Week 4 - SYNC Assignment

Group 3

Table of contents

First Assignment

Problem description

6.4.1 L 4.4 dining philosophers (II)

Implement a deadlock free solution with condition variables, by avoiding the *hold-and-wait* deadlock condition.

Solution

Our solution uses a mutex to protect the forks array. We keep track of an array of condition variables so that we can signal the other philosophers who need the fork to eat. When the forks are available, we notify the left and right philosophers to use the fork to eat.

```
from Environment import *

N = 5
forks = [True for i in range(N)]
mutex = MyMutex("mutex")
condition_variables = [MyConditionVariable(mutex, f"condition{i}") for i in
range(N)]


def philosopher_thread(i):
   while True:
       print("think()")
       mutex.wait()
       while not is_Available(i):
           condition_variables[i].wait()

       forks[left(i)] = False
       forks[right(i)] = False
       mutex.signal()

       print("eat()")

       mutex.wait()
       forks[left(i)] = True
       forks[right(i)] = True
       condition_variables[left(i)].notify()
       condition_variables[right(i)].notify()
       mutex.signal()
```

```
def left(i):
   return i


def right(i):
   return (i + 1) % N


def is_Available(i):
   if forks[left(i)] and forks[right(i)]:
       return True


def setup():
   subscribe_thread(lambda: philosopher_thread(0))
   subscribe_thread(lambda: philosopher_thread(1))
   subscribe_thread(lambda: philosopher_thread(2))
   subscribe_thread(lambda: philosopher_thread(3))
   subscribe_thread(lambda: philosopher_thread(4))
```

Output

```
think()
eat()
eat()
think()
think()
eat()
eat()
think()
think()
eat()
eat()
think()
think()
eat()
eat()
think()
eat()
think()
think()
eat()
eat()
```

A video is also included alongside this document.

Second Assignment

Problem description

6.4.2 M 5.1 dining savages

Implement with condition variables with the following requirements:

- real storage for servings (use `MyBag`)
- carnivore + vegetarian savages

- carnivore + vegetarian cooks

So the pot contains a variety of servings, and there are a variety of savages around the pot. And sometimes it might occur that the pot does contain servings, but not suited for the available savages.

Ensure that an arbitrary number of carnivore-savages and vegetarian-savages can be started.

Solution

For this assignment, we decided to create 4 conditional variables for each “entity” we have in the description so that we can easily keep track of the actions which need to be done. The main idea is that the savages keep eating from the pot until they have no more vegetarian/carnivore food. In this case, the cook (vegetarian/carnivore depending on the case) is notified and starts preparing the food while the savages are waiting. Once the cook finished preparing, he notifies all the savages to resume eating.

```
from Environment import *

bagCapacity = 6
bag = MyBag(bagCapacity, "storage")
mutex = MyMutex("mutex")
NR_OF_CARNIVORES = 10
NR_OF_VEGETARIANS = 6
cv_carnivore_savage = MyConditionVariable(mutex, 'cv_carnivore_savage')
cv_vegetarian_savage = MyConditionVariable(mutex, "cv_vegetarian_savage")
cv_carnivore_cook = MyConditionVariable(mutex, 'cv_carnivore_cook')
cv_vegetarian_cook = MyConditionVariable(mutex, "cv_vegetarian_cook")


def prepareFood(i):
```

```
    arr = ["Carnivore", "Vegetarian"]
    if arr.__contains__(i):
        return i


def vegetarianCookThread():
    while True:
        mutex.wait()
        while bag.contains(prepareFood("Vegetarian")):
            cv_vegetarian_cook.wait()

        while bag.size() < bagCapacity:
            bag.put(prepareFood("Vegetarian"))

        print("prepareFoodForVegetarian()")
        cv_vegetarian_savage.notify_all()
        print("Call all vegetarians to eat! ")
        mutex.signal()


def vegetarianSavagesThread():
    while True:
        mutex.wait()
        while not bag.contains(prepareFood("Vegetarian")):
            cv_vegetarian_savage.wait()

        print("vegetarianEat()")
        bag.get(prepareFood("Vegetarian"))

        if not bag.contains(prepareFood("Vegetarian")):
            cv_vegetarian_cook.notify()
            print("No more food available for vegetarians!")
        mutex.signal()


def carnivoreCookThread():
    while True:
        mutex.wait()
        while bag.contains(prepareFood("Carnivore")):
            cv_carnivore_cook.wait()

        while bag.size() < bagCapacity:
            bag.put(prepareFood("Carnivore"))

        print("prepareFoodForCarnivore()")
        cv_carnivore_savage.notify_all()
        print("Call all carnivores to eat!")
```

```
        mutex.signal()

def carnivoreSavagesThread():
    while True:
        mutex.wait()
        while not bag.contains(prepareFood("Carnivore")):
            cv_carnivore_savage.wait()

        print("CarnivoreEat()")
        bag.get(prepareFood("Carnivore"))

        if not bag.contains(prepareFood("Carnivore")):
            cv_carnivore_cook.notify()
            print("No more food available for carnivores!")
        mutex.signal()


def setup():
    subscribe_thread(vegetarianCookThread)
    subscribe_thread(carnivoreCookThread)
    for i in range(NR_OF_CARNIVORES):
        subscribe_thread(vegetarianSavagesThread)
    for i in range(NR_OF_VEGETARIANS):
        subscribe_thread(carnivoreSavagesThread)
```

