



PMR3412 - Redes Industriais - 2021

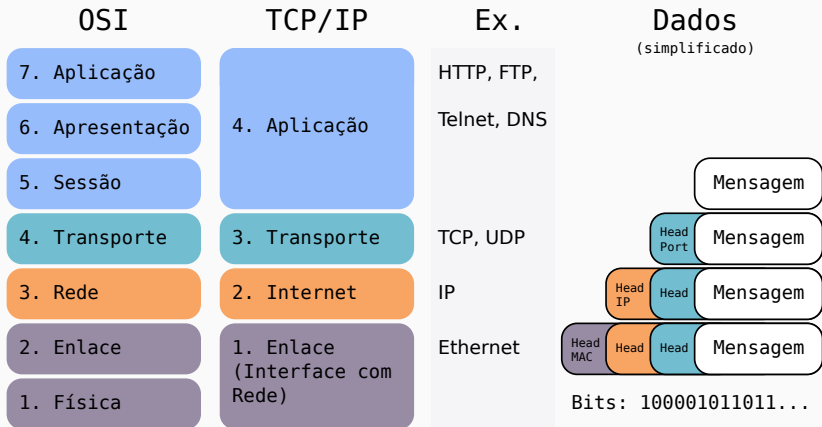
Aula 05 - Socket API e Introdução às Aplicações TCP/IP

Prof. Dr. André Kubagawa Sato

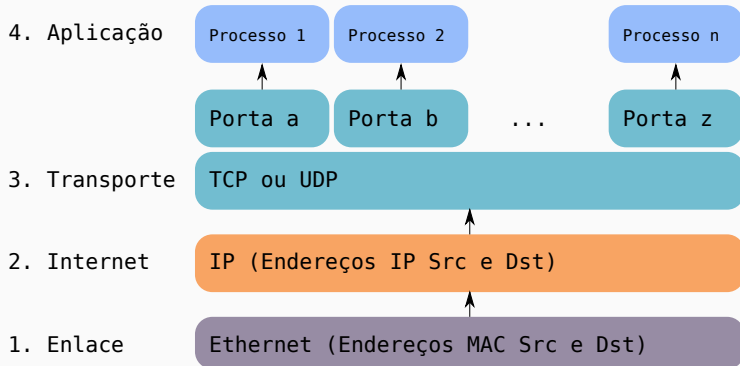
Prof. Dr. Marcos de Sales Guerra Tsuzuki

16 de Setembro de 2021

PMR-EPUSP

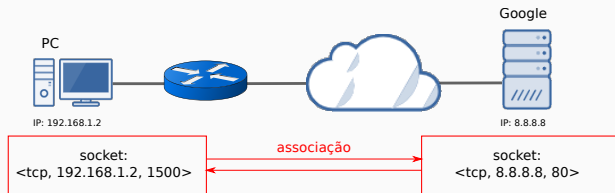


Revisão - Portas (Camada 3)



Socket API

- ▶ A interface Sockets é uma interface de programação de aplicações (API) genérico para protocolos de comunicação, o que inclui o TCP/UDP.
- ▶ Um endereço de socket é dado pela tupla: <protocolo, endereço local, porta local>
- ▶ Suportam os protocolos de camada 3: TCP e UDP.
- ▶ Em uma aplicação cliente-servidor, o socket do servidor geralmente "ouve" em uma porta, mas a conexão pode ser estabelecida em uma outra porta.

