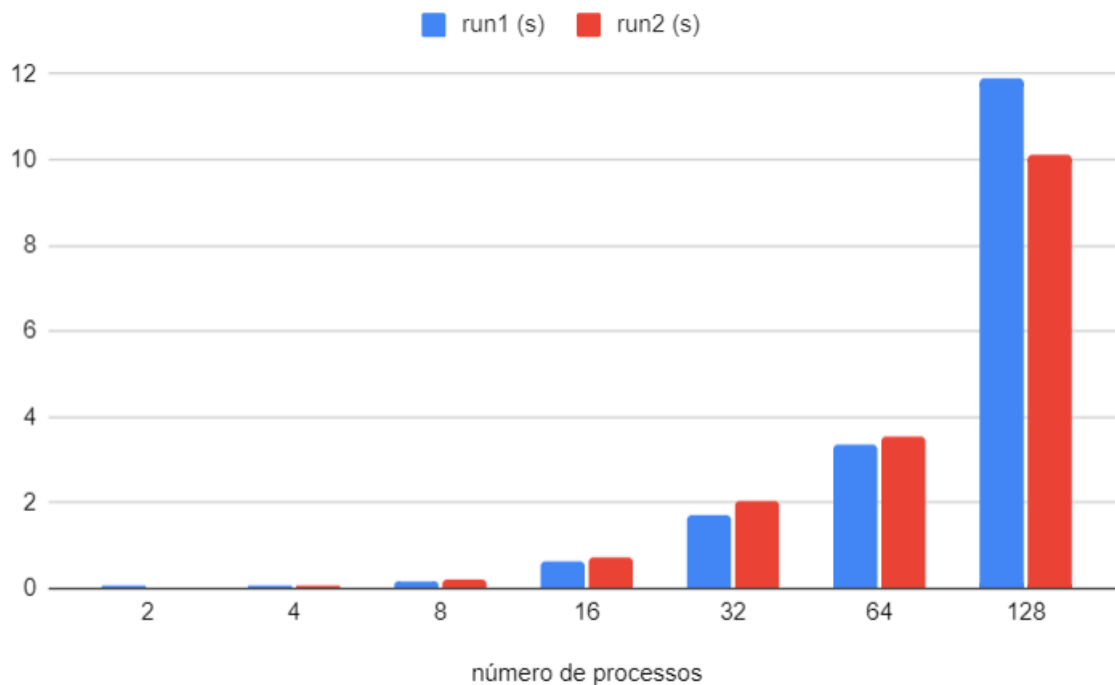
- ▼ $n = 2, r = 5, e k = 1$ - 9,110 segundos

- ▼ $n = 4, r = 5, e k = 1$ - 19,296 segundos
- ▼ $n = 8, r = 5, e k = 1$ - 39,556 segundos
- ▼ $n = 16, r = 5, e k = 1$ - 80,085 segundos
- ▼ $n = 32, r = 5, e k = 1$ - 161,411 segundos
- ▼ $n = 64, r = 5, e k = 1$ - 324,111 segundos



▼ Test 3 de escalabilidade do sistema.

- ▼ $n = 2, r = 3, e k = 0$ - 0,035 segundo
- ▼ $n = 4, r = 3, e k = 0$ - 0,083 segundos
- ▼ $n = 8, r = 3, e k = 0$ - 0,210 segundos
- ▼ $n = 16, r = 3, e k = 0$ - 0,728 segundos
- ▼ $n = 32, r = 3, e k = 0$ - 2,019 segundos
- ▼ $n = 64, r = 3, e k = 0$ - 3,565 segundos
- ▼ $n = 128, r = 3, e k = 0$ - 10,104 segundos



Conclusões

Dado o fato de que o servidor precisa organizar o acesso exclusivo à zona crítica, e ordenar os pedidos que chegam nele de maneira sequencial, o tempo de conclusão acaba sendo muito correlacionado (quase linearmente) à quantidade de acessos à zona crítica necessária ($n * r$).

Especialmente no teste de escalabilidade 1 e 2, onde o tempo de espera entre pedidos é mais significativo, dobrar a quantidade de processos resulta no dobro do tempo necessário para gerar o arquivo; Faz sentido, já que a maior parte do tempo percorrido vem do tempo de espera entre requisições, quando o processo pede o REQUEST, recebe seu GRANT rapidamente, o tempo total é majoritariamente determinado por $(n * r * k)$ segundos.

Já no teste de escalabilidade 3, com $k = 0$, caminhamos pra uma situação menos previsível onde todos os processos constantemente pedem acesso à região crítica, mas o servidor ainda consegue ordenar e excluir mutuamente o acesso. O resultado é que saímos um pouco do regime linear, de maneira que duplicar a quantidade de processos (e requisições/acessos à ZC necessários) mais que duplica o tempo necessários, especialmente no caso de 128 processos.

Isso se dá, provavelmente, ao fato da limitação do servidor à ter 100 sockets conectados ao mesmo tempo, nos testes de escalabilidade 1 e 2, dado que o tempo de sleep é o que mais demora, não temos todos os processos ao mesmo tempo tentando se comunicar com o servidor, o que é verdade no teste 3. Mesmo com essa superlinearidade, o tempo total necessário para resolver todos os pedidos é muito mais rápido ao deixar os pedidos pedirem sem tempo de espera, já que no nosso caso a ZC é $O(1)$ e os processos acabam esperando muito mais que efetivamente executando.