Project Report

Chain Smoker Synchronization Problem in C++ (Console Version)

Author: Muhammad Ashir
Domain: Operating Systems
Platform: Linux
Language: C++

1. Introduction

This project is a console-based implementation of the classical *Chain Smoker* synchronization problem using C++. The purpose of this simulation is to demonstrate multi-threading and inter-thread communication using semaphores, which are crucial concepts in Operating Systems. It elegantly models the complexities of resource synchronization, mutual exclusion, and the potential for deadlock in concurrent systems, making it a valuable case study. The scenario involves one agent and three smokers. Each smoker has an infinite supply of one of the three ingredients (tobacco, paper, or matches). The agent places two random ingredients on the table, and the smoker who has the third ingredient picks them up to make and smoke a cigarette.

This project was designed to run on **Linux**, leveraging **POSIX threads** (pthread) and **POSIX semaphores**. However, it may also be compiled and executed on other POSIX-compliant operating systems such as macOS and BSD, provided the necessary libraries are supported and installed.

2. Objectives

- To simulate the Chain Smoker synchronization problem using threads and semaphores.
- To reinforce concepts of mutual exclusion, deadlock avoidance, and thread synchronization.
- To develop a system-level program that models real-world concurrency problems in operating systems.

3. Language and Platform Details

- **Language Used:** C++
- **Compiler Used:** g++
- **Primary Target Platform:** Linux (tested with Ubuntu)
- **Other Compatible Platforms:** Any OS with POSIX-compliant threading and semaphore libraries
- **Dependencies:**

- pthread (-lpthread)
- semaphore (<semaphore.h>, usually included in glibc)

To compile:

```
g++ smokers.cpp -o smokers -lpthread
```

4. Standard Libraries and Their Purpose

Library	Purpose
<stdio.h>	Standard input/output for C functions like printf (though replaced by C++ streams here).
<stdlib.h>	Provides utility functions including rand() and srand().
<pthread.h>	Enables multi-threading using POSIX threads (pthread_create, pthread_join).
<semaphore.h>	Provides semaphores for synchronization (sem_init, sem_wait, sem_post).
<unistd.h>	Contains usleep() and sleep() for introducing delays.
<array>	Included but not used. Can be omitted.
<iostream>	Enables standard C++ input/output streams (cout, endl).

5. Program Components and Flow

5.1. Global Variables and Semaphores

- sem_t semaphoreSmokers[3]: One semaphore for each smoker, signaled by the agent when their required ingredients are available.
- sem_t semaphoreAgents: Ensures only one agent operation happens at a time.
- sem_t agentReady, sem_t smokerReady: Additional semaphores to control flow coordination.
- int semWait[3]: Tracks how many cigarettes each smoker must make (initialized to 7 per smoker).
- int glob: Stores the index of the smoker who should act next.
- bool allSmoked: Determines when all smoking is completed.
- string agentsDraw[]: List of the ingredient pairs thrown by the agent.

5.2. Threads

- **Smoker Threads (3 Total)**
  Each smoker has a separate thread. Every smoker has an infinite supply of one unique ingredient (e.g., Smoker 1 has tobacco). They wait for the other two ingredients from the agent and then proceed to "make and smoke" a cigarette.
    - Function: `void *smoker(void *pVoid)`
    - Key Logic:
        - Waits for signal from the agent using its respective semaphore.
        - Checks if it is their turn (based on `glob`).
        - Simulates making and smoking a cigarette with delays.
        - Signals the agent to continue.
- **Agent Thread**
  The agent randomly places two items on the table, triggering the appropriate smoker's semaphore if that smoker has the missing third item.
    - Function: `void *agent(void *pVoid)`
    - Key Logic:
        - Waits for the `agentReady` signal.
        - Randomly selects a pair of ingredients and posts the semaphore for the matching smoker.
        - Continues until all smokers finish smoking 7 times.
- **Pusher Thread**
  This auxiliary thread synchronizes the start of the agent once at least one smoker signals readiness.
    - Function: `void *func(void *pVoid)`

6. Smoking Logic

Each smoker performs the following steps in each round:

1. Waits for ingredients.
2. Checks whether it's their turn (`glob == their index`).
3. "Makes" a cigarette using a delay (`usleep`, `sleep`).
4. Prints colorful logs to the console for visibility.
5. Signals the agent to produce new ingredients.

The core synchronization loop operates as follows: A smoker thread begins its cycle by calling `sem_wait()` on its personal semaphore, which immediately blocks the thread until the agent signals it. The agent, after randomly choosing two ingredients, calls `sem_post()` on the corresponding smoker's semaphore. This awakens the smoker, who then proceeds to make and smoke a cigarette, simulated by `sleep()` calls. Once finished, the smoker calls `sem_post()` on the agent's semaphore. This final signal completes the cycle, waking up the agent and permitting it to distribute the next set of ingredients.

Example Output Snippet:

```
Smoker 1 waiting for paper & matches
==> Smoker1 is making a cigarette
Now Smoking
```

Color codes like \033[0;31m are used to add console color formatting for better visualization during runtime.

7. Number of Rounds

- Each smoker smokes **7 times** (configured via semWait[i]=7), making a total of **21 iterations** across all threads.
- The agent continues to supply items until all smoking is completed.

8. Observability and Debugging

- Random delays (usleep(rand() % 50000)) are used to simulate asynchronous behavior.
- Colored console messages clearly show the sequence and interleaving of thread operations.
- Smoking completion can be observed directly via the logs.

9. Portability

While this project is primarily developed and tested for Linux, it may also be executed on any OS that supports POSIX threading and semaphores. These include:

- macOS (with Xcode and Clang)
- BSD variants
- WSL (Windows Subsystem for Linux)

For native Windows environments, however, this code will not compile due to a lack of POSIX support. Porting it to Windows would require rewriting synchronization using Windows API threads and semaphores.

10. Limitations & Potential Improvements

- **Hardcoded Iterations:** Smoking count is fixed at 7 per smoker; this can be made dynamic.
- **No Exit Join on Agent Thread:** pthread_join for the agent and pusher threads is missing; the program exits once all smoker threads complete.

- **Minor Memory Leak:** new int[1] is used without delete; while not critical here, proper memory management is recommended.
- **Redundant Headers:** <array> is included but unused.
- **Function Modularity:** The main thread creation logic can be wrapped in a dedicated initializer for clarity.

11. Conclusion

This project successfully demonstrates multi-threaded synchronization using semaphores in C++, modeling a real-world concurrency problem from Operating Systems. It is educational in nature and helps understand race conditions, signaling, mutual exclusion, and how threads coordinate actions in shared memory space. By simulating the smokers and agent problem with randomized behavior and clear console output, this implementation reinforces key OS principles in an interactive and engaging way. Ultimately, this implementation serves as a practical testament to the fact that complex resource-sharing problems can be solved elegantly, but require carefully designed signaling mechanisms to prevent common concurrency pitfalls like deadlock and race conditions.