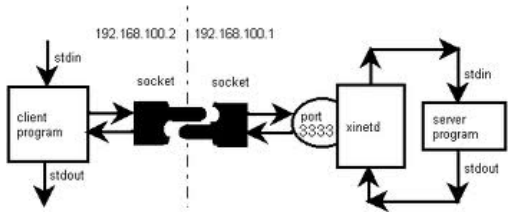
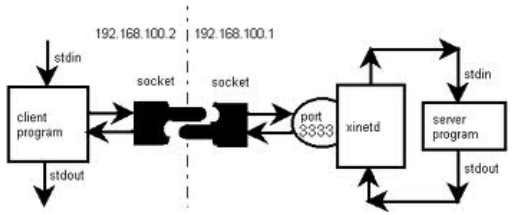


Programação de Sistemas

Sockets

Introdução à Internet (1)



A. Modelo de comunicação na rede de computadores

O mais divulgado é a Internet.

- Baseada numa pilha de protocolos com 4 níveis,
  - níveis mais elevados são mais abstractos e mais próximos do utilizador
  - níveis mais baixos são próximos das transferências de bits entre computadores.
- Cada nível usa serviços oferecidos pelo nível imediatamente inferior.

Utilizador

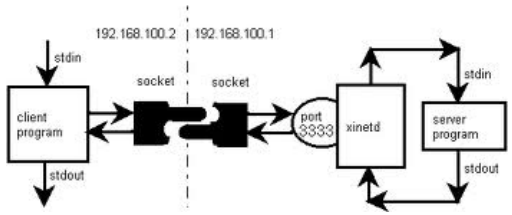
Aplicação
Transporte
Rede
Ligação

Meio físico

Leonard Kleinrock, chefe do laboratório da UCLA que desenvolveu e instalou primeiro encaminhador da ARPANET.

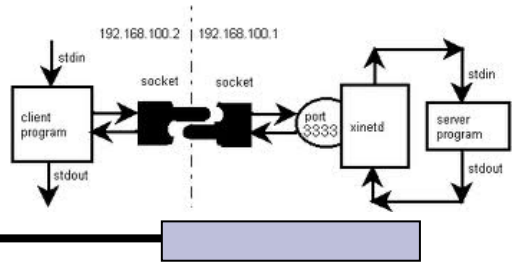
Primeira mensagem enviada da UCLA para SRI/Stanford em Setembro 1969.

Introdução à Internet (2)



4. Nível aplicação ("application"): contém as aplicações úteis para os utilizadores.
   - Objectivos: converter dados na representação local para representação canónica e implementar modelo de comunicação.
   - Exemplos de aplicações muito divulgadas:
     - WWW ("World Wide Web"), baseado no protocolo HTTP-Hypertext Transfer Protocol.
     - Correio electrónico (Email-"Electronic Mail"), baseado no protocolo SMTP-Simple Mail Transfer Protocol.
     - Transferência de ficheiros, baseado no protocolo FTP-File Transfer Protocol.
     - DNS ("Domain Name Service"): transcrição de nomes lógicos (nomes definidos numa hierarquia em árvore) para endereços de nós da rede.

Introdução à Internet (3)

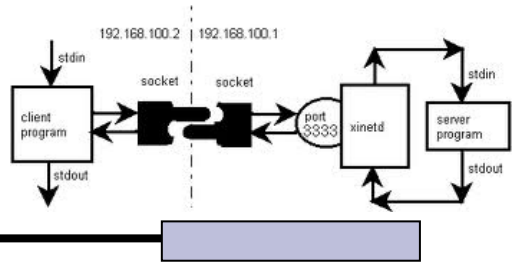


3. Nível transporte ("transport"):
   - Objectivos: transferir todos os dados ponto a ponto (nós da rede de computadores, que podem não se encontrar directamente ligados entre si).
   - Protocolos mais usados:
     - TCP-Transmission Control Protocol, de transferência fiável e ordenada de uma sequência de dados de qualquer dimensão.
     - UDP-User Datagram Protocol, de transferência não fiável de datagramas (bloco formatado de dados, com comprimento máximo).

Nota1: Arpanet tinha em vista garantir em caso de guerra a transmissão de mensagens mesmo que alguns nós intermédios fossem eliminados.

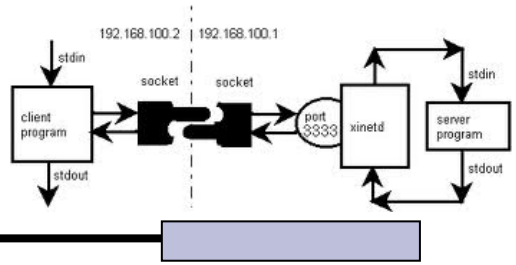
Nota2: protocolos TCP/IP definidos por Robert Kahn e Vinton Cerf.

Introdução à Internet (4)

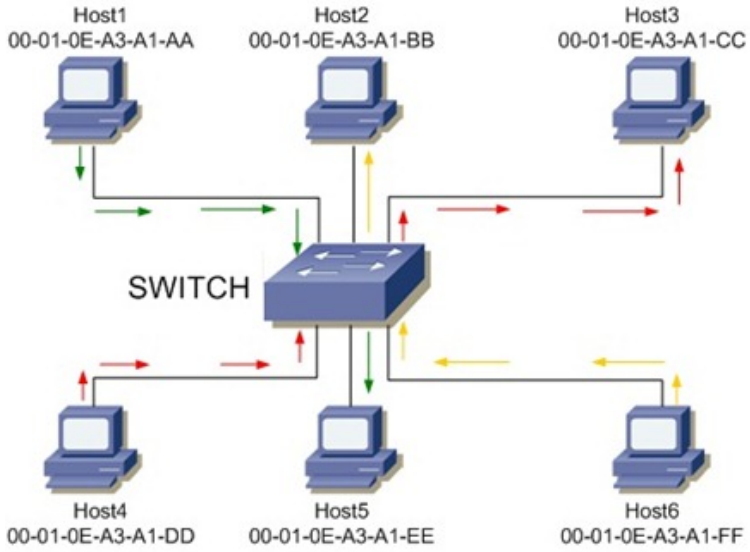


2. Nível rede ("network"):
   - Objectivos: encaminhamento de pacotes (designados aqui por datagramas) de uma rede para outra.
   - Protocolos mais usados
     - IP-Internet Protocol.
     - IPSec, que permite estabelecer em cima do IP uma rede segura (com mecanismos de cifra e autenticação).
1. Nível ligação ("link"):
   - Objectivos: encapsular bits em tramas ("frame"), com detector de erros de transmissão, e enviar as tramas de um computador para outro.
   - Exemplos:
     - Ethernet, numa rede local,
     - 802.11b, para Wi-Fi "Wireless Fidelity"

Introdução à Internet (5)

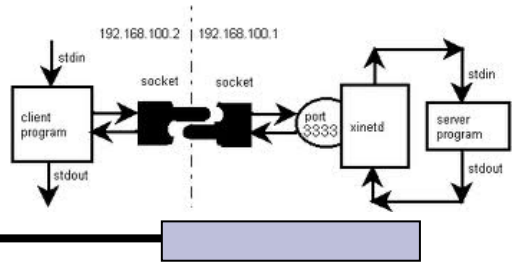


B. Nós intermédios: nós que implementam a transmissão de pacotes de um nó terminal (PC, servidor,...) para outro.
   - *Hub*- que interliga computadores de uma rede local, ou *Switch*- olham para os endereços fonte/ destino dos pacotes para serem transmitidos apenas pelas ligações necessárias.



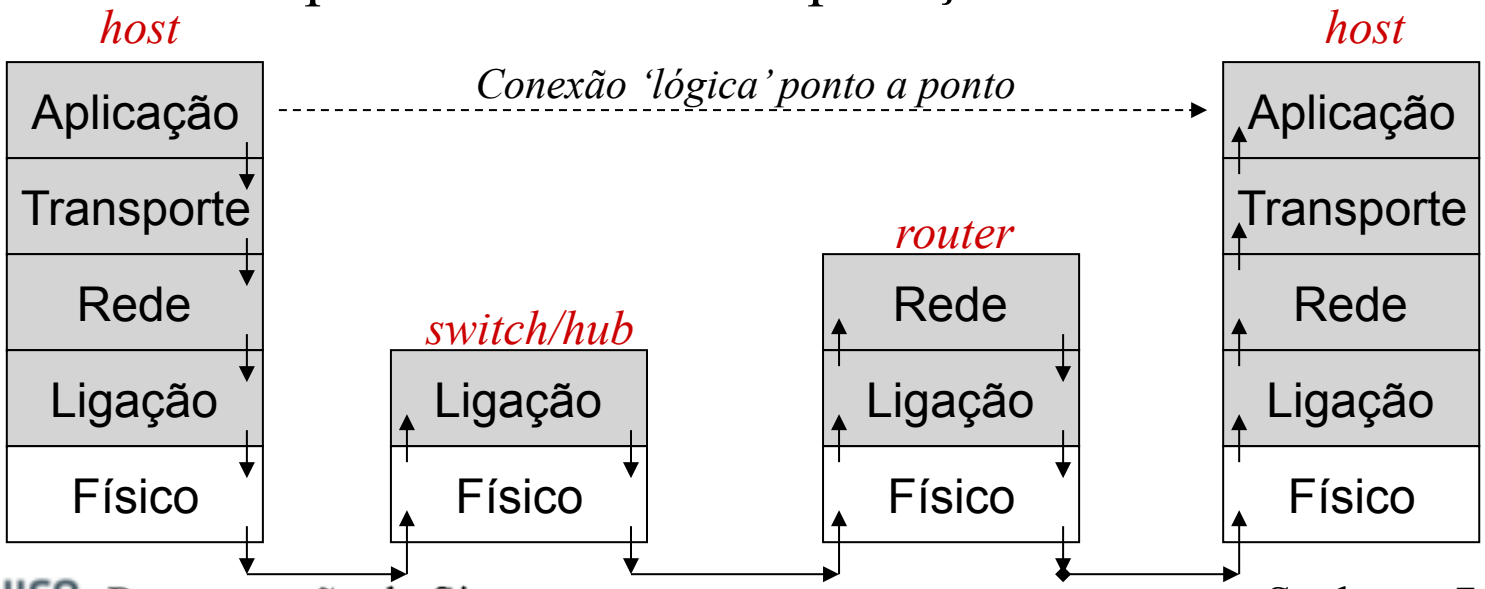
Nota: nas salas SCDEEC2 e SCDEEC3 os switch estão instalados em caixas fechadas.

