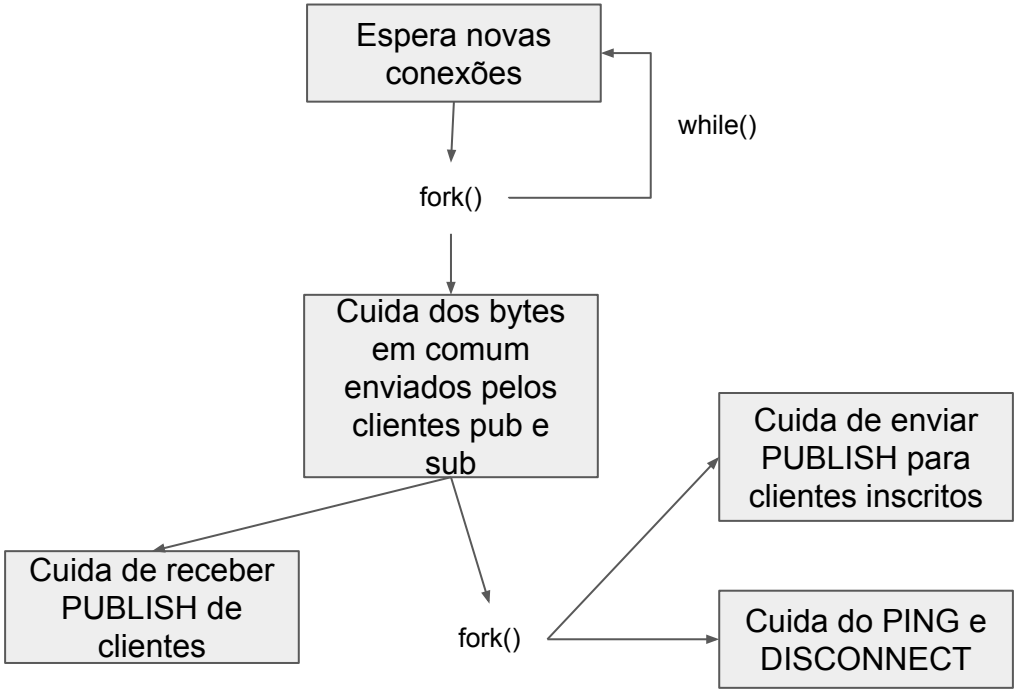
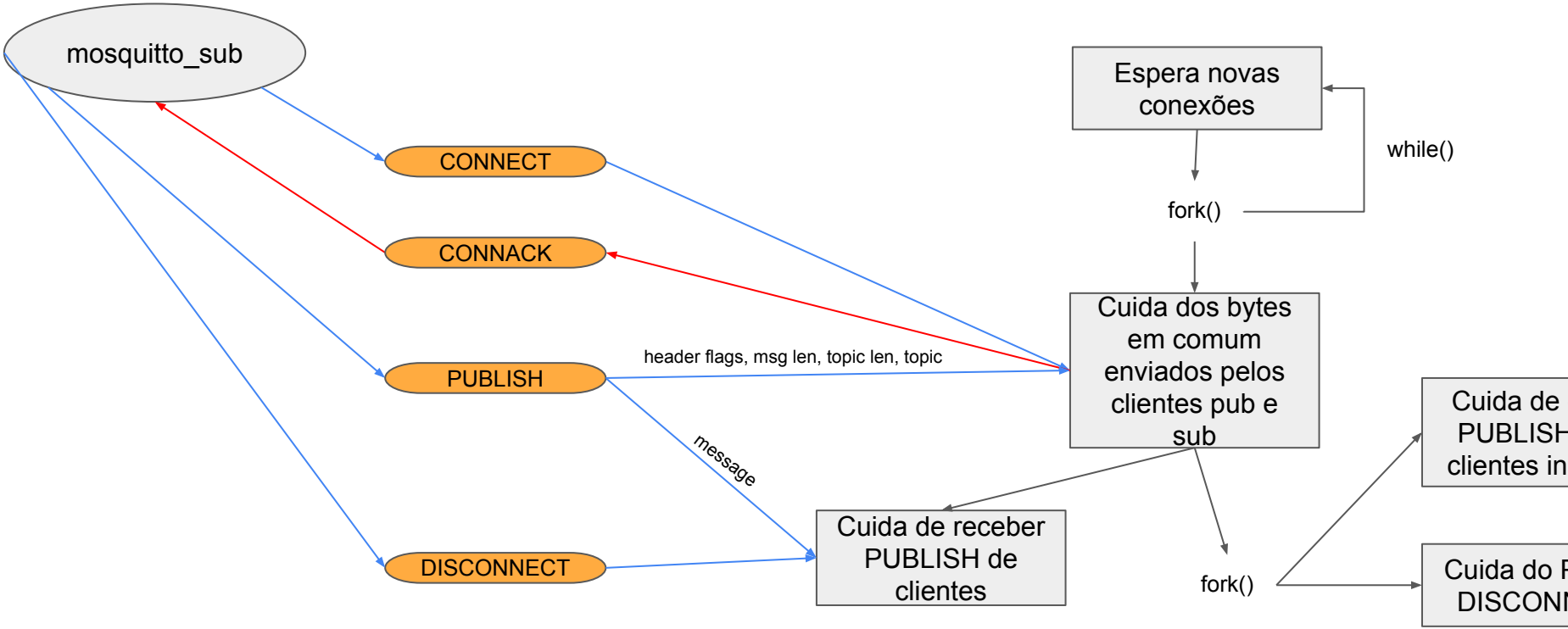
Estrutura geral

