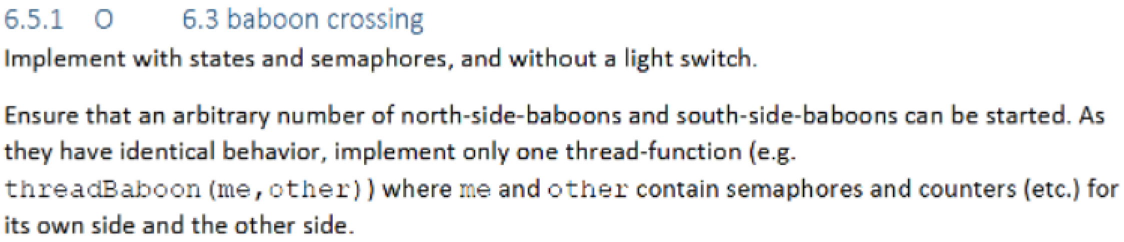
Week 5 - SYNC Assignment

Group 3

Table of contents

First Assignment

Problem description



6.5.1 O 6.3 baboon crossing

Implement with states and semaphores, and without a light switch.

Ensure that an arbitrary number of north-side-baboons and south-side-baboons can be started. As they have identical behavior, implement only one thread-function (e.g. `threadBaboon(me,other)`) where `me` and `other` contain semaphores and counters (etc.) for its own side and the other side.

Solution

To solve this problem, we used states and semaphores. We made a class Baboon with all the properties we need to solve this problem. Also, the ‘States’ class specifies all possible states (‘States’ acts as an enumerator). The semaphores act as a queue that stops baboons from entering the rope when there are baboons on the rope from the opposite direction.

The implementation works as follows:

First, when the baboon enters the thread, we add it to to the queue and check if the state is empty or the state is its state, then, we allow it to enter and we signal its semaphore, but if the state is the other’s state, then we change the state to Queued and waits.

When the state is Queued, we don't allow any baboon on my side to use the rope so that we make sure that each thread has a fair chance of using the rope, therefore making sure our solution is starvation-free.

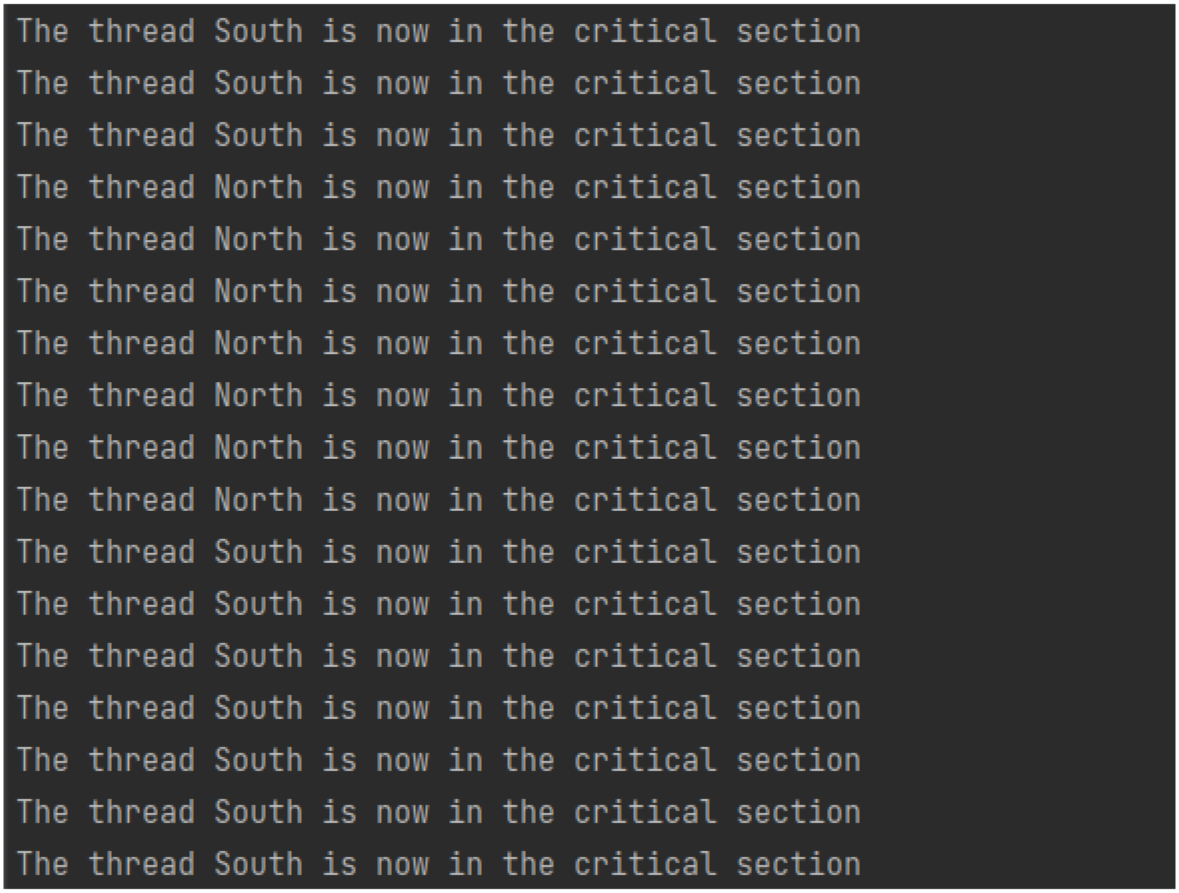
Afterwards, we used a capacity multiplex to make sure that only 5 baboons can enter the critical section at the same time.

