MQTT BROKER (simplificado)

EP1 - MAC0352 - 2022/1

Por: Jessica Yumi Nakano Sato (11795294)

Objetivos do Trabalho:

- Implementar um Broker simplificado que siga o protocolo MQTT v3.1.1 que seja capaz de:
    - Conectar clientes
    - Inscrever clientes em tópicos
    - Publicar mensagens em um tópico
    - Desconectar clientes
- QoS level 0
- Uma função e um tópico por cliente
- Sem implementação de usuários e senhas

Implementação

MQTT

Formato dos packets

- Cabeçalho Fixo (Fixed Header), composto de dois bytes:
  - 1° byte descreve o comando.
  - 2°- 5° byte descreve comprimento restante do packet
- Cabeçalho Variável (Variable Header)
- Payload

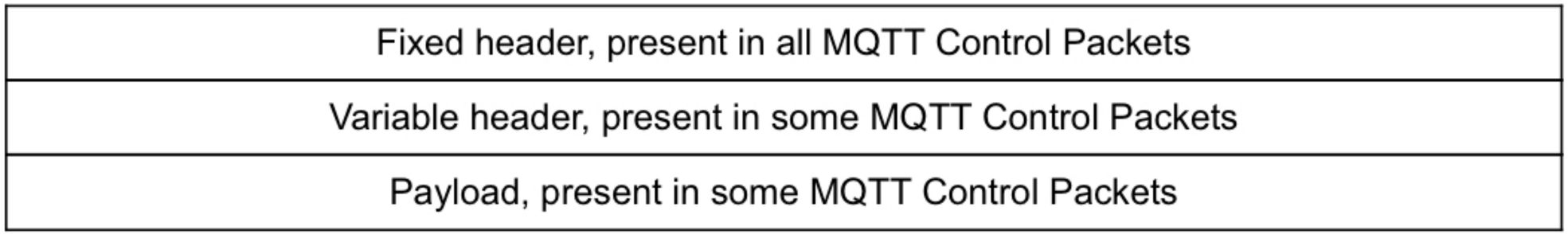


Figura 2.1 de docs.oasis-open.org/mqtt/mqtt/v3.1.1

Estrutura

- Para a estrutura principal do programa foi utilizado o código mac0352-servidor-exemplo-ep1.c disponibilizado pelo professor.
- Dentro da área indicada ao EP foi mantido o while que é responsável por ler os packets mandados e guardá-los em recvline.
- Nesse while, identificamos através do primeiro byte qual comando está sendo pedido pelo packet.
- Para cada comando, implementamos uma função que faz o tratamento do pacote recebido.

Comandos

Conexão

- CONNECT (0x10)
    - Sentido: Cliente -> Servidor
- CONNACK (0x20)
    - Sentido: Servidor -> Cliente

Comandos

Conexão

- Função: check_connect
  - Essa função é responsável por conferir o pacote de connect enviado pelo cliente (foi optato por apenas conferir se o protocolo especificado pelo pacote é o correto). Nesse caso é necessário conferir apenas os bytes 2-7.
  - Estando corretos a função devolve 0x00 que será colocado no último byte do connack enviado pelo broker.
  - Caso contrário ele devolverá 0x01, dessa forma o último byte do connack indicará que ouve um problema na conexão.

Comandos

Inscrição

- SUBSCRIBE (0x82)
    - Sentido: Cliente -> Servidor
    - Formato:
        - Variable Header: identificador do packet
        - Payload: tamanho e nome do código

Comandos

Inscrição

- SUBACK (0x90)
    - Sentido: Servidor -> Cliente
    - Mandado em resposta ao subscribe.
    - Formato:
        - Variable Header: identificador do pacote recebido no subscribe
        - Payload: retorno da inscrição (nosso caso 0x00)

Comandos

Inscrição

- Função: subscribe
- Nessa função criamos um arquivo especial (utilizando a função mkfifo) para cada subscriber no qual serão armazenados (e então enviados para os clientes) os packets que os publishers mandarem.
- O formato do nome do arquivo é temp.mac0352.<topic>.<pid>. Dessa forma teremos arquivos únicos para cada processo e ainda um fácil reconhecimento do tópico pelo publisher.
- Para constante leitura do arquivo e do socket optamos por utilizar threads.

