Ciências ULisboa | Informática

Sockets e Tratamento de Erros com Sockets

Pedro Ferreira, Vinicius Cogo

Tópicos

1. Introdução à API de Sockets
2. Sockets em Python
3. Exceções em Python: Revisão rápida
4. Tratamento de erros em programas com Sockets:
    1. Abordagens para o desenvolvimento de programas
    2. Erros no módulo de *sockets*

Alguns Comandos Importantes

- netstat –ni → Descobrir as interfaces de rede disponíveis
- netstat -atu → Descobrir os portos TCP e UDP em uso
- ip → Configuração das interfaces de rede
- ping → Testar conetividade a sistema remoto
- traceroute → Ver encaminhamento até sistema remoto

1. Introdução à API de Sockets

- Objetivo: sistema **cliente/servidor** que comunica através de **sockets**

- API introduzida para UNIX BSD4.1 em 1981

- Criado, usado e libertado **explicitamente** pelas aplicações

- Modelo cliente-servidor

- Tipos de serviço de transporte:
  - **Com ligação** (e.g., TCP – Transmission Control Protocol)
  - **Sem ligação** (e.g., UDP – User Datagram Protocol)

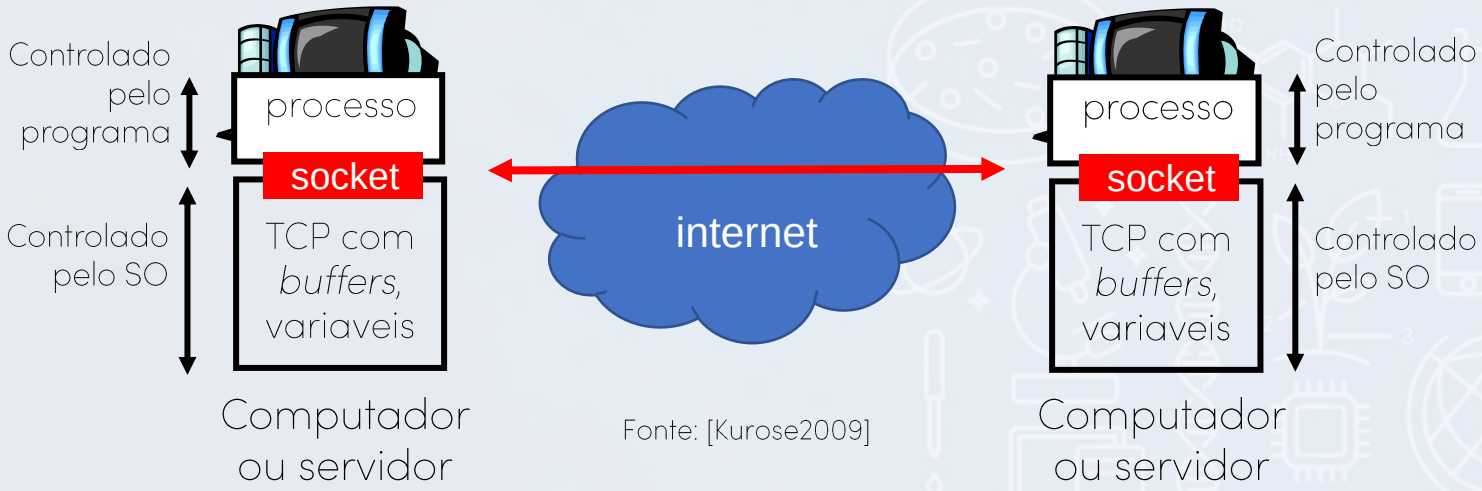
SOCKET

Interface ***criada pela aplicação mas controlada pelo S.O.*** (uma "porta") através da qual os processos podem enviar e receber mensagens de/para outros processos

Sockets TCP

- Socket: porta entre processos

- Serviço TCP:
  - **Ligação** estabelecida antes da comunicação
  - Geralmente oferece **garantias de fiabilidade**
  - Os pacotes são **recebidos na ordem** em que são enviados (ordem FIFO)



