Pandemic Simulator
CS5500 Parallel Programming
Benji Stewart, Connor Osborne

1 Description

As your final project for CS5500 Parallel Programming, you will be implementing a simple simulation program using MPI for simulating the spread of an infectious disease in a "city". The simulation will demonstrate the rate of spread for a disease among a population of people and the ratios, over time, of people susceptible to the disease, people infected with the disease, and people who have gained immunity to the disease.

2 Requirements

To simulate the spread of a disease amongst a population you will need to represent two different kinds of places, homes and non-homes.
Every home will start out with 4 "people", how you choose to represent these people is up to you, and for simplicity's sake, any one building will only be able to hold 4 people at one time.
A person will exist in one of 3 states at any time:

- Normal
- Infected
- Immune

As you can guess a normal person is capable of being infected if they are in the same building as an infected person. The percent chance that a normal person is infected is up to you. (Play around with the percentage to see what kind of results you get.) The percent chance of a person being infected will increase based on how many of the people in the building are infected. After being infected for 3 days a person will become immune. They will no longer be vulnerable to infection, nor will they continue to spread the disease.

You will need to have your simulation run for at least 25 days. At the beginning of each day the people will leave their homes and then go around to different non-homes 3 times before going back home at the end of the day. At the end of each day have a way of reporting how many people are still normal, how many people are infected, and how many people have become immune to the disease.

3 Helpful Hints

The best way to start is to have the processes split up such that half are "homes" and half are "non-homes."
Be sure to plan ahead on whether to use blocking and non-blocking MPI functions to avoid race conditions.
Use MPI probe functions for checking whether a process is ready to move on to the next set of work or not.