Apoio à Programação Distribuída

- bibliotecas
- bibliotecas+ferramentas
- linguagens de programação distribuídas

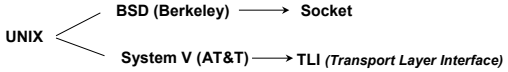
• flexibilidade de programação
• disponibilidade da ferramenta
• facilidade de desenvolvimento e reuso

Bibliotecas

- Uso de uma linguagem de programação tradicional com chamadas a uma biblioteca de programação distribuída.
- Exemplos:
  - soquetes (originalmente Unix): programação baseada no modelo cliente-servidor;
  - pvm, p4, Express >> mpi: programação baseada na comunicação entre processos pares.
- Dados são transmitidos através de streams de bytes ou buffers - interpretação correta fica por conta do programa.

Socket

- Forma de Comunicação entre Processos disponível no UNIX BSD



- Usado para um processo comunicar-se com outro que esteja em uma máquina qualquer
- Idéia similar a de descritor de arquivos
- O Socket é identificado por um inteiro chamado *descritor do socket*

descritor_do_socket = socket()

Socket

- Socket:
  - Espera uma conexão

    Inicia uma conexão
- Socket Passivo: espera por uma conexão (usado por Servidores)
- Socket Ativo: Inicia uma conexão (usado pelos Clientes)
- Complexidade do Socket: parâmetros que o programador pode setar

Socket

- Exemplo de alguns parâmetros:
  - Transferência de:
    - stream (TCP)
    - datagrama (UDP)
  - endereço remoto:
    - específico (geralmente usado pelo cliente)
    - inespecífico (geralmente usado pelo servidor)

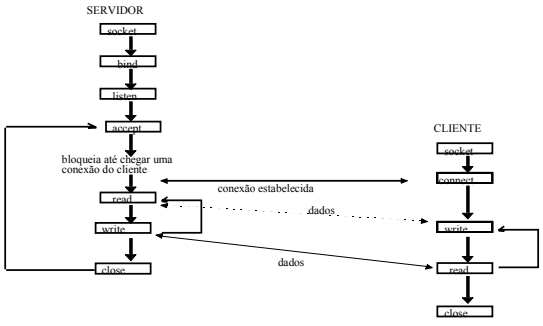
TCP/IP

- Protocolo de Comunicação
- Não estabelece como o Sistema Operacional (SO) vai se comunicar com ele.
- A interface do SO é responsável pela comunicação com o TCP/IP
- O UNIX faz isso usando socket.
- TCP/IP estabelece uma abstração chamada Porta.

Chamadas usadas no Socket

- socket( ) -> cria um socket usado para comunicação e retorna um descritor
- connect( ) -> depois de criar um socket um cliente chama connect para estabelecer uma conexão com um servidor, usando o descritor do socket
- write( ) -> para enviar dados através de uma conexão TCP
- read( ) -> para receber dados através de uma conexão TCP
- close( ) -> para desalocar o socket.
- bind( ) -> usado por servidores para especificar uma porta na qual ele irá esperar conexões.
- listen( ) -> servidores chamam o listen para colocar o socket no modo passivo e torná-lo disponível para aceitar conexões
- accept( ) -> depois que um servidor chama *socket* para criar um socket, *bind* para especificar seu endereço e *listen* para colocá-lo no modo passivo, ele deve chamar o *accept* para pegar a primeira solicitação de conexão na fila.

Exemplo do Uso de Socket



Definições de Constantes

```
#define SERV_TCP_PORT  6543
#define SERV_HOST_ADDR “192.43.235.6
```

Código do Servidor usando o Protocolo TCP/IP

```
main(int argc, char *argv[])
{   int                 sockfd, newsockfd, tam_cli, filho_pid;
    struct sockaddr_in  cli_addr, serv_addr;
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)   /* abre um socket TCP */
        erro;
    /* Liga o processo servidor ao seu endereço local */
    bzero((char *)&serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(SERV_TCP_PORT);
    if  (bind(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)  erro;
    listen(sockfd, 5);
    for (;;) {
        newsockfd = accept(sockfd, (struct sockaddr *)&cli_addr, &tam_cli);
        if (newsockfd <0)   erro;
        if (( filho_pid = fork()) < 0)  erro;
        else if (filho_pid == 0)   {        /* processo filho */
                close(sockfd);          /* fecha o socket original */
                str_echo(newsockfd); /* processa a solicitação */
                exit();
        }
        close(newsockfd);           /* processo pai */
    }
}
```

Código do Cliente usando o Protocolo TCP/IP

```
main(int argc, char *argv[])
{   int                 sockfd;
    struct sockaddr_in  serv_addr;
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)   /* abre um socket TCP */
        erro;
    /* Preenche a estrutura “serv_addr” com o endereço do servidor com o qual ele deseja se conectar*/
    bzero((char *)&serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR);
    serv_addr.sin_port = htons(SERV_TCP_PORT);
    if (connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)  erro;
    str_cli(stdin, sockfd);    /* envia a solicitação */
    close(sockfd);
    exit();
    }
}
```

Rotina Auxiliar do Servidor de Echo

```
/* Rotina que lê um stream de uma linha e envia a linha de volta ao cliente */
#define MAXLINE 512
str_echo(int sockfd)
{
    int n;
    char line[MAXLINE];
    for (;;) {
        n = readline(sockfd, line, MAXLINE);
        if (n ==0 ) return;  /* termina a conexão */
        else if (n < 0) erro;
        if (writen(sockfd, line, n) != n) erro;
    }
}
```

