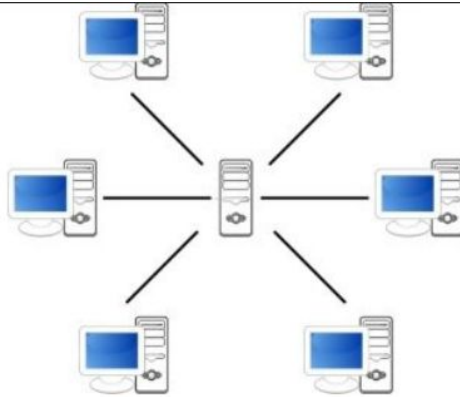
Computação Distribuída

Conceitos de Sistemas Distribuídos, Redes e Sistemas Operacionais

Slides baseados nas aulas dos professores Felipe Cunha, Cleiton Tavares, Hugo de Paula e Gleisson

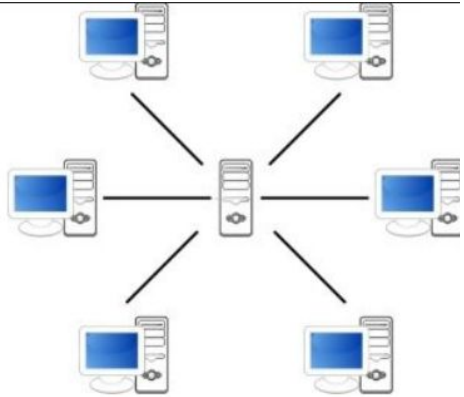
Sistema Distribuído



“ Um sistema distribuído é uma coleção de computadores autônomos conectados por uma rede e equipados com um sistema de software distribuído “

Distributed Systems: Concepts and Design. G. Coulouris , J. Dollimore , T. Kindberg . Addison Wesley, 1994. ISBN 0 201 62433 8

Sistema Distribuído



“ Um sistema distribuído é uma coleção de computadores independentes que aparenta ao usuário ser um computador único “

Distributed Operating Systems. A. S. Tanenbaum . Prentice Hall, 1995. ISBN 013219908 4

O que são sistemas distribuídos?

- São sistemas em que pelo menos um dos componentes está localizado em uma **rede de computadores** e coordenam suas ações através da **troca de mensagens**

- O **compartilhamento de recursos** é a motivação principal para construir sistemas distribuídos

Características dos Sistemas Distribuídos

- Quaisquer usuários ou aplicações podem interagir com um sistema de maneira consistente e uniforme, independentemente de onde a interação ocorra.

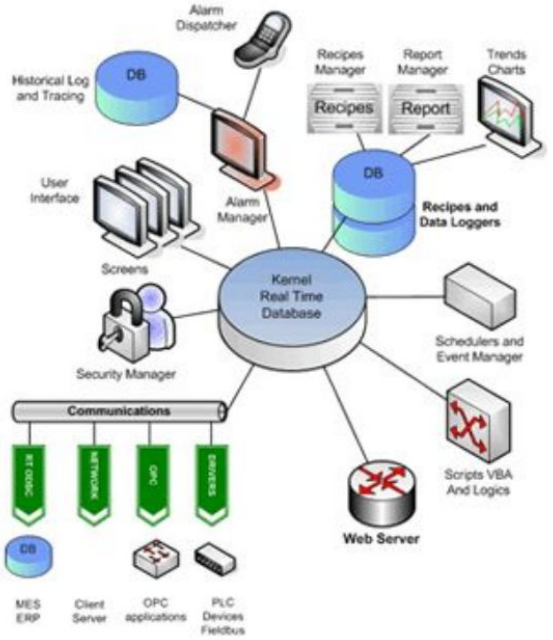
- Um sistema distribuído, em geral, está sempre disponível mesmo que uma ou outra parte esteja temporariamente avariada

- Usuários não devem perceber quais partes estão sendo substituídas ou consertadas

Características dos Sistemas Distribuídos

- Todo sistema distribuído consiste em componentes (computadores) autônomos

- Todas os usuários (pessoas ou programas) acham que estão tratando com um único sistema



Surgimento dos Sistemas Distribuídos

- Sistemas distribuídos são consequência dos eventos abaixo:

- Invenção de redes de computadores de alta velocidade (anos 70):
  - Rede local (Local Area Network LAN)
  - Rede global (Wide Area Network WAN)
- Desenvolvimento de microprocessadores potentes (anos 80).

Surgimento dos Sistemas Distribuídos

- De 1945 até 1985, computadores grandes e caros, destinados a umas poucas organizações
- Mas os computadores evoluíram em desempenho e caíram de preço
  - Inicialmente custavam US$ 10 milhões e processavam 1 instrução por segundo
  - 50 anos depois, custavam menos de US$ 1000 e executavam 1 bilhão de instruções por segundo

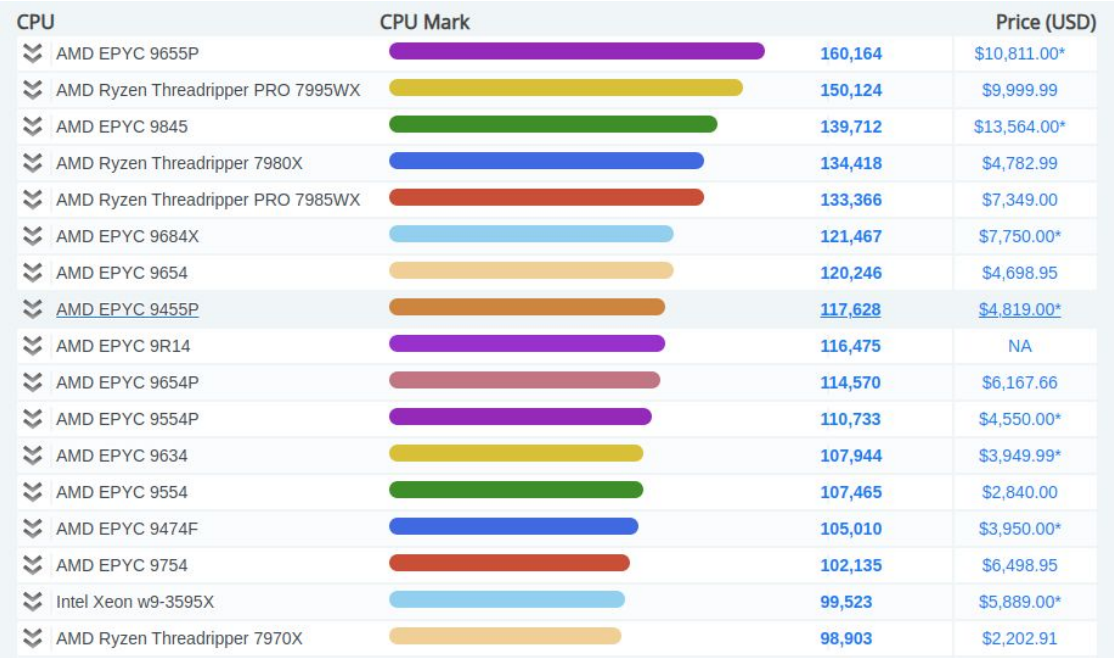
Transferência de um Hd de 5Mb em 1956

Vantagens dos Sistemas Distribuídos

- Os Sistemas Distribuídos (SD) possuem diversas vantagens sobre Sistemas Centralizados (SC) e PCs:

- Melhor relação custo/benefício:
  - Lei de Grosch - O poder de computação de um computador é proporcional ao quadrado do seu preço
  - No caso de microprocessadores, é mais barato se comprar vários processadores e montá-los em um sistema multiprocessador

Comparando Processadores



CPU	CPU Mark	Price (USD)
AMD EPYC 9655P	160,164	$10,811.00*
AMD Ryzen Threadripper PRO 7995WX	150,124	$9,999.99
AMD EPYC 9845	139,712	$13,564.00*
AMD Ryzen Threadripper 7980X	134,418	$4,782.99
AMD Ryzen Threadripper PRO 7985WX	133,366	$7,349.00
AMD EPYC 9684X	121,467	$7,750.00*
AMD EPYC 9654	120,246	$4,698.95
AMD EPYC 9455P	117,628	$4,819.00*
AMD EPYC 9R14	116,475	NA
AMD EPYC 9654P	114,570	$6,167.66
AMD EPYC 9554P	110,733	$4,550.00*
AMD EPYC 9634	107,944	$3,949.99*
AMD EPYC 9554	107,465	$2,840.00
AMD EPYC 9474F	105,010	$3,950.00*
AMD EPYC 9754	102,135	$6,498.95
Intel Xeon w9-3595X	99,523	$5,889.00*
AMD Ryzen Threadripper 7970X	98,903	$2,202.91

