Logan Sims
CSCI 322
HW04
2/23/2015

Part (a)

global invariant:
for all philosophers, if philo[i] is eating, philo[i-1] and philo[i+1] are not

Coarse-grained Solution:

```
####################################################
boolean eating[n]

process Philo[i=0 to n]
	loop
		<await (!eating[i-1] and !eating[i+1]) getForks();>
		eat();
		<releaseForks();>
	end loop;
end ;
####################################################
```

Fine-grained Solution Explanation:

The fine-grained solution realizes the coarse-grained await statement with an array of personal semaphores for each philosopher, an array of personal booleans, a while loop, and a semaphore called the baton. When a philosopher is delayed they will set their waiting variable to true release the baton, and enter a while loop. In this loop they wait on their personal semaphore until a philosopher wakes them up and gives them the baton. At this point they will see if they can eat, if not they will go back to sleep. By making sure picking up the left and right forks are an all or nothing situation it is guaranteed not to deadlock. The getForks and releaseForks() functions are simply setting eating[n] (n for the given philosopher) to true or false respectively. In the fine-grained solution during the release of the forks the baton is also passed to a waiting philosopher with the lowest index.

Part (b)

The coarse-grained solution remains the same due to the difference in the two problems being the implementation of the await statement.

global invariant:
for all philosophers[i], if philo[i] is eating, philo[i-1] and philo[i+1] are not.

Coarse-grained Solution:

```
####################################################
boolean eating[n];

process Philo[i=0 to n]
	loop
```

```
                <await (!eating[i-1] and !eating[i+1]) getForks();>
                eat();
                <releaseForks();>
        end loop
end
##################################################
```

Fine-grained Solution Explanation:

A FIFO queue has been added for this implementation. Now when a philosopher must delay to eat they enter a queue. Every time the eating state of a philosopher changes the baton is passed to the first philosopher in the queue, if they can eat they will, if not they will release the baton and stay first in the queue. If no one is in the queue the baton is simply released. Once a philosopher has entered the queue they are guaranteed to eat as they will be given infinite many chances to eat. Assuming weak fairness the philosopher will get to eat because as soon as it's eat condition becomes true, the philosopher will have a chance to see it.