Introdução à Internet (6)

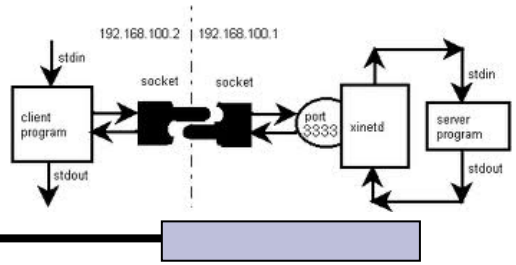


– *Router*- encaminha pacotes entre redes (ex: entre uma LAN-rede local e uma WAN-rede alargada).
Nota1: ao contrários dos hub/switch, os encaminhadores ("router") possuem endereços IP.
Nota2: "router" no SCDEEC instalado na sala dos portáteis.

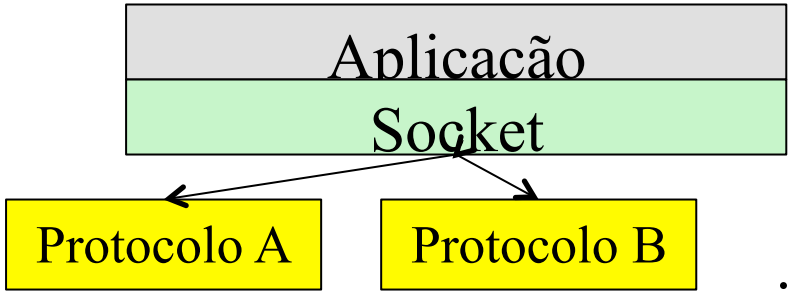
C. Caminho percorrido entre 2 aplicações



Introdução aos sockets (1)



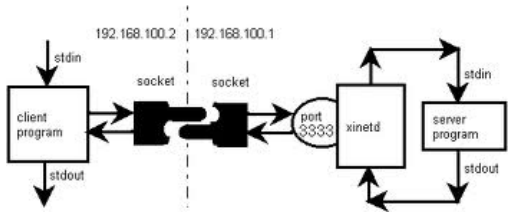
- Os *sockets*, introduzidos em 1981 no BSD 4.1, são uma API e definem os pontos de acesso das aplicações segundo o paradigma cliente/servidor.



...

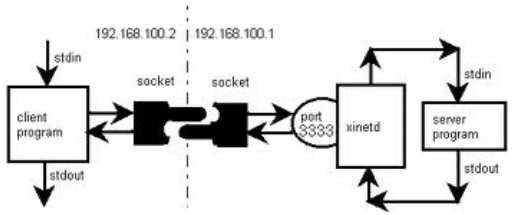
- Programação de sockets mais complexa que E/S de ficheiros
  - mais parâmetros
  - mais chamadas de sistema
- Principal diferença entre E/S de *sockets* e a E/S de ficheiros é a forma como a aplicação abre os descritores de sockets.

Introdução aos sockets (2)



- Os processos lêem e escrevem nos descritores de sockets pelas mesmas operações `read` e `write` dos descritores de ficheiros.
- Serviços disponibilizados pelos *sockets* [analogia]
  - A. `socket()` - definição [telefone]
  - B. `bind()` – associação do socket a um endereço [atribuir número ao telefone]
  - C. `listen()` - escutar [destinatário espera pelo toque da campaínha]
  - D. `connect()` – iniciar conexão [origem digita número do destinatário]
  - E. `accept()` - estabeler conexão [destinatário aceita chamada, levantando o ascultador]
  - F. `send()`,`recv()` - envio/recepção de dados [falar]
  - G. `close()` - fecho de conexão [pousar ascultador

Endereçamento de nós (1)

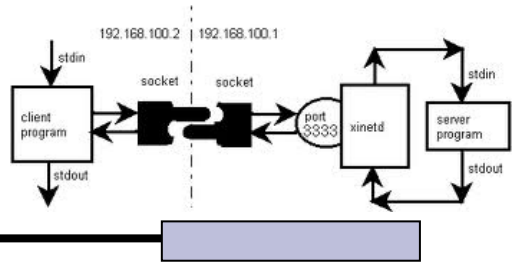


- No protocolo IP, cada nó possui um endereço único. O formato dos endereços depende da versão (IPv4 ou IPv6)

1. **IPv4**: actualmente em uso, norma rfc791 (1981)
   a. Espaço para endereço: 32 bits (4 Bytes), com limite teórico de 4G nós.
   b. Representação de um endereço: notação mais usada é a notação ponto-decimal, cada Byte indicado em decimal.
      Ex: o servidor da Área Científica de Computadores do DEEC tem endereço 193.136.143.1

      Nota: para aumentar legibidade em 1983 foi criado o sistema DNS - ”Domain Name System” para transcrever endereços IP para nomes lógicos na forma *ID.domínio*
      Ex: `asterix.ist.utl.pt` e `comp.ist.utl.pt` são nomes para o endereço 193.136.143.1

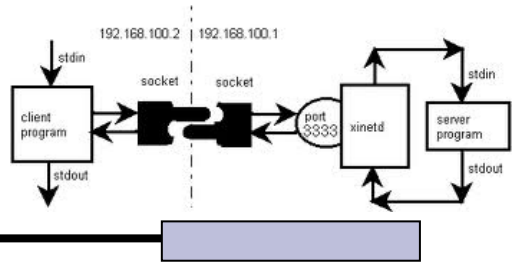
Endereçamento de nós (2)



c. Gama de endereços: representados segundo a representação CIDR- "Classless Internet Domain Routing" na forma base/ máscara. A base é o endereço inicial da gama e a máscara ("netmask") determina os bits fixos no prefixo, nas formas:
   - Completa, indicando o valor de todos os grupos de 8 bits (entre 0 e 255)
   - Curta, com o número de bits fixos.

Filtro curto	Máscara	Número de endereços
/24	255.255.255.0	256
/25	255.255.255.128	128
/26	255.255.255.192	64
/27	255.255.255.224	32
/28	255.255.255.240	16
/29	255.255.255.248	8
/30	255.255.255.252	4

Endereçamento de nós (3)

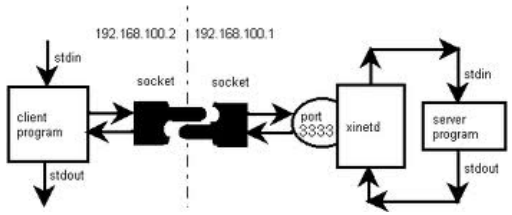


- Ex: `193.136.143.0/24` indica prefixo de 24 bits, podendo variar os 32-24=8 bits menos significativos: assim é coberta a gama `193.136.143.0` a `193.136.143.255`
- Ex: `193.136.143.120/29`, podem variar 3 bits menos significativos: assim é coberta a gama `193.136.143.120` a `193.136.143.127`

d. Reserva de endereços: a IANA-Internet Assigned Numbers Authority reservou gamas de endereços
   127.0.0.0/8 – loopback
   10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16 – redes locais
   255.255.255.255 – difusão

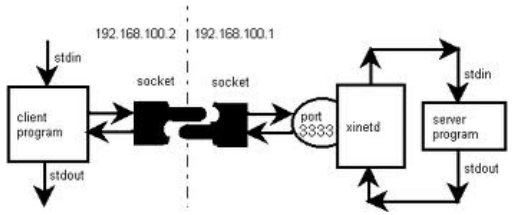
e. Dimensão máxima do datagrama, designada MTU ("Maximum Transmission Unit"), é 64KB-20B

Endereçamento de nós (4)



- Para prolongar a vida do espaço de endereçamento várias tecnologias têm sido desenvolvidas:
  - Translacção de endereços (NAT-"Network Address Translation") por uma "Firewall" entre a rede privada e local. Os endereços da rede interna são trocados pelo endereço externo da "Firewall".
  - Aluguer de endereços por serviço DHCP-"Dynamic Host Configuration Protocol"
    - Usado quando um PC em casa se liga a um ISP-Internet Service Provider.
    - Tempo de aluguer varia com o tipo de equipamento (ex: 2 dias para computador fixo e 1 hora para computador móvel) e pode ser estendido depois de gasto 50% do tempo de aluguer.
- A procura de endereços IP provém de diversos factores:
  - Aumento de utilizadores de Internet.
  - Popularidade de equipamentos móveis (portáteis, PDAs, telemóveis).
- IANA prevê para 2010 o esgotamento dos endereços IPv4.

Endereçamento de nós (5)



2. **IPv6**: em fase de instalação, norma rfc2373 (1998)
   a. Espaço para endereço: 128 bits (16 Bytes), com limite teórico de $3.4*10^{38}$ nós.
   b. Representação de um endereço: notação mais usada é dado por 8 grupos de 4 dígitos hexadecimais, separados por dois pontos (:). Uma única sub-sequência de quatro 0's pode ser compactada por um par de dois pontos.
      Ex: 2001:0db8:0000:0000:0000:0000:1428:57ab e 2001:db8::1428:57ab são endereços IPv6.

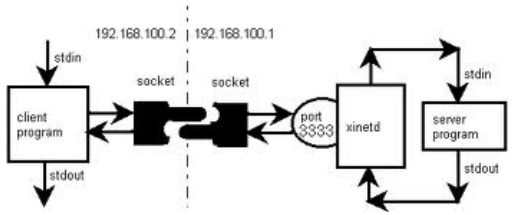
[Exemplo]

```
[rgc@asterix ~]$ host ipv6.l.google.com
ipv6.l.google.com has IPv6 address 2001:4860:0:1001::68
```

Nota 1: as pilha de protocolos IPv4 e IPv6 são incompatíveis.

Nota 2: IPv6 foi disponibilizado no núcleo do Linux 2.6.10.

Representação de Bytes (1)



- A distribuição dos Bytes de números de grande dimensão pode ser feita de duas formas[1]:
  - **Big-endian**: endereços mais baixos com índices superiores (ex: MC68000).
  - **Little-endian**: endereços mais baixos com índices inferiores (ex: Intel x86).

Ex: seja o inteiro 1025 (0x00000401), a representar numa máquina de 32-bits a partir do endereço base 0x10000.

0x10003	01
0x10002	04
0x10001	00
0x10000	00

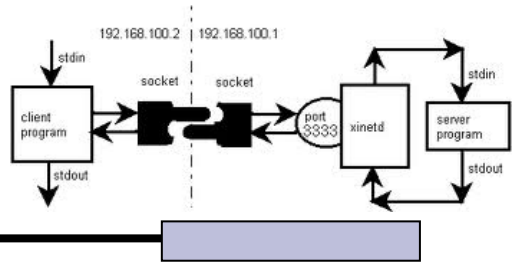
Big-endian

0x10003	00
0x10002	00
0x10001	04
0x10000	01

Little-endian

[1] termos recolhidos da novela Viagens de Gulliver, de Jonathan Swift, sobre os lados favoritos de quebrar em Lilliput os ovos cozidos.

