Redes de Computadores II

Sistemas Cliente/Servidor & Sockets

Prof. Elias P. Duarte Jr.
Universidade Federal do Paraná (UFPR)
Departamento de Informática
www.inf.ufpr.br/elias/redes

Sumário da Aula de Hoje

- Hoje retomamos a camada de aplicação
- 2 Classificações dos Sistemas Cliente Servidor:
  - Orientado à Conexão ou Não Orientado à Conexão
  - Concorrente e Iterativo
- Algoritmos de clientes e servidores
- A API (*Application Programmer Interface*) socket
  - programas que se comunicam pela rede

Sistemas Cliente-Servidor

- Considere dois processos que vão se comunicar usando TCP/IP ou UDP/IP
- Como começar a comunicação?

Sistemas Cliente-Servidor

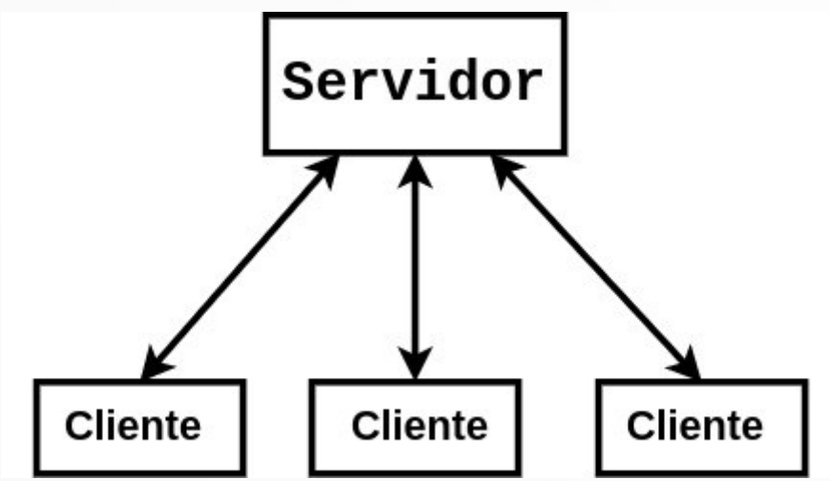
- O modelo Cliente-Servidor resolve exatamente este problema de iniciar a comunicação entre dois processos
- Solução é simples: um dos processos fica já em execução
- Escutando
- Aguardando uma comunicação de outro processo
- Que pode ser iniciada a qualquer momento

Sistemas Cliente-Servidor

- Servidor: o processo que fica em execução, escutando, aguardando a comunicação
- Cliente: o processo que inicia a comunicação, dispara a primeira mensagem para o servidor
- O Cliente oferece uma interface para um usuário (humano ou processo) que é quem demanda a comunicação

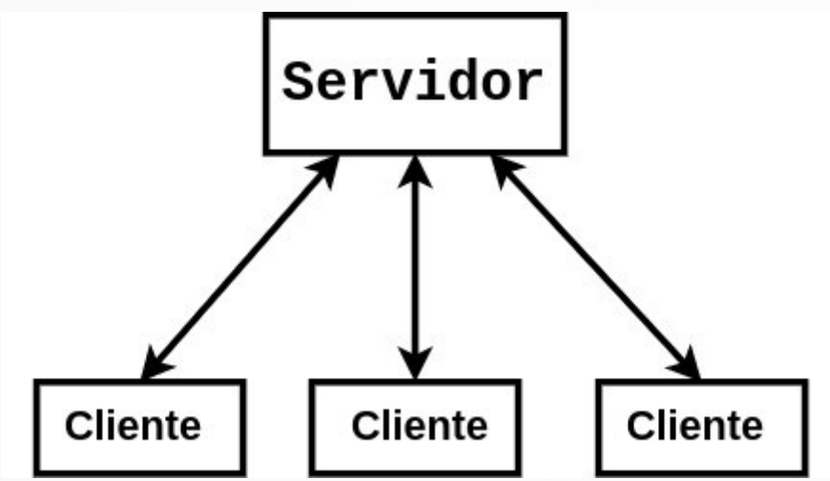
Sistemas Cliente-Servidor

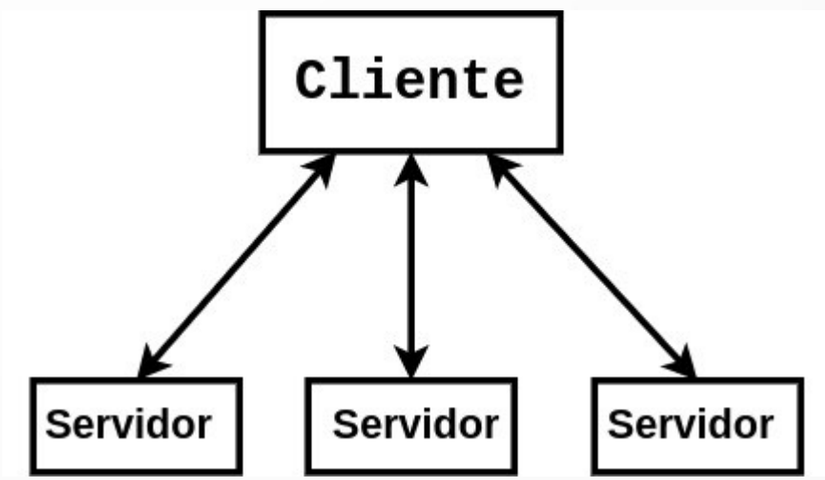
- Bancos de Dados X Redes de Computadores



Sistemas Cliente-Servidor

- Muitas possibilidades!





Sistemas Cliente-Servidor

- Importante perceber: tem mais a ver com o modelo de comunicação do que com o processo propriamente dito (mais leve/pesado etc.)
- Intermediário: *peer – ao mesmo tempo cliente e servidor*
- O modelo P2P (*Peer-to-Peer*) pode ser visto como alternativo ao modelo Cliente-Servidor
- Mas pode ser visto como parte do modelo: citar exemplo do DNS

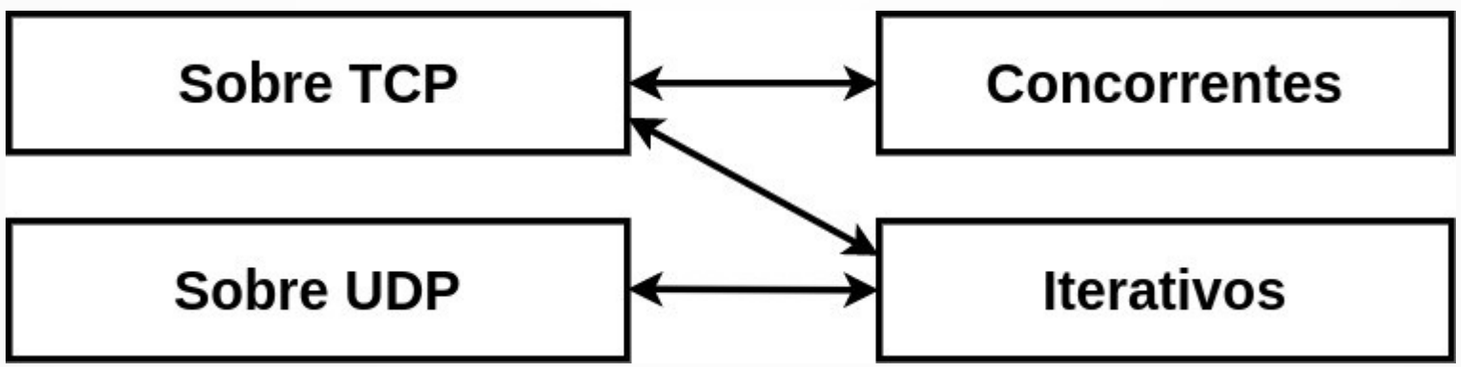
Sobre TCP e Sobre UDP

- A primeira classificação de sistemas Cliente-Servidor na Internet
- Sobre TCP/IP: Orientados à Conexão
  - tem que abrir a conexão antes de comunicar, depois fechar a conexão
- Sobre UDP/IP: Não Orientado à Conexão

Concorrentes Vs. Iterativos

- A segunda classificação de sistemas cliente servidor
- Iterativos: servidores atendem 1 cliente de cada vez
- Concorrentes: servidores atendem múltiplos clientes “ao mesmo tempo”

Ligue Elementos das Colunas



Os Algoritmos

- Cliente e Servidor TCP Iterativo
- Cliente e Servidor UDP Iterativo
- Cliente e Servidor TCP Concorrente

Cliente TCP

Cliente Orientado à Conexão

1. Determine o endereço IP e porta do servidor com o qual deseja se comunicar
2. Abra um socket usando uma porta local livre para comunicar
3. Estabeleça a conexão com o servidor
4. Comunique-se, de acordo com o protocolo de aplicação
5. Encerre a conexão e o socket

Servidor TCP Iterativo

Servidor TCP Iterativo

1. Abra um socket usando a porta conhecida para o serviço
2. Fique à escuta, aguardando requisições de conexão de clientes
3. Aceite requisição de conexão com o cliente: abra um novo socket para atendê-lo
4. Comunique-se, de acordo com o protocolo de aplicação
5. Encerre a conexão e volte ao passo 3

Servidor UDP Iterativo

Servidor UDP Iterativo

1. Abra um socket usando a porta conhecida para o serviço
2. REPITA: receba requisição do cliente, formule resposta e envie de volta ao cliente, de acordo com o protocolo de aplicação

Cliente UDP

Cliente Não Orientado à Conexão

1. Determine o endereço IP e porta do servidor com o qual deseja se comunicar
2. Abra um socket usando uma porta local livre para comunicar
3. Comunique-se, de acordo com o protocolo de aplicação
5. Encerre o socket

Servidor TCP Concorrente

Servidor TCP Concorrente
PAI
1. Abra um socket usando a porta conhecida para o serviço
2. Fique à escuta, aguardando requisições de conexão de clientes
3. Aceite requisição de conexão com o cliente: abra um processo filho para atendê-lo
FILHO
1. Estabeleça a conexão com o cliente, atenda-o em um novo socket
2. Comunique-se, de acordo com o protocolo de aplicação
3. Encerre a conexão e o processo

Servidor Concorrente: Múltiplos Processos

- Surge uma pergunta importante:
- Cada processo do mesmo servidor escuta em uma única porta ou são múltiplas portas?

Servidor Conconcorrente: Múltiplos Processos

- Surge uma pergunta importante:
- Cada processo do mesmo servidor escuta em uma única porta ou são múltiplas portas?
- Resposta: todos comunicam na mesma porta conhecida para o serviço
- Como assim? As portas são usadas na identificação dos processos que se comunicam

Sendo Mais Específico: Portas

Sendo Mais Específico: Portas

- As portas são usadas na identificação, mas não são os identificadores
- No caso de servidores concorrentes: o S.O. usa o (endereço IP, porta) do cliente para identificar para qual processo é cada mensagem
- Se está conectado: mensagem é para o processo filho correspondente
- Se não está conectado: mensagem é para o pai

