

Observed.notify  
Observer.update

observed.notify do Protocol muito  
provavelmente da classe Concurrent\_Observed

**Concurrent\_Observer**

**Concurrent\_Observed**  
+notify(buf : Buffer<Ethernet::Frame>\*, prot : Ethernet::Protocol) : boolean

**Conditional\_Data\_Observer**

**Conditional\_Data\_Observed**

**Engine**  
+receive(src : Address\*, prot : Protocol\_Number\*, data : void\*, size : unsigned int) : int

**Ethernet**

**Message**

**Communicator**  
+receive(message : Message\*) : boolean  
+update(obs : Channel::Observed\*, c : ...  
+send(message : Message) : boolean

**Protocol**  
+receive(buf : Buffer\*, from : Address, d...  
+update(obs : NIC::Observed\*, prot : NI...  
+send(from : Address&, to : Address&, d...

**NIC**  
+receive(buf : Buffer\*, src : Address\*, dst : Address\*, data : void\*, size : unsigned int) : unsigned int  
+free(buf : Buffer\*) : void  
+handle\_signal()  
+notify(buf : Buffer<Ethernet::Frame>\*, prot : Ethernet::Protocol) : boolean  
+alloc(dst : Address, prot : Protocol\_Number, size : unsigned int)  
+send(buf : Buffer\*) : int

Communicator::Observer = Concurrent\_Observer<typename Channel::Observer::Observed\_Data,  
typename Channel::Observer::Observing\_Condition>  
Communicator::Buffer = Channel::Buffer  
Communicator::Address = Channel::Address

**Address**

**Header**

**Packet**

**Buffer**

NIC::Observer = Conditional\_Data\_Observer<Buffer<Ethernet::Frame>, Ethernet::Protocol>  
NIC::Observed = Conditionally\_Data\_Observed<Buffer<Ethernet::Frame>, Ethernet::Protocol>  
NIC::Buffer = Buffer<Ethernet::Frame>  
NIC::Address = Ethernet::Address  
NIC::Protocol\_Number = Ethernet::Protocol

Protocol::PROTO = NIC::Protocol\_Number  
Protocol::Buffer = NIC::Buffer  
Protocol::Physical\_Address = NIC::Address  
Protocol::Port = XXX  
Protocol::Observer = Conditional\_Data\_Observer<Buffer<Ethernet::Frame>  
Protocol::Observed = Conditionally\_Data\_Observed<Buffer<Ethernet::Frame>

# Responsabilidades dentro da arquitetura

## Hierarquia dos componentes:

Engine -> NIC -> Protocol -> Communicator

- **NIC**

Fachada sobre a Engine. Responsável por:

- Gerenciar um pool de objetos Buffer (`_buffer_pool`) que encapsulam a memória alocada pela Engine.
- Orquestrar o fluxo de envio (`send`) e recepção (`handle_signal`).
- Implementar o padrão *Observed* para despachar frames recebidos aos Protocols corretos baseado no EtherType (`notify`).
- Manter estatísticas (`_statistics`).

- **Engine**

- Interação direta com Socket.
- Alocação/Liberação de memória bruta para os frames (`allocate_frame_memory`, `free_frame_memory`).
- Configuração de interface via `ioctl` e filtragem inicial (BPF).
- Configuração de notificação de eventos de I/O via Sinais (SIGIO).

- **Protocol / Communicator**

- Camadas superiores que lidam com a lógica específica do protocolo, demultiplexação por porta e API para a aplicação.

# Responsabilidades dentro da arquitetura

- **Protocol**

- Herda de NIC::Observer (Conditional\_Data\_Observer) para receber frames da NIC via update.
- Atua como Observed (Concurrent\_Observed) para notificar Communicators baseado na porta (lógica interna).
- Define formato do pacote (Protocol::Packet) dentro do payload Ethernet.

- **Communicator**

- Fornece API send/receive síncrona (bloqueante no receive) para a aplicação.

- **Buffers**

- Buffer encapsula ponteiro (data()) e metadados (size, in\_use), não gerencia memória diretamente.

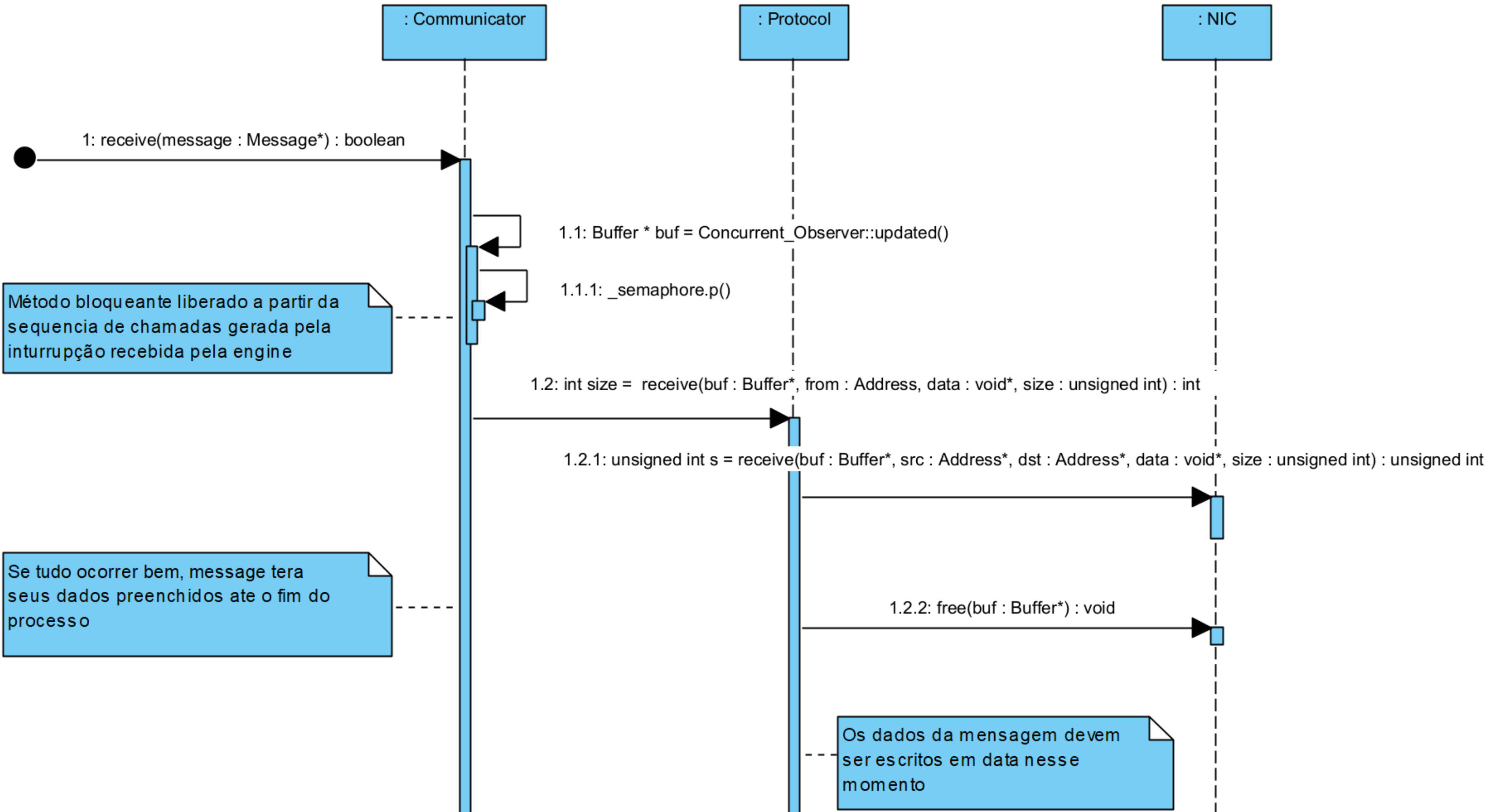
# Observers

- **NIC (Observed) -> Protocol (Observer)**

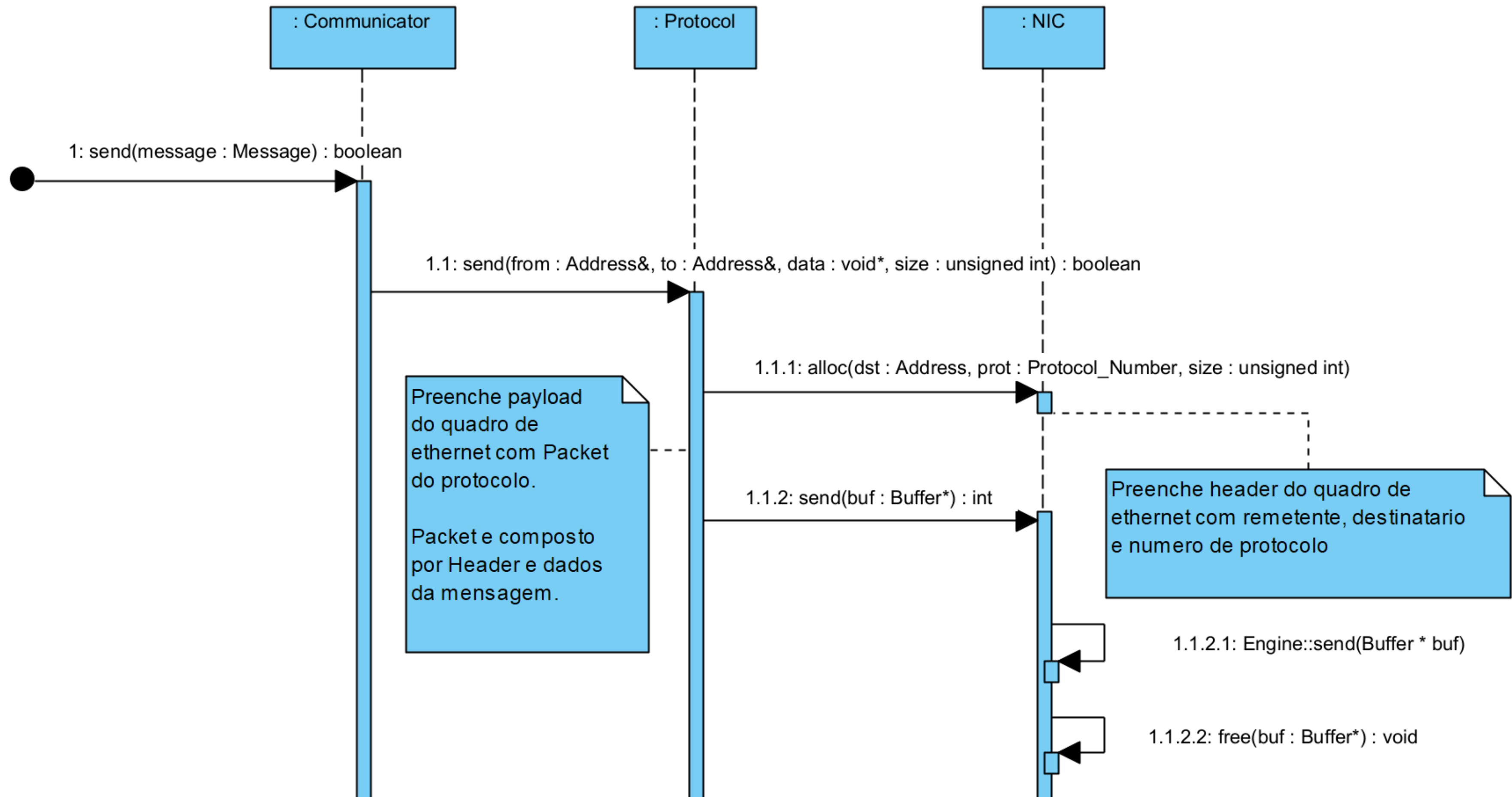
- NIC herda de `Conditionally_Data_Observed<BufferNIC, Ethernet::Protocol>`: Atua como a entidade observada.
- Protocol herda de `NIC::Observer (Conditional_Data_Observer<...>)`: Atua como o observador.
- **Registro**: Protocol chama `_nic->attach(this, PROTO)` no construtor, registrando interesse no seu `EtherType` específico (PROTO).
- **Notificação**: `NIC::handle_signal` chama `this->notify(ether_type, buf)`. `notify` encontra o Protocol cujo `rank()` (`EtherType`) corresponde e chama seu método `update(..., ether_type, buf)`.
- Permite que múltiplos protocolos coexistam na mesma interface física, com a NIC direcionando os frames corretamente.

