Broker MQTT - EP1

Bruno Hideki Akamine

NUSP: 11796322

Especificações

- **Bibliotecas adicionais**
  - dirent.h - Ver os arquivos presentes em um diretório
  - pthread.h - Criar threads para trabalharem em paralelo
  - Além das bibliotecas presentes no servidor de exemplo com pipe
- **Resolução de bytes**
  - Para trabalhar com os bytes utilizou-se o vetor de unsigned char recvline, onde cada byte correspondia a um char no vetor. Assim para fazer a checagem de pacotes ou flags convertemos o char para int (type casting).
- **Indentificação do cliente**
  - Não pegaremos o id do cliente presente no pacote *CONNECT*, utilizaremos o pid (process id) para fazer essa indentificação.
- **Keep Alive**
  - Não implementamos o mecanismo de *keep alive* para as conexões.

Suposições

- **Pacote CONNECT**
  - Oitavo byte - Flags
    - Clean Session - 1
    - Will Flag - 0
    - Will Qos 1 - 0
    - Will Qos 2 - 0
    - Will Retain - 0
    - User Name - 0
    - Password - 0
- **Pacote PUBLISH**
  - Primeiros 4 bits do primeiro byte
    - DUP Flag - 0
    - Qos Level 1 - 0
    - Qos Level 2 - 0
    - Retain - 0

Escolhas de implementação I

- Para cada conexão com um cliente sub criamos um arquivo FIFO (mkfifo) com o seguinte template : "temp.mac0352.brunoakamine.topic.pid", onde topic corresponde ao tópico em que o cliente sub se inscreveu e pid é o pid do processo do cliente sub. Assim, cada cliente cria um arquivo único em */tmp*.
- Para a publicação em um tópico *topic* por exemplo, vemos todos os arquivos em */tmp* que tem o template "temp.mac0352.brunoakamine.topic.", e escrevemos nesses arquivos. Utilizamos o arquvio FIFO para pordemos "travar" a entrada de leitura e assim, somente quando tiver alguma publicação o processo respectivo do cliente sub irá ler a mensagem. Para o write no FIFO, abrimos com a flag O_NOBLOCK para evitar possíveis dessincronizações que resultariam no write "travar" no arquivo.

Escolhas de implementação II

- Para cada cliente sub criamos duas threads. Ambas as threads recebem como argumento uma struct que contém o path até seu arquivo e um ponteiro para o descritor de seu socket de conexão.
- A primeria Thread (Thread1) estará lendo os comandos vindos de *connfd* (socket de conexão) para poder responder aos pings requests e para saber quando o sub envia o pacote *DISCONNECT*.
- A segunda Thread (Thread2) estará lendo o arquivo FIFO criado para aquela conexão. Assim, quando tiver um PUBLISH ele lerá o arquivo e escreverá em *connfd*.
- Aqui a indentificação do pacote de *DISCONNECT* tem papel crucial abrindo e fechando o arquivo respectivo da conexão e consequentemente fazendo com que saia do *while* na Thread2.

