**Overview of the Cigarette Smokers Problem**

The Cigarette Smokers problem is a classic synchronization challenge in computer science, demonstrating issues related to process coordination and resource allocation in a concurrent system.

Origin

The Cigarette Smokers problem was first presented by Suhas Patil in a technical report for the Massachusetts Institute of Technology in 1971 [[1]](#first). This problem emerged during a time when the study of synchronization in computing environments was becoming critical due to the advent of multiprocessor systems and complex operating systems.

The problem is a variant of the "Producers and Consumers" problem, which is itself a fundamental challenge in computer science. It was crafted to reflect real-world scenarios where deadlock prevention is crucial, illustrating a simplified but direct analogy to resource allocation.

Purpose

This problem is a staple in many operating systems and concurrency textbooks [[2]](#second), used to teach students about the complexities of process synchronization. It presents a clear scenario where not just the allocation, but also the correct sequence of resource release, is crucial.

Beyond its role in demonstrating synchronization issues, the problem serves as an educational tool, helping students and professionals understand the nuances of resource allocation and the importance of careful design in synchronization mechanisms to prevent deadlocks.

Context

The Cigarette Smokers Problem is a synchronization problem in computer science that demonstrates the complexities of allocating limited resources among processes without causing deadlock. The task is to ensure that each smoker gets a chance to roll a cigarette without indefinitely blocking others.

Even though it was conceptualized over five decades ago, the principles illustrated by the Cigarette Smokers problem are relevant in today’s multicore and distributed computing environments. The analogy helps explain why modern operating systems need sophisticated mechanisms like priority inheritance and advanced semaphore techniques.

The problem typically uses semaphores as a synchronization tool, but it can also be extended to demonstrate the use of monitors, mutexes, and condition variables and even in the context of discussion limitations of Semaphores [[1]](#first) use another premitive. These tools are fundamental in building robust multi-threaded applications that are free from race conditions, deadlocks, and starvation.

**Problem Definition**

* **Participants**: Three smokers and an agent.
* **Resources**: Three ingredients to make a cigarette – tobacco, paper, and matches.
* **Process**: Each smoker perpetually attempts to make and smoke cigarettes. However, each smoker possesses only one of the three required ingredients and needs the other two to proceed.

The agent represents a system or mechanism that distributes resources. The agent randomly selects and places two out of the three ingredients on the table. The smoker who has the third ingredient then makes a cigarette, smokes it, and signals the agent to distribute another set of two ingredients.

Application in Real Operating Systems

To explain the premise, the agent in the Cigarette Smokers Problem symbolizes an operating system that allocates resources, while the smokers represent applications requiring resources. The key is to ensure that if resources are available, the applications that need them are allowed to proceed.

Agent as Operating System

The agent represents the operating system's role in managing resources to avoid starvation and minimize deadlock risks, ensuring efficient use and distribution to meet the needs of various applications.

Resource Allocation Strategy

The operating system employs algorithms like the Banker’s Algorithm to prevent deadlocks and techniques like paging and segmentation for efficient memory management.

Smokers as Applications

In this analogy, smokers are akin to applications that request specific resources (CPU cycles, memory, I/O operations). The operating system evaluates these requests to optimize performance and user experience.

Process Scheduling

Process scheduling in an operating system is comparable to deciding which smoker gets the next set of resources. Scheduling algorithms determine which application runs when, ensuring smooth operation of critical applications.

Concurrency Management

Managing concurrency is crucial to prevent race conditions. The operating system uses locks, semaphores, and monitors to ensure that applications do not interfere with each other.

Security and Resource Protection

Similar to the agent ensuring correct resource distribution, the operating system protects resources from unauthorized access through measures like user authentication and process isolation.

Monitoring and Performance Tuning

The operating system continually monitors resource usage to optimize performance, similar to how an agent might adjust resource distribution based on demand.

Conclusion

The Cigarette Smokers Problem helps illustrate the complex resource management dynamics within operating systems. This analogy highlights the importance of scheduling, security, and resource allocation in maintaining system stability and efficiency. By using advanced resource management strategies, operating systems ensure optimal performance of all applications without delays or conflicts.

