Comp 304- Project 2 Spacecraft Control with Pthreads

*Andrew Bond, İrem Demir*

General Structure of Our Implementation:

**main:** argument parsing, storing time variables initially and creating queues, threads are done in here. Also, it starts event log with writing the header. Moreover, created threads are joined here as suggested in the documentation [1].

**LandingJob, LaunchJob, AssemblyJob, EmergencyJob**: thread functions that each responsible for specific job type. They create jobs with wanted probabilities in t seconds. They also decide the unique id for each job. This is made through incrementing a global variable. Since these may cause inconsistencies, a mutex is used for id creation.

**ControlTower:** thread function that is responsible for all traffic. We use this function to allocate jobs in a suitable pad. At the beginning we only consider one pad so, it was the one that does and logs the job. However, when we introduce pad B, we needed to differentiate between pads and jobs; therefore, we created two other threads as PadA and PadB. After this separation control tower got the responsibility of allocating job into one of them and doing and logging the job is now Pads’ responsibility.

**PadA, PadB:** two threads for jobs in the pads. It performs the job (sleeps for needed time to job), clear its queue and waits for others to come. It also creates log information for a job that is done.

**SnapShotPrint:** this is a thread that prints queues in each specified second

**Changes in queue.c:** request time which is the creation time of a job is added to Job structure. At first, this information is used for logging; however, later it is also used for starvation related problem. printQueue function is also added. This function is used for queue snapshots and prints the message and queue for a given type and second of a queue.

Part-II Starvation Cause & Our Solution:

Starvation the pieces occurs because if we give a static priority for each job then feeding the system with the one has higher priority will make others to never executed. For example, in the first setup since system can always produce land jobs then we will not have time to execute launching or assembly. Given suggestion by Space Y also does not solve starvation issue. Because when we start to do the launch or assembly, we may never switch back to land. Consider the following case. There are 3 launch jobs that are waiting so we started to do launch job. While doing that we keep having more launch and assembly job than we are finishing. So, we will favor them, and land jobs will continue to wait. Then we need a solution that does not end up with a specific job type always favored or disfavored (like if there are +3 launch or assembly land will always have to wait).

Our solution to this is the following. We implemented a priority in round robin style. Each pad has a current priority, and if any tasks of the priority type are available, they will be done. But if no tasks of the priority type are waiting, then we can do a task of the other type. Moreover, this idea can also use to implement emergency queue given each pad have a priority on emergency so when it is available, we can have that.

Our Implementation in Summary:

Our implementation creates a starvation and deadlock free space controller. We accomplished this using appropriate threads, mutex locks and scheduling algorithm. Each thread is running in parallel in real time. We used mutexes very carefully to make sure no two action that may cause inconsistencies, deadlocks or breakdowns can occur at the same time. And in the second step we proposed a scheduling algorithm, for which we provide detail above, to prevent starvation.

References:

[1] <https://hpc-tutorials.llnl.gov/posix/joining_and_detaching/>