- http://www.cpubenchmark.net/high_end_cpus.html

Vantagens dos Sistemas Distribuídos

- Os Sistemas Distribuídos (SD) possuem diversas vantagens sobre Sistemas Centralizados (SC) e PCs:

- Capacidade de processamento além da capacidade física de um SC
  - É teoricamente impossível de se construir um computador centralizado que possa ser comparado a um sistema distribuído com uma grande quantidade de processadores
  - Diversos computadores distribuídos podem realizar partes de processamento, através da divisão de tarefas
    - Isso faz com que um SD tenha a velocidade muito superior a um SC

Vantagens dos Sistemas Distribuídos

- Os Sistemas Distribuídos (SD) possuem diversas vantagens sobre Sistemas Centralizados (SC) e PCs:

- Possuem maior confiabilidade e disponibilidade:
  - Permite uma distribuição do processamento entre diversos computadores, permitindo o melhor aproveitamento dos recursos computacionais de cada um deles
  - O sistema não cai como um todo, partes que falham podem ser substituídas sem a necessidade de parar todo o sistema
  - Permitem o crescimento incremental, possibilitando que o todo o sistema possa crescer

Desvantagens dos Sistemas Distribuídos

- **Complexidade de desenvolvimento**: problemas com consistência, verificabilidade e sistemas de troca de mensagens.
- **Rede**: a rede pode saturar ou causar outros problemas como atraso.
- **Segurança**: compartilhamento pode gerar acessos não autorizados a dados secretos.

Tendências em sistemas distribuídos

- **Rede pervasiva**: dispositivos conectados a qualquer momento em qualquer lugar.
- **Computação móvel e ubíqua**: integração de pequenos dispositivos (wearable, embarcados, redes de sensores, eletroportáteis) com sistemas distribuídos.
- **Sistemas multimídia distribuídos**: armazenamento, localização e transmissão de diversos tipos de mídia (áudio, vídeo, texto), preservando relação temporal.
- **Computação em nuvem e computação em grade**: computação massiva e paralela como serviço, provendo infraestrutura, plataforma ou aplicações distribuídas.

Exemplos

SETI@Home

- Projeto desenvolvido pela Universidade de Berkeley
- Procura sinais de vida inteligente no universo
- Captura 35Gb de dados por dia da antena de Arecibo , Porto Rico
    - Esses dados são divididos em 140.000 unidades de trabalho work units (WU)
    - Um computador doméstico demora cerca de 30 horas para analisar cada WU
    - É equivalente a 4.2 milhões de horas para processar os dados de apenas 1 dia

Exemplos

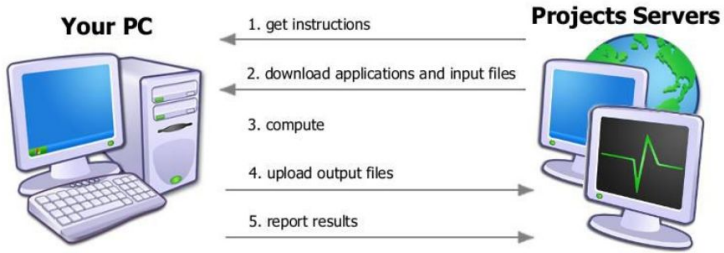
SETI@Home

- Como processar essa quantidade de dados?
  - Seria muito caro utilizar supercomputadores para essa tarefa
  - Proposta:
    - Utilizar um screen saver para processar esses dados enquanto os computadores não estão sendo utilizados
    - Usuários domésticos recebem uma uma WU, trabalham nela e devolvem o resultado para o SETI, que envia outra WU

Exemplos

BOINC

- O SETI@home deu origem ao BOINC (Berkeley Open Infrastructure for Network Computing)
- O BOINC é a base para diversos projetos científicos que necessitam de alto poder de processamento
- O usuário do BOINC pode escolher em qual projeto deseja colaborar com o poder computacional de seu computador



Exemplos

BOINC

- Até recentemente, O BOINC possuía:
  - 401.634 voluntários ativos
  - 516.840 computadores
- Média de processamento diário : 6,348 PetaFLOPS

http://boinc.Berkeley.edu

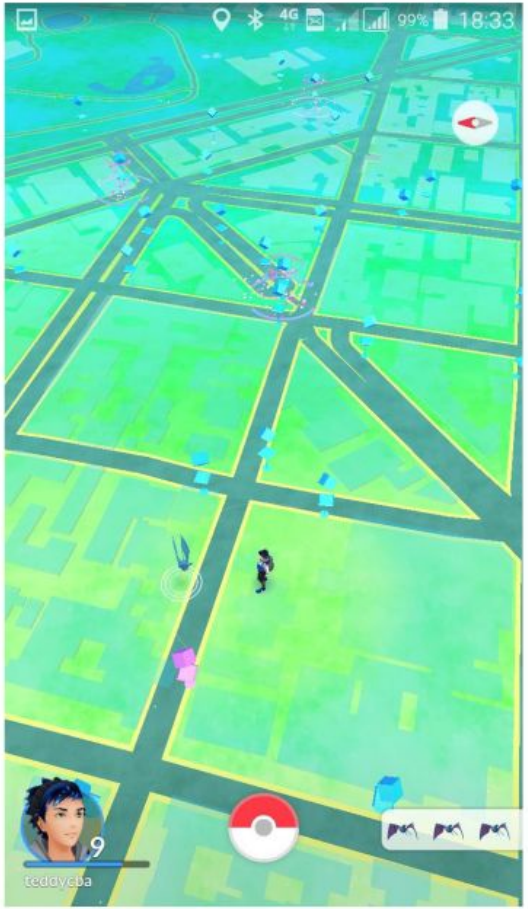
Exemplos

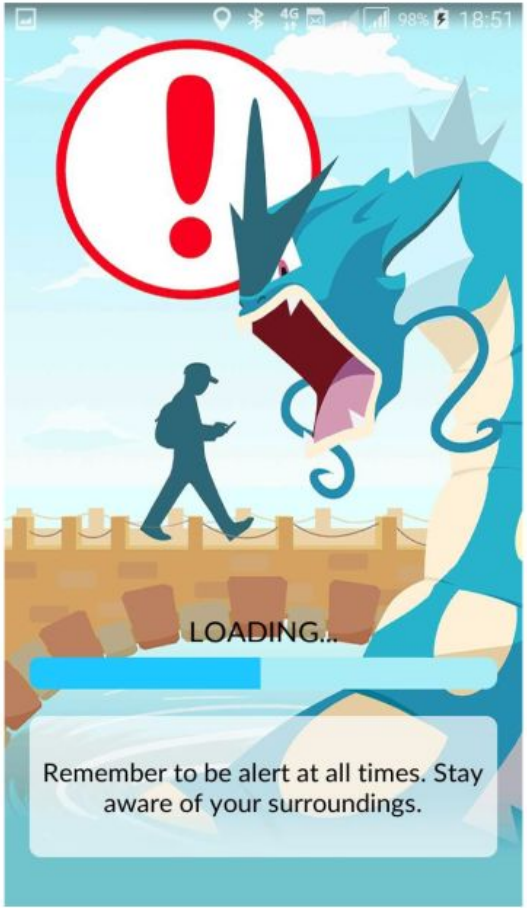
Serviços compartilhados online

- Uma empresa tem usuários em vários locais do planeta, cada um desses usuários compartilhando informações e consumindo recursos
- Os usuários só poderão interagir se estiverem utilizando uma conexão com a Internet, seja 3G/4G ou Wifi
- Cada usuário tem seu próprio repositório de dados local, mas eles são sincronizados com o servidor para que a interação possa ocorrer
- De qual empresa estamos falando?

Exemplos

- Waze?
- Whatsapp?
- Facebook?





