PMR3412 - Redes Industriais - 2023

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

Prof. Dr. Newton Maruyama

14 de Setembro de 2023

PMR-EPUSP

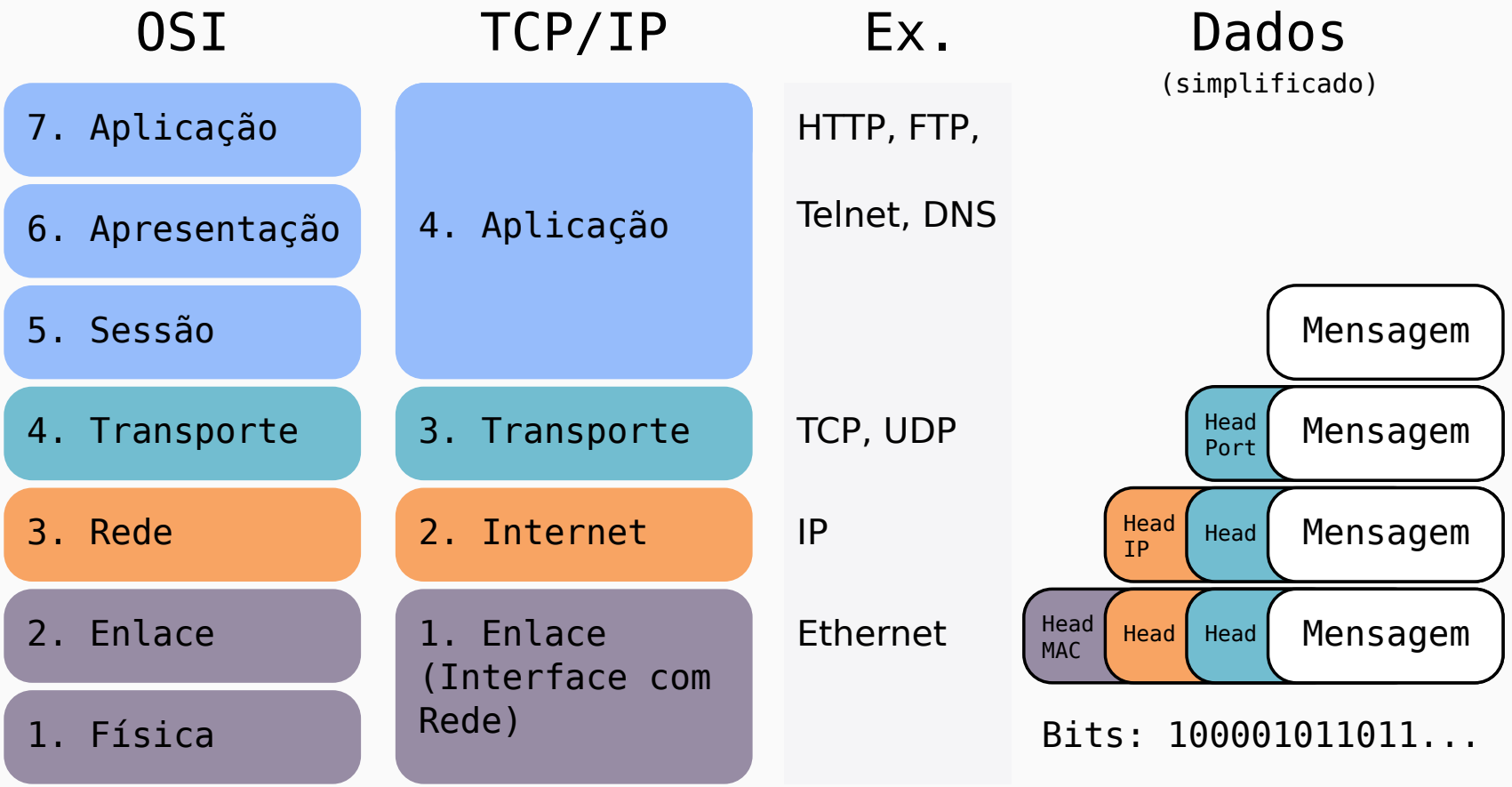
Notas de autoria (Disclaimer) - Equipe envolvida

Os slides que serão utilizados nesse ano são baseados no curso desenvolvido para os anos 2020, 2021 e 2022. Participaram da concepção do curso e desenvolvimento do material os seguintes professores:

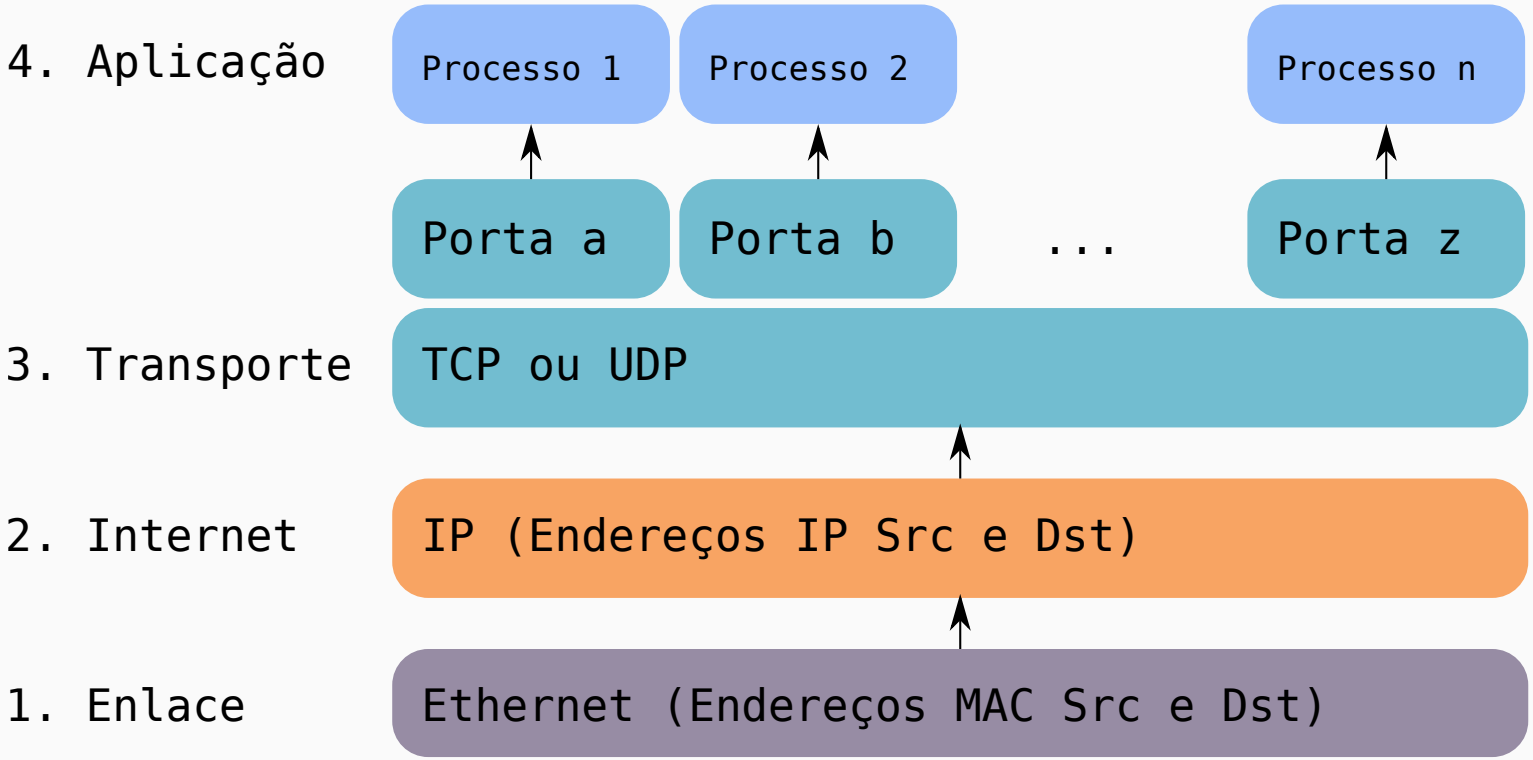
- Prof. Dr. André Kubagawa Sato
- Prof. Dr. Marcos de Sales Guerra Tsuzuki
- Prof. Dr. Edson Kenji Ueda
- Prof. Dr. Agesinaldo Matos Silva Junior
- Prof. Dr. André César Martins Cavalheiro

Conteúdo

Revisão - Relembrando o Modelo TCP/IP

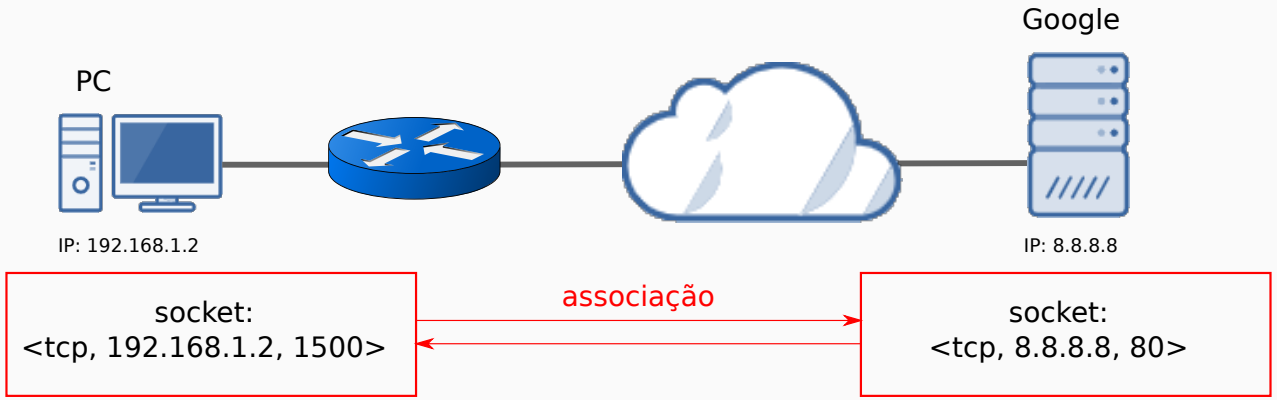


Revisão - Portas (Camada 3)