Os Sockets

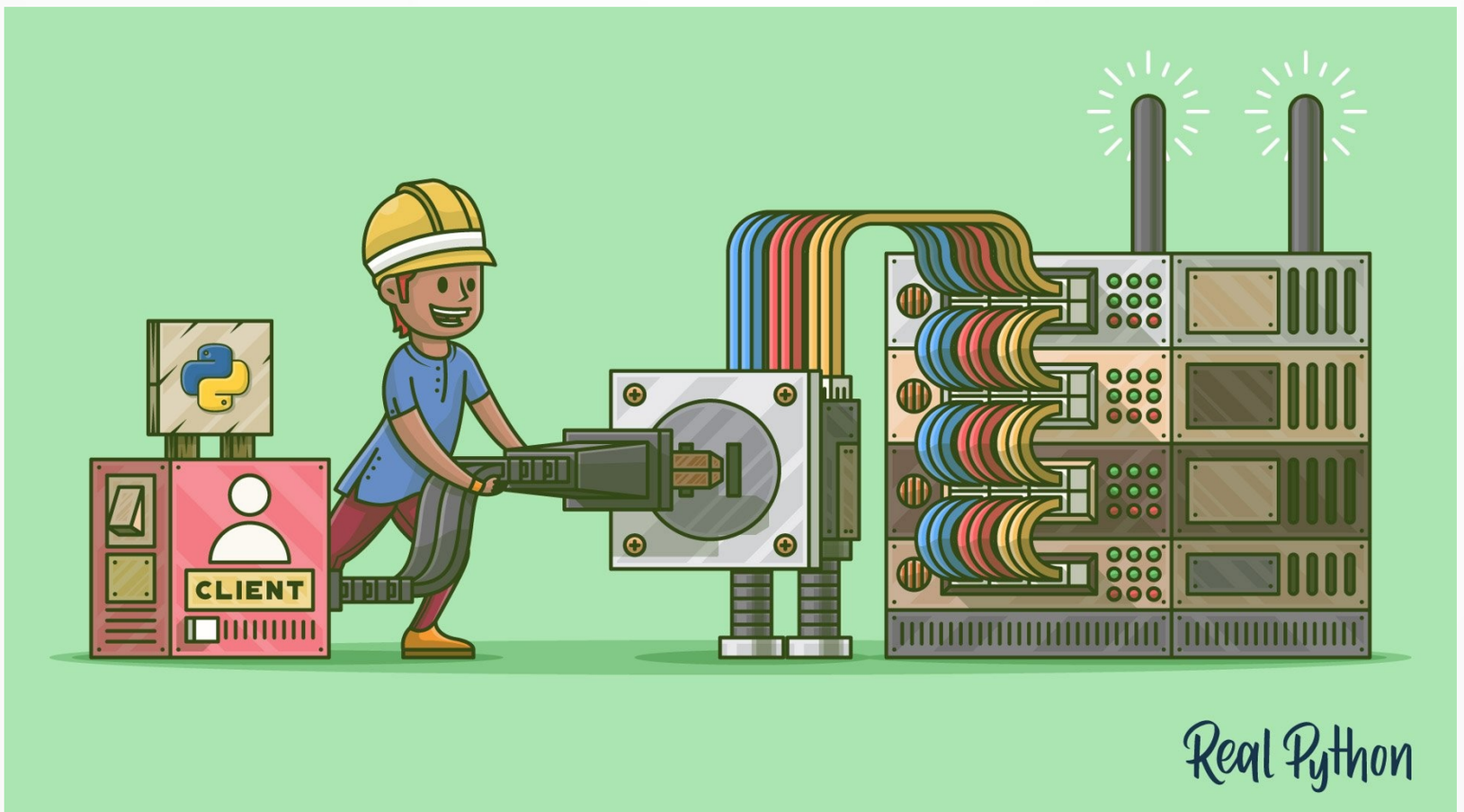
- Abstração originalmente do BSD Unix para comunicação de processos
- Hoje: presente em virtualmente todos os sistemas
- Frequentemente adaptado
- Por exemplo, no MS Windows: WinSock 
- Não é um padrão TCP: não há RFC para sockets!

