Sockets TCP

Servidor não bloqueante com select e concorrente com threads

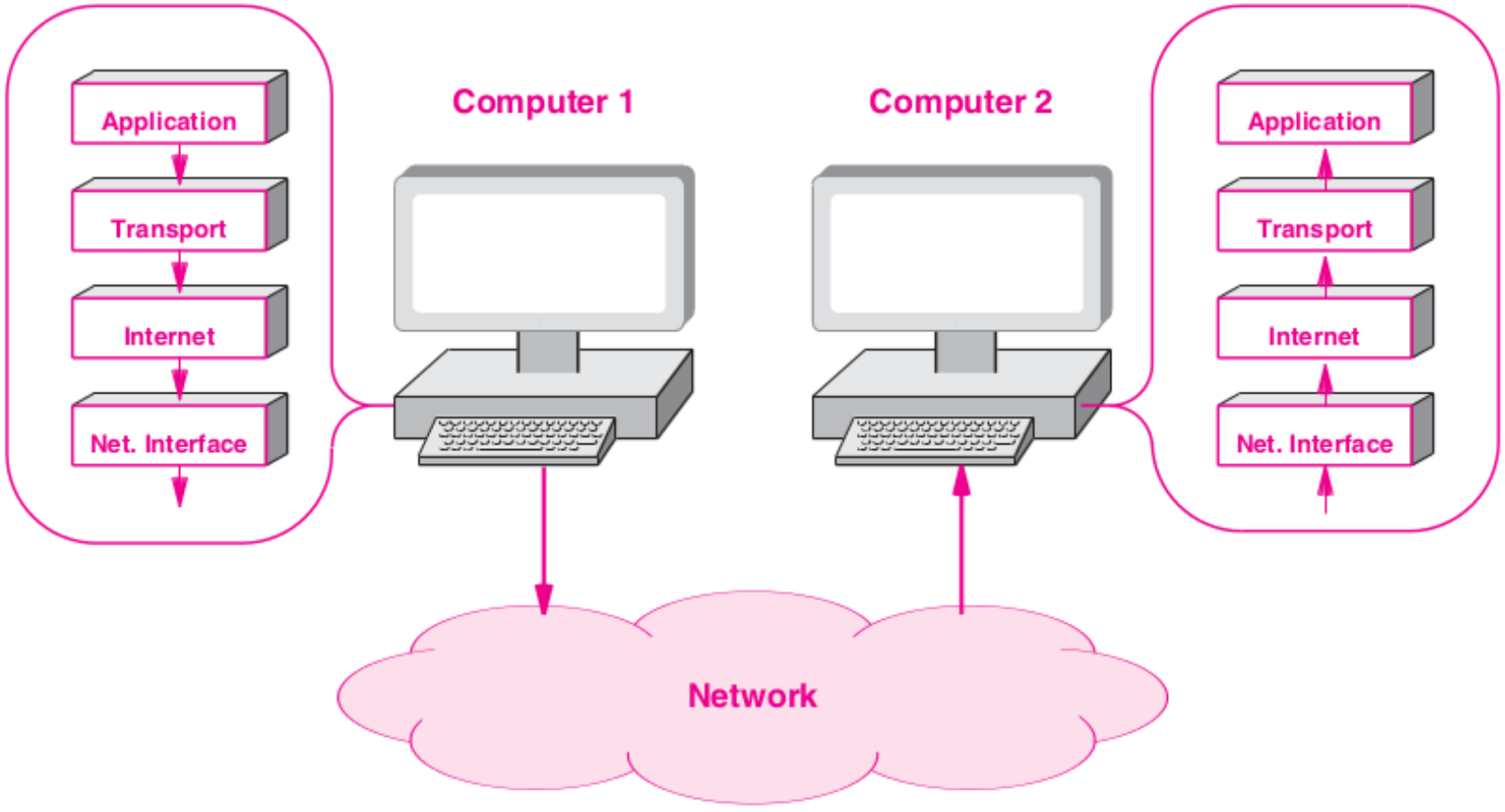
Prof. Jaime Cohen

2025

Universidade Estadual de Ponta Grossa

Sumário

Pilha de Protocolos e a Camada de Transporte

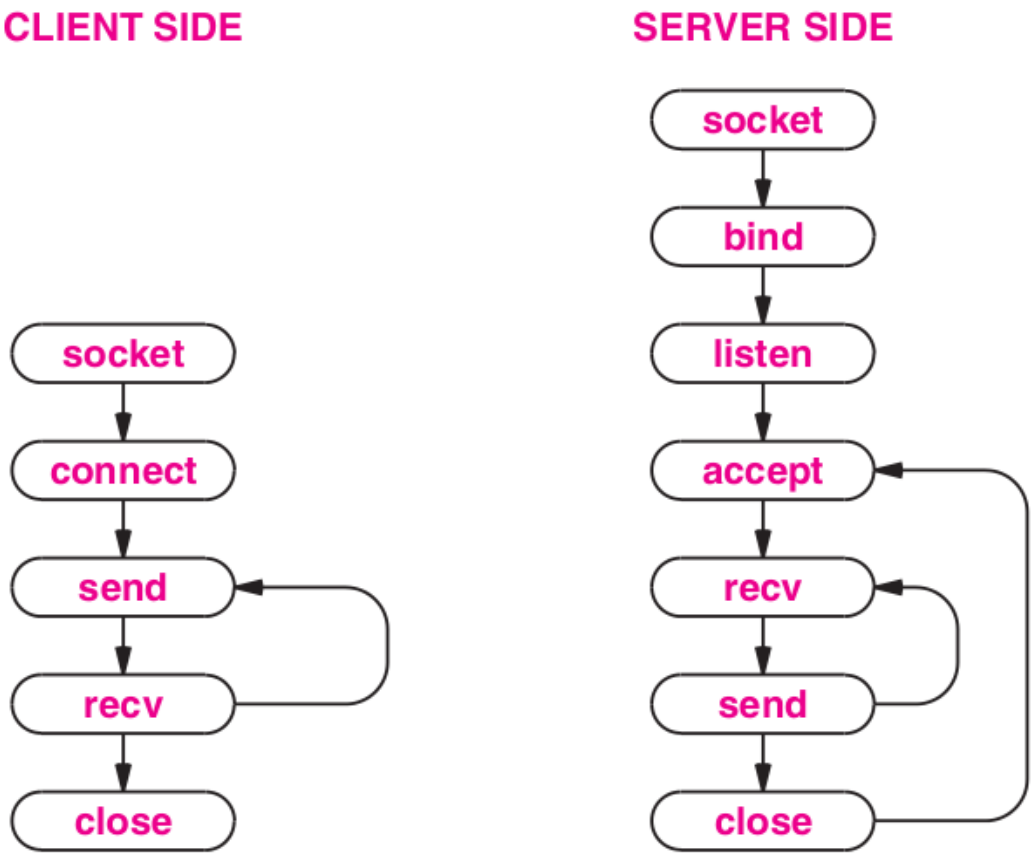


Pilha de Protocolos e a Camada de Transporte

UDP - User Datagram Protocol

TCP - Transport Control Protocol

Modelo de Programação Cliente/Servidor



Modelo de Programação Cliente/Servidor

Cliente TCP

- Crie um socket TCP (`socket`)
- Estabeleça uma Conexão (`connect`)
- Envie e receba dados (`send` e `recv`)
- Feche a conexão

Servidor TCP

- Crie um socket TCP (`socket`)
- Associe um número de porta e endereço IP (`bind`)
- Prepare o socket para receber conexões (`listen`)
- Repita
  - Aceite uma Conexão (`accept`)
  - Com concorrência: crie uma thread ou processo
    - Envie e receba dados (`send` e `recv`)
    - Feche a conexão

Modelo de Programação Cliente/Servidor

Ponto de final do socket (*endpoint*)

- (Endereço IP, Número de Porta)
- Cada um, cliente e servidor, tem um *endpoint*
- Ou seja: 4 valores identificam um socket TCP
  - Servidor: ('192.168.1.1', 80)
  - Client: ('192.168.0.100', 14001)

Cuidados ao programar com sockets TCP

Atenção:

1. Implementar os comandos `send()` e `recv()` em laços para garantir que todos os bytes foram enviados/recebidos.
2. Testar o status de erro de todos os comandos da API sockets.

Universidade Estadual de Ponta Grossa