Comandos

Inscrição

- As threads foram escolhidas pois permitem o compartilhamento de estruturas de dados. Dessa forma criamos a struct thread_arg que carrega o endereço do socket connfd e o pathname do arquivo gerado.

- Criamos duas threads:

  - A primeira é responsável pela leitura do socket, pegando e interpretando possíveis comandos enviados pelo cliente (como ping e disconect).

  - A segunda é responsável pela leitura do arquivo que armazena o packet mandado pelo publisher e retirado pelo subscriber.

Comandos

Publicação

- PUBLISH (0x30)
    - Sentido: Cliente (PUB) -> Servidor -> Cliente (SUB)
    - Para o QoS implementado (=0) não é preciso mandar um packet de volta.
    - Formato:
        - Variable Header: tamanho e nome do tópico
        - Payload: mensagem enviada
        - O tamanho da mensagem é calculado pelo Remaining Length e pelo Topic Length (tamanho do tópico)

Comandos

Publicação

- Função: publish
- Nessa função utilizamos a biblioteca dirent.h que facilita a busca de arquivos dentro de um diretório. Nesse caso estaremos fazendo a busca no diretorio / tmp e buscando os arquivos que tenham o template descrito na sessão anterior.
- Para cada arquivo que possua o mesmo tópico do publisher, fazemos a escrita do packet nesse arquivo.

Comandos

Ping

- PINGREQ (0xc0)
    - Sentido: Cliente -> Servidor
- PINGRESP (0xd0)
    - Sentido: Servidor -> Cliente
    - Mandado em resposta ao PINGREQ, caso não seja enviado o cliente fecha a conexão automaticamente.

Comandos

Desconexão

- DISCONNECT (0xe0)
    - Sentido: Cliente -> Servidor
    - Não é preciso mandar nenhum packet de volta

Funções Auxiliares

- print_hexa_bytes: utilizada durante o desenvolvimento do código. Criada para facilitar a visualização do packet recebido.
- thread_socket: função da thread responsável pela leitura do socket. Trata os comando ping e disconnect.
- thread_pipe: função da thread responsável pela leitura da pipe. Ela que irá de fato enviar o packet de publicação para o cliente subscriber.

Testes

Ambiente computacional e Rede

- Para o teste de nosso broker foram utilizados os clientes do mosquitto:
    - mosquitto_pub para publicações
    - mosquitto_sub para inscrições
- Rodamos o servidor na porta 1883 (porta padrão do mosquitto)
- Os testes foram feitos remotamente, ou seja, os clientes estavam em uma máquina virtual e o servidor numa outra máquina na mesma rede.
- Softaware utilizado para observação da Rede: Whireshark
    - Filtro: tcp port 1883
- Softaware utilizado para observação do consumo da CPU: psrecord, top

Ambiente computacional e Rede

- Atenção: importante ressaltar que o programa psrecord também contabiliza os filhos de um processo, entretanto ele faz isso somando o consumo de CPU de cada processo filho individualmente. Dessa forma, em alguns momentos, mesmo possuindo muitos processos filhos, o consumo de CPU aparece zerado devido a baixa precisão do medidor. Em outras palavras, como cada processo tem consumo baixo, sua %cpu é arredonda para zero fazendo com que mesmo que todos os processos, em conjunto, possuam alguma porcentagem da cpu, o resultado mostrado é zero.

