Universidade Federal de Mato Grosso do Sul

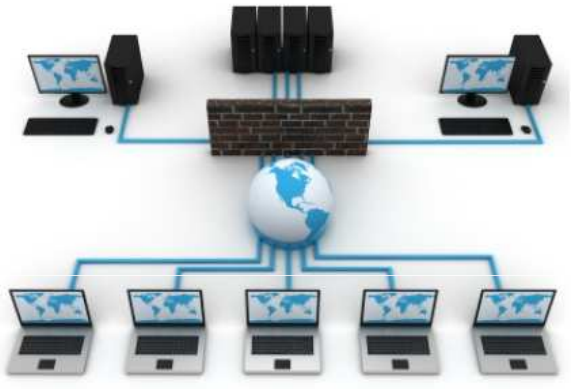
Faculdade de Computação

REDES DE COMPUTADORES

LABORATÓRIO – SOCKETS

prof. Hana Karina Salles Rubinsztejn -
hana@facom.ufms.br

Rede de Computadores



Dentro de uma rede de computadores, podem existir diversos tipos de equipamentos e utilizando vários meios físicos de comunicação: ethernet, wifi, bluetooth.

Camadas de Rede

- As sete camadas de rede tradicionais são divididas em dois grupos:
  - Superiores (7,6,5,4)
  - Inferiores (3,2,1)

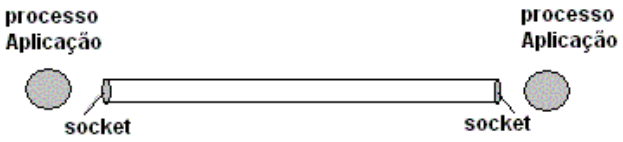
Notem que a ordem numérica das camadas e decrescente.

- A **interface de sockets** fornece uma **API** uniforme para as camadas inferiores de rede e nos permite implementar aplicações nas camadas superiores.

SOCKETS

Sockets

- Os **sockets** são os programas responsáveis pela **comunicação** ou **interligação** de outros programas na internet.
- Um **socket** pode ser entendido como uma ***porta de um canal de comunicação*** que permite a um processo executando em um computador enviar/receber mensagens para/de outro processo que pode estar sendo executado no mesmo computador ou num computador remoto.



Sockets

- Na **camada de transporte**, os sockets suportam especificamente dois protocolos:
  - **TCP** (Transmission Control Protocol)
  - **UDP** (User Datagram Protocol)

Sockets

- TCP:
  - Confiável (em ordem)
  - Controle de fluxo
  - Orientado à conexão
  - Duplex
  - FTP, telnet, http, SMTP
- UDP:
  - Sem confirmação
  - Sem retransmissão
  - Fora de ordem
  - NFS, TFTP

Detalhes sobre o sockets

- Os sockets não sabem quando estão sendo executados sobre ***ethernet***, ***token ring*** ou uma **conexão discada**, nem tem noção alguma sobre os protocolos de alto nível, como NFS, HTTP ou FTP.
- Em algumas situações, a interface de sockets não será a sua melhor escolha para programar para a rede.

Detalhes sobre o sockets

- Existem excelentes bibliotecas (em várias linguagens) que podem utilizar os protocolos em alto nível, sem deixar você se preocupar com os detalhes envolvidos na manipulação das conexões.

- As camadas mais baixas, como por exemplo, no domínio de aplicações para *drivers* de dispositivos tem muito mais interesse nos endereçamentos e manipulações manuais do *socket*.

Detalhes sobre o sockets

- Um "ponto final" para uma conexão IP de rede é determinado por duas coisas:
  - Endereço do host: Endereço IP
  - Número da Porta:

- Duas coisas determinam uma conexão (par socket)
  - ex: 206.62.226.35,porta 21 + 198.69.10.2,porta 1500
  - ex: 206.62.226.35,porta 21 + 198.69.10.2,porta 1499

- Um endereço IP (camada de rede) é representado como um dado de **32-bits**, usualmente representado por quatro campos numéricos separados por ponto, exemplo:
  - 200.129.202.132.
- A porta é um valor de 16-bits, simplesmente representado por um numero menor que 65536.

