Universidade Tecnológica Federal do Paraná
Departamento Acadêmico de Computação
Bacharelado em Ciência da Computação

Sistemas Distribuídos

Um pouco mais sobre Sockets

Prof. Rodrigo Campiolo

20/09/22

Tópicos

- Introdução
- Modelos de E/S
- Sockets em Java
- Websockets

Introdução

- Definições
  - **Bloqueante**: propriedade de uma operação E/S que aguarda (bloqueia) o término da E/S.
  - **Não-bloqueante**: propriedade de uma operação E/S que não aguarda (não bloqueia) durante a execução da E/S.

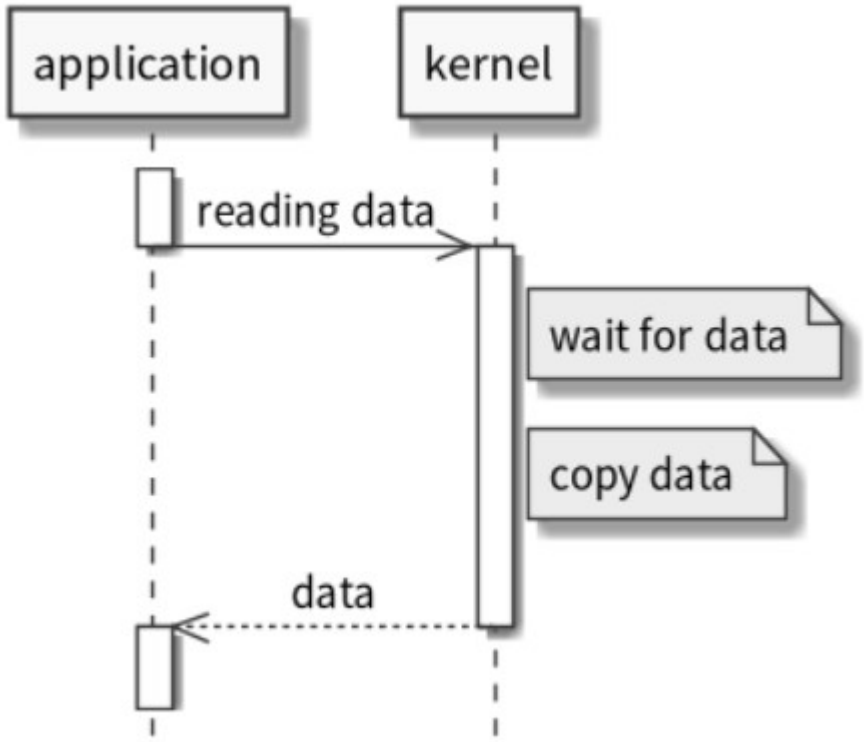
  - **E/S Síncrona**: bloqueia a tarefa até que a operação E/S seja completada.
  - **E/S Assíncrona**: não bloqueia a tarefa após a solicitação E/S, ou seja, ambas podem executar concorrentemente.

Modelos de E/S

- Modelos em conformidade com POSIX:
  - E/S síncrona bloqueante
  - E/S síncrona não-bloqueante
  - E/S assíncrona boqueante
  - E/S assíncrona não-bloqueante

