Universidade Federal de Mato Grosso do Sul

Faculdade de Computação

REDES DE COMPUTADORES

LABORATÓRIO

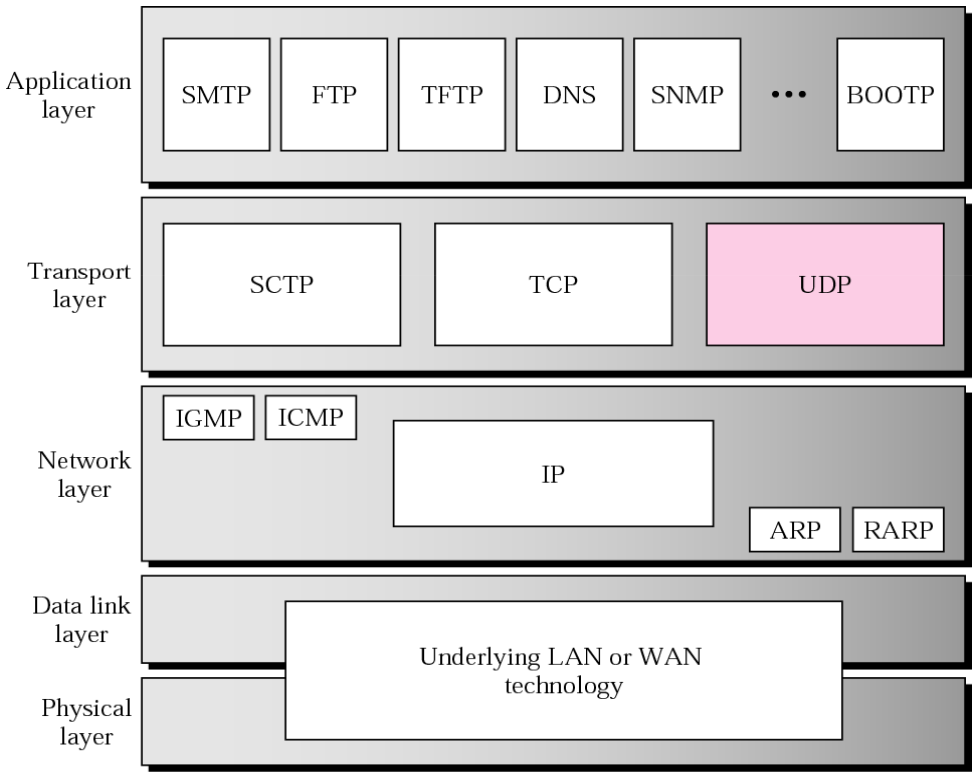
LAB -6

profa. Hana Karina S. Rubinsztejn - hana@facom.ufms.br

Conteúdo

- UDP – User Datagram Protocol Socket
- Exercício

Sockets



TCP vs UDP

- TCP:
  - Confiável (em orderm)
  - Controle de fluxo
  - Orientado à conexão
  - Duplex
  - FTP, telnet, http, SMTP

- UDP:
  - Sem confirmação
  - Sem retransmissão
  - Fora de ordem
  - NFS, TFTP

Tabela de Portas UDP: exemplo

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Nameserver	Domain Name Service
67	Bootps	Server port to download bootstrap information
68	Bootpc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

SOCKETS UDP

- A criação de um socket UDP é feita através da seguinte chamada:

```
int socket(int family, int type, int protocol);
socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)
```

- **1) Perspectiva do RECEPTOR UDP:** A sequência de chamadas para construção de um receptor UDP é a seguinte:

  a - Criar um socket UDP

  b - Preencher o sockaddr_in com o endereço e porta do receptor

  c - Associar o socket ao sockaddr_in (bind)