Programação com Sockets - Servidores e Clientes

- Um cliente por vez
- Múltiplos clientes com tratamento de eventos (select)
- Múltiplos clientes com múltiplas threads

Criação de Sockets

```python
import socket

# Cria um socket TCP (SOCK_STREAM)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print("Socket criado:", s)
s.close()
```

Universidade Estadual de Ponta Grossa

Criação de um servidor TCP simples

```python
import socket

HOST = 'localhost'
PORT = 5000

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()

print(f"Servidor escutando em {HOST}:{PORT}...")
conn, addr = s.accept()
print(f"Conexão recebida de {addr}")
conn.sendall(b'Bem-vindo ao servidor!\n')
conn.close()
s.close()
```

Teste do servidor

```
# execução do servidor
python servidor.py

# teste - o cliente será o comando telnet
telnet localhost 5000
```

Universidade Estadual de Ponta Grossa

Cliente

```python
# cliente.py
import socket

HOST = 'localhost'
PORT = 5000

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((HOST, PORT))
mensagem = s.recv(1024)
print("Mensagem recebida:", mensagem.decode())
s.close()
```

Acesso ao DNS - Sistema de Nomes de Domínio

- Comandos gethostbyname() e getaddrinfo()

Acesso ao DNS - Sistema de Nomes de Domínio

```python
import socket

dominio = 'www.google.com'
ip = socket.gethostbyname(dominio)
print(f"O IP de {dominio} é {ip}")

info = socket.getaddrinfo(dominio, 80)
for item in info:
    print(item)
```

