EP1 - MAC0352

Nome: Luca Lopes Barcelos
NUSP: 11221672

Implementação - Recebendo os pacotes

```
recvline[n] = 0;
printf("[Cliente conectado no processo filho %d enviou:] ", getpid());
int pkg_type = recvline[0] >> 4;
int topicSize, messageSize, multiplier, currByte, remainingLenght, pipe_fd;
char *topicName, *message, *pipe_name;
fprintf(stderr, "Package: %d\n", pkg_type);
for (int i = 0; i < n; i++)
    fprintf(stderr, "%02x ", recvline[i]);
fprintf(stderr, "\n");
```

Após ler a o soquete e armazenar o conteúdo em recvline, eu uso um right shift 4 vezes no primeiro byte de recvline para identificar o tipo de pacote que chegou, para agir de acordo.

Implementação - Tratando os pacotes - 1 Connect

```
switch (pkg_type)
{
case 1:
    fprintf(stderr, "CONNECT\n");
    char resp1[4] = {0x20, 0x02, 0x00, 0x00};

    if (write(connfd, resp1, 4) == -1)
    {
        perror("write 😕 \n");
        exit(6);
    }
    break;
```

Quando recebemos um pacote do tipo 1, Connect, devolvemos uma resposta com um pacote do tipo 2, Connack.

Implementação - Tratando os pacotes - 3 Publish

```
multiplier = 1;
remainingLenght = 0;
currByte = 0;

do
{
    currByte++;
    remainingLenght += (recvline[currByte] & 127) * multiplier;
    multiplier *= 128;
} while ((recvline[currByte] & 128) != 0);
```

1- Primeiro uso o algoritmo dado na especificação do MQTT para calcular o valor do campo Remaining Length.

```
topicSize = 0;
topicSize += (recvline[currByte + 1] >> 4) * 4096;
topicSize += (recvline[currByte + 1] & 15) * 256;
topicSize += (recvline[currByte + 2] >> 4) * 16;
topicSize += (recvline[currByte + 2] & 15);
```

2- Em seguida calculo o valor do tamanho, em bytes, do nome do tópico onde a mensagem vai ser publicada.

```
topicName = (char *)malloc(topicSize + 1);
memcpy(topicName, recvline + currByte + 3, topicSize);
topicName[topicSize] = '\0';

fprintf(stderr, "Topic: %s\n", topicName);

messageSize = remainingLenght - topicSize - 2;

fprintf(stderr, "Message Size: %d\n", messageSize);

message = (char *)malloc(messageSize + 1);
memcpy(message, recvline + currByte + 3 + topicSize, messageSize);
message[messageSize] = '\0';
```

3- Em seguida leio e armazeno o tópico e a mensagem a ser publicada.

```
if (asprintf(&pipe_name, "/tmp/temp.mac0352.%s.XXXXXX", topicName) == -1)
{
    perror("asprintf 😐 \n");
    exit(6);
}

if (mkfifo((const char *)pipe_name, 0666) == -1)
{
    perror("mkfifo :(\n");
}

pipe_fd = open(pipe_name, O_RDWR);
unlink((const char *)pipe_name);
for (int i = 0; i < n; i++)
    fprintf(stderr, "%02x ", recvline[i]);
fprintf(stderr, "Bytes written: %ld\n", write(pipe_fd, recvline, n));
close(pipe_fd);

break;
```

4- Por último uso o nome do tópico para criar um pipe FIFO, abro o pipe e coloco todo o pacote recebido no pipe, já que ele deve ser enviado exatamente igual para os subscribers.

Implementação - Tratando os pacotes - 8 Subscribe

```
char resp8[5] = {0x90, 0x03, 0x00, 0x01, 0x00};

if (write(connfd, resp8, 5) == -1)
{
    perror("write 😕 \n");
    exit(6);
}
```

1- Primeiro enviamos de volta um pacote do tipo 9, Suback, para dizer ao cliente que recebemos o pacote subscribe dele.

```
multiplier = 1;
remainingLenght = 0;
currByte = 0;

do
{
    currByte++;
    remainingLenght += (recvline[currByte] & 127) * multiplier;
    multiplier *= 128;
} while ((recvline[currByte] & 128) != 0);
```

2- Em seguida uso o algoritmo dado na especificação do MQTT para calcular o valor do campo Remaining Length.

```
topicSize = 0;
topicSize += (recvline[currByte + 1] >> 4) * 4096;
topicSize += (recvline[currByte + 1] & 15) * 256;
topicSize += (recvline[currByte + 2] >> 4) * 16;
topicSize += (recvline[currByte + 2] & 15);
```

3- Em seguida calculo o valor do tamanho, em bytes, do nome do tópico onde a mensagem vai ser publicada.

```
topicName = (char *)malloc(topicSize + 1);
memcpy(topicName, recvline + currByte + 5, topicSize);
topicName[topicSize] = '\0';
```

4- Em seguida armazeno o nome do tópico ao qual o cliente quer se inscrever.

```
if (asprintf(&pipe_name, "/tmp/temp.mac0352.%s.XXXXXX", topicName) == -1)
{
    perror("asprintf 😕 \n");
    exit(6);
}

if (mkfifo((const char *)pipe_name, 0666) == -1)
{
    perror("mkfifo :(\n");
}

while (1)
{
    fprintf(stderr, "Waiting for message...\n");
    pipe_fd = open(pipe_name, O_RDWR);
    unlink((const char *)pipe_name);
    if ((n = read(pipe_fd, (void *)recvline, MAXLINE)) > 0)
    {
        fprintf(stderr, "Reading from pipe %ld bytes\n", n);
        recvline[n] = 0;
        write(connfd, recvline, n);
    }
}
close(pipe_fd);

break;
```

4- Por último uso o nome do tópico para criar um pipe FIFO, abro o pipe e fico em um loop tentando ler dele. Caso consiga, envio o conteúdo lido como um pacote para o cliente.

Onde deu errado - Testes?

Não sei dizer ao certo o que está errado, mas sei quem tem relação com os pipes. Em meus testes consegui ler dos pipes uma vez, com o pacote sendo enviado para o cliente e a mensagem sendo exibida na tela do cliente, porém não consegui replicar isso nem com um código idêntico ao utilizado. No estado atual, eu consigo escrever o conteúdo necessário nos pipes, mas nunca nunca chego a ler. Também mantive um loop infinito no código que cuida do subscribe, pois foi assim que ele estava quando tive sucesso em enviar a mensagem para os clientes, mas sei que isso teria que ser mudado no código final, apenas não sei para o que. Por conta do código não estar funcionando como deveria e as mensagens não serem enviadas, não pude fazer os testes como esperado, então infelizmente não tenho dados de uso de CPU ou rede para meu broker.