Escolhas de implementação III

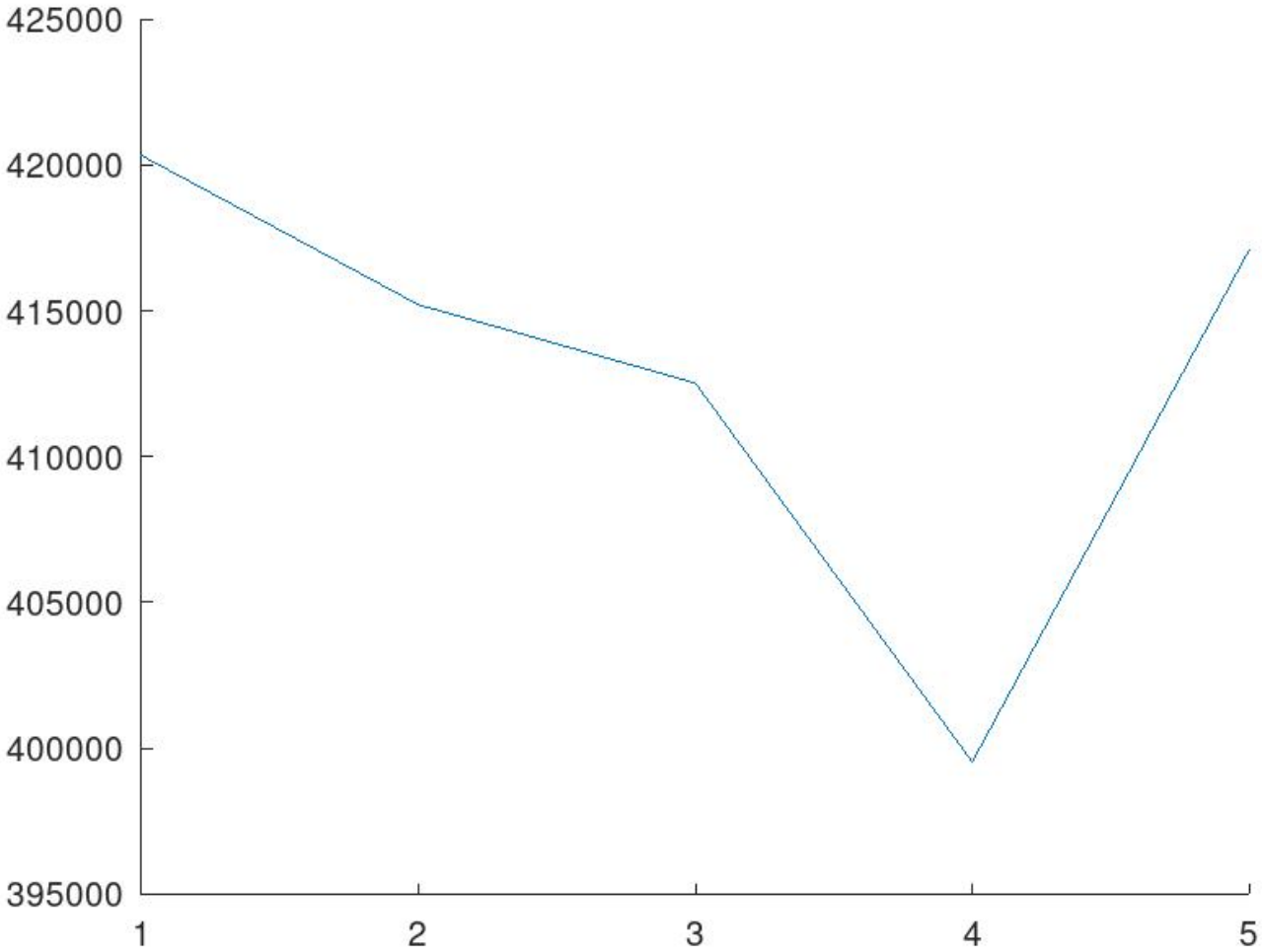
- A escolha de duas threads se deu para que conseguissimos escutar simultaneamente do socket de conexão e do arquivo FIFO criado para essa conexão específica. Note que se quisessemos deixar com que um sub pudesse se inscrever em mais de um tópico, bastaria criar uma thread para cada tópico e generalizar o processo de escuta no arquivo FIFO explicado acima.

Performace I

- Observações
  - Os testes foram feitos com uma única máquina
  - O broker foi executado na máquina local
  - Os clientes pub e sub se conectaram através de uma máquina virtual
  - Nos testes conectamos 100/1000 subs antes de começar as publicações (100/1000 respectivamente)
  - Desempenho de rede
    - 0 clientes conectados - Não teve nenhum uso de rede para este caso. Portanto a média e desvio padrão são iguais a 0.

Performace II

- 100 clientes conectados:

