İstanbul
Bilgi Üniversitesi

CMPE 312- LAB PROJECT REPORT

by

Onur Çalışkan, 119200059

Supervised by

Özgür Özdemir

Submitted to the
Faculty of Engineering and Natural Sciences
in partial fulfillment of the requirements for the

Bachelor of Science

in the
Department of Computer Engineering

2 June, 2023

Abstract

In this study, there are various researches and solutions about the Santa Claus problem, which is included in the concept of multithreading and parallel programming within the subject of operating systems

Table of Contents

1 Introduction

In this study, the Santa Claus problem was examined according to the last digit of the student number.

Santa Claus Problem:

The Santa Claus Problem is a commonly used scenario in concurrent programming. This scenario models the working process of Santa Claus, involving interactions between Santa Claus, elves, and reindeer. According to the scenario, Santa Claus lives at the North Pole and works to prepare toys during the Christmas season. Santa Claus is accompanied by elves who assist him, and reindeer who pull his sleigh.

The elves occasionally request help from Santa Claus. When an elf seeks assistance, Santa Claus helps them. However, the elves can only wait for assistance if there is a specific number of them that need to be helped by Santa Claus at the same time.

On the other hand, the reindeer are also crucial. The reindeer are trained to pull Santa's sleigh. However, they need to return from vacation and be ready before they can pull the sleigh. Once all the reindeer have returned, Santa Claus harnesses them to the sleigh and is ready to begin the delivery. The Santa Claus Problem requires proper synchronization and meeting the conditions of the scenario. Semaphores, mutexes, and other synchronization mechanisms are used to manage the interactions and waiting periods between the elves and reindeer effectively.

The problem requires proper handling of concurrent access to shared resources and ensuring that the different entities (Santa Claus, elves, and reindeer) coordinate their actions correctly. This involves using synchronization mechanisms such as semaphores and mutexes to control access to shared data, manage critical sections, and enable proper coordination between the threads or processes.

In operating systems, synchronization and inter-process communication are fundamental concepts that deal with coordinating the activities of concurrent processes or threads, preventing race conditions, and ensuring orderly access to shared resources. The Santa Claus Problem serves as an illustrative example within this domain, demonstrating how synchronization mechanisms can be utilized to achieve the desired coordination.

2 Methodology

Below is the pseudo-code design, which is the first step to solve the problem.

```
Define constants:
    NUM_REINDEER = 9
    MAX_ELVES_WAITING = 3

Define semaphores:
    santaSem, reindeerSem, elfSem
    reindeerMutex, elfMutex, elfCounterMutex

Define variables:
    numElvesWaiting = 0
    numElvesGettingHelp = 0
    reindeerArrived = 0

Define functions:
    prepareSleigh()
        Print "Santa Claus is preparing the sleigh."
        Sleep for 2 seconds

    getHitched(reindeerId)
        Print "Reindeer [reindeerId] is getting hitched to the
    sleigh."

    helpElves()
        Print "Santa Claus is helping the elves."
        Sleep for 1 second

    getHelp(elfId)
        Print "Elf [elfId] is getting help from Santa Claus."

    Santa Claus thread:
        While true:
            Wait on santaSem

            Wait on reindeerMutex
            If reindeerArrived equals NUM_REINDEER:
                Call prepareSleigh()
                For each reindeer:
                    Signal reindeerSem
                Set reindeerArrived to 0
            Release reindeerMutex

```

```
41              Wait on elfMutex
42              If numElvesWaiting is greater than or equal to
    MAX_ELVES_WAITING:
43                  Call helpElves()
44                  For each MAX_ELVES_WAITING elves:
45                      Signal elfSem
46                  Increment numElvesGettingHelp by
    MAX_ELVES_WAITING
47                  Decrement numElvesWaiting by MAX_ELVES_WAITING
48              Release elfMutex
49
50      Reindeer thread(reindeerId):
51          Sleep for a random duration between 1 and 5 seconds
52
53          While true:
54              Wait on reindeerMutex
55              Print "Reindeer [reindeerId] has returned from
    vacation."
56              Increment reindeerArrived by 1
57              If reindeerArrived equals NUM_REINDEER:
58                  Signal santaSem
59              Release reindeerMutex
60
61              Wait on reindeerSem
62              Call getHitched(reindeerId)
63              Signal reindeerSem
64
65              Sleep for a random duration between 1 and 5 seconds
66
67      Elf thread(elfId):
68          Sleep for a random duration between 1 and 5 seconds
69
70          While true:
71              Wait on elfMutex
72              Print "Elf [elfId] needs help from Santa Claus."
73              Increment numElvesWaiting by 1
74              If numElvesWaiting equals MAX_ELVES_WAITING:
75                  Signal santaSem
76              Release elfMutex
77
78              Wait on elfSem
79              Call getHelp(elfId)
80
81              Wait on elfCounterMutex
82              Increment numElvesGettingHelp by 1
83              If numElvesGettingHelp equals MAX_ELVES_WAITING:
84                  Set numElvesGettingHelp to 0
85                  Signal elfCounterMutex
86              Else:
```

```
87                  Signal elfSem
88                  Signal elfCounterMutex
89                  Wait on elfSem
90
91              Sleep for a random duration between 1 and 5 seconds
92
93  Main program:
94      Create and initialize semaphores
95
96      Create Santa Claus thread
97
98      Create reindeer threads
99          For each reindeer:
100             Create a thread and pass the reindeerId
101
102     Create elf threads
103         For each elf:
104             Create a thread and pass the elfId
105
106     Wait for all threads to finish
107
108     Destroy semaphores
```

