Assignment 2 Report

Author: Patrick Fleming
Student Number: C3253586

Testing

The cases tested for P1 where disproportionate numbers of humans/aliens and large numbers of both. Also tested when there are 0 or less the program doesn't do anything. The semaphore makes sure that there is no deadlock or starvation.

The cases tested for P2 if the processors set for the tasks is 0 the program starves as no thread can run. The other tasks with processors assigned will run. When the task to be run is a large number it can cause the other tasks to wait for it to complete but if there are more than 1 processor assigned the other tasks utilize the other processors to complete their task. Large numbers of tasks and low numbers of processes cause the completion time to run long, but they do all run and complete.

The cases tested for P3 are the same for P2 being that they are the same task but just different solutions.

Edge cases

I ran into a problem where the threads would start running as soon as they were created not allowing for them to run simultaneously to get around this I used a CountDownlatch to pause the task threads in there run so that they all start at the same time™(as close as I could get to the same time).

In the first and second section of the assignment I would pass the latch in the constructor but for p3 this cause thread exceptions because the latch was null to get around this, I created a function that returns the latch from the main program and used that to assign the latch in the class definitions.

For part 3 I was unsure how to use monitor to make sure that there isn't starvation or deadlock at first I thought maybe making the task that each thread has to complete be the critical section but then I can't simulate different amounts of task accessing depending on the amount of processors assigned so what I did was for the schedulers was give them a critical section that the task would have to access to get the processor and when they do they run and once they finish running they move into another critical section to release the processor and then notify the other tasks that they can try and get a processor to run.