



USP - UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E COMPUTAÇÃO (ICMC)
DISCIPLINA SISTEMAS OPERACIONAIS
DOCENTE RODOLFO IPOLITO MENEGUETTE

PROJETO FINAL
SISTEMA DE POSTO DE COMBUSTÍVEL

Carlos Filipe de Castro Lemos (12542630)
Gabriel Natal Coutinho (12543461)
João Gabriel Sasseron Roberto Amorim (12542564)
Rafael Zimmer (12542612)

SÃO CARLOS
2022

1. Introdução

Durante o semestre, foi lecionado na disciplina de Sistemas Operacionais sobre o uso e o funcionamento de Threads, Semáforos e Travas. De fato, há diversas tarefas em que suas técnicas não são apenas recomendadas, mas necessárias, como na computação paralela, em alta escala, em nuvem, para treinamento de modelos estatísticos ou operações em dados vetorizados.

No entanto, o uso das Threads podem provocar cenários repletos de complicações e problemas, devido a condições de concorrência ou impasses, como, por exemplo, “deadlocks”, “starvation”, gerando problemas conhecidos e debatidos, como o Jantar dos Filósofos. Com efeito, para solucionar esses tipos de problemas, foi ensinado sobre a utilização de Semáforos (Semaphoro) e Travas (Locks).

Nesse contexto, o professor especificou um trabalho final em grupo de 4 (quatro) pessoas com a finalidade de que fosse implementado um sistema em Python, C, C# ou C++ que aplicasse conceitos de threads e de semáforos discutidos e apresentados em sala de aula.

Assim, como projeto final da disciplina, propusemos um cenário de um posto de combustível, no qual existem clientes para serem atendidos durante o expediente (até o limite de 50) e recursos de abastecimento limitados (materializados em 5 bombas de gasolina que funcionam simultaneamente). Para tal, simulamos a chegada constante dos clientes e seu respectivo atendimento, considerando para cada qual diferentes tempos gastos de realização das atividades.

Os detalhes técnicos de utilização das threads e dos semáforos são explicados no tópico “Descrição do Sistema” e a forma de executar o programa está explicada no tópico “Como executar o sistema e requisitos”.

2. Descrição do sistema

Como mencionado anteriormente, o Sistema do Posto de Combustível é uma simulação escrita em linguagem Python que foi estruturada com base no modelo Produtor/Consumidor. Além disso, para cumprir com os requisitos mínimos, o sistema faz uso de threads para possibilitar não apenas o funcionamento de várias bombas ao mesmo tempo, mas também para permitir a chegada de novos clientes e também para realizar a impressão do sistema do posto no terminal em tempo real.

É importante notar que existe uma instabilidade intencional no programa, que provoca uma diferença aleatória de tempo entre as chegadas de novos clientes e a finalização do enchimento do tanque. Nesse contexto, é possível não haver nenhum cliente em espera, nenhuma bomba livre ou então haver tanto bombas livres como clientes em espera. Porém, como o sistema é dinâmico, os cenários irão ser alternados ativamente em um curto espaço de tempo.

Para realizar essa simulação, foram escritas 3 (três) funções específicas:

- *Função Produtora*: a primeira delas consiste basicamente em um loop infinito que faz a contabilização de quantos clientes estão na fila para

atendimento através de uma variável global (clientes). Nessa função, cada cliente chega em uma média de 1 segundo, com desvio padrão de 2 segundos, sendo certo que, dentro desse loop infinito, estão localizadas as funções que regulamentam os semáforos e as travas de entrada na região crítica. Abaixo, temos o código da função:

```
while True:
    # Caso haja espaço para espera de novos clientes,
    # e existam bombas que possam ser usados,
    # entrar no espaço crítico de entrada de novos clientes.
    clientes_esperando.acquire()
    bombas_disponiveis.acquire()

    time.sleep(abs(norm(1, 2))) # Média de 1 segundo, com desvio padrão de 2 segundos
    clientes += 1

    # Caso um novo cliente entre em espera,
    # contabilizá-lo e enviá-lo a uma bomba quando possível.
    bombas_disponiveis.release()
    clientes_abastecendo.release()
```

