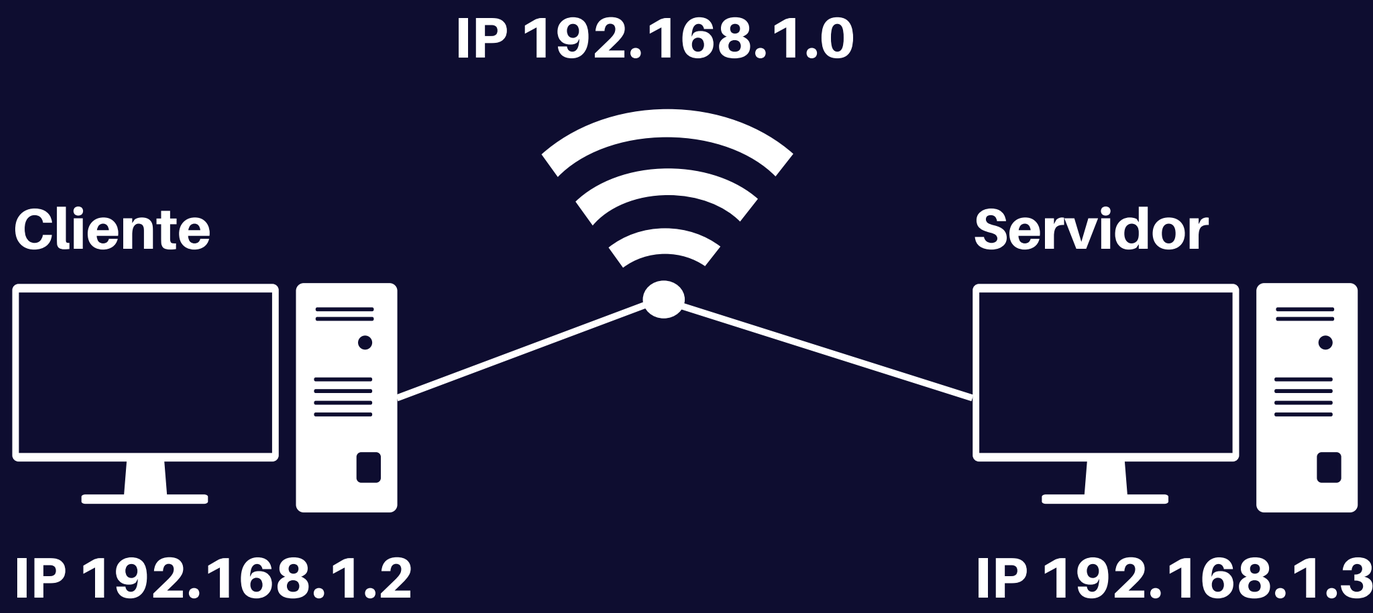
Compressed Network Communication

Criação de um cliente/servidor de telnet com a E/S passada atraves de um soquete TCP e compactação da comunicação entre ambos

