Interfaces e Periféricos
Redes como interfaces

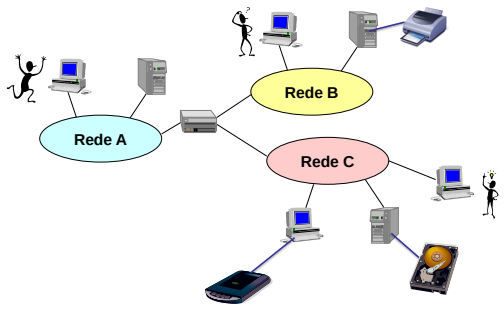
Prof. João Paulo A. Almeida
(jpalmeida@inf.ufes.br)

2007/02 - INF02788

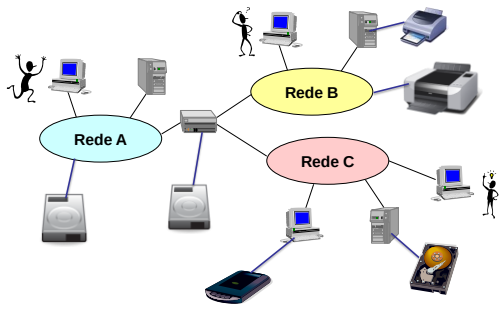
Histórico

- Surgimento e popularização das redes criou a necessidade e oportunidade de usá-las como mecanismo de interação entre um computador e "periféricos"
- Ao mesmo tempo miniaturização e barateamento de dispositivos de rede permitiram que "periféricos" se tornassem autônomos e conectados diretamente na rede
- Mainframe com seu terminais...
  - Terminais ficaram mais poderosos e com mais periféricos que poderiam ser compartilhados

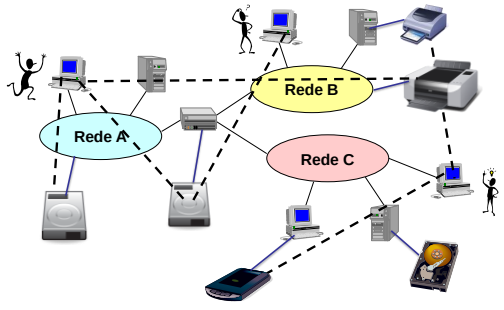
Evolução das redes



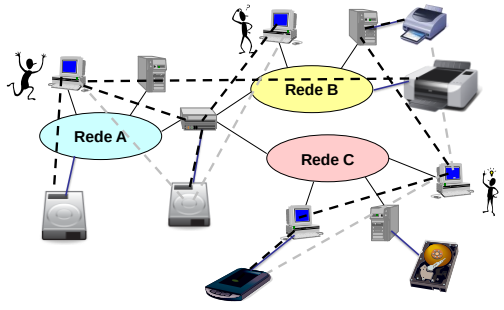
Evolução das redes



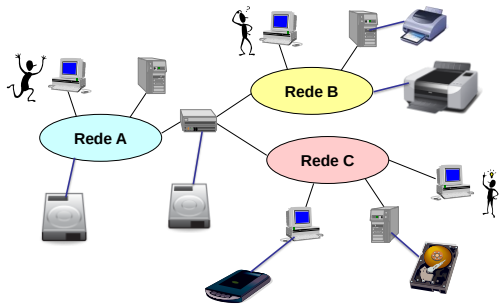
Abstração da infraestrutura de rede como interface



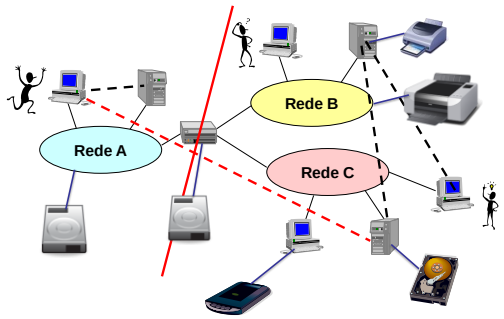
Abstração da infraestrutura de rede como interface



