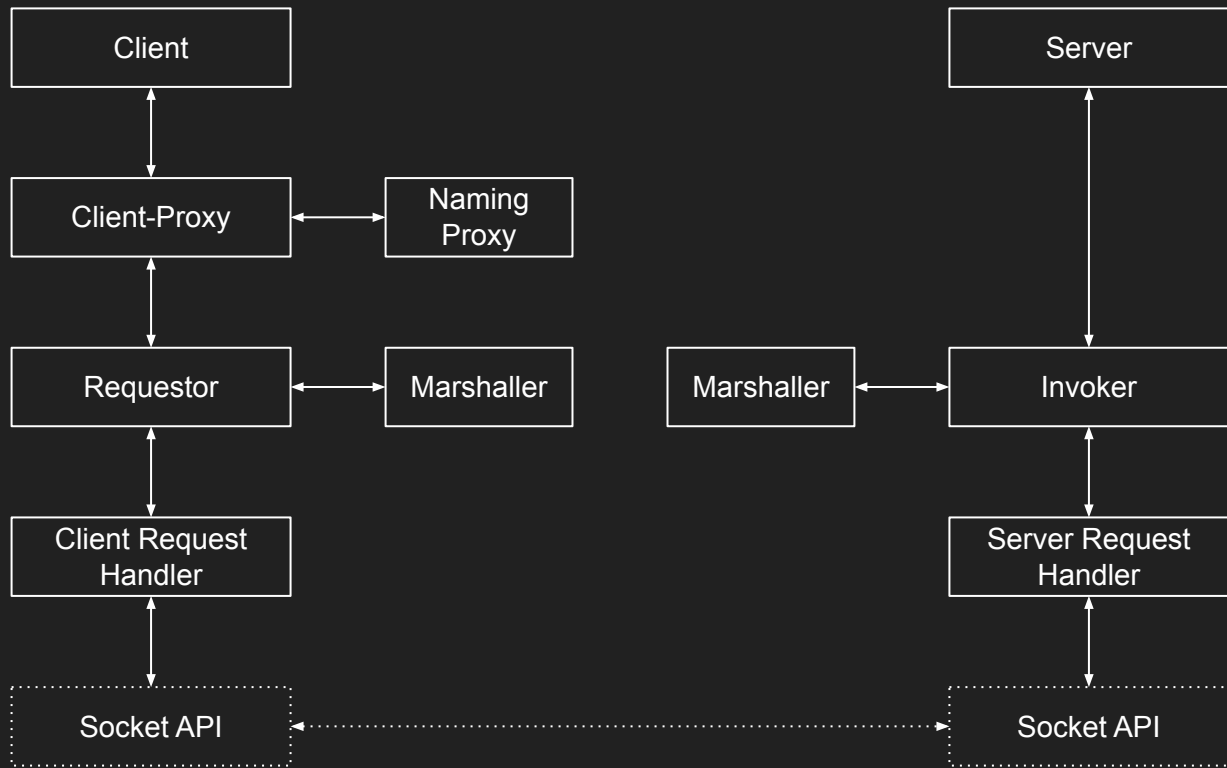


Aplicação Cliente/Servidor usando MyRPC em C



Client

```
1 static void send_request(uint_16 a, uint_16 b) {  
2     int_32 result = 0;  
3  
4     if (calc.add(a, b, &result)) {  
5         /* ... */  
6     } else  
7         die(EXIT_FAILURE, errno, "Calc failed");  
8 }
```

Client Proxy

```
1 static bool calc_invoke(const char *method, const uint_16 a, const uint_16 b, int_32 *const result) {
2     struct data *request, *reply = NULL;
3     struct value reply_value;
4
5     if (requestor != NULL && !requestor_is_active(requestor)) {
6         requestor_destroy(requestor);
7         requestor = NULL;
8     }
9
10    if (requestor == NULL && (host_addr != NULL || np_lookup("calc", &host_addr)))
11        requestor = requestor_new(host_addr);
12
13    if (requestor != NULL) {
14        request = data_new(2);
15        data_push(request, UINT, sizeof(uint_16), &a);
16        data_push(request, UINT, sizeof(uint_16), &b);
17
18        if (requestor_invoke(requestor, method, request, &reply)) {
19            data_pop(reply, &reply_value);
20
21            if (reply_value.value != NULL && reply_value.type == INT && reply_value.size == sizeof(int_32)) {
22                *result = *((int_32 *) reply_value.value);
23
24                data_destroy(request);
25                data_destroy(reply);
26
27                return true;
28            } else
29                errno = ENOMSG;
30
31            data_destroy(reply);
32        }
33
34        data_destroy(request);
35    } else
36        errno = EHOSTUNREACH;
37
38    return false;
39 }
```

