**National University of Computer &**

**Emerging Sciences Karachi Campus**



**Chain Smoker Problem System Call**

**Project Report**

**Operating System**

**Section: BSE-4B**

**Group Members:**

K21-3829 Hassan Bin Haneef

K21-3916 Muhammad Raza

K21-3883 Abdul Haseeb

1. **Introduction:**

This project is dedicated to creating a system call that deals with the chain smoker problem. A system call is a request for a service that is made by the application programs to the operating system; these can be either user system call (without kernel intervention) or kernel system call (with kernel intervention).

1. **Project Specification:**

The main function deals with the creation, and deletion of threads and semaphores. The problem at hand involves four processes, namely three smoker processes and one agent process. Each smoker process is responsible for creating and smoking a cigarette, and requires a combination of three items: tobacco, paper, and matches. One of these items is present in each smoking procedure, with each smoker using either tobacco, paper, or matches. The agent process has an infinite supply of all three items and is responsible for placing two of these items on the table. The smoker with all three items will then proceed to light the cigarette.

**Tools and Technologies:**

**Programming Language:** C language

**Virtual Machine:** VMware Workstation 17

**Platform:** Ubuntu 16.04

1. **Problem Analysis:**

**Problem:** How to manage the creation and smoking of cigarettes by three chain smokers, each requiring a specific combination of tobacco, paper, and matches, so that no race condition occurs?

**Input:** Semaphore addresses.

**Process:** To ensure that only one smoker can smoke a cigarette at a time, semaphore wait, and post calls can be used. Similarly, to ensure that the agent process places only two items on the table at a time, semaphores can be used to prevent race conditions.

1. **Solution Design:**

Agent function uses a semaphore post-call to decrease the value of the resource that the agent puts on the table. If a smoker needs more of the acquired resource to light a cigarette, the agent function will use a semaphore wait-call.

The chain smokers with the resource use their respective functions that have semaphore wait-call indicating that they are acquiring the other two resources, if one of the resources is held by the other chain smoker then it uses semaphore post-call to increment the value of the resource indicating that it is releasing the grabbed resource.

For example: If the smoker with tobacco requires matches and paper, he will use the “smoker with tobacco” function to have the other two resources.

If the required resource is taken by another smoker, the semaphore post-call will be used to release the picked resource.

If the smoker wants more resources to light a cigarette, then it uses a semaphore post-call that signals the agent to provide him with more.

1. **Project Breakdown:**
2. **Results:**

After concerted team efforts, we have successfully developed a solution that prevents the occurrence of deadlock in the chain smoker's problem. By implementing the system calls, we have been able to create a system that is free from race conditions and ensures the smooth operation of processes without deadlock. The solution we have developed can be seen as an example of how an operating system can effectively manage resources and avoid conflicts, enabling the seamless execution of complex tasks.

1. **Acknowledgement:**

We would like to acknowledge the following links that helped us in understanding the problem better:

1. Blog website: [Link](https://w3.cs.jmu.edu/kirkpams/OpenCSF/Books/csf/html/CigSmokers.html)
2. YouTube: [Link](https://youtu.be/ee03HrHBWLA)

The following GitHub link redirects to our project code: