## Week 4

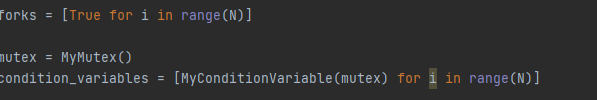
### L 4.4 dining philosophers (II)

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

Hint (change font color):   
use Booleans to indicate if the forks are in use and condition variables to notify changes in the status of the forks. There is no need for separate mutexes for each fork.   
(end of hint)

**Solution:**

This solution uses a mutex to protect the forks variable. We do keep track of an array of conditional variables. One for each of the philosophers (and each fork).





The solution works by having a conditional variable signal the other philosophers who need the forks. When new forks become available.

### M 5.1 dining savages

Implement with condition variables with the following requirements:

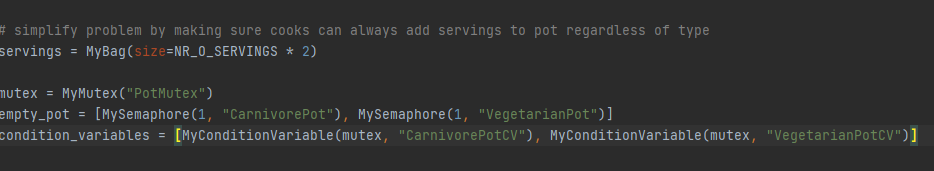
* real storage for servings (use MyBag)
* carnivore + vegetarian savages
* carnivore + vegetarian cooks

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 our solution we used the MyBag class, a mutex, two semaphores and two conditional variables. The mutex is used to protect the content of the MyBag class. The semaphores are to signal the cooks who need to make the food. The two conditional variables are used to single the hungry savages when food is ready.



For this solutions we have two types of threads. A savage thread and a cook thread. The cook thread is responsible for the food.

**Cook Thread**

First we use our semaphore where we initialized it with 1. So the cook can continue on its first run to serve food. After putting food in our bag we signal the savage of that food type. This is where we notify the conditional variable for. 

**Savage Thread**

The savage thread first check if there is any food. If there is not it will wait for the conditional variable. If a savage has eaten and the pot is empty we signal our cook to make more.



### N 4.2 readers-writers

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

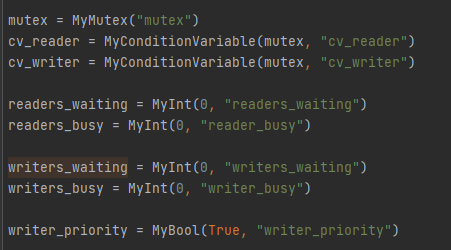
Ensure that an arbitrary number of reader and writer threads can be started (e.g. N=7).

Hint (change font color):   
mutex = MyMutex("mutex")  
cv\_reader = MyConditionVariable(mutex,"cv\_reader")  
cv\_writer = MyConditionVariable(mutex,"cv\_writer")  
nrof\_reader\_busy = MyInt(0,"reader\_busy")

nrof\_writer\_busy = MyInt(0,"writer\_busy")  
nrof\_reader\_wait = MyInt(0,"reader\_wait")  
nrof\_writer\_wait = MyInt(0,"writer\_wait")  
writer\_prio = MyBool(False,"writer\_prio")   
(end of hint)

**Solution:**

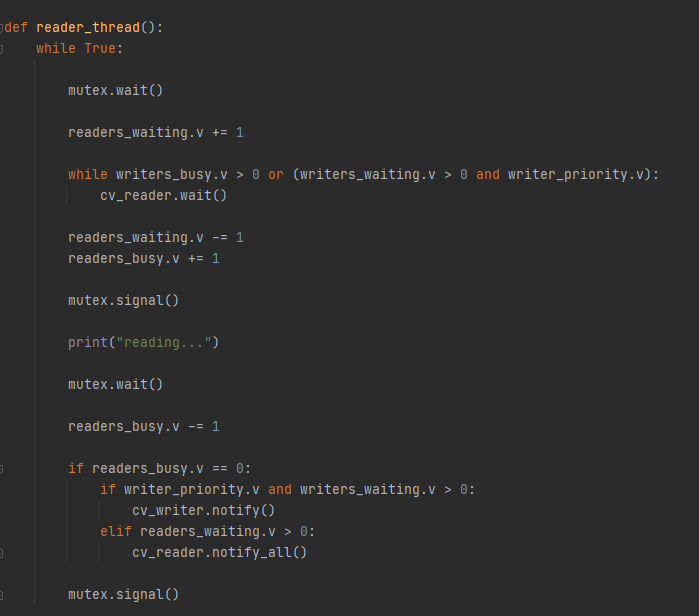
These are the variables that we used. We have several counters for each type and a configurable priority.



**Reader Thread:**

The reader thread works by keeping track of the amount of readers busy and waiting. We use the conditional variable to check for how many writers are busy. If there are any writer busy we wait for that writer to finish. If the writer has priority and there are writers waiting we will also stay wait for the conditional variable.

When all threads have stopped reading. We have two options either the writer has priority and they are waiting we make sure they can continue. Otherwise we check if there are readers . If there are readers we notify them.



**Writers Thread:**

For the writer thread we also keep track of the amount of thread waiting and are busy. We use the conditional variable to check for how many writers are busy. There can only be one writer at a time. We also check whether any readers are busy. If there are any ready busy we wait for that reader to finish. If the reader has priority and there are readers waiting we will also stay wait for the conditional variable.

When the write thread has stopped writing. We have the same options as the reader thread.

**s**