Aplicação Cliente/Servidor usando TCP/UDP em C

Server - Listen

```
1  static int_32 setup_listener(const enum protocol protocol, const uint_16 port) {
2     int_32 socket_fd, optval = 1;
3     struct sockaddr_in address = {
4             .sin_family = AF_INET,
5             .sin_addr = {.s_addr = htonl(INADDR_LOOPBACK)},
6             .sin_port = htons(port)
7     };
8
9     if ((socket_fd = socket(AF_INET, protocol == TCP ? SOCK_STREAM : SOCK_DGRAM, PF_UNSPEC)) < 0)
10        die(EXIT_FAILURE, errno, "Failed to create socket");
11
12    if (setsockopt(socket_fd, SOL_SOCKET, SO_REUSEPORT, &optval, sizeof(optval)) < 0)
13        die(EXIT_FAILURE, errno, "Failed to set socket option (SO_REUSEPORT)");
14
15    if (bind(socket_fd, (struct sockaddr *) &address, sizeof(address)) < 0)
16        die(EXIT_FAILURE, errno, "Failed to bind");
17
18    if (protocol == TCP && listen(socket_fd, 16) < 0)
19        die(EXIT_FAILURE, errno, "Failed to listen");
20
21    return socket_fd;
22 }
```

Server - Accept

```
1  static __attribute__((noreturn)) void *accept_connections(const struct context *const ctx) {
2     struct context *handle_ctx;
3     int_32 client_fd;
4     pthread_t thread;
5     pthread_attr_t thread_attr;
6
7     pthread_attr_init(&thread_attr);
8     pthread_attr_setdetachstate(&thread_attr, PTHREAD_CREATE_DETACHED);
9
10    for (;;) {
11        if (ctx->protocol == TCP) {
12            if ((client_fd = accept(ctx->server_fd, NULL, NULL)) < 0) { /* Print error message */
13                continue;
14            }
15
16            handle_ctx = malloc(sizeof(struct context));
17            memcpy(handle_ctx, ctx, sizeof(struct context));
18            handle_ctx->client_fd = client_fd;
19
20            pthread_create(&thread, &thread_attr, (void *(*)(void *)) handle_tcp, handle_ctx);
21        } else {
22            handle_udp(ctx, ctx->server_fd);
23        }
24    }
25 }
```

Server - Handle TCP

```
1  static void *handle_tcp(struct context *const ctx) {
2      ssize bytes;
3      struct request req;
4      uint_32 result;
5
6      if ((bytes = read(ctx->client_fd, &req, sizeof(req))) < 0) { /* Print error message */
7      } else {
8          result = calc(req);
9
10         if ((bytes = write(ctx->client_fd, &result, sizeof(result))) < 0) { /* Print error message */
11         } else
12             log_debug(NOISY, NOERR, "T%d: Wrote %td bytes", ctx->thread_index, bytes);
13     }
14
15     for (shutdown(ctx->client_fd, SHUT_WR), bytes = -1; bytes != 0;) {
16         if ((bytes = read(ctx->client_fd, &req, sizeof(req))) < 0) { /* Print error message */
17             break;
18         }
19     }
20
21     close(ctx->client_fd);
22     free(ctx);
23
24     pthread_exit(NULL);
25 }
```

Server - Handle UDP

```
1  static __always_inline void handle_udp(const struct context *const ctx, const int_32 socket_fd) {
2     ssize bytes;
3     struct request req;
4     struct sockaddr_in address;
5     uint_32 addr_len = sizeof(struct sockaddr_in), result;
6
7     if ((bytes = recvfrom(socket_fd, &req, sizeof(req), 0, (struct sockaddr *) &address, &addr_len)) < 0) {
8         log_debug(DEBUG, errno, "T%d: Failed to recvfrom", ctx->thread_index);
9         return;
10    } else
11        log_debug(NOISY, NOERR, "T%d: Received %td bytes", ctx->thread_index, bytes);
12
13    result = calc(req);
14
15    if ((bytes = sendto(socket_fd, &result, sizeof(result), 0, (struct sockaddr *) &address, addr_len)) < 0) {
16        log_debug(DEBUG, errno, "T%d: Failed to sendto", ctx->thread_index);
17        return;
18    } else
19        log_debug(NOISY, NOERR, "T%d: Sent %td bytes", ctx->thread_index, bytes);
20 }
```