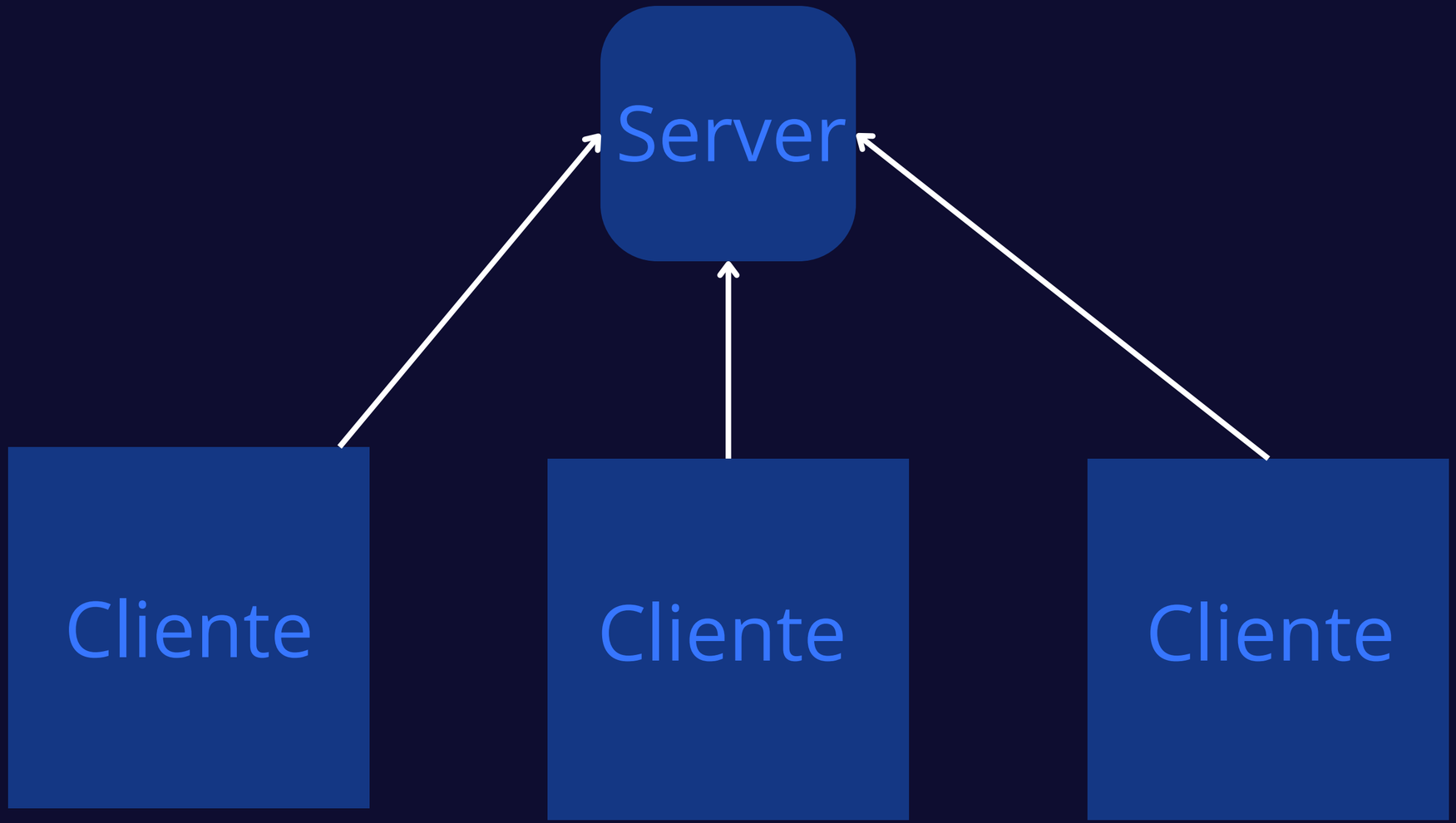
Angelo Ferro, Kaio Guilherme, Lucas Prado

Ambiente de teste

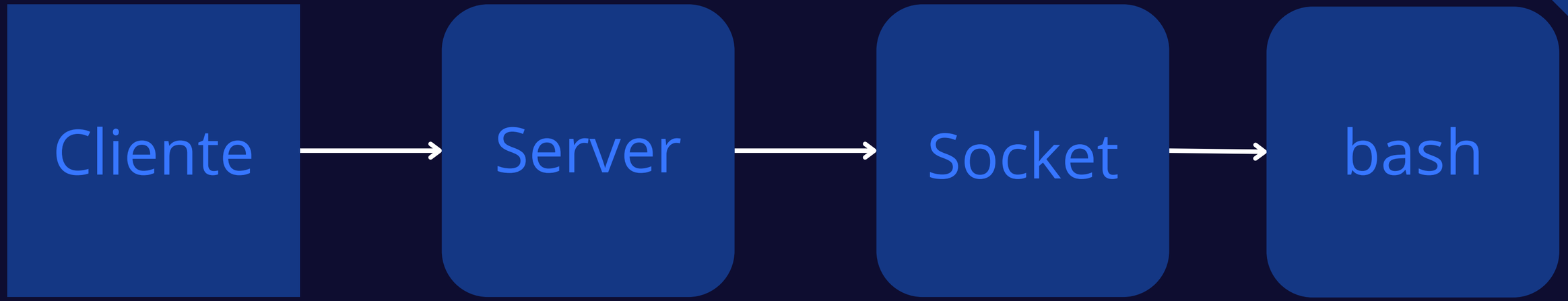
- Rede_experimental 192.168.1.0
- Criado container Cliente dentro da rede
- Criado container Servidor dentro da rede