Universidade Estadual de Ponta Grossa

Comando não bloqueante - select()

rlist, wlist, elist = select(entrada, saída, exceção, timeout)

Argumentos

- entrada: eventos de entrada
- saída: prontos para envio
- exceção: eventos de erros
- timeout: tempo para desbloqueio

Devolve

- rlist: sockets que receberam dados
- wlist: sockets preparados para envio de dados
- elist: socket com erros

Aplicação Cliente/Servidor

Exemplo: Monitor de Recursos

- Aplicação completa usando sockets: monitor de sistemas
- Servidor concorrente com `select()`
- Servidor concorrente com threads
- Tratamento de erros com `try` / `except`
- Como receber mensagens de tamanho fixo pelo socket

Universidade Estadual de Ponta Grossa

Monitor de Sistemas - Especificação

- O servidor é o monitor de carga de CPU e memória disponível
  - imprime os nomes das máquinas e as informações
- Os clientes enviam uma mensagem ao servidor a cada 3 segundos
- As mensagens tem tamanho fixo

Monitor de Sistemas - Projeto

- Clientes enviam mensagens de tamanho fixo: 67 bytes (65 caracteres)
- A mensagem:

Hostname:pc-home Carga de CPU: 0.5% Memória Disponível: 50.4%

Monitor de Sistemas - Projeto

- Servidor aceita várias conexões de clientes
  - Versão com `select()`
  - Versão com uma *thread* por cliente

Monitor de Sistemas - Projeto

- Tratamento de erros
  - `try` / `except`

Universidade Estadual de Ponta Grossa

Monitor de Sistemas - Projeto

- As informações são obtidas no linux usando a biblioteca psutils
  - `psutil.cpu_percent(interval=1.0)`
  - `psutil.virtual_memory()`

Universidade Estadual de Ponta Grossa

Comandos para o Cliente

1. `hostname = socket.gethostname()`
2. `s.connect((host, port))`
3. `s.sendall(mensagem.encode())`
4. `time.sleep(3)`
5. `s.close()`
6. string.encode()

Universidade Estadual de Ponta Grossa

Cliente: Estrutura

```
while True:
    mensagem = monta_mensagem()
    sock.sendall(mensagem.encode())
    time.sleep(3)
```

Tratamento de Erros

Erros são tratados com os comandos:

- try
- except
- else
- finally

Universidade Estadual de Ponta Grossa

Tratamento de Erros

```
try:
  # bloco de código que produz exceções
except:
  # código em caso de erro
else:
  # código em caso de nenhum erro
finally:
  # sempre é executado
```

Comando receive (recv)

```
data = sock.recv(max)
```

- bloqueia até:
  - dados foram recebidos do socket TCP
  - a conexão foi fechada
- recebe o máximo de `max` bytes
- se `data` é vazio, então a conexão foi fechada

Recebendo Mensagens pelo Socket

```
chunks = []
count = 0
while count != 67:
    data = dataSocket.recv(67-count)
    if not data:
        dataSocket.close()
        break # ou return
    else:
        chunks.append(data.decode())
        count += len(data)
print(''.join(chunks)+'\n')
```

Comandos para o Servidor

1. `s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)`
2. `s.bind((host, port))`
3. `s.listen(5)`
4. `sock = s.accept()`
5. `data = sock.recv(num_maximo_de_bytes)`
6. `select()`
7. `s.close()`
8. string.decode()

Universidade Estadual de Ponta Grossa

Servidor sem Bloqueio: select

rlist, wlist, elist = select(entrada, saída, exceção, timeout)

Universidade Estadual de Ponta Grossa

Servidor sem Bloqueio: Estrutura do Código

```
rlist, wlist, elist = select.select(readSockets, [], [], 2)
if [rlist, wlist, elist] == [ [], [], [] ]:
    print('Fazendo qualquer processamento\n')
else:
    for sock in rlist:
        if sock is acceptSocket:
          # novo cliente se conectando
              # executa accept()
              # inclui novo socket na lista readSockets
        else:
          # evento em um dataSocket
              # receba e imprima mensagem de um cliente
```

Threads

- Threads são linhas de execução concorrentes em um mesmo processo.

Universidade Estadual de Ponta Grossa

Múltiplas Threads

```
import threading

def handle_client(socket, addr)
  # código para

...
# para criar uma nova thread
dataSocket, addr = acceptSocket.accept()

thread = threading.Thread(target=handle_client, args=(dataSocket,
                                                      addr))
thread.start()
```

Código Completo

Links para os códigos completos:

- Cliente: cliente.py
- Servidor não bloqueante com select: servidor4-select.py
- Servidor multithread: servidor5-multithread.py

Instruções para execução

- Aplicação Cliente/Servidor: Monitoramento de Sistemas

Universidade Estadual de Ponta Grossa

Recursos online

- Socket Programming HOWTO
- Python - Network Programming
- Python and Basic Networking Operations | Developer.com