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 "escuta" em uma porta, mas a conexão, para troca de dados, pode ser estabelecida em uma outra porta.



Socket API - Histórico e Serviços

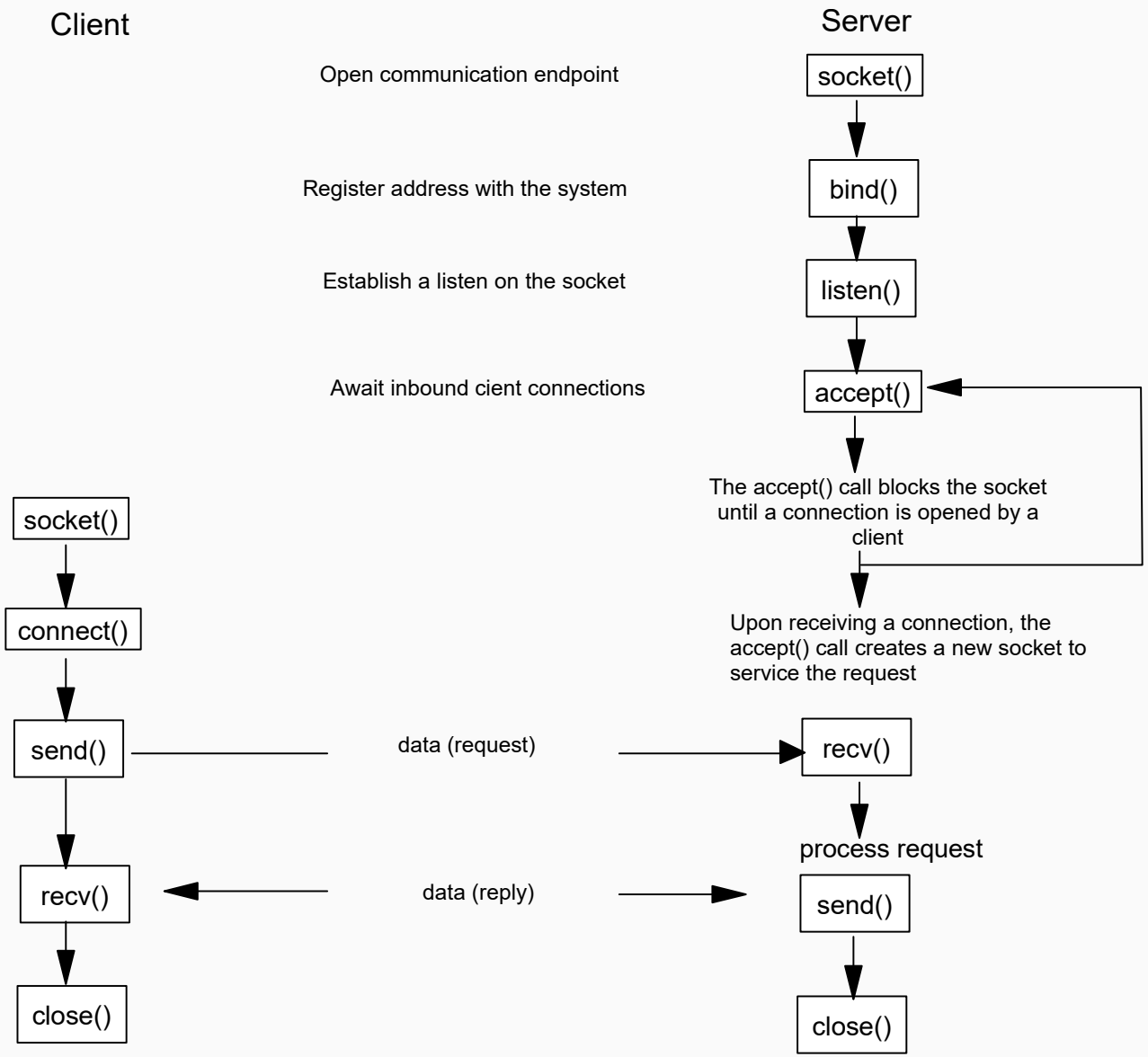
- A API (*application programming interface*) socket foi desenvolvida como uma interface de programação de comunicação genérica. Possibilita comunicação entre processos (locais ou remotos).
- Introduzida no sistema operacional *4.2BSD Unix operating system* (1983), se tornando o padrão da indústria. 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 (Network File System).
  - Raw Sockets: acesso direto às camadas inferiores, utilizado para testar protocolos novos. Exemplo: comando ping.