Programação com Sockets TCP

1. O **servidor** em execução cria uma **socket de escuta** (*listening socket*) e espera contactos de clientes

2. O **cliente** contacta o servidor:
   - Cria **socket** TCP local (especifica o endereço IP e o porto do processo servidor)
   - O cliente **estabelece a ligação TCP** entre os dois (*3-way handshake*)

3. Ao ser contactado, o **servidor** cria uma **nova socket TCP para a ligação** (*connection socket*), para através dele comunicar com o cliente

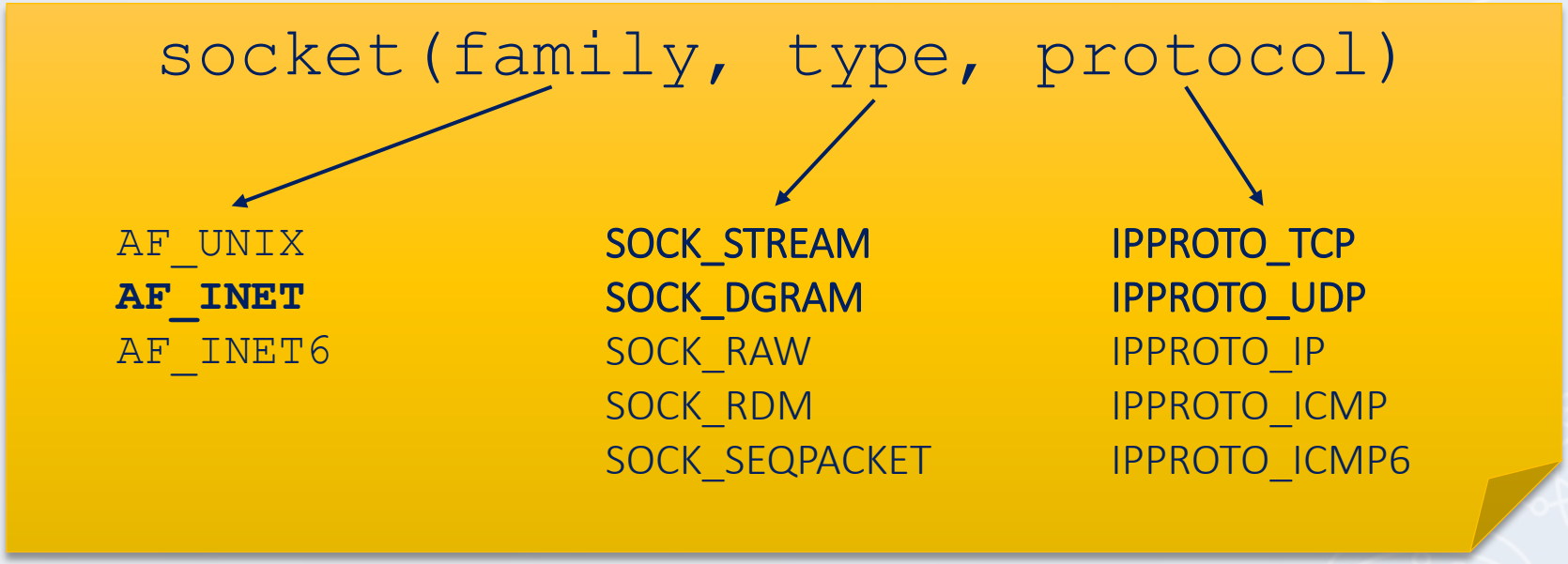
4. A ***listening socket*** fica **livre** e permite receber contactos de outros clientes

Para a aplicação:

TCP oferece um serviço fiável, transferindo bytes ordenadamente (como um "pipe") entre o cliente e o servidor.