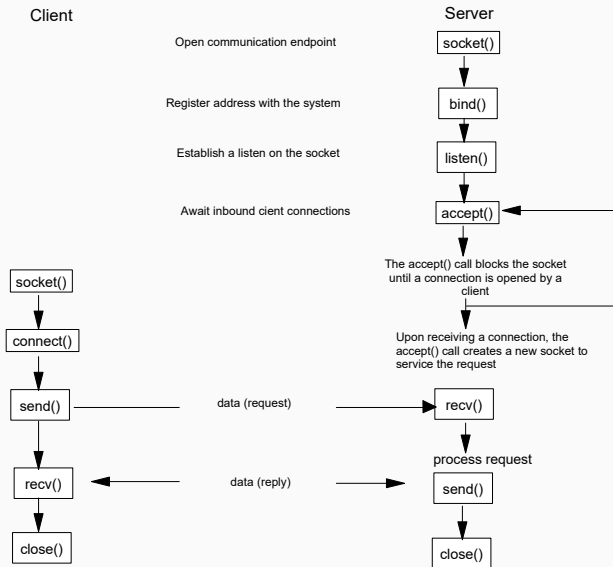


- ▶ A API (*application programming interface*) socket foi desenvolvida como uma interface de programação de comunicação genérica. Possibilita comunicação entre processos (locais ou remotos).
- ▶ Introduzida no sistema operacional *4.2BSD Unix operating system*, virou o padrão da indústria. Possui compatibilidade com o IPv4 e o IPv6, entre outros protocolos.
- ▶ Providencia três tipos de serviços:
 - ▶ Stream Sockets: sem limites para os dados, envia através de uma conexão confiável (TCP). Exemplos: HTTP, FTP.
 - ▶ Datagram Sockets: sem conexão (UDP), cada datagrama é enviado como pacote independente. Exemplos: NFS.
 - ▶ Raw Sockets: acesso direto às camadas inferiores, utilizado para testar protocolos novos. Exemplo: comando ping.

- ▶ As principais funções devem permitir a aplicação a realizar as seguintes ações:
 - ▶ Inicializar um socket.
 - ▶ Fazer o *bind* (registrar) um socket a um endereço de porta.
 - ▶ Escutar em um socket por conexões de entrada.
 - ▶ Aceitar uma conexão de entrada.
 - ▶ Conectar externamente a um servidor.
 - ▶ Enviar e receber dados em um socket.
 - ▶ Fechar um socket.
- ▶ O padrão da indústria é baseado no *Berkeley sockets* (ou BSD socket API), que fornece as funções: `socket`, `bind`, `listen`, `accept`, `connect`, `sendmsg/recvmmsg` e `close`.

Socket API - Exemplo de cenário cliente servidor

- ▶ Exemplo de comunicação socket API com conexão (TCP):



- ▶ O UDP pode ser utilizado para comunicação sem conexão.
- ▶ Neste caso só são utilizadas as funções `bind`, `sendto` e `recvfrom`.
- ▶ Assim, o socket de destino/origem deve ser fornecido a cada envio/recebimento de datagrama.

Client/server connection	Establish	Send	Receive
Connection-oriented server	<code>bind()</code> <code>listen()</code> <code>accept()</code>	<code>send()</code> <code>sendto()</code> <code>write()</code>	<code>recv()</code> <code>recvfrom()</code> <code>read()</code>
Connection-oriented client	<code>connect()</code>	<code>send()</code> <code>sendto()</code> <code>write()</code>	<code>recv()</code> <code>recvfrom()</code> <code>read()</code>
Connectionless server	<code>bind()</code>	<code>sendto()</code>	<code>recvfrom()</code>
Connectionless client	<code>bind()</code>	<code>sendto()</code>	<code>recvfrom()</code>

O módulo `socket` do Python

- ▶ A documentação pode ser vista em <https://docs.python.org/3/library/socket.html>
- ▶ Exemplo: função da criação de um socket

Functions

Creating sockets

The following functions all create [socket objects](#).

`socket.socket(family=AF_INET, type=SOCK_STREAM, proto=0, fileno=None)`

Create a new socket using the given address family, socket type and protocol number. The address family should be `AF_INET` (the default), `AF_INET6`, `AF_UNIX`, `AF_CAN`, `AF_PACKET`, or `AF_RDS`. The socket type should be `SOCK_STREAM` (the default), `SOCK_DGRAM`, `SOCK_RAW` or perhaps one of the other `SOCK_*` constants. The protocol number is usually zero and may be omitted or in the case where the address family is `AF_CAN` the protocol should be one of `CAN_RAW`, `CAN_BCM` or `CAN_ISOTP`.