Sockets

- Não há exclusividade:
- É possível usar sockets para a comunicação usando outra pilha de protocolos
- É possível fazer programação de sistemas TCP/IP usando outras API’s

Socket: Aspecto Abstração

- Socket em inglês: tomada

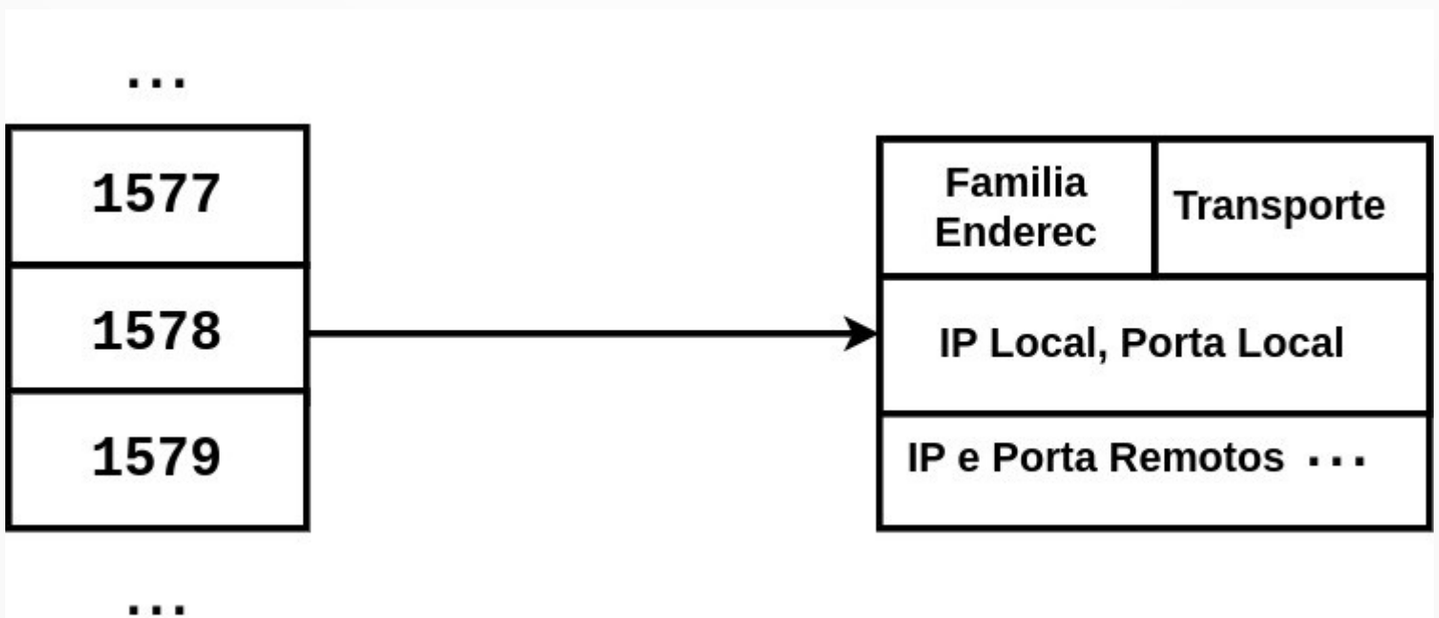


Socket: Enquanto API

- Neste aspecto: grande sucesso
- API: *Application Programming Interface*
- Interface para programar algo, no caso comunicação de dados
- Conjunto de funções:
  - você inclui uma biblioteca
  - tem acesso a todas as funções necessárias para construir programas que se comunicam usando TCP/IP

Sockets no Linux

- Entidades da mesma natureza que processos, arquivos, etc.
- O identificador do socket é chamado de descritor
- O descritor é um inteiro que indexa uma tabela de apontadores para estruturas de dados com informações sobre o socket



As Funções Básicas do Socket

- Na próxima aula vamos ver, digitar, compilar e executar um sistema cliente-servidor inteiro sobre TCP/IP
- Vamos usar a linguagem C e sockets Linux
- Hoje vamos aprender as funções básicas

Abrindo um Socket

- Aloca uma entrada na tabela de descritores e cria a estrutura de dados correspondente

  sock_desc = socket(AF_INET, transport, protocol)