Testes

Nenhum cliente

- Não foi identificado nenhum packet passando pela porta 1883.
- %CPU: 0.0

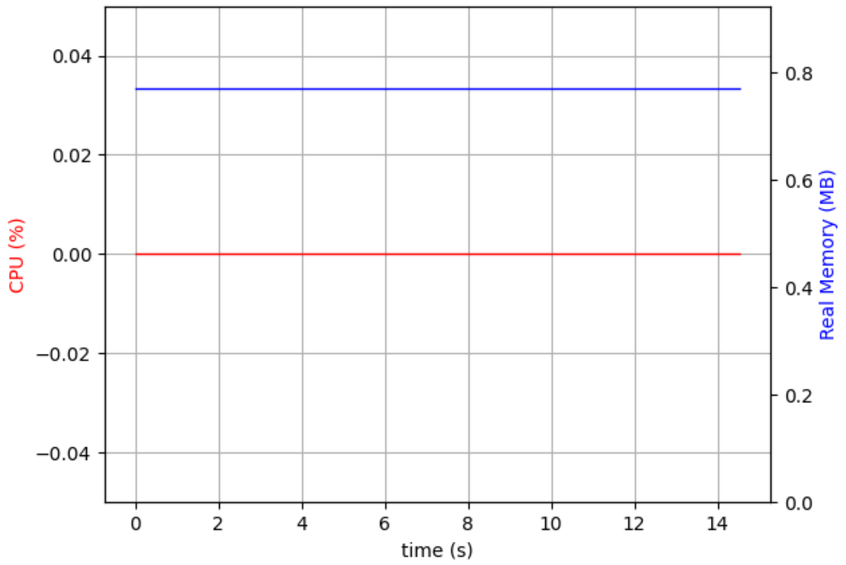


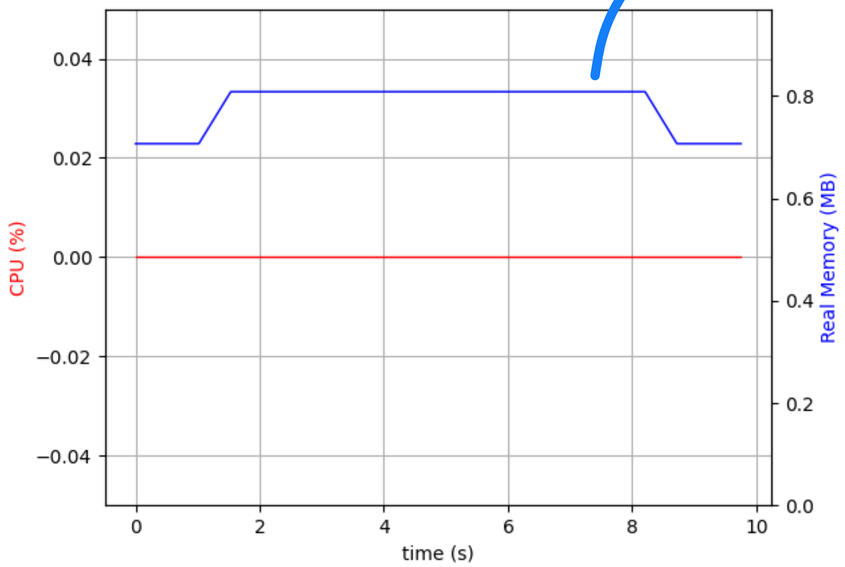
Imagem gerada pelo programa psrecord

Testes

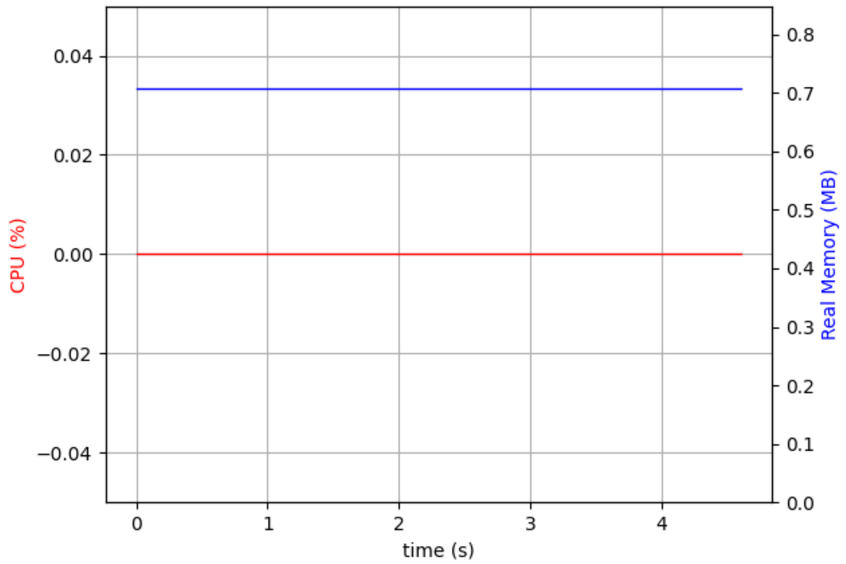
1 cliente

- SUB:
  - %CPU: 0.0
  - Consumo da Rede:
    - Bytes: 1213
    - Total de Packets: 17

Criação e deleção do arquivo temporário (pipe).