2. Sockets em Python

- A interface Python é idêntica à interface C no UNIX
- Está disponível no módulo **`socket`**
- A função `socket` deste módulo devolve um objeto da classe `socket`
- Os métodos desse objeto implementam as chamadas ao sistema



Sockets em Python

- Utilizações mais frequentes para o desenvolvimento de aplicações

```
import socket as s
s.socket(s.AF_INET, s.SOCK_STREAM)
s.socket(s.AF_INET, s.SOCK_DGRAM)
```

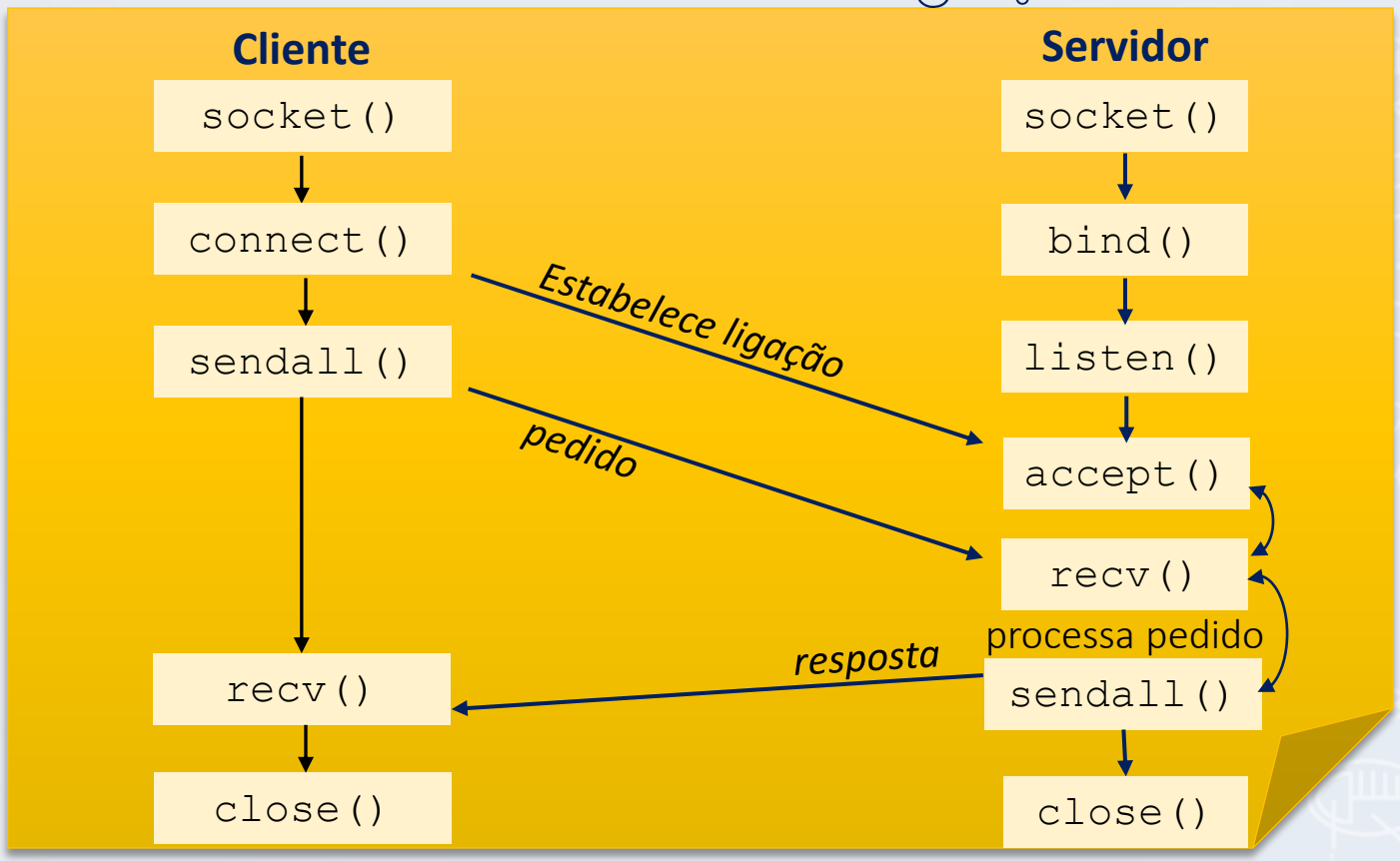
→ Vai usar IPPROTO_TCP

→ Vai usar IPPROTO_UDP

- O parâmetro *protocol*, é determinado pelo sistema em função dos restantes parâmetros

