Trabalho de Redes

Flávio Lúcio Corrêa Júnior

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

---

1 Introdução:

O trabalho consiste em simular um ambiente de chamada escolar online usando os conhecimentos sobre programação em sockets usando a interface POSIX vistos em sala. Para isso escreveu-se três programas: servidor, cliente-aluno, cliente-professor, representando respectivamente um gerenciador/sistema, um aluno e um professor.

2 Diferenças Semânticas:

A principal diferença semântica dos programas cliente e servidor encontra-se no fato de que o primeiro implementa uma conexão ativa via sockets, ou seja, ele tenta conectar ao servidor via `connect()`, enquanto o segundo usa uma conexão passiva com a chamada `bind()` + `listen()`. Dessa forma, ao abrir uma conexão cliente-servido, ambos iniciam uma troca de mensagens onde o fluxo e lógica do programa é decidido nos primeiros 8 bytes (senha) enviados por um dos possíveis clientes (professor ou aluno). Assim, o servidor é capaz de decidir qual função `handler` irá tratar este cliente.

No caso do cliente aluno, o final da mensagem sempre está pré-determinado pois estas têm tamanho fixo, o qual permite uma comunicação direta e sem a necessidade de checagens por caracteres especias. Por outro lado, a comunicação com o cliente professor, ao enviar a lista com a matrícula dos alunos presentes, não possui tal característica. Isso pois os dados envolvidos na troca de mensagens têm tamanho variado, fazendo-se a necessário checar pelos caracteres terminadores, neste caso o `\n` e o `\0`. Para tal, o seguinte algoritmo foi implementado:

```
Enquanto estivermos recebendo mensagens faça:
    Procure pelo caracter `\0` na mensagem
    Se este não está presente:
        Inclua-o no final do buffer e imprima na saída padrão
    Caso contrário:
        Imprima o buffer na saída padrão e pare o loop
```

3 Estratégias Para Multiplas Conexões:

Implementou-se um `poll` de `threads` usando a biblioteca `pthread` onde, para cada nova conexão, uma thread é gerada para tratá-la. Dessa forma, a main thread sempre ficará aceitando conexões.

4 Tratamento de Erros:

O tratamento de erros de conexão foi feito de forma simples em ambos os tipos de programa. No cliente, caso ocorra um fechamento prematuro da conexão, o programa chama a função `exit()` e encerra o

processo. No servidor, caso ocorra algum erro inesperado, a thread que estava tratando tal cliente retorna e abre um espaço para uma nova conexão no poll de threads. Além disso, caso ocorram erros relacionados ao sistema operacional com as chamadas de sistemas usadas para iniciar o servidor, apenas encerramos o processo do mesmo.