Universidade Federal do Rio Grande do Sul - Instituto de Informática
INF0115 - Sistema Operacionais II

19/08/2024

Trabalho Prático Parte 2:

WakeOnLan

Prof. Dr. Weverton Luis da Costa Cordeiro
Henrique Carniel da Silva (*hcsilva@inf.ufrgs.br*) - 00335626
Ian Kersz Amaral (*ikamaral@inf.ufrgs.br*) - 00338368
Nathan Alonso Guimarães (naguimaraes@inf.ufrgs.br) - 00334437

Descrição do ambiente de testes

Para efetuar os testes no trabalho, foi utilizado um ambiente Docker baseado em Ubuntu 22.04 LTS. Como compilador, o g++ versão 12 foi utilizado, pois é o mesmo disponível nas máquinas do instituto de informática. O código também foi testado nas máquinas disponibilizadas em sala de aula.

Eleição de líder

A implementação utilizada para a eleição de líder foi o algoritmo do *bully*/valentão. O identificador de eleição primário utilizado foi a versão da tabela de replicação, assim garantindo que somente clientes com a tabela mais atualizada serão eleitos como *backup* primário (líder/manager). Como critério de desempate, o IPv4 das máquinas é utilizado, pois é um número de fácil acesso, já que é recebido em todos os pacotes, e que não haverá colisão. Após uma eleição, todos os clientes tentam procurar pelo novo *manager* eleito.

Dentro do subserviço de eleição, é feito o setup do socket UDP utilizado durante a eleição, tão como é adquirido o valor do IPv4 da máquina atual. Após isso, é iniciado um loop semi-infinito, até que a variável global compartilhada “run” seja

falsa. Dentro desse loop, o código verifica sempre por duas condições que levam ao mesmo caminho final:

1. Uma requisição de forçar a eleição (“force_election” = *true*) é feita por outro subserviço do programa.
2. É recebido um pacote de *Sleep Service Election* (SSEL) pelo subserviço.

No caso que uma dessas condições seja verdadeira, significa que uma eleição deve ser feita. A eleição retorna se a estação atual foi eleita como manager ou não. Esse retorno é salvo na variável compartilhada “is_manager”, usada nos demais subserviços. Por último, a variável compartilhada “force_election” é sempre colocada para falso.

Dada que uma eleição foi convocada, as máquinas começam a enviar pacotes por um socket aberto na porta 14444, reservada para lidar com esse subserviço. Todas as máquinas trocam entre si, por *broadcast*, um pacote SSEL contendo seus identificadores, enquanto souberem que ainda possuem a chance de se tornarem *manager*. Caso a máquina venha a receber neste socket um pacote contendo um identificador menor (ou igual, mas com máquina de origem com IPv4 menor) do que o seu próprio, esta responde com um pacote de *Sleep Service Election Greater* (SSELGT). Quando qualquer máquina recebe um pacote de SSELGT, a mesma sabe que há alguém com o identificador maior e que, portanto, não tem chance de se tornar *manager* nesta eleição. Isso tudo ocorre em no máximo 5 turnos, os quais garantem que todos os computadores conectados na rede possam enviar seus números na eleição, somente passando de turno quando não existe mais nenhum pacote em trânsito.

Os casos em que uma eleição é forçada são quando:

1. O subserviço de eleição se inicia pela primeira vez, para garantir, de forma simples, que sempre haja um *manager* no sistema.
2. O *manager* se desconecta do serviço, sendo perceptível do lado do cliente pelo recebimento de um pacote do tipo *Sleep Service Exit* (SSE).
3. O *manager* recebe um pacote de *Sleep Service Request* (SSR), pois este pacote pode ser somente enviado por um *manager*, logo, existem, erroneamente, pelo menos 2 *managers* ativos no sistema.

4. Um cliente detecta que não recebeu pacote de SSR há muito tempo, identificando um *timeout* do *manager*.
5. Um *manager* recebe um pacote de SSR_ACK, mas o identificador enviado não pertence a ele. Portanto, o cliente tem outro *manager*.

Replicação

A estratégia de replicação passiva escolhida se baseia em ter uma tabela de *backup*, a qual é diferente da tabela principal que é utilizada para as outras funções e inclui todos os usuários da aplicação. Quando um processo identifica que ele transicionou de cargo, é efetuado uma ação corresponde na tabela principal, sendo elas:

- **Cliente para *manager*:** copiar a tabela de *backup* para a tabela principal, com exceção se sua própria entrada.
- ***Manager* para cliente:** remover todos os elementos da tabela principal, tirando o *manager* atual, se ele existir nela já.

Como *manager*

O *manager* tenta consumir uma atualização enviada por outro subserviço para a tabela de *backup*, das opções que podem ser:

1. ***ADD*:** um computador foi adicionado na rede.
2. ***REMOVE*:** um computador foi removido da rede.
3. ***CHANGE*:** o *status* de um computador foi modificado.

Caso esse consumo seja mal sucedido, o subserviço de replicação dorme por um tempo até tentar consumir novamente. Caso contrário, a versão da tabela é incrementada, e juntamente com a tabela de *backup*, serializada em um pacote de *Sleep Service Replication* (SSREP). Esse pacote é enviado por *broadcast* na porta designada para esse subserviço (13333).

Como cliente

Pacotes de SSREP são recebidos, desserialiados e suas duas componentes separadas, sendo elas a versão da tabela *backup* e tabela em si, a variável da tabela de *backup* é então atualizada com a nova tabela recebida, e a variável compartilhada “table_version” é atualizada com o número recebido.

Problemas e Soluções

Como mencionado anteriormente, para fazer os testes em rede, foi utilizado um ambiente *Docker*, o qual instancia 4 computadores que iniciam o serviço de *wakeonlan*. Durante os testes com essa ferramenta, o serviço de eleições estava funcionando de forma previsível e consistente, agindo de forma correta em todos os testes. Quando testado no ambiente real das máquinas do Instituto de Informática, foi identificado problemas no serviço de eleição, os quais não apareciam no ambiente controlado do Docker. Para solucionar esse problema, os próximos testes foram executados somente nas máquinas físicas do instituto até arrumar todos os problemas encontrados.

Um dos problemas encontrados foi a utilização de *timeouts* para o *manager* dormir, eles não ocorriam de forma consistente. Em algumas máquinas, após acordar do sono, o *timeout* havia passado, já em outras, mesmo dormindo fisicamente muito mais do que o *timeout*, com grande diferença de magnitude, elas não se identificavam como *timed-out*, mesmo executando a partir do mesmo código fonte. Para corrigir isso, a implementação da auto-identificação que o manager dormiu foi transferida para a recepção de pacotes, que, em teoria são impossíveis, caso o serviço se encontre em um estado coerente, como um manager receber um pacote só enviado por outros managers.

Outro problema enfrentado foi que não havíamos contado, nas primeiras versões do código, com todos os casos em que se deve forçar uma eleição. Um dos casos não contabilizados era sobre como lidar quando se detecta a presença de 2 (ou mais) *managers* simultâneos: a detecção do problema era trivial, mas não sabíamos como resolver. Até que pensamos numa solução elegante, utilizando os

outros subserviços implementados na parte 1, como a detecção de pacotes exclusivos de um *manager* feitos por outro *manager*.