THE SANTA CLAUSE PROBLEM

Abdelrahman Abdin

CMPE 312: Operating Systems

May 30, 2023

İstanbul
Bilgi University

Introduction

John Trono defined the Santa Clause synchronization problem in his 1994 paper[1], I will explain it in my own words, Santa Clause likes to spend his time sleeping in his office, his reindeer (which are nine in number) enjoy their time in a tropical island vacation but return at christmas in order to help Santa deliver presents, the last one wakes up Santa once all nine of them arrive and Santa attaches them to the sleigh to deliver presents, the elves are many in number[2] and work on making toys but sometimes they have problems making toys and need Santa's help, but Santa's napping time is important so the elves are only allowed to wake Santa up when there are three of them with problems, when they wake him up, he helps those three elves and goes back to sleep.

This problem relates to multi-proccess communication, where there is more than one kind of proccess utilizing the main proccess to gain access to the resources it has, this is applicable to the access and usage of device drivers in an operating system, where multiple user and system programs access the driver with different requests concurrently.

Methodology

Analysis

Now, let us analyze this problem first and then plan our solution, the main things we have to watch out for are:

1. Only the third elf to arrive can wake up Santa, the first two must wait at the door.
2. Santa must help the three elves that arrived first and wait for any other elves to wake him up again before helping them.
3. Santa can only be woken up by elves ***or*** reindeer whoever arrives first.
4. The reindeer wait for Santa to fasten them to the sleigh before helping deliver gifts.
5. all reindeer must help santa deliver gifts before he can go back to sleep.
6. deer and Elves should not be allowed to arrive at the same moment, one of them must arrive first so we can always know which one santa should help.
7. the last deer to arrive should be the one to wake up santa.

Taking all of these into account I will implement a solution using semaphores and mutexes, and a barrier, I will be using a mutex to insure only one elf or reindeer can wake up Santa and be helped by him at a time, and I will use semaphores to keep track of the elves and reindeer, and the barrier to make sure all reindeer finish their job before santa goes back to sleep.

[1] J. A. Trono. A new exercise in concurrency. SIGCSE Bull., 26(3):8–10, 1994.
[2] not defined in the original problem other than being more than three.

Psuedocode

Setup

First we set up our costants, mutexes, semaphores, and barrier flag

```
# Constants
constant DEER = 9  # Controls the number of reindeer
constant ELVES = 20  # Controls the number of elves

# Mutexes and Semaphores
# for santa
semaphore wakeSanta  # Semaphore that Santa will watch and wake up with a signal
mutex santaLock  # Lock for access to Santa

# for elves
integer santaDoor = 0  # Keeps track of elves waiting at the door (only 3 allowed)
mutex elfQueMutex  # control the access to the santaDoor integer
semaphore helpQue  # Semaphore to keep track of elves while Santa helps each one
# Semaphore to ensure the last elf gets helped by Santa and no one jumps in line
semaphore lastElf

# for deer
integer deerInStable = 0  # Keep track of deer waiting to be attached to the sleigh
semaphore deerWaiting  # Semaphore for deer waiting to be attached
# Barrier to wait until all deer are attached before going to deliver presents
boolean deerReady = false
# Semaphore to make sure all deer are finished before Santa returns home
semaphore deerFinished
mutex deerMutex  # Mutex to control access to the deerInStable integer
```

The number of reindeer and elves can be changed to anything, but the number of elves is hardcoded to be at least 3 for them to be helped by Santa.

Elf

Here we define the gethelp function for elves and the elf thread function.

```
# Function: Elf gets help from Santa
def getHelp()
  print "Elf (current_thread) got help from Santa"
end
# Function: Elf thread
def elf()
  while true do
    workDuration = random_number(0, 99)  # Get a random number between 0 and 99
    print "Elf (current_thread) is working for workDuration"
    # Elf works for a random amount of time and then runs into a problem
    sleep(workDuration)
    # Lock Santa to ensure proper access to santaDoor and deerInStable
    lock(santaLock)
    print "Elf (current_thread) has a problem and goes to Santa for help"
    lock(elfQueMutex)
    santaDoor = santaDoor + 1
    if santaDoor == 3 then  # Only the third elf triggers this
      signal(wakeSanta)  # Wake Santa
      wait(lastElf)  # Wait until Santa finishes helping the other two elves
      wait(helpQue)  # Elf gets help from Santa
      getHelp()
      santaDoor = santaDoor - 3  # Reset santaDoor to 0
      unlock(santaLock)
      # No elves can get in queue until the last elf is done with Santa
      unlock(elfQueMutex)
    else
      unlock(santaLock)  # Release the locks since this is not the last elf
      unlock(elfQueMutex)
      wait(helpQue)  # Elf gets help from Santa
      getHelp()
    end if
  end while
end
```

Here the elf bahavior is defined by a simple if statement that checks if it is the last elf, the elf loops infinitely and at the start of the loop it works (sleeps) for a random amount of time between 0-99, until it encounters an issue and needs Santa's help, then the elf locks Santa, this is in order to stop a reindeer from changing deerInStable (represents the number of deer waiting for Santa) at the same time as an elf changes santaDoor (which represents the number of elves waiting of Santa), this lock could be delayed until we are sure that this elf will be the third elf at the door, but that would add complexity to the code without much gain (due to the small number of elves), if we had an order of magnitude more elves then this optimization would be good to make.