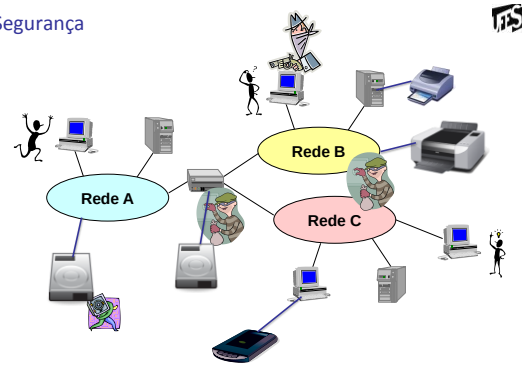
Latência (atrasos na comunicação)
Comportamento imprevisível da rede



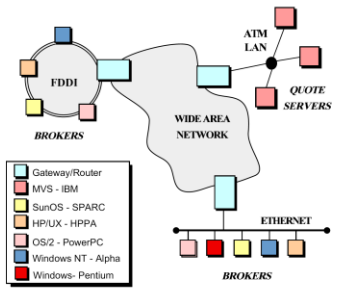
Falhas e Particionamento de rede



Segurança



Heterogeneidade



Fonte: http://www.cs.wustl.edu/~schmidt/PDF/MT-Orbix4.pdf

Programação de sistemas distribuídos

- Como programar a interação entre uma unidade de processamento e os periféricos (e/ou outras unidades de processamento com seus periféricos?
- Modelos de programação para sistemas distributídos
  - Sockets em TCP/IP

Sockets

- Interface de software para comunicação de um programa de usuário (parte de aplicação) com a pilha de protocolos TCP/IP
- Biblioteca de chamadas / API
- Socket é uma estrutura de dados
- Cliente e servidor se comunicam através de um par de sockets
  - Endereço IP (32-bits na versão 4) + número da porta (16-bits = 64K portas)

http://pages.cpsc.ucalgary.ca/~ijirasek/courses/cpsc441/slides/sockets.ppt#3

Sockets: usando serviços no nível de transporte

- Protocolos no nível de aplicação (ex., HTTP, DNS, SMTP) usam os serviços da camada de transporte
- A camada de transporte provê:
- Comunicação entre processos
  - Multiplexação/demultiplexação baseado no conceito de portas
- Entrega confiável de dados (em ordem): TCP
  - connection setup (handshake)
  - congestion control
  - flow control
  - Stream (SOCK_STREAM)
- Não confiável e sem ordem garantida de datagramas: UDP
  - Para se usar "best-effort" IP
  - Datagram (SOCK_DGRAM)

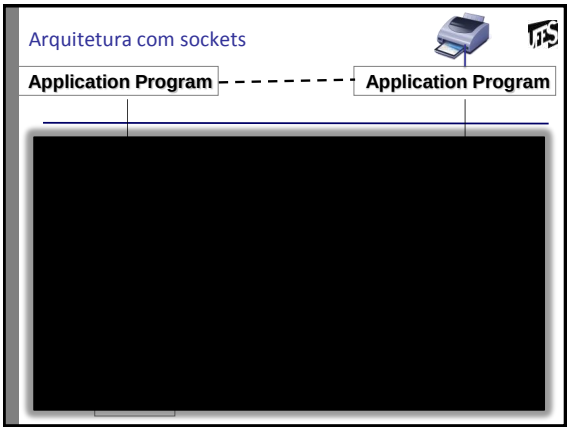
http://lrcwww.epfl.ch/sc250_2004/lecture_notes/sc250_04_7.pdf

Sockets

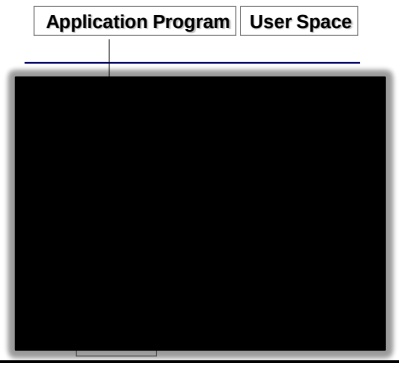
- Uma Application Programming Interface (API) provê funções, tipos de dados, estruturas de dados e constantes
  - Flexível, simples de usar, padronizado (portabilidade)
- A API BSD sockets provê uma abstração similar a um sistema de arquivos para sockets (open, close, read, write)
  - BSD sockets (originários do BSD 4.1 Unix em 1981) são a forma mais popular (API mais popular) disponível em: FreeBSD, Linux, Windows, Mac OS X, ...
- Sockets também podem ser usados para comunicação entre processos no UNIX