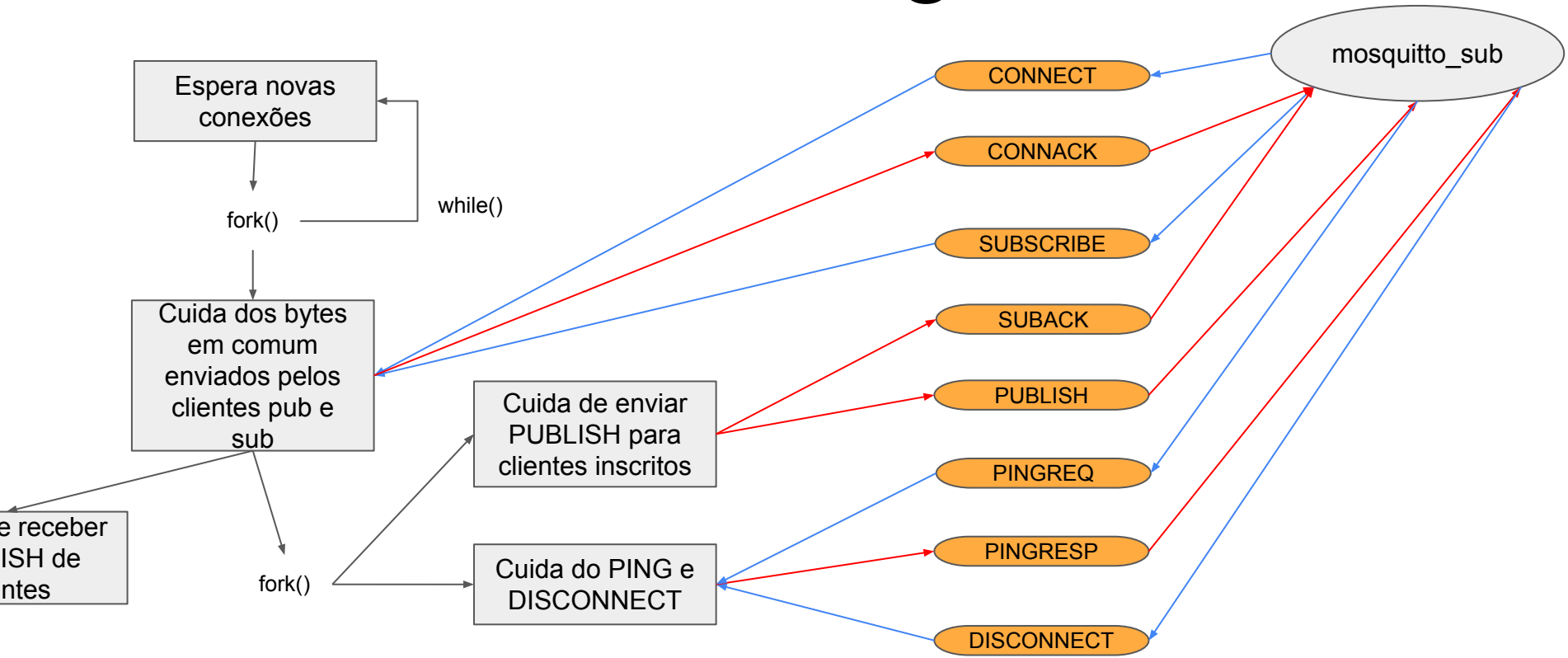


Estrutura geral



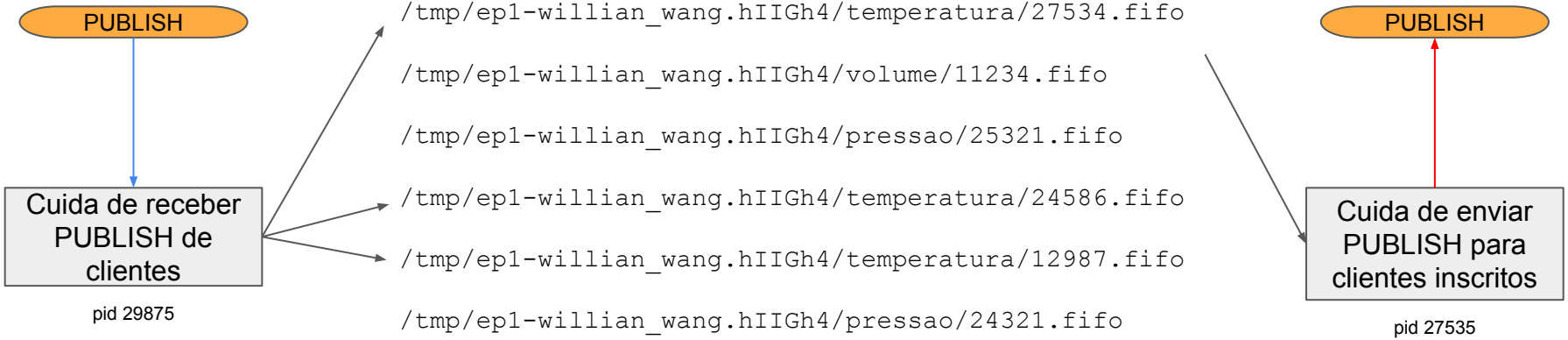
Um processo novo é criado e destruído com a conexão do mosquitto_pub

