EP1

”Implementação de um servidor broker”

MAC0352 – Redes de computadores e Sistemas Distribuídos

Nome: Pedro Fernandes

NUSP: 5948611

Estratégia de aprendizado

Objetivos

- Entender o funcionamento broker.
- Uso de pipes como alternativa de comunicação entre processos.
- Pensar no funcionamento lógico/computacional do trabalho do servidor no envio e recebimento de mensagens.
- Entender o funcionamento dos clientes (publicadores e assinantes)
- Entender o funcionamento do protocolo MQTT

Estratégia de aprendizado

Criação de uma troca de mensagens simples como a vista em aula:

Servidor:

- socket[] -> bind[] -> listen[] -> accept[] -> send/recv[]

Cliente:

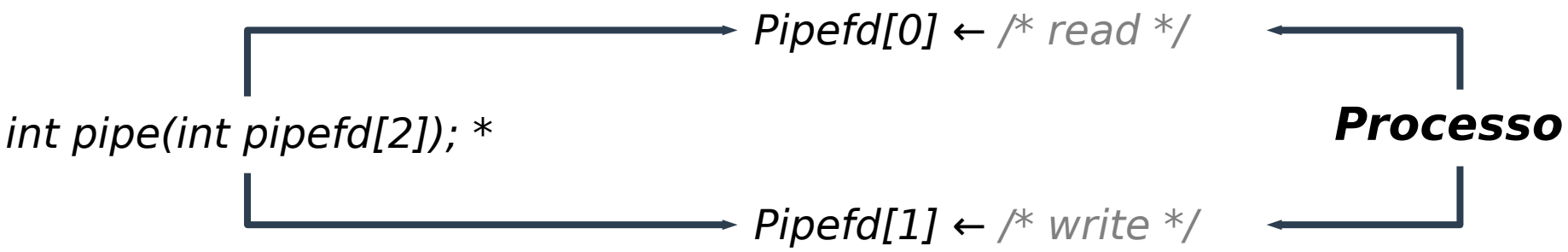
- socket[] -> connect[] -> send/recv[]

Estratégia de aprendizado

Criação de um servidor do zero. Contudo, inspirado no servidor echo fornecido pelo professor e os vistos em aula.

- **Essa decisão foi movida puramente por uma questão de aprendizado, para o total entendimento de como ocorre a comunicação em rede pelos sockets, como também, a comunicação entre processos.**

- **Decisão de projeto: Uso de pipes (unidirecionais, necessita de dois para comunicação a dois fins)**



Comandos Protocolo MQTT

O funcionamento de um servidor mosquitto MQTT, em um caso de QoS desligado (QoS = 0), resumisse, parcialmente, à uma troca de mensagens TCP como as abaixo:

Recebe: CONNECT (Requisição de conexão do cliente)

Envia : CONNACK (Reconhecimento da conexão)

Recebe:

- **PUBLISH (Uma publicação em um tópico de um cliente)**
  - **SUBACK (Reconhecimento da incrição)**

ou

- **SUBSCRIBE (Uma inscrição em um tópico de um cliente)**
  - **PUBACK (Reconhecimento da publicação)** — * — **Ausente Q0S = 0**

Recebe : DISCONNECT (Pedido do cliente de encerramento de conexão)

*: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

"A PUBACK Packet is the response to a PUBLISH Packet with QoS level 1."

Algumas Decisões de projeto - Processos

Processos

- **Processo Pai** – Cuida da aceitação de nova conexões
- **1 Processo filho** para um cliente publicador:

  Após o servidor identificar o cliente como publicador. Ele seleciona sua mensagem e faz o trabalho de envio por “pipes” para quaisquer outros processos filhos dos inscritos.

- **1 Processos filho** que cria outro **Processo filho** para o cliente inscrito:

  Após o servidor identificar o cliente como inscrito. Ele seleciona o tópico de inscrição pertinente e divide-se em dois processos, um para envio de mensagens publicadas naquele tópico e outro para a escuta por um pedido de desconexão por parte do cliente.

Algumas Decisões de projeto - Processos

Processos

Processo Pai

- Aceita conexão

Processo Filho do Publicador

- Faz o trabalho de pipes para outros processos

Processo Filho do inscrito

- Recebe mensagens por pipe e envia para cliente

Processo “Neto” do inscrito

- Espera por pedido de desconexão do cliente.

Algumas Decisões de projeto - Estr. de Dados

A estrutura de dados escolhida foi a de listas ligadas. Subdivididas em estrutura de “nós de tópicos” que dão inicio a outra estrutura de “nós de pipes”.

- Com isso a estrutura de dados cresce conforme necessidade do programa.

- Auxilia no manejo de memória.

Algumas Decisões de projeto - Memória

Como cada processo copia as condições de memória do processo pai, os processos agem da seguinte forma:

- O processo filho responsável pelo cliente **inscrito** primeiramente libera toda sua estrutura de dados referentes a tópicos e pipes exceto a estrutura de nó de pipe referente a ele próprio. Deste modo evita acúmulo desnecessário de pipes abertos bem como outros bugs.

- O Processo responsável pelo cliente **publicador** realiza o trabalho de pipes usando a estrutura de dados presente no processo pai e libera também toda sua estrutura alocada antes de finalizar sua operação.

Desempenho - Sistema

Foram dois sistemas em que foram realizados os testes:

- **Sistema do servidor (DESKTOP):**

OS: Ubuntu 20.04.4 LTS x86_64

Kernel: 5.13.0-40-generic

Terminal: gnome-terminal

CPU: AMD Ryzen 3 3300X (8) @ 3.800GH

Memória: 15907MiB

- **Sistema dos clientes (NOTEBOOK):**

OS: Ubuntu 20.04.4 LTS x86_64

Kernel: 5.13.0-39-generic

Terminal: gnome-terminal

CPU: Intel i3-2350M (4) @ 2.300GH

Memória: 3872MiB

Desempenho - Sistema - Rede

Rede local com cabo ethernet:

- Velocidade máxima: 1000 Mb/s
- Método IPv4: Automatic (DHCP)

Desempenho - Metodologia

Foi implantado um shell (bash) script para o sistema dos clientes que manteve a mesma lógica para a criação de clientes publicadores e inscritos.:

De 1 a N :