**Different Versions of The Problem**

**1. The Impossible Version**

**Description**

In this version, as defined by Patil, there are two significant restrictions:

1. The agent's code, which supplies ingredients, cannot be modified and should depend on abstaractions, not implementations. This reflects situations in real systems where the operating system's core functionalities are not frequently changed.
2. The use of conditional statements or arrays of semaphores is prohibited.

**Analysis Based on Patil's and Parnas's Papers**

* **Patil's Viewpoint**: According to Patil, these constraints make the problem unsolvable with Dijkstra Semaphores because they overly simplify the synchronization tools available, thereby not allowing flexible response mechanisms to the dynamic allocation of resources.
* **Parnas's Critique**: Parnas argues that the prohibition against using conditional statements and semaphore arrays imposes artificial constraints that do not typically apply in real-world systems. He suggests that such limitations artificially inflate the complexity and unsolvability of problems which could otherwise be managed with more advanced synchronization mechanisms.

**References and Applications**

* Patil, Suhas S., "Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination Among Processes" [1] emphasizes the restrictive nature of these conditions.
* Parnas, D. L., "On a Solution to the Cigarette Smokers' Problem (without conditional statements)" [3] criticizes these artificial restrictions and offers alternative solutions using semaphore arrays.

**2. The Interesting Version**

**Description**

This scenario retains the restriction of non-modifiability of the agent's code but removes the other limitations, allowing the use of conditional statements and semaphore arrays.

**Analysis Based on Patil's and Parnas's Papers**

* **Patil's Impact**: Removing the restrictions on conditionals and semaphore arrays challenges Patil's original claim about the insufficiency of basic semaphore primitives, opening the door to more sophisticated and realistic synchronization strategies.
* **Parnas's Contributions**: By allowing more complex coordination mechanisms, this version aligns more with Parnas's view, demonstrating that the problem can be solved with enhanced semaphore functionalities and more detailed process coordination.

**References and Applications**

* Parnas's paper [3] provides a practical ideas that uses these more flexible tools, showing that with the right synchronization primitives, the system can efficiently manage resource allocation without deadlocks.

**3. The Trivial Version**

**Description**

In this variant, the agent not only places ingredients but also signals which smoker should proceed next, based on the available ingredients. This direct intervention simplifies the synchronization to a great extent, making the problem trivial and the synchronization mechanism almost redundant.

**Analysis Based on Patil's and Parnas's Papers**

* **Patil's Perspective**: While Patil does not specifically address this version, his analysis implies that such oversimplification removes the educational and practical value of the problem as a tool for understanding complex synchronization.
* **Parnas's Views**: Parnas would likely criticize this version for not realistically representing the challenges of process synchronization in operating systems, as it bypasses the complexity typical of real-world resource allocation.

**References and Applications**

* This version is less discussed in academic literature due to its limited application in teaching or understanding advanced synchronization concepts.

**Analysis**

Lets again dive into definition of the problem:

The CSP involves four agents (or processes); one of the agents is the supplier and the rest three are smokers:

1. Supplier: supplier possesses infinite amount of the three basic ingredients for making cigarettes: tobacco, wrapping paper, and match
2. Smoker with tobacco (TB): this agent possesses infinite amount of tobacco and expects the other two ingredients (wrapping paper and match) from the supplier,
3. Smoker with wrapping paper (WP): this agent possesses infinite amount of wrapping paper and expects the other two ingredients (tobacco and match) from the supplier, and
4. Smoker with match (MA): this agent possesses infinite amount of matches and expects the other two ingredients (tobacco and wrapping paper) from the supplier.

Let us imagine that the supplier and the three smokers are sitting around a table. The procedure consists of the following steps:

1. Though the supplier has an infinite supply of all the three ingredients for making cigarettes, the supplier arbitrarily selects only two of the ingredients and places them on the table.
2. The smoker who has the remaining ingredient can take the two ingredients, and then make and smoke a cigarette. Upon completion of smoking, the supplier will be signaled.
3. For the next cigarette, the supplier again arbitrarily selects two of the three ingredients and places them on the table. This cycle is repeated forever.