Estrutura geral



Dois processos novos são criados e destruídos com a conexão do mosquitto_sub

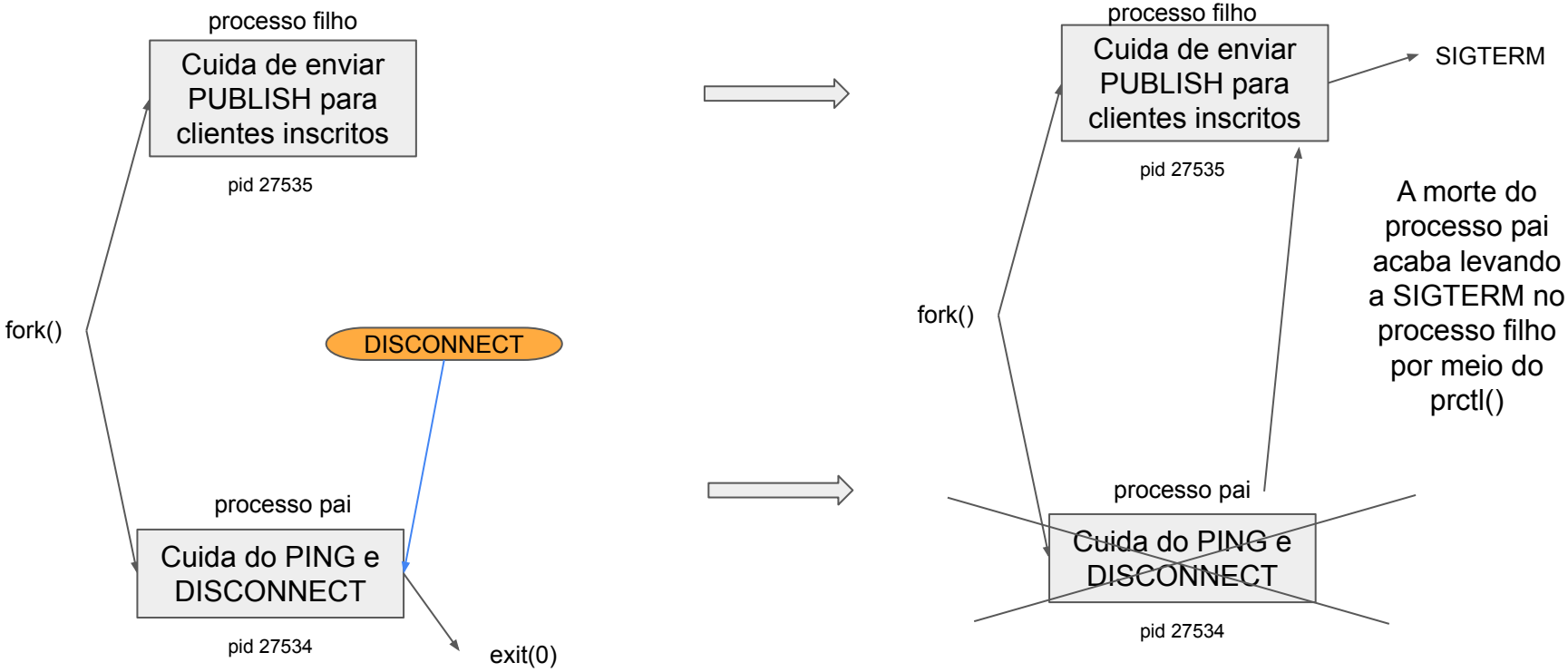
Comunicação entre processos



O processo cuidando de receber o PUBLISH dos clientes deve escrever a mensagem completa do PUBLISH (incluindo header flags, byte por byte) em cada um dos arquviso FIFO listados no diretório temporário correspondente ao tópico.

O processo que está cuidando de enviar as mensagens para os clientes inscritos ficará lendo o arquivo FIFO sem parar e redirecionando ao cliente qualquer byte que tenha conseguido ler. Note que o nome do arquivo FIFO usa o PID do processo pai, já que ele foi criado antes de realizar o segundo fork().

Comunicação entre processos



Decisões de projeto

- Processamento do conteúdo das mensagens do protocolo MQTT
  - Por causa do modelo de stream dos bytes, o encapsulamento do descritor de arquivo em um FILE possibilitou o uso de getc() para consumir bytes individualmente, sem precisar ficar manipulando os bytes do buffer do read().

- Concorrência
  - A escrita da mensagem de PUBLISH nos arquivos FIFO são feitos em sua forma integral em um único open() e close(), evitando assim a mistura de bytes de diferentes mensagens PUBLISH.

- Performance
  - Apesar do overhead do fork() em relação ao uso de threads, nenhuma memória de tamanho significativo é copiado de maneira redundante e a programação do código ficou mais simples.
