Nome: João Felipe Amaral Santiago
Matrícula: 201720884

Funcionamento Geral

Inicialmente o middleware irá tentar encontrar os participantes do grupo, enviando mensagens via IP Multicast no endereço 239.0.0.0 porta 4321. Nesta etapa de identificação, CausalMulticast irá descobrir e armazenar os endereços IP unicast dos participantes, definirá as portas para posterior comunicação unicast, definirá o índice que o usuário terá na matriz lógica e o tamanho da mesma.
Posteriormente serão explicados em mais detalhes aspectos de implementação desta etapa inicial.

Finalizada tais operações iniciais, o usuário já está apto a enviar mensagens aos participantes do grupo. Portanto, quando houver o interesse de se comunicar com os demais membros será necessário chamar o mcsend().

Em síntese, o método mcsend() lê a mensagem que o usuário quer enviar e cria um pacote com informações adicionais as quais serão necessárias para viabilizar ordem, estabilização e comunicação unicast. Depois de agregar tais informações, o método irá questionar de que modo o pacote deve ser enviado. Há 3 possibilidades de envio:

- Enviar mais tarde;
- Enviar para todos ou
- Enviar para um usuário específico.

Depois de enviar via socket UDP unicast as mensagens, o vetor e a matriz lógica serão atualizados.

Além do envio, o middleware gerenciará o recebimento dos pacotes. Desse modo, cada usuário terá um socket UDP ouvido na porta (definida na execução inicial do middleware) adequada, esperando novos dados. Quando um pacote chegar, 3 operações serão feitas.

- Depósito dos pacotes no buffer de mensagens;
- Atualização da Matriz Lógica e
- Política de envio da mensagem à camada de aplicação do usuário.

A última funcionalidade geral do middleware é a execução do algoritmo de estabilização de mensagens.

Aspectos de Implementação

A seguir são descritos os funcionamentos dos principais métodos e threads do middleware.

FindMembers

Seu construtor definirá os valores de IP multicast, porta do usuário (pode mudar ou não, no decorrer da execução) e ID do usuário.
O método run() de sua thread, inicialmente irá se comunicar, via socket UDP com o DNS da google. Tal conexão servirá apenas para viabilizar a descoberta do IP unicast (classe B) da máquina que

roda o pacote CausalMulticast. Em seguida, um novo socket UDP será criado de tal forma que irá participar da conexão IP Multicast, no endereço e porta explicados anteriormente. O próximo passo é criar a mensagem para ser enviada na comunicação multicast. A seguir é mostrado o formato da mensagem enviada em FindMembers.

|msg1| |IP do usuário| |ID| |porta| |índice no vetor lógico| |TIMES| |SP|

msg1 = “UNICAST_IP_INFOS”.
IP do usuário = resultado da conexão com o DNS (8.8.8.8).
ID = nome que o usuário define na chamada do middleware.
porta = inicialmente começa em 1010.
índice no vetor lógico = inicialmente é 0.
TIMES = quantidade de mensagens que o usuário enviou na comunicação multicast.
SP = flag de que indica se a porta está definida ou não

Quando FindMembers recebe um pacote ele verifica se o IP do usuário, ID do mesmo e a porta já estão na lista de membros do middleware. Caso eles não estejam, verificações adicionais serão feitas, do contrário a thread irá dormir por 500 milisegundos.

No momento em que a verificação constar que é necessário analisar mais a fundo o pacote, a thread irá checar a validade do valor da porta e do índice que acabou de receber respeitando o seguinte critério.

```
SE (user_port =< msg_port E user_time < msg_time)
user_port = msg_port + 1;

SE (user_index =< msg_index E user_time < msg_time)
      user_index = msg_index + 1;
```

Tais condições visam garantir que o usuário 0 usará a porta 1010 e terá o índice 0 na matriz lógica, posteriormente o usuário 1 usará a porta 1011 e seu índice será 1, e assim sucessivamente. As definições de porta e índice na matriz lógica são realizadas, portanto, em tempo de execução e não em nível de programação, o que significa que há um certo tempo necessário até que todos os participantes consigam definir adequadamente tais parâmetros. Por essa razão dados de usuários só são adicionados na lista de membros depois que TIMES > 3;

Por último, o vetor e a matriz lógica de cada usuário aumentam suas dimensões à medida que novos membros são identificados.

