Bacharelado em Ciência da Computação Redes de Computadores 1º Semestre de 2024

André Luiz Cecato Justus[1], Beatriz Nahas[2], Marjorie Aparecida Cortez Daenecke[3], Rafael Correia Alves[4]

[1]Instituto Federal do Paraná - Câmpus Pinhais (IFPR)
CEP – 83330-200 – Pinhais – PR – Brazil

{andre.justus800@gmail.com,nahas204@gmail.com,marjoriedaenecke@gmail.com
,correiarafael2021@gmail.com

Abstract. *This article describes the development of a client-server communication system using sockets in the C programming language. The goal is to synchronize the date and time of a computer that suffers from a CMOS battery failure, preventing the update of date.*

Resumo. *Este artigo descreve o desenvolvimento de um sistema de comunicação cliente-servidor utilizando sockets na linguagem de programação C. O objetivo é sincronizar a data e hora do computador, que sofre com a falha da bateria CMOS, impedindo a atualização automática.*

1. Introdução

A sincronização de data e hora é um requisito fundamental para o funcionamento adequado de sistemas computacionais. A precisão da data e hora afeta diretamente a autenticidade de operações como assinaturas digitais e controle de versões. Alice enfrenta um problema recorrente com seu computador, onde a data e hora são redefinidas toda vez que o sistema é ligado, devido à falha da bateria da CMOS. Sem a possibilidade de substituir a bateria ou o hardware, propomos uma solução de comunicação cliente-servidor para corrigir a data e hora do computador de Alice automaticamente, aproveitando um outro computador na rede que possui a data e hora corretas.

2. Solução Proposta

A solução consiste na implementação de uma aplicação cliente-servidor utilizando o protocolo TCP. O computador de Alice (cliente) conecta-se a um servidor em outro computador da rede, recebe a data e hora corretas e as exibe para ajuste manual ou automatizado.

2.1. Estrutura da Comunicação Cliente-Servidor

A solução foi desenvolvida em linguagem C, utilizando as bibliotecas padrão para manipulação de sockets. A seguir, cada parte do código é descrita em detalhes.

2.1.1. Implementação do Cliente

O código do cliente (tcp_cliente.c) é responsável por estabelecer a conexão com o servidor, receber a data e a hora atuais e exibi-las.

Código do Cliente

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>

#define BUFSIZE 1024
```

As bibliotecas “stdio.h” e “stdlib.h” fornecem funções para entrada/saída e alocação de memória. string.h contém funções para manipulação de strings, enquanto “arpa/inet.h” e “unistd.h” são usadas para manipulação de sockets e operações de rede.

A macro “BUFSIZE” define o tamanho do buffer usado para armazenar dados recebidos do servidor.

```
void DieWithSystemMessage(const char *msg) {
   perror(msg);
   exit(1);
}

void DieWithUserMessage(const char *msg, const char *detail) {
   fprintf(stderr, "%s: %s\n", msg, detail);
   exit(1);
}
```

Essas funções tratam erros do sistema e do usuário. “DieWithSystemMessage” usa “perror” para exibir mensagens de erro relacionadas ao sistema, enquanto “DieWithUserMessage” exibe erros definidos pelo usuário com um detalhamento adicional.

```
int main(int argc, char *argv[]) {
  if (argc < 2 || argc > 3)
    DieWithUserMessage("Parameter(s)", "<Server Address> [<Server Port>]");
```

O “main” verifica se o número de argumentos fornecidos está correto. O cliente requer o endereço IP do servidor e, opcionalmente, a porta TCP. Caso contrário, uma mensagem de erro é exibida.

```
  char *serverIP = argv[1];
  in_port_t serverPort = (argc == 3) ? atoi(argv[2]) : 7;
```

O endereço IP do servidor é armazenado em “serverIP”. A porta do servidor é lida do segundo argumento, se fornecida, ou o cliente usa a porta padrão 7, tradicionalmente associada ao serviço de eco.

```
  int sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
  if (sock < 0)
    DieWithSystemMessage("socket() failed");
```

A função socket cria um novo socket para a comunicação. “AF_INET” indica que o socket usa endereços IPv4, “SOCK_STREAM” especifica um socket de fluxo (TCP), e “IPPROTO_TCP” define o protocolo. Se a criação falhar, o programa exibe uma mensagem de erro e termina.

```
  struct sockaddr_in serverAddress;
  memset(&serverAddress, 0, sizeof(serverAddress));
  serverAddress.sin_family = AF_INET;
```

A estrutura “sockaddr_in” armazena o endereço do servidor. A função “memset” zera essa estrutura, e “sin_family” é configurado para “AF_INET”, especificando que o endereço é IPv4.

```
  int returnVal = inet_pton(AF_INET, serverIP, &serverAddress.sin_addr.s_addr);
  if (returnVal == 0)
    DieWithUserMessage("inet_pton() failed", "Invalid address string");
  else if (returnVal < 0)
```

```
    DieWithSystemMessage("inet_pton() failed");
```

“inet_pton” converte o endereço IP do servidor de string para o formato binário usado pela rede. Se o endereço for inválido, uma mensagem de erro é exibida.

```
  serverAddress.sin_port = htons(serverPort);
```

A porta do servidor é configurada usando “htons”, que converte o número da porta para o formato de rede (“big-endian”).