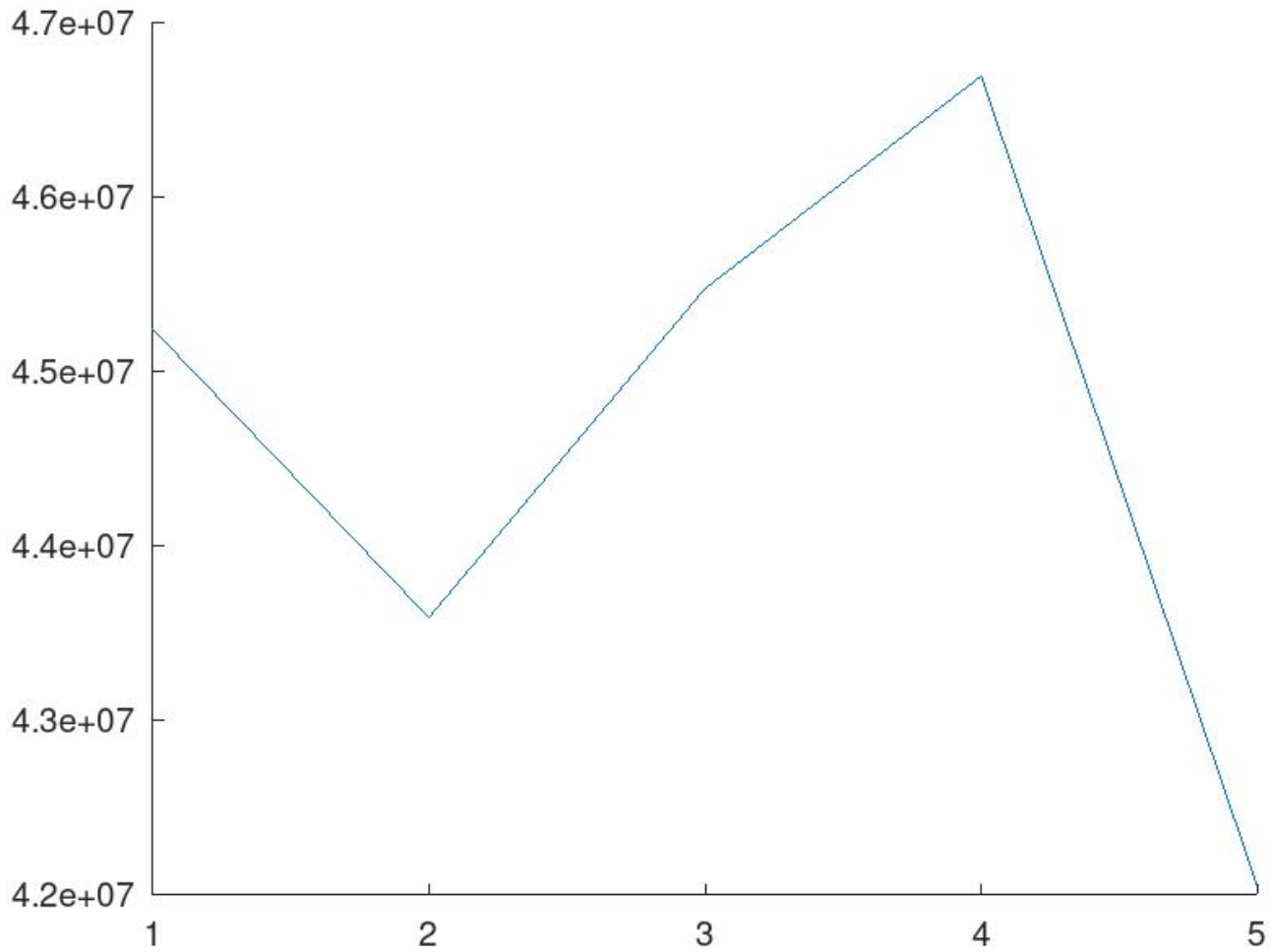


Performace III

No eixo eixo x corresponde ao número do teste e no eixo y corresponde a quantidade de bytes totais.
média = 412937.8
desvio padrão = 7170.034990

Performace IV

- 1000 clientes conectados:



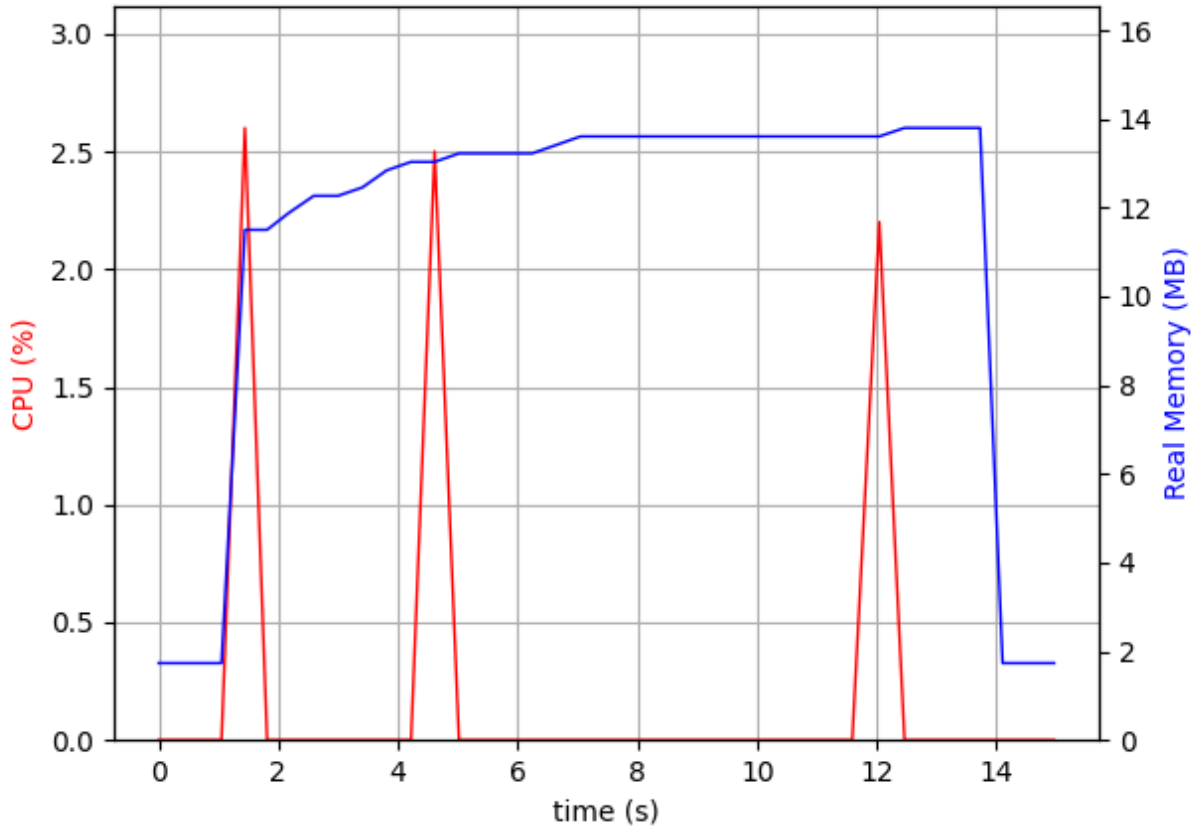
Performace V

0 clientes conectados - Não teve nenhum uso de rede para este caso. Portanto a média, desvio padrão e intervalo de confiança são iguais a 0. Note que no último teste, apesar de estar bem baixo ele ainda esta na ordem de 4.2×10^7.
média = 44607392
desvio padrão = 1617507.03400

- Desempenho CPU - Para este caso utilizou-se a ferramenta psrecord, porém os resultados gerados eram gráficos informando o uso de CPU ao longo do tempo se mostraram bem instável. Para cada caso (menos no de 0 clientes, já que em todos os casos o uso de CPU era muito baixo e portanto era uma constante em 0), mostraremos um dos gráficos gerados nem um dos teste para mostrar como o gráfico para cada caso se parece. Além disso para calcular a média e a variância usaremos o maior valor mostrado em cada teste.
  - 0 clientes - O uso de CPU quando somente o broker estava ativo era tão próximo de 0 que a ferramente que utilizei aproximou para 0%. Logo consideraremos a média e o desvio padrão como iguais a zero.