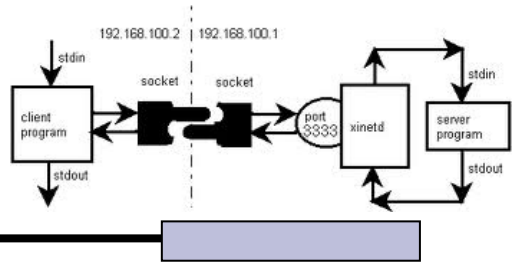
Representação de Bytes (2)



- Vantagens da representação big-endian:
  - Inteiros armazenados na mesma ordem das cadeias de caracteres (da esquerda para a direita).
  - Sinal do número determinado olhando para o Byte de endereço base.
- Vantagens da representação little-endian:
  - Facilita conversão de representações com diferentes tamanhos (ex: inteiro 12 é representado pelo Byte 0x0C e pela palavra 0x000C: a localização de deslocamento mantém o mesmo valor, o que não acontecia na representação bi-endien).
- Na rede Internet, os endereços são sempre representados em big-endian.

  Nota: os primeiros nós encaminhadores ("router") da rede ARPANET, designados Interface Message Processor, eram máquinas Honeywell DDP-516 de 16 bits com representação big-endian.

Representação de Bytes (3)

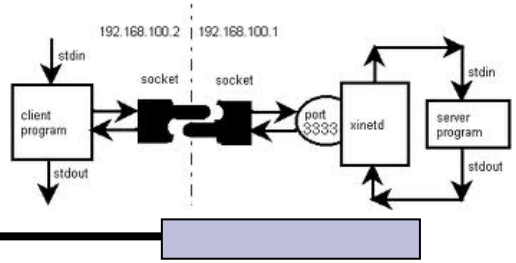


- No API de sockets deve-se sempre chamar as funções de conversão da representação de inteiros :
  - Dimensões: 4 Bytes (long) e de 2 Bytes (short)
  - Locais: nó ("host") e protocolo IP ("network")

```
#include <arpa/inet.h>
uint32_t htonl(uint32_t);     /* 4B do nó para rede */
uint16_t htons(uint16_t);     /* 2B do nó para rede */
uint32_t ntohl(uint32_t);     /* 4B da rede para nó */
uint16_t ntohs(uint16_t);     /* 2B da rede para nó */
```

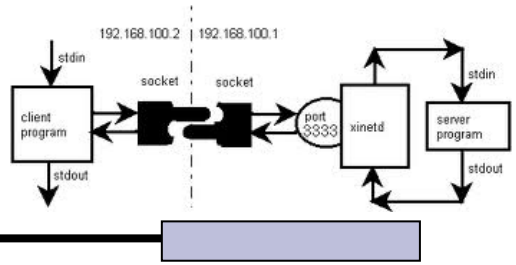
Nota: se o processador tiver arquitectura big-endien, as funções não fazem nada.

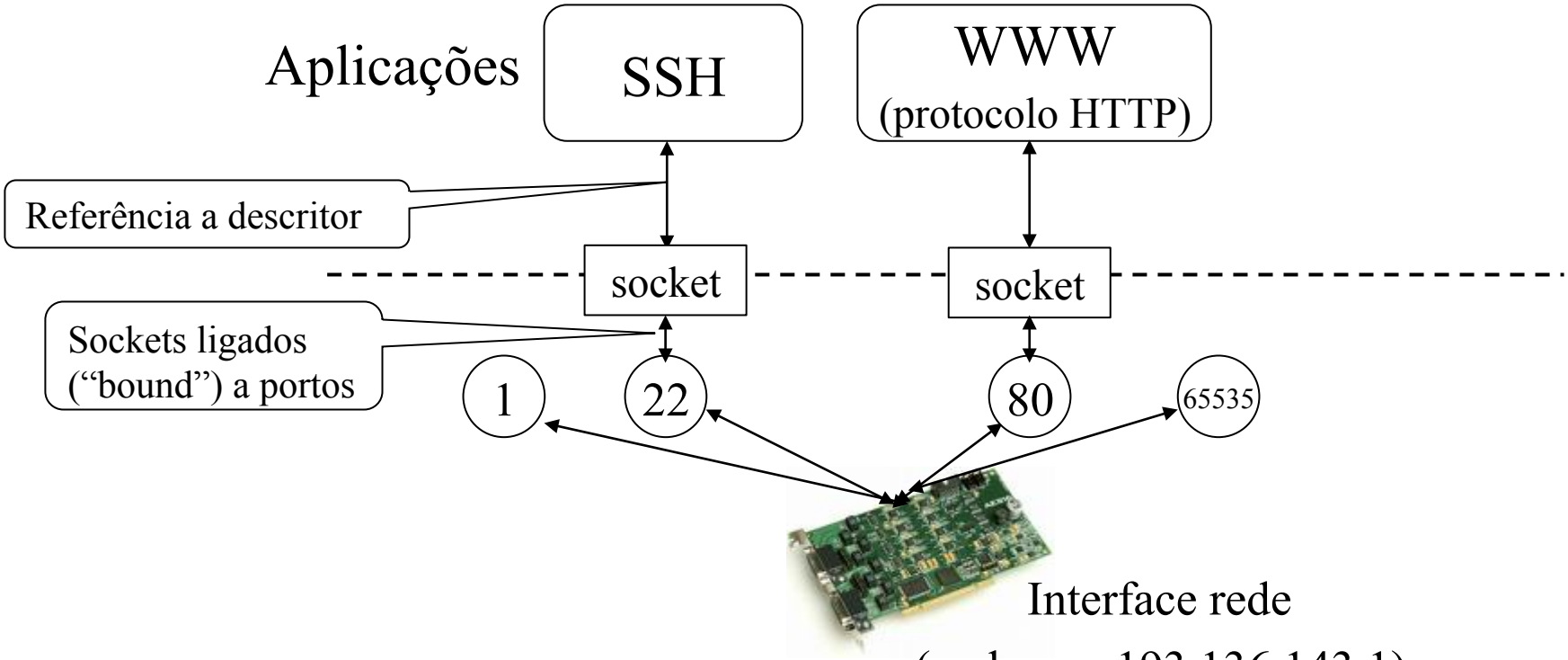
Portos (1)



- O *socket* disponibiliza uma interface de envio/recepção de dados através de um porto.
- Cada tipo de socket pode ser ligado a 64K portos.
  - Os primeiros 1K portos (1-1023) são reservados pelo IANA para serviços específicos, listados no ficheiro `/etc/services`, com privilégios de *root*:
    - 22 : SSH
    - 53 : DNS
    - 80 : WWW
    - 115 : FTP seguro
    - 443 : WWW seguro
  - Os portos 49152-65535 são de evitar, por serem usados dinamicamente pelos sistemas operativos.

Portos (2)

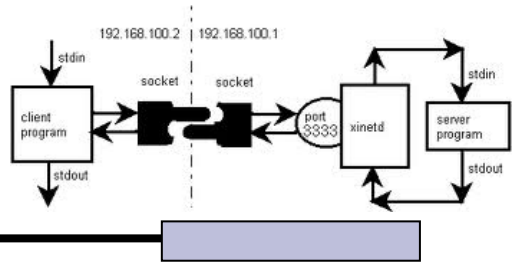




Nota:

- Aplicações podem usar mais do que um socket
- Sockets podem ser acedidas por várias aplicações

Domínio de um socket (1)

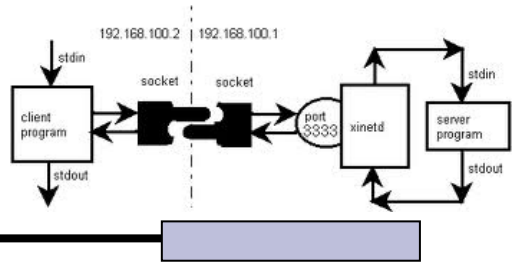


- O domínio do *socket* determina a natureza da comunicação, incluindo o formato dos endereços do nós de computadores.
- Os domínios, todos com prefixo `AF_`, são definidos no cabeçalho `<bits/socket.h>`, lido a partir do `<sys/socket.h>`

Domínio	Valor	Descrição
AF_UNSPEC	0	Não especificado
AF_UNIX	1	Espaço de nomes Unix (tubos e ficheiros)
AF_INET	2	Internet IPv4
AF_INET6	10	Internet IPv6

Figura 16.1 Advanced Programming in the UNIX Environment

Domínio de um socket (2)



- A API socket define no cabeçalho `<sys/socket.h>` uma estrutura genérica para endereços de todos os domínios.

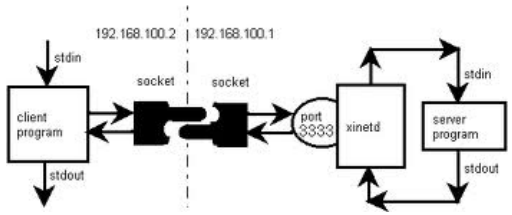
```
struct sockaddr {
  sa_family_t sa_family;
  char        sa_data[14];
};
```

- Para o domínio AF_INET, o cabeçalho `<netinet/in.h>` define a estrutura

```
struct sockaddr_in {
  short          sin_family;
  u_short        sin_port;       /* número de porto */
  struct in_addr sin_addr;       /* endereço IP */
  char           sin_zero[8];    /* não usado */
};
```

- Para o domínio AF_INET6 é usada a estrutura `sockaddr_in6`

Domínio de um socket (3)

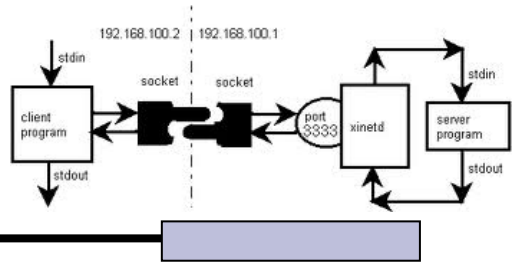


- Transcrição de endereço entre notação binária e cadeia de caracteres em notação ponto decimal efectuada por funções

  POSIX:
  ```
  #include <sys/types.h>
  #include <sys/socket.h>
  #include <arpa/inet.h>
  char *inet_ntop(int,void *,char *,socklen_t);
  int inet_pton(int, char *, void *);
  ```

  – O 1º parâmetro identifica a família.
  – O 2º parâmetro referencia a localização fonte (endereço a transcrever).
  – O 3º parâmetro referencia a localização destino
  – O 4º parâmetro `socklen_t` indica espaço (`INET_ADDRSTLEN` para família `AF_NET` e `INET6_ADDRSTLEN` para família `AF_NET6`)

Tipo de socket (1)