Output

```
CarnivoreEat()
vegetarianEat()
CarnivoreEat()
CarnivoreEat()
vegetarianEat()
No more food available for vegetarians!
CarnivoreEat()
No more food available for carnivores!
prepareFoodForCarnivore()
Call all carnivores to eat!
CarnivoreEat()
CarnivoreEat()
prepareFoodForVegetarian()
Call all vegetarians to eat!
vegetarianEat()
CarnivoreEat()
```

A video is also included alongside this document.

Third assignment

Problem description

6.4.3 N 4.2 readers-writers

Implement with condition variables, and make it configurable who has priorities.

Ensure that an arbitrary number of reader and writer threads can be started (e.g. N=7).

Solution

To solve this exercise, we use two condition variables. One for the writer and one for the reader.Also, we made two threads for both readers and writers.
Our solution is as follows:

Reader thread:

Reader thread works by keeping track of the number of active and waiting readers. We use condition variables to check how many writers are active. If there is any writer active, the reader waits for the writer to finish. If the writer has priority and there are writers waiting, then we allow them to enter and the waiting reader has to wait. When all threads have stopped reading. We have to check if the writer has priority, and they are waiting, then we notify them to enter. Otherwise, we check if there are waiting readers to notify them.

Writer thread:

For the writer thread, the same logic applies.

```
from Environment import *


def reader_thread():
   while True:
       mutex.wait()
       while active_writers.v > 0 or (waiting_writers.v > 0 and priority.v):
           waiting_readers.v += 1
           cv_reader.wait()
           waiting_readers.v -= 1
       active_readers.v += 1
       mutex.signal()
```

```
        print("read()")

        mutex.wait()
        active_readers.v -= 1
        if active_readers.v == 0 and (
                (priority.v and waiting_writers.v > 0) or (waiting_writers.v > 0
and waiting_readers.v == 0)):
            cv_writer.notify()
        mutex.signal()


def writer_thread():
    while True:
        mutex.wait()
        while active_readers.v > 0 or active_writers.v > 0 or (waiting_readers.v >
0 and not priority.v):
            waiting_writers.v += 1
            cv_writer.wait()
            waiting_writers.v -= 1
        active_writers.v += 1
        mutex.signal()

        print("write()")

        mutex.wait()
        active_writers.v -= 1
        if waiting_readers.v > 0 and not priority.v:
            cv_reader.notify_all()
        elif waiting_writers.v > 0:
            cv_writer.notify()
        mutex.signal()


mutex = MyMutex("mutex")
cv_reader = MyConditionVariable(mutex, "cv_reader")
cv_writer = MyConditionVariable(mutex, "cv_writer")
active_readers = MyInt(0, "active_readers")
active_writers = MyInt(0, "active_writers")
waiting_readers = MyInt(0, "waiting_readers")
waiting_writers = MyInt(0, "waiting_writers")
priority = MyBool(False, "priority")
NR_OF_READERS = 6
NR_OF_WRITERS = 6


def setup():
    for i in range(NR_OF_READERS):
```

```
        subscribe_thread(reader_thread)
    for i in range(NR_OF_WRITERS):
        subscribe_thread(writer_thread)
```

Output

In the picture below, the writers have priority and the program was debugged by checking each thread one by one.

```
read()
read()
write()
write()
write()
write()
write()
write()
```

A video is also included alongside this document.