Socket API - Ações principais

- As principais funções devem permitir a aplicação a realizar as seguintes ações:
  - Inicializar um socket.
  - Fazer operação de *bind* (registrar) um socket a um endereço de porta.
  - Escutar a solicitação de 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 na versão *Berkeley sockets* (ou BSD socket API), que fornece as funções: `socket`, `bind`, `listen`, `accept`, `connect`, `sendmsg`/`recvmsg` e `close`.

Socket API - Exemplo de cenário cliente-servidor

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



Socket API - Outros cenários

- 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	`bind()`
	`listen()`
	`accept()`
		`send()`
		`sendto()`
		`write()`
			`recv()`
			`recvfrom()`
			`read()`

Connection-oriented client	`connect()`	`send()`
		`sendto()`
		`write()`
			`recv()`
			`recvfrom()`
			`read()`

Connectionless server	`bind()`	`sendto()`	`recvfrom()`
Connectionless client	`bind()`	`sendto()`	`recvfrom()`

O módulo `socket` da linguagem Python

Módulo socket - Documentação

- 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()`.

Demonstração - Servidor ECHO

- 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)
```

Demonstração - Servidor ECHO

- A declaração de um objeto do tipo socket é feito da seguinte forma:

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

  onde `socket.AF_INET` indica a utilização de endereços do tipo IPV4, e `socket.SOCK_STREAM` indica utilização do protocolo de transporte TCP.

- A associação entre o endereço IP e a porta utilizada é feito da seguinte forma:

```
s.bind((HOST, PORT))
```

- O servidor entra no estado de listen:

```
s.listen(1)
```

  O parâmetro 1 indica que o servidor aceita conexão com um único cliente.

Demonstração - Servidor ECHO

- A instrução:

```
conn, addr = s.accept
```

  inicialmente bloqueia a execução do código do servidor e o coloca em estado de espera de uma requisição de conexão de um cliente.
- Se houver a requisição de um cliente (através da função `s.connect()`) é realizado um *handshaking* de três vias.
- `conn` é um novo socket definido para troca de dados.
- `addr` é o endereço do novo socket (IP,Porta).
- A instrução:

```
data = conn.recv(1024)
```

  recebe a mensagem do cliente na variável `data` onde o parâmetro 1024 indica o tamanho do buffer.
- O servidor envia a mesma mensagem que recebeu para o cliente através da instrução:

```
conn.sendall(data)
```

Demonstração - Servidor ECHO

- 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))
```

Demonstração - Servidor ECHO

- O código do cliente deve saber a priori qual o endereço IP e porta do servidor:

```
HOST = 'localhost' # The remote host
PORT = 50007       # The same port as used by the server
```

- Também é definido um socket:

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

- Cliente solicita conexão com o servidor:

```
s.connect((HOST, PORT))
```

- Depois envia a mensagem e recebe a resposta do servidor:

```
s.sendall(b'Hello, world')
data = s.recv(1024)
```

- *Saída* esperada:
  - **Cliente:** `Received b'Hello, world'`
  - **Servidor:** `Connected by ('127.0.0.1', porta)`

Demonstração - Monitoração de sockets

- Inicialmente, vamos executar o programa servidor da seguinte forma:
  `python3 TCPServer.py`
- Comando `netstat`:
  `netstat -ant`
  O resultado:

```
(base) maruyama@DragonX:~/Documents/Graduacao/PMR3412/Slides/V2022/Aula5$ netsta
t -ant
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 192.168.122.1:53        0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:50007           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 10.10.180.95:42228      142.251.129.197:443     TIME_WAIT
tcp        0      0 10.10.180.95:40938      142.250.0.188:5228      FIN_WAIT2
tcp        0      0 10.10.180.95:47030      142.250.219.198:443     TIME_WAIT
tcp6       0      0 ::1:631                 :::*                    LISTEN
```

- Comando `lsof`:
  `lsof -i -n`
  O resultado:

```
(base) maruyama@DragonX:~/Documents/Graduacao/PMR3412/Slides/V2022/Aula5$ lsof -i:50007 -n
COMMAND  PID     USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
python3 4535 maruyama    3u  IPv4 103834      0t0  TCP *:50007 (LISTEN)
```

- Servidor UDP:

```
# Server Side
import socket

UDP_IP = "127.0.0.1"
UDP_PORT = 5005
sock = socket.socket(socket.AF_INET,    # IPv4
                     socket.SOCK_DGRAM) # UDP
sock.bind((UDP_IP, UDP_PORT))
while True:
    data, addr = sock.recvfrom(1024) # buffer size is 1024 bytes
    print("received message: %s" % data)
```

- Note que não é utilizado nesse caso as funções `listen()` e `accept()`.