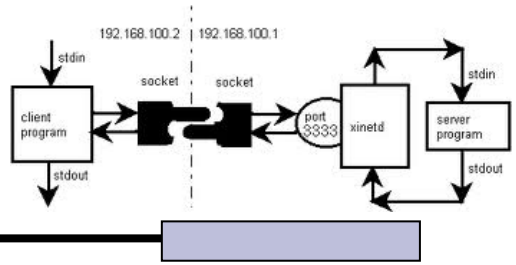


- O tipo de *socket* determina as características de comunicação.
- Os tipos , todos com prefixo SOCK_, são definidos no cabeçalho `<bits/socket.h>`, lido a partir do `<sys/socket.h>`

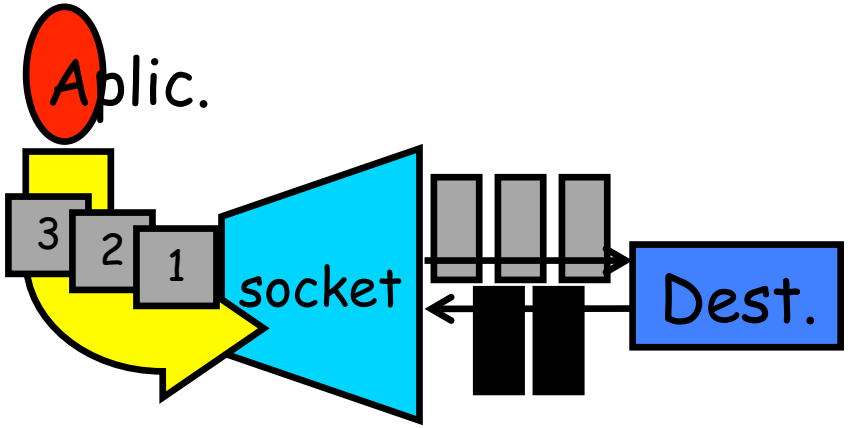
Tipo	Valor	Característica da comunicação
SOCK_STREAM	1	Fluxo fiável de dados
SOCK_DGRAM	2	Orientado ao serviço
SOCK_RAW	3	Entrega directamente datagrama ao protocolo de rede
SOCK_SEQPACKET	10	Igual a SOCK_STREAM, mas orientado à mensagem

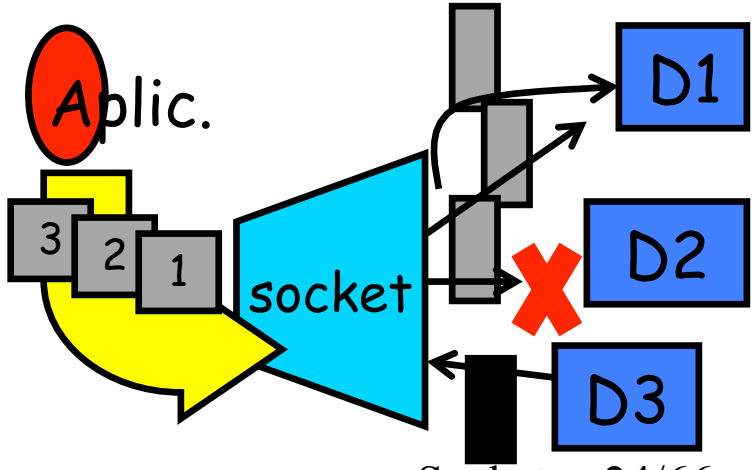
Figura 16.2 Advanced Programming in the UNIX Environment

Tipo de socket (2)

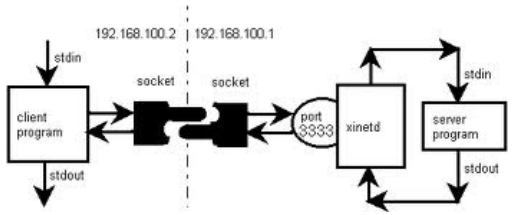


- SOCK_STREAM
  - Entrega fiável.
  - Ordem garantida (1º a enviar é o 1º a ser entregue,…)
  - Orientado à ligação (todos os pacotes seguem depois de ter sido estabelecda a ligação)
  - Bidireccional
- SOCK_DGRAM
  - Entrega não fiável (pacotes podem ser perdidos ou alterados)
  - Não há garantia de ordem na chegada
  - Ligação inexistente (aplicação define destino para cada pacote)
  - Pode enviar ou receber





Tipo de socket (3)



- A escolha do tipo depende da aplicação

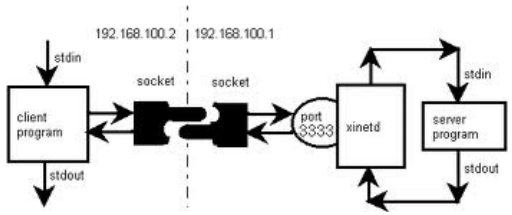
Aplicação	Sensível à perda de dados?	Largura de banda	Restrições de tempo?
FTP	Sim	Elástica	Não
Email	Sim	Elástica	Não
WWW	Sim	Elástica	Não
Áudio	Não	5 Kbps	100 ms
DNS	Não	Baixa	Sim
Financeira	Sim	Elástica	Sim e não

Seleccionado tipo SOCK_STREAM

Seleccionado tipo SOCK_DGRAM

Seleccionado tipo SOCK_DGRAM ou SOCK_STREAM

Obtenção do endereço IP (1)

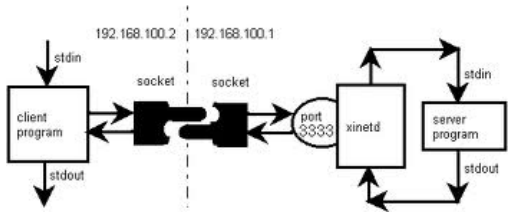


A. Para se obter o endereço IP a partir do nome do nó e o número de porto a partir do nome do serviço, usar a função

POSIX:
```
#include <sys/socket.h>
#include <netdb.h>
int getaddrinfo(char *, char *,
                struct addrinfo *,
                struct addrinfo **);
```

- O 1º parâmetro referencia localização do nome do nó.
- O 2º parâmetro referencia localização do nome do serviço.
- O 3º parâmetro é critério de filtragem de endereços.
- O 4º parâmetro indica endereço para onde é inserida a resposta, numa lista.

Obtenção do endereço IP (2)

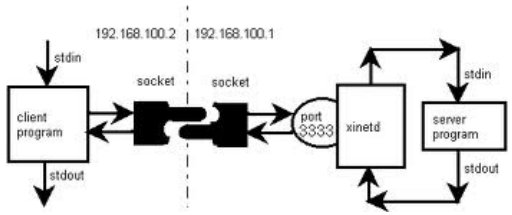


- A estrutura `addrinfo` possui obrigatoriamente os seguintes campos:

```
struct addrinfo {
  int         ai_flags;
  int         ai_family;      /* domínio */
  int         ai_socktype;    /* tipo socket */
  int         ai_protocol;    /* protocolo */
  socklen_t ai_addrlen;       /* comprimento endereço */
  struct sockaddr *ai_addr;           /* endereço */
  char        *ai_cannonname;         /* nome nó  */
  struct addrinfo *next;              /* seguinte na lista */
  …
};
```

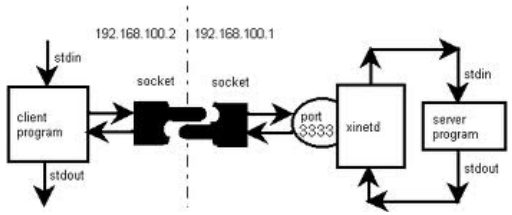
- Para filtragem nula, os campos `int` devem ter valor 0 e os ponteiros devem ser `NULL`.

Obtenção do endereço IP (3)



- Mensagem explicativa de erro não é dada por `perror(int)`, mas
```
#include <netdb.h>
char *gai_strerror(int);
```
- O campo `ai_flags` determina a forma de processar os nomes e endereços
  - `AI_ADDRCONFIG`: Interrogar para qualquer tipo de endereço configurado (IPv4 ou IPv6).
  - `AI_CANONNAME`: Requere nome canónico.
  - `AI_NUMERICHOST`: Endereço do nó em formato numérico.
  - `AI_NUMERICSERV`: Serviço retornado como porto numérico.
  - `AI_V4MAPPED`: Se o endereço tiver formato IPv4, mapeá-lo para formato IPv6.

Obtenção do endereço IP (4)



B. Na biblioteca normalizada do C, usar

```
#include <netdb.h>
struct hostent *gethostbyname(char *);
```

– O parâmetro pode ser dado em "dot-notation" ou nome.

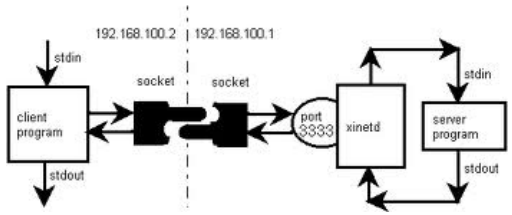
```
typedef struct {
  char *h_name;                 /* Official name of host.  */
  char **h_aliases;             /* Alias list.  */
  int h_addrtype;               /* Host address type.  */
  int h_length;                 /* Length of address.  */
  char **h_addr_list;           /* List of addresses from name server.  */
#define h_addr h_addr_list[0]   /* Address, for backward compatibility.  */
} hostent ;
```

C. No "shell" o endereço pode ser recolhido na forma `host nome`

[rgc@asterix ~]$ host ottawa.ist.utl.pt

ottawa.ist.utl.pt has address 193.136.143.66

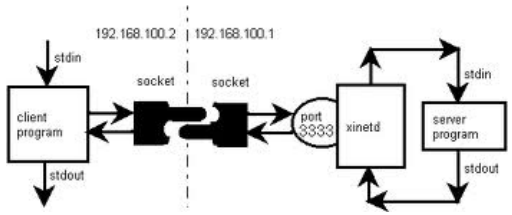
Exemplo de identificação (1)



- Consideremos um programa que lista o endereço de um nó e o porto de um serviço

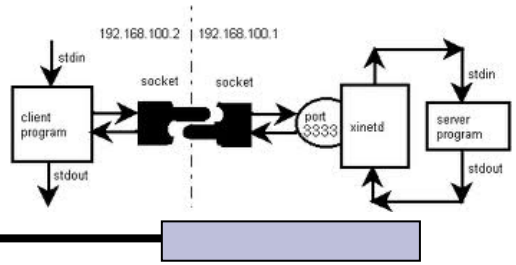
```
#include <netdb.h>
#include <arpa/inet.h>

void print_protocol(struct addrinfo *aip) {
   printf(" protocol ");
   switch (aip->ai_protocol) {
      case 0: printf("default"); break;
      case IPPROTO_TCP: printf("TCP"); break;
      case IPPROTO_UDP: printf("UDP"); break;
      case IPPROTO_RAW: printf("raw"); break;
      default: printf("unknown (%d)", aip->ai_protocol);
   }
}
```

