# Sistemas de Operação – Sockets

- O que é um socket?
- Uma interface de comunicação entre processos que podem ou não residir na mesma máquina, mas que não precisam estar relacionados.
- É usado normalmente para implementar um modelo cliente/servidor em aplicações.
- Funcionamento geral:
  - uma aplicação cria um socket
  - o tipo do socket determina o estilo de comunicação (fiável vs. não fiável)
  - envia dados para o socket para transmissão para a rede
  - recebe dados do socket (enviados através da rede por alguma aplicação)

# Existem 2 tipos principais de sockets

- `SOCK_STREAM` (a.k.a. TCP) - definem canais:
  - ligados bi-direccionais,
  - fiáveis na entrega,
  - com ordem de comunicação assegurada,
  - *analogia*: ligação telefónica (precisa que se estabeleça um circuito virtual)
- `SOCK_DGRAM` (a.k.a. UDP) - definem canais:
  - sem noção de ligação (os pacotes incluem o endereço de cada pacote)
  - não são fiáveis na entrega,
  - não garantem ordem
  - *analogia*: os serviços de correio postal (e.g. CTT).

# Criação de sockets em C

- `int sockid= socket( domain, type, protocol)`
  - `sockid`: descriptor do socket, um inteiro; valor -1 se não puder ser criado;
  - `domain`: inteiro a especificar o domínio da comunicação, e.g. `AF_INET` (protocolo IPv4) é comum;
  - `type`: tipo de comunicação: `SOCK_STREAM` ou `SOCK_DGRAM`
  - `protocol`: especifica o protocolo (habitualmente é `0` i.e. protocolo IP); ver lista em `/etc/protocols`.
- Esta função apenas cria a interface do socket!

# Atribuir um nome ao socket – a função bind

- Para que um socket possa ser usado por aplicações cliente numa rede tem de ter nome;
- `int status= bind( sockid, &addrport, size)`
  - `status`: código de erro; -1 se o bind falhar;
  - `sockid`: descriptor do socket;
  - `addrport`: estrutura com o endereço (IP) e porta da máquina (habitualmente `INADDR_ANY` - é escolhido um endereço local)
  - `size`: o tamanho (em bytes) da estrutura `addrport`
- Nota: cada máquina tem 65536 portas, estando algumas reservadas para aplicações específicas e.g. portas 20 e 21 FTP, 80 HTTP, etc.

# O bind nem sempre é necessário

Quando o socket é do tipo:

- `SOCK_DGRAM`
  - se apenas fizer envio, não precisa do bind: o OS procura uma porta sempre que necessário;
  - se recebe, então precisa do bind;
- `SOCK_STREAM`
  - destino determinado no setup da comunicação;
  - não precisa de saber a porta que envia (no setup da comunicação, a parte que recebe é informada da porta)

# Estabelecimento da comunicação: SOCK_STREAM

- Não precisa de se estabelecer uma ligação prévia
- As ligações são estabelecidas por dois tipos de participantes:
  - passivo: espera que um participante activo faça o pedido de ligação;
  - activo: inicia um pedido de ligação ao lado passivo;
- uma vez estabelecida a aligação, os participantes activo e passivo são análogos, i.e.
  - ambos podem enviar e receber dados
  - qualquer um pode terminar a ligação

# Estabelecimento da comunicação (cont.)

- Os passos no estabelecimento de uma ligação são:
  - Passo 1: (part. passivo) fica à escuta de pedidos de ligação
  - Passo 2: (part. activo) faz pedido e estabelece ligação
  - Passo 3: (part. passivo) aceita pedido de ligação
  - Passo 4: (part. passivo) comunica dados
  - Passo 5: (part. activo) comunica dados
- quando um pedido de ligação é aceite, tal acontece num novo socket
- o socket antigo continua à escuta de outros participantes activos

# Funções de ligação: listen e accept

Funções invocadas pelo participante passivo:

- `int status= listen( sockid, queuelen)`
  - `status`: 0 se estiver em escuta; -1 se houver erro;
  - `sockid`: descriptor do socket;
  - `queuelen`: número de participantes activos que podem esperar por uma ligação
  - esta função é **não-bloqueante**, isto é retorna de imediato
- `int newsock= accept( sockid, &name, &namelen)`
  - `newsock`: o novo socket para a comunicação
  - `sockid`: descriptor do socket de escuta
  - `name`: endereço do participante activo (`struct sockaddr`)
  - `namelen`: sizeof(name)

- esta função é bloqueante, isto é, só retorna após ser estabelecida a ligação;

# Pedido de ligação

- `int status= connect( sockid, &name, &namelen)`
  - `status`: 0 se a ligação for bem sucedida; -1 caso contrário;
  - `sockid`: descriptor do socket de ligação;
  - `name`: endereço do participante passivo (`struct sockaddr`)
  - `namelen`: sizeof(name)
- esta função é **bloqueante**, isto é só retorna após ser estabelecida a ligação;

