Dining Philosophers

Free of Deadlocks And Starvation Version

The versions I used is actually vulnerable to deadlocks. However, I implement a way of detecting and recovering from the deadlock making it a deadlock free. It is also free from starvation since each philosopher is ensured at least one use of both forks at the same time.

Implementaiton

- In the beginning all the threads grab the left fork using `sem_wait()`
  - A 1 second `sleep()` is put right after the `sem_wait()` to allow all the other threads to grab the left fork before any thread manages to get the right fork.
- For the right fork, `sem_timedwait()` is used in order to check for deadlocks.
- A certain timestamp in the future is given to the function for it to return after a certain amount of blocking time.
  - If the time passed is 'current_time + 1 sec' then it will try to obtain the semaphore
  - If it couldn't, it will block for 1 sec and then return with an error code
    - `0` for success
    - `-1` for failure
- I passed 1 sec, although 0 seconds would do the same job.
- In the case that I get back a `-1`, I release the left fork using `sem_post()`
- A random time for `sleep()` is called
- Then I loop back to try and get left and right again, in which case, other threads may be done using.

Deadlock Version

The version with deadlock is similar to the deadlock free version. The only difference is whether I loop back to try and get left and right fork after some random time or just exist the function reporting a deadlock.

Starvation Version

This version focused on trying to get one of the philosophers to starve while at the same time avoiding deadlocks. So, I tried to apply the idea of using `room` from the book, but instead call it `seats`. What I try to do is allow up to only 4 to eat per run. I create 4 semaphore seats, and the seats are given out on a First-Come-First-Served bases. The first thread/philosopher comes in to take `seat 0`, then the second philosopher comes in to take `seat 1` and so on and so forth. The unlucky thread doesn't get a seat and ends up starving.

Implementation

- The idea of grabbing a seat uses `sem_timedwait()` to check whether someone else grabbed this seat
- In this scenario, `time.tv_sec` is passed `0` instead of `1` since I don't want it to wait for one second if it finds that the seat is taken. Rather, I want to know that the seat is taken and move on to another one.
- Also, after grabbing a seat, a thread might finish execution right away, so when the last thread comes he could take that seat. Therefore, I call `sleep(1)` which allows all the seats to be assigned before any of the philosophers start eating.

- If `sleep(1)` was not used and two or three threads tried to grab the same seat, they would just wait for whoever is on it to finish and then grab it.