Python sockets - Exemplo UDP

- Cliente UDP:

```
# UDP Client Side
import socket

UDP_IP = "127.0.0.1"
UDP_PORT = 5005
MESSAGE = b"Hello, World!"
print("UDP target IP: %s" % UDP_IP)
print("UDP target port: %s" % UDP_PORT)
print("message: %s" % MESSAGE)

sock = socket.socket(socket.AF_INET,    # Internet
                     socket.SOCK_DGRAM) # UDP
sock.sendto(MESSAGE, (UDP_IP, UDP_PORT))
```

Python sockets com Múltiplas conexões

- No exemplo anterior o servidor admite apenas conexão com um cliente.
- Quando o servidor deve lidar com múltiplas conexões a solução mais eficiente é obtida através da utilização de bibliotecas para processos concorrentes como a `asyncio` por exemplo.
- Uma solução mais simples pode ser realizada utilizando a função `select()`.
- É possível realizar demultiplexação de operações de entrada/saída, como por exemplo leitura e escrita de arquivos. Ou seja, permite chavear entre diversas operações, dando a sensação de execução aparentemente simultânea (concorrente).
- Sua implementação depende do sistema operacional mas a biblioteca `selector` da linguagem Python permite uma padronização das chamadas de funções.
- Essa solução não é escalável pois sua implementação não é *multithread*.
- É necessário configurar o sockets para modo não blocante: `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*):

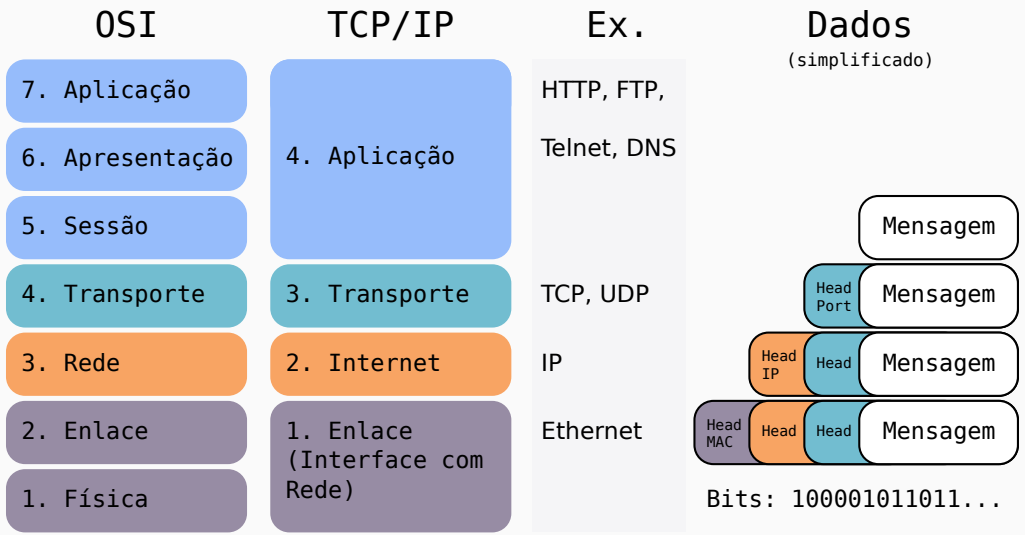
```
def accept(sock, mask):
    conn, addr = sock.accept()  # Should be ready
    print('Connected by', addr)
    conn.setblocking(False)
    sel.register(conn, selectors.EVENT_READ, read)

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()
```

