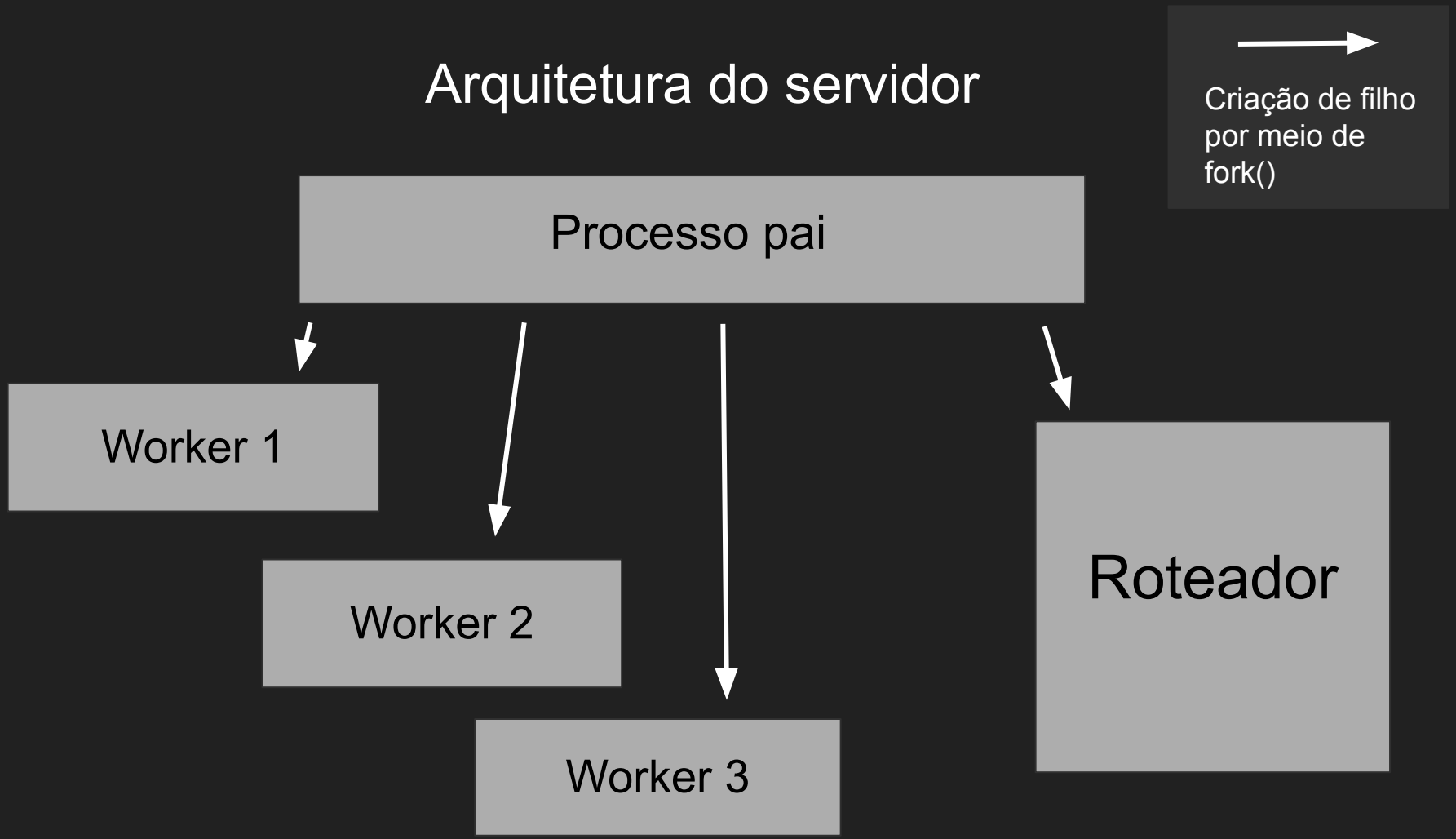
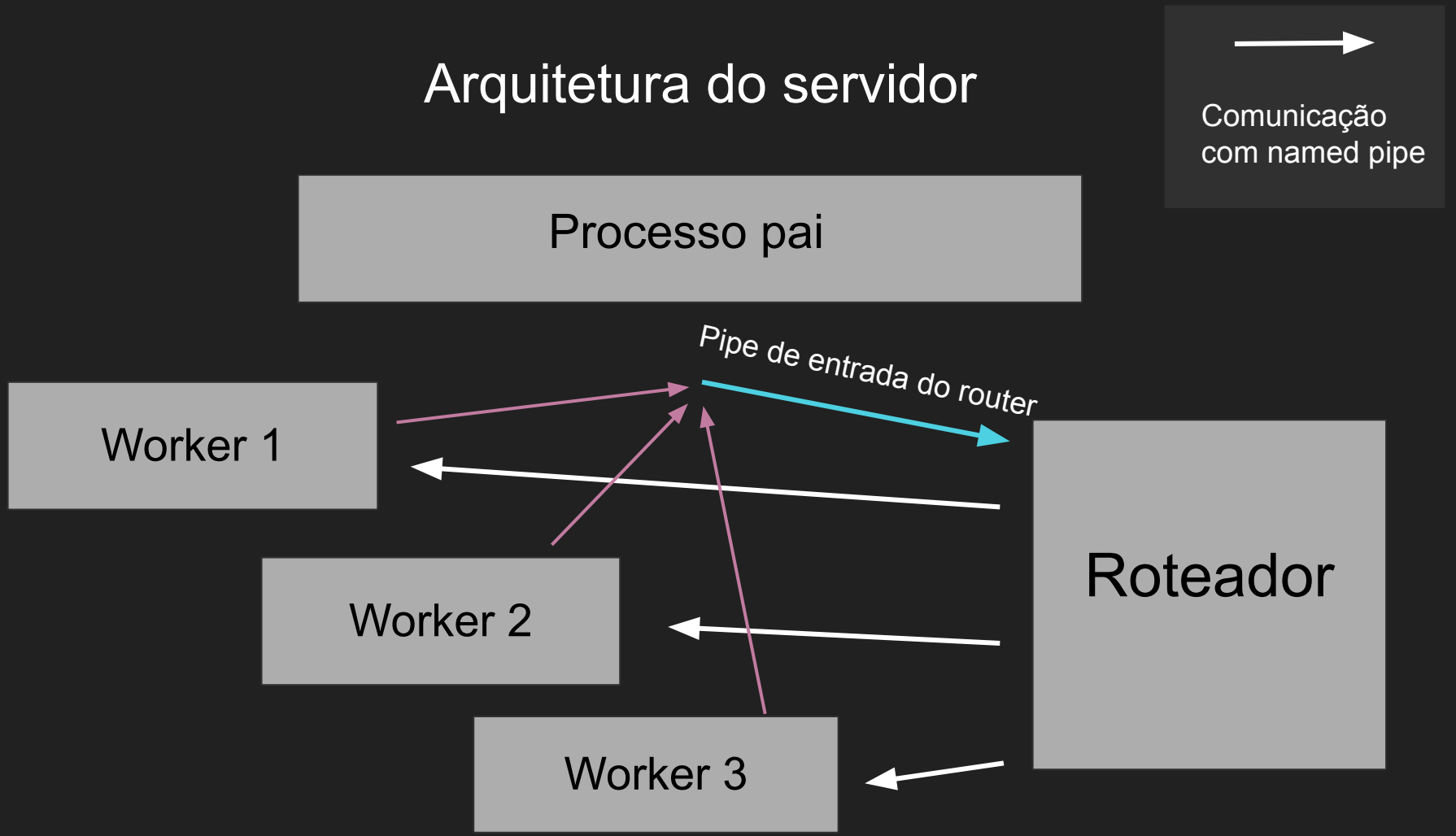
Implementação de um servidor de MQTT em C