- **Protocol (Observed) -> Communicator (Observer)**

- Protocol herda de `Concurrent_Observed<BufferNIC, Port>`: Atua como observado para seus clientes (Communicators).
- Communicator atua como observador final.
- **Registro**: Communicator chama `_channel->attach(this, port)` no construtor, registrando interesse em uma Port específica.
- **Notificação**: `Protocol::update` (chamado pela NIC) extrai a porta do cabeçalho do protocolo e chama `this->notify(port, buf)`. `notify` encontra o Communicator cujo `rank()` (Porta) corresponde e chama seu `update(..., port, buf)`.
- Usa `Concurrent_Observer` com `std::counting_semaphore` para permitir que `Communicator::receive` bloqueie até a chegada de um pacote.



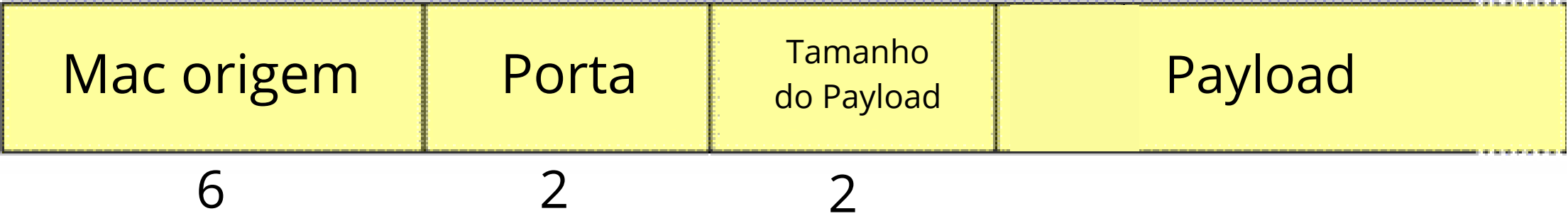
sd [Send]