Exemplo de identificação (2)



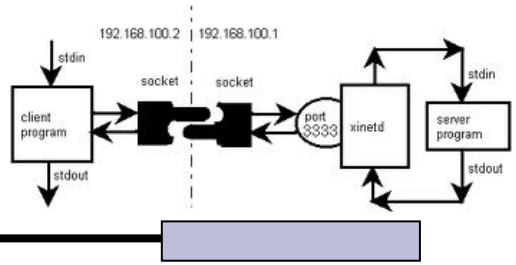

```
int main(int argc, char *argv[]) {
struct addrinfo          *ailist, *aip, hint;
struct sockaddr_in       *sinp;
int                      err;
const char               *addr;
char                     abuf[INET_ADDRSTRLEN];

   if (argc != 3) {
      printf("usage: %s nodename service\n", argv[0]);
      exit(1); }
```

Exemplo de identificação (3)



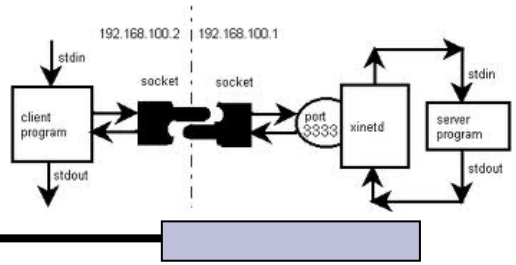

```
hint.ai_flags = AI_CANONNAME;
hint.ai_family=hint.ai_socktype=hint.ai_protocol=hint.ai_addrlen=0;
hint.ai_canonname = NULL; hint.ai_addr = NULL; hint.ai_next = NULL;
if ((err = getaddrinfo(argv[1], argv[2], &hint, &ailist)) != 0) {
   printf("getaddrinfo error: %s", gai_strerror(err));
   exit(1); }
for (aip = ailist; aip != NULL; aip = aip->ai_next) {
   print_protocol(aip);
   printf("\n\thost %s", aip->ai_canonname?aip->ai_canonname:"-");
   if (aip->ai_family == AF_INET) {
      sinp = (struct sockaddr_in *)aip->ai_addr;
      addr = inet_ntop(AF_INET,&sinp->sin_addr,abuf,INET_ADDRSTRLEN);
      printf("; address %s", addr?addr:"unknown");
      printf("; port %d", ntohs(sinp->sin_port)); }
   printf("\n"); }
exit(0); }
```

Exemplo de identificação (4)




```
[rgc@asterix List]$ list lect14.cs.ucdavis.edu ssh
 protocol TCP
   host lect14.cs.ucdavis.edu; address 169.237.5.137; port 22
 protocol UDP
   host -; address 169.237.5.137; port 22
[rgc@asterix List]$
```

Protocolo



- O protocolo de nível transporte adoptado na comunicação é determinado pelo par (domínio,tipo)

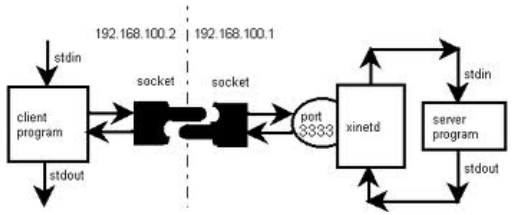
Tipo \ Domínio	AF_UNIX	AF_INET	AF_UNSPEC
SOCK_STREAM	SIM	TCP	SPP
SOCK_DGRAM	SIM	UDP	IDP
SOCK_RAW	-	IP	SIM
SOCK_SEQPACKET	-	-	SPP

Server Information Manager Protocol

Internet Datagram Protocol

Sequenced Packet Protocol

Localização dos socket em Linux



Nível da Internet

Nível do Linux

Utilizador

Cliente ou Servidor

Aplicação

API socket

Socket

Transporte

TCP ou UDP

Núcleo

Rede

IP

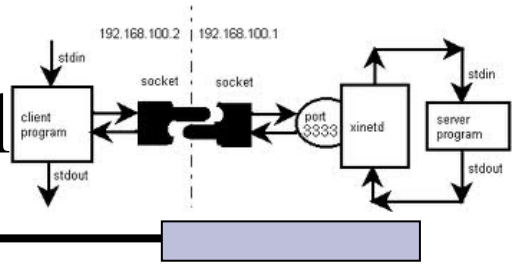
Ligação

Gestor periférico

Controlador periférico

Hardware

Estruturas de dados–panorama global



- A manipulação dos sockets envolve várias estruturas de dados.

Tabela de descritores

0:
1:
2:
⋮

Estrutura `addrinfo`

proto family: PF_INET
service: SOCK_STREAM
Local address:
Remote address:

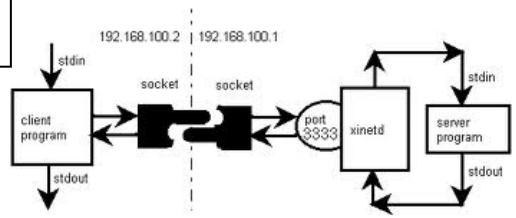
Estrutura `sockaddr`

address family: AF_INET
host IP: 193.136.143.1
port: 80

Implementação *sockaddr_in*

Criação de um socket

APUE 16.2



- Um *socket* é criado em C, pela chamada de sistema

  POSIX:
  ```
  #include <sys/socket.h>
  int socket(int, int, int);
  ```

  – O 1º parâmetro identifica o domínio de comunicação.

  – O 2º parâmetro identifica o tipo de sockect.

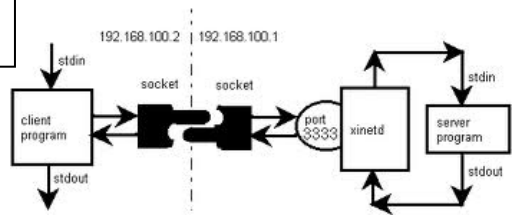
  – O 3º parâmetro identifica o protocolo de transporte (códigos afixados em `/etc/protocols`, normalmente usado 0-protocolo por omissão).

  – O valor retornado é o descritor do socket, ou -1 em caso de erro.

Nota: a função `socket()` não determina de onde os dados veêm os dados, nem para onde eles vão – apenas cria a interface.

APUE 16.3.4

Associação a endereço/porto (1)



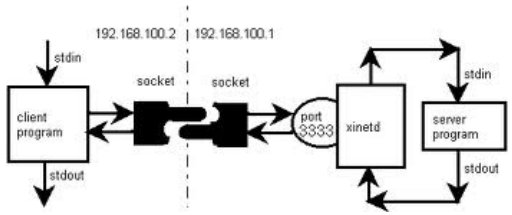
- Um *socket* é associado ao porto, por forma a ficar com com acesso exclusivo, através da função

  POSIX:
  ```
  #include <sys/socket.h>
  int bind(int,
           struct sockaddr*,
           socktlen_t);
  ```

  – O 1º parâmetro identifica o descritor do *socket*.
  – O 2º parâmetro referencia a estrutura contendo o endereço IP.
  – O 3º parâmetro identifica a dimensão da estrutura contendo o endereço IP.
  – O valor retornado é 0(-1) em caso de sucesso (erro).

Nota: ligação a portos até 1024 necessita privilégios de *root*.

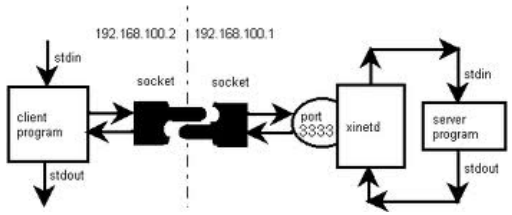
Associação a endereço/porto (2)



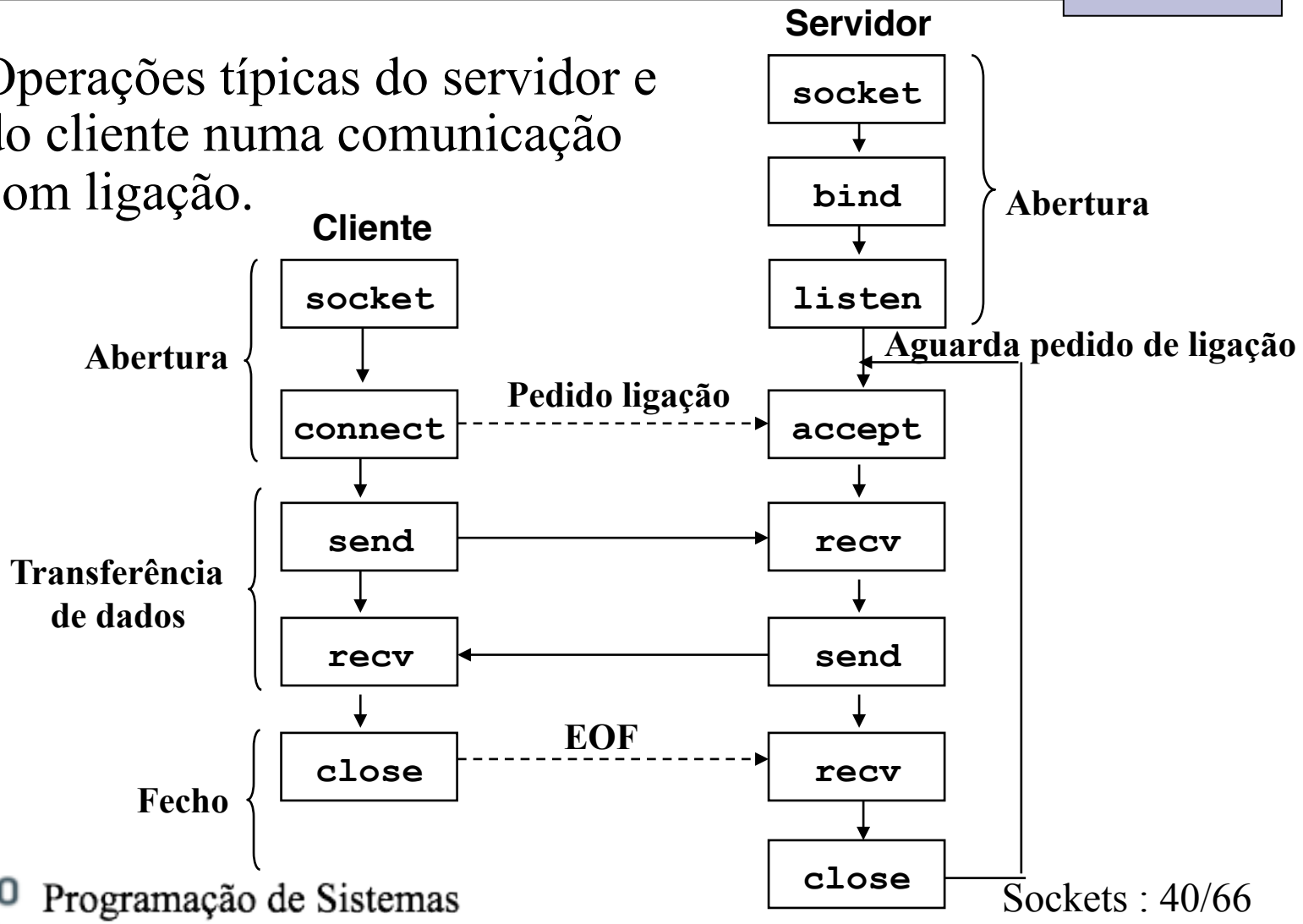
- No domínio da Internet, a constante `INADDR_ANY` determina que o *socket* fica automaticamente associado a todos os endereços Internet do nó local.
  Nota: uma "firewall" possui duas placas de rede, uma ligada à Internet e outra ligada à rede local.