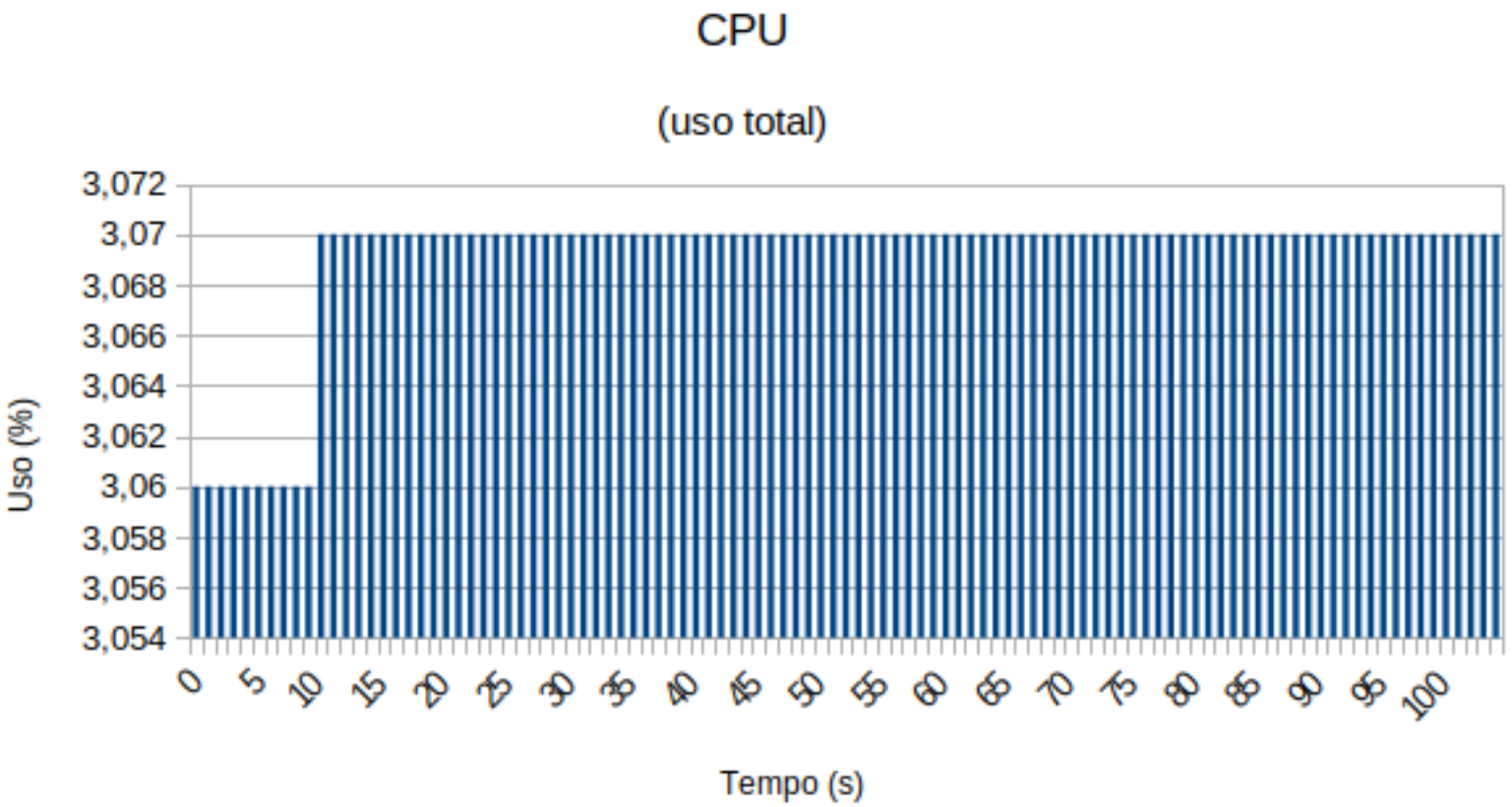
- Seleciona aleatoriamente 10 tópicos (strings)
- Cria um cliente aleatório mas com a seguinte proporcionalidade:
  - 1/5 publicadores
  - 4/5 inscritos
- Caso inscrito fica funcionando até o fim do script + sleep final
- Caso publicador publica “A message from $i” sendo i o número de criação.

Final :

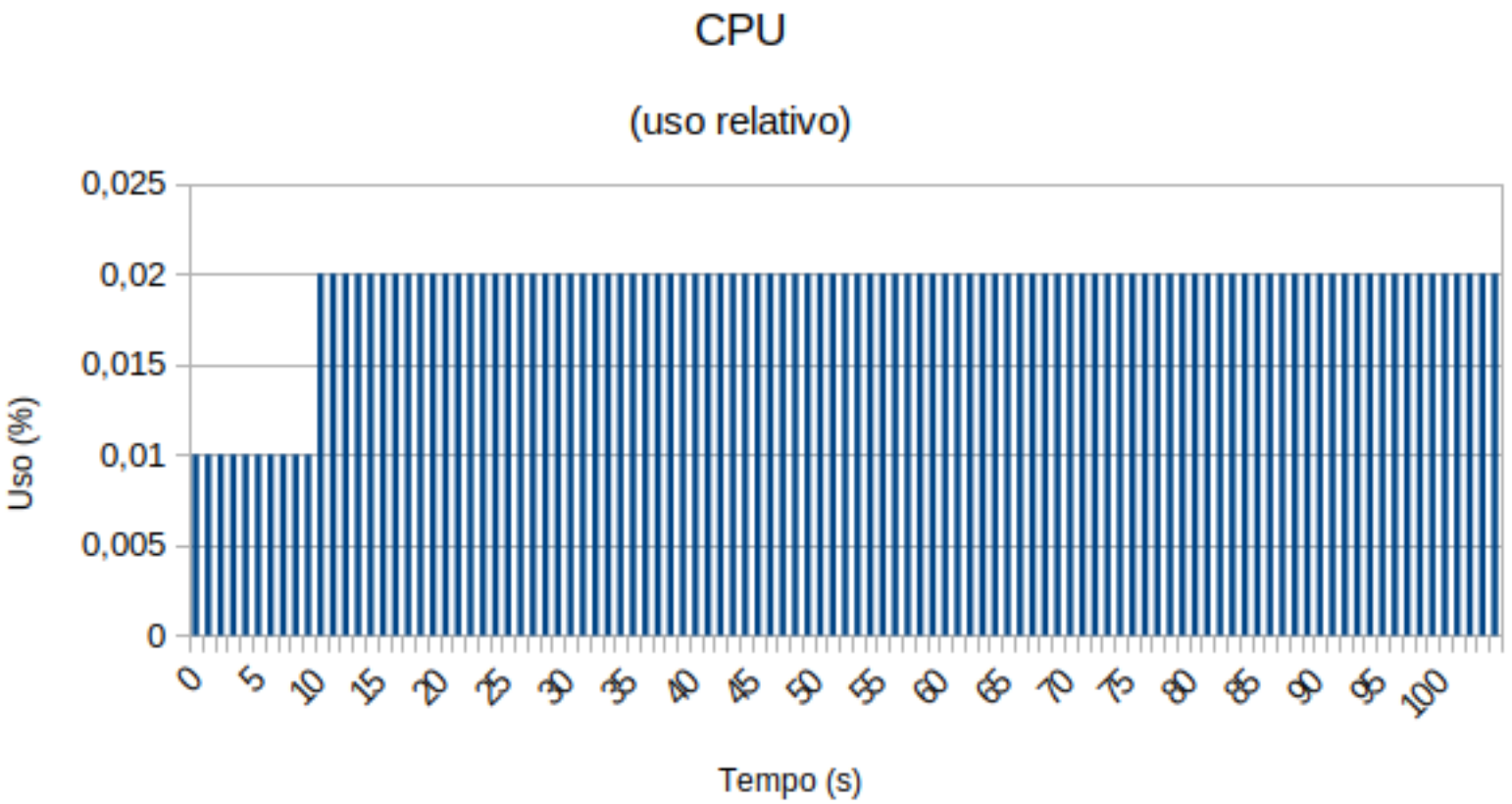
- Sleep de 10 segundos seguido pelo envio múltiplo de do sinal SIGINT* para todos os clientes mosquittos.