Then, if the elf is the last one it wakes up Santa and waits for Santa to help the other two and then help it before releasing the lock on Santa, if the elf is first or second, it just releases the lock on Santa and waits.

Reindeer

Here we define the deliver presents function for deer and the deer thread function.

```
# Function: Deliver presents
def deliverPresents()
  lock(deerMutex)
  deerInStable = deerInStable - 1
  print "Reindeer (current_thread) helped deliver presents"
  unlock(deerMutex)
  signal(deerFinished)
end

# Function: Reindeer thread
def deer()
  while true do
    vacationTime = random_number(0, 49)  # Take a vacation for 0 to 49 seconds
    print "Reindeer (current_thread) is vacationing for vacationTime"
    sleep(vacationTime)
    # Lock Santa to ensure proper access to deerInStable and santaDoor
    lock(santaLock)
    lock(deerMutex)
    deerInStable = deerInStable + 1
    if deerInStable == DEER then
      print "Reindeer (current_thread) arrived at
                     the North Pole and will wake up Santa"
      signal(wakeSanta)  # Wake up Santa
      # Unlock deerMutex so each reindeer can remove themselves from the stable
      unlock(deerMutex)
      wait(deerWaiting)  # Wait to be attached to the sleigh
      while not deerReady  # Wait here until all deer are attached
      end while
      deliverPresents()
      unlock(santaLock)  # Release Santa after all presents are delivered
    else
      print "Reindeer (current_thread) arrived at
                     the North Pole and is waiting for Santa"
      unlock(santaLock)  # Release locks since this is not the last reindeer
      unlock(deerMutex)
      wait(deerWaiting)  # Wait to be attached to the sleigh
      # Wait for all deer to be attached, then deliver presents
      while not deerReady
      end while
      deliverPresents()
    end if
  end while
end
```

Reindeer run in an infinite loop just like Santa and the elves, a deer takes a vacation (sleeps) for a random amount of time between 0-49 (these are arbitrary values), and after it flies back to the

northpole it locks Santa and goes into the stable, if it is the last it wakes up Santa, if not then it releases the locks on Santa and waits to be attached to the sleigh, after it is attached it waits at the barrier for deerReady to change which signals Santa fastening all deer to the sleigh, after that happens all deer help deliver presents, here we used the extra deerFinished semaphore to signal to Santa when **all** of the deer finished delivering presents, as Santa should only return once all reindeer successfully helped with the present delivery (this isn't guranteed otherwise as the scheduler can neglect to run one of the reindeer threads).

Note here that santaLock *can* be released before all deer deliver their presents (if the last deer is not picked last by the scheduler) but this doesn't cause an issue as Santa only returns after all deer signal deerFinished, so an elf locking santaLock after one of the deer finishes is fine as Santa will just return and be immedietly woken up (because wakeSanta would have been signaled by the elf already) and would help the elves, avoiding any deadlock.

Santa

Here we define Santa's behavior in the thread function.

```
# Function: Santa thread
def santa()
  while true do
    print "Santa is sleeping in his office"
    wait(wakeSanta)  # Wait to be awakened by the reindeer or the elves
    # Check who woke Santa up (either reindeer or elves, not both)
    if deerInStable == DEER then
      for i = 0 to DEER do
        signal(deerWaiting)  # Attach all deer to the sleigh
        print "Santa fastened a deer to the sleigh"
      end for
      print "Santa fastened all the deer and will deliver the presents"
      deerReady = true  # After attaching all deer, flip the barrier flag
      # Wait until all deer help deliver presents and then go back to the North Pole
      for i = 0 to DEER-1 do
        wait(deerFinished)
      end for
      deerReady = false  # Flip the barrier for the next loop
      print "Santa delivered all presents and returned to the North Pole"
    else if santaDoor == 3 then # check if it was elves that woke up santa
      signal(helpQue)  # If it was the elves that woke Santa, help all three elves
      print "Santa helped an elf"
      signal(helpQue)
      print "Santa helped an elf"
      # Signal to the last elf that the other two are finished, then help it
      signal(lastElf)
      signal(helpQue)
      print "Santa helped three elves and closed his door"
    end if
  end while
end
```