Portas

- Número das portas:
  - 0-1023: reservado, deve ser root ("**Well Known Ports**")
  - 1024 – 49151: pode ser registrado ("**Registered Ports**")
  - 49152 – 65535: Geral ("**Dynamic and/or Private Ports**")

- Serviços mapeados pela IANA estão em /etc/services:
  - ftp 21/tcp
  - telnet 23/tcp
  - domain 53/tcp e udp
  - snmp 161/udp

Número de Portas Bem Conhecidas

Well known port numbers

Service	Port no	Protocol	
echo	7	UDP/TCP	sends back what it receives
discard	9	UDP/TCP	throws away input
daytime	13	UDP/TCP	returns ASCII time
chargen	19	UDP/TCP	returns characters
ftp	21	TCP	file transfer
telnet	23	TCP	remote login
smtp	25	TCP	email
daytime	37	UDP/TCP	returns binary time
tftp	69	UDP	trivial file transfer
finger	79	TCP	info on users
http	80	TCP	World Wide Web
login	513	TCP	remote login
who	513	UDP	different info on users
Xserver	6000	TCP	X windows (N.B. >1023)

Cliente-Servidor

- Escrevendo nosso primeiro Cliente-Servidor
  - cliente/servidor eco

- **Características de Clientes**
  - Sempre inicia pedidos de servidores;
  - Espera por respostas; Recebe respostas;
  - Normalmente interage diretamente com os usuários finais através de qualquer interface com o usuário;

- **Características do Servidor**
  - Sempre espera por um pedido de um cliente;
  - Atende os pedidos e, em seguida, responde aos clientes com os dados solicitados;
  - Pode se comunicar com outros servidores para atender uma solicitação específica do cliente;

Cliente-Servidor

- Passos para escrever um cliente usando *socket*
  - Os passos que envolvem escrever uma aplicação cliente diferem superficialmente entre o TCP e o UDP.
  - Em ambos os casos:

1. Criar o *socket*
2. TCP: estabelecer a conexão com o servidor
3. Enviar algum dado para o servidor
4. Receber algum dado devolta.
5. *algumas vezes o envio e recepção podem alternar por um tempo*
6. TCP: fechar a conexão

Cliente-Servidor

- Passos para escrever o Servidor
  - Um servidor é um pouco mais complicado que o cliente, principalmente porque ele precisa estar apto a gerenciar vários clientes realizando conexões.
  - Basicamente, existem dois aspectos em um servidor:
    1. Escutar por conexões de clientes
    2. Manipular cada conexão estabelecida
       - Enviar e receber mensagens do cliente

  - Pode-se separar o manipulador de uma conexão em uma função de suporte.
    - Durante o curso vamos aprender algumas formas diferentes de fazer isso (fork, threads).