Modelos de E/S

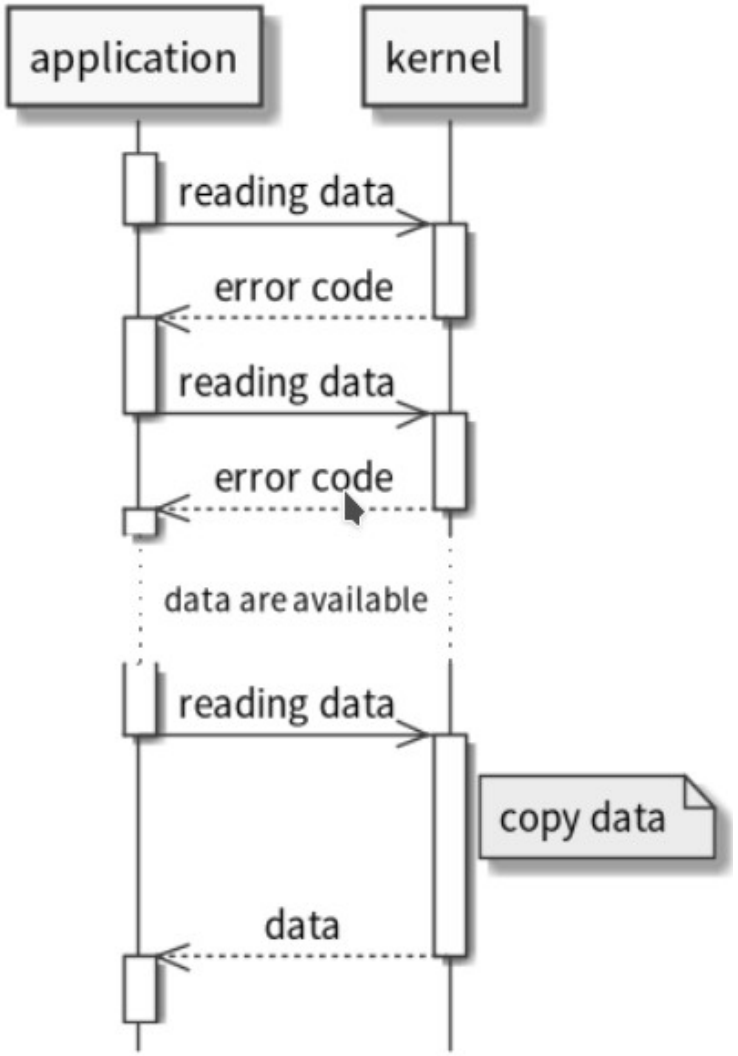
- E/S Síncrona Bloqueante
  - Modelo simples.
  - Padrão E/S.
  - Tarefa é bloqueada.



Fonte: LIAKH, 2020

Modelos de E/S

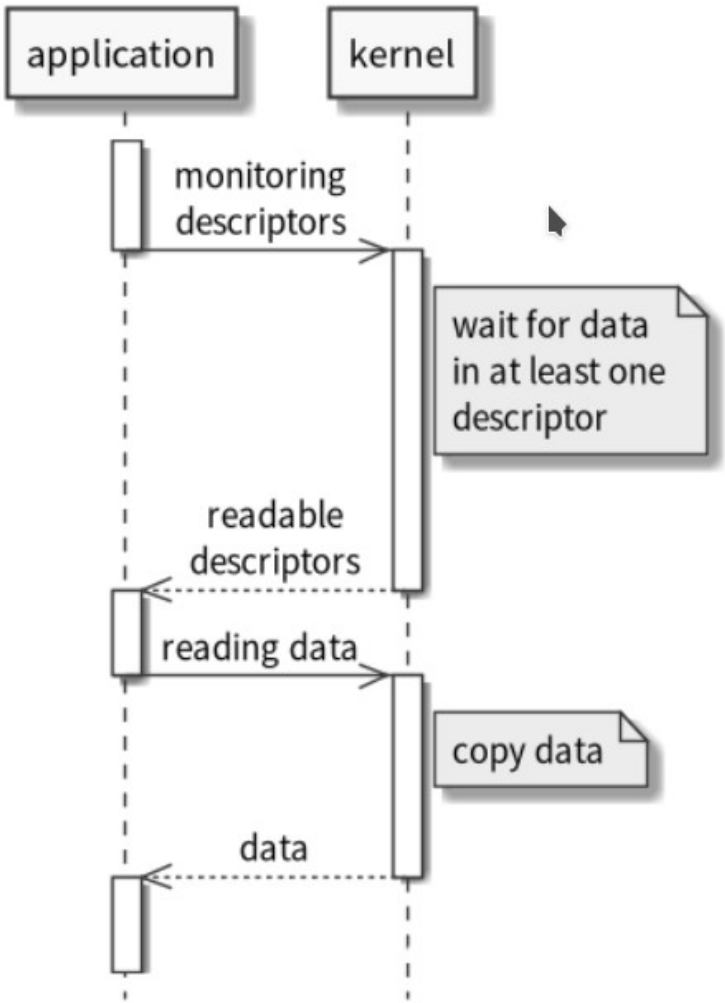
- E/S Síncrona Não-bloqueante
  - Tarefa não-bloqueada.
  - Precisa de verificação de término (*polling*).
  - Latência de E/S.



