User guide:

The program coordinates the execution of a chef and 3 saladmaker programs where the chef provides the vegetables and the saladmakers produce the salads. At the end of the program, the following statistics are produced:

- Number of salads produced.
- Weight of each vegetable and total weight of vegetables used by each salad maker
- Total waiting and total working time for each saladmaker
- A temporal log to reveal timeline of operation of each saladmaker
- Listings of time periods in which at least two saladmakers were busy

The user will invoke the program using the following invocation and flags:

The chef program must be first invoked before any of the saladmakers as follows

```
./chef -m numOfSalads -n cheftime
```

Here -m is the flag for number of salads to be produced and -n is the flag for the approximate break time for chef

The chef program will output a shared memory id which then must be provided as an argument for invocation of all saladmakers as follows:

```
./saladmakerX -m saladmakertime -s shmid
```

Here ***X*** indicates the number of saladmaker (from 1 to 3 inclusive) -m flag is the approximate time for preparation of salad and -s is the flag for shared memory id.

Providing all these flags is necessary for the successful invocation of the programs.

Technical implementation:

The chef program creates shared memory and attaches to it while also outputting the shared memory id. The shared memory key is generated through the use of ftok(). Further, semaphores are used to control the synchronization of the four processes. Chefsemaphore is used to implement any waiting on chef, saladmaker semaphores are used to implement any waiting on saladmakers, mutex semaphore is used for controlled access to shared memory data, while the waitsaladmaker semaphores are used to wait for end of execution of saladmakers before the chef can finally terminate.

The logical flow of the program is as follows:

The saladmakers call wait() on their semaphores and thus have to wait for vegetables to be provided by the chef. As soon as chef picks up the vegetables and decides on a saladmaker to be used, that respective saladmaker is allowed to proceed and make a salad. Meanwhile, the chef uses wait on its semaphore which is signaled to proceed when the saladmaker has picked up its vegetables. An random wait near a certain value of seconds is used as a break for the chef.

On the saladmaker end, the saladmaker waits for the chef to signal it in order to pick up the vegetables. The saladmaker, after picking up the vegetables, signals the chef to proceed with execution so that it can pick more vegetables while the saladmaker makes its salad. This allows for parallel working of saladmakers at times.

Meanwhile, the weights of vegetables being used keep updating. As soon as all the salads are made, a flag in the shared memory is set to true so that all these weight statistics can be stored in the shared memory for access by the chef. Further, the time of start and end of wait and work times for saladmakers is taken relative to the start of the chef program and the total is also updated in the shared memory for later access by the chef. When the completion flag in shared memory is set to true, the saladmakers exit while the chef waits for them to exit using the waitsaladmakers semaphores. As soon as all the saladmakers end their execution, the chef program prints the calculated statistics, as well as the temporal log and the overlapping intervals calculated and stored in two different files.

For calculation of overlapping intervals, as soon as active saladmakers number exceeds 1 and there was no previously started overlapping interval, the time is noted as the start of interval time. Then, as soon as active saladmakers remaining becomes equal to 1, and there was a previously started overlapping interval, the time is noted as the end of interval time. The flags are then reset while the intervals are recorded.

Design Choices:

The use of structs to store shared memory data allows for convenient and easy access to data in the shared memory. Further, using the semaphores and flags chosen in the above described way allow for effective synchronization of tasks as required in the task description. All other design choices have been thoroughly explained under technical implementation.