3 Implementation

Detailed Implementation pool of the Solution code is below as a table.

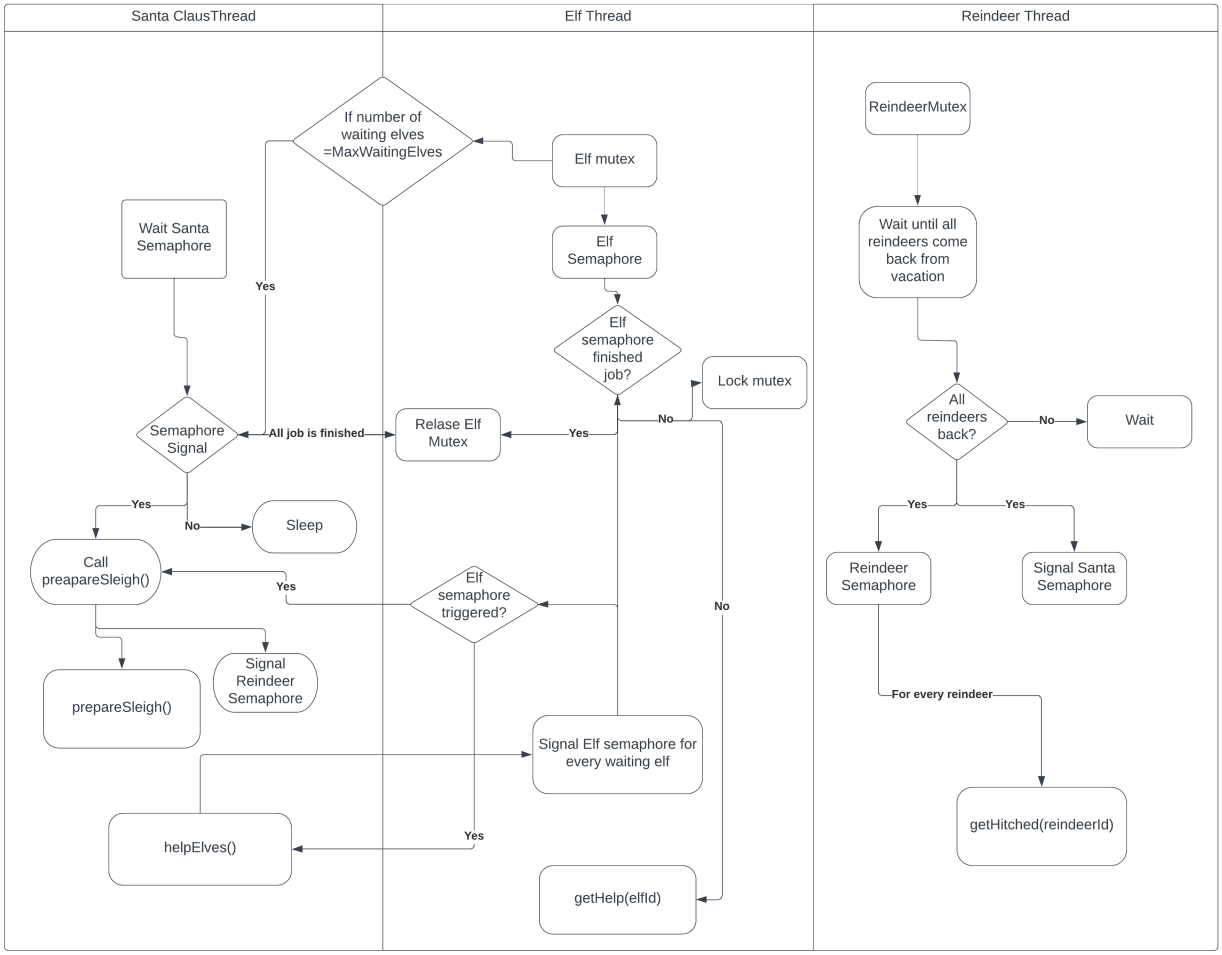

Figure 1: Function and Thread Invocation Simulation

Simulation of the problem solution:

1. Initialize semaphores and create threads for Santa Claus, reindeer, and elves.
2. The Santa Claus thread starts running.
3. The reindeer and elf threads start running and perform their respective tasks.
4. The Santa Claus thread waits for signals from the reindeer and elves.

