UNIVERSIDADE FEDERAL DE SERGIPE – SISTEMAS DE INFORMAÇÃO
SISTEMAS DISTRIBUÍDOS
Verenilson da Silva Souza, Guilherme Andrade de Medeiros

Aplicação do Algoritmo de Berkeley

1. Introdução:

Antes de nos aprofundarmos no código, vamos ver uma breve explicação, o algoritmo de Berkeley é um método para sincronizar os relógios em um sistema distribuído. Ele funciona da seguinte maneira:

1. O servidor de tempo envia uma mensagem para todos os clientes solicitando a leitura de seus relógios.

2. Os clientes respondem com a leitura de seus relógios e a hora de recebimento da mensagem do servidor.

3. O servidor calcula a média das leituras dos relógios dos clientes e o tempo de viagem de ida e volta da mensagem.

4. O servidor envia uma mensagem para todos os clientes com a média dos relógios e o tempo de viagem de ida e volta da mensagem.

5. Os clientes ajustam seus relógios para a média dos relógios mais o tempo de viagem de ida e volta da mensagem.

Vantagens do Algoritmo de Berkeley:

- É simples de implementar.
- É tolerante a falhas de clientes.
- É escalável para grandes sistemas distribuídos.

Desvantagens do Algoritmo de Berkeley:

- Não é preciso, pois depende do tempo de viagem da mensagem.
- Pode ser lento em redes com alta latência.

Aplicações do Algoritmo de Berkeley:

- Sincronização de relógios em sistemas de computação em nuvem.
- Sincronização de relógios em sistemas de comunicação.
- Sincronização de relógios em sistemas de controle distribuídos.

2. Explicando o código:

Para melhor entender como o programa funciona, vamos ver brevemente como ele está estruturado:

1. Classe ServidorUDP:

Esta classe implementa um sistema de sincronização de horário entre um servidor master e vários servidores slaves, o master solicita os horários dos slaves e calcula a nova média de horário. A nova média de horário é enviada para os slaves, que ajustam seus relógios de acordo com o master.

1.1. Atributos:

- **myId**: Porta que o servidor está escutando.
- **idMaster**: Porta do servidor master.
- **portaMasterMultCast**: Porta multicast utilizada pelo master para enviar mensagens.
- **portaSlaveMultCast**: Porta multicast utilizada pelos slaves para enviar mensagens.
- **novaMediaDeHorarioEmMinutos**: Armazena a nova média de horário após o cálculo.
- **myTimeZone**: Armazena o horário local do servidor.
- **horasEmMinutosDosServidores**: Lista que armazena os horários dos slaves em minutos.

1.2. Métodos:

run(): Método principal que é executado quando a thread do servidor é iniciada.
Verifica se o servidor é o master ou um slave e chama a função masterUDP() ou slaveUDP() respectivamente.
masterUDP(): Implementa a lógica do servidor master.

- Envia uma mensagem multicast para os slaves solicitando seus horários.
- Espera 3 segundos para receber as respostas dos slaves.
- Calcula a nova média de horário a partir dos horários recebidos.
- Envia a nova média de horário para os slaves via multicast.
- Calcula a diferença entre o horário local e a nova média de horário e imprime na tela.

masterCalculaNovoHorario()*:* Calcula a nova média de horário a partir dos horários recebidos dos slaves.
masterEsperaRespostas(): Recebe as respostas dos slaves com seus horários.
enviarMultcastUDP(): Envia uma mensagem multicast para um determinado endereço e porta.
slaveUDP(): Implementa a lógica do servidor slave.

- Espera a mensagem multicast do master solicitando o horário.
- Envia seu horário para o master via multicast.
- Espera a mensagem multicast do master com a nova média de horário.
- Calcula a diferença entre o horário local e a nova média de horário e imprime na tela.
- slaveEsperaMensagem(): Espera a mensagem multicast do master com a nova média de horário.
- pegarHorarioEmMinutos(): Converte o horário local em minutos.
- resultadoFinalDeHorario(): Calcula a diferença entre o horário local e a nova média de horário e imprime na tela.
- sleep(): Faz a thread dormir por um determinado tempo em milissegundos.

2. Classe HorarioDoPC:

Esta classe é responsável por calcular a média dos três horários informados.

2.1. Atributos:

Método *GetHorarioAtualDoPc():*

- Obtém a data e hora atual do sistema usando a classe LocalDateTime.
- Cria um array bidimensional horarios contendo 3 horários (hora, minuto, segundo).
- Calcula a soma das horas, minutos e segundos dos 3 horários armazenados no array.
- Calcula a média das horas, minutos e segundos.
- Ajusta a média dos minutos e segundos para considerar valores excedentes (por exemplo, se a média dos segundos for 65, ajusta para 5 e incrementa os minutos em 1).
- Exibe a média calculada no formato HH:MM:SS.
- Retorna uma string contendo a data e hora atual do sistema.

3. Classe App:

Esta classe inicia 3 servidores UDP individualizados, cada um com um horário inicial diferente. Os servidores utilizam a classe *ServidorUDP* previamente explicada para sincronizar seus horários através do protocolo UDP multicast. É utilizada a classe *LocalDateTime* para representar as datas e horários.

3.1. Atributos:

Método *main(String[] args) throws Exception*:

- Cria um array de threads udp com 3 elementos para representar os servidores individuais.
-

- Define um array timeZone de objetos LocalDateTime para armazenar os horários iniciais de cada servidor.
- Preenche o array timeZone chamando a função getDateTimes().
- Percorre o array udp e para cada elemento:
- Cria uma nova instância da classe ServidorUDP com os parâmetros:
- Porta individual do servidor (a partir de 6001)
- Porta do servidor master (6000)
- Horário inicial do servidor (obtido do array timeZone)
- Inicia a thread associada ao servidor usando start().

Método *getDateTimes()*:

- *Cria um array timeZone de objetos LocalDateTime com tamanho 3.*
- *Define horários iniciais específicos para cada servidor:*
    - *Servidor 1: Hora 3h00min00s*
    - *Servidor 2: Hora 2h50min00s*
    - *Servidor 3: Hora 3h25min00s*
- *Retorna o array timeZone preenchido com os horários iniciais.*

4. Conclusão:

Este projeto fornece uma base sólida para um sistema de sincronização de horário. Com algumas melhorias e adaptações, podem ser utilizados em diversos tipos de aplicações que necessitem de sincronização precisa de tempo entre vários dispositivos.