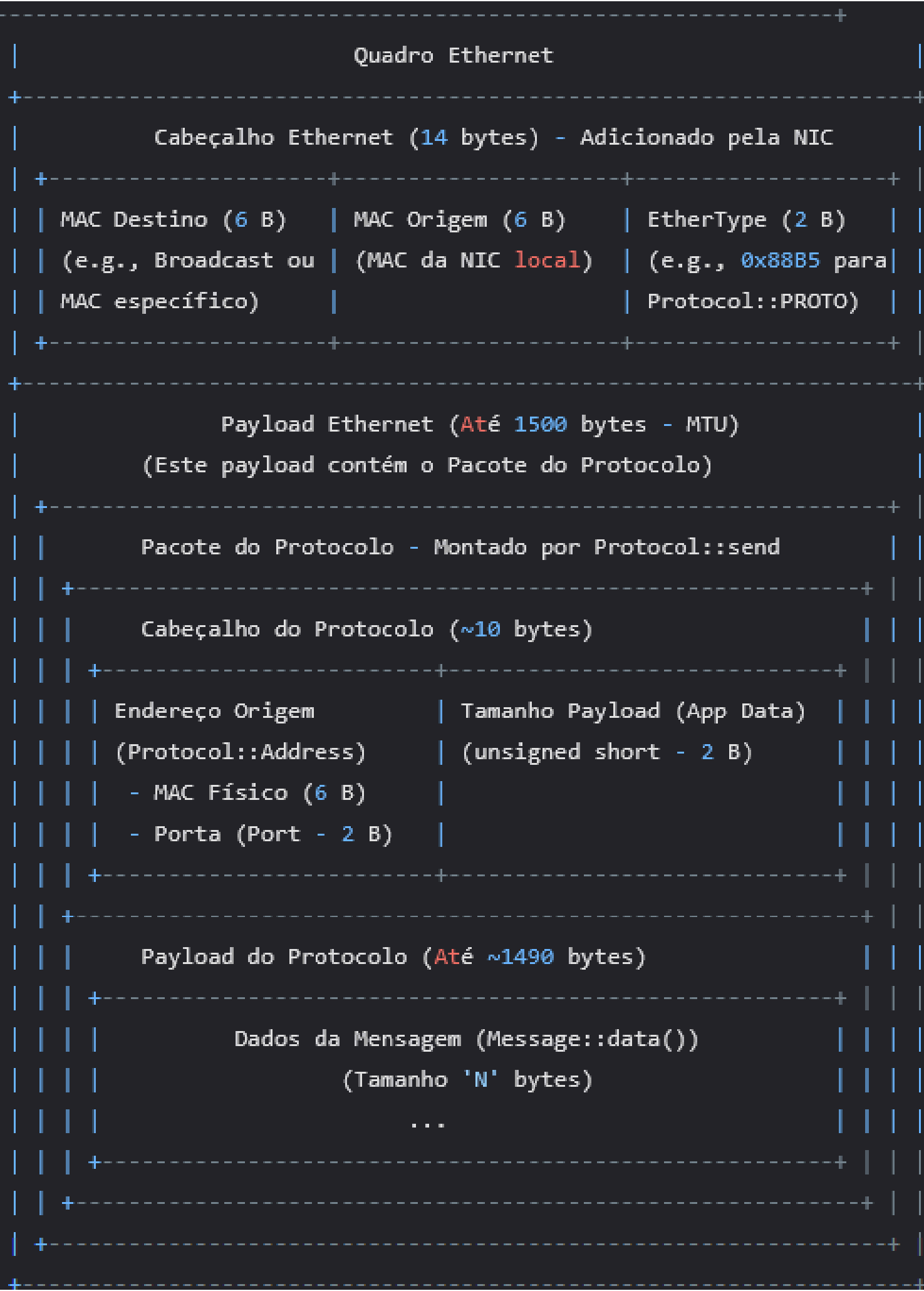


Quadro Ethernet



Packet do Protocolo







## SA\_NODEFER

### ASYNC-SIGNAL SAFETY [top](#)

The mutex functions are not async-signal safe. What this means is that they should not be called from a signal handler. In particular, calling `pthread_mutex_lock` or `pthread_mutex_unlock` from a signal handler may deadlock the calling thread.

[https://man7.org/linux/man-pages/man3/pthread\\_mutex\\_lock.3.html](https://man7.org/linux/man-pages/man3/pthread_mutex_lock.3.html)

### SA\_NODEFER

Do not add the signal to the thread's signal mask while the handler is executing, unless the signal is specified in `act.sa_mask`. Consequently, a further instance of the signal may be delivered to the thread while it is executing the handler. This flag is meaningful only when establishing a signal handler.

<https://man7.org/linux/man-pages/man2/sigaction.2.html>

```
void Engine::setupSignalHandler() {
    // Armazena a função de callback
    struct sigaction sigAction;
    sigAction.sa_handler = Engine::signalHandler;
    sigAction.sa_flags = SA_RESTART;

    // Limpa possíveis sinais existentes antes da configuracao
    sigemptyset(&sigAction.sa_mask);

    // Configura sigaction
    // nullptr indica que nao queremos salvar a sigaction anterior
    if (sigaction(SIGIO, &sigAction, nullptr) < 0) {
        perror("sigaction");
        exit(EXIT_FAILURE);
    }
}
```