Sockets TCP em Python – Visão Geral

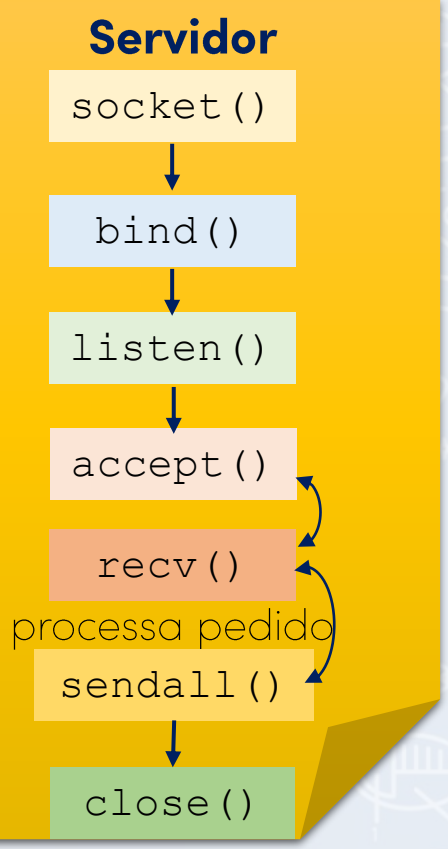
Cliente/Servidor com ligação TCP



Sockets TCP em Python – Servidor.py

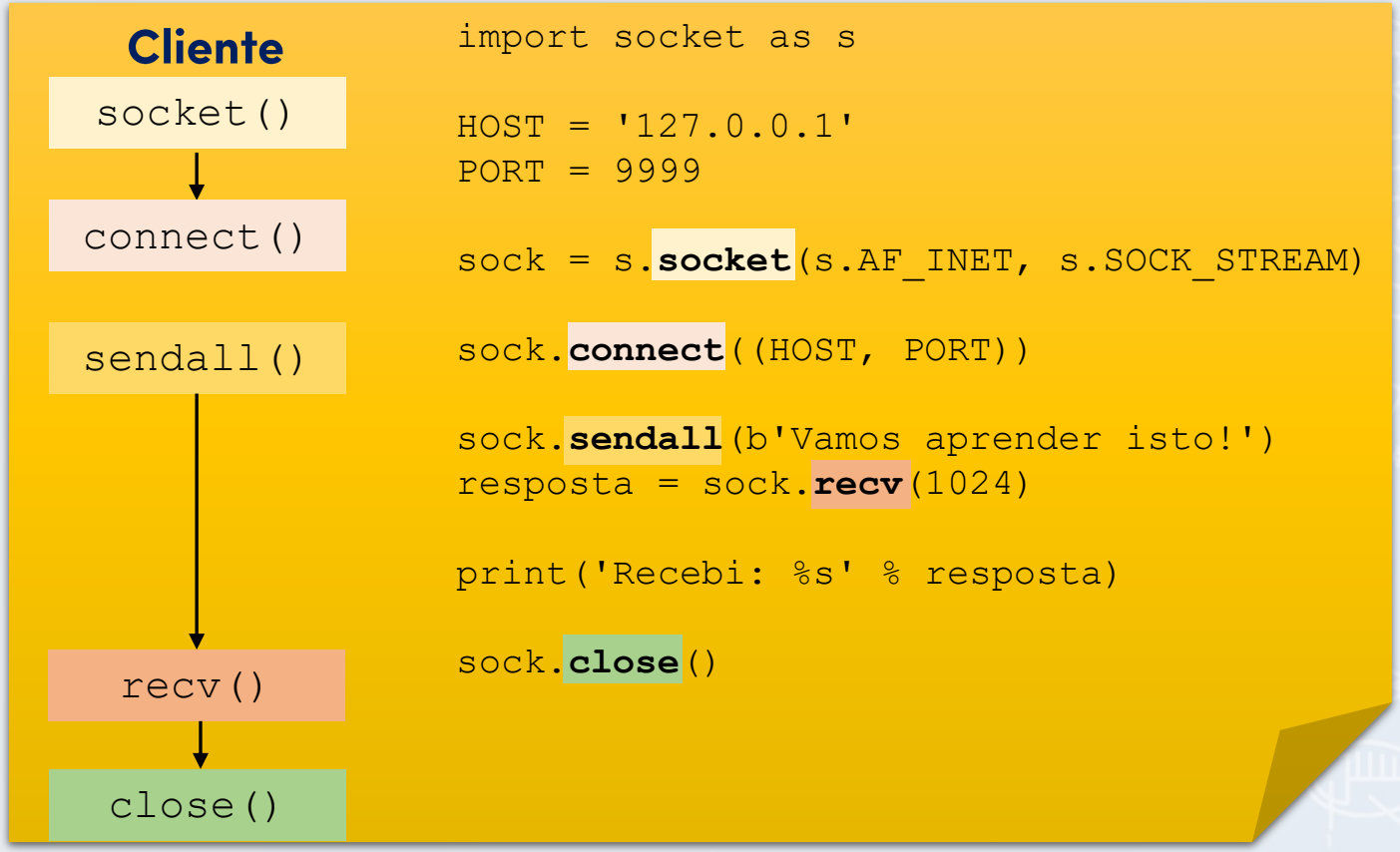
Servidor com ligação TCP em Python

```
import socket as s

HOST = '127.0.0.1'
PORT = 9999

sock = s.socket(s.AF_INET, s.SOCK_STREAM)

sock.bind((HOST, PORT))
sock.listen(1)
(conn_sock, (addr,port)) = sock.accept()

print('ligado a %s %s' % (addr,port))

msg = conn_sock.recv(1024)
print('recebi: %s' % msg)
conn_sock.sendall(b'Aqui vai a resposta')

sock.close()
```



Sockets TCP em Python – Cliente.py

Cliente com ligação TCP em Python



Sockets em Python – Limitações

- SO **pode não** conseguir **enviar tudo** o que queremos:
  - `socket.send()`
    pode não entregar todos os bytes ao SO
    - devolve o número de bytes efetivamente entregues
  - `socket.sendall()`
    resolve este problema

- TCP divide o *stream* em pacotes mais pequenos
  - `socket.recv()`
    pode não ler todos os bytes até ao limite indicado;
  - para **ler exatamente *n* bytes**,
    temos de **fazer um ciclo** até serem lidos os *n* bytes
    através de chamadas sucessivas a `socket.recv()`

Sockets em Python – Limitações

- Em **execuções sucessivas** do **servidor** na mesma máquina, o `socket.bind()` pode gerar o seguinte **erro**:

```
"OSError: … Address already in use"
```

- Isso ocorre porque o **SO não liberta** imediatamente os recursos da **socket de ligação** esperando que o fim da ligação (i.e., pacotes FIN) possa ainda ocorrer
- **Solução**: utilizar a opção **`SO_REUSEADDR`** (antes de `bind`) para indicar que a aplicação atual aceita uma porta onde ligações anteriores ainda podem estar a finalizar

```
import socket as s
…
sock.setsockopt(s.SOL_SOCKET, s.SO_REUSEADDR, 1)
sock.bind((HOST, PORT))
```

3. Exceções em Python – Revisão rápida

- Quando ocorre um **erro** durante uma execução em Python, é gerada uma **exceção**

