# Trabalho Prático - 3

Francisco J. Lucca, Marlon Varoni, João Almeida

30 de novembro de 2021

## 1 Introdução

Neste relatório será descrita a implementação e demonstração de resultados obtidos sob o trabalho prático 3, estipulado pelo professor Sérgio Johanm. Este trabalho tem como objetivo aprofundar o conhecimento relacionado aos tópicos abordados até o momento, que incluem o modelo cliente/servidor, RPC, RMI, P2P, comunicação coletiva e relógios lógicos.

A tarefa consiste na implementação de um programa distribuído que implemente o algoritmo de Berkeley para sincronização de diversos processos executando em nodos independentes. Este programa deve receber como entrada um arquivo de configuração e um número que identificará o processo em uma das linhas do arquivo de configuração. O processo mestre será identificado pela *id* com valor 0. Todos os processos devem possuir acesso a uma cópia do arquivo.

## 2 Organização do Código

O programa codificado em Python utiliza bibliotecas nativas da linguagem para controle de threads, sockets e tempo de processamento.

### 2.1 Arquivo de Configuração

Cada linha do arquivo de configuração possui o formato **id;host;port;time;delay**, onde:

1. *id* é um número inteiro que identifica o processo;
2. *host* é o hostname ou endereço IP da máquina (nodo) que executa o processo;
3. *port* é o número da porta que o processo vai escutar;
4. *time* é a hora com a qual o processo será iniciado;
5. *delay* é o atraso de processamento no processo (em ms);

### 2.2 Funções Implementadas

Para desenvolvimento do sistema criamos dois arquivos, um para instanciação do nodo master e outro para instanciação dos nodos slaves. Ambos são instanciados através do arquivo **main.py**, conforme será demonstrado no capítulo 3. A seguir explicamos as funções dos nodos master e slave:

- recebeClockTime: Utilizado para receber o tempo do relógio de um *client* conectado informado por parâmetro. Os valores de hora, diferença de tempo e do *client* conectado são armazenados em um *array*.
- fazConexao: Função de *thread* mestre implementada para aceitar um *client* através da porta informada. A conexão é criada por meio do método *accept*, que retorna um novo objeto de *socket* para enviar e receber dados na conexão, e um endereço vinculado ao *socket* na outra extremidade da conexão.

- mediaDiferencaClock: Soma todas as diferenças e realiza o cálculo da diferença média do relógio.
- sincronizaClocks: É a *thread* de sincronização mestre usada para gerar ciclos de sincronização do relógio na rede. O tempo de sincronização é obtido pelo soma do tempo atual com a media de diferença de tempo calculada anteriormente em *getAverageClockDiff*.
- iniciaClockMaster: Função usada para inicializar o nodo mestre na porta informada por parâmetro. Nesta função é criado o *socket* e posteriormente é inicializado as *threads* de sincronização.
- mandaClock: Fornece o tempo do relógio no cliente que é enviado a cada 5 segundos.
- recebeClock: É a *thread* do *client* usada para receber o tempo sincronizado por parâmetro.
- inicialClockNodo: Inicializa um nodo escravo, criando o *socket* e as *threads* necessárias para sincronização.

# 3 Instanciando Processos

Os processos master e slave da aplicação podem ser gerados por meio da execução dos comandos abaixo, respectivamente:

```
$ python main.py 0 <host> <port> <time> <delay>

$ python main.py <id> <host> <port> <time> <delay>
```

# 4 Demonstração da Implementação

Para a demostração da implementação serão usadas três maquinas virtuais, uma delas irá rodar o nodo *Master* e todas as outras nodos de clientes. O que esperamos é que todas elas estejam sincronizadas.