Requestor Invoke

```
1 bool requestor_invoke(struct requestor *requestor, const char *const method, const struct data *const request, struct data **const reply) {
2     struct value bytes_value = {0};
3     rh_server_msg *msg;
4
5     marshall(request, requestor->host_addr->service_name, method, &bytes_value);
6
7     if (bytes_value.size > 0 && rh_send_to_server(requestor->conn_ctx, bytes_value.value, bytes_value.size)) {
8         free(bytes_value.value);
9
10        log_print(NOISY, "Sent message with %ld bytes to server", bytes_value.size);
11
12        if (NULL != (msg = rh_receive_from_server(requestor->conn_ctx))) {
13            log_print(NOISY, "Received message with %ld bytes from server", msg->data_size);
14
15            bytes_value.type = BYTES;
16            bytes_value.size = msg->data_size;
17            bytes_value.value = msg->data;
18
19            unmarshall(&bytes_value, NULL, NULL, reply);
20
21            rh_server_msg_destroy(msg);
22
23            if (*reply)
24                return true;
25            else
26                errno = ENOMSG;
27        } else {
28            requestor->closed = true;
29            log_debug(DEBUG, errno, "Failed to receive message from server");
30        }
31    } else {
32        requestor->closed = true;
33        free(bytes_value.value);
34
35        log_debug(DEBUG, errno, "Failed to send message to server");
36    }
37
38    return false;
39 }
```

Marshaller

```
1 void marshall(const struct data *const data, const char *const service, const char *const method, struct value *const value) {
2     uint_8 i;
3     const struct value *d_value;
4
5     if (service)
6         add_bytes(value, service, 'S', strlen(service));
7
8     if (method)
9         add_bytes(value, method, 'M', strlen(method));
10
11     for (i = 0; (d_value = data_get_value(data, i)) != NULL; ++i) {
12         if (d_value->size > 255) {
13             value->size = 0;
14             errno = EINVAL;
15             return;
16         }
17
18         #if __BYTE_ORDER == __BIG_ENDIAN
19             ...
20         #endif
21
22         add_bytes(value, d_value->value, d_value->type == BYTES ? 'B' : (d_value->type == INT ? 'I' : 'U'), d_value->size);
23     }
24 }
25
```

Marshaller

```
1 void unmarshall(const struct value *const value, char **const service, char **const method, struct data **const data) {
2     uint_8 i; void *bytes = NULL; byte marker; uint_8 size = 0; enum type type;
3
4     if (value->size > 0)
5         *data = data_new(1);
6
7     for (i = 0; i < value->size; i += (2 + size)) {
8         get_bytes(((byte *) value->value) + i, &bytes, &marker, &size);
9
10        switch (marker) {
11            case 'S':
12                if (service != NULL) {
13                    *service = malloc(size + 1);
14                    memcpy(*service, bytes, size);
15                    (*service)[size] = '\0';
16                }
17                continue;
18            case 'M':
19                if (method != NULL) {
20                    *method = malloc(size + 1);
21                    memcpy(*method, bytes, size);
22                    (*method)[size] = '\0';
23                }
24                continue;
25            case 'B':
26                type = BYTES;
27                break;
28            case 'I':
29                type = INT;
30                break;
31            case 'U':
32                type = UINT;
33                break;
34            default:
35                data_destroy(*data);
36                *data = NULL;
37                free(bytes);
38                errno = EINVAL;
39                return;
40        }
41
42        #if __BYTE_ORDER == __BIG_ENDIAN
43            ...
44        #endif
45
46        data_push(*data, type, size, bytes);
47    }
48
49    free(bytes);
50 }
```

Server

```
1 void run_server(const enum protocol protocol, const uint_16 port, const uint_8 thread_num) {
2     struct service *service = service_new("calc", methods_capacity: 4);
3     struct invoker *invoker = invoker_new(protocol, port, thread_num);
4
5     service_add_method(service, "add", calc_add);
6     service_add_method(service, "sub", calc_sub);
7     service_add_method(service, "mul", calc_mul);
8     service_add_method(service, "div", calc_div);
9
10    invoker_run(invoker, service);
11 }
```


Invoker

```
1 void invoker_run(struct invoker *const invoker, const struct service *const service) {  
2     invoker->service = service;  
3  
4     for (uint_8 i = 0; i < invoker->threads_num; ++i)  
5         thpool_add_work(invoker->thpool, (void (*)(void *)) run_server, invoker);  
6  
7     thpool_wait(invoker->thpool);  
8  
9     die(EXIT_FAILURE, NOERR, "Server died");  
10 }
```

Invoker

```
1  static __attribute__((noreturn)) void run_server(const struct invoker *const invoker) {
2      rh_server_ctx *server_ctx;
3      rh_client_msg *msg;
4      struct req *req = NULL;
5
6      if (NULL == (server_ctx = rh_server_new(invoker->protocol, invoker->port)))
7          die(EXIT_FAILURE, errno, "Failed to start server");
8      else
9          log_print(INFO, "Server is running");
10
11     for (;;) {
12         if (req == NULL)
13             req = malloc(sizeof(struct req));
14
15         if (NULL != (msg = rh_receive_from_client(server_ctx))) {
16             req->msg = msg;
17             req->invoker = invoker;
18
19             thpool_add_work(invoker->thpool, (void (*)(void *)) process_req, req);
20             req = NULL;
21         }
22     }
23 }
```

Avaliação comparativa de desempenho

