UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL - UFMS FACULDADE DE COMPUTAÇÃO - FACOM

Disciplina: Redes de Computadores

Trabalho 1: Rede de Compartilhamento de Arquivos - RCA

Autor: Vinícius Santos Silva

Professora: Profª. Hana Karina S. Rubinsztejn

Data: Junho de 2025

Sumário:

---

1. Descrição da Rede de Compartilhamento de Arquivos (RCA)

A Rede de Compartilhamento de Arquivos (RCA) implementada neste trabalho é um sistema peer-to-peer (P2P) onde cada instância do programa opera simultaneamente como cliente e servidor. Isso significa que qualquer nó na rede pode tanto solicitar arquivos de outros nós quanto atender a solicitações de arquivos de outros.

Ao ser iniciado, cada programa (peer) identifica os arquivos presentes em seu diretório corrente, preparando-os para compartilhamento. A descoberta de outros peers e dos arquivos disponíveis na rede ocorre por meio de mensagens de broadcast. Um peer pode solicitar uma lista de todos os usuários ativos na rede ou uma lista consolidada de todos os arquivos compartilhados por todos os peers.

A busca por um arquivo específico também é realizada via broadcast. Se um peer não souber onde um arquivo está, ele envia uma pergunta para todos os demais na rede. Os peers que possuem o arquivo respondem informando que o têm e qual o seu tamanho.

Uma vez que um arquivo é localizado, a transferência é realizada através de uma conexão TCP direta entre o solicitante (cliente) e o detentor do arquivo (servidor), garantindo uma entrega confiável dos dados. O sistema é projetado para ser concorrente, utilizando threads para permitir que um peer continue a processar novas solicitações e interagir com o usuário enquanto transferências de arquivos ocorrem em paralelo.

2. Protocolo Implementado

O protocolo define como os peers trocam informações de controle e dados.

2.1. Comunicação e Portas

- As mensagens de controle (descoberta, busca, etc.) são trocadas utilizando o protocolo UDP, permitindo o uso de broadcast para consultas à rede.
- A transferência de arquivos é realizada sobre TCP, assegurando a entrega ordenada e confiável dos dados.
- Todos os programas na RCA utilizam uma porta comum para comunicação, definida como `5555` no código (`PORTA`).

2.2. Formato das Mensagens de Controle (UDP)

As mensagens de controle são representadas por uma `struct` em C, contendo os seguintes campos:

```
#define TAM_NOME_ARQ 100

typedef struct {
    uint8_t tipo_msg;                 // Tipo da mensagem
    char nome_arquivo[TAM_NOME_ARQ]; // Nome do arquivo (opcional)
    uint32_t tamanho_arquivo;        // Tamanho do arquivo
(resposta)
} mensagem_udp_t;
```

- `tipo_msg`: Um byte que identifica o tipo da mensagem.
- `nome_arquivo`: Um array de caracteres (`TAM_NOME_ARQ` definido como 100) para o nome do arquivo.
- `tamanho_arquivo`: Um inteiro de 4 bytes sem sinal, usado principalmente na mensagem `RESPOSTA_ARQUIVO` para informar o tamanho do arquivo encontrado.

2.3. Tipos de Mensagem Os seguintes tipos de mensagem de controle (campo `tipo_msg`) são definidos (conforme `rca.c` e relatório preliminar):

- `1 (LISTA_USUARIOS)`:
    - **Cliente envia (Broadcast):** Solicita a lista de IPs dos usuários ativos na RCA.

    - **Servidor responde (Unicast para o solicitante):** Envia uma mensagem do mesmo tipo para indicar sua presença. O IP do remetente é extraído pelo solicitante.
- `2 (LISTA_ARQUIVOS)`:
    - **Cliente envia (Broadcast):** Solicita a lista de arquivos de todos os usuários da rede.
    - **Servidor responde (Unicast para o solicitante):** Envia uma mensagem deste tipo para cada arquivo que possui, contendo o nome do arquivo. O IP é o do remetente.
- `3 (PROCURA_ARQUIVO)`:
    - **Cliente envia (Broadcast):** Procura por um arquivo específico na RCA, enviando o nome do arquivo.
    - **Servidor (se possuir o arquivo):** Responde com uma mensagem `RESPOSTA_ARQUIVO`.
- `4 (RESPOSTA_ARQUIVO)`:
    - **Servidor envia (Unicast para o solicitante):** Informa que possui o arquivo procurado, seu nome e seu tamanho.
- `5 (TRANSFERENCIA_ARQUIVO)`: Este tipo é usado internamente para fins de logging no servidor TCP para registrar o evento de transferência de arquivo. Não é um tipo de mensagem UDP trocado para controle.