***:** $: kill -INT $(pgrep mosquitto_sub)

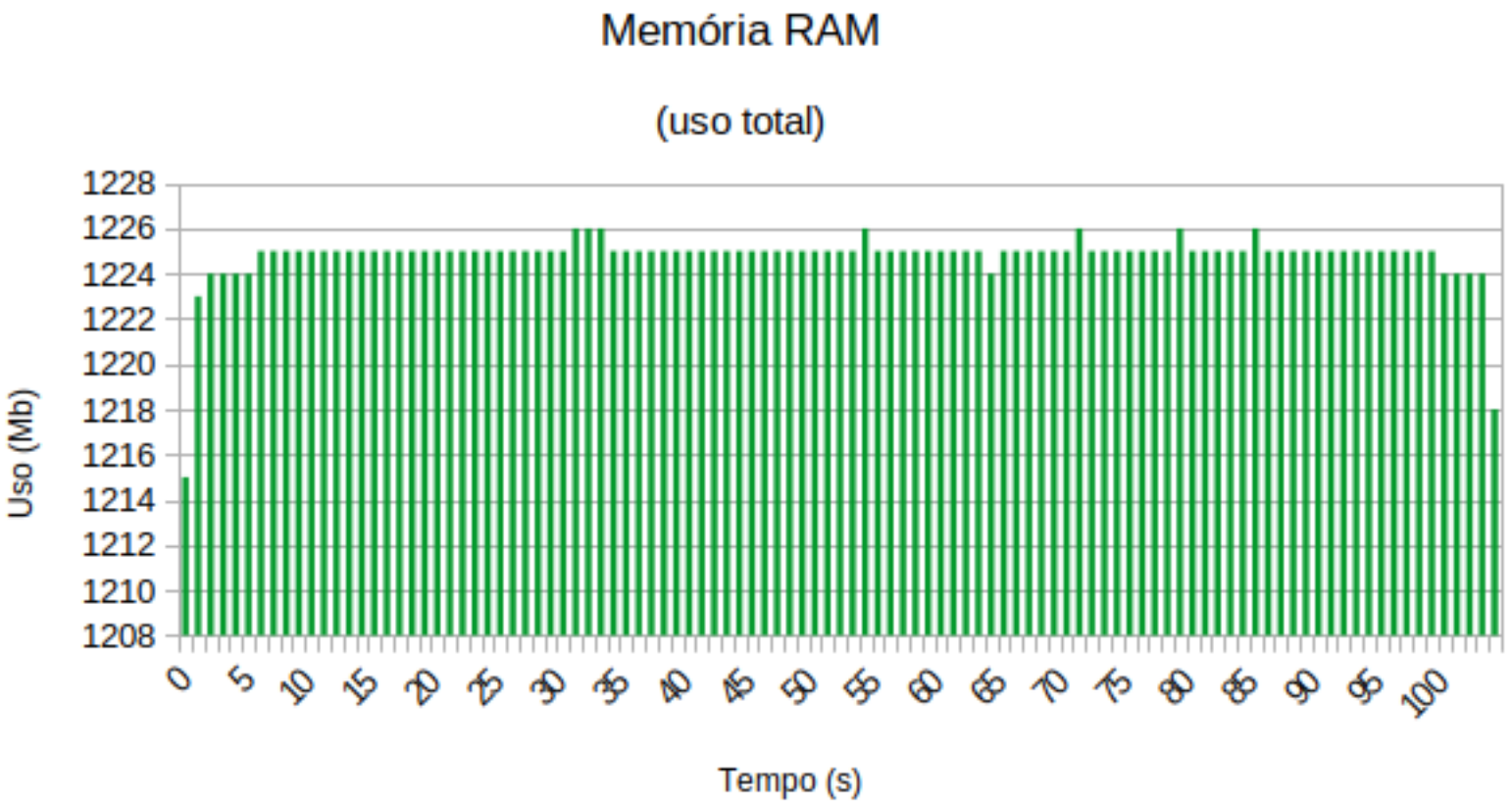
Desempenho - 1º Cenário – Somente Servidor



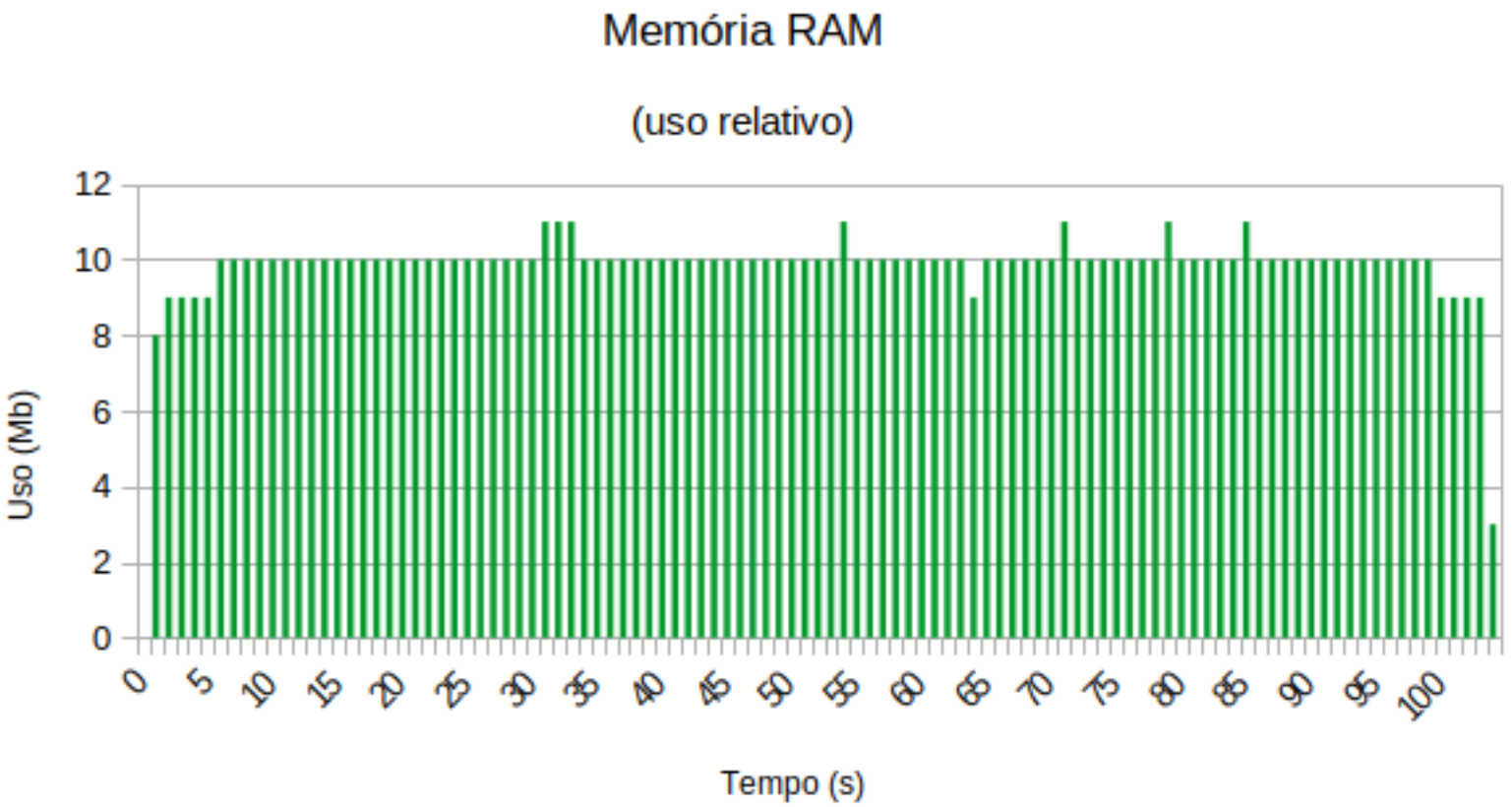
Desempenho - 1º Cenário – Somente Servidor