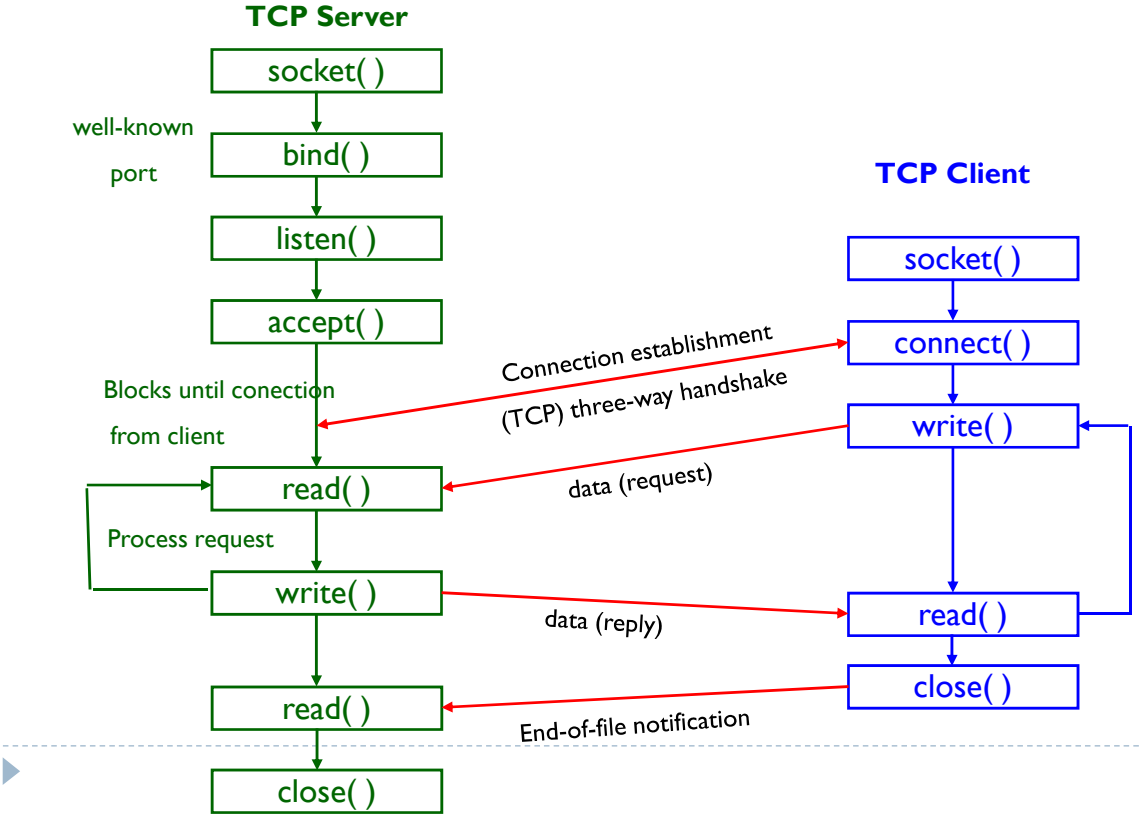
API Sockets

`socket()`	*Cria o socket* Retorna um "socket descriptor" (inteiro) ou -1 em caso de erro.
`bind()`	*Conecta um socket (descriptor) em uma porta.* Retorna 0 se sucesso e -1 se erro. **int bind(int sockfd, struct sockaddr *myaddr, int addrlen);**
`listen()`	*Inicia a escuta na porta, definindo o máximo número de requisições simultâneas.*
`accept()`	*Bloqueia até receber uma conexão. Retorna o socket descriptor de conexão com o cliente.*
`connect()`	*Faz uma conexão com um socket de escuta.* *Retorna 0 se sucesso.* **int connect (int sockfd, struct sockaddr* dest, int destlen);**

`read(), write();`
`recvfrom(), sendto();`
`recv(), send()`
	Funções usadas para transmitir e receber dados.

`close(), shutdown()`	*Encerra a conexão de socket.*

TCP – Cliente/Servidor



Criando um Socket

- **#include <sys/types>**
- **#include <sys/socket.h>**

- **int socket(int family, int type, int protocol);**
  - **RETORNO:** socket descritor, um inteiro (como um identificador de arquivo)
  - **family** especifica a família de protocolo:
    - AF_INET (or PF_INET) for TCP/IP protocols
    - AF_NS (or PF_NS) for Xerox NS protocols
    - AF_UNIX (or PF_UNIX) for Unix internal protocols
  - **type** especifica o tipo de serviço
    - SOCK_STREAM for stream socket (TCP)
    - SOCK_DGRAM for datagram socket (UDP).
    - SOCK_RAW for raw datagrams (IP)
  - **protocol** especifica o protocolo
    - geralmente 0, que significa o default.

Estrutura de endereço

Estrutura de endereços genérica:

```
struct in_addr {
   u_long s_addr;
};
```

Estrutura especializada para endereços da IP :

```
struct sockaddr_in {
    u_short           sin_len;
    short             sin_family;
    u_short           sin_port;
    struct in_addr    sin_addr;
    char              sin_zero[8]; /* não é usado */
};
```

Definidas em <sys/socket.h>

Bind()

- Associa o socket criado a porta local do sistema operacional.
  - Nesta associação **é verificado** se a porta já não está sendo utilizada por algum outro processo.
- Será através desta associação (**porta**) que o programa irá receber dados (**bytes**) de outros programas.
- Função do lado **Servidor**.

- **int** status = bind (**int** sockfd, **struct sockaddr** *addr, **socklen_t** addrlen);

Exemplo de BIND

```
int mysock,err;
struct sockaddr_in myaddr;

mysock = socket(AF_INET,SOCK_STREAM,0); /* criação do socket */

/* Inicialização da estrutura de endereço */
myaddr.sin_family = AF_INET;
myaddr.sin_port = htons( portnum );   /* porta a conectar */
myaddr.sin_addr = htonl( ipaddress);  /* Endereço IP */

/* Associar o socket ao endereço */
err = bind(mysock, (sockaddr *) &myaddr,  sizeof(myaddr));
```

Funções Complementares

- unsigned long inet_addr(char *);
  - Converte um endereço IP representado em String para um valor de 32 bits. Retorna 1 se sucesso e 0 se erro.
- char *inet_ntoa(struct in_addr);
  - Converte um endereço IP inteiro em string.
- bcopy (char *src, char *dest, int nbytes);
  - Similar a memcpy
- bzero (char *dest int nbytes);
  - Zera a estrutura de endereços