Fonte: LIAKH, 2020

Modelos de E/S

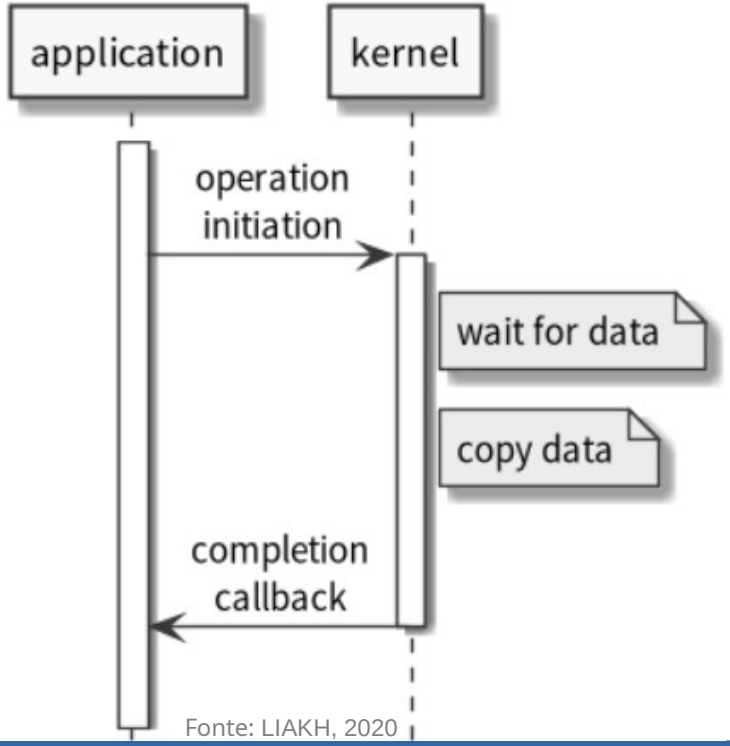
- E/S Assíncrona Bloqueante
  - Denominada também de E/S não-bloqueante com notificação bloqueante.
  - Tarefa realiza E/S em vários descritores.
  - Faz uso de multiplexação por meio da chamada de sistema *select*.