Santa runs in an infinite loop and waits for wakeSanta to be signaled by either the last Reindeer

or the third elf, and because of our usage of the santaLock when changing both deerInStable and santaDoor there is no way for both of our conditions here to return true so we avoid a deadlock where Santa wakes up and helps the deer while the elves are waiting for him (and are still holding santaLock), Santa's normal behavior has him handling either the elves or the deer, whoever woke him, if it was the deer he attaches them all to the sleigh and then signals that all of the deer are ready, then he waits using the loop for all 9(can be changed using the DEER constant) deer to be finished delivering gifts, after they are all finished he returns the ready flag and goes back home to sleep again (waiting on the signal to wakeSanta).

If the elves are the ones that woke Santa, Santa helps the first elf, then the second, then tells the last one to come in and helps him, this is implemented this way to insure that another elf that just arrived can't skip in line (as the last elf is holding a lock on Santa door until he finishes getting help).

Main

Initialization of mutexes and semaphores and creation of all of the threads in the main function.

```
# Main function
def main()
  initialize(santaLock)
  initialize(wakeSanta, 0)
  initialize(helpQue, 0)
  initialize(elfQueMutex)
  initialize(lastElf, 0)
  initialize(deerMutex)
  initialize(deerWaiting, 0)
  initialize(deerFinished, 0)

  # Create a thread for each reindeer
  reindeerThreads = array of threads with size DEER
  for i = 0 to DEER-1 do
    create_thread(reindeerThreads[i], deer)
  end for

  # Create the Santa thread
  create_thread(santaThread, santa)

  # Create a thread for each elf
  elfThreads = array of threads with size ELVES
  for i = 0 to ELVES-1 do
    create_thread(elfThreads[i], elf)
  end for

  # Thread join is not needed since all threads run in an infinite loop
  # and will never return, so we only have one to stop the program from returning
  join_thread(santaThread)
  return 0
end
```

The main function is simple, we just initialize our semaphores and mutexes, then create threads for each deer, elf, and for Santa, then run those threads with the functions we had defined, we have a

join statement for Santa to insure that the program doesn't terminate, none of our threads will return anyway since they all implement an infinite loop.

Implementation

In this section I'll demonstrate my implemintation in C of the psuedocode, I will change the both constants for the number of deer and number of elves to 4 to make the output easier to follow and explain (and I changed the sleep time for both to 0-19 to illustrate both), but it also works fine with 9 Reindeer and 20 Elves and the normal timing.

I will insert my commentary on what the program is doing between the lines of the output.

```
$ ./Santa_clause_problem
reindeer (92) is vacationing for 3
reindeer (88) is vacationing for 6
reindeer (84) is vacationing for 15
reindeer (80) is vacationing for 17
Santa is sleeping in his office
elf (76) is working for 13
elf (72) is working for 15
elf (68) is working for 6
elf (92) is working for 12
```

In this section the program runs the main() function, it first initializes the mutexes and semaphores (which is not seen here, it happens in the background), and then it starts the reindeer threads using the deer() function as the thread function and the first thing that happens within this function is the deer taking a random number and vacationing (sleeping) for that long, then Santa thread is initialized using the santa() function as the thread function the santa function prints Santa's state and then waits on the wakeSanta semaphore, after that we can see the elf threads get initialized using the elf() function as the thread function and they take a random number and work(sleep) for that long just like the deer.

```
reindeer (92) arrived to the north pole and is waiting for Santa
reindeer (88) arrived to the north pole and is waiting for Santa
elf (68) has a problem and goes to Santa for help
elf (92) has a problem and goes to Santa for help
elf (76) has a problem and goes to Santa for help
Santa helped an elf
Santa helped an elf
Santa helped three elves and closed his door
Santa is sleeping in his office
elf (68) got help from Santa
elf (68) is working for 9
elf (92) got help from Santa
elf (92) is working for 1
elf (76) got help from Santa
elf (76) is working for 2
```

Here we can see that two of the Reindeer (number 92 and 88) return after the random time they chose ends, but because those were the only two then they'll just wait in the stable after they see that the amount of deer in the stable is not equal to all of the deer (which is decided by the DEER

constant), after this we can see three elves (68, 92 and 76) run into problems and go to Santa for help, the first two see that they aren't three yet and so just wait on helpQue, the third elf after seeing the other two at the door wakes up Santa using the wakeSanta semaphore and then he waits on the lastElf semaphore, then Santa helps each elf (by signaling the help que for the first two elves, then signaling lastElf and helpQue again for the last elf), once each elf gets the signal on helpQue they run the getHelp function, Santa is free after signaling the elves.

```
elf (92) has a problem and goes to Santa for help
reindeer (84) arrived to the north pole and is waiting for Santa
elf (72) has a problem and goes to Santa for help
elf (76) has a problem and goes to Santa for help
Santa helped an elf
Santa helped an elf
Santa helped three elves and closed his door
Santa is sleeping in his office
elf (76) got help from Santa
elf (76) is working for 7
elf (92) got help from Santa
elf (92) is working for 10
elf (72) got help from Santa
elf (72) is working for 19
```