1. Impossible Version

Constraints and Key Points:

* The agent's code cannot be modified and should depend on abstractions, not implementations.
* Use of conditional statements or arrays of semaphores is prohibited.
* Presented by Patil as an unsolvable problem with Dijkstra's semaphore primitives alone.

Analysis:

* The restrictions render basic semaphore primitives insufficient for dynamic resource allocation, thus making the problem unsolvable within these bounds.
* These restrictions are artificially limiting and not reflective of real-world scenarios, where more complex coordination mechanisms are allowed and effective.
* In discussing solutions for this version, the suitability of using Petri nets as a modeling tool is highlighted. Patil suggested Petri nets could offer unique advantages in modeling synchronization challenges where conventional semaphore systems might fall short. They are particularly effective in capturing complex dependencies and interactions among concurrent processes, making them ideally suited for tackling problems like the Impossible Version where intricate synchronization and deadlock avoidance are critical.

2. Interesting Version

Constraints and Key Points:

* The agent's code remains unmodifiable.
* The prohibition on using conditional statements is lifted, as is the restriction on semaphore arrays.

Analysis:

* Without the constraints, Patil's original argument about the limitations of semaphore primitives is challenged, indicating that more sophisticated synchronization strategies could be valid.
* This version is more aligned with real-world applications, demonstrating solvability with enhanced semaphore functionalities and advanced process coordination.
* For this version, semaphore systems provide a straightforward and naturally easier approach. These systems are more intuitive for modeling synchronization and coordination without the complex graphical representations needed in Petri nets, making them preferable in scenarios that allow for some flexibility in synchronization mechanisms.

3. Trivial Version

Constraints and Key Points:

* The agent not only places ingredients but also signals which smoker should proceed, greatly simplifying the coordination required.

Analysis:

* This setup likely oversimplifies the synchronization challenge to the point that it loses its educational and practical value as a complex problem.
* Likely criticizes this version for its unrealistic simplicity and for not presenting genuine synchronization challenges typical in operating systems.
* The trivial nature of this version reduces the necessity for complex synchronization tools like Petri nets. Semaphore systems, with their direct approach to resource allocation and process scheduling, are ideally suited here. They offer an effective solution for simplifying synchronization demands, especially in scenarios that trivialize the coordination among processes.

With a clear understanding of these tools and their appropriate contexts, we are now well-prepared to proceed to the actual implementation phase of the solutions.

# Implementation with Explanation

Context:

* Programming Language: Python with SYNC framework from Littele Book Of Semaphores[[2]](#second), MATLAB (GPenSIM)

Impossible Version:

The Petri Net with Inhibitor Arcs offers a sophisticated method to model and manage the synchronization and resource allocation challenges in the Cigarette Smokers Problem (CSP), particularly under the constraints of the Impossible Version.

**What are Petri Nets?**

Petri nets are a mathematical modeling tool used in computational fields to describe and analyze the distribution of discrete resources among competing processes.

A Petri net is composed of places, transitions, and arcs:

* **Places** can hold tokens, representing resources or conditions.
* **Transitions** connect places and represent events that can occur, transferring tokens between places based on defined rules.
* **Arcs** connect places to transitions (and vice versa), guiding the flow of tokens and thus controlling the system's state transitions.

**Inhibitor Arcs in Petri Nets**

Inhibitor arcs are a special type of arc used in Petri nets to enhance control over process synchronization by preventing a transition from firing unless a specific condition is met. Unlike regular arcs, which only dictate the movement of tokens if certain resource counts are reached, inhibitor arcs can prevent a transition from firing if tokens are present in a linked place.