```
  if (connect(sock, (struct sockaddr *)&serverAddress, sizeof(serverAddress)) < 0)
    DieWithSystemMessage("connect() failed");
```

“connect” tenta estabelecer uma conexão com o servidor. Se falhar, o programa termina com uma mensagem de erro.

```
  char buffer[BUFSIZE];
  ssize_t numBytes = recv(sock, buffer, BUFSIZE - 1, 0);
  if (numBytes < 0)
    DieWithSystemMessage("recv() failed");
  else if (numBytes == 0)
    DieWithUserMessage("recv()", "connection closed prematurely");
```

“recv” recebe a data e hora do servidor. O número de bytes recebidos é verificado; se for zero, significa que a conexão foi encerrada prematuramente.

```
  buffer[numBytes] = '\0';
  printf("Recebemos a data e a hora do servidor: %s\n", buffer);

  close(sock);
  exit(0);
}
```

O buffer é finalizado com “\0” para garantir que seja uma string válida, e a data e hora são exibidas. O socket é fechado e o programa termina.

2.1.2. Implementação do Servidor

O código do servidor (tcp_server.c) é responsável por escutar conexões de clientes, enviar a data e hora atuais e encerrar a conexão.

Código do Servidor

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <time.h>

#define BUFSIZE 1024
#define MAXPENDING 5
```

Além das bibliotecas vistas no cliente, o servidor inclui “time.h” para obter a data e hora. “MAXPENDING” define o número máximo de conexões pendentes.

```
void DieWithUserMessage(const char *msg, const char *detail) {
    perror(msg);
    fprintf(stderr, "%s\n", detail);
    exit(1);
}

void HandleTCPClient(int clientSocket) {
    char buffer[BUFSIZE];
    time_t currentTime = time(NULL);
    struct tm *localTime = localtime(&currentTime);
    strftime(buffer, BUFSIZE, "%Y-%m-%d %H:%M:%S", localTime);
```

“HandleTCPClient” obtém a data e hora atual com “time” e “localtime”, e a formata com “strftime”. O resultado é armazenado em “buffer”.

```
    ssize_t numBytesSent = send(clientSocket, buffer, strlen(buffer), 0);

    if (numBytesSent < 0) {
        DieWithUserMessage("send() failed", "Unable to send data");
    }

    close(clientSocket);
}
```

A data e hora são enviadas ao cliente usando send. O socket do cliente é fechado após o envio dos dados.

```
int main(int argc, char *argv[]) {
    if (argc != 2) {
        DieWithUserMessage("Parameter(s)", "<Server Port>");
    }

    in_port_t serverPort = atoi(argv[1]);
```

“main” verifica se a porta foi fornecida como argumento e a armazena em “serverPort”.

```
    int serverSocket;
    if ((serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
        DieWithUserMessage("socket() failed", "Unable to create socket");
    }
```

Um socket TCP é criado para o servidor.

```
    struct sockaddr_in serverAddress;
    memset(&serverAddress, 0, sizeof(serverAddress));
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
serverAddress.sin_port = htons(serverPort);
```

O servidor é configurado para aceitar conexões de qualquer endereço IP (INADDR_ANY) na porta especificada.

```
 if (bind(serverSocket, (struct sockaddr*)&serverAddress, sizeof(serverAddress)) < 0)
{
    DieWithUserMessage("bind() failed", "Unable to bind to port");
}
```

“bind” associa o socket à porta configurada. Se falhar, o programa exibe uma mensagem de erro.

```
if (listen(serverSocket, MAXPENDING) < 0) {
    DieWithUserMessage("listen() failed", "Unable to listen for connections");
}
```

“listen” faz o servidor aguardar conexões de clientes, com até “MAXPENDING” conexões simultâneas pendentes.

```
for (;;) {
    struct sockaddr_in clientAddress;
    socklen_t clientAddressLen = sizeof(clientAddress);
        int clientSocket = accept(serverSocket, (struct sockaddr*)&clientAddress,
&clientAddressLen);
    if (clientSocket < 0) {
      DieWithUserMessage("accept() failed", "Unable to accept client connection");
    }
```

O “loop” for infinito mantém o servidor em execução, aceitando conexões de clientes com “accept”.

```
    char clientName[INET_ADDRSTRLEN];
        if (inet_ntop(AF_INET, &clientAddress.sin_addr.s_addr, clientName,
sizeof(clientName)) != NULL) {
```

```
            printf("Handling client %s/%d\n", clientName, ntohs(clientAddress.sin_port));
        } else {
            puts("Unable to get client address");
        }

        HandleTCPClient(clientSocket);
    }

    close(serverSocket);
    return 0;
}
```

Após aceitar uma conexão, o servidor obtém o endereço do cliente com “inet_ntop” e o exibe. “HandleTCPClient” é então chamado para processar a requisição. O socket do servidor é fechado ao final do programa.

3. Conclusão

A solução desenvolvida permite que Alice sincronize a data e a hora do seu computador com outro na rede, superando as limitações impostas pela falha da bateria CMOS. A comunicação cliente-servidor via TCP, implementada em C, demonstrou ser eficaz para garantir a precisão temporal em sistemas computacionais restritos, como o de Alice.

4. Referências

- Slides da Aula 13 de Redes de Computadores, feitos pelo Professor Guilherme Werneck de Oliveira,