Documentação Técnica: Reliable Multicast com Lamport Clocks

Equipe: Thiago Coutinho, Bruno Castro, Rodrigo Queiroz, Elis, Malu Brito

---

1. Arquitetura e Funcionamento

Visão Geral:

Implementação de um sistema distribuído com multicast confiável usando TCP sockets e relógios lógicos de Lamport. Cada processo (`P1`, `P2`, `P3`) atua como servidor e cliente:

- Servidor: Escuta em uma porta específica (ex: `localhost:5001`)
- Cliente: Conecta-se aos outros processos via TCP
- Protocolo: Flooding com controle de relógio lógico

Fluxo de Mensagens:

1. Processo envia mensagem com timestamp de Lamport
2. Destinatários atualizam seu relógio lógico (`max(relógio_local, timestamp_recebido) + 1`)
3. Mensagem é entregue e registrada com novo timestamp

Componentes:

- `process.py`: Lógica principal dos processos
- `config.json`: Configuração de hosts/ports dos processos

---

2. Trechos Relevantes de Código

Configuração Inicial (process.py):

```
class Processo:
    def __init__(self, id_processo, config_grupo, modo_auto=False):
        self.id = id_processo
```

```
        self.relogio_lamport = 0  # Inicializa relógio lógico
        self.entregues = set()    # Conjunto de IDs de mensagens entregues
        # Configura servidor TCP
        self.socket_servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket_servidor.bind((self.host, self.porta))
        # Conecta a outros processos
        for pid in config_grupo:
            if pid != self.id:
                    self.conectar_processo(pid, host, porta)
```

Envio Multicast Confiável:

```
def multicast_confiavel(self, dados):
    self.relogio_lamport += 1  # Incremento local antes do envio
    msg = {
        'remetente': self.id,
        'seq': self.contador_sequencia,
        'dados': dados,
        'timestamp': self.relogio_lamport
    }
    # Auto-registro para evitar retransmissão desnecessária
    self.entregues.add((self.id, self.contador_sequencia))
    # Envio para todos os processos conectados
    for pid in self.sockets:
      self.sockets[pid].sendall(json.dumps(msg).encode())
```

Recebimento e Sincronização Lamport:

```
def processar_mensagem(self, msg):
    # Atualização do relógio de Lamport
    self.relogio_lamport = max(self.relogio_lamport, msg['timestamp']) + 1

    if (msg['remetente'], msg['seq']) not in self.entregues:
        self.entregues.add((msg['remetente'], msg['seq']))
        # Retransmissão para outros processos (flooding)
        for pid in self.sockets:
            if pid != msg['remetente']:
                    self.enviar_mensagem(pid, msg)
```

---

3. Algoritmo de Multicast e Lamport

Multicast Confiável (Flooding):

1. Identificação Única: Cada mensagem possui ID `(remetente, sequência)`

2. Retransmissão: Processos retransmitem mensagens inéditas
3. Entrega: Mensagem é entregue apenas uma vez por processo

Relógios de Lamport:

1. Incremento Local:

```
self.relogio_lamport += 1  # Antes de enviar mensagem
```

2. Sincronização na Recepção:

```
self.relogio_lamport = max(self.relogio_lamport, msg['timestamp']) + 1
```

3. Ordenação Lógica: Garante relação "happened-before" entre eventos

---

4. Execução e Timestamps

Saída de Terminal (Modo Automático):

```
[P1] [Lamport=1] 10:00:00.123 - Enviado: Olá a todos!
[P2] [Lamport=2] 10:00:00.234 - Entregue: Olá a todos! | De=P1
[P3] [Lamport=2] 10:00:00.245 - Entregue: Olá a todos! | De=P1
[P2] [Lamport=3] 10:00:02.450 - Enviado: Como vocês estão?
[P1] [Lamport=4] 10:00:02.465 - Entregue: Como vocês estão? | De=P2
[P3] [Lamport=3] 10:00:02.480 - Entregue: Como vocês estão? | De=P2
[P3] [Lamport=4] 10:00:05.789 - Enviado: Mensagem final de P3
[P1] [Lamport=5] 10:00:05.800 - Entregue: Mensagem final de P3 | De=P3
[P2] [Lamport=5] 10:00:05.810 - Entregue: Mensagem final de P3 | De=P3
```

Análise de Ordenação:

Evento	P1	P2	P3
Envio "Olá" (P1)	1	-	-
Recepção "Olá" (P2)	-	2	-
Recepção "Olá" (P3)	-	-	2

Envio "Como" (P2)	-	3	-
Recepção "Como" (P1)	4	-	-
Recepção "Como" (P3)	-	-	3→4
Envio "Final" (P3)	-	-	4
Recepção "Final" (P1)	5	-	-
Recepção "Final" (P2)	-	5	-

---

5. Referências

1. LAMPORT, L. Time, Clocks, and the Ordering of Events in a Distributed System. 1978.
2. TANENBAUM, A. S. Distributed Systems: Principles and Paradigms. Cap. 5.
3. Documentação Python: socket