5. Reindeer return from vacation and notify Santa Claus
6. Santa Claus prepares the sleigh and signals the reindeer to get hitched.
7. Elves request help from Santa Claus and wait until the maximum number of elves is reached.
8. Santa Claus helps the elves and signals them to continue their work.
9. The threads continue running in a loop, simulating the ongoing activities.

Explanation of the invokes of methods:

- prepareSleigh() function is invoked by the Santa Claus thread to simulate the sleigh preparation.
- getHitched() function is called by the reindeer threads when they are getting hitched to the sleigh.
- helpElves() function is invoked by the Santa Claus thread to simulate helping the elves.
- getHelp() function is called by the elf threads when they receive help from Santa Claus.
- Santa Claus thread waits on santaSem to receive signals from the reindeer and elves.
- reindeer threads wait on reindeerMutex to ensure synchronized access to the reindeerArrived variable.
- Reindeer threads signal reindeerSem to indicate that they are ready to be hitched to the sleigh.
- Elf threads wait on elfMutex to ensure synchronized access to the numElvesWaiting variable.
- Elf threads wait on elfCounterMutex to ensure synchronized access to the numElvesGettingHelp variable.
- Santa Claus thread signals santaSem to wake up and handle the next set of tasks.
- Simulation continues with the threads performing their respective actions in a loop.

4 Conclussion

The Santa Claus problem is a classic synchronization problem that involves coordinating the activities of Santa Claus, reindeer, and elves. In this problem, Santa Claus must prepare his sleigh when all the reindeer have returned from vacation or help a group of waiting elves. The goal is to ensure that Santa Claus doesn't start preparing the sleigh until all the reindeer have arrived, and he doesn't help the elves until a enough number of them are waiting.

The solution algorithm presented here provides an efficient and effective approach to solving the Santa Claus problem. It uses semaphores and mutexes to synchronize the actions of Santa Claus, reindeer, and elves, ensuring proper coordination between them. The use of semaphores allows for blocking and signaling of threads, ensuring that Santa Claus waits until the conditions are met before proceeding.

Overall, the presented algorithm and its implementation efficiently solve the Santa Claus problem by properly coordinating the actions of Santa Claus, reindeer, and elves. The use of semaphores and mutexes ensures synchronization and avoids race conditions. The algorithm demonstrates good efficiency in handling the arrival of reindeer and the waiting elves, and the implementation provides a practical demonstration of the algorithm in action.

4.1 Code Results

Terminal output after run code

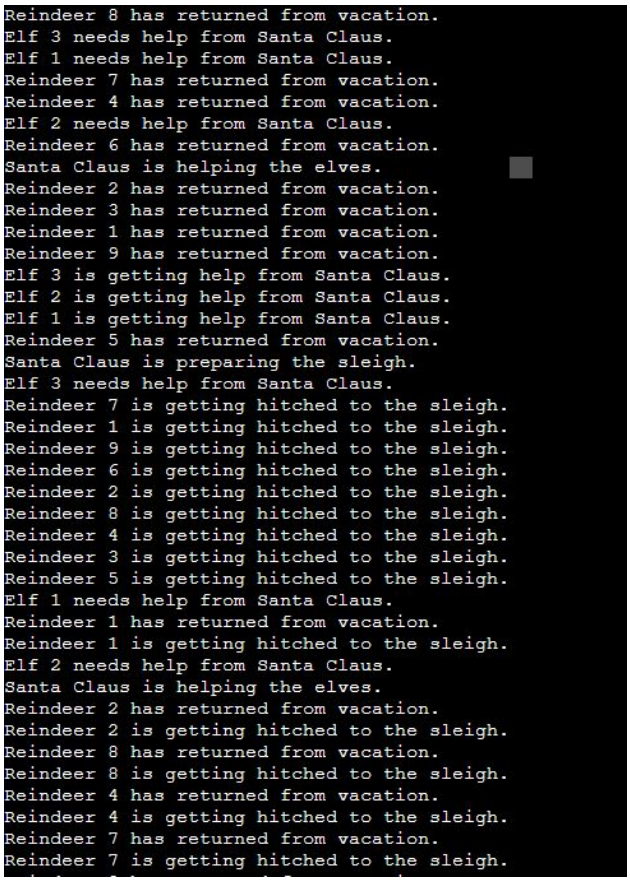

Figure 2: Function and Thread Invocation Simulation

5 References

Resources referenced in the project

https://www.researchgate.net/publication/221004442$_{S}olving_{t}he_{S}anta_{C}laus_{P}roblem_{a}_{C}ompar$

https://dl.acm.org/doi/10.1145/1132516.1132522