Computação Distribuída

- É um método de processamento computacional na qual diferentes partes de um programa rodam simultaneamente em um ou mais computadores através de uma rede de computadores
  - É um tipo de processamento paralelo

Computação Distribuída

- Sistema de processamento distribuído ou paralelo:
  - É um sistema que interliga vários nós de processamento (computadores individuais), não necessariamente homogêneos de maneira que um processo de grande consumo seja executado no nó “mais disponível”, ou mesmo subdividido por vários nós
  - Envolve Paralelismo : divisão de uma tarefa em sub-tarefas coordenadas e que são executadas simultaneamente em processadores distintos

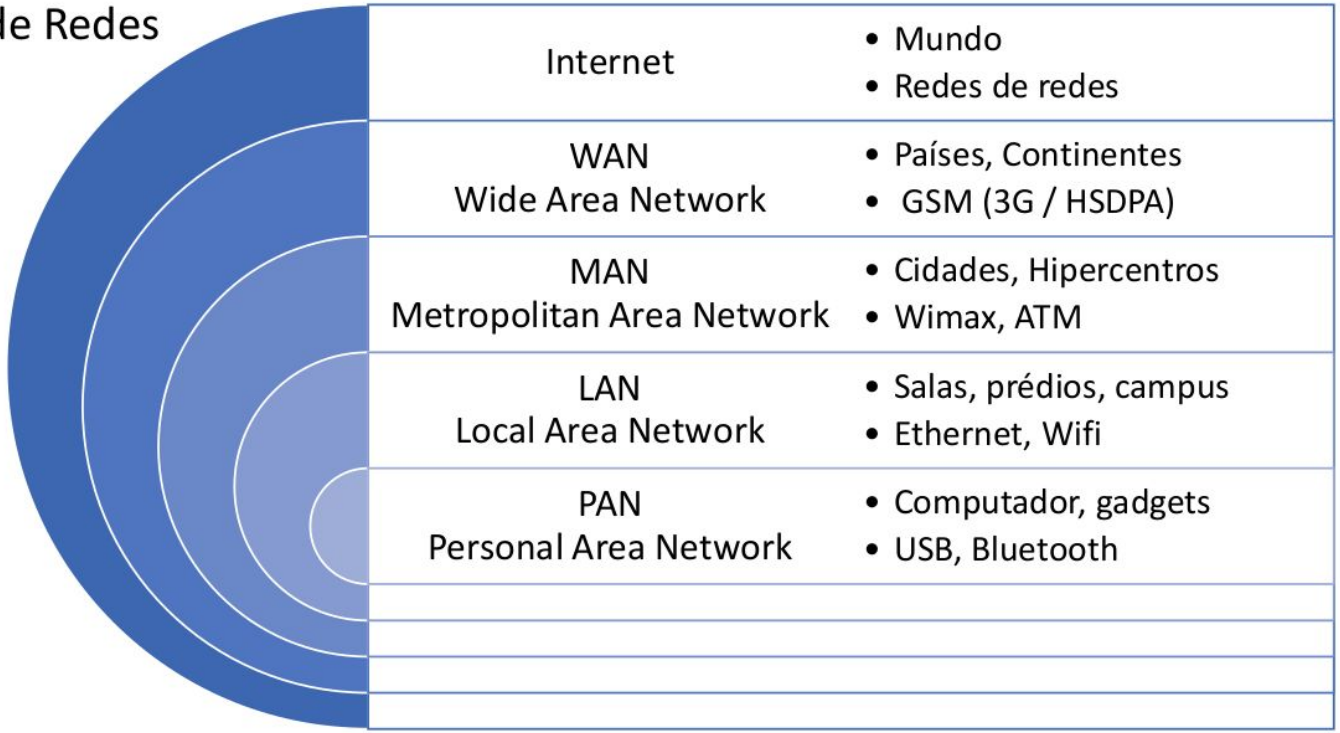
Computação Paralela x Computação Distribuída

- Computação paralela
  - Comunicação via memória compartilhada
  - Objetivo: aumentar throughput (vazão) ou reduzir tempo de serviço.
- Computação distribuída
  - Comunicação via troca de mensagens
  - Objetivo: compartilhar recursos fisicamente dispersos, aumentar a escalabilidade no atendimento a clientes.

Conceitos de Redes de Computadores

Fundamentos de Redes

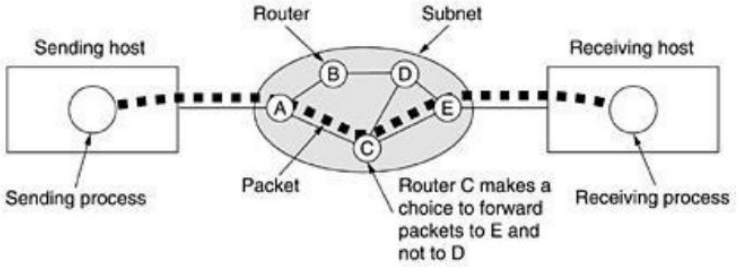
- Tipos de Redes



Fundamentos de Redes

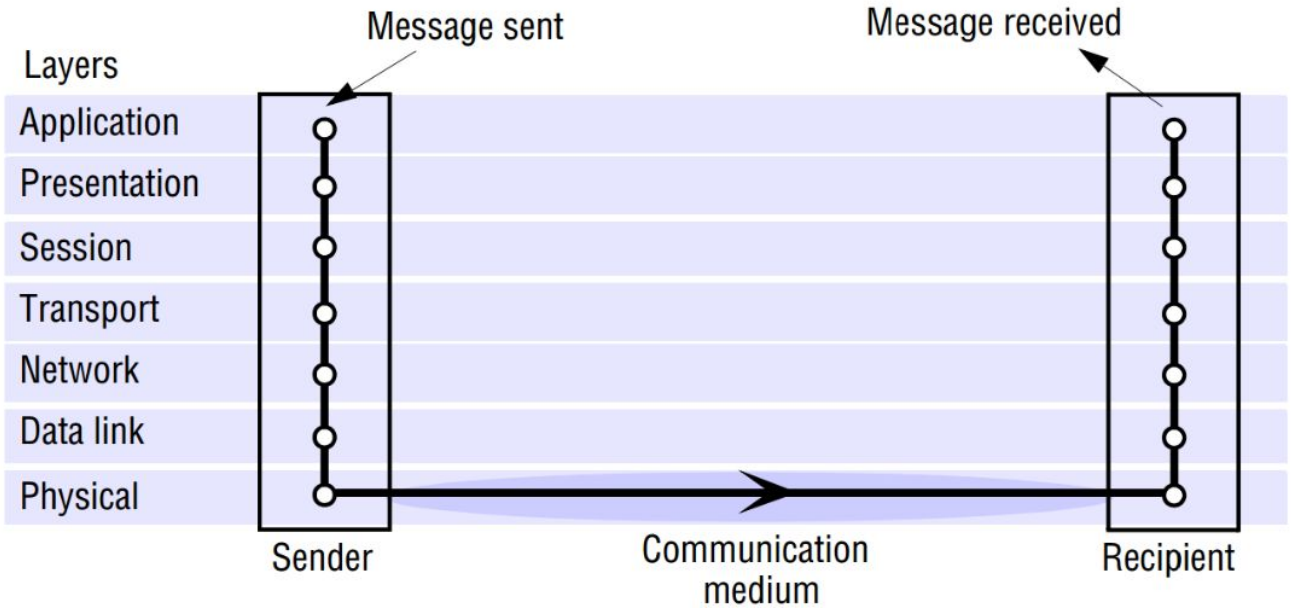
Esquemas de comutação

- Broadcast
  - Tudo é enviado para todos os nodos.
- Comutação de circuitos
  - Estabelecimento de canais dedicados de comunicação
- Comutação de pacotes
  - Sistema de armazenamento e encaminhamento de pacotes com base nas informações de origem e destino.