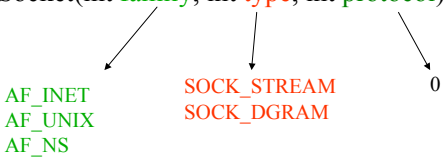
Rotina Auxiliar do Cliente

```
/* Rotina que lê o conteúdo do arquivo FILE *fp, escreve cada linha no socket,  lê uma linha
     de volta do socket, e escreve esta linha na saída padrão */

#define MAXLINE 512
str_cli(FILE *fp, int sockfd)
{
     int n;
     char sendline[MAXLINE], recvline[MAXLINE + 1];
     While (fgets(sendline, MAXLINE, fp) != NULL) {
          n = strlen(sendline);
          if (writen(sockfd, sendline, n) != n) erro;
          n = readline(sockfd, recvline, MAXLINE); /* lê a linha enviada pelo servidor */
          if ((n < 0) erro;
          fputs(recvline, stdout);   /* imprime a linha na saída padrão */
          if (writen(sockfd, line, n) != n) erro;
     }
}
```

Chamadas do Socket

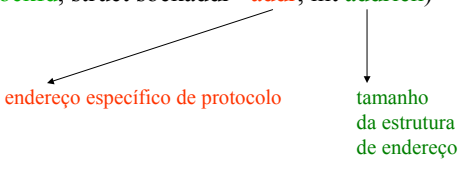
- Socket(int family, int type, int protocol)



sockfd = socket(AF_INET, SOCK_STREAM, 0)

Chamadas do Socket

- bind(int sockfd, struct sockaddr *addr, int addrlen)



sockfd = socket(AF_INET, SOCK_STREAM, 0)

Usos do Bind

- Servidor registrar seu endereço
- Cliente registrar seu endereço
- Cliente usando protocolo sem conexão solicitar ao sistema a atribuição de um endereço

Endereçamento

- O socket oferece uma estrutura diferente para cada protocolo onde o programador especifica os detalhes de endereçamento
  - Para o protocolo TCP/IP a estrutura *sockaddr_in* especifica o formato de endereço

```
struct sockaddr_in {
    u_short sin_family;  /* tipo de endereço */
    u_short sin_port;    /* número da porta */
    u_long sin_addr;     /* endereço IP */
}
```

Definindo o endereço de um servidor

```
struct sockaddr_in {
    u_short sin_family;  /* tipo de endereço */
    u_short sin_port;    /* número da porta */
    u_long sin_addr;     /* endereço IP */
}
```

```
struct sockaddr_in  serv_addr;
bzero((char *)&serv_addr, sizeof(serv_addr));
serv_addr.sin_family =  AF_INET;
serv_addr.sin_addr.s_addr = htonl (INADDR_ANY)
serv_addr.sin_port = htons("6644")
```

Portas

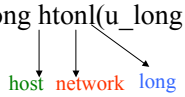
- Formas de Processos serem associados a uma porta:
  - o processo solicita uma porta específica (tipicamente usado por servidores)
  - o sistema automaticamente atribui uma porta para o processo (tipicamente usado por clientes)
    - o processo especifica porta = 0 antes de fazer a chamada ao bind. O bind atribui uma porta automaticamente

TCP/IP - Tabela de Portas

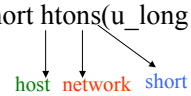
Portas reservadas	1-1023
Portas automaticamente atribuídas pelo sistema	1024 - 5000

Funções para conversão para formato padrão

- u_long htonl(u_long hostlong)



- u_short htons(u_long hostshort)



Operações sobre Bytes

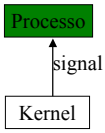
- Não são manipuladas por operações padrão sobre strings pois:
  - Não terminam com nulo
  - Podem ter bytes nulos dentro deles
- bcopy(char *src, char *dest, int nbytes)
  - Move o *nbytes* da origem (src) para o destino
- bzero(char *dest, int nbytes)
  - zera nbytes do destino
- int bcmp(char *ptr1, char *ptr2, int nbytes)
  - Compara dois strings. Retorna 0 se os dois são idênticos.

INADDR_ANY

- Para máquinas com duas interfaces de rede, usa-se o INADDR_ANY para estabelecer que o servidor está “escutando” nas duas redes

I/O Assíncrono

- Processo informa ao kernel para notificá-lo quando um determinado descritor está pronto para I/O.



I/O Assíncrono

- Processo usa a função ***signal*** para estabelecer um handler para o sinal SIGIO
- Processo usa a função ***fcntl*** para setar o ID do processo para receber sinais SIGIO.
- Processo habilita o I/O assíncrono usando a chamada ***fcntl***

Desvantagem

- O processo pode ter somente um handler para um dado sinal
- Se o I/O assíncrono é habilitado para mais que um descritor, quando o sinal chega, o processo não sabe qual descritor está pronto para I/O

Select

- Chamada do sistema que permite que um processo instrua o kernel para observar múltiplos eventos e "acorda-lo" quando um dos eventos acontecer
- Útil para o kernel ser instruído para notificar o processo apenas quando dados estão disponíveis em um dos sockets, arquivos, ... usados pelo processo.

Chamada Select

```
int select(int maxfdpl, fd_set *readfds, fd_set
   *writefds, fd_set *exceptfds, struct timeval
   *timeout);
```

maxfdpl número do maior descritor a ser testado + 1

readfds, writefds, exceptfds descritores de arquivos observados (pronto para leitura, pronto para escrita e de exceções)

Timeout: - quando diferente de **NULL** e de **zero**, aponta um intervalo de tempo fixo para a chamada ficar monitorando os descritores.

- igual a **NULL**, o select permanece indefinidamente monitorando os descritores

- igual a **zero**, a chamada retorna imediatamente após verificar os descritores