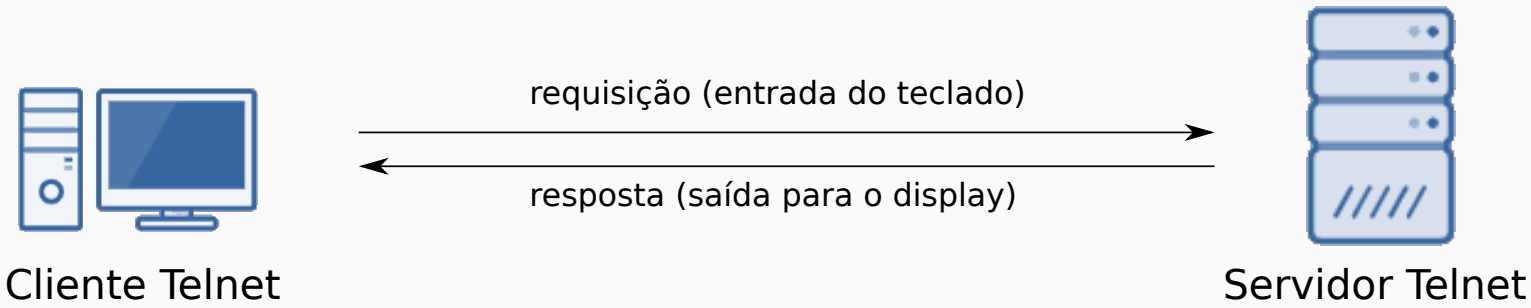
Exemplos de Aplicações TCP/IP

Aplicações TCP/IP - Introdução

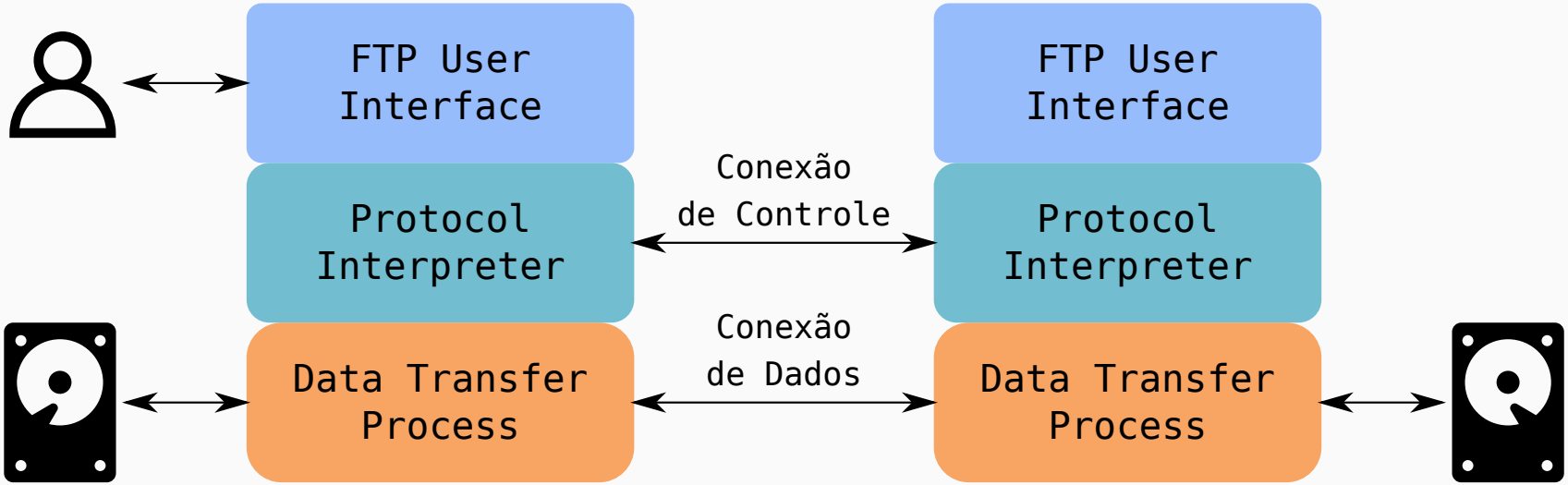
- Permite utilizar os serviços do protocolo TCP/IP, disponibilizando recursos para comunicação e transferência de dados entre hosts.
- Aplicações podem ser desenvolvidas pelo usuário ou nativas (que já estão incluídas no TCP/IP). Principais protocolos já incluídos: HTTP, DNS, FTP, SMTP.
- Fazem uso dos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