Fundamentos de Redes

Camadas do Modelo OSI (Interconexão de Sistemas Abertos)

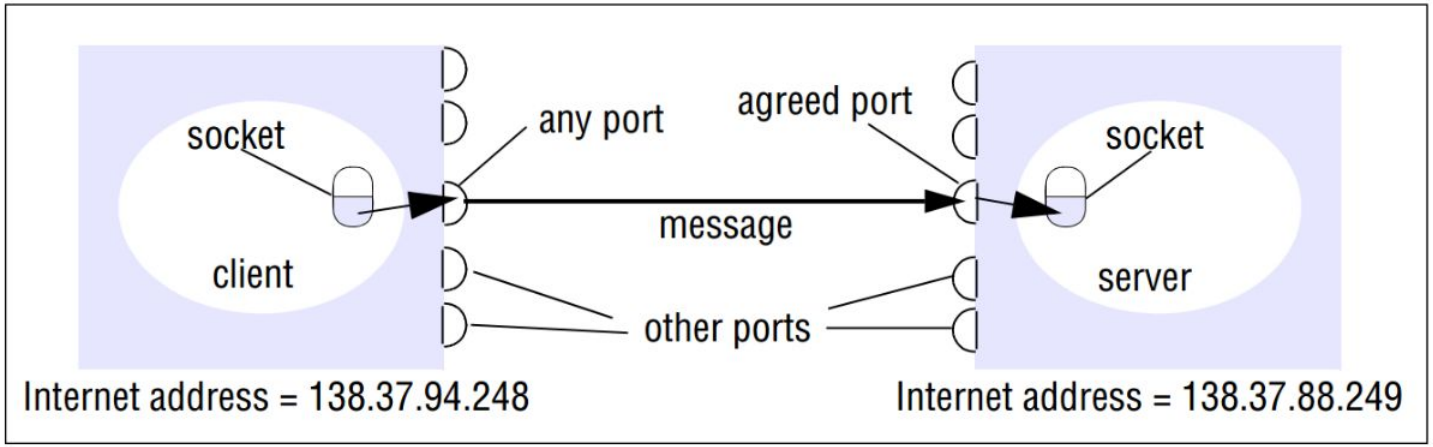


Fundamentos de Redes

#	Camada	Descrição	Protocolos
7	Aplicação	Atende aos requisitos de comunicação de aplicativos específicos, definindo uma interface para um serviço	HTTP, SMTP, SNMP, FTP, Telnet, SSH, NFS, DNS
6	Apresentação	Transmitem dados em uma representação de rede independente das usadas em cada nó. Criptografia, se exigida é feita nesta camada	Segurança TLS, SMB, AFP
5	Sessão	Realiza operações relacionadas com a confiabilidade das conexões, detecção de falhas e recuperação automática	SIP, SSH, RPC, NetBIOS, ASP
4	Transporte	Nível mais baixo de manipulação das mensagens que são endereçadas para portas de comunicação	TCP, UDP, SPX
3	Rede	Transfere pacotes com base no endereçamento dos nodos, o que pode envolver o roteamento entre redes	IP, ICMP, IGMP, X.25, ARP, RARP, BGP, OSPF, RIP, IPX
2	Enlace de dados	Transmite pacotes entre nodos fisicamente conectados	Ethernet, Token Ring, PPP, HDLC, Frame Relay, ISDN, ATM, Wi-Fi
1	Física	Transmite sequências de dados binários envolvendo hardware e seus circuitos	Elétrico, radio, laser