Desempenho - 1º Cenário – Somente Servidor



Desempenho - 1º Cenário – Somente Servidor

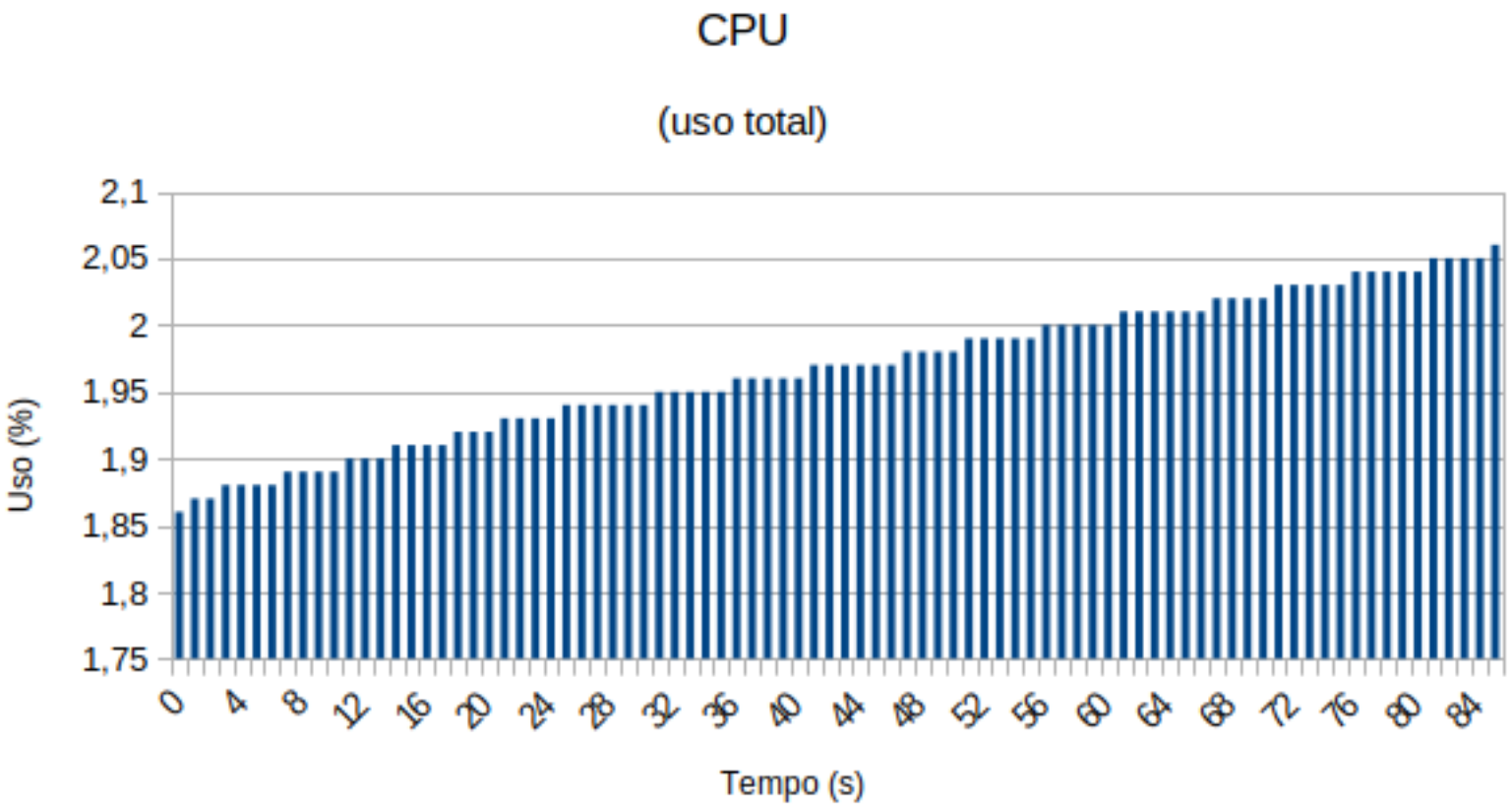


Desempenho - 1º Cenário – Somente Servidor

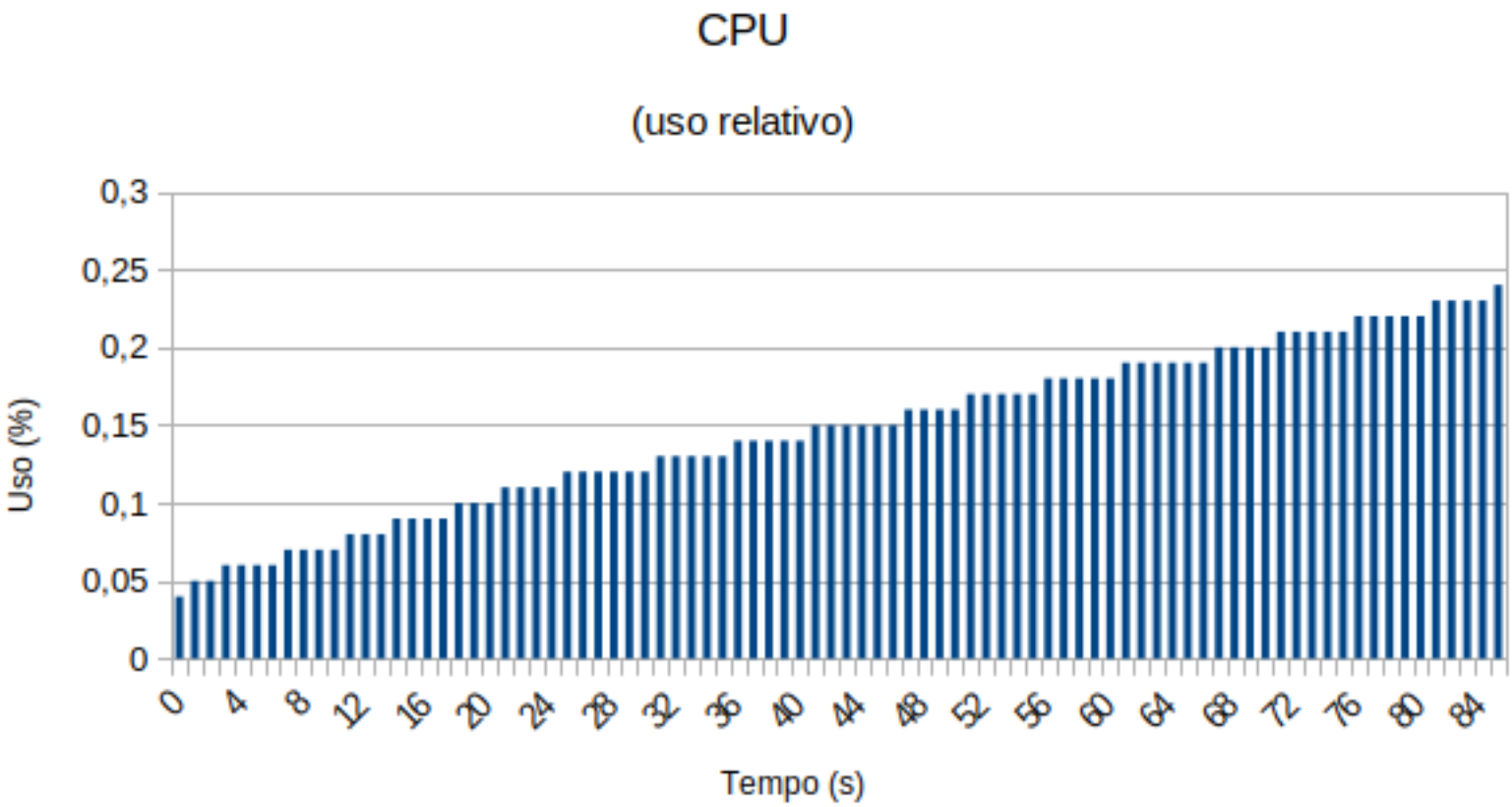
Dados:

	CPU (%) – Total	CPU (%) – Relativo	Memória RAM (Mb) – Total	Memória RAM (Mb) – Relativo
Média	3,070	0,020	1225,000	10,000
Desvio Padrão	0,003	0,003	1,259	1,259
Variância	0,000	0,000	1,585	1,585

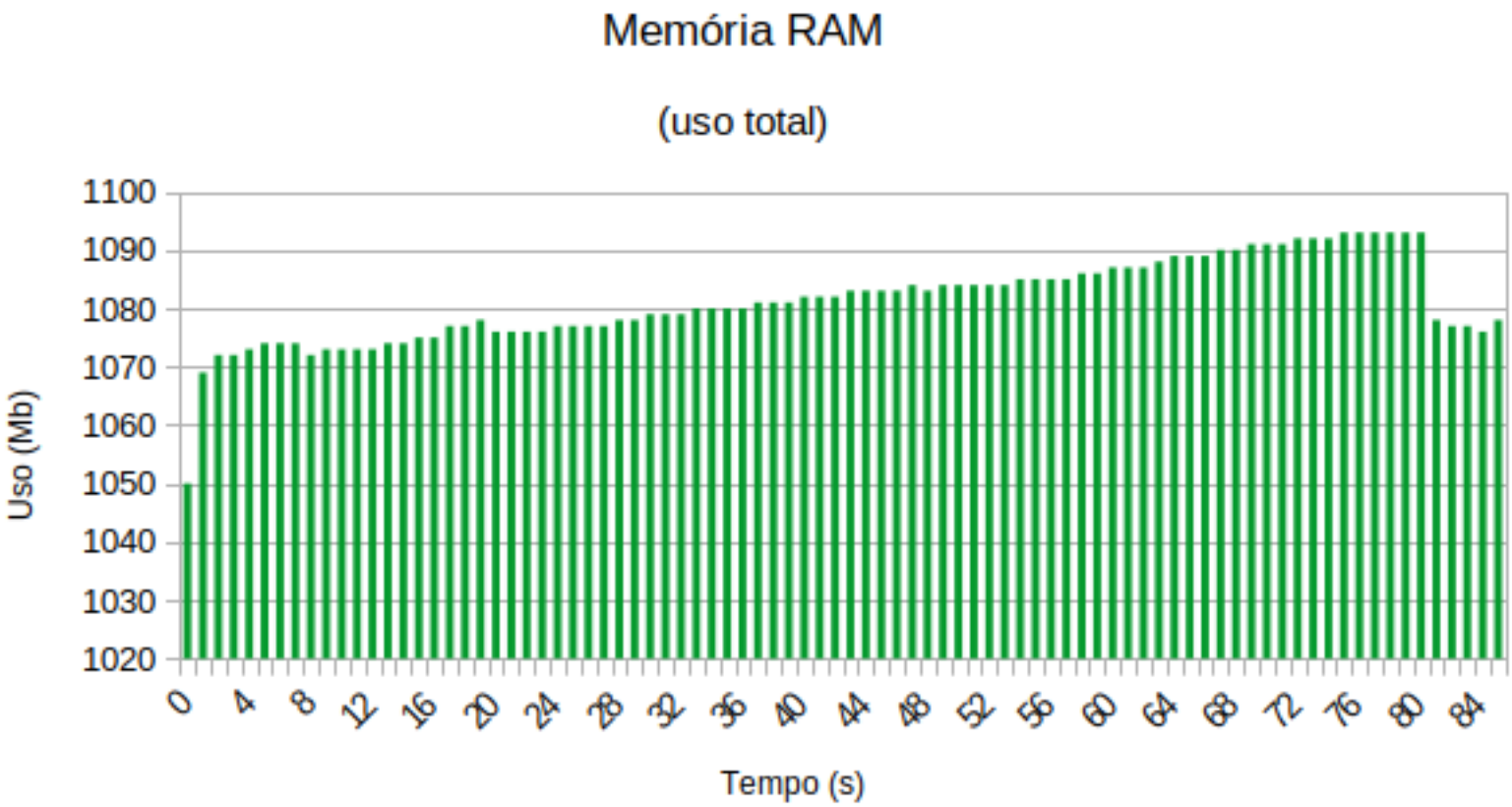
Desempenho - 2º Cenário – 100 clientes



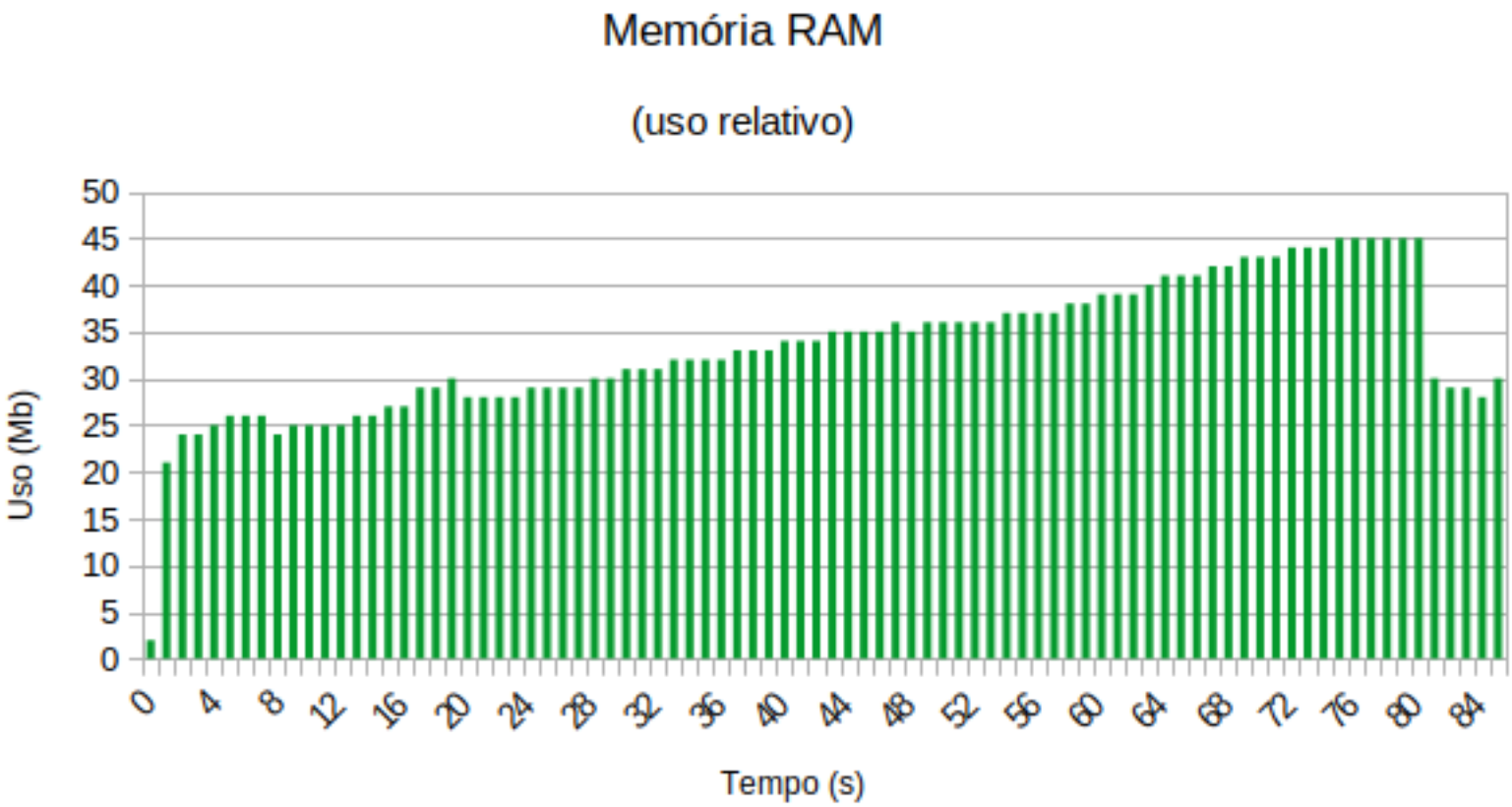
Desempenho - 2º Cenário – 100 clientes



Desempenho - 2º Cenário – 100 clientes



Desempenho - 2º Cenário – 100 clientes



Desempenho - 2º Cenário – 100 Clientes

Dados:

	CPU (%) – Total	CPU (%) – Relativo	Memória RAM (Mb) – Total	Memória RAM (Mb) – Relativo
Média	1,970	0,150	1081,000	33,000
Desvio Padrão	0,053	0,053	7,295	7,295
Variância	0,003	0,003	53,216	53,216