João Renner Rudge, NUSP 11276221

Arquitetura do servidor



Arquitetura do servidor



Arquitetura do servidor

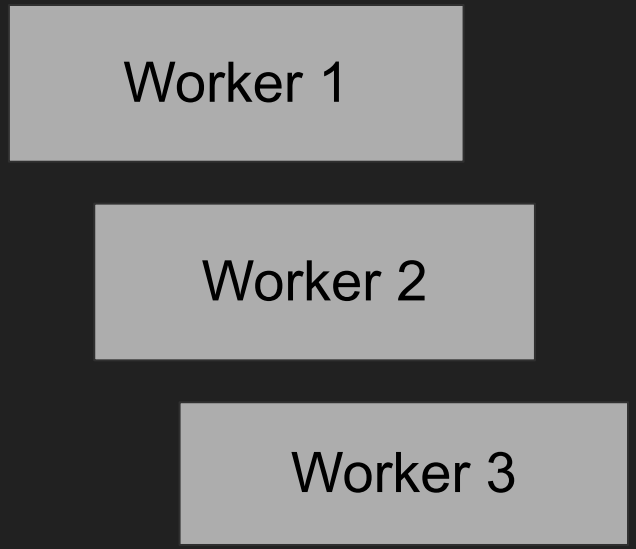
Processo pai

- Responsável por escutar por novas conexões, e criar processos filhos por meio da função fork
- Não se comunica com outros processos