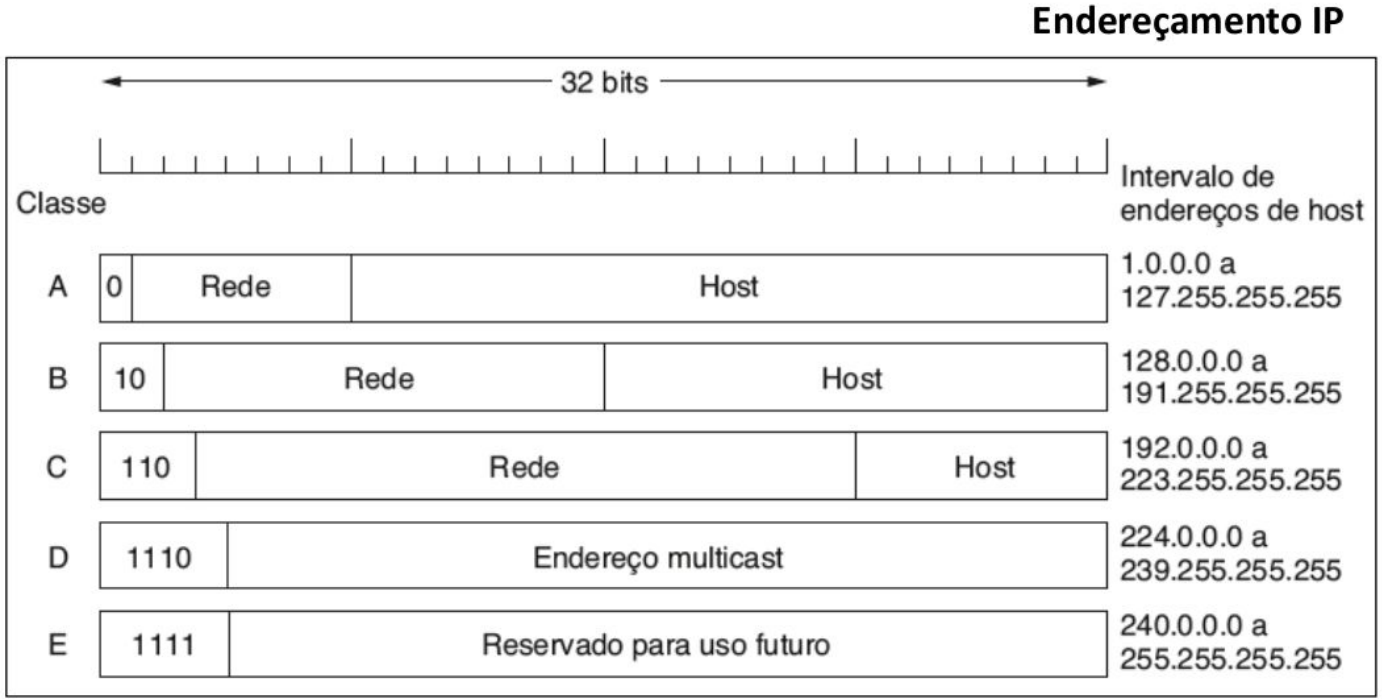
Fundamentos de Redes

Portas



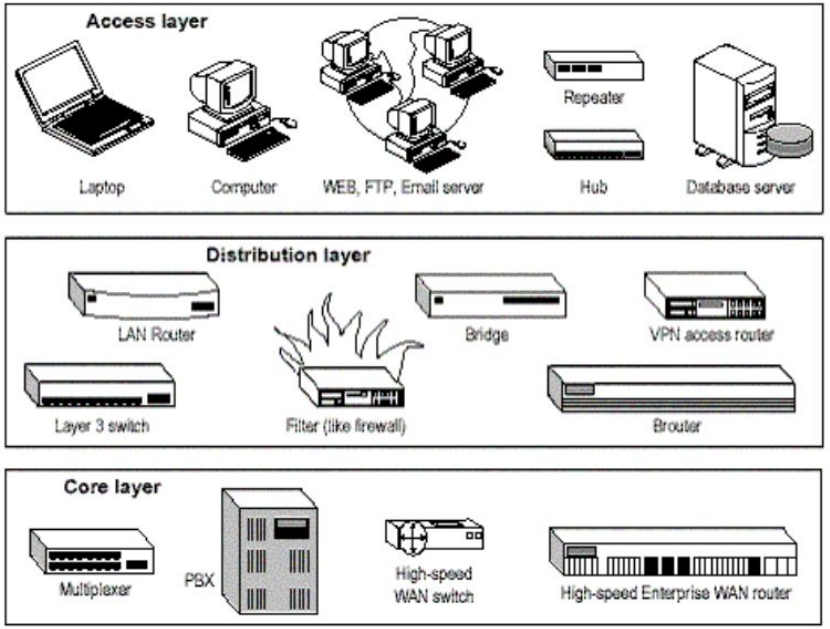
Fundamentos de Redes

Endereçamento



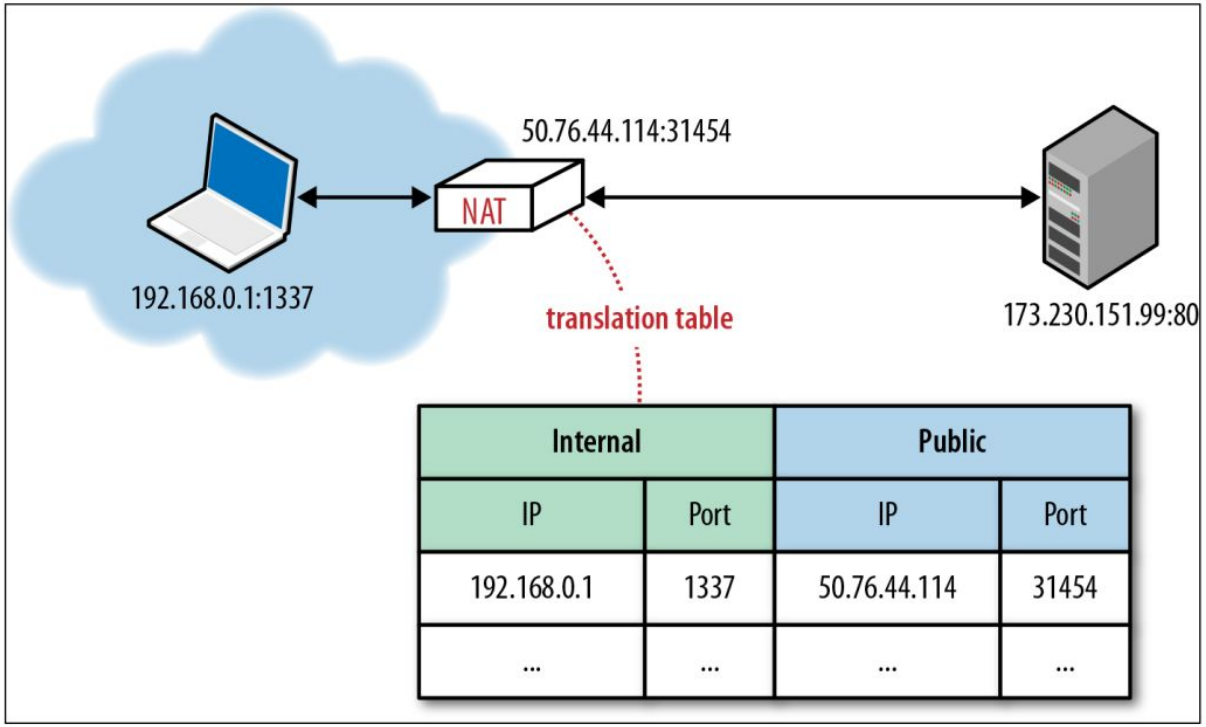
Fundamentos de Redes

- Interligação de Redes
  - Roteador
  - Ponte (Bridge)
  - Hub
  - Switch
  - Modem
  - Virtual Private Network (VPN)
  - Network Address Translation (NAT)
  - Firewall



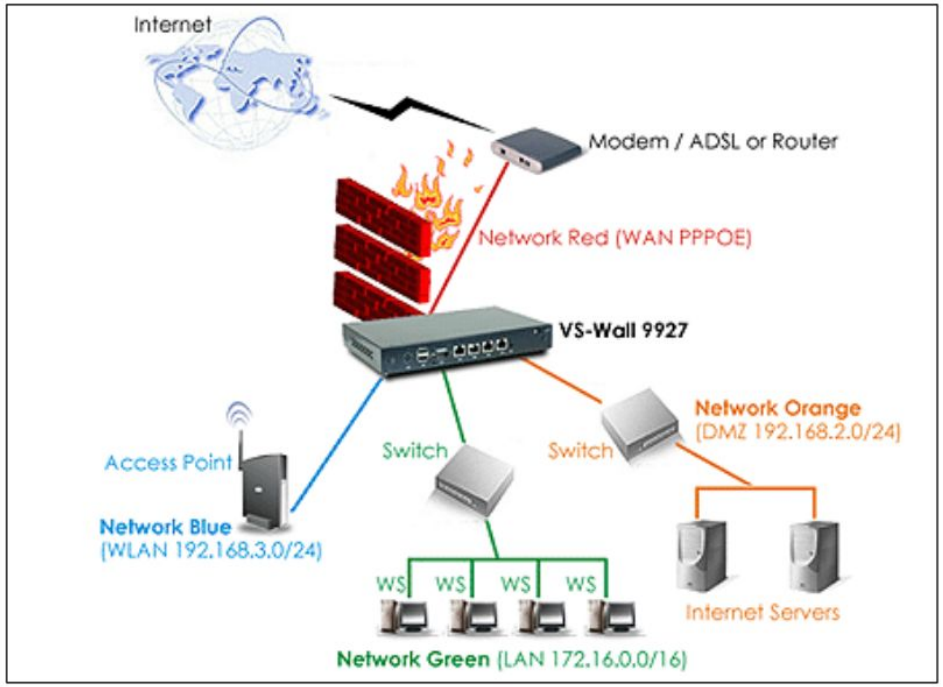
Fundamentos de Redes

Interligação de Redes – Network Address Translation (NAT)



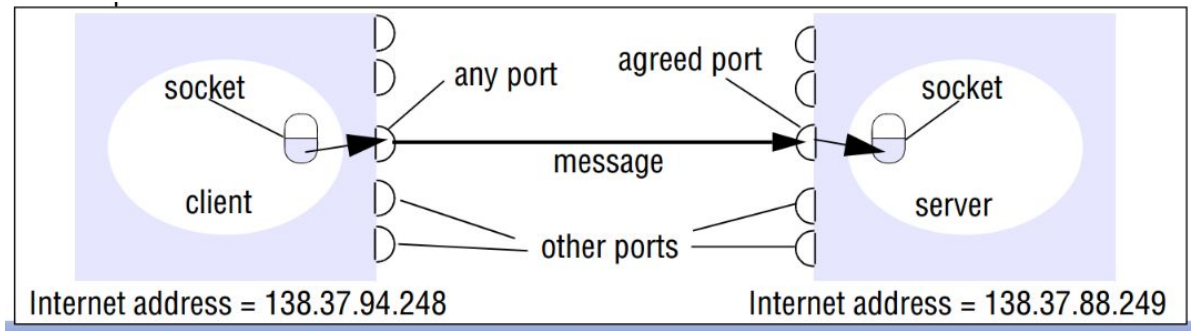
Fundamentos de Redes

Firewall



Fundamentos de Redes

- Características da comunicação entre processos
  - Comunicação síncrona e assíncrona
  - Destinos de mensagem
  - Confiabilidade
  - Ordenamento
- Sockets
  - De acordo com KUROSE: “socket é a interface entre a camada de aplicação e a de transporte dentro de uma máquina”.
- Cada socket tem um endereço único na internet. Este endereço é formado por um número IP e por um número de porta.



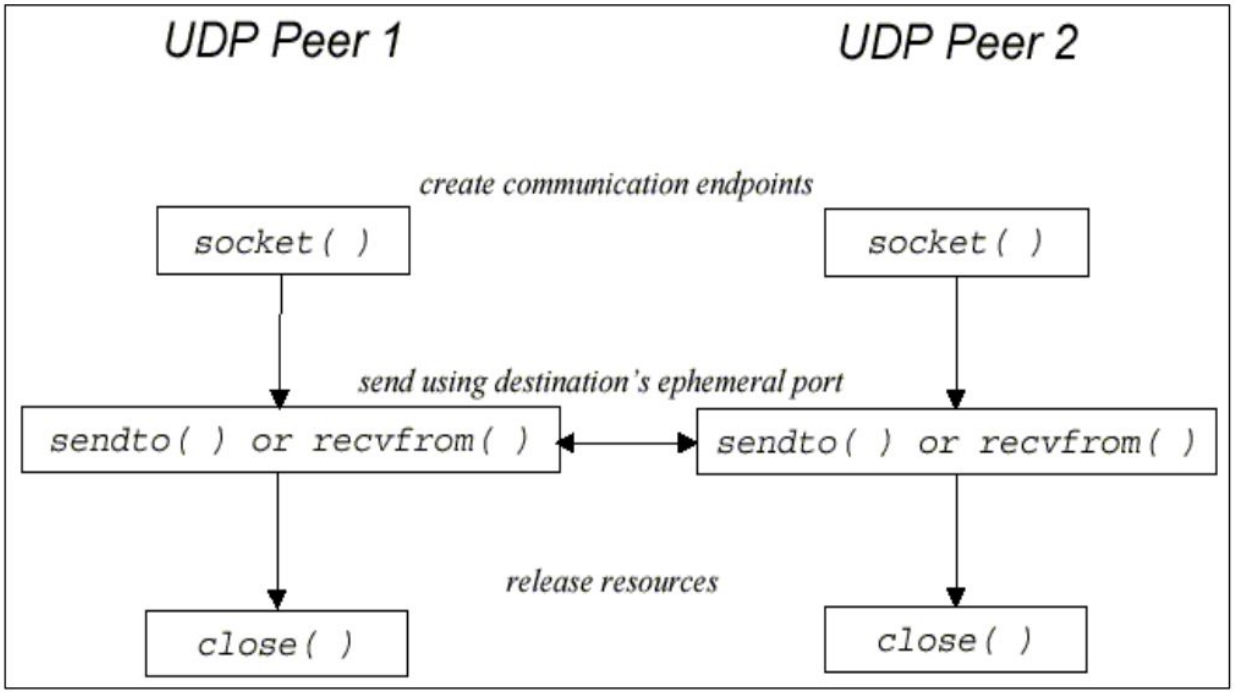
Fundamentos de Redes

API de protocolos Internet – Sockets

- Uma API de comunicação via sockets oferece classes que permitem o envio e o recebimento de dados
- Os sockets podem utilizar:
  - UDP (User Datagram Protocol): Comunicação via datagramas não orientada a conexão
  - TCP (Transmission Control Protocol): Comunicação via stream orientada a conexão
- Ao socket deve ser fornecido o endereço de IP e a porta de destino
- Na plataforma Java, a API de sockets provê as seguintes classes:
  - Comunicação via datagramas: DatagramSocket e DatagramPacket
  - Comunicação via stream: ServerSocket e Socket