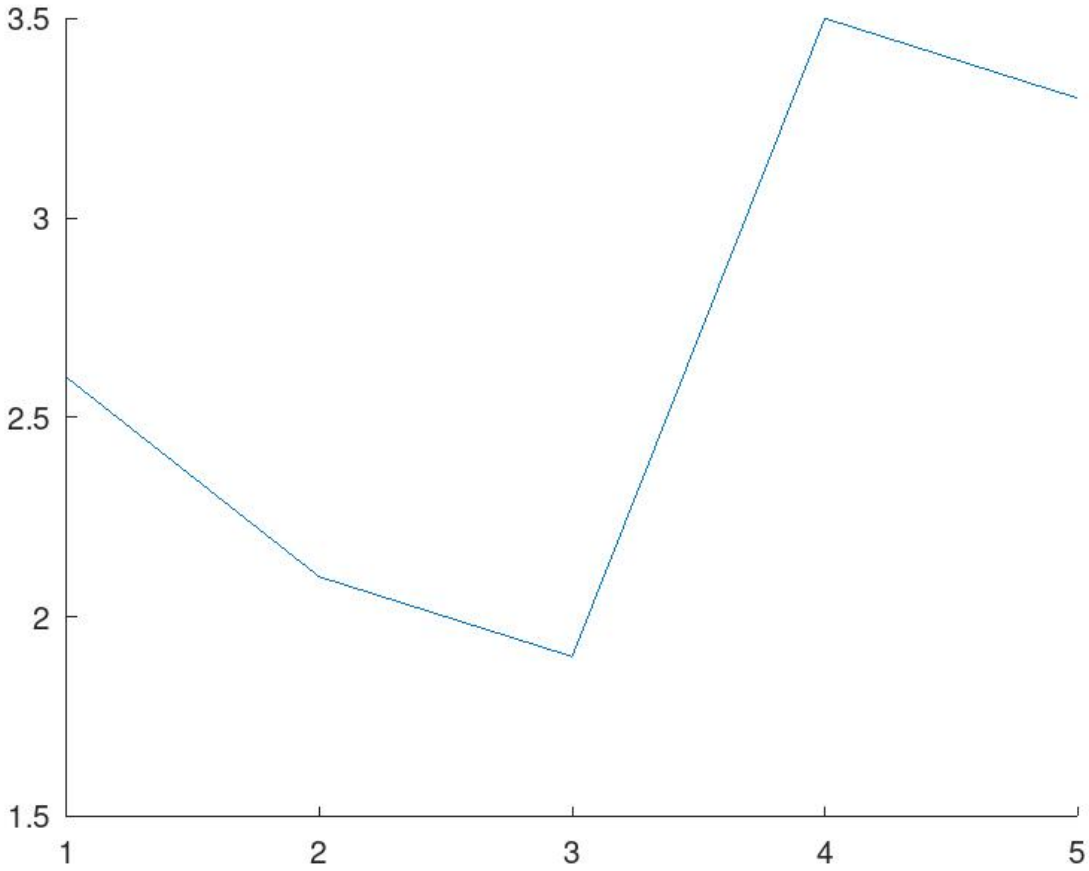
Performace VI

- 100 clientes:



O gráfico abaixo mostrará a utilização máxima de CPU para cada teste.