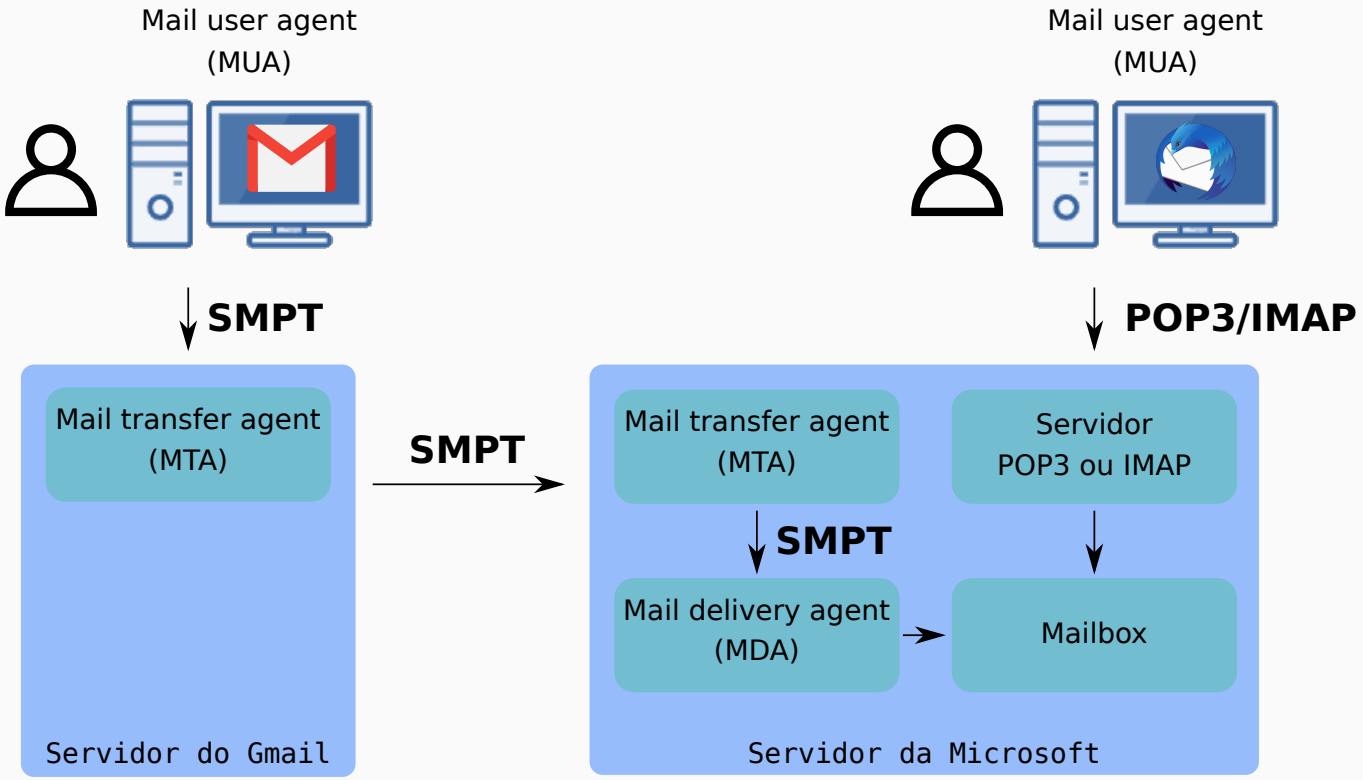
- Especificado na RFC 854 e RFC 855.
- Aplicação segue o cliente servidor.
- Providencia uma interface padronizada para que o cliente possa acessar recursos de um servidor.
- O objetivo é agir como um terminal local conectado ao servidor.
- Em geral, opera em linha de comando (não providencia interface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



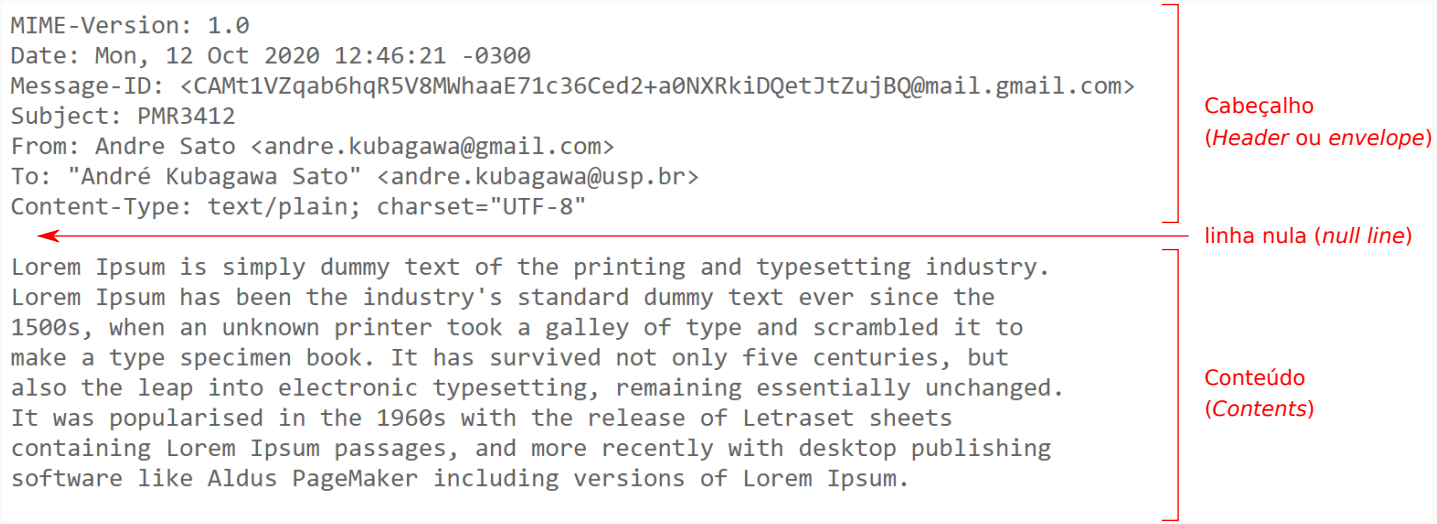
- Protocolo de transferência de arquivos de um host para o outro (ver RFC 959).
- Utiliza o protocolo de transporte TCP para implementar dois tipos de conexão
- Conexão de controle: servidor escuta a porta 21. Utilizada para comandos do cliente para manipular arquivos, fazer login e finalizar sessão.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



- 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.



