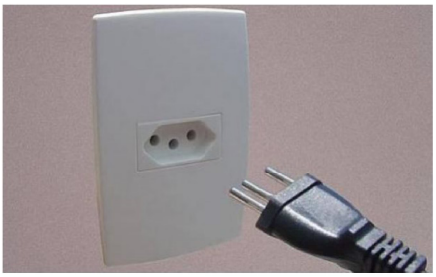
Redes de Computadores

Sockets

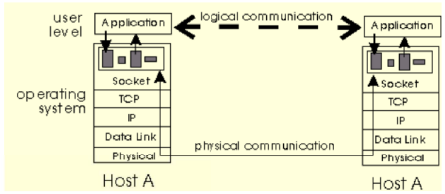
Interface: *sockets*

- Surgiu originalmente no sistema operacional *Unix BSD (Berkeley Software Distribution)*
- São programas responsáveis pela interação/comunicação de programas ao longo da internet
- É a interface padrão para comunicação entre processos em redes TCP/IP
- Programar com sockets pode ser visto como desenvolver um protocolo de aplicação

Interface: *sockets*



Interface: *sockets*



Interface: *sockets* (com conexão)

•Para que dois processos se comuniquem, eles devem estabelecer uma conexão – mas como?

•Parte passiva da abertura da conexão:

°um processo se diz pronto a receber conexões

°não há necessariamente identificação do outro

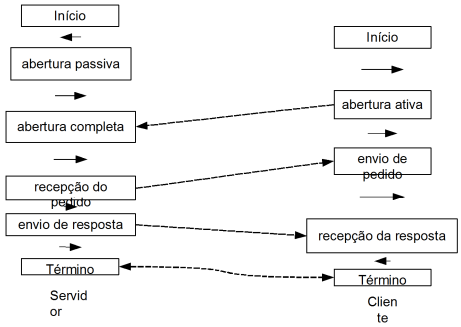
•Parte ativa da abertura da conexão:

°processo sabe como alcançar seu interlocutor

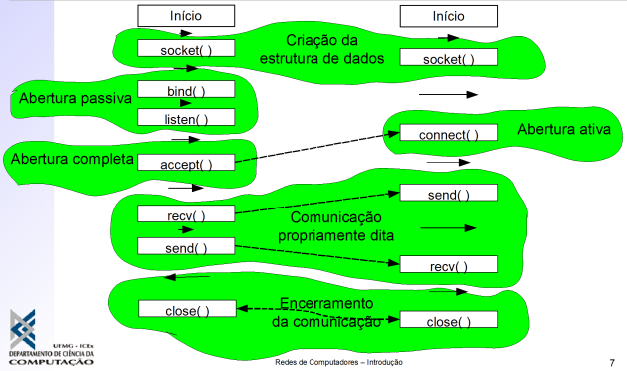
°esse interlocutor já está pronto para receber

•Fechamento da conexão (explícito ou implícito)

Comunicação entre processos



Interface: *sockets*



Interface: *sockets*

- Criação de um socket

AF_INET, AF_UNIX, AF_OSI (ou PF_*)

0 (*default*)

```
int socket(int domain, int type, int protocol)
```

SOCK_STREAM, SOCK_DGRAM, outros

Interface: *sockets*

- Abertura passiva

```
int bind(int sckt, struct sockaddr *addr, int addr_len);
int listen(int sckt, int backlog);
int accept(int sckt, struct sockaddr *addr, int addr_len);
```

- Abertura ativa

```
int connect(int sckt, struct sockaddr *addr, int addr_len);
```

Representação de endereços

- Fechamento:

```
int close(int socket);
```


Interface: *sockets*

- Abertura passiva

```
int bind(int socket, struct sockaddr *address, int addr_len);
```

- Atribui um endereço IP e uma porta a um socket
- *int* socket*: é o socket criado pela função socket()*
- struct sockaddr *address*: é a estrutura de* endereçamento que contém as informações necessárias para o estabelecimento da associação
- *int* addr_len*: é o tamanho dessa estrutura, pois,* dependendo da família e do protocolo utilizados, ele varia

Interface: *sockets*

- struct sockaddr: endereço em cada "família"

```
struct sockaddr_in {
    sa_family_t         sin_family; /* AF_INET */
    unsigned short int  sin_port;
    struct in_addr      sin_addr;
};
```

- O primeiro item define o tipo de família do protocolo a ser usado
- O segundo define o número da porta TCP ou UDP a ser usada
- O terceiro item é o endereço IP do host destino

Interface: *sockets*

- Operações especiais:
  - Como converter um string de/para um endereço?

```
struct in_addr inet_addr(char* addr);
char* inet_ntoa(struct in_addr inaddr);
```

  - Como converter inteiros da representação da máquina para a representação da rede?

```
htons( ), htonl( ), ntohs( ), ntohl( )
```

Interface: *sockets*

- Operações especiais:
  - Como obter a identificação de quem se conectou?

```
getpeername(sock, &saddr_in,
sizeof(saddr_in));
short int port = ntohs(saddr_in.sin_port);
struct in_addr addr = s_addr_in.sin_addr;
```


Interface: *sockets*

- Operações especiais:
  - Como obter o nome da minha máquina?

```
int gethostname(char* ret_name, int namelen);
```

  - Como obter um endereço a partir do nome da máquina?

```
struct hostent* gethostbyname(char*
hostname);
```


Interface: *sockets*

- Envio e recebimento de mensagens

```
int send(int socket, char *msg, int mlen, int
  flags /* 0 */ );

int recv(int socket, char *buf, int blen, int flags /*
  MSG_WAITALL */);
```

- Também pode-se usar write() / read(), mas eu não recomendo...