http://lrcwww.epfl.ch/sc250_2004/lecture_notes/sc250_04_7.pdf

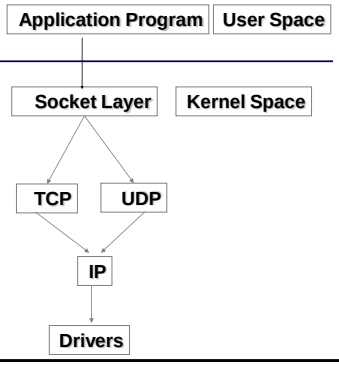
Arquitetura com sockets



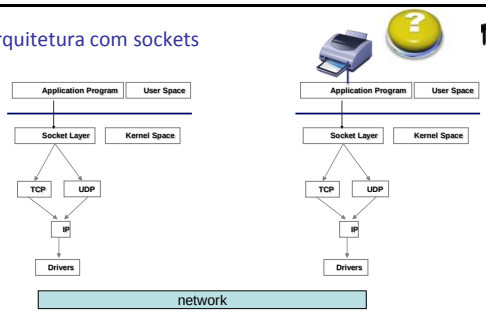
Arquitetura com sockets



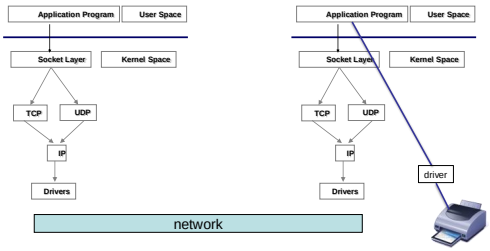
Arquitetura com sockets



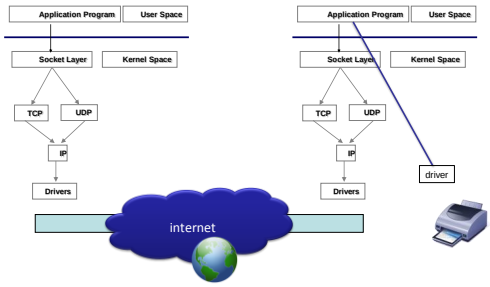
Arquitetura com sockets



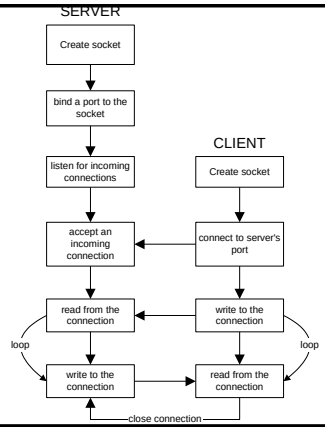
Arquitetura com sockets



Arquitetura com sockets



Sockets



Cliente/Servidor com sockets

- Cliente:
  - Inicia o contato com o servidor
    - (tem que saber o endereço IP ou hostname, e a porta)
  - Normalmente requisita execução de um serviço
- Servidor:
  - Espera passivamente (em uma certa porta)
  - Normalmente roda constantemente à espera de clientes

Servidor TCP

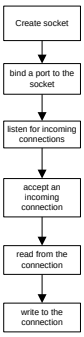
- sock_init() Create the socket
- bind() Register port with the system
- listen() Establish client connection
- accept() Accept client connection
- read/write read/write data
- close() shutdown

Cliente TCP

- sock_init () Create socket
- connect() Set up connection
- write/read write/read data
- close() Shutdown

Sockets

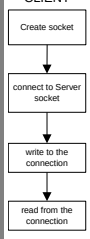
SERVER




```
int socket(int domain, int type, int protocol)
sockfd = socket(PF_INET, SOCK_STREAM, 0);

int bind(int sockfd, struct sockaddr *server_addr, socklen_t length)
bind(sockfd, &server, sizeof(server));

int listen( int sockfd, int num_queued_requests)
listen( sockfd, 5);

int accept(int sockfd, struct sockaddr *incoming_address, socklen_t length)
newfd = accept(sockfd, &client, sizeof(client)); /* BLOCKS */

int read(int sockfd, void * buffer, size_t buffer_size)
read(newfd, buffer, sizeof(buffer));

int write(int sockfd, void * buffer, size_t buffer_size)
write(newfd, buffer, sizeof(buffer));
```