- 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 (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	Video que requer display
Audio	basic	Audio 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`

- O *media type* multipart permite que o corpo da mensagem contenha múltiplos objetos de diferentes tipos.
- Cada parte é delimitada pelo parâmetro boundary, por exemplo:
  `Content-Type: multipart/mixed; boundary="19950213091"`
- Os quatro subtipos:
  - Mixed: partes independentes que são transmitidas juntas
  - Parallel: igual Mixed, mas sem ordenação.
  - Alternative: as partes são versões alternativas da mesma informação.
  - Digest: múltiplas mensagens transmitidas juntas. O tipo padrão de cada parte é message/rfc822, que deve possuir From, Subject e Date.

```
MIME-Version: 1.0
Date: Tue, 13 Oct 2020 08:50:45 -0300
Message-ID: <CA+6Ep9zHKz6KseULM6sNkR9=gBdcoN7fmd6c-WwDdHsO6TyhnQ@mail.gmail.com>
Subject: PMR3412
From: "André Kubagawa Sato" <andre.kubagawa@usp.br>
To: "André Kubagawa Sato" <andre.kubagawa@usp.br>
Content-Type: multipart/mixed; boundary="000000000000747f6f05b18c063b"

--000000000000747f6f05b18c063b
Content-Type: multipart/related; boundary="000000000000747f6e05b18c063a"

--000000000000747f6e05b18c063a
Content-Type: multipart/alternative; boundary="000000000000747f6c05b18c0639"

--000000000000747f6c05b18c0639
Content-Type: text/plain; charset="UTF-8"

Lorem Ipsum is simply dummy text of the printing and typesetting industry.
[image: image.png]

--000000000000747f6c05b18c0639
Content-Type: text/html; charset="UTF-8"
Content-Transfer-Encoding: quoted-printable

<div dir=3D"ltr">Lorem Ipsum is simply dummy text of the printing and types=
etting industry.=C2=A0<div><img src=3D"cid:ii_kg7wluwf0" alt=3D"image.png" =
width=3D"452" height=3D"12"><br></div><div>=C2=A0<br></div></div>

--000000000000747f6c05b18c0639--
--000000000000747f6e05b18c063a
Content-Type: image/png; name="image.png"
Content-Disposition: attachment; filename="image.png"
Content-Transfer-Encoding: base64
X-Attachment-Id: ii_kg7wluwf0
Content-ID: <ii_kg7wluwf0>


--000000000000747f6e05b18c063a--
--000000000000747f6f05b18c063b
Content-Type: text/plain; charset="US-ASCII"; name="lorem.txt"
Content-Disposition: attachment; filename="lorem.txt"
Content-Transfer-Encoding: base64
X-Attachment-Id: f_kg7wm5lv1
Content-ID: <f_kg7wm5lv1>

TG9yZW0gSXBzdW0gaXMgc2ltcGx5IGR1bW15IHRleHQgb2YgdGhlIHByaW50aW5nIGFuZCB0eXBl
c2V0dGluZyBpbmR1c3RyeS4gTG9yZW0gSXBzdW0gaGFzIGJlZW4gdGhlIGluZHVzdHJ5J3Mgc3Rh
bmRhcmQgZHVtbXkgdGV4dCBldmVyIHNpbmNlIHRoZSAxNTAwcywgd2hlbiBhbiB1bmtub3duIHBy
aW50ZXIgdG9vayBhIGdhbGxleSBvZiB0eXBlIGFuZCBzY3JhbWJsZWQgaXQgdG8gOGUgc3BlY2ltZW4gYm9vay4gSXQgaGFzIHN1cnZpdmVkIG5vdCBvbmx5IGZpdmUgY2VudHVyaWVz
LCBidXQgYWxzbyB0aGUgbGVhcCBpbnRvIGVsZWN0cm9uaWMgdHlwZXNldHRpbmcsIHJlbWFpbmlu
ZyBlc3NlbnRpYWxseSB1bmNoYW5nZWQuIEl0IHdhcyBwb3B1bGFyaXNlZCBpbiB0aGUgMTk2MHMg
d2l0aCB0aGUgcmVsZWFzZSBvZiBMZXRyYXNldCBzaGVldHMgY29udGFpbmluZyBMb3JlbSBJcHN1
bSBwYXNzYWdlcywgYW5kIG1vcmUgcmVjZW50bHkgd2l0aCBkZXNrdG9wIHB1Ymxpc2hpbmcgc29m
dHdhcmUgbGlrZSBBbGR1cyBQYWdlTWFrZXIgaW5jbHVkaW5nIHBhc3NhZlcnNpb25zIG9mIExvcmVtIElw
c3VtLg==
```

- 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

Referências - Aula 05

- 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 da linguagem Python: `https://docs.python.org/3/library/socket.html`.

The End!