No auxílio à manutenção dos requests foi implementada uma estrutura de dados queue, do tipo fifo para garantir que os pedidos são todos processados por ordem de chegada. Esta implementação por nossa parte deveu-se principalmente ao facto da biblioteca de c não possuir este tipo de estruturas.

srv_utils.h

//estrutura que define um elemento na fila, guardando o seu request respetivo, e o elemento seguinte que está ao seu lado em next.

```
struct queueEl {
 tlv_request_t request;
 struct queueEl *next;
};
typedef struct queueEl queue_el_t;
```

//estrutura que define a fila fila, guardando em head o elemento no início da fila, em tail todos restantes.

```
typedef struct queue {
 queue_el_t *head;
 queue_el_t *tail;
} queue_t;
```

```
//adiciona elemento à fila, criando novo elemento alocando-o na memória
void queuePush (tlv_request_t request);
//remove elemento da frente da fila retornando-o
tlv_request_t queuePop ();
//destroi a fila, percorrendo e libertando a memória que foi alocada, na sua criação
void queueDelete ();
//diz se a fila está vazia
int queueEmpty ();
```

srv_utils.c

```
//fila usada pelo servidor para guardar os pedidos
queue_t requestQueue;

void queuePush (tlv_request_t request) {
 queue_el_t *new;

 new = (queue_el_t *)malloc (sizeof (queue_el_t));
 if (new == NULL)
   return;

 new->request = request;
 new->next = NULL;

 if (requestQueue.tail == NULL) {
   requestQueue.head = requestQueue.tail = new;
 } else {
   requestQueue.tail->next = new;
   requestQueue.tail = new;
 }
}

tlv_request_t queuePop () {
 queue_el_t *first;

 first = requestQueue.head;
 requestQueue.head = first->next;

 if (requestQueue.head == NULL)
   requestQueue.tail = NULL;

 tlv_request_t ret;
 ret = first->request;
 free (first);

 return ret;
```

```
}


void queueDelete () {
 queue_el_t *aux;

 while (requestQueue.head != NULL) {
   aux = requestQueue.head->next;
   free (requestQueue.head);
   requestQueue.head = aux;
 }
}

int queueEmpty () {
 return requestQueue.head == NULL;
}
```