- A associação pode não ser necessária para alguns tipos de sockets
  - SOCK_DGRAM: se apenas forem enviados dados, porque o SO encontra um porto cada vez um pacote é enviado.
  - SOCK_STREAM: o destino é determinado na altura do estabelecimento da ligação.

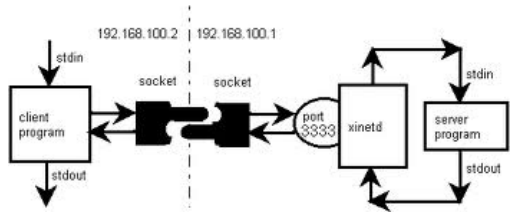
CcL:Comunicação com ligação



- Operações típicas do servidor e do cliente numa comunicação com ligação.



CcL:Estabelecimento da ligação (1)



- Para serviços orientados à ligação (sockets de tipo `SOCK_STREAM` ou `SOCK_SEQPACKET`) é necessário firmar a ligação antes dos dois nós trocarem dados.

A. O cliente deve solicitar do servidor a ligação

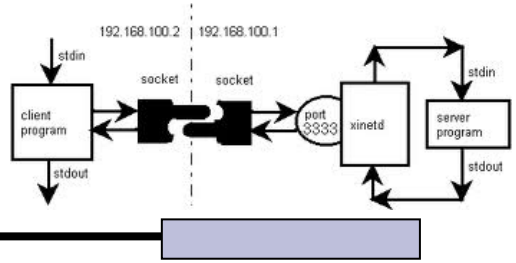
POSIX:
```
#include <sys/socket.h>
int connect(int,
            struct sockaddr*,
            socktlen_t);
```

– O 1º parâmetro identifica o descritor do *socket*.

– O 2º parâmetro identifica o endereço do servidor.

– O 3º parâmetro identifica a dimensão da estrutura contendo o endereço IP.

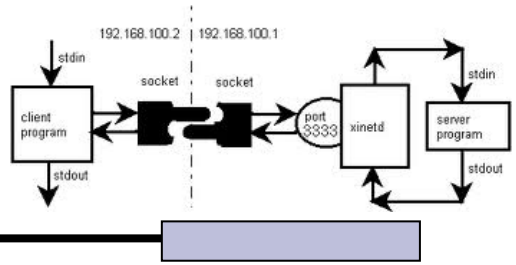
CcL:Estabelecimento da ligação (2)



- O servidor pode encontrar-se indisponível por razões diversas
  - Servidor desligado temporariamente
  - Problemas na rede
  - Sobrecarga de pedidos
- Em caso de insucesso deve haver um intervalo de espera até ser formulado novo pedido de ligação

```
#define MAXSLEEP 128
int nsec;
for( nsec=1; nsec<=MAXSLEEP; nsec<<1 )
   if( connect(socketfd,addr,len)==0 ) {
      /* ligação aceite */
   }
   sleep(nsec);
}
```

CcL:Estabelecimento da ligação (3)



B. O servidor deve disponibilizar-se a acolher pedidos de ligações:

POSIX:
```
#include <sys/socket.h>
int listen(int,int);
```

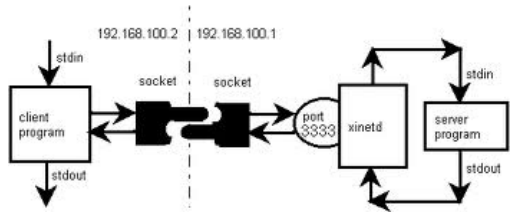
– O 1º parâmetro identifica o descritor do socket do servidor.

– O 2º parâmetro identifica o número máximo de pedidos que podem ficar pendentes numa fila ("backlog").

Nota1 : a função `listen` não bloqueia.

Nota2 : pedidos enviados depois da fila esgotada são rejeitados.

CcL:Estabelecimento da ligação (4)



- Um pedido de ligação é aceite pelo servidor através de

  POSIX:
  ```
  #include <sys/socket.h>
  int accept(int,
             struct sockaddr *,
             socklen_t *);
  ```

  – O 1º parâmetro identifica o descritor do *socket* do servidor.

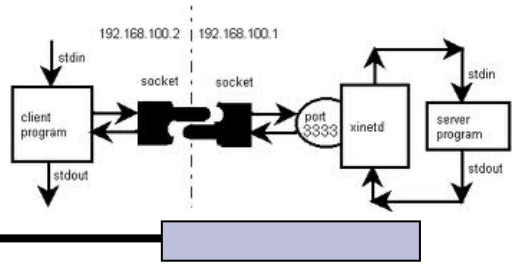
  – O 2º parâmetro é critério de filtragem dos clientes.

  – A função retorna o descritor do **novo socket**, sobre o qual são feitas todas as escritas/leituras na ligação criada.
  O socket inicial fica disponível para receber novos pedidos de ligação.

  Nota1 : a função `accept` bloqueia até à chegada de um pedido de estabelecimento de ligação.

  Nota2: se o servidor não se importar com o cliente, os 2º e 3º parâmetros devem ser `NULL`.

CcL:Leitura e escrita de dados



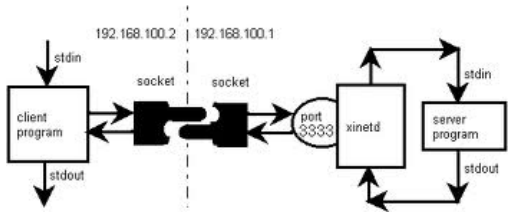
- A comunicação com ligação é feita de forma semelhante à dos ficheiros e dos tubos

  POSIX:
  ```
  #include <sys/types.h>
  #include <sys/socket.h>
  ssize_t send(int, char *, int, int);
  ssize_t recv(int, char *, int, int);
  ```

  – O 1º parâmetro é o descritor de ficheiro.
  – O 2º parâmetro é o endereço dos dados de utilizador
  – O 3º parâmetro é o número de Bytes a comunicar.
  – O 4º parâmetro identifica opções, normalmente 0.
  – A função devolve o número de Bytes efectivamente transferidos. Em caso de erro, o valor devolvido é -1 e a causa do erro é afixada na variável `errno`.

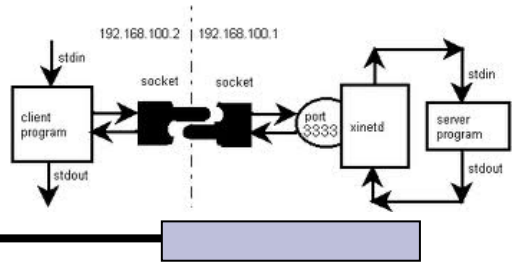
  Nota 1: podem ser usadas as funções `read()` e `write()`
  Nota 2: as funções `send` e `recv` são bloqueantes.

5º EXERCÍCIO TEORICO-PRATICO



- Pretende-se ligar dois nós por *sockets* segundo o modelo cliente-servidor. O protocolo de rede é o TCP.
  - O programa do servidor recebe como parâmetro na linha de comando o porto do *socket*.
    O servidor deve ser lançado primeiro.
  - O programa do cliente recebe comos parâmetros da linha de comando o endereço do nó do servidor e o porto do *socket*.
- O cliente recebe do terminal a mensagem a enviar para o servidor.
  1. O servidor envia a mensagem recebida para o terminal, responde ao cliente e encerra.
  2. O cliente fica à espera da resposta, que é afixada no terminal, e encerra.

5º EXERCÍCIO TEORICO-PRATICO



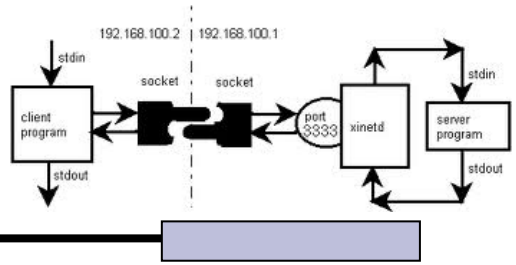
- Sessão no nó servidor

```
[rgc@asterix Cliente-servidor]$ server 26000
Here is the message: Greetings from Canada!

[rgc@asterix Cliente-servidor]$
```

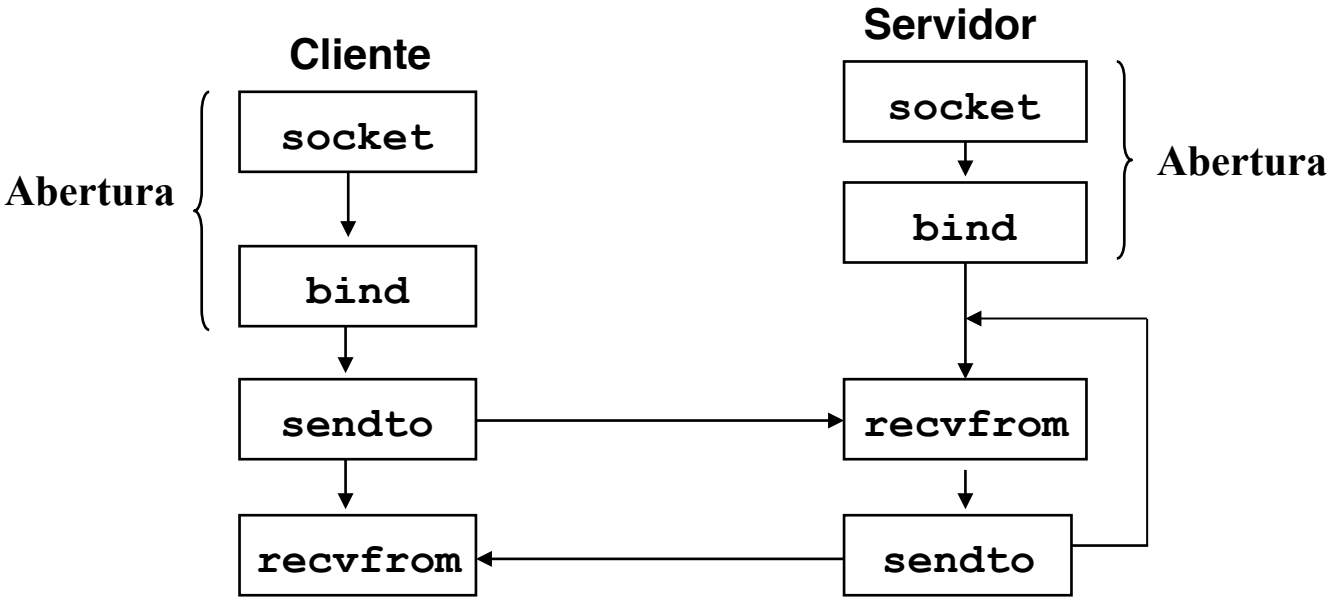
- Sessão no nó cliente

```
[rgc@ottawa ~]$ client 193.136.143.1 26000
Please enter the message: Greetings from Canada!
I got your message
[rgc@ottawa ~]$
```

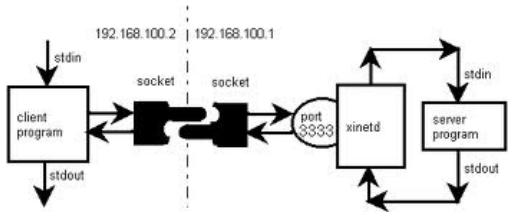
CsL:Comunicação sem ligação



- Operações típicas do servidor e do cliente numa comunicação sem ligação.



