Paradigma Cliente/Servidor

Mário Meireles Teixeira. UFMA-DEINF

Comunicação em Sistemas Distribuídos

- Os processos em um SD estão lógica e fisicamente separados. Precisam se comunicar para que possam interagir
- O desempenho de um SD depende criticamente do desempenho do seu subsistema de comunicação
- O paradigma de comunicação mais comumente usado para a interação entre processos é o *Paradigma Cliente-Servidor*

Conceitos: Paradigma C/S

- **Cliente**
  - Necessita do acesso aos recursos administrados pelos servidores para realizar suas tarefas. Dele partem as solicitações de serviço
  - Elemento *pró-ativo*, consumidor de serviços
- **Servidor**
  - Responsável por gerenciar um determinado tipo de recurso do sistema, administrando o acesso concorrente dos clientes ao mesmo
  - Elemento *reativo*, fornecedor de serviços

Conceitos: Paradigma C/S

- Todo recurso compartilhado em um SD (HW, SW ou dados) está sob a guarda de um processo servidor
- A relação cliente/servidor se estabelece entre cada interação entre processos, sendo um conceito de software, não de hardware
  - Clientes e Servidores podem executar na mesma máquina
  - Um servidor pode ser cliente de outro processo
- Servidor x Serviço
- Cliente x Usuário

Conceitos: Paradigma C/S

- Quem são os clientes ?
  - Pode ser qualquer computador (*workstation*) conectado ao sistema através da rede
- Quem são os servidores ?
  - Geralmente computadores de grande capacidade, que oferecem aos clientes recursos como discos, impressoras, bancos de dados, conexões com outras redes, serviços da Web...

Visão de Hardware

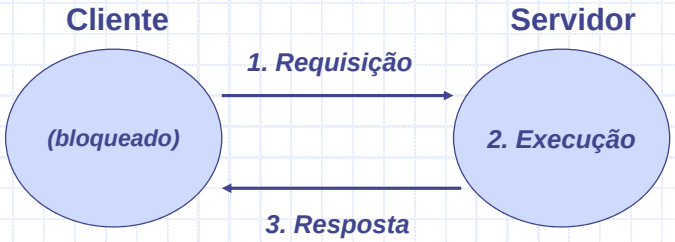
Vantagens e Desvantagens

- Vantagens:
  - Compartilhamento de recursos
  - Balanceamento de carga
  - Tolerância a falhas
  - Escalabilidade
  - Transparência
  - Autonomia e Flexibilidade
  - Capacidade de processamento local e remota
  - Filosofia de *Sistemas Abertos*
  - Multiplataforma
  - Custos menores...

Vantagens e Desvantagens

- Desvantagens:
  - Administração do sistema mais complexa
  - Variados pontos de falha no sistema
  - Dificuldades na interação entre componentes de fornecedores diferentes
  - Novo (?) paradigma de desenvolvimento de software

Comunicação no Modelo C/S



Comunicação no Paradigma C/S

- No Paradigma Cliente-Servidor, a comunicação objetiva principalmente a realização de serviços:
  - Cliente envia requisição ao servidor
  - Servidor recebe a mensagem e processa a solicitação
  - Servidor retorna os resultados ao cliente
- A comunicação entre as partes é implementada usando-se *passagem de mensagens*
- Em nível de programação, normalmente utilizam-se abstrações como *sockets*, *RPCs*, *objetos distribuídos*, *serviços web*

Comunicação no Paradigma C/S

- Um processo servidor pode ter vários clientes e não precisa ter um conhecimento prévio a respeito deles
- Os clientes, antes de contactar um servidor pela primeira vez, geralmente consultam algum Serviço de Nomes (*binder, port mapper*) existente no sistema
- Cliente ou Servidor são papéis que os processos assumem durante uma interação em particular

Passagem de Mensagens

- Os processos comunicam-se através do envio de mensagens, utilizando primitivas do tipo *send/receive*
- Solução de "baixo nível"
  - O programador tem que se preocupar com o envio de mensagens, sincronização, erros de transmissão, mensagens perdidas, timeouts, detalhes de protocolos...
  - Porém, oferece melhor desempenho
- Exemplos de interfaces de passagem de mensagens:
  - *Sockets (TCP/IP)*, IPX/SPX (Netware), TLI, NetBIOS (IBM e Microsoft)

Passagem de Mensagens

- Primitivas de Comunicação:
  - Send (msg, dest)
  - Receive (msg, orig)
- **Interação Síncrona**
  - No Send síncrono, o processo emissor fica bloqueado até que ocorra o Receive
  - O Receive síncrono faz com que o receptor fique bloqueado até a chegada de uma mensagem
  - Neste caso, diz-se que o Send e o Receive são *bloqueantes*