Fundamentos de Redes

API de protocolos Internet – Sockets – UDP



Fundamentos de Redes

Programando o cliente

```
import java.net.*;
import java.io.*;
public class UDPClient{
    public static void main(Stringargs[]){
        // args give message contents and server hostname
        DatagramSocket aSocket = null;
        try {
            aSocket = new DatagramSocket();
            byte [] m = args[0].getBytes();
            InetAddress aHost = InetAddress.getByName(args[1]);
            int serverPort = 6789;

            DatagramPacket request = new DatagramPacket(m, m.length(), aHost, serverPort);
            aSocket.send(request);

            byte[] buffer = new byte[1000];
            DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
            aSocket.receive(reply);
            System.out.println("Reply: " + new String(reply.getData()));
        } catch (SocketException e){System.out.println("Socket: " + e.getMessage());
        } catch (IOException e){System.out.println("IO: " + e.getMessage());
        } finally { if(aSocket != null) aSocket.close();}
    }
}
```

Fundamentos de Redes

Programando o servidor

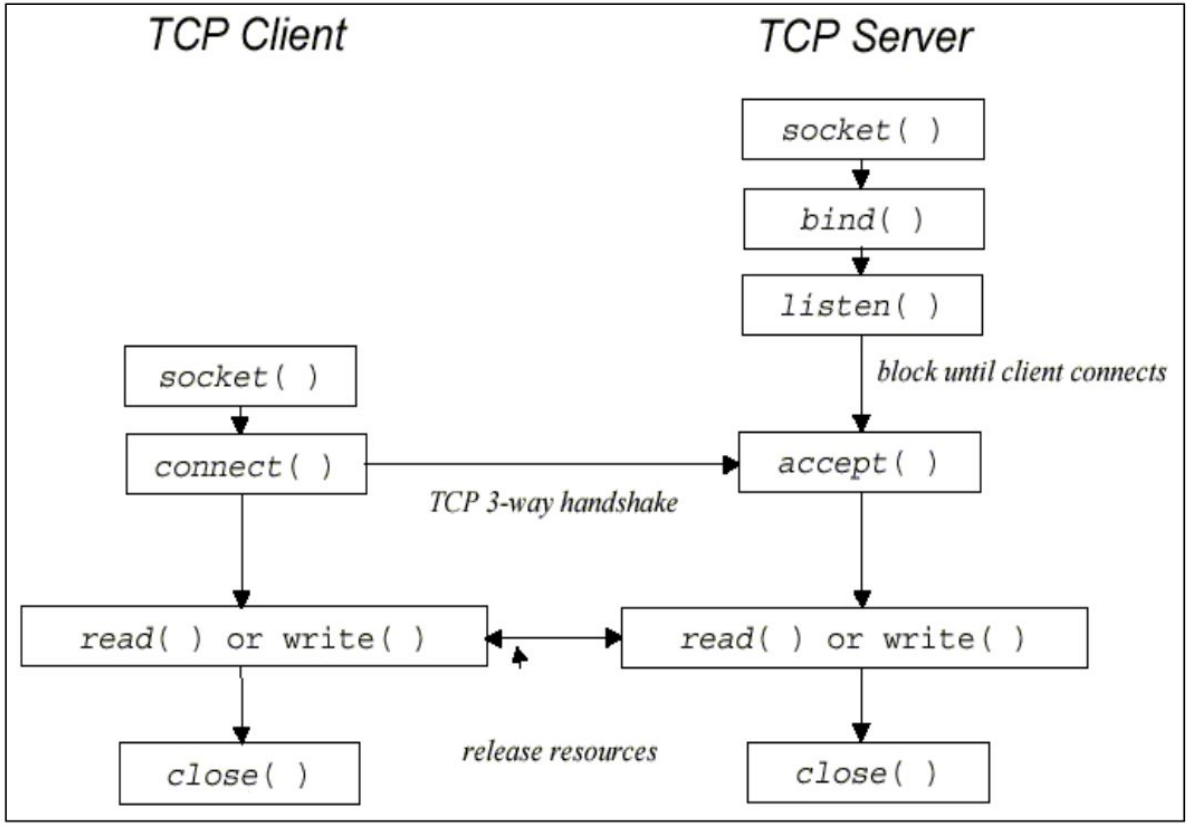
```
import java.net.*;
import java.io.*;
public class UDPServer{
    public static void main(Stringargs[]){
        DatagramSocket aSocket = null;
        try{
            aSocket = new DatagramSocket(6789);
            byte[] buffer = new byte[1000];
            while(true){
                DatagramPacket request = new DatagramPacket(buffer, buffer.length);
                aSocket.receive(request);

                DatagramPacket reply = new DatagramPacket(request.getData(),
                request.getLength(), request.getAddress(), request.getPort());
                aSocket.send(reply);
            }
        } catch (SocketException e){System.out.println("Socket: " + e.getMessage());
        } catch (IOException e) {System.out.println("IO: " + e.getMessage());
        } finally {if (aSocket != null) aSocket.close();}
    }
}
```