Servidor



Servidor



Servidor

```c
void run(char *command, char *output, int output_size)
{
    printf("%s\n", command);

    FILE *file = popen(command, "r");

    if (file == NULL)
        printf("Unable to Open file");

    int new_size = fread(output, sizeof(char), output_size, file);

    if (ferror(file) != 0)
        printf("Error reading file");
    else
        output[new_size++] = '\0';

        pclose(file);

        if (strcmp(output, "") == 0){
            printf("command doesn't exist.\n");
            strcpy(output, "command doesn't exist.");
        }
}
```

Servidor

```c
int main(int argc, char **argv)
{
    char result[return_size];
    char comand[return_size];
    char optc = 0;
    int port;
    float version = 0.4;
    int server = 0;
    char *cprflag;

    struct option longopts[] = {
        {"port", required_argument, NULL, 'p'},
        {"log", no_argument, NULL, 'l'},
        {"versao", no_argument, NULL, 'v'},
        {0, 0, 0, 0}};

    if (argc == 1){ // Sem argumentos
        printf("Parametros faltando\n");
        exit(0);
    }
```

Servidor

```c
while ((optc = getopt_long(argc, argv, "v:p:l", longopts, NULL)) != -1)
{
    switch (optc){
        case 'v': // Ajuda
            printf("Versão %f\n", version);
            exit(0);
        case 'p': // port
            port = atoi(optarg);
            printf("port: %d\n",port);
            break;
        case 'l': // log
            printf("log:\n");
            break;
        default: // Qualquer parametro nao tratado
            printf("Parametros incorretos.\n");
            exit(1);
        }
}
```

Servidor

```c
// dados do servidor
int client, valread;

struct sockaddr_in caddr;
struct sockaddr_in saddr = {
    .sin_family = AF_INET,
    .sin_addr.s_addr = htonl(INADDR_ANY),
    .sin_port = htons(port)};
int csize = sizeof caddr;

if ((server = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
    printf("\n Socket creation error \n");
    return -1;
}

if (bind(server, (struct sockaddr *)&saddr, sizeof saddr) < 0)
{
    perror("bind failed");
    return -1;
}

if (listen(server, 5) < 0)
{
    perror("listen");
    return -1;
}
```

Servidor

```
client = accept(server, (struct sockaddr *)&caddr, &csize);

    while (1){
    // Uncompress
    ulong message_size;
    ulong message_byte_size;

    recv(client, &message_size, sizeof(ulong), 0);
    recv(client, &message_byte_size, sizeof(ulong), 0);

    char *buff = (char *)malloc(message_byte_size * sizeof(char));

    recv(client, buff, message_byte_size, 0);


    char *buffer_uncompress = uncompress_buffer(buff, message_size, message_byte_size);

    if (strncmp(buffer_uncompress, "exit()", 6) == 0)
    {
        printf("exit");
        close(client);
    }else{
        run(buffer_uncompress, result, return_size);


        ulong buffer_size = strlen(result) * sizeof(char) + 1;
        ulong buffer_byte_size = compressBound(buffer_size);


        char *buffer_compress = compress_buffer(result);

        send(client, &buffer_size, sizeof(ulong), 0);
        send(client, &buffer_byte_size, sizeof(ulong), 0);
        send(client, buffer_compress, buffer_byte_size, 0);
    }
```