Server - Response

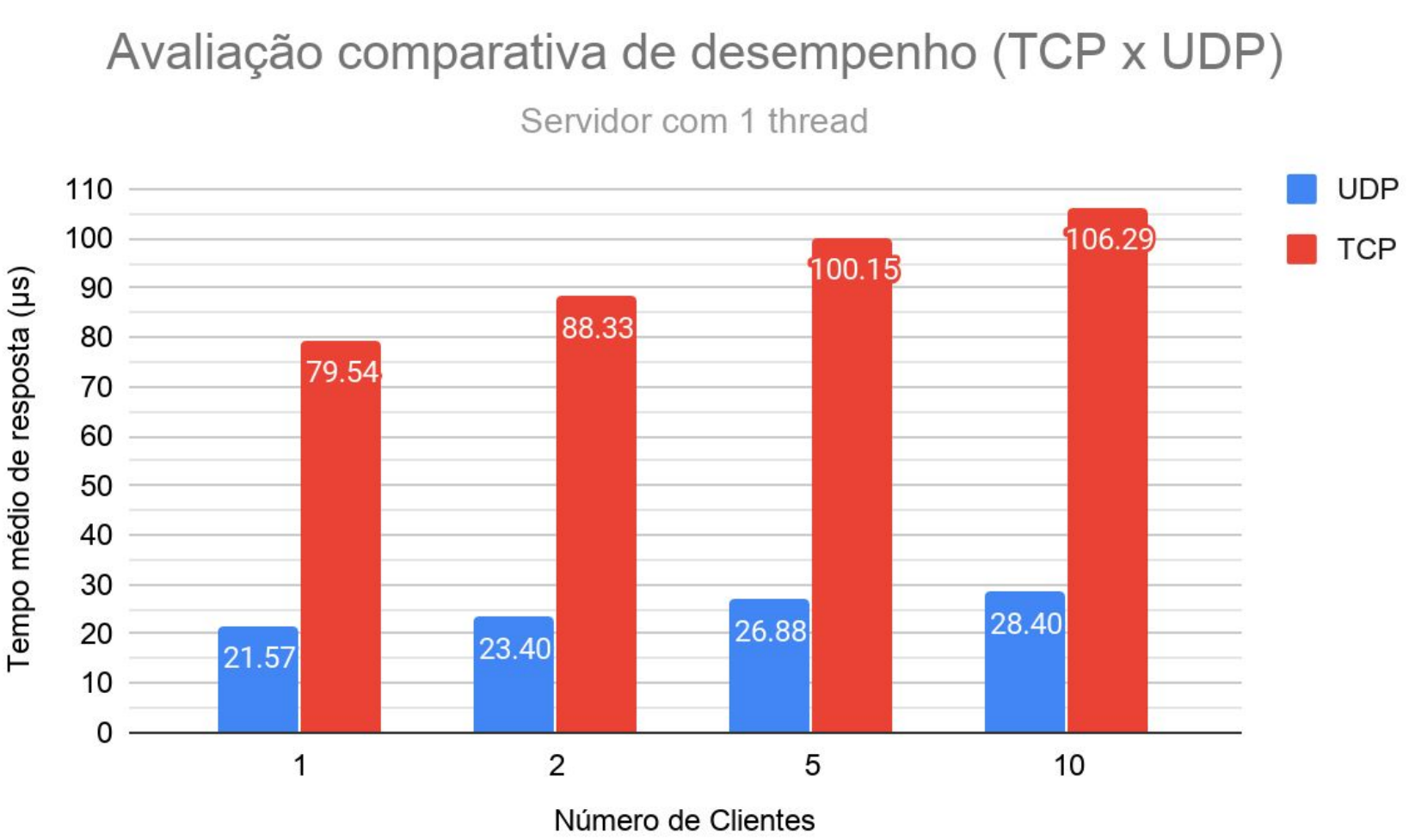
```
1   static __always_inline uint_32 calc(const struct request req) {
2      switch (req.op) {
3          case ADD:
4              return ntohs(req.a) + ntohs(req.b);
5          case SUB:
6              return ntohs(req.a) - ntohs(req.b);
7          case MUL:
8              return ntohs(req.a) * ntohs(req.b);
9          case DIV:
10             return ntohs(req.a) / ntohs(req.b);
11     }
12
13     return 0;
14  }
```

