Aplicação Cliente/Servidor usando `PROTOBUF-C-RPC` em C

Protocol Buffer

```
1  syntax = "proto3";
2
3  service Calc {
4    rpc Add (Request) returns (Reply) {}
5    rpc Sub (Request) returns (Reply) {}
6    rpc Mul (Request) returns (Reply) {}
7    rpc Div (Request) returns (Reply) {}
8  }
9
10 message Request {
11   uint32 a = 1;
12   uint32 b = 2;
13 }
14
15 message Reply {
16   int64 r = 1;
17 }
```

Server

```
1  void run_server(const struct context *const ctx) {
2     char *name = malloc(6);
3     static struct _Calc_Service calc_service = CALC__INIT(calc_);
4     Server *server;
5
6     sprintf(name, "%d", ctx->port);
7
8     server = new_server(name, (Service *) &calc_service);
9
10    if (!server)
11        die(EXIT_FAILURE, EAGAIN, "Could not run the server");
12
13    for (;;)
14        dispatch_run();
15 }
```

Server - Calc

```
1   static void calc_add(Calc_Service *service, const Request *input, Reply_Closure closure, void *closure_data) {
2      Reply reply = REPLY__INIT;
3
4      reply.r = input->a + input->b;
5
6      closure(&reply, closure_data);
7   }
8
9   static void calc_sub(Calc_Service *service, const Request *input, Reply_Closure closure, void *closure_data) {
10     Reply reply = REPLY__INIT;
11
12     reply.r = input->a - input->b;
13
14     closure(&reply, closure_data);
15  }
16
```

Client - Request

```
1  static __always_inline uint_8 send_request(const char *const hostname, const Request *const request) {
2     Service *service;
3     uint8_t count = 0;
4     bool is_done = false;
5
6     service = new_service(hostname);
7
8     while (!service_is_connected(service) && count++ < 10)
9         dispatch_run();
10
11    if (!service_is_connected(service))
12        die(EXIT_FAILURE, EHOSTDOWN, "Could not connect to the server");
13
14    calc__sub(service, request, reply_handler, &is_done);
15
16    while (!is_done)
17        dispatch_run();
18
19    service_destroy(service);
20
21    return EXIT_SUCCESS;
22 }
```

Client - Benchmark

```c
uint_8 run_client_benchmark(const struct context *const ctx) {
   double total_time = 0, *times = malloc(sizeof(double) * ctx->benchmark_num), min = 0, avg, max = 0, mdev = 0;
   clock_t begin; /* ... */

   for (uint_16 i = 0; i < ctx->benchmark_num; ++i) {
       begin = clock();
       send_request(/* ... */);
       total_time += (times[i] = (double) (clock() - begin) / (CLOCKS_PER_SEC / 1000000.0));
       log_print(NOISY, "%.5d: %.3f µs\n", i + 1, times[i]);
   }

   avg = total_time / ctx->benchmark_num;
   min = times[0];

   for (uint_16 i = 0; i < ctx->benchmark_num; ++i) {
       max = max > times[i] ? max : times[i];
       min = min < times[i] ? min : times[i];
       mdev += pow(times[i] - avg, 2);
   }

   mdev = sqrt(mdev / ctx->benchmark_num);

   for (uint_16 i = 0; i < ctx->benchmark_num; ++i)
       printf("%.3f\n", times[i]);

   printf("min/avg/max/mdev = %.3f/%.3f/%.3f/%.3f µs\n", min, avg, max, mdev);

   return EXIT_SUCCESS;
}
```

PROTOBUF-C-RPC wrappers

```c
static __always_inline Server *new_server(const char *name, Service *service) {
   Server *server = protobuf_c_rpc_server_new(PROTOBUF_C_RPC_ADDRESS_TCP, name, service, protobuf_c_rpc_dispatch_default());

   if (server)
       protobuf_c_rpc_server_set_error_handler(server, error_handler, NULL);

   return server;
}

static void idle(Dispatch *dispatch __attribute__((unused)), void *func_data __attribute__((unused))) {}

static __always_inline void dispatch_run(void) {
   protobuf_c_rpc_dispatch_add_idle(protobuf_c_rpc_dispatch_default(), idle, NULL);
   protobuf_c_rpc_dispatch_run(protobuf_c_rpc_dispatch_default());
}

static __always_inline Service *new_service(const char *const hostname) {
   Service *service = protobuf_c_rpc_client_new(PROTOBUF_C_RPC_ADDRESS_TCP, hostname, &calc__descriptor, protobuf_c_rpc_dispatch_default());
   protobuf_c_rpc_client_set_error_handler((Client *) service, error_handler, NULL);

   return service;
}

static __always_inline bool service_is_connected(Service *service) {
   return (bool) protobuf_c_rpc_client_is_connected((Client *) service);
}

static __always_inline void service_destroy(Service *service) {
   protobuf_c_service_destroy(service);
}
```

Avaliação comparativa de desempenho (TCP x UDP)

Servidor com 1 thread