  - O maior consumo de memória possível é do conteúdo da mensagem em PUBLISH, que é guardada de maneira integral na memória ao invés de consumir ele por pedaços. No entanto, essa decisão facilitou bastante na resolução de problemas de concorrência como descrito anteriormente.

Testes - metodologia

- O servidor broker foi instalado em dentro de um ambiente Docker, na qual a porta 1883 foi exposta, com o Wireshark confirmando que a conexão aconteceu entre os IPs 172.27.0.1 (cliente) e 172.27.0.2 (servidor).
- Todos os dados foram coletados através do comando “docker stats”, que, de forma geral, apresentou dados mais corretos do que outros métodos envolvendo somar manualmente vários números de baixa precisão numérica indicando o uso de recursos de cada processo. A taxa de amostragem máxima conseguida por esse método foi de 0.5/segundo.
- O script run_tests.sh foi criado para automatizar a chamada de clientes e salvar os dados relevantes.
- Hardware utilizado:
  - 1 core com Intel(R) Xeon(R) Platinum 8171M CPU @ 2.60GHz
  - 1 GB RAM + 3 GB swap

Testes - CPU

I. *Apenas o broker, sem nenhum cliente conectado:*
   O uso de CPU indicou 0.0%. A função accept() bloqueia o processo, fazendo com o uso real de CPU realmente seja nulo.

II. Broker + 100 mosquitto_sub + 100 mosquitto_pub, 32 tópicos distribuídos entre os clientes:
   max CPU: 6.8%
   …

III. Broker + 1000 clientes sub + 1000 clientes pub
   max CPU: 10%
   …