ABSTRACT

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 neighboring 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. 1 Implement your solution to 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.

To Solve

We have implemented a C program to help us understand a solution to this problem. The concepts used in the development of the project and a brief description about each of them are discussed in the following subsections.

DEADLOCK & STARVATION

Deadlock occurs when each process holds a resource and waits for another resource held by any other process. In this no process holding one resource and waiting for another gets executed.

Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time. In starvation resources are continuously utilized by high priority processes.

Problem of starvation can be resolved using Aging. Aging priority of long waiting processes is gradually increased.

In this scenario, the bridge represents the shared resource, and the villagers represent the processes or threads that need to access the resource. When one group of villagers continuously crosses the bridge, it blocks other villagers from crossing, leading to starvation. To prevent starvation, we can implement a mechanism that ensures that villagers from both villages have a fair chance to cross the bridge.

SOFTWARE & HARDWARE REQUIREMENT

C Compiler

C is a programming language that is an efficient compiler to native code, and its application. We used a software that is suitable for compiling the program.

Shell Scripting

A shell script is a computer program designed to be run by the UNIX shell, a command line interpreter. Operations performed by shell scripts include file manipulation, program execution, printing text, sets up the environment, runs the program, and does any necessary cleanup.

Visual Studio Code

Visual Studio Code is a free and open-source, source code editor developed by Microsoft for Windows, Linux and macOS.

ALGORITHM

The given code implements a simulation of villagers crossing a bridge from North to South (NTS) and from South to North (STN).

The villagers are represented by threads, and the simulation ensures proper synchronization and control of access to shared resources using mutexes.

Here's an algorithmic breakdown of the code:

- Include necessary header files
- Declare global variables
- Define display function
- Define NTS function
- Define STN function
- Main function
- Initialize random number generator
- Initialize start time, mutexes, and display initial state
- Create an array of threads and initialize counters
- Initialize an array to store village IDs
- Create threads for each village with necessary conditions
- Wait for all threads to finish
- Use exit(0) to terminate

This algorithm outlines the main structure of the given code, emphasizing the key components such as mutexes, thread creation, and simulation logic for villagers crossing the bridge. The actual implementation details of the functions (display, NTS, STN) are provided in the original code.

CODE DETAILS

Headers included

<pthread.h>

<time.h>

<iostream.h>

<unistd.h>

<iomanip.h>

Variables

pthread_mutex_t mutex, atb, bta , mutex1;

int atb_cnt, bta_cnt, atb_cyc, bta_cyc;

Functions

MAIN

- All variables are initialized
- villagers(threads) are created

THREAD FUNCTION

- Two thread functions are created indicating the direction of the travel.
- Each of the thread functions has four stages.

Stage 1: WAITING

- Lock our bridge

  Here, Our Bridge means bridge side of the current thread.

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 (ex. 4sec)

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

DISPLAY

Displays the stage of each thread concurrently with alignment.

RESULT AND DISCUSSION

To compile : Use this format,

cd Documents

cd project

chmod +x STARVE.c (to access file & get permission)

gcc -pthread STARVE.c -o STARVE

./STARVE

```
:~$ cd Documents
:~/Documents$ cd Project
:~/Documents/Project$ gcc -pthread STARVE.c -o STARVE
:~/Documents/Project$ ./STARVE
```

To run arguments :

1. No arguments : ./a.out

- Creates random number of villagers in North and South
- Allows a villager to starve for count = 3, atmost

2. One argument : ./a.out s

- Creates random number of villagers in North and South
- Allows a villager to starve for count = s (integer), atmost