Fundamentos de Redes

API de protocolos Internet

Sockets – TCP



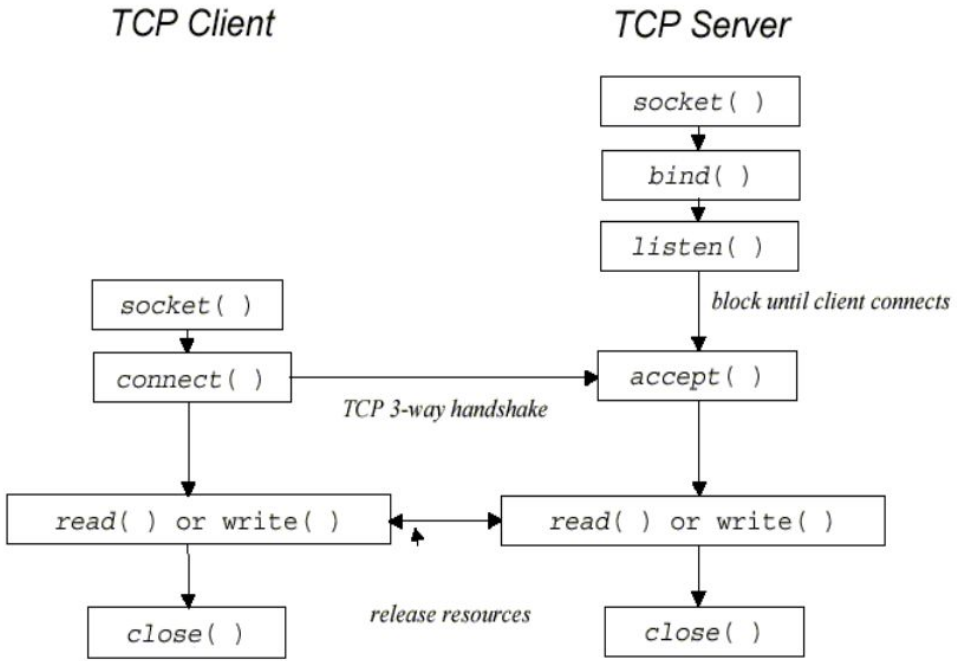
Fundamentos de Redes

Programando o cliente

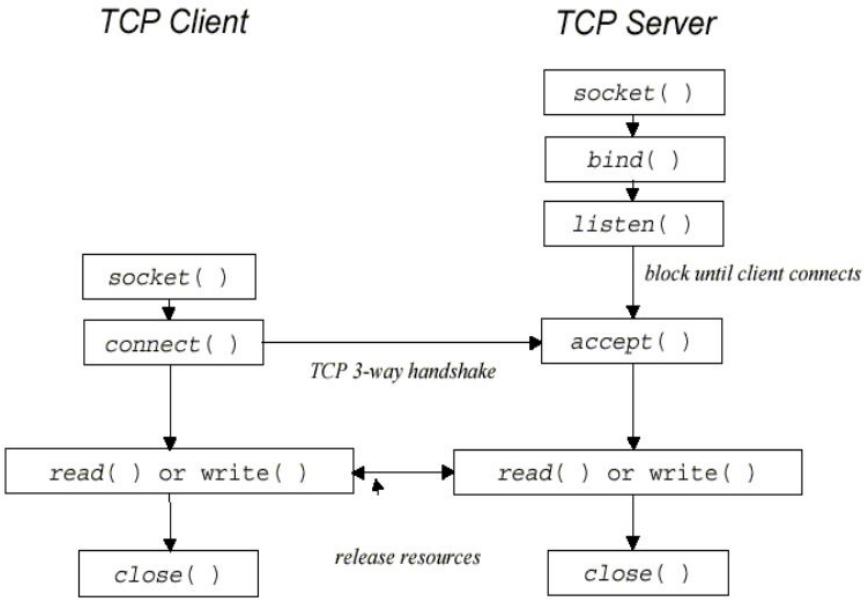
Inicialização
- **gethostbyname** – traduz nome do servidor
- **socket** – cria o socket
- **connect** – conecta à porta do servidor

Trasmissão:
- **send** – envia msg para o servidor
- **recv** – recebe msg do servidor

Encerramento
- **close** – fecha o socket



Fundamentos de Redes



Programando o servidor

Inicialização
- **socket** – cria o socket
- **bind** – associa o socket ao endereço local
- **listen** – associa socket a requisições de entrada

Trasmissão:
- **accept** – aceita conexão de cliente
- **recv** – recebe msg do cliente
- **send** – envia msg para o cliente

Encerramento
- **close** – fecha o socket

Fundamentos de Redes

Programando o cliente

```
import java.net.*;
import java.io.*;
public class TCPClient {
    public static void main (String args[]) {
        // arguments supply message and hostname of destination
        Socket s = null;
        try{
            int serverPort = 7896;
            s = new Socket(args[1], serverPort);
            DataInputStream in = new DataInputStream( s.getInputStream() );
            DataOutputStream out = new DataOutputStream( s.getOutputStream() );

            out.writeUTF(args[0]); // UTF is a string encoding; see Sec 4.3
            String data = in.readUTF();
            System.out.println("Received: " + data) ;
        } catch (UnknownHostException e){
            System.out.println("Socket:" + e.getMessage());
        } catch (EOFException e){System.out.println("EOF:" + e.getMessage());
        } catch (IOException e){System.out.println("IO:" + e.getMessage());
        } finally {if(s!=null) try { s.close();} catch (IOException e){/*close failed*/}}
    }
}
```

Fundamentos de Redes

Programando o servidor

```
import java.net.*;
import java.io.*;
public class TCPServer {
    public static void main (String args[]) {
        try{
            int serverPort = 7896;
            ServerSocket listenSocket = new ServerSocket(serverPort);
            while(true) {
                Socket clientSocket = listenSocket.accept();
                Connection c = new Connection(clientSocket);
            }
        } catch(IOException e) {System.out.println("Listen :"+e.getMessage());}
    }
}
class Connection extends Thread {
     .........
}
```

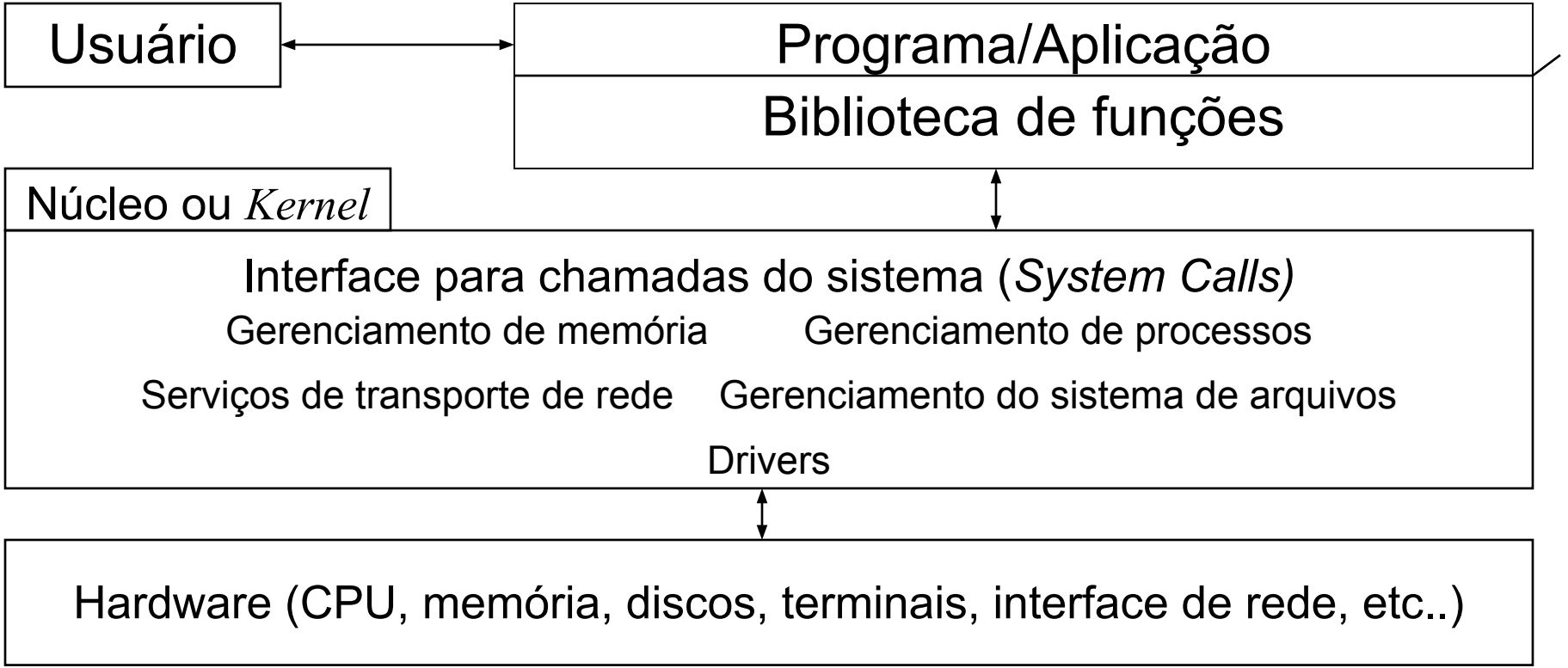
Fundamentos de Redes

Programando o servidor

```
import java.net.*;
import java.io.*;
public class TCPServer {
 .........
}
class Connection extends Thread {
     DataInputStream in;
     DataOutputStream out;
     Socket clientSocket;
     public Connection (Socket aClientSocket) {
          try {
               clientSocket = aClientSocket;
               in = new DataInputStream( clientSocket.getInputStream());
               out = new DataOutputStream( clientSocket.getOutputStream());
               this.start();
          } catch(IOException e) {System.out.println("Connection:"+e.getMessage());}
     }
     public void run(){
          try { // an echo server
               String data = in.readUTF();
               out.writeUTF(data);
          } catch(EOFException e) {System.out.println("EOF:"+e.getMessage());
          } catch(IOException e) {System.out.println("IO:"+e.getMessage());
          } finally { try {clientSocket.close();}catch (IOException e){/*close failed*/}}
     }
}
```

Conceitos de Sistemas Operacionais

Usuário e o Sistema Operacional



Usuário e o Sistema Operacional

Kernel: Núcleo do sistema operacional.

- Drivers: executam o acesso direto ao hardware.
- Sistema de arquivos: provê interface para acesso e armazenamento dos dados.
- Gerenciamento de processos: responsável por escalonar e compartilhar os recursos e o tempo entre os processos.
- Gerenciamento de memória: trata do compartilhamento da memória física entre os processos.
- Serviços de transporte de rede: provê comunicação máquina-máquina ou processo-processo através da rede.