Interface: *sockets*

- Melhor prevenir que remediar: teste o valor de retorno!

```
if ( (code=syscall()) < 0 ) {
  perror("syscall"); exit(1);
}
...
if ( (ptr=ptr_returning_function() ) == NULL ) {
  perror("ptr_returning_function"); exit(1);
}
```


Interface: *sockets*

- Bibliotecas:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
```

- Compilando:

```
gcc my_socket_program.c -o my_socket_program
```

- Alguns ambientes requerem lincar explicitamente:

```
cc my_socket_program -o my_socket_program -lsocket
  -lnsl
```


Exemplo: cliente

```
#include ...
#include <sys/socket.h>

int main(int argc, char **argv)
{
    int s;
    struct sockaddr_in dest;
    char msg_write[100], msg_read[100];
    s = socket(AF_INET, SOCK_STREAM, 0));

    bzero(&dest, sizeof(dest));
    dest.sin_family = AF_INET;
    dest.sin_port = htons(9999);
    inet_aton("127.0.0.1", &dest.sin_addr.s_addr);

    connect(s, (struct sockaddr*)&dest, sizeof(dest));

    do {
        scanf("%s",msg_write);
        write (s, msg_write, strlen(msg_write)+1);
        read (s, msg_read, MAXBUF);
    } while (strcmp(msg_read,"bye"));

   close(s);
}
```

Exemplo: servidor

```
#include ...
#include <sys/socket.h>

int main(int argc, char **argv)
{   int s, client_s;
    struct sockaddr_in self, client;
    int addrlen = sizeof(client);
    char msg_write[100], msg_read[100];

    s = socket(AF_INET, SOCK_STREAM, 0);

    bzero(&self, sizeof(self));
    self.sin_family = AF_INET;
    self.sin_port = htons(9999);
    self.sin_addr.s_addr = INADDR_ANY;

    bind(s, (struct sockaddr*)&self, sizeof(self));
    listen(s, 5);

    while (1) {
        client_s = accept(s, (struct sockaddr*)&client, &addrlen);

        do {
            read (client_s, msg_read, MAXBUF);
            write (client_s, msg_read, strlen(msg_read)+1);
            } while (strcmp(msg_read,"bye"));
            close(client_s);
    }
}
```


Medições de desempenho

- Planejamento de experimentos
  - O que se deseja medir?
  - Como será feita a medição?
  - Quais as variáveis que se pretende avaliar?
  - Que fatores externos podem afetar os resultados?

Medições de desempenho

- Medição de tempo
  - Grão grosso: time (comando da shell)
  - Grão fino: gettimeofday()

```
int gettimeofday(struct timeval *tp, NULL);
struct timeval {
  time_t tv_sec;
  long   tv_usec;
};
```

  - Grão muuuito fino:

```
gettimeofday(&antes,NULL);
for (i=0;i<MUITAS_VEZES;++i) { ... }
gettimeofday(&depois,NULL);
```


Medições de desempenho

- Medição de banda:

```
char ack, dados[N_BUFFERS][TAM_BUF];

gettimeofday(&antes,NULL);

for (i=0;i<N_BUFFERS;++i) {
        send( sd, dados[i], TAM_BUF, 0 );
}
recv( sd, &ack, sizeof(ack), MSG_WAITALL );

gettimeofday(&depois,NULL);
```


Medições de desempenho

- Medição de latência:

```
char dados[TAM_BUF];

gettimeofday(&antes,NULL);

for (i=0;i<N_VEZES;++i) {
        send( sd, dados, sizeof(dados), 0 );
        recv( sd, dados, sizeof(dados), MSG_WAITALL );
}
gettimeofday(&depois,NULL);
```

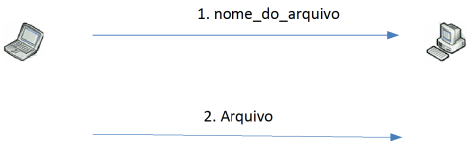

Medições de desempenho

- Sempre indicar o ambiente
  - Tipo de máquina, SO e rede
  - Configurações especiais de kernel, rede, etc.
- Considerar "aquecimento"
  - Efeitos devidos a caches de dados e instruções
  - Comportamento especial no início de certas conexões
- Considerar variabilidade do ambiente
  - Não basta experimentar; tem que repetir e repetir e repetir...
  - Indicar variância, ou intervalo de confiança, ou max/min, etc.

•Descrição Mini FTP

- Objetivo: Aprender a programação com sockets.
- Implementar um cliente e um servidor ftp.
- Cliente envia:
  - nome_do_arquivo
- Servidor abre o arquivo e o envia

•Mini FTP



•O que se pede

- Código do cliente
- Código do servidor
- Makefile
- Breve manual de usuário e decisões de projeto

•O que deve ser entregue

- Código do cliente e do servidor
- Um arquivo zip com todo o seu sistema e um arquivo **`leiame`** com informações sobre a compilação desse sistema.
- Um arquivo no formato pdf sobre o seu sistema incluindo um breve manual de usuário e decisões de projeto.

•O que deve ser entregue

- Data de entrega: 27/04/2011 até às 23:59 para o endereço eletrônico **`submission.vieira@gmail.com`**, com assunto [REDES-TP1] “seu nome”