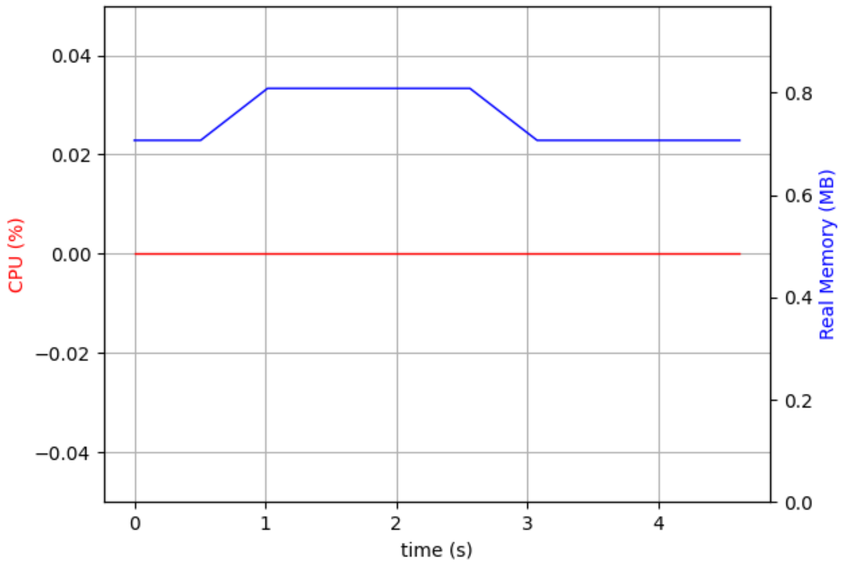
- PUB
  - %CPU: 0.0
  - Consumo da Rede
    - Bytes: 863
    - Total de Packets: 12



Testes

2 clientes SUB-PUB

- Mensagem enviada: oi
- Tópico: teste
- %CPU: 0.0
- Consumo da Rede:
  - Bytes: 2015
  - Total de Packtes: 28

