// fsm.h

typedef struct fsm\_state {

const char \*name; /\*\*< Nome para debug \*/

uint8\_t id; /\*\*< ID numérico \*/

const fsm\_transition\_t \*transitions;/\*\*< Transições possíveis \*/

uint8\_t num\_transitions; /\*\*< Quantas transições \*/

fsm\_action\_t default\_action; /\*\*< Ação padrão \*/

uint16\_t timeout\_ms; /\*\*< Timeout \*deste\* estado (0=desabilitado) \*/

} fsm\_state\_t;

#define FSM\_STATE(\_name, \_state\_id, \_transitions, \_default\_action, \_timeout) \

{ #\_name, \_state\_id, \_transitions, sizeof(\_transitions)/sizeof(fsm\_transition\_t), \_default\_action, \_timeout }

// fsm.c

extern uint16\_t get\_current\_time\_ms(void);

void fsm\_run(fsm\_t \*fsm) {

if (!fsm || !fsm->current\_state) return;

const fsm\_state\_t \*s = fsm->current\_state;

// 1) se este estado tem timeout configurado, verifica se excedeu

if (s->timeout\_ms > 0) {

uint16\_t now = get\_current\_time\_ms();

uint16\_t elapsed = now - fsm->state\_entry\_time;

if (elapsed >= s->timeout\_ms) {

// limpa fila e dispara evento de timeout

fsm->event\_queue.head = fsm->event\_queue.tail;

fsm\_handle\_event(fsm, FSM\_EVENT\_STATE\_TIMEOUT);

// atualiza timestamp (para não disparar repetidamente)

fsm->state\_entry\_time = now;

}

}

// 2) processamento normal de eventos…

…

// modbus\_server.h ou modbus\_common\_events.h

#define FSM\_EVENT\_STATE\_TIMEOUT 0xFF

// Exemplo: RECEIVING

static const fsm\_transition\_t state\_receiving\_transitions[] = {

FSM\_TRANSITION(MODBUS\_EVENT\_RX\_BYTE\_RECEIVED, modbus\_server\_state\_receiving, action\_start\_receiving, NULL),

…,

// Timeout só aqui:

FSM\_TRANSITION(FSM\_EVENT\_STATE\_TIMEOUT, modbus\_server\_state\_error, action\_handle\_error, NULL)

};

const fsm\_state\_t modbus\_server\_state\_receiving =

FSM\_STATE("RECEIVING", MODBUS\_SERVER\_STATE\_RECEIVING,

state\_receiving\_transitions, action\_start\_receiving,

/\*timeout\_ms=\*/5000);

Para estados onde não quer timeout, basta usar timeout\_ms = 0:

const fsm\_state\_t modbus\_server\_state\_idle =

FSM\_STATE("IDLE", MODBUS\_SERVER\_STATE\_IDLE,

state\_idle\_transitions, action\_idle,

/\*timeout\_ms=\*/0);

Inicialização

void fsm\_init(fsm\_t \*fsm, const fsm\_state\_t \*initial\_state, void \*user\_data) {

…

fsm->current\_state = initial\_state;

fsm->user\_data = user\_data;

fsm->event\_queue.head = fsm->event\_queue.tail = 0;

fsm->state\_entry\_time = get\_current\_time\_ms();

}

// dentro de fsm\_run, quando fizer

fsm->current\_state = transition->next\_state;

fsm->state\_entry\_time = get\_current\_time\_ms();

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

/\*\*

\* @brief Reinicia o Modbus de forma “soft” ou “hard”.

\*

\* - soft: limpa buffers e volta ao estado idle

\* - hard: além de soft, reinicia UART e re‐inicializa a FSM por completo

\*

\* @param ctx Ponteiro para o contexto Modbus.

\* @param hard true para hard reset, false para soft reset.

\*/

static void modbus\_restart(modbus\_context\_t \*ctx, bool hard)

{

// limpa buffers e mensagens

modbus\_reset\_message(ctx);

ctx->raw\_data.rx\_count = ctx->raw\_data.tx\_count = 0;

ctx->raw\_data.rx\_index = ctx->raw\_data.tx\_index = 0;

if (hard && ctx->transport.restart\_uart) {

// reinicialização “hard”

ctx->transport.restart\_uart();

// re‐cria FSM a partir do estado idle

fsm\_init(&ctx->fsm, &modbus\_server\_state\_idle, ctx);

}

// se soft, FSM continua no mesmo state, mas sem eventos pendentes

}

static void action\_handle\_error(fsm\_t \*fsm)

{

modbus\_server\_data\_t \*server = (modbus\_server\_data\_t \*)fsm->user\_data;

modbus\_context\_t \*ctx = server->ctx;

bool hard;

// Decida aqui quais erros exigem hard reset

switch (server->msg.error) {

case MODBUS\_ERROR\_CRC:

case MODBUS\_ERROR\_TRANSPORT:

hard = true;

break;

default:

hard = false;

}

LOG(LOG\_LEVEL\_WARNING,

"Error %d detected, performing %s reset",

server->msg.error,

hard ? "HARD" : "SOFT");

// Executa o reset

modbus\_restart(ctx, hard);

// Por fim, dispara transição “volta ao idle”

fsm\_handle\_event(fsm, MODBUS\_EVENT\_RESTART\_FROM\_ERROR);

}