**Implementation Details:**

* **Model Structure**: In the Petri Net with Inhibitor Arcs model, the supplier (agent) is represented by transitions and places which control the flow of ingredients based on the availability signaled by inhibitor arcs. These arcs prevent a transition (smoker process) from firing unless certain conditions are met, effectively managing resource allocation without direct conditional logic.
* **Operation**: The supplier selects two out of the three ingredients at random and places them on the table (represented by the transition firing and placing tokens in respective places). The smoker who has the third ingredient, and thus can complete the cigarette, is allowed to proceed only if the inhibitor conditions (lack of his own ingredient on the table) are satisfied.
* **Inhibitor Arcs Functionality**: These arcs ensure that a smoker cannot proceed to take ingredients if they already possess them. For example, if the ingredients on the table are tobacco and paper, the inhibitor arcs prevent the smoker who owns either tobacco or paper from proceeding, thus allowing only the smoker with matches to proceed.

A diagram of a smoker

Description automatically generated

Figure-2 shows the Petri Net model for the CSP that uses the inhibitor arc extension. On the supplier side, the inhibitor arcs make sure that the supplier selects two different ingredients, arbitrarily. The rest of the net make sure that the smoker without the two ingredients takes these two and start smoking. [[5]](#five)

**Application Flow in the CSP with Inhibitor Arcs**

In the CSP modeled with Petri nets using inhibitor arcs, the flow and control mechanisms work as follows:

1. **Resource Distribution by the Supplier**:
   * The supplier (agent) selects two out of the three necessary cigarette-making ingredients (tobacco, paper, matches) and places them on a shared table (represented as places in the Petri net).
   * Each ingredient is modeled as a place within the Petri net. The placement of tokens in these places indicates the availability of the respective ingredients on the table and linked to all transitions of smokersm, except smoker who already produce thois material.
2. **Smoker Activation**:
   * Each of the three smokers requires two specific ingredients that they do not own to begin making a cigarette. For instance, the smoker who owns tobacco needs paper and matches.
   * Transitions in the Petri net represent the action of a smoker beginning to make a cigarette. These transitions are connected by arcs to the places representing the ingredients they need.
   * So make a cigarette is only possible when both of the required places are fired, in exmplae of smoker who owns tobacco needs paper and matches (pPW and pMA) to be fired, to roll a cigarette.
3. **Inhibitor Arc Functionality**:
   * Inhibitor arcs are used to prevent a smoker from attempting to make a cigarette if they already have access to any ingredient on the table that matches their own. This is crucial to prevent deadlock scenarios where a smoker might hoard resources without being able to proceed.
   * For example, if the table has tobacco and paper, inhibitor arcs prevent the smoker who owns tobacco or paper from starting their transition(making a cigarette), becouse they are not linked within a Net, thus only allowing the smoker who has matches to proceed.
4. **Synchronization and Deadlock Prevention**:
   * The correct functioning of transitions ensures that only the eligible smoker can proceed at any time, maintaining the flow of resource usage without deadlock.
   * This model effectively uses the inhibitor arcs to manage synchronization by enforcing the rules dictated by the CSP constraints: no direct modification of the agent's code and no use of conditional statements or semaphore arrays.

**Conclusion**

The implementation of the CSP using Petri nets with inhibitor arcs meets the challenges imposed by the Impossible Version's constraints. This approach not only adheres to the non-modifiability of the agent's code and avoids explicit conditional statements but also effectively manages the synchronization of resource allocation among smokers. The use of Petri nets allows for clear visualization and analysis of the process flows, ensuring that all system states can be reached without deadlocks, thus making the system robust and reliable in scenarios requiring strict synchronization.

# Interesting Version

# Tools and References

[1] Limitations and Capabilities

[2] Little book of semaphores

[3] ON A SOLUTION TO THE CIGARETTE SMOKERS' PROBLEM

(without conditional statements)

[4] <https://www.geeksforgeeks.org/bankers-algorithm-in-operating-system-2/>

[5] **Revisiting Petri Net modeling of the Cigarette Smokers’ Problem: A GPenSIM Approach**

Reggie Davidrajuh

*Electrical & Computer Engineering*

University of Stavanger Stavanger, Norway