Passagem de Mensagens

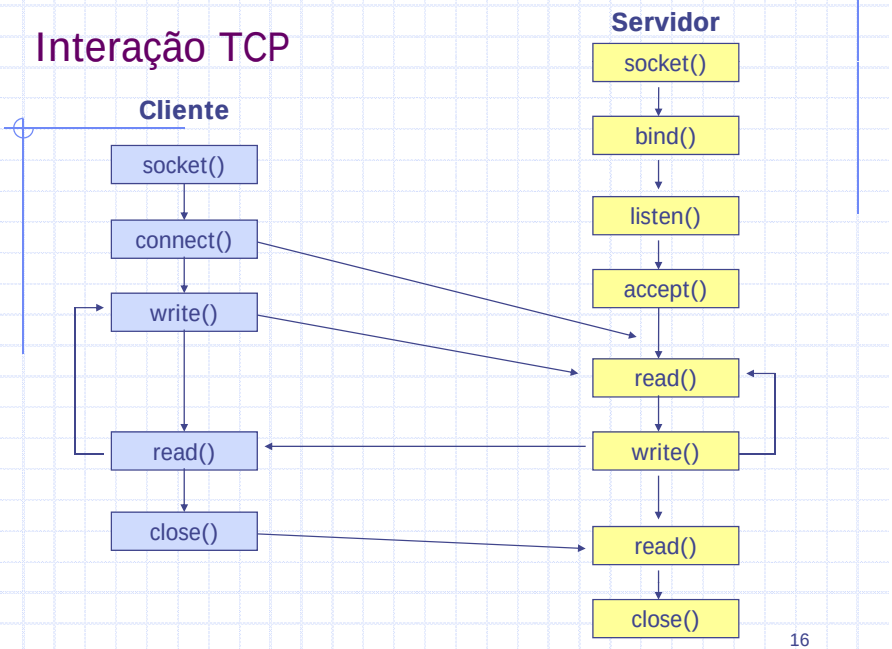
- **Interação Assíncrona**
  - Após o Send, o processo emissor está liberado tão logo a mensagem tenha sido copiada para um buffer local
  - O programa receptor apenas informa sua intenção de receber uma mensagem, alocando um buffer para recepção
    - O receptor fica sabendo da chegada de uma mensagem através de *polling* ou interrupções
  - Diz-se que o Send e o Receive são *não-bloqueantes*
  - A comunicação assíncrona oferece melhor desempenho, mas sua programação é mais difícil que a forma síncrona
  - As primitivas também podem ser *confiáveis* e *não-confiáveis*

Programação com Sockets

Principais Funções – Sockets

SOCKET	Cria um ponto de comunicação (similar a um descritor de arquivo)
BIND	Especifica o endereço local do socket
LISTEN	Especifica o número de requisições pendentes (tamanho da fila)
ACCEPT	Bloqueia-se aguardando um pedido de conexão
CONNECT	Solicita uma conexão
WRITE/SEND/ SENDTO	Envia dados
READ/RECV/ RECVFROM	Recebe dados
CLOSE	Fecha a conexão/socket

Interação TCP



Programa Cliente TCP

```
#define SERVER_PORT 12345
#define BUF_SIZE 4096

int main(int argc, char **argv)
{
  int c, s, bytes;
  char buf[BUF_SIZE];
  struct hostent *h;
  struct sockaddr_in channel;

  h = gethostbyname(argv[1]);
  if (!h) fatal("gethostbyname failed");

  s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
  if (s < 0) fatal("socket");
  memset(&channel, 0, sizeof(channel));
  channel.sin_family= AF_INET;
  memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
  channel.sin_port= htons(SERVER_PORT);
  c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
  if (c < 0) fatal("connect failed");

  write(s, argv[2], strlen(argv[2])+1);

  while (1) {
        bytes = read(s, buf, BUF_SIZE);
       if (bytes <= 0) exit(0);
        write(1, buf, bytes);
  }
}
```