Arquitetura do servidor

- Responsável por distribuir mensagens, e guardar relação de subscrições e pipes de workers
- Apenas pode ser acessado por um pipe, e os clientes usam locks para evitar situações de concorrência
- Guarda subscrições em uma lista ligada, com memória alocada de forma dinâmica

Arquitetura do servidor



- Se conectam diretamente com clientes
- Usam sockets não bloqueadores para poderem checar comunicações vindas dos clientes, assim como o router frequentemente.
- São terminados quando um cliente pede para desconectar-se

Hierarquia de funções

Principais funções do código que compõe o roteador:

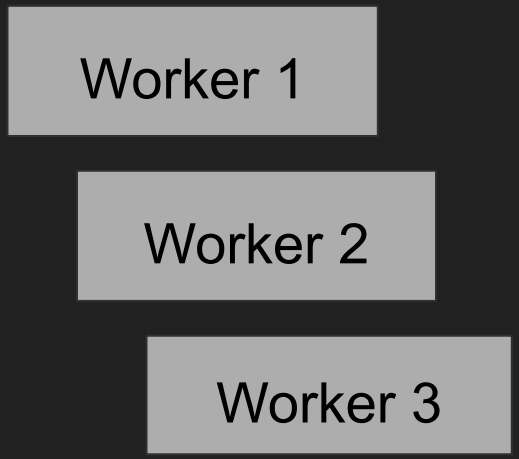
`void ROUTER ()`
-> Apenas lê a requisição e a redireciona para `ROUTER_handlePublish()` ou `ROUTER_handleSubscribe()`

`ROUTER_handleSubscribe()`
-> Adiciona node com o ID e tópico do cliente atual na lista ligada de subscriptions.

`ROUTER_handlePublish()`
-> Percorre a lista ligada de subscrições e notifica todo cliente que estiver marcado como subscrito naquele tópico

`notifyClient()`
-> Manda mensagem para pipe com o worker de um cliente no formato correto

Hierarquia de funções



Principais funções do código que compõe os workers:

`int handleClientConnection (int connectionSocket)`
-> Recebe o primeiro pacote de conexão, e lida com os pacotes subsequentes. Também recebe mensagens do roteador.

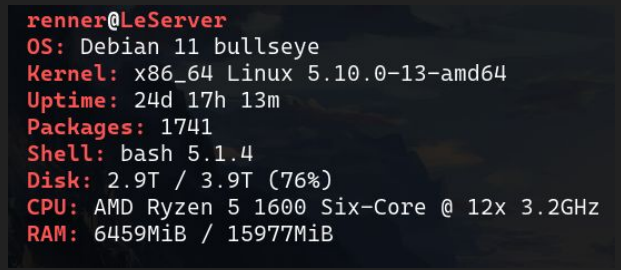
`void lockRouter ()`
-> Trava o named pipe do roteador, impedindo concorrência. O roteador se libera automaticamente depois de ser chamado.

`packet receivePacket (int connectionSocket)`
-> Recebe o pacote e o retorna para `handleClientConnection`

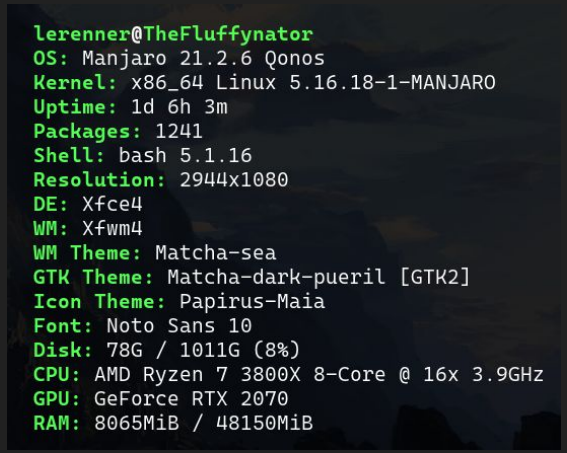
Análise da performance do programa

O programa foi compilado e executado em um sistema com Manjaro Linux, usando-se o compilador GCC 11.2.0 sem otimizações.

Nos testes de estresse, os múltiplos clientes são executados em meu servidor, que se encontra na mesma rede interna do meu computador pessoal, podendo se comunicar por uma conexão ethernet gigabit.



Meu servidor



Meu computador pessoal

Análise da performance - CPU

- Sem novas conexões, o programa usa uma quantidade negligível de tempo de CPU

- Com 100 clientes conectados ao mesmo tempo, o uso de CPU pelo programa aumenta, mas o inesperado é ver que a cpu fica com utilização de quase 100% por causa de tarefas de kernel

- Com 1000 clientes conectados ao mesmo tempo, o computador começa a ficar irresponsivo, uma vez que mesmo com 10 vezes menos clientes a cpu já estava com 100% de utilização