Finally, we decrement the baboonsCount and check if it is 0 and the state is Queued, then we signal all the baboons who are waiting in the queue. Otherwise we set the state to empty.

```
from Environment import *


def person_thread(me, other):
   while True:
       mutex.wait()

       me.queueCounter.v += 1

       while state.v == other.transition:
           me.queue_cv.wait()

       me.queueCounter.v -= 1

       me.counter.v += 1

       if state.v == States.NEUTRAL:
           state.v = me.rule

       elif state.v == other.rule and me.counter.v > other.counter.v:
           state.v = me.transition

       while state.v == other.rule or state.v == me.transition:
           me.cv.wait()

       mutex.signal()

       print(f"walking field...{me.name}")

       mutex.wait()

       me.counter.v -= 1

       if state.v == me.rule and other.counter.v > me.counter.v:
           state.v = other.transition

       if state.v == other.transition and me.counter.v == 0:
           state.v = other.rule
           if other.counter.v > 0:
               other.cv.notify_all()
           if me.queueCounter.v > 0:
               me.queue_cv.notify_all()

       if state.v == me.rule and me.counter.v == 0:
           state.v = States.NEUTRAL

       mutex.signal()


class States:
   NEUTRAL = "neutral"
```

```
   HEATHENS_RULE = "heathens rule"
   PRUDES_RULE = "prudes rule"
   HEATHENS_TRANSITION = "transitioning to heathens"
   PRUDES_TRANSITION = "transitioning to prudes"


class Person(object):
   def __init__(self, queueCounter, counter, queue_cv, cv, rule, transition, name):
       self.queueCounter = queueCounter
       self.counter = counter
       self.queue_cv = queue_cv
       self.cv = cv
       self.rule = rule
       self.transition = transition
       self.name = name


mutex = MyMutex("mutex")

heathensQueue_cv = MyConditionVariable(mutex, "heathensQueue_cv")
heathens_cv = MyConditionVariable(mutex, "heathens_cv")

prudesQueue_cv = MyConditionVariable(mutex, "prudesQueue_cv")
prudes_cv = MyConditionVariable(mutex, "prudes_cv")

heathensQueue = MyInt(0, "heathensQueue")
prudesQueue = MyInt(0, "prudesQueue")

heathensCounter = MyInt(0, "heathensCounter")
prudesCounter = MyInt(0, "prudesCounter")

state = MyString(States.NEUTRAL, "state")


def setup():
   prude = Person(prudesQueue, prudesCounter, prudesQueue_cv, prudes_cv,
States.PRUDES_RULE,
                  States.PRUDES_TRANSITION, "prude")
   heathen = Person(heathensQueue, heathensCounter, heathensQueue_cv, heathens_cv,
States.HEATHENS_RULE,
                    States.HEATHENS_TRANSITION, "heathen")
   for i in range(5):
       subscribe_thread(lambda: person_thread(heathen, prude))
   for i in range(5):
       subscribe_thread(lambda: person_thread(prude, heathen))
```

