

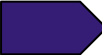
**AMQP BROKER**

# arquivos

1. **amqp.h e amqp.c:** Parsing, unparsing e envio dos pacotes
2. **queue.h e queue.c:** CRUD da estrutura que armazena todos os dados
3. **hardcode.h e hardcode.c:** Auxílio ao amqp.\* fornecendo pacotes prontos do Rabbitmq
4. **main.c:** Broker que usa forks para ser paralelo

# queues\_data: Estruturaração e armazenamento

```
void create structure queues data();  
void initialize structure queues data();  
void free structure queues data();  
void free shared data(void* p, size_t size);  
void* malloc shared data(size_t size);
```

 Alocação de memória

```
void print consumers(int i);  
void print names();  
void print messages(int i);  
void print queues data();
```

 Debugging

```
typedef struct queue t{  
    char** queue name;  
    char*** queue messages;  
    int** queue consumers;  
} queue;
```

 Definição

```
void add queue(char* queue name);  
void publish(char* queue name, char* msg);  
void add consumer(char* queue name, int* connfd);  
int consume(char* queue name, int* connfd, char* msg);  
int get id(char* queue name);  
int move consumer to last position(int i);  
void remove message(int i);
```

 CRUD

# #define

```
#define MAX_QUEUE_SIZE 100
#define MAX_QUEUE_NAME_SIZE 100
#define MAX_MESSAGE_SIZE 100
#define MAX_CONSUMER_NUMBER 100
#define MAX_MESSAGE_NUMBER 100
#define MAXLINE 4096
#define FIXED_CHANNEL 0x1
```

 amqp.h e queues.h

```
enum frame_type_t{
    PROTOCOL    = 0x41,
    METHOD       = 0x01,
    HEADER       = 0x02,
    BODY         = 0x03,
    HEARTBEAT    = 0x08
};
```

```
enum class_type_t{
    CONNECTION = 0xa,
    CHANNEL    = 0x14,
    QUEUE      = 0x32,
    BASIC      = 0x3c,
};
```

```
enum method_type_t{
    Muitos nomes, acredita.
}
```

```
typedef struct frame_t{
    u_int8_t type;
    u_int16_t channel;
    u_int32_t length;
    u_int16_t class;
    u_int16_t method;
} frame;
```

 amqp.h

# comandos amqp

1. **amqp-declare-queue:** Criação da fila em `queues_data`, se ela não existir;
2. **amqp-publish:** Armazenamento da mensagem na fila, se ela existir;
3. **amqp-consume:** Se a fila existir, o cliente é tido como subscriber até que o processo morra;

**OBS:** `send_queue_declare_ok()` e `send_basic-deliver()` são os únicos métodos do `amqp.h` que não utilizaram os pacotes hardcodados do RabbitMQ. Foi necessário criar esses pacotes se atendo à cada detalhe, porque o `frame_length` e outros parâmetros foram alterados com frequência.

# Notas sobre os testes dos cenários

1. Foi criado um container para o broker usando Docker;
2. Os comandos do amqp-tools foram feitos da minha máquina
3. Os dados da CPU e REDE foram obtidos usando docker stats

## **sudo lshw -short**

|               |           |         |   |
|---------------|-----------|---------|---|
| /0/100/1c.4/0 | enp2s0    | network | RTL810xE PCI Express Fast Ethernet controller |
| /0/100/1c.5/0 | wlp3s0    | network | QCA9377 802.11ac Wireless Network Adapter     |
| /0/1c         | processor |         | Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz      |
| /0/0          | memory    |         | 8GiB System memory                            |

# gráficos