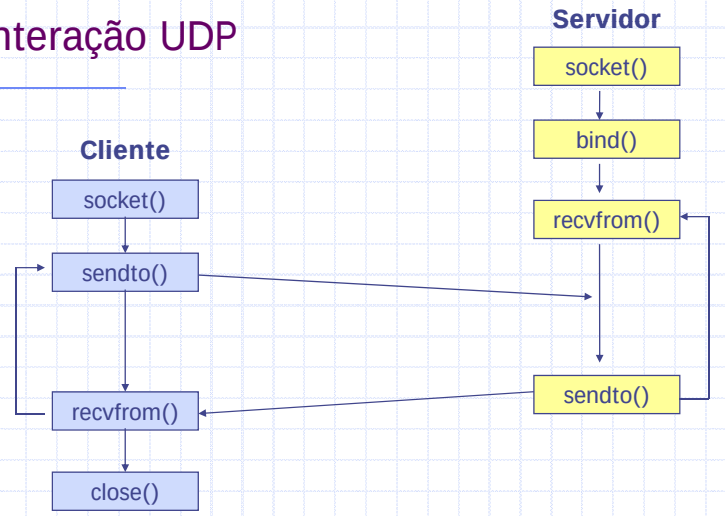

Programa Servidor TCP

```
#define SERVER_PORT 12345

int main(int argc, char *argv[])
{
  int s, b, l, fd, sa, bytes, on = 1;
  char buf[BUF_SIZE];
  struct sockaddr_in channel;
  memset(&channel, 0, sizeof(channel));
  channel.sin_family = AF_INET;
  channel.sin_addr.s_addr = htonl(INADDR_ANY);
  channel.sin_port = htons(SERVER_PORT);

  s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
  if (s < 0) fatal("socket failed");
  b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
  if (b < 0) fatal("bind failed");
  l = listen(s, QUEUE_SIZE);
  if (l < 0) fatal("listen failed");

  while (1) {
        sa = accept(s, 0, 0);
        if (sa < 0) fatal("accept failed");
        read(sa, buf, BUF_SIZE);
        fd = open(buf, O_RDONLY);
        if (fd < 0) fatal("open failed");
        while (1) {
                bytes = read(fd, buf, BUF_SIZE);
                if (bytes <= 0) break;
                write(sa, buf, bytes);
        }
        close(fd);  close(sa);
  } }
```


Programa Servidor Concorrente – fork()

```
int main(int argc, char **argv)
{
     int  listenfd, connfd;
     pid_t     childpid;
     socklen_t clilen;
     struct sockaddr_in  cliaddr, servaddr;
     void sig_chld(int);

     listenfd = Socket(AF_INET, SOCK_STREAM, 0);

     bzero(&servaddr, sizeof(servaddr));
     servaddr.sin_family      = AF_INET;
     servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
     servaddr.sin_port        = htons(SERV_PORT);

     Bind(listenfd, (SA *) &servaddr, sizeof(servaddr));
     Listen(listenfd, LISTENQ);

     for ( ; ; ) {
          clilen = sizeof(cliaddr);
          connfd = accept(listenfd, (SA *) &cliaddr, &clilen);

          if ( (childpid = Fork()) == 0) {   /* child process */
               Close(listenfd);    /* close listening socket */
               str_echo(connfd);   /* process the request */
               exit(0);
          }
          Close(connfd);           /* parent process */
     }
}
```


Interação UDP



Programa Cliente UDP

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

     bzero(&servaddr, sizeof(servaddr));
     servaddr.sin_family = AF_INET;
     servaddr.sin_port = htons(7);
     Inet_pton(AF_INET, argv[1], &servaddr.sin_addr);

     sockfd = Socket(AF_INET, SOCK_DGRAM, 0);

     dg_cli(stdin, sockfd, (SA *) &servaddr, sizeof(servaddr));

     exit(0);
}
void dg_cli(FILE *fp, int sockfd, const SA *pservaddr, socklen_t
               servlen)
{
     int  n;
     char sendline[MAXLINE], recvline[MAXLINE + 1];

     while (Fgets(sendline, MAXLINE, fp) != NULL) {
          Sendto(sockfd, sendline, strlen(sendline), 0, pservaddr,
                   servlen);
          n = Recvfrom(sockfd, recvline, MAXLINE, 0, NULL, NULL);
          recvline[n] = 0;    /* null terminate */
          Fputs(recvline, stdout);
     }
}
```


Programa Servidor UDP

```
int main(int argc, char **argv)
{
     int  sockfd;
     struct sockaddr_in  servaddr, cliaddr;

     sockfd = Socket(AF_INET, SOCK_DGRAM, 0);

     bzero(&servaddr, sizeof(servaddr));
     servaddr.sin_family      = AF_INET;
     servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
     servaddr.sin_port        = htons(SERV_PORT);

     Bind(sockfd, (SA *) &servaddr, sizeof(servaddr));

     dg_echo(sockfd, (SA *) &cliaddr, sizeof(cliaddr));
}
void dg_echo(int sockfd, SA *pcliaddr, socklen_t clilen)
{
     int  n;
     socklen_t len;
     char      mesg[MAXLINE];

     for ( ; ; ) {
          len = clilen;
          n = Recvfrom(sockfd, mesg, MAXLINE, 0, pcliaddr, &len);

          Sendto(sockfd, mesg, n, 0, pcliaddr, len);
     }
}
```