Output

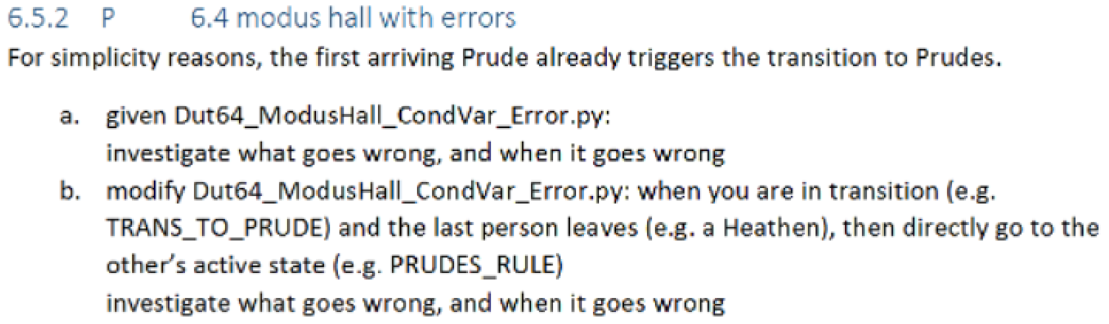


```
The thread South is now in the critical section
The thread South is now in the critical section
The thread South is now in the critical section
The thread North is now in the critical section
The thread North is now in the critical section
The thread North is now in the critical section
The thread North is now in the critical section
The thread North is now in the critical section
The thread North is now in the critical section
The thread North is now in the critical section
The thread South is now in the critical section
The thread South is now in the critical section
The thread South is now in the critical section
The thread South is now in the critical section
The thread South is now in the critical section
The thread South is now in the critical section
The thread South is now in the critical section
```

A video is also included alongside this document.

Second Assignment

Problem description



6.5.2 P 6.4 modus hall with errors

For simplicity reasons, the first arriving Prude already triggers the transition to Prudes.

a. given Dut64_ModusHall_CondVar_Error.py:
   investigate what goes wrong, and when it goes wrong
b. modify Dut64_ModusHall_CondVar_Error.py: when you are in transition (e.g. TRANS_TO_PRUDE) and the last person leaves (e.g. a Heathen), then directly go to the other's active state (e.g. PRUDES_RULE)
   investigate what goes wrong, and when it goes wrong

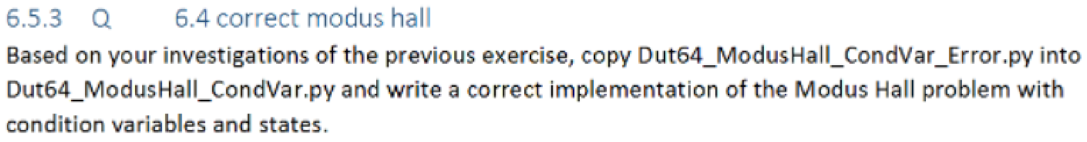
Solution

a) We have investigated the given solution and realized that at a certain time, **starvation** occurs for the Heathens.
b) We have created a new file with the same code, where we added a new condition so that if we are in transition and the last person leaves, we will go directly to the other person's active state. After investigating, we noticed that a **deadlock** occurred.

Videos showcasing the problems will be provided alongside this document.

Third assignment

Problem description



6.5.3 Q 6.4 correct modus hall

Based on your investigations of the previous exercise, copy Dut64_ModusHall_CondVar_Error.py into Dut64_ModusHall_CondVar.py and write a correct implementation of the Modus Hall problem with condition variables and states.

Solution

For this assignment, we had to refactor the previous one making sure that this implementation avoided the above mentioned problems.

Firstly, we solved this assignment using states and condition variables. We made a class Person with all the properties we need to solve this problem. Also, the ‘States’ class specifies all possible states (‘States’ acts as an enumerator).

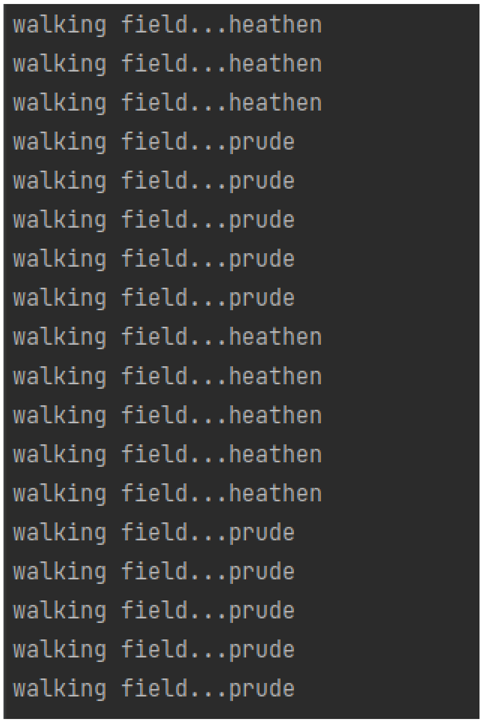
The solution works as follows:

- There is a variable that keeps track of the state of the road. There are 5 possible states: Neutral, Heathens Rule, Prudes Rule, Transition to Heathens and Transition to Prudes.
- When the road is controlled by one of the two groups, the other one will be stopped by a condition variable (cv) that will act as a queue. When the road is in either of the two transition states, both parties are prevented from crossing the path until everyone currently on the path leaves
- Allowing only one person to pass the path at a given time is achieved using a shared mutex.
- A queue and a counter are used for both groups to keep track of how many people are waiting in the queue and how many people are currently on the path.
- The queue condition variable and queue counter is used to make sure that both groups have a fair chance to enter the road. therefore making sure our solution is starvation-free.

```
from Environment import *


def person_thread(me, other):
   while True:
       mutex.wait()

       me.queueCounter.v += 1

       while state.v == other.transition:
           me.queue_cv.wait()

       me.queueCounter.v -= 1

       me.counter.v += 1

       if state.v == States.NEUTRAL:
           state.v = me.rule

       elif state.v == other.rule and me.counter.v > other.counter.v:
           state.v = me.transition

       while state.v == other.rule or state.v == me.transition:
           me.cv.wait()

       mutex.signal()

       print(f"walking field...{me.name}")

       mutex.wait()

       me.counter.v -= 1

       if state.v == me.rule and other.counter.v > me.counter.v:
           state.v = other.transition

       if state.v == other.transition and me.counter.v == 0:
           state.v = other.rule
           if other.counter.v > 0:
               other.cv.notify_all()
           if me.queueCounter.v > 0:
               me.queue_cv.notify_all()

       if state.v == me.rule and me.counter.v == 0:
           state.v = States.NEUTRAL

       mutex.signal()


class States:
   NEUTRAL = "neutral"
   HEATHENS_RULE = "heathens rule"
   PRUDES_RULE = "prudes rule"
   HEATHENS_TRANSITION = "transitioning to heathens"
   PRUDES_TRANSITION = "transitioning to prudes"


class Person(object):
   def __init__(self, queueCounter, counter, queue_cv, cv, rule, transition, name):
       self.queueCounter = queueCounter
       self.counter = counter
       self.queue_cv = queue_cv
       self.cv = cv
       self.rule = rule
       self.transition = transition
       self.name = name


NR_OF_PRUDES = 6
NR_OF_HEATHENS = 9
mutex = MyMutex("mutex")

heathensQueue_cv = MyConditionVariable(mutex, "heathensQueue_cv")
heathens_cv = MyConditionVariable(mutex, "heathens_cv")

prudesQueue_cv = MyConditionVariable(mutex, "prudesQueue_cv")
prudes_cv = MyConditionVariable(mutex, "prudes_cv")

heathensQueue = MyInt(0, "heathensQueue")
prudesQueue = MyInt(0, "prudesQueue")

heathensCounter = MyInt(0, "heathensCounter")
prudesCounter = MyInt(0, "prudesCounter")

state = MyString(States.NEUTRAL, "state")


def setup():
   prude = Person(prudesQueue, prudesCounter, prudesQueue_cv, prudes_cv,
States.PRUDES_RULE,
                  States.PRUDES_TRANSITION, "prude")
   heathen = Person(heathensQueue, heathensCounter, heathensQueue_cv, heathens_cv,
States.HEATHENS_RULE,
                    States.HEATHENS_TRANSITION, "heathen")
   for i in range(NR_OF_HEATHENS):
       subscribe_thread(lambda: person_thread(heathen, prude))
   for i in range(NR_OF_PRUDES):
       subscribe_thread(lambda: person_thread(prude, heathen))
```

Output

.



```
walking field...heathen
walking field...heathen
walking field...heathen
walking field...prude
walking field...prude
walking field...prude
walking field...prude
walking field...prude
walking field...heathen
walking field...heathen
walking field...heathen
walking field...heathen
walking field...heathen
walking field...prude
walking field...prude
walking field...prude
walking field...prude
walking field...prude
```

A video is also included alongside this document.