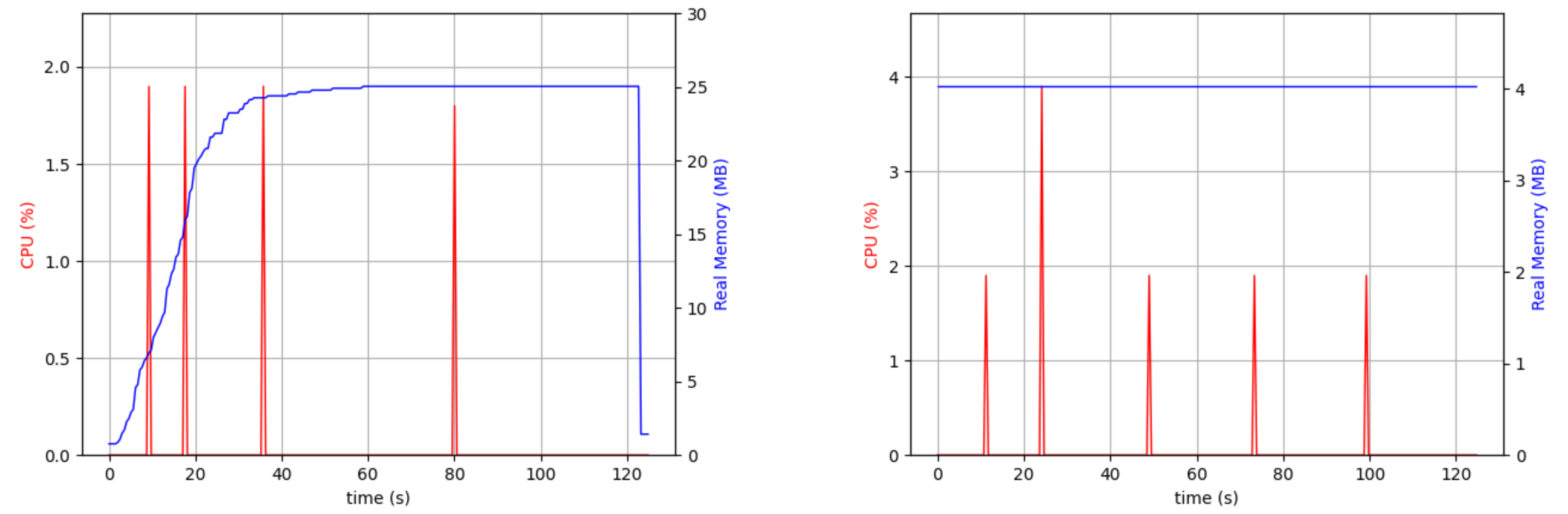


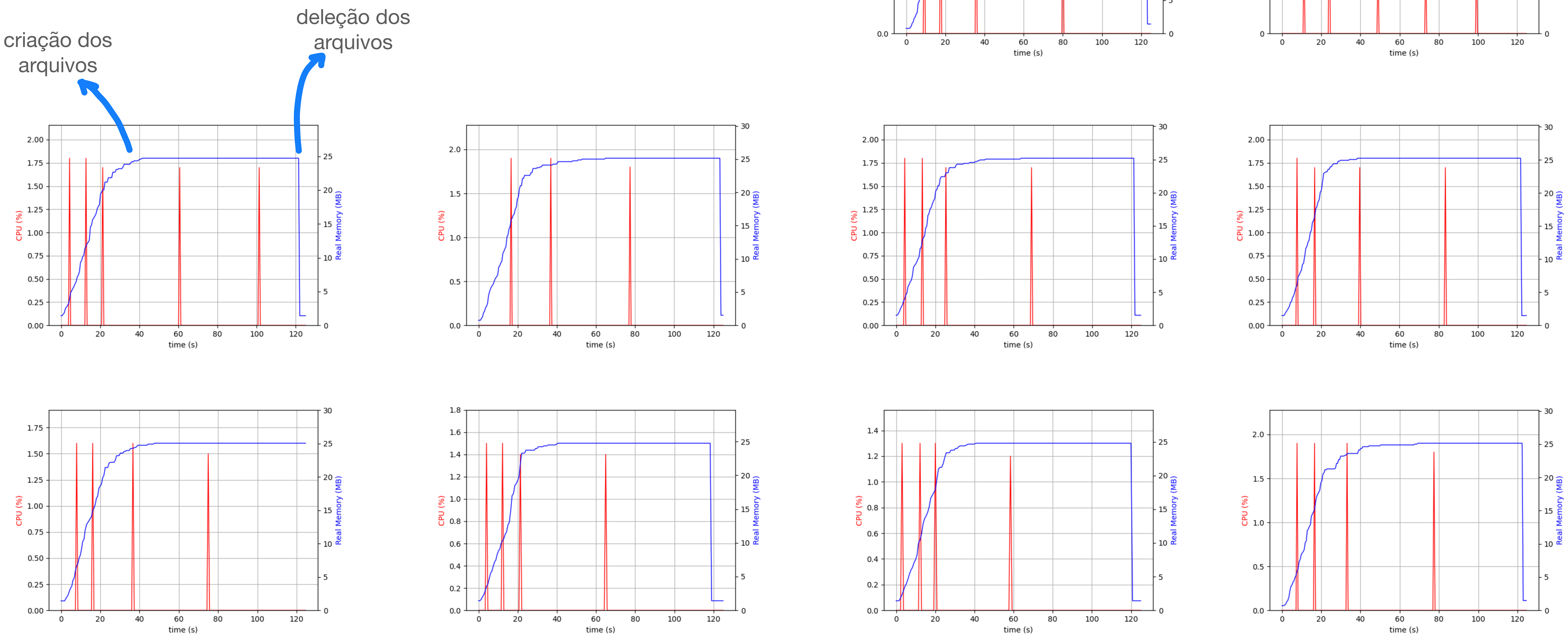
Testes

- Para os próximos testes (200 e 2000 clientes) foi feito um script em que:
  - 100 (ou 1000) clientes SUB se conectavam ao servidor
  - 100 (ou 1000) clientes PUB publicavam seu número no tópico dos SUB
  - Após todas as publicações todos os SUBs eram fechados.
- Foram feitas 10 medições, observando: quantidade de pacotes e bytes trocados, consumo de CPU e memória.
- Observação: testes “mal sucedidos” em que não houve envio completo das mensagens, também foram contabilizados.