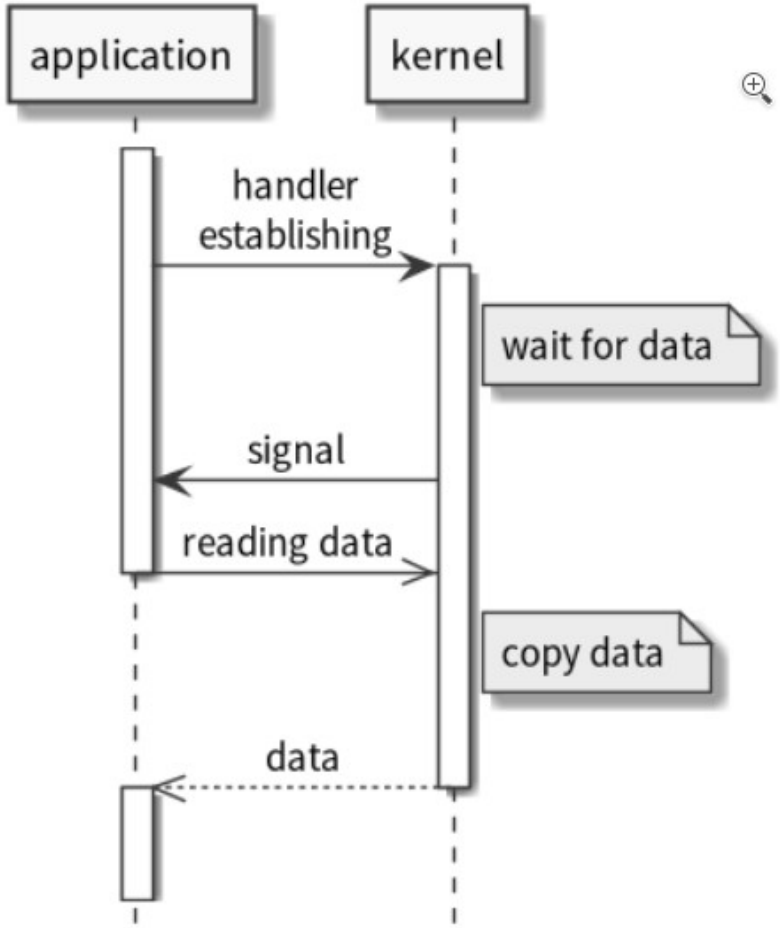
Fonte: LIAKH, 2020

Modelos de E/S

- E/S Assíncrona Não-bloqueante
  - Tarefa não bloqueada.
  - Pode ser implementada por *sinais* ou *callbacks*.
  - Desempenho.



Fonte: LIAKH, 2020



Fonte: LIAKH, 2020

Sockets em Java

- APIs
  - Java IO e Java Networking
    - Operações bloqueantes baseadas em *streams.*
  - Java NIO
    - Operações não bloqueantes.
    - Faz uso de canais, buffers e seletores.
  - Java NIO2
    - Operações assíncronas não bloqueantes.
    - Novas classes para manipular sockets.

Java IO e Networking

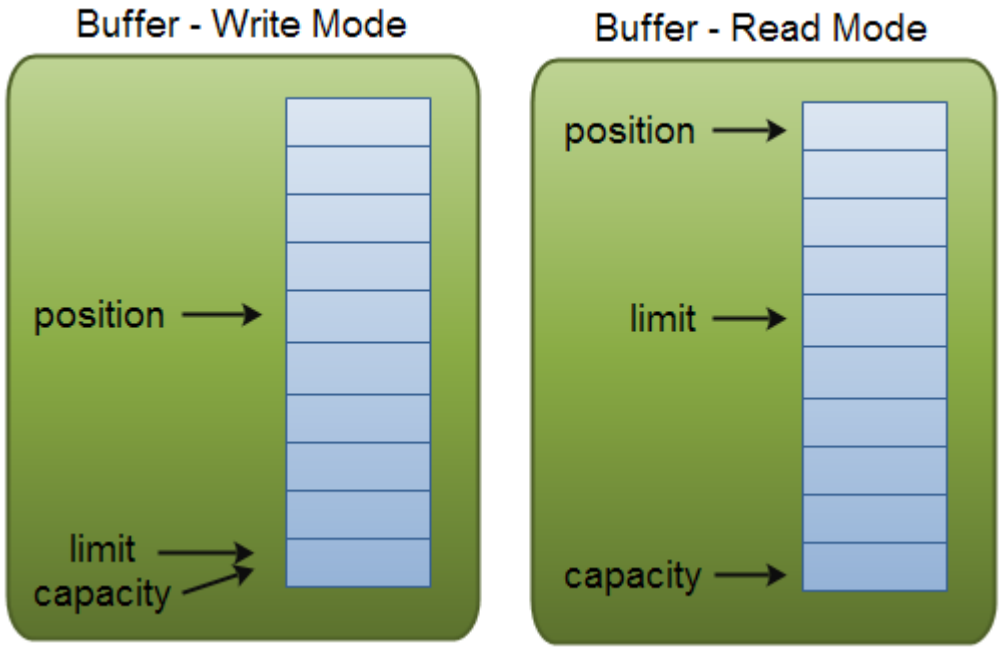
- Principais classes
  - *InputStream* e *OutputStream*
    - Leitura e escrita de dados nos sockets por meio de operações bloqueantes.
  - *Socket* e *ServerSocket*
    - Estabelecer conexões entre clientes e servidores.
    - Possibilitar a comunicação em rede usando operações bloqueantes.
  - *InetAddress*
    - Representação de endereço IP.