Sockets

CLIENT




```
int socket(int domain, int type, int protocol)
sockfd = socket(PF_INET, SOCK_STREAM, 0);

int connect(int sockfd, struct sockaddr *server_address, socklen_t length)
connect(sockfd, &server, sizeof(server));

int write(int sockfd, void * buffer, size_t buffer_size)
write(sockfd, buffer, sizeof(buffer));

int read(int sockfd, void * buffer, size_t buffer_size)
read(sockfd, buffer, sizeof(buffer));
```

Sockets: Java (Cliente TCP)

```
import java.io.*;
import java.net.*;

class TCPClient {

   public static void main(String argv[]) throws Exception
   {
      String sentence;
      String modifiedSentence;

      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      Socket clientSocket = new Socket("hostname", 6789);

      DataOutputStream outToServer =
        new DataOutputStream(clientSocket.getOutputStream());
```

Create input stream

Create client socket, connect to server

Create output stream attached to socket

Sockets: Java (Cliente TCP) 2

```
      BufferedReader inFromServer =
            new BufferedReader(new
        InputStreamReader(clientSocket.getInputStream()));

      sentence = inFromUser.readLine();        aaaa

      outToServer.writeBytes(sentence + '\n');

      modifiedSentence = inFromServer.readLine();
                                                AAAA
      System.out.println("FROM SERVER: " +
                          modifiedSentence);

      clientSocket.close();

   }
}
```

Create input stream attached to socket

Send line to server

Read line from server

Sockets: Java (Servidor TCP)

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception
    {
      String clientSentence;
      String capitalizedSentence;

      ServerSocket welcomeSocket = new ServerSocket(6789);

      while(true) {
        Socket connectionSocket = welcomeSocket.accept();

        BufferedReader inFromClient =
          new BufferedReader(new
            InputStreamReader(
              connectionSocket.getInputStream()));
```

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

http://lrcwww.epfl.ch/sc250_2004/lecture_notes/sc250_04_7.pdf

Sockets: Java (Servidor TCP)

```
        DataOutputStream  outToClient =
          new DataOutputStream(
                connectionSocket.getOutputStream());

        clientSentence = inFromClient.readLine();        aaaa

        capitalizedSentence = clientSentence.toUpperCase() + '\n';
                                                          AAAA
        outToClient.writeBytes(capitalizedSentence);
      }
    }
}
```

End of while loop, loop back and wait for another client connection

Create output stream, attached to socket

Read in line from socket

Write out line to socket

http://lrcwww.epfl.ch/sc250_2004/lecture_notes/sc250_04_7.pdf

Servidor single-threaded x multi-threaded

•Vários clientes podem fazer pedidos ao mesmo tempo para o servidor. O servidor então pode:

- a) atender um cliente por vez; outros clientes tem que esperar (servidor não concorrente, ou single-threaded)
- b) atender vários clientes simultaneamente (servidor concorrente ou multi-threaded)
  - De qualquer forma, muitos dispositivos requerem acesso dedicado ou pelo menos "serializado" (ou seja, não permitem acesso paralelo).
  - A solução em muitos casos é a criação de *spool*s, filas de trabalhos a serem efetuados

Non-concurrent server

