Data: June 13, 2018
Aluno: Ramon Melo
Professor: Daniel Figueiredo

1 Objetivo

Construir um sistema distribudo cujo mecanismo de ordenao total de eventos seja baseado no algoritmo *Totally Ordered Multicast.*

2 Decises de Projeto

Balance used *#4*
Magnesium from sample bottle *#1*

Ao contrrio dos trabalhos anteriores, a ordenao total de eventos distribudos ocorre numa camada de abstrao significativamente acima do hardware, de forma que implementaes de baixo nvel e acesso direto ao metal no compem mais o conjunto de ferramentas desejvel ao desenvolvimento da aplicao. Pelo contrrio, bibliotecas de sockets, processos e *user-level threads* esto disponveis em praticamente todas as linguagens de programao. Desta forma, deseja-se do ecossistema caractersticas como o suporte a construtos de programao orientada a objetos, tais como herana e polimorfismo; legibilidade do cdigo; e agnosticismo em relao ao sistema operacional.

A linguagem escolhida para este trabalho foi Python 3.5, edio que nativa maioria esmagadora de sistemas operacionais baseados no padro POSIX. Em especial, esta verso significativamente mais eficiente que as baseadas em Python 2 - mais tradicionais - e traz nativamente bibliotecas como a `multiprocessing`Foundation (2017) (*user-level threads* em concorrncia, uma das poucas implementaes que suporta o ecossistema Windows).

Dada a natureza dos requisitos, que exige estruturas de dados razoavelmente similares mas autnomas, e ao prazo de entrega, que forou a realizao de *sprints* bastante curtos (em mdia, dois por semana), o paradigma de orientao objetos foi uma escolha natural.

3 Implementao

H cinco classes que serviram de fundao para o trabalho. O diagrama de classes est representado graficamente na Figura 1.

A classe `Process` encapsula funcionalidades dos mdulos `socket` McMillan (2017) e `multiprocessing`. importante observar que este ltimo expe a

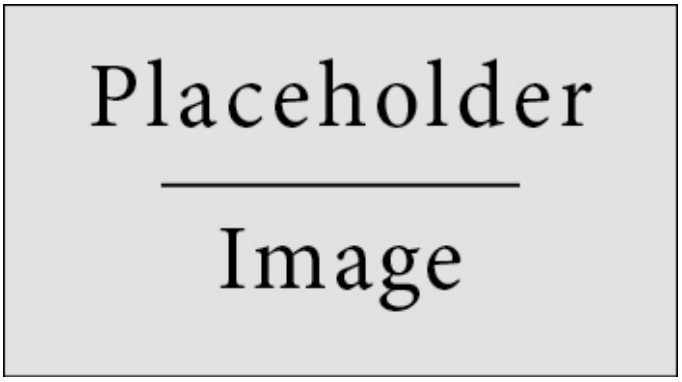

Figure 1: Figure caption.

mesma *API* tanto para processos, quanto para *threads*. Devido aos requerimentos do enunciado, foi utilizado somente o *namespace* `dummy`, que inicia *user-level threads* dentro do mesmo processo e permite o compartilhamento implcito de memria entre elas.

A classe `Thread` concentra os mtodos executados concorrentemente. Em especial, o mtodo `run()` foi construdo desde o princpio de forma limitada ao padro *thread-safe*. Herdam desta classe `ListenerThread` (orientada a servios de servidor) e `EmitterThread` (orientada a servios de cliente).

Os objetos que representam os eventos so derivados da classe `Event`. Particularmente, os que representam as mensagens derivam da classe `Message`. So eles: `SentMessage` (mensagem enviada pelo processo que a criou), `ReceivedMessage` (mensagem recebida pelo processo que a criou) e `AckMessage` (confirmao a ser enviada pelo processo que a criou). As mensagens so as responsveis pela execuo, visto que representam os eventos na abstrao do mecanismo *Totally Ordered Multicast* enunciada a este trabalho. Portanto, a classe `ReceivedMessage` a responsvel por gravar pertinentemente as mensagens no disco uma vez que estejam aptas.

O relgio lgico de Lamport representado pela classe `LogicalClock`, que encapsula cadeados (*locks*) para a manuteno do carter *thread-safe* do mtodo `Thread.run()`.

Por fim, as mensagens que aguardam execuo so armazenadas num objeto `MessageQueue`, que encapsula uma fila do tipo *FIFO* (*First In, First Out*, "fila indiana") e cadeados para seu acesso concorrente.

Todos os objetos so pertencentes ao objeto `Process` que coordena a execuo local do programa. Isto porque, para garantir o acesso implcito memria compartilhada, o mdulo `multiprocessing` exige que as estruturas de dados sejam referenciadas pela abstrao do processo (e no pela abstrao das *threads*).

Mass of magnesium metal = sdsdsd

4 Estudos de Caso

5 Consideraes Finais

The most obvious source of experimental uncertainty is the limited precision of the balance. Other potential sources of experimental uncertainty are: the reaction might not be complete; if not enough time was allowed for total oxidation, less than complete oxidation of the magnesium might have, in part, reacted with nitrogen in the air (incorrect reaction); the magnesium oxide might have absorbed water from the air, and thus weigh "too much." Because the result obtained is close to the accepted value it is possible that some of these experimental uncertainties have fortuitously cancelled one another.

Random citation Tyrvinen (2016) embeddeed in text.

References

Foundation, P. S. (2017). Official 3.5.2 documentation: Process-based parallelism. `https://docs.python.org/3.5/library/multiprocessing.html`. Accessado em 14/06/2018.

McMillan, G. (2017). Official 3.5.2 documentation: Socket programming. `https://docs.python.org/3/howto/sockets.html`. Accessado em 14/06/2018.

Tyrvinen, J. (2016). Concurrent solution for lamport clocks. `https://github.com/religiosa/lamportClocks`. Accessado em 14/06/2018.