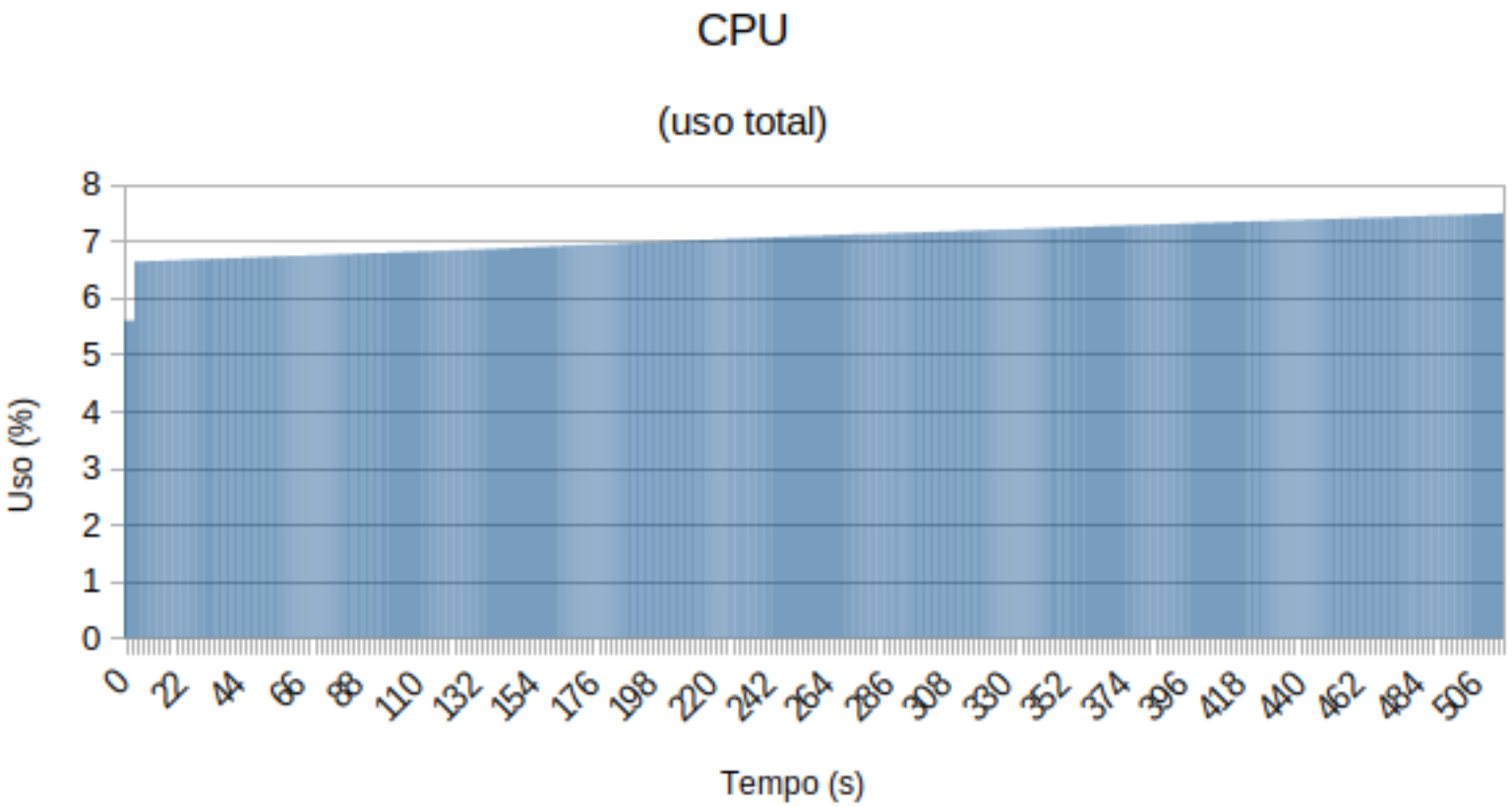
Desempenho - 2º Cenário – 100 clientes

WIRESHARK

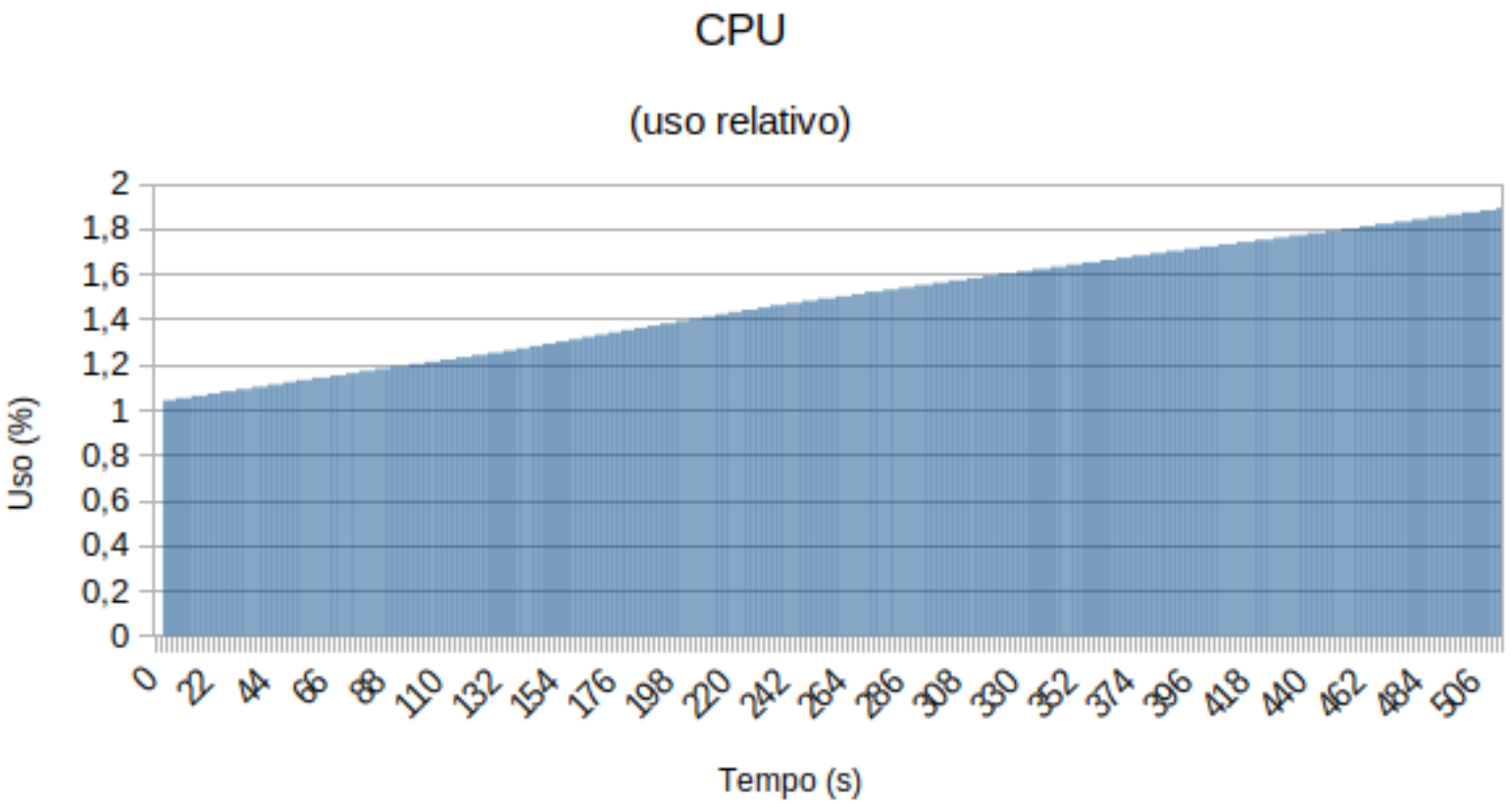
- O número de quadros referentes a todo o processo (clientes e servidor) foi de **1569** quadros

- O número de bytes referentes a todo o processo (clientes e servidor) **0.116** Mbytes (**116484** bytes)