Cliente

```
64
65      char *ip;
66      char buffer[1024];
67      char message[1024];
68      int opt;
69      int port;
70      int pflag = 0;
71      int lflag = 0;
72      int stop = 1;
73      char *hostname;
74
75      struct option longopts[] = {
76          {"ip", required_argument, NULL, 'i'},
77          {"port", required_argument, NULL, 'p'},
78          {"log", required_argument, NULL, 'l'},
79          {"hostname", required_argument, NULL, 'h'},
80          {"compress", no_argument, NULL, 'c'},
81          {0, 0, 0, 0}};
82
83      if (argc == 1)
84      { // Sem argumentos
85          printf("Parametros faltando\n");
86          exit(0);
87      }
88
```

Cliente

```
89    while ((opt = getopt_long(argc, argv, "i:p:l:h:c", longopts, NULL)) != -1)
90    {
91        switch (opt)
92        {
93        case 'i':
94            ip = optarg;
95            printf("ip: %s\n", ip);
96            break;
97        case 'p':
98            port = atoi(optarg);
99            printf("port: %d\n", port);
100           break;
101       case 'l':
102           log_filename = optarg;
103           logfd = creat(log_filename, S_IRWXU);
104           logflag = 1;
105           if (logfd < 1)
106           {
107               fprintf(stderr, "can't open file %s\n", log_filename);
108               exit(1);
109           }
110           lflag = 1;
111           break;
112       case 'h':
113           hostname = optarg;
114           break;
115       case 'c':
116           cprflag = 1;
117           break;
118       default:
119       {
120           fprintf(stderr, "unrecognized argument");
121           exit(1);
122       }
123       }
124   }
```

Cliente

```
125     // dados do cliente é ips
126     int server;
127     int client = 0, valread, client_fd;
128     struct sockaddr_in serv_addr = {
129         .sin_family = AF_INET,
130         .sin_port = htons(port)};
131
132     if (inet_pton(AF_INET, ip, &serv_addr.sin_addr) <= 0)
133     {
134         printf(
135             "\nInvalid address/ Address not supported \n");
136         return -1;
137     }
138
139     if ((client = socket(AF_INET, SOCK_STREAM, 0)) < 0)
140     {
141         printf("\n Socket creation error \n");
142         return -1;
143     }
144
145     if ((client_fd = connect(client, (struct sockaddr *)&serv_addr,
146                              sizeof(serv_addr))) < 0)
147     {
148         printf("\nConnection Failed \n");
149         return -1;
150     };
151     // if (cprflag){
152     // send(client, "1", 1, 0);
153     while (stop)
154     {
155         printf("%s@%s ~# ",hostname, ip);
156         fgets(message, 1024, stdin);
```

Zlib

- Compactação/ compressão

```
26  char *compress_buffer(char *buffer, ulong original_size, ulong compressed_buffer_size)
27  {
28      char *output = (char *)malloc(destLen * sizeof(char));
29
30      int test = compress(output, &destLen, buffer, buffer_size);
31
32      if (test == Z_OK)
33          return output;
34      else if (test == Z_BUF_ERROR)
35          error_output("Could Not Compress Because Buffer Is Too Small");
36      else if (test == Z_MEM_ERROR)
37          error_output("Could Not Compress Because There Was Not Enough Memory");
38      else
39          error_output("Could Not Compress");
40  }
41
42  char *uncompress_buffer(char *buffer, ulong original_size, ulong compressed_buffer_size)
43  {
44      char *output = (char *)malloc(original_size * sizeof(char));
45
46      ulong destLen = compressed_buffer_size;
47
48      int test = uncompress(output, &original_size, buffer, compressed_buffer_size);
49
50      if (test == Z_OK)
51          return output;
52      else if (test == Z_BUF_ERROR)
53          error_output("Could Not Uncompress Because Buffer Is Too Small");
54      else if (test == Z_DATA_ERROR)
55          error_output("Could Not Uncompress Because Data Is Incomplete Or Corrupted");
56      else
57          error_output("Could Not Uncompress");
58
59      return output;
```

Agradeço!