Here we see an elf arrive waiting on helpQue, and then a reindeer arrives and waits on deerWaiting, before two more elves arrives, the second waits on helpQue, and the third wakes up Santa and then waits on lastElf (this elf does not let go of santaLock, so no more elves or deer can arrive until Santa signals lastElf after helping the elves), then we can see Santa helps the elves and they all go back to work after getting the help.

```
reindeer (80) arrived to the north pole and will wake up Santa
Santa fastened a deer to the sleigh
Santa fastened a deer to the sleigh
Santa fastened a deer to the sleigh
Santa fastened a deer to the sleigh
Santa fastened a deer to the sleigh
Santa fastened all the deer and will deliver the presents
reindeer (92) helped deliver presents
reindeer (92) is vacationing for 3
reindeer (80) helped deliver presents
reindeer (80) is vacationing for 6
reindeer (88) helped deliver presents
reindeer (88) is vacationing for 0
reindeer (84) helped deliver presents
reindeer (84) is vacationing for 6
Santa delivered all presents and returned to the north pole
Santa is sleeping in his office
```

Here the last Reindeer arrives and sees all of the other Reindeer already waiting, so the last one goes to wake up Santa by Signaling wakeSanta (after locking santaLock), and then this last deer goes back to the stable and waits on deerWaiting, then we can see that Santa fastens all of the Reindeer to the sleigh by signaling deerWaiting for each Reindeer, after this is done Santa sets deerReady to true

which was the barrier that all of the deer were waiting on, this Signals all of the Reindeer to deliver presents by calling the deliverPresents() function, then Santa waits in a loop for every deer to signal deerFinished (which they do in the deliverPresents() function after they are done helping), then after every deer is finished delivering presents Santa resets the deerReady flag to false and then returns to his office in the northpole to sleep (wait on wakeSanta semaphore).

```
reindeer (88) arrived to the north pole and is waiting for Santa
elf (68) has a problem and goes to Santa for help
elf (76) has a problem and goes to Santa for help
```

Then here we can see that the loop will continue forever with deer returning from their vacations and elves running into problems while working, with Santa helping them.

History of the problem

There have been many papers discussing this problem and proposing new solutions to it since it was published, in this section I will highlight three of these papers and quickly summarize what they did differently.

1. In his 1998 paper[3] Mordechai Ben-ari points out one issue with Trono's solution that was presented in the original paper and then he solves this issue by making an implemntation in Ada95 protected objects and Rendezvous which are more advanced synchronization constructs that are available in Ada95 and some other languages, falicitate making a more provably correct solution that can withstand a malicious scheduler, he also implements the same solution in Java which turns out more complex due to a lack of those advanced synchronization constructs.

2. In a 2003 publication[4] Nick Benton implements a solution to the problem in polyphonic C# (an extension to C# that provides more advanced concurency constructs), the solution he presents is shorter, more elegant and more effecient when compared to Ben-ari's solution in Ada95.

3. In a 2008 paper[5] Jason Hurt and Jan B. Pedersen show multiple solutions in multiple language using a multitude of different concurrency mechanisms and use the Santa Cluase problem as a way to compare the strengths and weaknesses of these different concurrency mechanisms.

Conclusion

The solution I present in this paper is not groundbreaking as C doesn't implement any advanced concurrency mechanisms that would allow for a more effecient or elegant solution, but, my solution does not run into the problem of Santa delivering presents and returning to the north pole while Reindeer are still waiting in the stable and not helping that is present in the original solution that Trono presented in his paper[1] (Ben-Ari pointed out this issue in his paper[3]).

[3] M. Ben-Ari. How to solve the Santa Claus problem. Concurrency: Practice and Experience, 10(6):485–496, 1998.

[4] N. Benton. Jingle Bells: Solving the Santa Claus Problem in Polyphonic C#. Technical report, Microsoft Research, Cambdridge UK, 2003.

[5] Jason, H. U. R. T., and Jan B. Pedersen. "Solving the Santa Claus Problem: A comparison of various concurrent programming techniques." Communicating Process Architectures 2008: WoTUG-31 66 (2008): 381.