CsL:Leitura e escrita de dados (1)



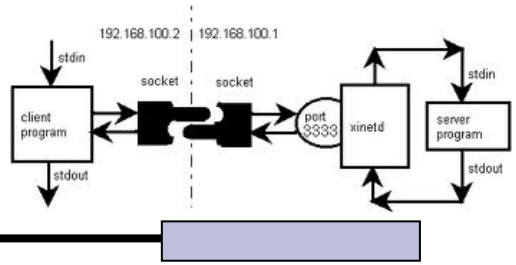
- Na comunicação sem ligação são necessários parâmetros extra

  POSIX:
  ```
  #include <sys/types.h>
  #include <sys/socket.h>
  ssize_t sendto(int, char *, int, int,
                 struct sockaddr *,
                 socklen_t);
  ```

  – Os 4 primeiros parâmetros têm o mesmo significado de `send`.
  – O 5º parâmetro identifica o endereço do destino.

Nota: o UDP só transfere mensagens até 8KB

CsL:Leitura e escrita de dados (2)

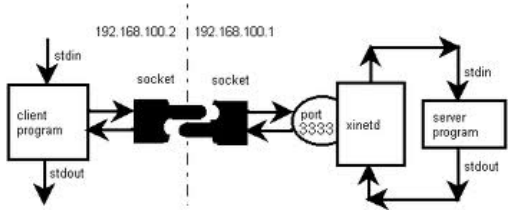


- Na comunicação sem ligação a função `recv` pode ser usada para leitura.
- Se o programador tiver necessidade de conhecer o nó que enviou os dados, usar a função

POSIX:
```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t recvfrom(int, char *, int, int,
                 struct sockaddr *,
                 socklen_t);
```

  - Os 4 primeiros parâmetros têm o mesmo significado de `recv`.
  - O 5º parâmetro identifica o endereço da fonte.

CcL e CsL: Fecho da ligação

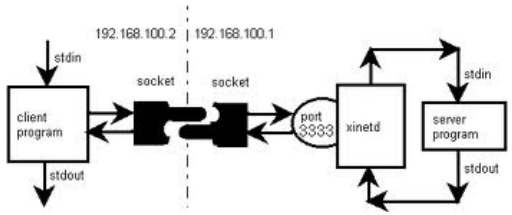


- O servidor e o cliente devem fechar o *socket* assim que terminarem a comunicação

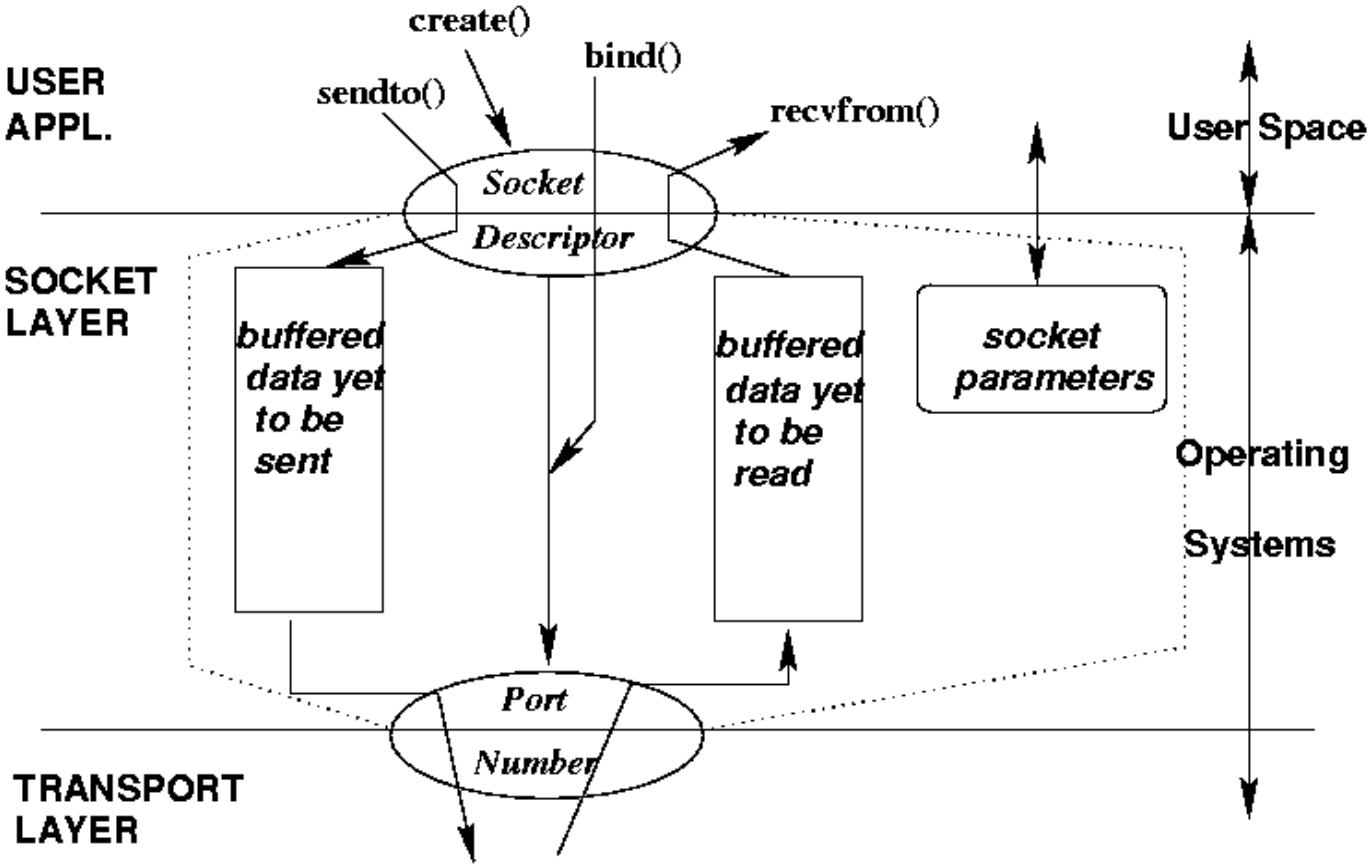
  POSIX:
  ```
  #include <sys/socket.h>
  int close(int);
  ```

  – O parâmetro identifica o descritor do *socket* do servidor.
- A função
  – Fecha a ligação para `SOCK_STREAM`.
  – Liberta o porto usado pelo *socket*.

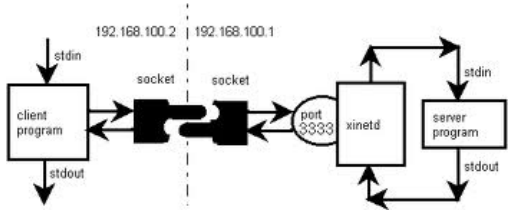
Chamadas – panorama global



- Chamadas mais importantes da API de sockets



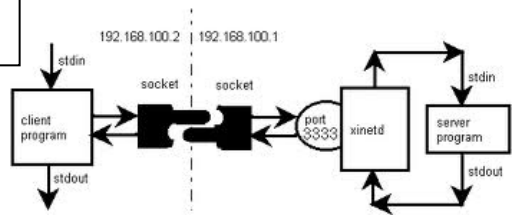
Erros na programação em sockets



- Erros mais frequentes na programação em sockets:
  - Ordenação incorrecta de Bytes, por não inserção de chamadas às funções `hton()` e `ntoh()`.
  - Desacordo sobre dimensão dos registos de dados (fixos ou variáveis).
  - Problemas no `select()`.
  - Não observação do protocolo
    - Problema não abrangido nesta disciplina,
    - A ter em atenção nas disciplinas
      - Redes de Computadores
      - Software de Telecomunicações

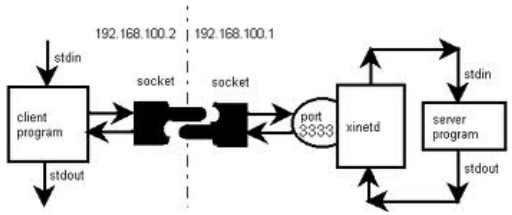
APUE 14.5.1

Chamadas bloqueantes (1)



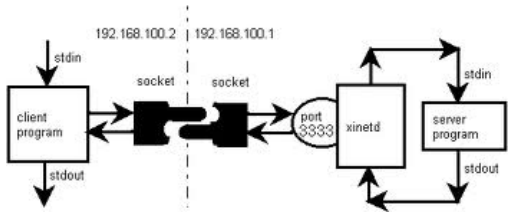
- Muitas das chamadas da API sockets são bloqueantes: quando o processo chama uma função bloqueante, ele fica suspenso à espera do evento.
  - `accept`: à espera que o pedido de ligação chegue.
  - `connect`: à espera que o pedido de ligação seja estabelecida.
  - `recv`, `recvfrom`: à espera que o pacote de dados seja recebido.
  - `send`, `sendto`: à espera que os dados sejam transferidos para a memória tampão ("buffer") do socket.
    Nota: este bloqueio não depende da rede, mas provoca atraso no processo.
- Para aplicações simples, o bloqueio é benéfico (ex: evita consumo de tempo de CPU).