Abrindo um Socket

- Aloca uma entrada na tabela de descritores e cria a estrutura de dados correspondente

  sock_desc = socket(AF_INET, transport, protocol)
- AF_INET: Address Family InterNET
- transport: SOCK_STREAM para TCP
- transport: SOCK_DGRAM para UDP
- protocol: 0 para família de protocolos TCP/IP

Fechando um Socket

- Libera a estrutura de dados e a entrada na tabela de descritores

  close(sock_descr)

Servidor Especifica Porta Local

- Servidor especifica a porta conhecida do serviço:

  bind(sock_descr, enderec_local, sizeof(enderec_local))

Servidor Especifica Porta Local

- Servidor especifica a porta conhecida do serviço:

  bind(sock_descr, enderec_local, sizeof(enderec_local))
- Para especificar endereços: vamos usar um registro da biblioteca socket

  sockaddr_in
- Tem campos para endereço IP, porta, protocolo, etc.

Cliente Especifica Endereço Remoto

- Cliente especifica endereço do servidor:

  connect(sock_descr, enderec_serv, sizeof(enderec_serv))

Cliente Especifica Endereço Remoto

- Cliente especifica endereço do servidor:

  connect(sock_descr, enderec_serv, sizeof(enderec_serv))
- Para especificar enderec_serv:

  sockaddr_in
- UDP pode usar connect!
  - Apenas armazena, especifica endereço remoto dos datagramas que vai enviar
- No caso do TCP: estabelece conexão

Transmissão de Dados

- A função default para ser usada

  write(sock_descr, buffer, num_bytes_transmitir)

Transmissão de Dados

- A função default para ser usada

  write(sock_descr, buffer, num_bytes_transmitir)
- 4 alternativas:
  - send: permite que o programador manipule flags que controlam a transmissão, por exemplo: dados urgentes
  - writev: writev, especifica um vetor de apontador para os dados a serem transmitidos
  - sendto: UDP, permite especificar o endereço do destinatário
  - sendmsg: todas as possibilidades acima

Recebendo Dados

- A função default para ser usada

  read(sock_descr, buffer, num_bytes_máximo)

Recebendo Dados

- A função default para ser usada

  read(sock_descr, buffer, num_bytes_máximo)
- 4 alternativas:
  - recv: permite que o programador manipule flags que controlam a transmissão, por exemplo: dados urgentes
  - readv: writev, especifica um vetor de apontador para os dados a serem transmitidos
  - recvfrom: UDP, permite especificar o endereço do destinatário
  - recvmsg: todas as possibilidades acima

O que faltou?
Abrimos, fechamos, especificamos endereço local e remoto, fizemos transmissão e recepção...

Servidor: Escuta e Conexão

- Um servidor fica em modo de escuta
  - só executa uma vez

  listen(sock_descr, tam_fila)
- Tamanho sugerido para a fila de clientes?

Servidor: Escuta e Conexão

- Um servidor fica em modo de escuta
  - só executa uma vez

  listen(sock_descr, tam_fila)
- Tamanho sugerido para a fila de clientes: 5
- Parece pouco, mas:
  - Nos servidores concorrentes: cliente só fica até abrir o processo filho para atendê-lo
  - Nos servidores iterativos: deve ser suficiente, caso contrário transforme em concorrente 

Servidor: Escuta e Conexão

- Para estabelecer uma conexão solicitada pelo cliente

  accept(sock_descr, enderec_cli, sizeof(enderec_cli))
- Para especificar enderec_cli:
- sockaddr_in

Completamos!
Conjunto básico de operações para programas sistemas cliente servidor TCP/IP!

Conclusão

- Revisamos o conceito de sistemas cliente-servidor
- Vimos as duas classificações
  - TCP & UDP
  - Concorrente & Iterativo
- Algoritmos dos clientes e servidores
- A API Socket
- Próxima aula: um sistema cliente-servidor inteiro: vamos digitar, compilar e executar

Obrigado!

Lembrando: a página da disciplina é:
https://www.inf.ufpr.br/elias/redes