UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
DEPARTAMENTO DE INFORMÁTICA APLICADA

TRABALHO DE SISTEMAS OPERACIONAIS II - PARTE 2
REPLICAÇÃO E ELEIÇÃO DE LÍDER
EQUIPE MEAN GIRLS

Guilherme Ribeiro Rodrigues (0024299)
Flávio Eduardo Keglevich de Buzin (00229724)
Lucas Matheus Dias Brum (00261614)
Bruno Longo Farina (00263412)

- Descrição do ambiente de testes:
    - Ambiente 1: Computadores disponíveis no laboratório do INF da UFRGS.
      Especificações: 4 GB de memória RAM, processador AMD Phenom II X2 B55, 2 núcleos, clock máximo de 3 GHz.
      Sistema operacional: Ubuntu 16.04.
      Compilador: gcc (Ubuntu 5.4.0-6ubuntu1~16.04.12) 5.4.0 20160609

    - Ambiente 2: Máquinas virtuais rodando no VirtualBox conectadas na mesma rede NAT emulada.
      Especificações: 2 GB de memória RAM, processador emulado Intel Core i7-1260P, 12 núcleos, clock máximo de 4,7 GHz.
      Sistema operacional: Xubuntu 22.04.2 LTS.
      Compilador: gcc 11.3.0 (Ubuntu 11.3.0-1ubuntu1~22.04)

- Funcionamento do algoritmo de eleição de líder implementado, justificando a escolha:

    O algoritmo utilizado é uma versão levemente modificada do *Algoritmo do Valentão*. A principal diferença é que cada máquina possui uma cópia das tabela do status global da rede, junto com um relógio que cresce monotonicamente a cada modificação na tabela; em cada passagem de mensagens durante a execução do algoritmo, é passada também a tabela global, de tal forma que, se uma máquina X recebe uma tabela com um relógio mais recente, ela assume que essa tabela é a mais correta, e sobrescreve a sua representação privada. O objetivo é permitir que as tabelas mais recentes (e que representem melhor o estado do sistema) sejam propagadas com mais facilidade e frequência. Com exceção desse detalhe, o algoritmo permanece o mesmo.

    Foi escolhido o *Algoritmo do Valentão* pela simplicidade, por garantir que sempre haverá um líder em operação e também porque as suposições que ele faz são verdadeiras no nosso ambiente de rede. O contra é que pode levar a múltiplas eleições em curtos intervalos de tempo se a rede for instável ou os nós falharem com frequência.

- A implementação da replicação passiva na aplicação:

    - uso da diretiva `gethostname(char *name, size_t len)` que retorna o nome do host terminado em NULL em name, que terá um tamanho de *len* bytes. O retorno da função é do tipo int, que retorna zero se conseguiu atribuir o hostname com sucesso ou -1 em caso de erro;

    - *multicast*: efetua um unicast iterado entre todos os hosts do sistema (emulando um sistema multicast) para que todos os hosts tenham uma copia da tabela (replicação passiva);

    - *multithread*: threads para monitoramento, recebimento e envio de tabela, rotina de servidor, enviar estado corrente, identificação de novos clientes e atualização da tabela, cliente mandando mensagens de “keep-alive” a fim de organização da tabela;

    - *funções relevantes*: *electionroutine*, *checkCurrentStatus*, *receiveNewConnections*, *send_table*, *serverRotine*, *receive_table*, *sendCurrentStatus*, *monitoring;*

    - *tabela*: uma espécie de tabela de índices na qual é persistido os itens com dados como ip number, mac address, status (awaken ou asleep), flag isServer (servidor do sistema ou nao) e posição. Também possui um clock, relógio lógico que é incrementado depois de comandos de atualização da tabela (adição/remoção na/da tabela) e mudança de estado;

    - macros com números de portas para conexão dos sockets:
      #define PORT 40000 -> conexão normal
      #define PORTBROADCAST 5000 -> porta para conexão broadcast
      #define PORTSTATUSBASE 5000 -> usado pra monitorar cada cliente (com deslocamento que é a posição na tabela)
      #define MAXCONNECTIONS 3 -> define quantos hosts vão se conectar no sistema
      #define PORTSENDIPSERVER 6000 -> usado pra enviar o ip do servidor em broadcast
      #define PORTNEWCONNECTION 7000 -> para recepção de mensagens novas
      #define PORTSENDTABLE 8000 -> porta para envio da tabela

    - Uso de mutex para escrita e leitura na tabela (lock e unlock);

    - Uso de semáforos para começar threads de monitoramento pra cada cliente;

    - *Algoritmo do Valentão*: para realizar a eleição do novo líder.

- Problemas e desafios encontrados:
    - Dificuldade de *debugar* a aplicação fora do ambiente dos laboratórios do INF, portanto foi criado também um ambiente virtual no VirtualBox (parcialmente resolvido!);
    - Dificuldade na programação de uma aplicação de rede mais complexa usando a linguagem C (sistema de tipos, manipulação de *strings*, código verboso, tratamento de erros, etc);
    - Problemas na funcionalidade de obter o *hostname* da máquina rodando a aplicação (resolvido!);
    - Problemas na funcionalidade de obter o número de IP da máquina rodando a aplicação, considerando múltiplas placas de rede diferentes (resolvido!);
    - Implementar o algoritmo de eleição de líder levando em conta as atualizações da tabela e a possibilidade de tabelas desatualizadas;
    - *Bug* no algoritmo de eleição onde as máquinas entram em *loop* e não conseguem decidir um líder;
    - Dificuldades em entender o fluxo de controle de uma aplicação em C com tantas *threads* rodando partes distintas da aplicação;
    - Problemas de *race conditions* na comunicação tanto na rede, quanto entre as *threads* locais;
    - Não foi implementada a funcionalidade do envio protocolo Wake-on-Lan.