Ok! Com a inclusão do `Makefile`, a seção "Como Executar o Programa" do relatório fica mais completa e padronizada. O `Makefile` que você forneceu automatiza a compilação com flags importantes para warnings e debugging.

Aqui está apenas a seção **"3. Como Executar o Programa"** do relatório, modificada para incluir o uso do `Makefile`:

---

3. Como Executar o Programa

3.1. Pré-requisitos:

- Certifique-se de ter o compilador GCC e a ferramenta `make` instalados no Linux.

3.2. Compilação:

- O projeto inclui um `Makefile` que automatiza o processo de compilação. Para compilar o programa, vá até o diretório raiz do projeto (onde o arquivo `rca.c` e o `Makefile` estão localizados) e execute o seguinte comando no terminal:

  ```
  make
  ```

- Este comando invocará o GCC com as flags especificadas no `Makefile` (`-Wall -Wextra -g -pthread`), que incluem todas as otimizações de aviso, informações de depuração e a vinculação da biblioteca de threads POSIX. O executável resultante será nomeado `rca`.

  Caso não utilize o `Makefile`, o comando manual para compilação (equivalente ao que o `Makefile` executa) seria:

  ```
  gcc -Wall -Wextra -g -pthread rca.c -o rca
  ```

3.3. Limpeza dos Arquivos Compilados (Opcional):

- O `Makefile` também fornece uma regra para limpar os arquivos gerados pela compilação (o executável `rca`) e o arquivo de log (`log.txt`). Para isso, execute:

  ```
  make clean
  ```

3.4. Execução:

- Após a compilação, o executável `rca` estará disponível no diretório.
- Para simular a rede P2P, execute o programa compilado (`rca`) em máquinas diferentes.
- Os peers devem estar na mesma rede local para que a comunicação via broadcast funcione corretamente.
- Comando para executar em cada terminal:

  ```
  ./rca
  ```

3.5. Interação:

- Após a inicialização, cada instância do programa apresentará um menu de cliente no terminal.
- O usuário pode escolher opções como listar usuários, listar todos os arquivos na rede, procurar por um arquivo específico ou baixar um arquivo.
- As respostas às solicitações de listagem e busca serão exibidas no terminal do cliente que fez a requisição.
- Um arquivo de log chamado `log.txt` será criado/atualizado no diretório de execução, registrando as atividades do servidor de cada peer.
-

4. Funcionalidades Implementadas (Avaliação dos Requisitos)

A seguir, uma análise de conformidade das funcionalidades implementadas no código `rca.c` em relação aos requisitos especificados na descrição do trabalho no ava.

- **Programa funciona como cliente e servidor ao mesmo tempo**: **Implementado.** O programa utiliza threads para executar as funcionalidades de servidor UDP, servidor TCP e a interface de cliente (no main thread) concorrentemente.

- **Identificar arquivos no diretório corrente e prepará-los para compartilhamento**: **Implementado.** A função `listar_arquivos` é chamada pelo servidor UDP quando necessário para obter os arquivos do diretório atual.

- **Exibir opções ao usuário (listar usuários/arquivos RCA, procurar, buscar arquivo)**: **Implementado.** O `cliente_udp` apresenta um menu com estas opções.

- **Verificar se arquivo já existe localmente antes de buscar**: **Implementado.** Ao tentar baixar um arquivo (opção 4 do cliente), o sistema verifica se o arquivo já existe. Se sim, informa o usuário e pergunta se deseja sobrescrever antes de prosseguir com o download.

- **Perguntar aos demais programas se não souber onde o arquivo está (broadcast)**: **Implementado.** A busca por arquivo (`PROCURA_ARQUIVO`) é feita via broadcast UDP.

- **Resposta do peer com tamanho do arquivo**: **Implementado.** O servidor, ao encontrar um arquivo solicitado, responde com uma mensagem `RESPOSTA_ARQUIVO` contendo o nome e o tamanho do arquivo.