Outro ponto importante é o fato de que a thread FindMembers roda a cada 500 milisegundos.

mcsend()

Toda vez que um usuário tiver interessado em enviar mensagens para os membros do grupo precisará executar esse método. Ele começa perguntando ao usuário se ele quer enviar uma nova mensagem para todos ou não. Em seguida, a mensagem a ser transmitida pelos demais membros recebe algumas informações adicionais. A seguir elas são expostas em mais detalhes.

|msg| |VC| |MC| |SENDER| |VC[SENDER]| |port| |dlvrd|

msg = mensagem que o usuário deseja que seja entregue aos membros do grupo

VC = vetor lógico de ordenamento do remetente
MC = vetor lógico de estabilização do remetente
SENDER = ID do remetente
VC[SENDER] = índice do remetente
porta = porta do remetente
dlvrd = 0 caso a mensagem ainda não tiver sido entregue, do contrário é 1

Caso o usuário escolha enviar para todos, um método será chamado para que todos os membros da lista recebam o pacote. Do contrário, será instanciada a classe OBJ_sen_after para cada membro existente na lista de membros. Por sua vez, cada instância dessa classe será adicionada na lista Late_sends(). Tal mecanismo, permite que, em momento oportuno, cada mensagem atrasada possa chegar corretamente para seus destinos adequados.

Após atualizar Late_sends, o usuário precisará definir de que modo ele enviará as mensagens atrasadas. As opções foram explicadas previamente e podem ser checadas na página 1 desta documentação. A medida que as mensagens atrasadas são entregues, a lista Late_sends é atualizada, removendo tais mensagens. Por fim, o vetor e a matriz lógica são incrementadas, seguindo os algoritmos de estabilização e ordenamento causal.

O envio de mensagens é feito de maneira unitária, levando em consideração IP e porta do destinatário, tais dados estão presentes na lista de membros.

rcv_pkg

Esta thread recebe os pacotes enviados por mcsend(), ela irá configurar um socket UDP para escutar na porta definida na thread FindMembers. Como explicado anteriormente, a definição de portas e índices são feitas em tempo de execução, isso significa que rcv_pkg não pode ouvir em um socket válido enquanto que essas questões não estejam estabelecidas. Portanto, rcv_pkg, irá aguardar, indiretamente, até que TIMES > 3.

Concluída a etapa de estabilização da porta, o socket UDP começará a ouvir novas mensagens. Como explicado anteriormente, quando um pacote chega é realizado deposito do que chega no buffer, atualização da matriz lógica e verificação do ordenamento.

deposit()

Cada vez que esse método é chamado o buffer recebe um novo valor, copiando os dados antigos em um buffer auxiliar, para depois aumentar a capacidade de elementos, que em seguida receberá todos os valores do buffer auxiliar e enfim, na última posição o novo elemento será adicionado.

order_pkgs()

Haverá um loop que varrerá todo o buffer de mensagens.
Primeiramente, o método verifica o valor do pacote no frame |DLVRD|, o qual indica se a mensagem em questão já foi ou não entregue ao usuário. Caso DLVRD = 0, significa que a informação ainda não chegou ao usuário e a verificação segue. A próxima etapa é comparar o vetor lógico associado à mensagem com o vetor lógico do usuário que executa o método order_pkgs().

Caso a comparação indique que a entrega é válida, DLVRD deve ser setado para 1 e enfim a mensagem é enviada ao usuário.

Outro aspecto relevante de order_pkgs é o fato dele repetir o verificação de todos pacotes presentes nos buffers a cada nova mensagem entregue. Isso segue a lógica de que, se uma mensagem foi entregue, talvez outras já estejam aptas a chegar na camada de aplicação.

discart()

Utiliza um buffer auxiliar, copiando todos os elementos do buffer principal. Posteriormente é feito um redimensionamento do buffer principal, com capacidade reduzida. Por último, todos os elementos presentes do buffer auxiliar, com exceção do pacote que se quer excluir, são realocados no novo buffer.

Stabilize_pkgs

Esta thread é executada a cada 27 segundos. É realizada uma varredura por todos os elementos do buffer de mensagens. Lê-se a posição do remetente e do msg.VC de cada pacote. Por último, analisa-se se a condição de descarte seja satisfeita.

Aplicação

Implementa um simples chat, de tal modo que a cada nova mensagem recebida do middleware uma lista de strings adiciona tal nova informação. Há, na main(), um loop infinito o qual printa todos os elementos da lista. Após pintar as mensagens um scanner irá aguardar por mensagens que o usuário tenha interesse de enviar.
Há uma mensagem especial na aplicação. Caso o usuário digite `appchat -r` as mensagens serão reapresentadas no terminal, do contrário tudo que é coletado pelo scanner é enviado para mcsend().