Chamadas bloqueantes (2)



- Para aplicações complexas, o bloqueio é prejudicial:
  - Ligações múltiplas.
  - Transmissões e recepções simultâneas.
  - Processamento simultâneo não relacionado com a rede.
- Várias estratégias podem ser seguidas para tornear o problema do bloqueio:
  - Programação multiprocesso, ou multi-thread (desvantagens: programação mais complexa, problemas de sincronização,...).
  - Desligar o bloqueio, com a função `fcntl` de controlo dos descritores de ficheiros (desvantagem: programação mais complexa de baixo nível).
  - Usar a chamada de sistema `select` (desvantagem: serializa tratamento, resposta prolongada-exemplo com acesso a disco-atrasa o tratamento das mensagens seguintes).

Chamadas bloqueantes (3)

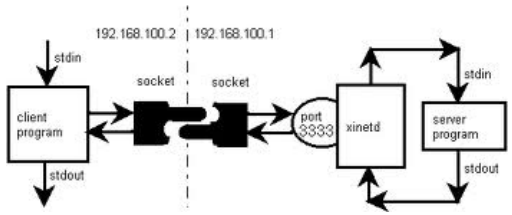


POSIX:
```
#include <sys/select.h>
int select(int,fd_set *,fd_set *,fd_set *,
           struct timeval *);
```

- O 1º parâmetro identifica o código do maior descritor de socket + 1.
- O 2º parâmetro referencia lista de descritores de leitura.
- O 3º parâmetro referencia lista de descritores de escrita.
- O 4º parâmetro referencia lista de descritores para verificar se ocorreu uma excepção.
- O 5º parâmetro determina o intervalo de retorno, caso nada tenha ocorrido (NULL para intervalo infinito).
- A função devolve o número de descriptores afectados por sinais. Em caso de erro, o valor devolvido é -1.

Nota: `select` pode ser usado para descritores de ficheiros.

Chamadas bloqueantes (4)



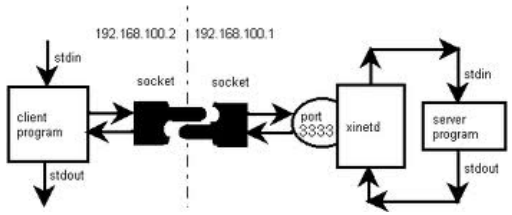
- Os descriptores de sockets são referenciados numa tabela de bits, de tipo `struct fd_set`.
  Para acesso foram criadas várias funções:

```
void FD_ZERO(fd_set *); /* limpa campos da estrutura */
void FD_CLR(int, fd_set *); /* coloca bit a 0 */
void FD_SET(int, fd_set *); /* coloca bit a 1 */
int FD_ISSET(int, fd_set *); /* devolve valor do bit */
```

- O socket de descriptor i está pronto para leitura / escrita / excepção se o bit estiver a 1 nos parâmetros número 2(readfds), 3(writefds) ou 4(exceptfds).

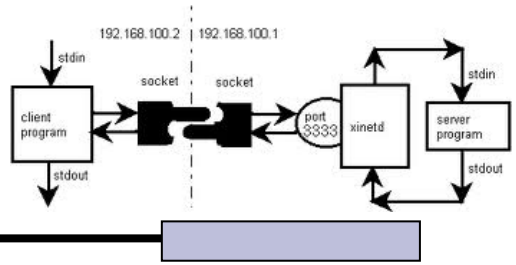
Nota: A configuração deve ser efectuada sempre antes da chamada ao `select`.

Chamadas bloqueantes (5)



- Os selectores apresentam o inconveniente de serializar o tratamento dos pedidos, o que degrada o desempenho do sistema.
- Quem usar selectores no projecto de avaliação arrisca-se a ser
  - fuzilado,
  - queimado em praça pública,
  - depois de aplicada uma enorme lista de maldades terríveis! ☹

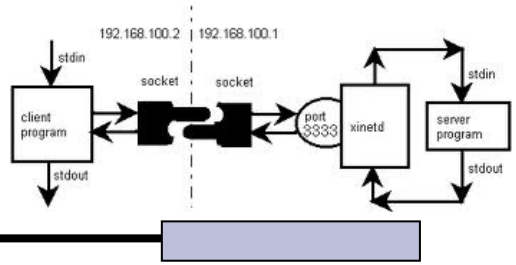
Chamadas bloqueantes (6)



Exemplo

- Pretende-se desenvolver um programa cliente-servidor, com o servidor a enviar para o terminal dados vindos de
  1. Teclado, ou
  2. Rede, com transmissão de protocolo UDP e porto indicado na linha de comando.
- Se, ao fim de 5 segundos nada for recebido, o servidor deve afixar uma mensagem de aviso no terminal.

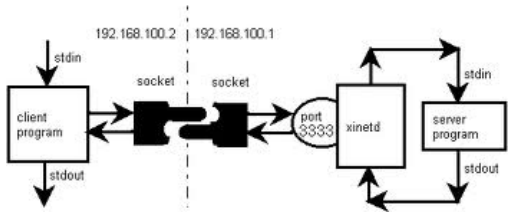
Chamadas bloqueantes (7)



Código do servidor

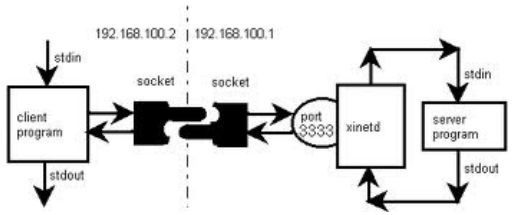
```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <netinet/in.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>

#define WATCH 5
#define LEN 80
void error(char *msg) {
    perror(msg);
    exit(1); }
```

Chamadas bloqueantes (8)



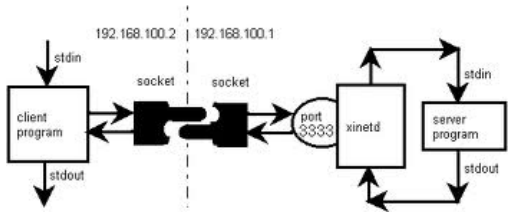

```
int main(int argc, char **argv) {
   int sockfd;
   struct sockaddr_in serv_addr;
   struct sockaddr *cli_addr;
   socklen_t cli_len;

   fd_set rfds;
   struct timeval tv;
   int retval;

   char buf[LEN];
   int size;

   /* open socket for Intenet communication */
   if (argc < 2) {
      fprintf(stderr, "ERROR, no port provided!\n");
      exit(1); }
   sockfd=socket(AF_INET, SOCK_DGRAM, 0);
   if (sockfd<0) error("ERROR opening socket");
```

Chamadas bloqueantes (9)



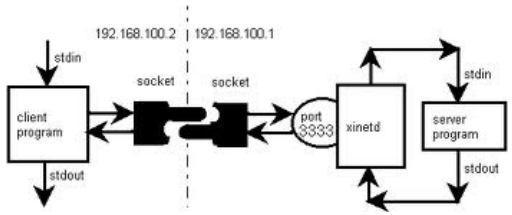

```
/* binds socket to port */
bzero((char *) &serv_addr, sizeof(serv_addr));
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons( atoi(argv[1]) );
if (bind(sockfd, (struct sockaddr *) &serv_addr,
    sizeof(serv_addr)) < 0) error("ERROR on binding");

for(;;) {
   /* configure select parameters */
   tv.tv_sec = WATCH; tv.tv_usec = 0;
   FD_ZERO(&rfds);
   FD_SET(0, &rfds);          /* stdin is fd[0] */
   FD_SET(sockfd, &rfds);
```

Chamadas bloqueantes (10)




```
 retval = select(sockfd+1, &rfds, NULL, NULL, &tv);

     if (retval == -1) error("ERROR on select");
     else if (retval) {
        if (FD_ISSET(0, &rfds)) {
            size = read(0, &buf, LEN);
            buf[--size] = '\0'; /* replace \n by string delimiter */
            printf("  *** Data from keyboard:%s ***\n",buf); }
         if (FD_ISSET(sockfd, &rfds)) {
            size = recvfrom(sockfd, &buf, LEN, 0, cli_addr, &cli_len);
            printf("  *** Data from abroad:%s ***\n",buf); }
         }
     else printf("No data within %d seconds.\n",WATCH);
   }
   exit(EXIT_SUCCESS);
}
```

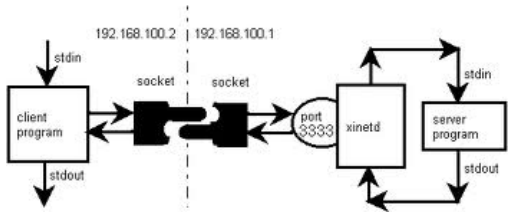
Chamadas bloqueantes (11)



Código do cliente

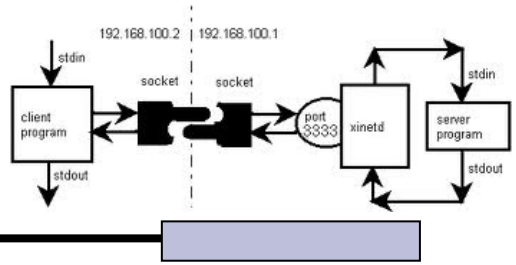
```
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <netinet/in.h>
#define LEN 80

void error(char *msg) {
    perror(msg);
    exit(1); }

int main(int argc, char **argv) {
   int sockfd;
   struct sockaddr_in serv_addr;

   char buf[LEN];
   int len,nbytes;
```

Chamadas bloqueantes (12)




```
/* open socket for Intenet communication */
if (argc < 2) {
     fprintf(stderr, "ERROR, no port provided!\n");
     exit(1); }
  sockfd=socket(AF_INET, SOCK_DGRAM, 0);
  if (sockfd<0) error("ERROR opening socket");

 serv_addr.sin_family = AF_INET;
 serv_addr.sin_addr.s_addr = inet_addr("193.136.143.1");;
 serv_addr.sin_port = htons( atoi(argv[1]) );

 for(;;) {
    printf("Message to be sent\n");
    len=read(0, &buf, LEN);
    buf[len-1] = '\0';
    nbytes = sendto( sockfd, buf, len, 0,
          (struct sockaddr*)&serv_addr, sizeof(serv_addr));
    if(nbytes<0) error("ERROR on sendto"); }
}
```

Chamadas bloqueantes (13)




```
[rgc@asterix Select]$ server 20000
Hello all!
  *** Data from keyboard:Hello all! ***
No data within 5 seconds.
  *** Data from abroad:Hello from Canada! ***
No data within 5 seconds.
Bye!
  *** Data from keyboard:Bye! ***

[rgc@asterix Select]$
```

```
[rgc@ottawa Select]$ client 20000
Message to be sent
Hello from Canada!
Message to be sent

[rgc@ottawa Select]$
```