- **Buscar arquivo via conexão TCP**: **Implementado.** O download de arquivos é realizado através de uma nova conexão TCP com o peer que possui o arquivo.

- **Funcionalidades do Servidor:**
    - **Log de solicitações (IP, tipo, data/hora)**: **Implementado.** A função `registrar_log` grava essas informações. É chamada tanto para requisições UDP recebidas quanto para transferências TCP. O log também inclui nome do arquivo e tempo de transferência quando aplicável.

    - **Responder consulta sobre seu IP ("usuário" de rede)**: **Implementado.** Em resposta a `LISTA_USUARIOS`, o servidor envia uma mensagem, e o cliente extrai o IP do remetente.

    - **Fornecer a lista de arquivos contidos em seu diretório**: **Implementado.** Em resposta a `LISTA_ARQUIVOS`, o servidor envia mensagens para cada um de seus arquivos.

    - **Responder positivamente caso possua determinado arquivo**: **Implementado.** Através da mensagem `RESPOSTA_ARQUIVO`.

    - **Enviar um determinado arquivo solicitado, cronometrar tempo e salvar no log**: **Implementado.** O `servidor_tcp_thread_envio` envia o arquivo, e o tempo de transferência é registrado no log.

- **Funcionalidades do Cliente:**
    - **Solicitar (via BROADCAST) a lista de usuários da RCA (retornar IP)**: **Implementado.**
    - **Solicitar (via BROADCAST) a lista de arquivos de todos os programas**: **Implementado.**
    - **Procurar (via BROADCAST) um arquivo específico na RCA**: **Implementado.**
    - **Baixar um arquivo de um determinado servidor: Implementado.**
    -
- **Todos os programas devem utilizar a mesma porta**: **Implementado.** Definido como `PORTA 5555`.

- **Formato da lista de arquivos na linha de comando (`<IP> <nome do arquivo>`)**: **Implementado.** O cliente exibe as informações nesse formato ao listar arquivos da rede.

- **Cliente e servidor devem ser o mesmo programa**: **Implementado.**

- **Protocolo de troca de mensagens de controle usando UDP e cabeçalhos de tamanho fixo**: **Implementado.** Conforme descrito na seção "Protocolo Implementado".

- **Existência de funcionalidades distintas (interface, consulta, processamento, transferência) e concorrência**: **Implementado.** Threads são usadas para interface com o usuário (thread principal para o cliente), processamento de mensagens UDP (`servidor_udp`), escuta de conexões TCP (`servidor_tcp`), e transferências de arquivos individuais (`servidor_tcp_thread_envio`). Isso permite atender consultas enquanto se transfere arquivos e lidar com múltiplas transferências em paralelo.

- **Uso de `setsockopt` para `SO_BROADCAST` e endereço de broadcast**: **Implementado.** O cliente UDP configura `SO_BROADCAST` e envia para `INADDR_BROADCAST`.

- **Troca de arquivos via conexão TCP e teste com diferentes tipos de arquivos**: **Implementado.** A transferência é binária (`"rb"` e `"wb"`), o que suporta diversos tipos de arquivo. O teste efetivo com diferentes tipos (imagens, PDF, etc.) é responsabilidade do usuário durante a demonstração.

- **Tratamento de temporizações (e.g., `SO_RCVTIMEO`)**: **Implementado.** O cliente UDP utiliza `SO_RCVTIMEO` no socket de recebimento de respostas para evitar bloqueio indefinido.

- **Compilação no Linux utilizando gcc**: **Implementado.** O código utiliza bibliotecas C padrão e POSIX threads, compatível com GCC no Linux.

- **Comentar todas as linhas do código**: **Implementado (com ressalvas).**

5. Funcionalidades Extras

Nesta implementação, o foco principal foi atender a todos os requisitos obrigatórios especificados no enunciado do trabalho. Não foram implementadas funcionalidades extras.

6. Conclusão

O sistema RCA foi implementado com sucesso, atendendo à grande maioria dos requisitos propostos. A arquitetura peer-to-peer com funcionalidades de cliente e servidor concorrentes foi feita utilizando threads, e os protocolos UDP para controle e TCP para transferência de dados também foram implementados. O programa tem a capacidade de descoberta na rede, busca e download de arquivos, além de registrar as operações importantes.