# Envio/Recepção de dados (SOCK_STREAM)

- `int count= send( sockid, &buf, len, flags)`
  - `count`: bytes enviados (-1 se erro)
  - `sockid`: descriptor do socket de comunicação;
  - `buf`: `char[]`, buffer a ser transmitido
  - `len`: tamanho do buffer (bytes)
  - `flags`: opções especiais (habitualmente 0)
- `int count= recv( sockid, &buf, len, flags)`
  - `count`: bytes recebidos (-1 se erro)
  - `buf`: `void[]`, buffer recebido
  - `len`: tamanho do buffer (bytes)
  - `flags`: opções especiais (habitualmente 0)
- funções **bloqueantes**; retornam após envio/recepção;

# Envio/Recepção de dados (SOCK_DGRAM)

- `int count= sendto( sockid, &buf, len, flags, &addr, addrlen)`
  - `count, sockid, buf, len`: como no `send`
  - `addr`: endereço do destino (`struct sockaddr`)
  - `addrlen`: sizeof(addr)
- `int count= recvfrom( sockid, &buf, len, flags, &addr, addrlen)`
  - `count, sockid, buf, len`: como no `recv`
  - `addr`: endereço da origem (`struct sockaddr`)
  - `addrlen`: sizeof(addr)
- funções **bloqueantes**; retornam após envio/recepção;

# Fecho de ligação

- Quando se termina de usar um socket, deve-se fechá-lo;
- `int status= close( sockid)`
  - `status`: 0 se bem sucedido; -1 caso contrário;
  - `sockid`: descriptor do socket a fechar;
- a operação de fecho liberta uma ligação (`SOCK_STREAM`) e liberta a porta usada pelo socket;

# A estrutura `sockaddr`

- Genérica:

```
struct sockaddr{
  u_short sa_family; // address family, hab. AF_UNIX
  char sa_data[14];  // como usar os últimos 14 bytes
};
```

- Específica para Internet:

```
struct sockaddr_in{
  short sin_family; // address family, hab. AF_INET
  u_short sin_port; // # porta (0-65535)
  struct in_addr sin_addr; // internet address
  char sa_zero[8];  // não usado
};
struct in_addr {
  unsigned long s_addr; // 32-bit  (4 bytes)
};
```

# Exemplo de um servidor (envia “Ola Mundo”)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define MYPORT 3490 // porta de ligação para clientes
#define BACKLOG 10 // max ligações pendentes

int main(void)
{
  int sockfd, new_fd;          // escuta em sockfd, novas ligações em newfd
  struct sockaddr_in my_addr; // minha info de endereço
  struct sockaddr_in cl_addr; // info de endereço de clientes
  socklen_t sin_size;

  if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("socket");
    exit(1);
  }
  my_addr.sin_family = AF_INET; // network byte order
  my_addr.sin_port = htons(MYPORT); // converte para short
```

```
my_addr.sin_addr.s_addr = INADDR_ANY; // det. automaticamente IP
memset(&(my_addr.sin_zero), '\0', 8); // resto do struct a zero

if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1) {
  perror("bind");
  exit(1);
}
if (listen(sockfd, BACKLOG) == -1) {
  perror("listen");
  exit(1);
}
while(1) {                   // ciclo de espera de ligações
  sin_size = sizeof(struct sockaddr_in);
  if ((new_fd= accept(sockfd,(struct sockaddr *)&cl_addr, &sin_size)) == -1) {
    perror("accept");
    continue;
  }
  printf("server: pedido de ligacao de %s\n",inet_ntoa(cl_addr.sin_addr));

  if (!fork()) {     // processo filho para responder ao pedido
    close(sockfd);   // não precisa do listener
    if (send(new_fd, "Ola, mundo!\n", 14, 0) == -1) {
      perror("send");
      close(new_fd); exit(0);
    }
  }
  close(new_fd);  // pai já não precisa deste descriptor
  return 0;
}
```

```
}
```

# Exemplo de um cliente

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>

#define PORT 3490       // porta de ligação
#define MAXDATASIZE 100 // num. max de bytes no buffer

int main(int argc, char *argv[])
{ // server_addr must be initialized to the server address (IP and port)
   int sockfd, numbytes;
   char buf[MAXDATASIZE];
   struct hostent *he;
   struct sockaddr_in cl_addr; // info do endereço de ligação

   if (argc != 2) {
     fprintf(stderr,"usar: client hostname\n");
     exit(1);
   }
```

```
  if ((he=gethostbyname(argv[1])) == NULL) {  // host info
    perror("gethostbyname");
    exit(1);
  }

  if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
    perror("socket");
    exit(1);
  }

  cl_addr.sin_family = AF_INET;    // network byte order
  cl_addr.sin_port = htons(PORT);  // short
  cl_addr.sin_addr = *((struct in_addr *)he->h_addr);
  memset(&(cl_addr.sin_zero), '\0', 8);  // resto da estrutura a zero

  if (connect(sockfd, (struct sockaddr *)&server_addr, sizeof(struct sockaddr)) == -1) {
    perror("connect");
    exit(1);
  }

  if ((numbytes=recv(sockfd, buf, MAXDATASIZE-1, 0)) == -1) {
    perror("recv");
    exit(1);
  }
  buf[numbytes] = '\0';
  printf("Recebeu: %s",buf);
  close(sockfd);
  return 0;
}
```