3. Two arguments : ./a.out m n

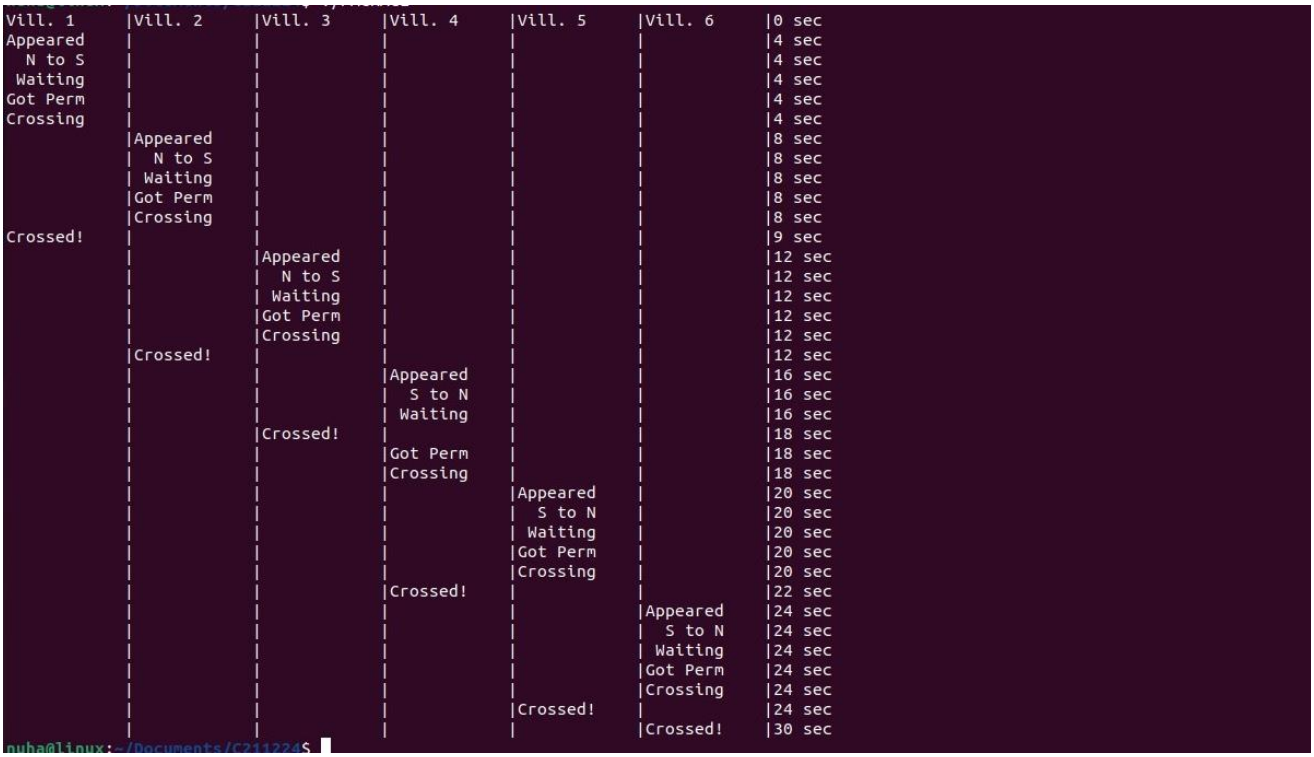
- Creates m (integer) & n (integer) number of North & South villagers respectively and randomly
- Allows a villager to starve for count = 3, atmost

4. Three arguments : ./a.out m n s

- Creates m (integer) & n (integer) number of North & South villagers respectively and randomly
- Allows a villager to starve for count = s (integer), atmost

OUTPUT

We gave the three arguments m, n & s as 3, 1 & 2 respectively and thus we get the output with 3 villagers going from A to B, 1 villager going from B to A and a villager can starve for a maximum count of 2.



This Figure is a sample output of three villagers from one side and one villager from another village. The starvation count is two which means no three villagers can move in the same direction. The corresponding timing is also printed for convenience of visualization.

CONCLUSION

In our problem, starvation occurs when villagers enter the bridge continuously while the other villagers wait for the bridge to be free. Hence, to solve this we give the other villagers a chance to cross by blocking after a fixed number of villagers cross continuously. During the course of the project, basics behind every concept were understood and this knowledge was used in helping build the system. Familiarity with various technologies, such as the Shell Terminal, was crucial for carrying out the project successfully.