# Sistemas Distribuídos

## Comunicação entre Processos

março de 2021

# Comunicação entre Processos

- troca de mensagens é sempre a primitiva básica

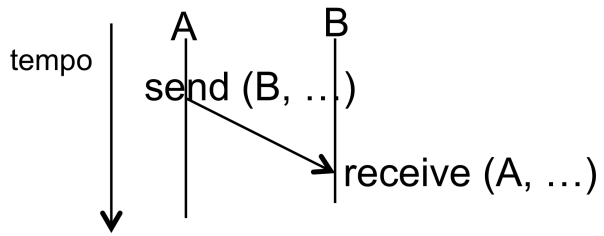
- sobre essa primitiva podemos construir outras visões da comunicação

# Mensagens Básicas

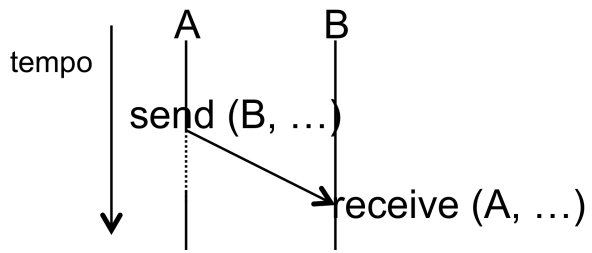
```
send (destino, &mensagem)

receive (origem, &mensagem)
```

- questões
  - semântica de operações
  - especificação de origem e destino
  - formato da mensagem

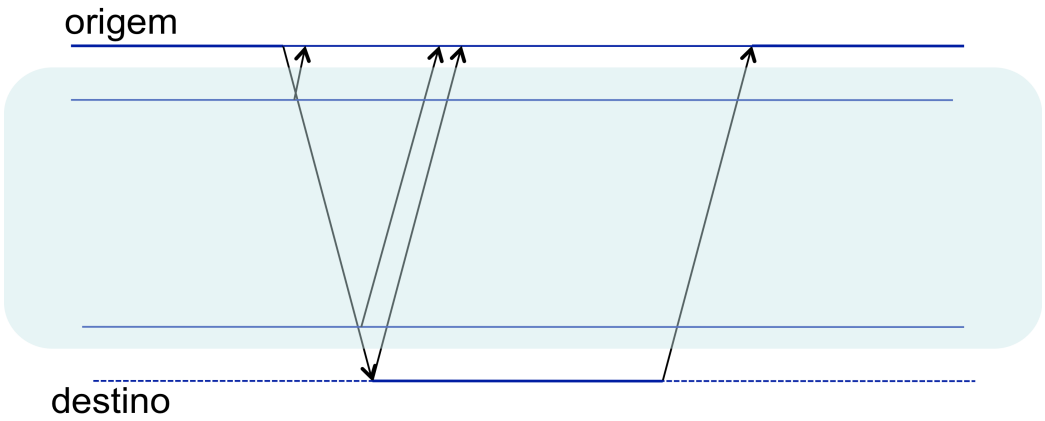
- envio assíncrono: execução procede imediatamente
  - bufferização




- envio síncrono: execução procede quando destinatário recebe mensagem
  - determinismo



# Variantes síncronas e assíncronas




algumas bibliotecas oferecem primitivas com várias diferentes semânticas de garantia de envio

# recebimento síncrono e assíncrono

- recebimento síncrono é o convencional
  - execução procede quando há algo a tratar
  - alternativa de recebimento com timeout

**bloqueio!**

suspensão da linha de execução corrente até chegada da mensagem

# bloqueio e recebimento explícito

- espera por mensagens de tipos específicos?
  - podem haver outros tipos de mensagens pendentes
- espera por qualquer tipo de mensagem
  - código se torna um enorme emaranhado

# recebimento assíncrono

- orientação a eventos: chegada de mensagem é encarada como evento a ser tratado
  - recebimento fica implícito

# Especificação de destino

- endereçamento direto (acoplamento)
  - endereços e volatilidade
  - serviços de nomes
- caixas de correios e canais (desacoplamento)
  - endereços bem conhecidos
  - comunicação com par "qualquer"

# Formato de mensagens

- no nível mais baixo: sequências de bytes
  - interpretação por conta do programa

# Formato de mensagens

- apoio de bibliotecas ou linguagens a empacotamento e desempacotamento
  - marshalling e unmarshalling
- abstrações em vários níveis

# Comunicação no SO: a API de sockets

- comunicação “crua”, com baixo nível de abstração
- API possibilita acesso a uma série de protocolos
  - TCP
  - UDP
  - ... mas muitos outros

# Comunicação no SO: a API de sockets

- comunicação "crua", com baixo nível de abstração
- API possibilita acesso a uma série de protocolos
  - TCP
  - UDP
  - ... mas muitos outros
- abstração em bibliotecas padrão é a de arquivos
- mensagens são sequência de bytes

# TCP: padrão cliente-servidor

- servidor sempre a espera de comunicação
  - em programas convencionais: bloqueado
- cliente sempre inicia a comunicação
  - assimetria: cliente deve saber localizar servidor
  - envio de requisição associado a um ou mais recebimentos

**pseudo-servidor**

```
while (true) do {
  aguarda requisição de qualquer cliente C
  processa requisição
  responde a cliente C
}
```

# exemplo servidor muito simples – Lua e luasocket

```
-- load namespace
local socket = require("socket")
-- create a TCP socket and bind it to the local host, at any port
local server = assert(socket.bind("*", 0))
-- find out which port the OS chose for us
local ip, port = server:getsockname()
-- print a message informing what's up
print("Please telnet to localhost on port " .. port)
print("After connecting, you have 10s to enter a line to be echoed")
-- loop forever waiting for clients
while 1 do
  -- wait for a connection from any client
  local client = server:accept()
  -- make sure we don't block waiting for this client's line
  client:settimeout(10)
  -- receive the line
  local line, err = client:receive()
  -- if there was no error, send it back to the client
  if not err then client:send(line .. "\n") end
  -- done with client, close the object
  client:close()
end
```

# Tarefas

- instalar luasocket e criar par de processos cliente/servidor simples (pode ser o exemplo do eco com um cliente criado por vocês)
- ler páginas 173–184 do livro do Steen (RPC)