**OPERATING SYSTEM PROJECT**

**SINGLE LANE BRIDGE PROBLEM**

**Problem Statement:**

A single-lane bridge connects the two Vermont villages of North Tunbridge and South Tunbridge. Farmers in the two villages use this bridge to deliver their produce to the neighbouring town. The bridge can become deadlocked if a northbound and a southbound farmer get on the bridge at the same time. (Vermont farmers are stubborn and are unable to back up).Using semaphores and/or mutex locks, design an algorithm in pseudocode that prevents deadlock. Implement your solution using POSIX synchronization. In particular, represent northbound and southbound farmers as separate threads. Once a farmer is on the bridge, the associated thread will sleep for a random period of time, representing traveling across the bridge. Design your program so that you can create several threads representing the northbound and southbound farmers.

## **Workflow:**

**Headers included:**

<pthread.h>

<stdio.h>

<unistd.h>

<stdlib.h>

<time.h>

**Variables:**

pthread\_mutex\_t mutex, nts, stn, mutex1;

int ntscnt, stncnt;

int ntscyc, stncyc;

int onnts, onstn;

## **Functions:**

**MAIN:**

* All variables are initialized.
* Villagers (threads) are created.

**THREAD FUNCTION:**

* Two thread functions are created indicating the direction of the travel.
* Each thread functions has four stages, those are waiting, got permission, crossing the bridge and then the last stage is crossed.
* **Stage 1: WAITING**

Lock the our bridge

* **Stage 2: GOT PERMISSION**

Lock mutex

○ Increment the cnt and cyc count

○ If (this thread is the 1st thread getting permission)

■ Then lock opposite bridge

○ If (the cyc count not equal to starvation\_count)

■ Then unlock our bridge

○ Else

■ Cyc count made zero

Unlock mutex

* **Stage 3: CROSSING**

Sleep for random time

* **Stage 4: CROSSED**

Lock mutex

○ Decrement cnt

○ If no one on bridge

■ Unlock opposite bridge

Unlock mutex

If opposite bridge is locked

○ Unlock our bridge and opposite bridge

Exit thread

**NOTE :**

● “our bridge” means the bridge side of the current thread and similarly “opposite bridge” means the same.

● “cnt” means the count of villagers from their respective side( atb\_cnt or bta\_cnt) similarly “cyc” means the same.

**DISPLAY:**

* Display the stage of each thread concurrently with alignment.

## **Solution:**

We have implemented a C program to help us understand a solution to this problem preventing deadlock and starvation.

**To compile:**

cc PACKAGE.c -lpthread

**To run:**

1. **./a.out**
   * Creates random number of villagers in North and South
   * Allows a villager to starve for count = 3, atmost
2. **./a.out st**
   * Creates random number of villagers in North and South
   * Allows a villager to starve for count = st (integer), atmost
3. **./a.out n1 n2**
   * Creates n1 (integer) & n2 (integer) number of North & South villagers respectively and randomly
   * Allows a villager to starve for count = 3, atmost
4. **./a.out n1 n2 st**
   * Creates n1 (integer) & n2 (integer) number of North & South villagers respectively and randomly
   * Allows a villager to starve for count = s t(integer), atmost