Testes

200 clientes SUB-PUB





Testes

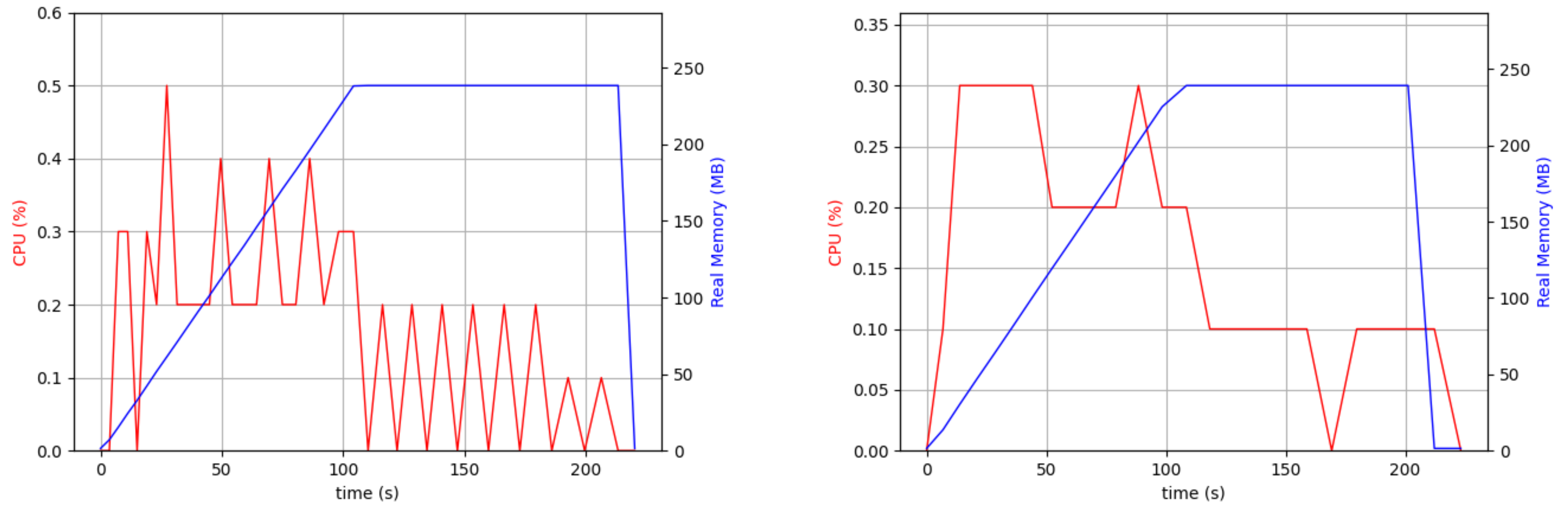
200 clientes SUB-PUB

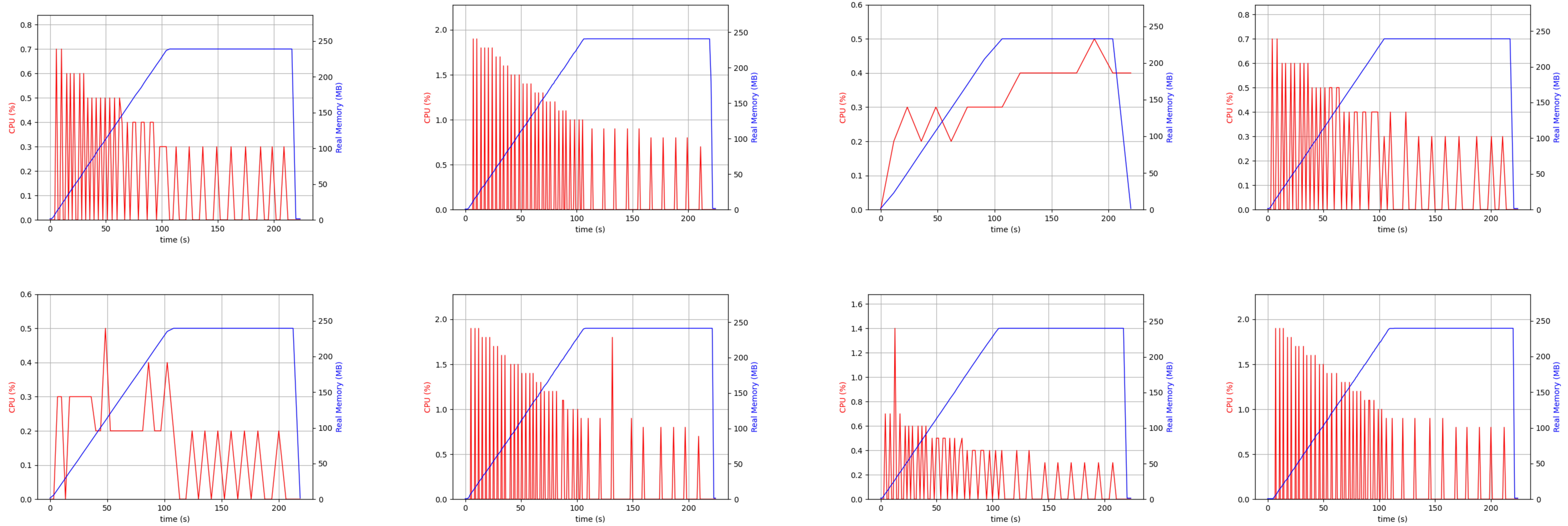
- Consumo da CPU:
  - Maior Pico:
    - Média: 1,9182
    - Desvio Padrão: 0,6805
  - Menor Pico:
    - Média: 1,6454
    - Desvio Padrão: 0,1955
- Consumo da Rede:
  - Packets:
    - Média: 4.384
    - Desvio Padrão: 281, 75
  - Bytes:
    - Média: 316.295,27
    - Desvio Padrão: 20.819,51

Como explicado anteriormente e observando os gráficos, foi optado observar e estudar os picos da utilização da CPU, sem considerar os momentos zerados

Testes

2000 clientes SUB-PUB





Testes

2000 clientes SUB-PUB

- Consumo da CPU:
  - Maior Pico:
    - Média: 0,9278
    - Desvio Padrão: 0,6190
  - Menor Pico:
    - Média: 0,5055
    - Desvio Padrão: 0,53877
- Consumo da Rede:
  - Packets:
    - Média: 44.136,33
    - Desvio Padrão: 2.248,94
  - Bytes:
    - Média: 3.175.747
    - Desvio Padrão: 163.105,50

Como explicado anteriormente e observando os gráficos, foi optado observar e estudar os picos da utilização da CPU, sem considerar os momentos zerados

Conceitos Aprendidos

Conceitos Aprendidos

Protocolo MQTT

- Embora não tenha sido implementado uma versão completa do broker, teve que se fazer um estudo afundo sobre os protocolos do tipo PUB/SUB e mais especificamente o MQTT

- Além disso, de uma forma geral foi possível, mais concretamente, entender o funcionamento e a importância dos protocolos durante uma comunicação na rede

Conceitos Aprendidos

Sockets, Arquivos e Threads

- Utilizando o código base do professor pudemos aprender o que e como se utilizar de sockets e fazer conexões por meio da rede através deles.
- Foi necessário um profundos conhecimento sobre a utilização de arquivos para esse EP já que a comunicação principal do servidor com os clientes era por meio de arquivos.
- Além disso, foi necessário também pesquisar sobre o que é, como funciona e como utilizar threads para que essa implementação funcionasse.

Problemas e melhorias

Problemas e melhorias

- Existe uma ineficiência em abrir e fechar o arquivo para cada operação de read e write (vide função thread_pipe e publish). O ideal seria manter o arquivo aberto e só fechá-lo na desconexão do cliente sub.

- Uma melhoria que deve ser feita é um melhor tratamento de erros, que, no momento, estão sendo apenas imprimidos na tela do servidor.

- E por fim, outra melhoria seria um melhor armazenamento dos arquivos de pipe (em diretórios por tópico por exemplo) para melhor eficiência na busca pelo arquivo.

Referências

- Protocolo MQTT: Oasis MQTT v 3.1.1
- Mosquitto: https://mosquitto.org/documentation/
- Melhor entendimento dos pacotes: openlabpro
- Threads: https://pt.wikipedia.org/wiki/POSIX_Threads
- Diretorios: http://wiki.inf.ufpr.br/maziero/doku.php?id=pua:operacoes_em_diretorios
- Dirent.h: https://stackoverflow.com/questions/46555447/c-use-of-dirent-h
- Mkfifo: https://man7.org/linux/man-pages/man3/mkfifo.3.html
- psrecord: https://github.com/astrofrog/psrecord

Fim :)