Atividades

- Baixar os arquivos cliente_echo.c e servidor_echo.c no ambiente do curso.
  - Compile usando gcc, exemplo:

    % gcc codigo.c -o codigo

Cliente-Servidor

- Cliente echo

cliente_echo.c: cabeçalho

```
#include <stdio.h>          /* printf */
#include <stdlib.h>         /* exit */
#include <string.h>         /* bzero */
#include <sys/socket.h>     /* struct sockaddr, socket */
#include <netinet/in.h>     /* struct sockaddr_in */
#include <arpa/inet.h>      /* inet_pton, htons */
#include <unistd.h>         /* read */

#define BUFFSIZE 32
#define SA struct sockaddr
```

Cliente-Servidor

- Cliente echo

cliente_echo.c: preparando estruturas

```
int main(int argc, char *argv[])
{
  int sockfd, received = 0, bytes = 0;
  struct sockaddr_in servaddr;
  char buffer[BUFFSIZE];

  if (argc != 4) error("Use: TCPecho <server_ip> <palavra> <porta>");

  sockfd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
  bzero(&servaddr, sizeof(servaddr));
  servaddr.sin_family = AF_INET;
  servaddr.sin_addr.s_addr = inet_addr(argv[1]);
  servaddr.sin_port = htons(atoi(argv[3]));
```

Cliente-Servidor

- Cliente echo

cliente_echo.c: conexão e envio de informações

```
  connect(sockfd, (SA *) &servaddr, sizeof(servaddr));
  send(sockfd, argv[2], strlen(argv[2]), 0);
```

- Realizamos a conexão passando para a função `connect` o descritor do *socket* (que é um inteiro), o endereço da estrutura que preenchemos e o tamanho desta estrutura.
- Um detalhe importante fica por conta do *cast* `(SA *)` que força a conversão para a estrutura usada na Internet (`sockaddr`).

Cliente-Servidor

- Cliente echo

cliente_echo.c: recebendo *echo* da mensagem

```
  printf("Recebido: ");
  while(received < strlen(argv[2]))
    {
      bytes = recv(sockfd, buffer, BUFFSIZE-1, 0);
      received += bytes;
      buffer[bytes] = '\0';
      printf("%s", buffer);
    }
  printf("\n");
  close(sockfd);
  exit(0);
}
```

  - Enquanto não recebermos a mensagem completa de volta, ficamos aguardando que ela seja transferida.

Cliente-Servidor

- Servidor echo
  - Cabeçalho similar ao do cliente

servidor_echo.c: declaração de variáveis

```
int main(int argc, char *argv[])
{
  int listenfd, connfd, clientlen, n;
  char buffer[BUFFSIZE];
  struct sockaddr_in servaddr, client;

  if(argc != 2)
    error("Use: SERVERecho <porta>");
```

Cliente-Servidor

- Servidor echo

servidor_echo.c: zerando e inicializando a estrutura

```
listenfd = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);

bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(atoi(argv[1]));

bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
listen(listenfd, MAXPENDING);
```

  - Zeramos o endereço do servidor e preenchemos a estrutura de tal forma a definir que a conexão que estamos fazendo é do tipo TCP/IP.

Cliente-Servidor

servidor_echo.c: laço de recebimento e resposta da mensagem

```
for( ; ; )
  {
    n = -1;
    clientlen = sizeof(client);
    connfd = accept(listenfd, (SA *) &client, &clientlen);
    n = recv(connfd, buffer, BUFFSIZE, 0);

    while(n > 0)
      {
        send(connfd, buffer, n, 0);
        n = recv(connfd, buffer, BUFFSIZE, 0);
      }

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

Cliente-Servidor

- Servidor echo
  - O laço vai ser executado até que o servidor seja terminado, os passos que serão executados neste trecho final do código são:
    1. Aceitar a conexão do cliente e armazenar em connfd
    2. Receber alguma quantidade de bytes em n
    3. Enquanto tiver algo a enviar, ele envia, recebe o retorno do que ainda não foi enviado e fica neste passo até terminar.

Exercícios

- Vamos criar novas funcionalidades!!!
  - 1 - Altere o servidor echo para mostar o IP e porta de conexão do cliente
  - 2 - Faça com que o servidor envie uma mensagem: "conexão realizada“ para o cliente e que o cliente imprima esta mensagem
  - Comente o seu código explicando o que está acontecendo em cada passo do programa.