  d - Entrar no loop de transmissão e recepção

SOCKETS UDP

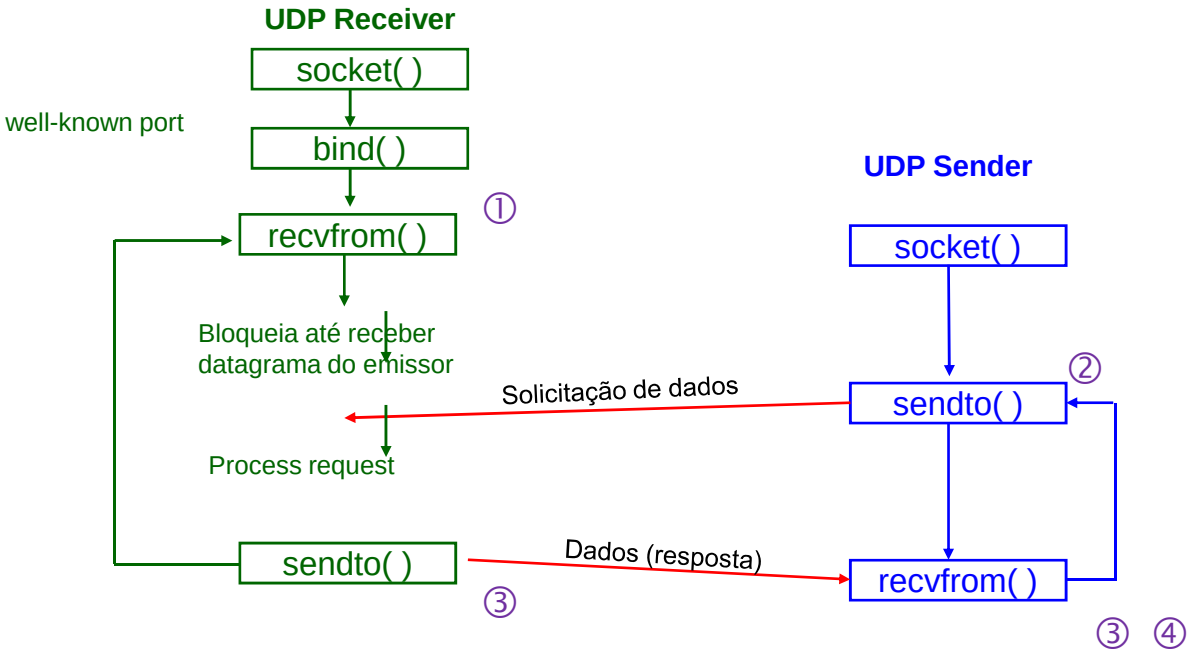
- **2) Perspectiva EMISSOR UDP:** A sequência de chamadas para construção de um emissor UDP é a seguinte:

  a - Criar um socket UDP

  b - Preencher o sockaddr_in com o endereço e porta do receptor para onde os dados serão enviados

  c - Enviar os dados para o endereço definido pelo sockaddr_in

UDP



UDP

① Note que o receptor deve indicar ao seu socket qual a porta em que ele espera receber as mensagens

- Para isso (e por isso) é feito o `bind`, que serve apenas para associar o end IP e porta específica ao socket (que é usado na `recvfrom`)

② No emissor não é preciso fazer `bind` no socket criado

- pois internamente o socket irá selecionar uma porta qualquer disponível, por onde será enviada a mensagem †
- Mas deve-se especificar o end. do destinatário na função `sendto` já que não há uma conexão pré-estabelecida (como ocorria no TCP)

③ Quando o emissor enviou a msg, ele associou internamente uma porta ao seu socket (② †), que permanece aberta. E esse mesmo end/porta será usado pelo receptor para responder à mensagem (usando `sendto`)

- Por isso, o emissor não precisa definir a porta de recebimento (que seria feito pelo `bind`) da resposta ④ no socket do `recvfrom`.

Funções do UDP: Enviando dados

- ssize_t **sendto**(int **sockfd**, void ***buff**, size_t **nbytes**, int **flags**, const struct sockaddr* **to**, socklen_t **addrlen**);
  - sockfd: é um socket
  - buff: é o dado a ser enviado
  - to: endereço do destino
- Retorna o número de bytes enviados

Funções do UDP: Recebendo dados

- ssize_t **recvfrom**(int **sockfd**, void ***buff**, size_t **nbytes**, int **flags**, struct sockaddr* **from**, socklen_t ***fromaddrlen**);
  - sockfd: é um socket
  - buff: é o endereço da dado recebido
  - from: endereço do emissor

- Retorna o número de bytes recebidos e colocados no buff

Exercícios

- Um exemplo simples de cliente/servidor "echo" UDP está disponível no moodle: um receptor (servidor) que ecoa o texto enviado por um emissor (cliente) utilizando UDP.

1. Altere o programa do emissor para que ele:
   a) Receba o endereço (IP e porta) do receptor como parâmetro; e

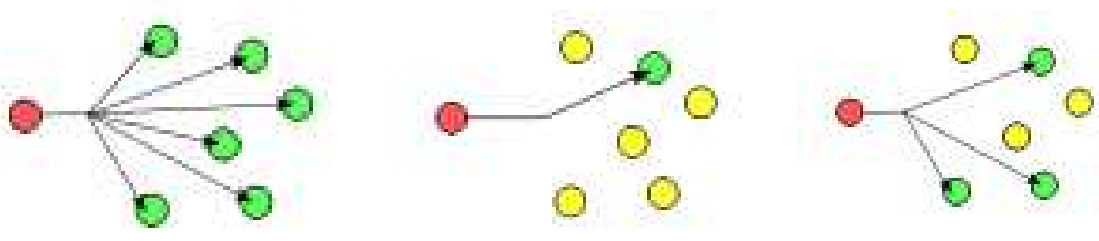
2. Verifique o que acontece nesse exemplo, se o emissor enviar uma mensagem para um receptor que não existe.