Java NIO

- Principais classes
  - *Buffer*
    - Bloco de memória para escrita e leitura de dados.
    - Propriedades
      - *capacidade*: número máximo de elementos.
      - *limite*: limita o último elemento.
      - *posição*: próximo elemento a ser lido ou escrito.
    - Operações
      - *clear()*: limite = capacidade e posição = 0 (escrita).
      - *flip()*: limite = posição e posição = 0 (leitura).
      - *rewind()*: posição = 0 (releitura) .
      - *compact()*: copia elementos não lidos para início do buffer (escrita).
      - *allocate()*: alocar um buffer.

Java NIO

- Principais classes
  - *Buffer*



Fonte: JENKOV, 2014

Java NIO

- Principais classes
  - *Channel*
    - Representa um canal entre entidades relizar operações de leitura e escrita.
    - Possibilita operações bloqueantes e não-bloqueantes.
    - *Channel* lê dados para *Buffer.*
    - *Channel* escreve dados de *Buffer*.
  - *SocketChannel* e *ServerSocketChannel*
    - Comunicação TCP usando canais.
    - *configureBlocking()*: método para definir se operações são bloqueantes ou não-bloqueantes.

Java NIO

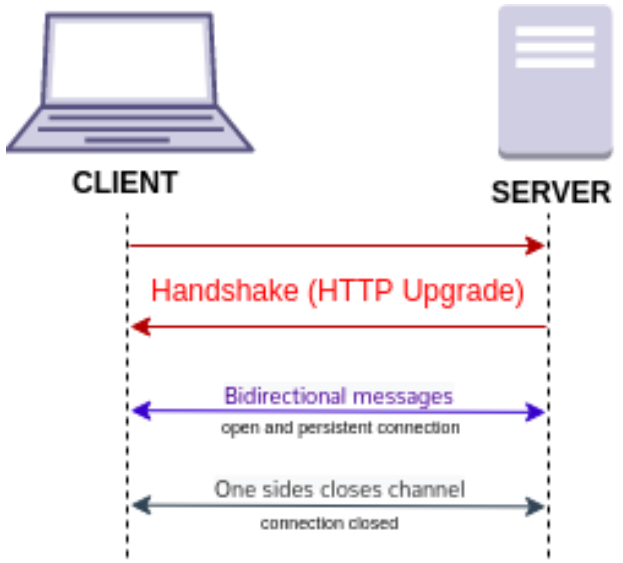
- Principais classes
  - *Selector*
    - Gerencia múltiplos canais com uma única tarefa.
    - Inspeciona canais e dispara eventos: leitura, escrita, aguardar conexões, conectar.
    - Para usar com canais é necessário registrar o canal com o seletor.
    - Operações
      - *select()*: possibilita selecionar o canal; operação bloqueante.

Java NIO2

- Principais classes
  - *AsynchronousSocketChannel e AsynchronousServerSocketChannel*
    - Canal assíncrono para comunicação por fluxo.
    - Podem ser usados com *Future* (representa um resultado de uma operação assíncrona) ou *CompletionHandler* (processa resultados de uma operação assíncrona).

