**COMP304 Project 2**

Murat Güç - 68967 Yakup Can Karacaoğlu - 69063

**PART I**

In the part I, new threads, queues and mutexes are created for a synchronized santa gifting operation. There are 4 threads and they take elf a, elf b, santa and controller as their functions. The queues are created to keep track of waiting gifts for specific operations, so there are painting, assembly, quality assurance , packaging and delivery queues in the code. All queues also have mutexes for the mutual exclusions, because the queues can be modified by different threads. For instance, a task is enqueued to the packaging queue in the controller thread, but it can be dequeued by elf A or elf B. So, mutexes should be used and “pthread\_mutex\_t” are used in the implementation.

Most of the work is done by the controller thread in the code. Elves and Santa just dequeue from their queues and sleep the necessary amount of time, then print into the log file. The log file is the output documentation of the simulation.The controller queue evaluates the demands of every task, and adds them into the necessary queue. It evaluates the demands by checking the demands array of the gifts, the array element is 1 if the specific demand is needed.

Lastly, the simultaneous works of painting and QA in type 4 and the simultaneous works of assembly and QA in type 5 should be mentioned. When any of this type of gift reaches the controller function, the controller adds it to both queues. But the demands of these kinds of gifts should be controlled globally, because they will turn back to the controller twice. One turns back from assembly or painting, and the other turns back from qa, only one of them should be added to the packaging queue. This global checking is done by using six separated queues and their mutexes. The queues check if a gift with a determined ID is added to a queue and done its operation.

**PART II**

It is implemented in the project2\_2.c file. In this part, firstly, we have changed the possibility from %40 to %50. When we see the results, QA has been suffering from starvation. Then, to resolve that issue, we have added a qa\_flag variable to keep track of whether it has done the qa or not in the Santa part. Algorithm became as follows: if the size of the qa queue is bigger than or equal to 3, it proceeds with qa. Otherwise, it checks the delivery queue to figure out if it is empty or not, then in terms of not empty case, it proceeds with delivery. Otherwise, Santa does qa.

**PART III**

It is implemented in the project2\_2.c file. In the Task struct data type within queue.c, is\_emergency flag is added to handle if the task is emergency or not. It is initialized to 0 at first. Then, to implement an emergency situation, we thought that urgent tasks must be added to the top of the queue. Corresponding code to add to top function is written under Enqueue method as “else if statement” which checks is\_emergency is equal to the 1. Then, while creating tasks under main in the project2\_2.c file, it assigns an emergency flag to the 1 at each 30 seconds, and sends it to the controller queue.

**Notes:**

* Events\_1.log is the log file of project2.c, and Events\_2.log is the log file of project2\_2.c.
* Usual behavior of .c files is to extract a log file named events.log.
* The gcc compilation should be done with -lpthread flag.

**GitHub Link:**

https://github.com/cankaracaoglu/COMP304\_Fall22\_Project\_2