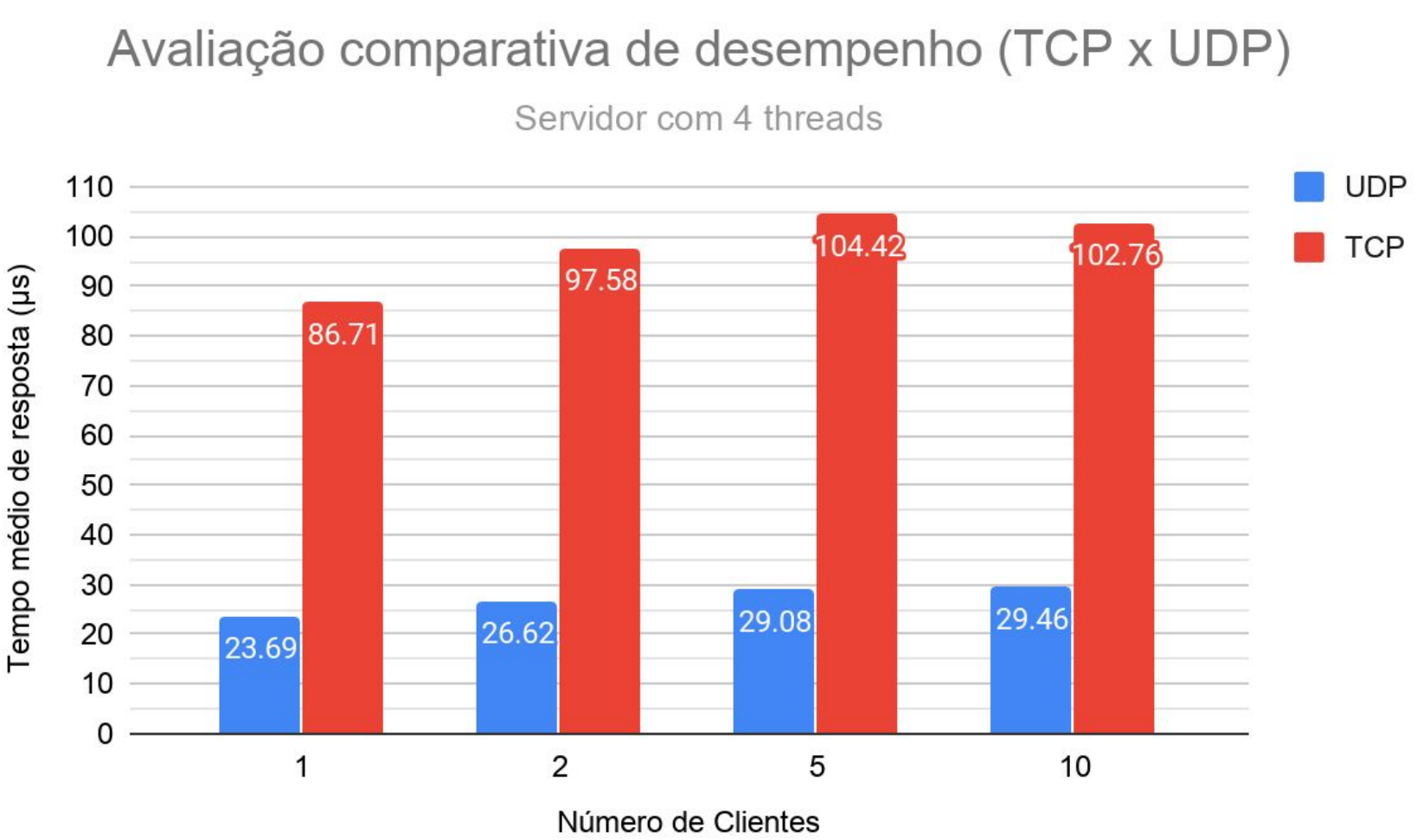
Client - Request

```c
static __always_inline uint8_t send_request(const struct context *const ctx) { /* Variables declarations */
    if ((socket_fd = socket(AF_INET, ctx->protocol == TCP ? SOCK_STREAM : SOCK_DGRAM, PF_UNSPEC)) < 0)
        die(EXIT_FAILURE, errno, "Failed to create socket");

    if (setsockopt(socket_fd, SOL_SOCKET, SO_RCVTIMEO, &time, sizeof(time)) < 0)
        die(EXIT_FAILURE, errno, "Failed to set socket option (SO_RCVTIMEO)");

    if (ctx->protocol == TCP && connect(socket_fd, (const struct sockaddr *) &address_out, sizeof(address_out)) < 0)
        die(EXIT_FAILURE, errno, "Failed to create socket");

    if (ctx->protocol == UDP && bind(socket_fd, (struct sockaddr *) &address_in, sizeof(address_in)) < 0)
        die(EXIT_FAILURE, errno, "Failed to bind");

    req = random_request();

    if (ctx->protocol == TCP) {
        if ((bytes = write(socket_fd, &req, sizeof(req))) < 0)
            die(EXIT_FAILURE, errno, "Failed to write");

        if ((bytes = read(socket_fd, &result, sizeof(result))) < 0)
            die(EXIT_FAILURE, errno, "Failed to read");
    } else {
        if ((bytes = sendto(socket_fd, &req, sizeof(req), 0, (const struct sockaddr *) &address_out, sizeof(address_out))) < 0)