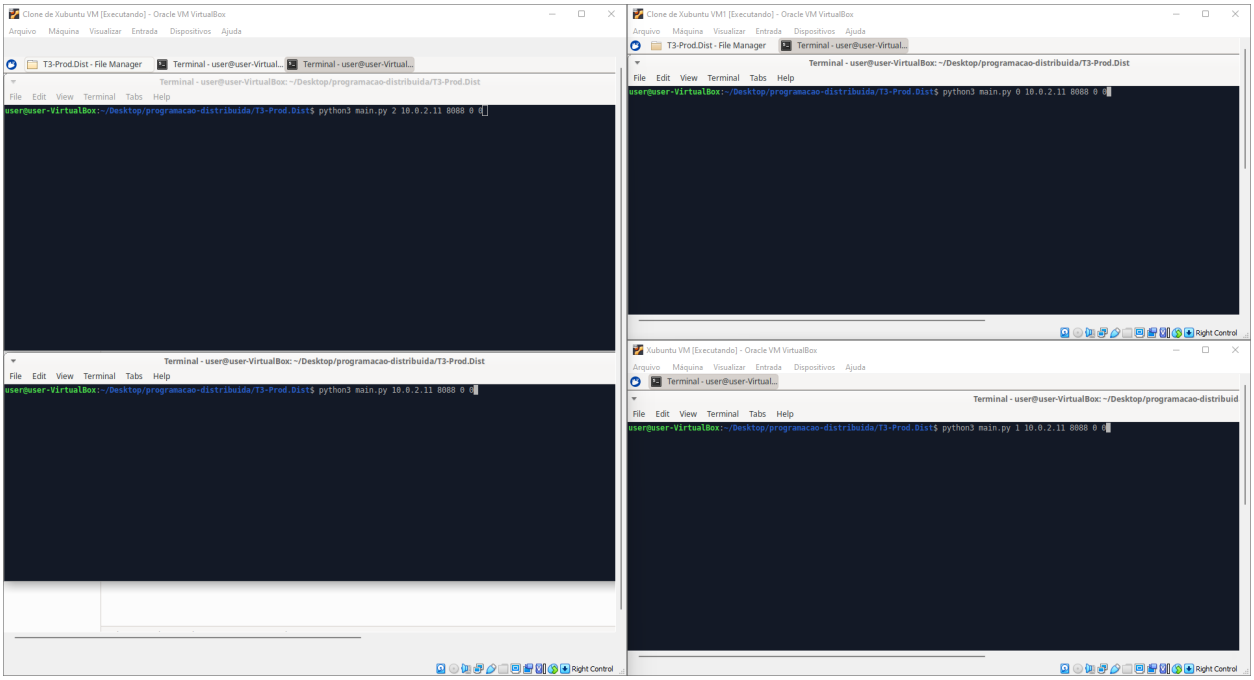

Figura 1: Configuração inicial

Na figura acima temos uma maquina que irá rodar o nodo mestre (superior direto), o endereço usado para rodar o nodo mestre é: *10.0.2.11:8088* e os clientes que irão se conectar nela.

Logo ao executar o nodo mestre temos o seguinte *output*:

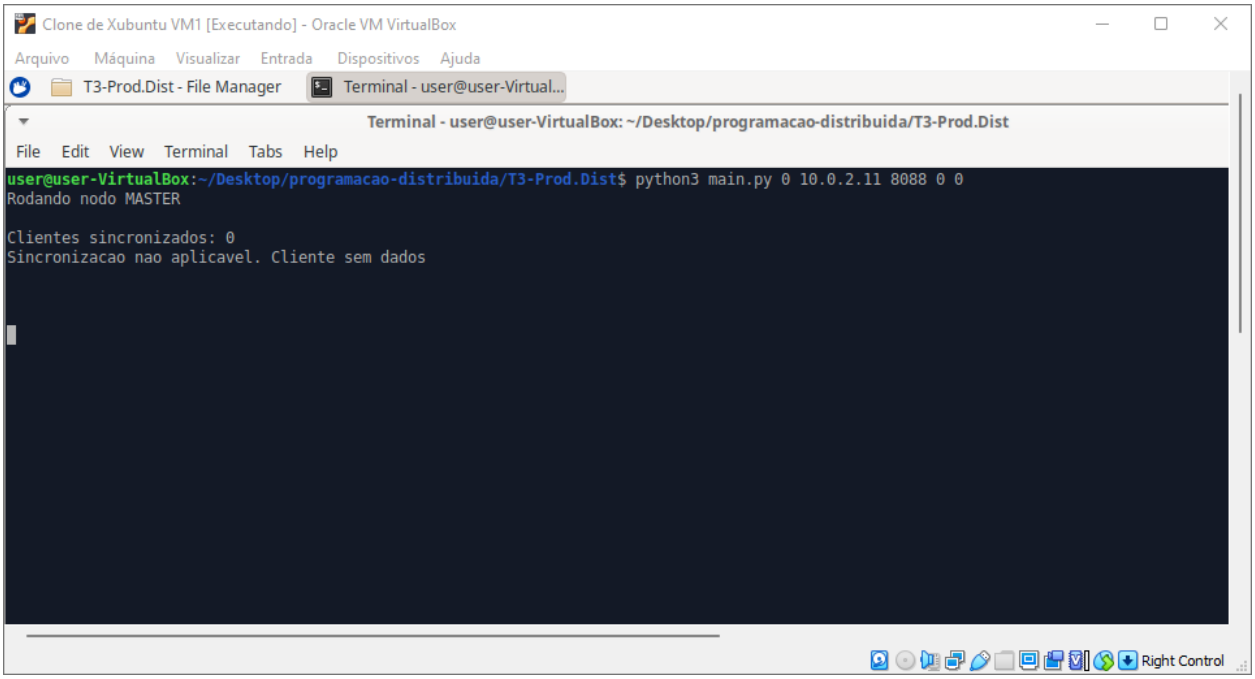


Figura 2: Nodo mestre

Ao executarmos os clientes temos os seguintes *outputs*:

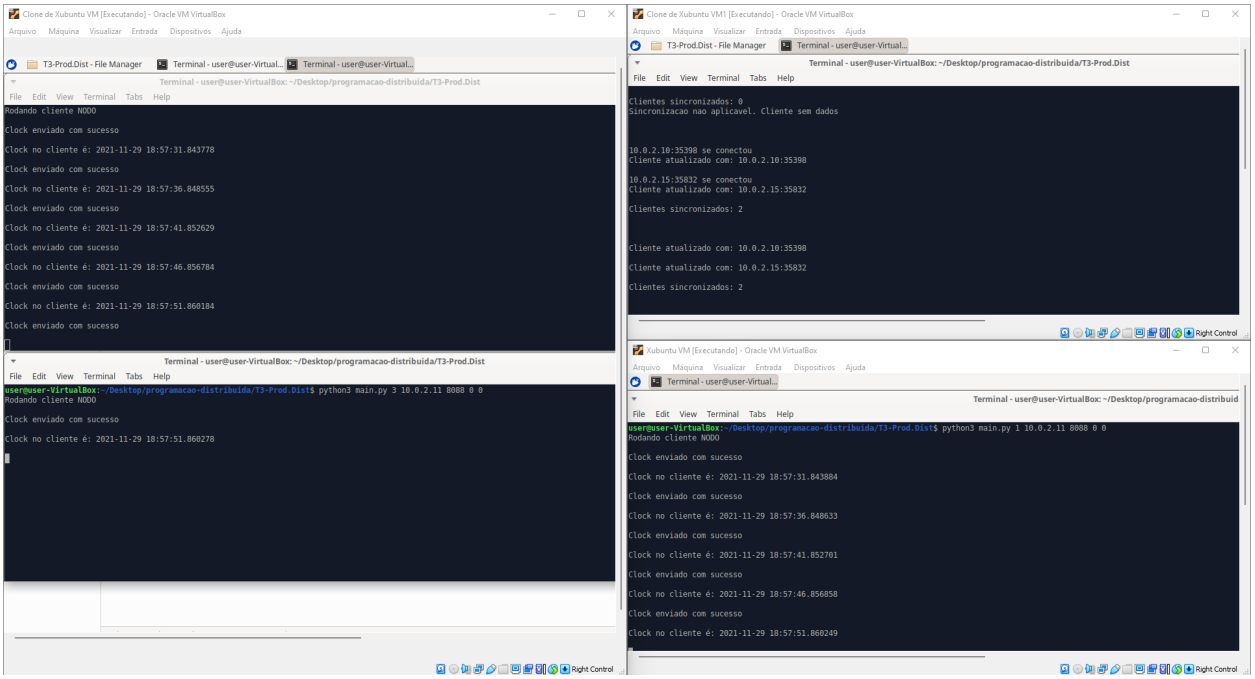

Figura 3: Executando os nodos

# 5 Conclusão

Concluímos com o desenvolvimento deste trabalho que é possível obter um bom resultado ao implementar o algoritmo de Berkeley para sincronização de processos executados em nodos independentes.