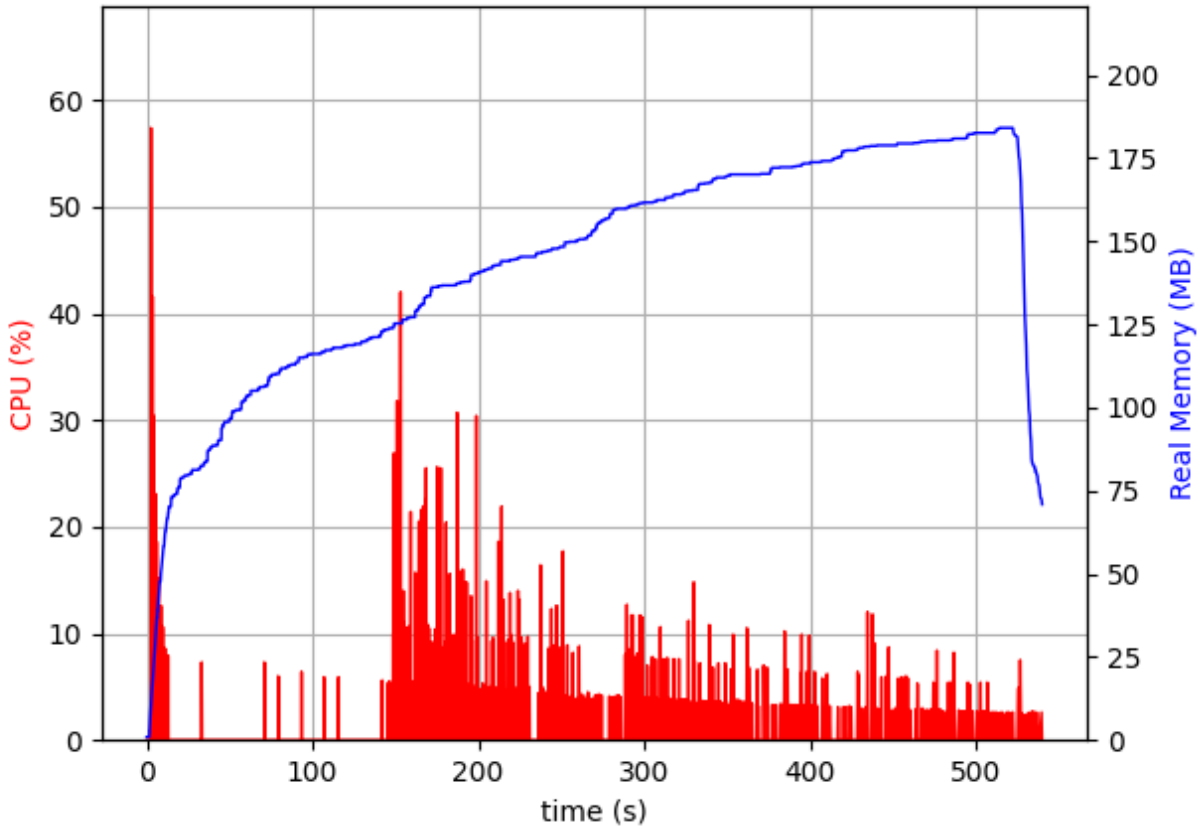
Performace VII



média = 2.68%
desvio padrão = 0.6337%

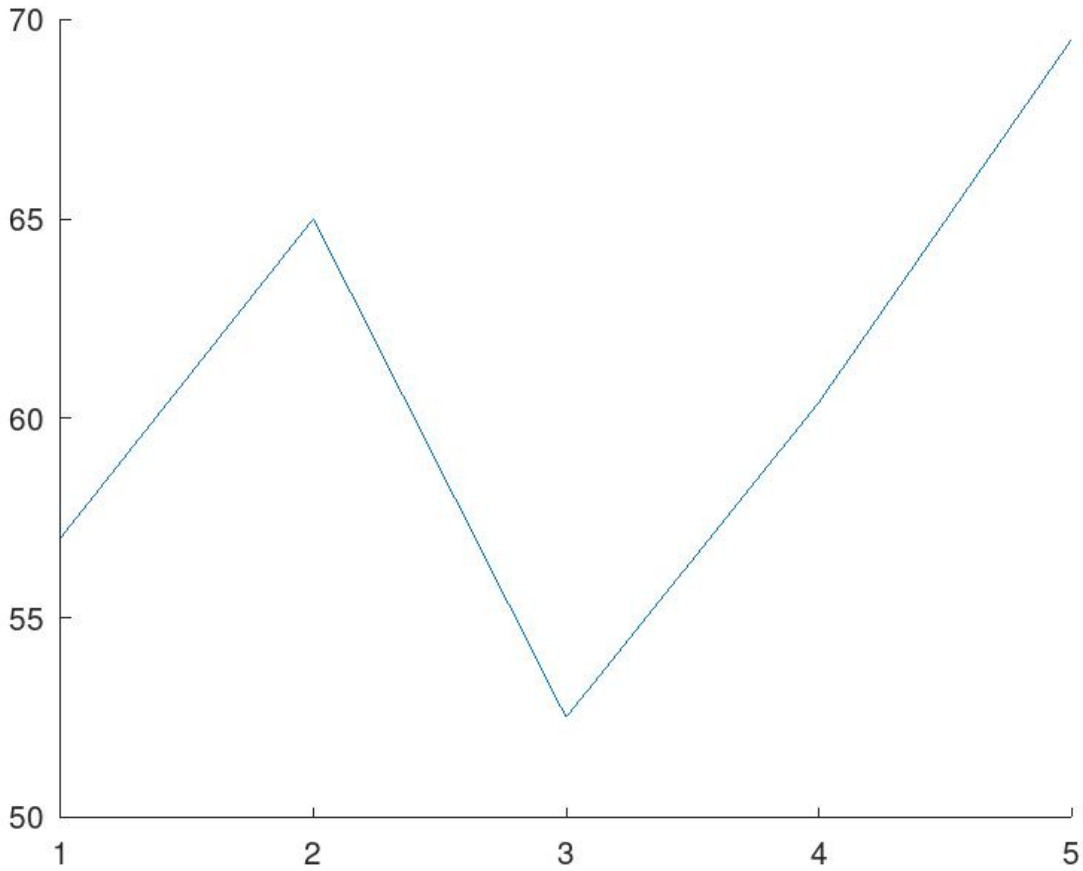
Performace VIII

- 1000 clientes:



O gráfico abaixo mostrará a utilização máxima de CPU para cada teste.

Performace IX



média = 60.88%
desvio padrão = 5.9462%

Considerações finais

- Com o aumento da quantidade de clientes aparecem problemas, não consegui ao certo indentificar o problema.
- Suspeita : devido a dessincronização de processos e threads, causando o fechamento dos arquivos FIFO de alguns processo de forma indesejada. Uma possível soluçao para essa suspeita seria a utilização de semafóros.
- Podemos ver a ocorrência desse erro nos gráficos de uso de CPU pelo tempo de um dos testes, note que quanto mais a frente ao tempo a utilização de CPU vai caindo, isso provavelmente se deve ao fechamento indesejado de vários arquivos FIFO fazendo com que os clientes sub não recebem as mensagens do pub.