If `fileno` is specified, the values for `family`, `type`, and `proto` are auto-detected from the specified file descriptor. Auto-detection can be overruled by calling the function with explicit `family`, `type`, or `proto` arguments. This only affects how Python represents e.g. the return value of `socket.getpeername()` but not the actual OS resource. Unlike `socket.fromfd()`, `fileno` will return the same socket and not a duplicate. This may help close a detached socket using `socket.close()`.

► Código servidor:

```
# Echo server program
import socket

HOST = ''          # Symbolic name meaning all available interfaces
PORT = 50007      # Arbitrary non-privileged port
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen(1)
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data: break
            conn.sendall(data)
```

▶ Código cliente:

```
# Echo client program
import socket

HOST = 'localhost' # The remote host
PORT = 50007       # The same port as used by the server
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b'Hello, world')
    data = s.recv(1024)
print('Received', repr(data))
```

▶ *Output* esperado:

- ▶ Cliente: Received b'Hello, world'
- ▶ Servidor: Connected by ('127.0.0.1', porta)

Python sockets com Múltiplas conexões

- ▶ Uma das estratégias mais simples para lidar com sockets puro, devido a sua simplicidade e eficiência.
- ▶ Realiza demultiplexação de operações de entrada/saída, como por exemplo leitura e escrita de arquivos. Ou seja, permite chavear entre diversas dessas operações, possibilitando execução aparentemente simultânea.
- ▶ Sua implementação depende do sistema operacional. No entanto, a biblioteca `selector` do Python abstrai as diferenças.
- ▶ Não é multithread, sua execução não ocorre em paralelo. Isto tem consequências diretas na escalabilidades.
- ▶ Para utilizá-lo, é necessário configurar o sockets para modo não bloqueante com `socket.setblocking(False)`

Python sockets - Servidor com múltiplas conexões (I) usando select

► Código servidor:

```
# Echo multiconn server program
import selectors
import socket

HOST = '0.0.0.0' # Symbolic name meaning all available interfaces
PORT = 50007 # Arbitrary non-privileged port

sel = selectors.DefaultSelector()

sock = socket.socket()
sock.bind((HOST, PORT))
sock.listen(100)
sock.setblocking(False)
sel.register(sock, selectors.EVENT_READ, accept)

while True:
    events = sel.select()
    for key, mask in events:
        callback = key.data
        callback(key.fileobj, mask)
```

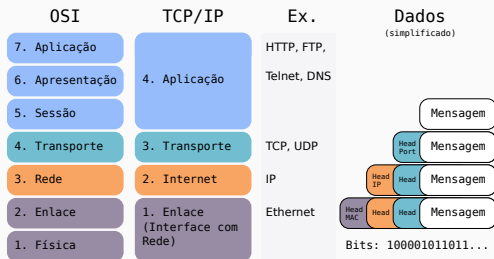

- ▶ Código servidor (funções de *callback*):

```
def accept(sock, mask):
    conn, addr = sock.accept() # Should be ready
    print('Connected by', addr)
    conn.setblocking(False)
    sel.register(conn, selectors.EVENT_READ, read)

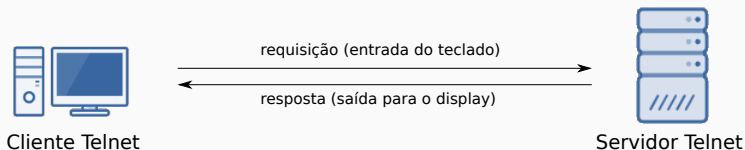
def read(conn, mask):
    data = conn.recv(1000) # Should be ready
    if data:
        conn.send(data) # Hope it won't block
    else:
        sel.unregister(conn)
        conn.close()
```