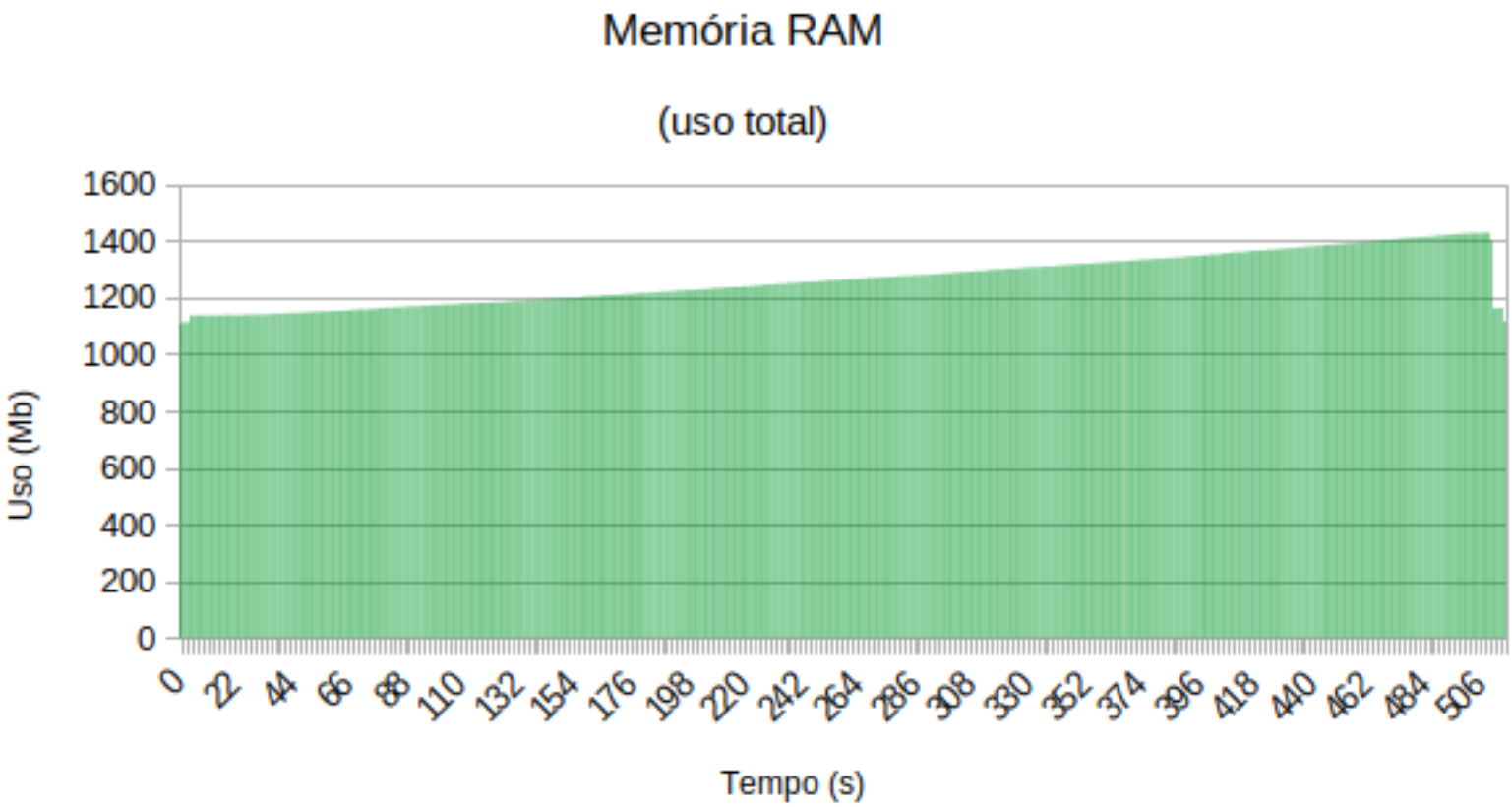
Desempenho - 3º Cenário – 1000 clientes



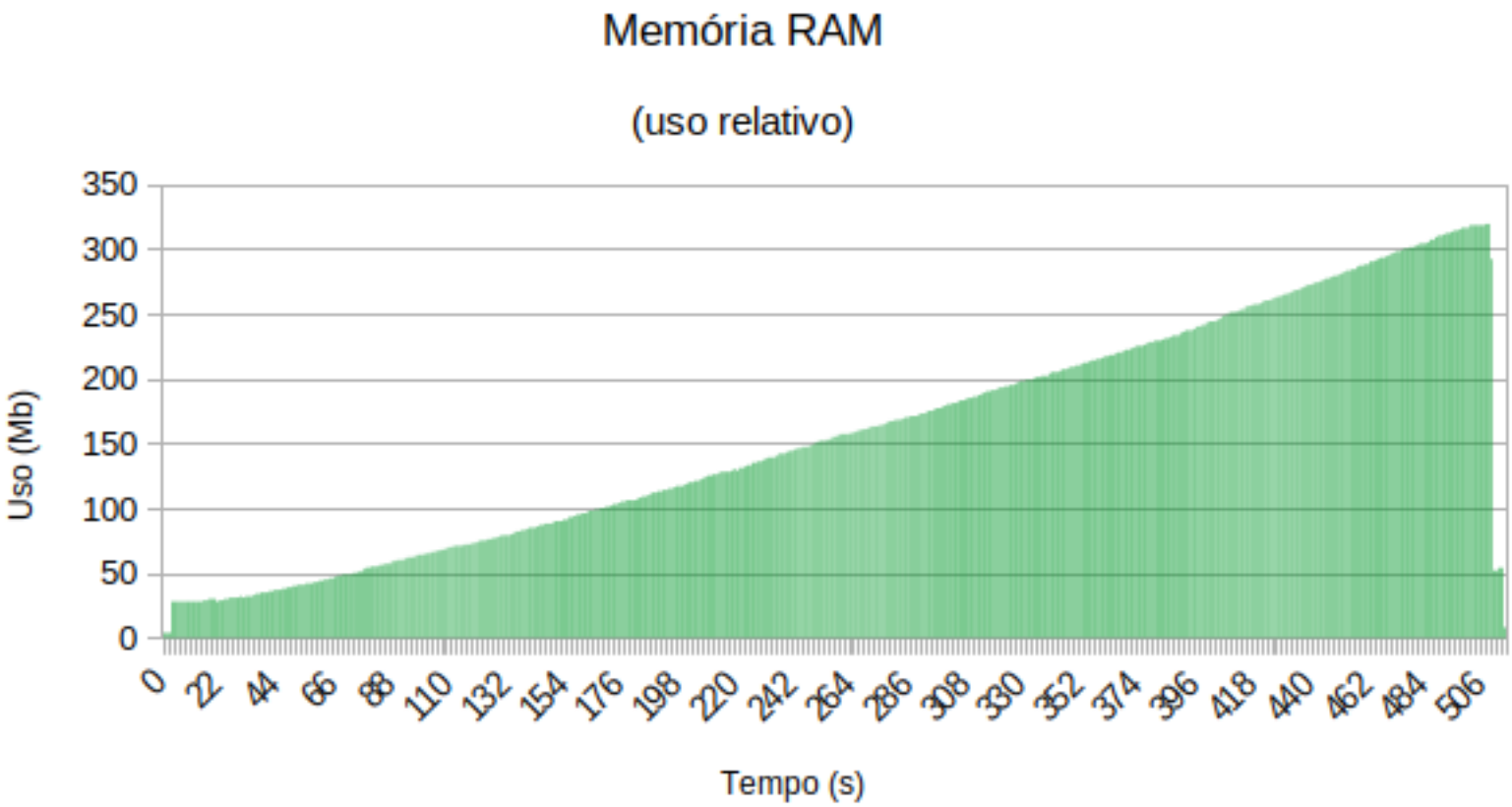
Desempenho - 3º Cenário – 1000 clientes



Desempenho - 3º Cenário – 1000 clientes



Desempenho - 3º Cenário – 1000 clientes



Desempenho - 3º Cenário – 1000 Clientes

Dados:

	CPU (%) – Total	CPU (%) – Relativo	Memória RAM (Mb) – Total	Memória RAM (Mb) – Relativo
Média	7,080	1,490	1260,000	152,000
Desvio Padrão	0,280	0,280	88,948	88,948
Variância	0,079	0,079	7911,828	7911,828

Desempenho - 3º Cenário – 1000 clientes

WIRESHARK

- O número de quadros referentes a todo o processo (clientes e servidor) foi de **34072** quadros

- O número de bytes referentes a todo o processo (clientes e servidor) **2.611** Mbytes (**2610799** bytes)

Desempenho - Conclusão

- O Desempenho relativo ao programa de CPU evoluiu de:

  0,02% (0 clientes) → **0,15%** (100 clientes) → **1,49%** (1000 clientes)

- A utilização máxima de memória RAM evoluiu:

  11Mb (0 clientes) → **45Mb** (100 clientes) → **319Mb** (1000 clientes)

  Obs: O aumento da entrada de clientes de 100 para 1000 teve um reflexo praticamente linear em uso de CPU e memória RAM.

Conclusão Final

O entendimento do funcionamento de um cliente do tipo broker é intrínseco a lógica da computação paralelizada (mesmo caso for implementado em um único processo). O conhecimento da comunicação em rede usando sockets junto com o funcionamento dos processos em um sistema UNIX garante o bom funcionamento do servidor.

O servidor reagiu bem a elevada carga de 1000 clientes comportando-se de forma linear. Que era um dos objetivos que influenciaram a escolha da estrutura de dados e constante alocação e desalocação de memória.

Bibliografia

- Cooper, M. (2014). Advanced bash scripting guide: An in-depth exploration of the art of shell scripting (Vol. 1). Domínio público, 10 Mar 2014.
- Kurose, J. and Ross, K., 2016. Computer Networking. Harlow, United Kingdom: Pearson Education Canada.

Sites:

- Oasis: docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html (*)
- Wikipedia: https://en.wikipedia.org (*)
- Michael Kerrisk: man7.org (*)
- Die(Dice): https://linux.die.net/ (*)

*: Último acesso 21/04/2022