Lab 3
Group 5

Data structure

- A lock to prevent multiple threads running getSlot/leaveSlot
- Condition variable matrix for different directions and priorities
- Task count matrix for different directions and priorities
- Current direction variable
- Running tasks count variable

Algorithms

In order for the requirements to meet its purpose, the algorithm needs to prioritize the task in each direction. When a task wants a slot it checks if there are no more than 3 tasks running as well as the current direction, if the statement is not met then the task will be placed in a queue that is specified by the task's direction and priority.

When exiting the "running tasks" we decrement the amount of running tasks and hand-over/signaling the spot to the tasks with the same direction and highest priority. If there are no more tasks in the same direction we broadcast/signaling the tasks in the opposite direction with the highest priority first.

Synchronization

We chose to use a lock to only allow one process at a time to get or leave a slot. Additionally, using conditional variables we choose which tasks should be run next based on priority.

Rationale

We choose to use lock in order to achieve mutual exclusion and conditional variables keep track of the queue for each direction and priority. We could have used flags and a pointer to which thread is running its task, but we did not choose this method as it is more complex than it reasonably needs to be.

Extra note

Timer from lab 2 had to be modified as it was not working properly.