Exemplos de Aplicações TCP/IP

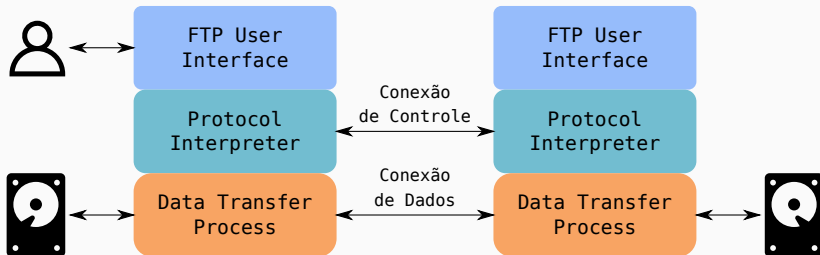
- ▶ Permite utilizar os serviços do protocolo TCP/IP, disponibilizando recursos para comunicação e transferência de dados entre hosts.
- ▶ Aplicações podem ser desenvolvidas pelo usuário ou nativas (que já estão incluídas no TCP/IP). Principais protocolos já incluídos: HTTP, DNS, FTP, SMTP.
- ▶ Fazem uso dos mecanismos de transporte UDP (mais rápido) ou TCP (mais confiável). O protocolo TCP é o mais popular, pois já providencia mecanismos de recuperação de erro e controle de fluxo.
- ▶ Envia comandos como sequência de caracteres.
- ▶ Corresponde a camada 4 do modelo TCP/IP:



- ▶ Especificado na RFC 854 e RFC 855.
- ▶ Aplicação segue o cliente servidor.
- ▶ Providencia uma interface padronizada para que o cliente possa acessar recursos de um servidor.
- ▶ O objetivo é agir como um terminal local conectado ao servidor.
- ▶ Em geral, opera em linha de comando (não providencia interface gráfica)

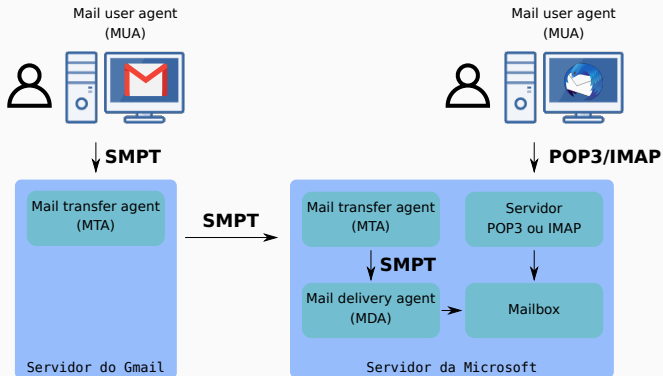


- ▶ Protocolo de transferência de arquivos de um host para o outro (ver RFC 959).
- ▶ Utiliza o protocolo de transporte TCP para implementar dois tipos de conexão
- ▶ Conexão de controle: servidor escuta a porta 21. Utilizada para comandos do cliente para manipular arquivos, fazer login e finalizar sessão.
- ▶ Conexão de dados: geralmente estabelecida na porta 20 do servidor. Utilizada para realizar a transferência de dados.



Aplicações TCP/IP - Protocolos de Email

- ▶ Protocolos principais: SMTP (“envio”), POP3 e IMAP (“recebimento”).
- ▶ Modelo com três tipos de componentes:
 - ▶ Mail user agent (MUA): interface do usuário para ler/enviar email.
 - ▶ Mail transfer agent (MTA): funciona como um roteador de emails.
 - ▶ Mail delivery agent (MDA): faz a entrega das mensagens para o local apropriado.