Evitando o travamento da aplicação

- Em UDP, quando um problema de comunicação ocorre, a aplicação não é notificada.
- Por isso é necessário utilizar um *timeout* nas operações bloqueantes
  - como por exemplo, na operação de aguardar o recebimento de um pacote.
- Formas de implementar o *timeout*
  - Tratando sinais do SO (`SIGALRM`) e com a função `setitimer`
  - usando opções de socket (com a função `setsockopt`)
  - por meio da função `select` do unix;

Broadcast X Multicast

- Broadcast
  - Envia um pacote uma vez e entrega para **todos** os host de uma rede local
- Multicast
  - Envia um pacote uma vez e entrega apenas para os host de um **grupo** de hosts (hosts na internet)



Broadcast Unicast Multicast

Multicast

- É a entrega de informação para múltiplos destinatários simultaneamente usando a estratégia mais eficiente, onde as mensagens só passam por um *link* uma única vez.
- As informações são duplicadas somente quando o link para os destinatários se divide em duas direções.
- Em comparação com o Multicast, a entrega simples ponto-a-ponto é chamada de **Unicast,** e a entrega para todos os pontos de uma rede chama-se **Broadcast**.

Broadcast

- O broadcast em geral é implementado utilizando-se o UDP
  - Como o TCP precisa abrir uma conexão antes de enviar, seria necessário conhecer todos os PCs da rede, e enviar separadamente a mensagem a cada um.
  - Isso, além de complicado, é um desperdício !!!
- No broadcast com UDP, a mensagem é enviada uma única vez para todos.
  - Para isso é preciso especificar um endereço de broadcast

Broadcast

- Para enviar para todos os hosts da rede local devemos configurar o endereço de broadcast

```
char *broadcastIP = "192.168.0.255" // ou <rede>.255

struct sockaddr_in sock_in;
sock_in.sin_addr.s_addr = inet_addr (broadcastIP);
OU
sock_in.sin_addr.s_addr = htonl(-1);
/* envia a mensagem para 255.255.255.255 */
```

Broadcast

- Além disso, é necessário avisarmos o socket do **emissor** de que utilizaremos o broadcast.
  - Isso pode ser feito configurando o socket com a função ***setsockopt***

- **int** setsockopt(int *socket*, int *level*, int *option_name*, const void **option_value*, socklen_t *option_len*);

  socket: socket criado para udp
  level: especifica o nível do protocolo no qual reside a opção = SOL_SOCKET
  option_name: Nome da opção = SO_BROADCAST
  option_value: 1 para habilitar e 0 para desabilitar (OBS: deve ser colocado em uma variável inteira)
  Retorna 0 se ok e -1 se der erro

EX: config = setsockopt (socketname, SOL_SOCKET, SO_BROADCAST, &setar, sizeof(int));

Opções do socket

- Algumas opções do socket, configuradas via ***setsockopt***, para o **level**: SOL_SOCKET
- **option_name**:
  - SO_BROADCAST
    - permite o envio de pacotes para endereço de broadcast
  - SO_RCVTIMEO e
  - SO_SNDTIMEO:
    - Ativa o ***timeout*** para operações de recebimento ou envio, respectivamente
    - O argumento em *option_value* é uma *struct timeval*.
    - Se a função bloqueante sair por *timeout* retorna -1 com *errno* setado em EAGAIN ou EWOULDBLOCK
  - SO_REUSEADDR
    - Permite o *bind* de mais de um processo na mesma porta local (exceto TCP)

Exercícios

- Desenvolva um difusor de notícias:
  a) Emissor lê uma mensagem do teclado e a divulga a todos os receptores da rede (via *broadcast*).
     - OBS: Receba como parâmetro a porta pelo qual os receptores estarão esperando as mensagens
  b) Os receptores recebem a mensagem propagada, e a imprimem na tela.
     - OBS: Receba como parâmetro a porta na qual espera-se as msgs e faça a associação do socket a ela (*bind*)
  c) Crie uma mensagem de terminação no emissor, que deve ser propagada também aos receptores.

Exercícios

- OBS: Para fins de teste/debug com vários receptores na mesma máquina, ative a opção que permite que vários processos (e seus respectivos sockets) se associem à mesma porta de uma máquina.
  - Para isso, use o comando abaixo nos **receptores**

```
setsockopt(socket, SOL_SOCKET, SO_REUSEADDR, &setar,
sizeof(int) );
```

  - Isso evita erro no `bind`

Referências

- **Programação UDP e TCP sobre "Sockets de Berkeley“**
  - http://www.dei.isep.ipp.pt/~andre/documentos/sockets-berkeley.html