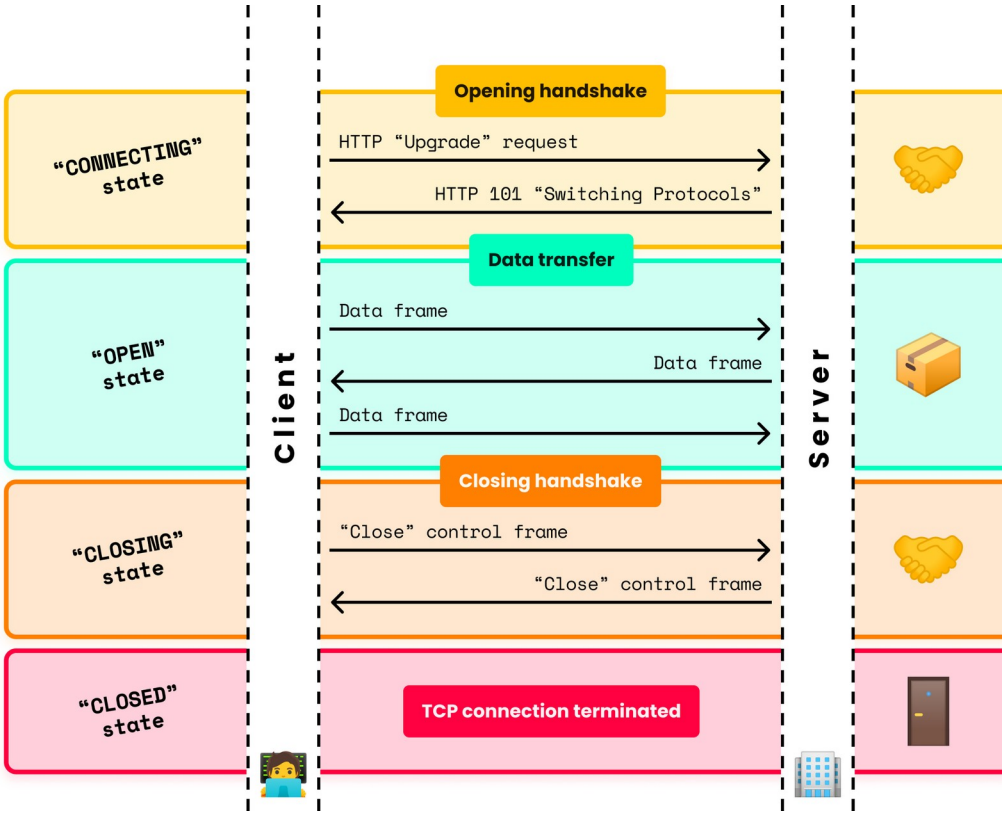
WebSocket

- Possibilita a comunicação bidirecional entre o navegador do usuário com o servidor Web por meio do protocolo TCP.
- O protocolo é definido no RFC 6455
  - *Handshake*
  - Transferência de dados
  - Finalização do canal



Fonte: https://en.wikipedia.org/wiki/WebSocket

WebSocket




```
GET /chat HTTP/1.1
Host: server.example.com
Connection: upgrade
Upgrade: websocket
Origin: <http://example.com>
Sec-WebSocket-Key: NnRlZW4gYnl0ZXMgbG9uZw==
Sec-WebSocket-Protocol: html-chat, text-chat
Sec-WebSocket-Version: 13
```

```
HTTP/1.1 101 Switching Protocols
Connection: upgrade
Upgrade: websocket
Sec-WebSocket-Accept: 5TJpHv9RoAl7w8ytsXcWxTO
Sec-WebSocket-Protocol: new-chat
```

OPCODE (4 bits)

- 0x0: quadro continuação
- 0x1: quadro texto
- 0x2: quadro binário
- 0x9: quadro ping
- 0xA: quadro pong

Fonte: MAGRINI, 2021

Referências

LIAKH, Aliaksandr. **Java sockets I/O: blocking, non-blocking and asynchronous**. 2020. Disponível em *https://liakh-aliaksandr.medium.com/java-sockets-i-o-blocking-non-blocking-and-asynchronous-fb7f066e4ede*. Acessado em 16/09/2022.

JONES, M. **JonesBoost application performance using asynchronous I/O**. IBM Developer. 2006. Disponível em https://developer.ibm.com/articles/l-async/. Acessado em 16/09/2022.

JENKOV, Jakob. Java NIO Buffer. 2014. Disponível em https://jenkov.com/tutorials/java-nio/buffers.html. Acessado em 16/09/2022.

MAGRINI, Damiano. WebSockets Demystified, Part 1: Understanding the Protocol. Level up Coding. 2021. Disponível em https://levelup.gitconnected.com/websockets-demystified-part-1-understanding-the-protocol-fccca2ca75eb. Acessado em 17/09/2022.