- ▶ SMTP se baseia em um sistema de entrega ponta a ponta, conectando na porta TCP 25 do servidor para entregar o email.
- ▶ A mensagem SMTP é dividida em:
 - ▶ Cabeçalho: definido pelo RFC 2822, terminado por uma linha nula (linha vazia seguida de <CRLF>)
 - ▶ Conteúdo: tudo após a linha nula, consistindo de sequência de linhas de caracteres ASCII.

```
MIME-Version: 1.0
Date: Mon, 12 Oct 2020 12:46:21 -0300
Message-ID: <CAMt1VZqab6hqR5V8MWhaaE71c36Ced2+a0NXRkiDQetJtZuJBQ@mail.gmail.com>
Subject: PMR3412
From: Andre Sato <andre.kubagawa@gmail.com>
To: "André Kubagawa Sato" <andre.kubagawa@usp.br>
Content-Type: text/plain; charset="UTF-8"
```

Cabeçalho
(Header ou envelope)

linha nula (null line)

```
←
Lorem Ipsum is simply dummy text of the printing and typesetting industry.
Lorem Ipsum has been the industry's standard dummy text ever since the
1500s, when an unknown printer took a galley of type and scrambled it to
make a type specimen book. It has survived not only five centuries, but
also the leap into electronic typesetting, remaining essentially unchanged.
It was popularised in the 1960s with the release of Letraset sheets
containing Lorem Ipsum passages, and more recently with desktop publishing
software like Aldus PageMaker including versions of Lorem Ipsum.
```

Conteúdo
(Contents)

- ▶ Multipurpose Internet Mail Extensions (MIME): inclui objetos além de mensagens ASCII no corpo das mensagens.
- ▶ O corpo da mensagem é descrito pelo campo *Content-Type* na forma:
`Content-Type: type/subtype ;parameter=value`
- ▶ Deve sempre conter o par tipo/subtipo, com ou sem parâmetros.
- ▶ O valor padrão é `text/plain` se omitido.
- ▶ Inicialmente foram definidos sete tipos de content-type (RFC 2046):

Tipo	Subtipos	Descrição
Text	plain	Texto, com parâmetros para definir codificação de caracteres
Multipart	mixed, parallel, alternative, digest	Múltiplos tipos, delimitados pelo parâmetro boundary
Message	rfc822, partial, external-obdy	Múltiplas mensagens trasmitidas juntas
Image	jpeg, gif	Imagem que requer display
Video	mpeg	Vídeo que requer display
Audio	basic	Áudio que requer display
Application	postscript, octet-stream	Para tipos que não se encaixam nas outras categorias

- ▶ Atualmente definidos como *Media Types*, possuem registros controlados pela IANA em <https://www.iana.org/assignments/media-types/media-types.xhtml>

- ▶ Codificação para representar dados binários com caracteres ASCII. É especificada pelo campo `Content-Transfer-Encoding`.
- ▶ O Base64 considera apenas 73 caracteres seguros. Deste modo, utiliza caracteres de 6 bits, o que resulta em um total de 64 caracteres.
- ▶ O processamento se dá a cada 24 bits (= 3 bytes). Assim, para cada 3 bytes de entrada, o Base64 gera 4 caracteres (bytes de saída).
- ▶ Cada byte de saída é convertido para um caractere ASCII de acordo com a tabela *Base64 alphabet* (RFC 4648).
- ▶ Padding é realizado concatenando caracteres = de 6 bits.

orig.: 00000001 00000010 00000011

base64: 000000 010000 001000 000011 → 0 16 8 3 → AQID

orig.: 00000001 00000010 00000011 11001110

base64: 000000 010000 001000 000011 110011 100000 →
0 16 8 3 51 32 (pad) (pad) → AQIDzg==

Referências

- ▶ Para o curso: livro da IBM “TCP/IP Tutorial and technical overview” (disponível em <https://www.redbooks.ibm.com/redbooks/pdfs/gg243376.pdf>).
- ▶ Para esta aula: seções 11.1, 11.2.1 e capítulo 15.
- ▶ Documentação do módulo socket do Python: <https://docs.python.org/3/library/socket.html>.

The End!