            die(EXIT_FAILURE, errno, "Failed to sendto");

        if ((bytes = recvfrom(socket_fd, &result, sizeof(result), 0, (struct sockaddr *) &address_in, &addr_len)) < 0)
            die(EXIT_FAILURE, errno, "Failed to recvfrom");
    }

    close(socket_fd);

    return EXIT_SUCCESS;
}
```

Client - Request

```c
static __always_inline struct request random_request() {
   struct request req = {
           .a = htons((rand() % UINT16_MAX) + 1),
           .b = htons((rand() % UINT16_MAX) + 1)
   };

   switch (rand() % 4) {
       case 0:
           req.op = ADD;
           break;
       case 1:
           req.op = SUB;
           break;
       case 2:
           req.op = MUL;
           break;
       case 3:
           req.op = DIV;
           break;
   }

   return req;
}
```


Avaliação comparativa de desempenho (TCP x UDP)
Servidor com 1 thread
Tempo médio de resposta (µs)
110
100
90
80
70
60
50
40
30
20
10
0
21.57
79.54
23.40
88.33
26.88
100.15
28.40
106.29
1
2
5
10
Número de Clientes
UDP
TCP


Avaliação comparativa de desempenho (TCP x UDP)
Servidor com 4 threads
Tempo médio de resposta (µs)
110
100
90
80
70
60
50
40
30
20
10
0
86.71
97.58
104.42
102.76
23.69
26.62
29.08
29.46
1
2
5
10
Número de Clientes
UDP
TCP