- **Se** a exceção **não for tratada**, o **programa** é **terminado**
  e é apresentada uma mensagem de erro

```
>>> (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

- Existem muitas **exceções pré-definidas**, cujo tipo é apresentado na última linha do erro

- Além das exceções pré-definidas,
  podemos **definir novas exceções** através da classe **`Exception`**

Exceções em Python – Tratamento

- A instrução *try*

```
try:
    # bloco de código A:
    # onde podem ocorrer erros
except TipoDeExceção:
    # bloco de código para tratar
    # a exceção TipoDeExceção
except:
    # bloco de código para tratar
    # outras exceções
else:
    # bloco de código que executa
    # se o bloco A não gerar
    # exceções
finally:
    # bloco de código que executa
    # após try/except's/else
```

- Exemplo

```
from sys import argv

try:
  print(int(argv[1])/int(argv[2]))
except ZeroDivisionError as e:
  print("Divisão por zero!")
  print(e)
except:
  print("Ocorreu um erro!")
else:
  print("Não houve erros!")
finally:
  print("Vou limpar a casa!")
```

```
Executar: python3 exemplo.py 1 0
          python3 exemplo.py 1 A
          python3 exemplo.py 1 2
```

Tratamento de Exceções – Algumas Abordagens

- Na escrita de uma **biblioteca** com acessos à rede?

    1. Não se tratam exceções:
       o programador que usa a biblioteca tratá-las-á ao nível da aplicação

    2. Levantar exceções geradas pela biblioteca:
       o programador verá uma exceção relacionada com a tarefa
       e não com detalhes de nível mais baixo (e.g., *sockets*, ficheiros)

Tratamento de Exceções – Algumas Abordagens

- Na escrita de um **programa** com acessos à rede?

    1. Colocar **cada chamada** à rede **numa** instrução ***try/except***.
       Pode tornar-se difícil em programas grandes
       e ser pouco informativo para o utilizador.

    2. Colocar **blocos de código** maiores,
       que desempenham uma tarefa específica,
       numa instrução ***try/except***.
       No *except*:
        - Mostrar mensagem adequada à tarefa
        - Gerar outra exceção para ser tratada
          numa parte mais geral do programa

4. Erros no Módulo `socket`

- O módulo **`socket`** pode gerar várias exceções:
  - `socket.`**`error`**
    Quando o erro é relacionado com a *socket*
  - `socket.`**`herror`**
    Quando o erro é relacionado com algumas funções relativas a resolução de nomes (e.g., **`gethostbyname()`**)
  - `socket.`**`gaierror`**
    Quando o erro é relacionado com as funções **`getaddrinfo()`** e **`getnameinfo()`**
  - `socket.`**`timeout`**
    Quando ocorre um *timeout* numa *socket* onde previamente foi chamada **`settimeout(x)`**

Erros no Módulo socket – Exemplo

```
import socket as s
from sys import argv, exit, stdout

sock = s.socket(s.AF_INET, s.SOCK_STREAM)
try:
    sock.connect((argv[1], 22))
except s.error as err:
    print("ERRO: ", err)
    exit(1)

sock.settimeout(5)
try:
    dados = sock.recv(1024)
    print("Recebi: ", dados)
    stdout.flush()
    dados = sock.recv(1024)
except s.timeout as st:
    sock.settimeout(None)
    print("Sem dados: ", st)

sock.close()
```

```
Executar: python3 exemplo.py blah
          python3 exemplo.py localhost
```

Bibliografia

- Brandon Rhodes and John Goerzen.
  Foundations of Python Network Programing, third edition, Apress.

- Python online documentation: socket — Low-level networking interface.
  (https://docs.python.org/3/library/socket.html)

- Python online documentation: Errors and Exceptions.
  (https://docs.python.org/3/tutorial/errors.html)

- Python online documentation: Built-in Exceptions
  (https://docs.python.org/3/library/exceptions.html#bltin-exceptions)