```
sock_init()     Create the socket

bind()          Register port with the system

while(1)        Start the loop. This loop will
{               service each request sequentially
  listen()      Establish client connection
  accept()      Accept client connection
  read/write    read/write data
}               End loop

close()         Shutdown
```

Concurrent server

```
Sock_init()     Create the socket

bind()          Register port with the system

listen()        Establish client connection

accept()        Accept client connection

fork()          Create a child process which
                will now communicate with client

read/write      read/write data in child process

exit()          exit child process
```

Sockets

- Problemas??

Sockets

- Ordem dos bytes nos tipos de dados dependem da arquitetura da máquina
- host order:
  - 12 34 56 78 (Motorola) big endian
  - 78 56 34 12 (Intel) little endian

network order: 12 34 56 78

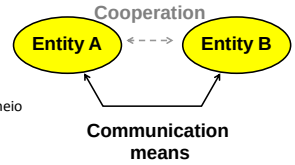
- Funções de conversão:
  - u_long htonl(u_long hostlong);
  - u_short htons(u_short hostshort);
  - u_long ntohl(u_long netlong);
  - u_short ntohs(u_short netshort);

Sockets

- Baixo nível de abstração...
  - Principalmente quando o que você queria fazer era pedir para outra máquina executar um serviço... ou simplesmente usar um procedimento remoto
  - Endereços IPs/DNS/portas
  - Detectar time-outs
  - Threading
- Como codificar os dados de uma aplicação?
  - Por exemplo, como enviar uma lista, um array, um struct, um objeto
- Tem que se criar um "protocolo" sempre...

Protocolo

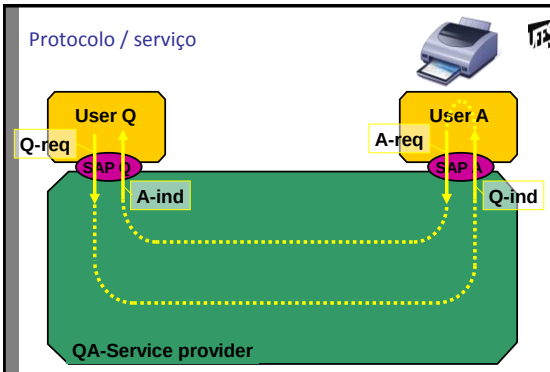
- Definição clássica
  - Mensagens e regras que determinam ações a serem tomadas de acordo com a recepção das mensagens
- Mais precisamente...
  - Entidades (de protocolo) cooperando através de um meio de comunicação



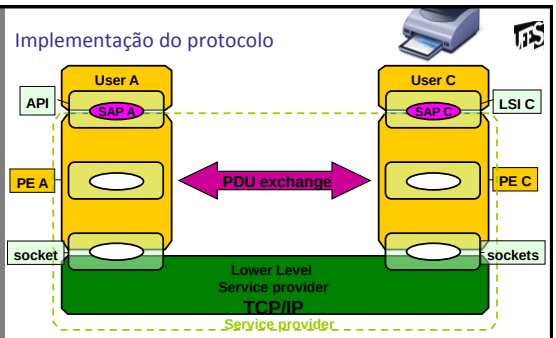
Protocolo

- Definir/especificar:
  - Formatos de mensagens
    - Tanto tipos de dados abstratos quanto a forma de se codificá-los na rede
  - Comportamento
    - Em que momento cada mensagem pode ser enviada e o que deve ser feito para cada lidar com cada mensagem

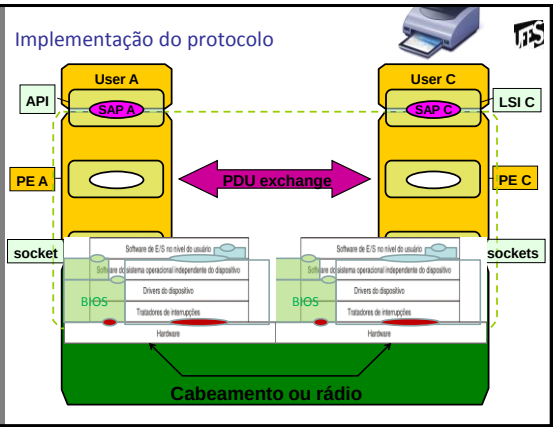
Protocolo / serviço



Implementação do protocolo



Implementação do protocolo



Exemplos: protocolos para impressão

- Line printer protocol
- http://www.ietf.org/rfc/rfc1179.txt
- IPP
- http://tools.ietf.org/html/rfc2910
- SMB/CIFS
- http://www.samba.org/cifs/docs/what-is-smb.html

Exemplos: protocolos para compartilhamento de disco

- NFS

- SMB/CIFS
- http://www.samba.org/cifs/docs/what-is-smb.html