Imagem 1. Código-Loop do Produtor

- *Função Consumidora:* a segunda delas consiste também em um loop infinito assistido por três variáveis globais: a) bombas; b) clientes; c) clientes_finalizados. Vale acrescentar que, dentro desse loop infinito, temos a atuação dos semáforos e das travas, controlando a quantidade de clientes que são atendidos pelas bombas de combustíveis. Após a utilização da região crítica (abastecimento), os semáforos são atualizados, bem como as travas são liberadas, deixando, dessa forma, as variáveis atualizadas para serem impressas na tela de exibição. Abaixo, temos o código da função:

```
while True:
    # Caso haja bombas disponíveis, entrar na zona crítica (de abastecimento)
    # O tempo de abastecimento médio é em torno de 4s
    clientes_abastecendo.acquire()
    bombas_disponiveis.acquire()

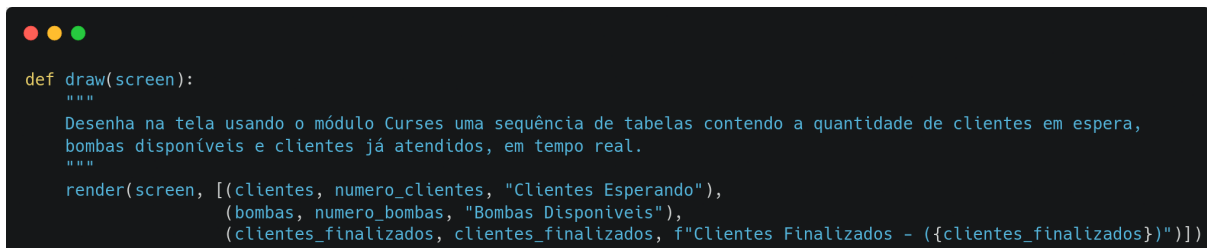
    bombas -= 1
    clientes -= 1
    time.sleep(abs(norm(4, 1))) # Média de 4 segundos, com desvio padrão de 1 segundo

    # Após a realização do abastecimento, liberar a bomba e contabilizar o cliente finalizado
    clientes_finalizados += 1
    bombas += 1

    bombas_disponiveis.release()
    clientes_esperando.release()
```

Imagem 2. Código-Loop do Consumidor

- *Função Draw*: a terceira função é responsável pela apresentação de um relatório, em tempo real, que permite a visualização da quantidade de clientes em espera, da quantidade de bombas disponíveis e da quantidade de clientes atendidos. Isso somente é possível porque as duas primeiras funções são chamadas por instâncias diversas que usam threads e permitem a execução em paralelo com impressão.



```
def draw(screen):  
    """  
    Desenha na tela usando o módulo Curses uma sequência de tabelas contendo a quantidade de clientes em espera,  
    bombas disponíveis e clientes já atendidos, em tempo real.  
    """  
    render(screen, [(clientes, numero_clientes, "Clientes Esperando"),  
                    (bombas, numero_bombas, "Bombas Disponiveis"),  
                    (clientes_finalizados, clientes_finalizados, f"Clientes Finalizados - ({clientes_finalizados})")])
```

Imagem 3: Função “draw”, que imprime n caracteres “*”, iguais ao primeiro número.

3. Como executar o sistema e requisitos

Para executar a simulação do sistema e visualizar seus efeitos, é necessário primeiro baixar o repositório “SO-Threads” no seguinte endereço eletrônico: <https://github.com/ethoshomo/SO-Threads.git>

Nesse repositório, existem apenas dois arquivos de interesse, disponibilizados na pasta “Código”: “main.py” e “draw.py”. O primeiro deles contém as principais funcionalidades do sistema, como as funções que controlam os threads e os loops de simulação, enquanto que segundo imprime o relatório no terminal.

Por isso, não há requisitos complexos a serem considerados, pois é necessário apenas utilizar um ambiente Python 3. No entanto, é conveniente destacar que, caso não se esteja usando um sistema Unix, será necessário instalar a biblioteca “[windows-curses](#)” para visualizar o funcionamento; e, caso todos os requisitos anteriores tenham sido cumpridos, é necessário utilizar um terminal com pelo menos 24 linhas de caracteres disponíveis.

Enfim, para finalmente executar o programa, basta acessar um terminal e executar o arquivo “main.py” por meio do seguinte comando: `$ python main.py`

4. Conclusão

Por fim, é importante salientar que conseguimos assimilar os conteúdos apresentados e discutidos em sala de aula por meio do trabalho prático que exigia a escrita de um programa que utiliza semáforos e threads.

Como o programa foi implementado baseado em uma simulação, bastou utilizar apenas uma interface simples no Terminal para satisfazer nossas necessidades. Mas, é importante destacar que o uso das threads deixou a interface dinâmica e permitiu uma apresentação mais fluida que uma mais inocente.

Por outro lado, percebemos que é possível construir lógicas mais complexas utilizando os conceitos da disciplina e do projeto. A entrada e a saída das zonas críticas, por exemplo, podem ser modularizadas e usadas como encapsulamento de outras funções. Da mesma forma, essas técnicas também podem ser adaptadas para outras tarefas, como, por exemplo, na visualização computacional.

Assim, por todas essas razões, concluímos que o trabalho foi um sucesso e nos capacitou para trabalhar com instrumentos mais elaborados.