Processos e Programas

Programa:

- uma lista de instruções que especificam uma seqüência de comandos a serem executados.
- entidade passiva.

Processo:

- programa em execução.
- uma tarefa que tem controle sobre um espaço de endereçamento.
- entidade ativa.

Processos

Atributos:

- Process ID (PID): o identificador de processo é um inteiro único que identifica um processo.
- Parent Process ID (PPID): o identificador do processo pai possui o PID do processo que criou o novo processo.
- Real User ID (UID): identificador do usuário que iniciou o processo.
- Effective User ID: usuário que possui os direitos de acesso ao processo.
- Diretório corrente
- Tabela de file descriptors: identifica todos os canais de dado abertos (arquivos, pipes, FIFO, etc...)

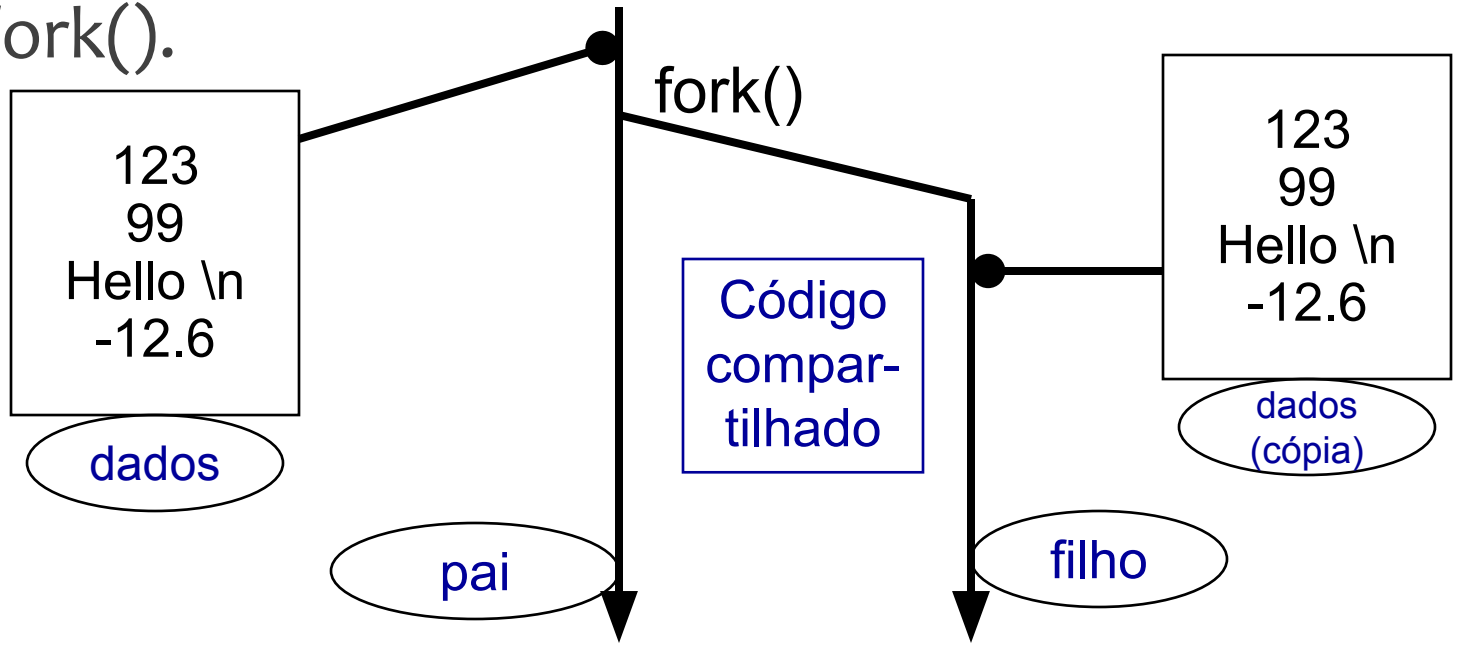
Processos

Atributos:

- Variáveis de Ambiente
- Espaço de código: região de memória onde o programa está carregado
- Espaço de dados: região de memória onde as variáveis globais e os dados estáticos estão carregados.
- Stack (pilha): região de memória para variáveis locais
- Heap: região de memória para dados alocados dinamicamente.
- Prioridade: parâmetro que controla o escalonamento do processo
- Disposição dos sinais: dados tipo flag que identificam a quais sinais o processo deverá responder

Processos Pai e Filho

Um processo pai cria um processo filho através do comando fork().



Processos Pai e Filho

Atributos herdados:

- PID → PPID
- Real UID, Effective UID, Variáveis de Ambiente

Atributos copiados:

- Dados estáticos, Pilha, Heap, File descriptors (se for ponteiro para arquivo é compartilhado), tratamento dos sinais

Atributos compartilhados:

- Código, ponteiros para arquivos.

Comunicação entre Processos

Comunicação entre processos (IPC) de um mesmo sistema centralizado: utiliza memória compartilhada

- Pipes
- FIFO
- Fila de Mensagens
- Memória Compartilhada

Pipes

Forma mais antiga de IPC (Unix, década 70)

Pipe: fila de comunicação **unidirecional** gerenciada pelo kernel

Criação de um pipe:

- int pipe (int d [2]);

Escrita e leitura

int write (d[1], “hello world”);

int read (d[0], buffer, tam_buffer);

Pipes

Pipes para comunicação em um mesmo processo não são de grande utilidade

Pipes entre processos diferentes:

- Criado por um processo pai
- Em seguida, pai executa um fork()
- Então, pipe é usado para comunicação entre pai e filho

Exemplo: Pai → Filho

```
int main () {
    int n, d[2];
    pid_t pid;
    char line[MAX_LINE];
    if (pipe (d) < 0)
        printf (“Erro na criação do pipe”);
    else if ((pid = fork()) < 0)
        printf (“Erro no fork”)};
```

Exemplo: Pai ⟶ Filho

```
else if (pid > 0) {  // processo pai: retorna pid= PID filho
    close (d[0]); // fecha a parte de leitura do PIPE
    write (d[1], “hello world\n”);
 }
 else { // processo filho: pid = 0
      close (d[1]); // fecha a parte de escrita do PIPE
      n= read (d[0], line, MAX_LINE);  // síncrona
      write (1, line, n);                  // 1 = stdout
 }
}  // main
```

Pipes

Fluxo bidirecional: dois pipes

Comando Unix: who | sort | lpr

Desvantagens:

- Unidirecional (half-duplex)
- Somente pode ser usado entre processos pai e filho

FIFO

First In, First Out

Também chamados de named pipes , ou seja, possuem um nome, o que permite que sejam usados entre processos que não tenham um ancestral comum.

Manipulados como se fossem arquivos

- Criação: int mkfifo (const char *name, mode_t mode)
- Outras funções: open, read, write, close

Fila de Mensagens

Lista de mensagens armazenada pelo kernel

Permite comunicações que não sejam FIFO

Mensagens:

```
struct {
long mtype;
char text [512];
}
```

Fila de Mensagens

Recebimento de Mensagens:

- int msgrcv (int msgid, void *ptr, size_t nbytes, long type, int flags)
- type = 0: retorna 1a msg (ordem FIFO)
- type > 0: retorna 1a msg cujo campo “mtype” for igual a “type”
- type < 0: retorna 1a msg cujo campo “mtype” é o menor valor menor que ou igual ao valor absoluto de “type”

Fila de Mensagens

Vantagens:

- Bidirecionais
- Recebimento por mensagens tendo como base seu tipo (nem sempre é FIFO)

Desvantagem:

- Comandos próprios para manipulação (diferentes dos comandos tradicionais de IO)

Memória Compartilhada

Exemplo: Envio de arquivos de um servidor para um cliente via pipe (ou outra estrutura)

Problema: número de cópias do arquivo

- 1a cópia: buffer de leitura do kernel para buffer do servidor
- 2a cópia: buffer do servidor para pipe
- 3a cópia: pipe para buffer do cliente
- 4a cópia: buffer do cliente para buffer de escrita do kernel

Memória Compartilhada

Solução: memória compartilhada entre cliente e servidor (MC)

Apenas duas cópias do arquivo:

- 1a cópia: do buffer de leitura do kernel para MC
- 2a cópia: da MC para buffer de escrita do kernel

Desvantagem:

- Sincronização a cargo do programador (via semáforos etc)